ELSEVIER

Decision Support Systems 15 (1995) 251–266

Decision Support Systems

Invited Paper

Design and natural science research on information technology ☆

Salvatore T. March *, Gerald F. Smith

Information and Decision Sciences Department, Carlson School of Management, University of Minnesota, 271 19th Avenue South, Minneapolis, MN 55455, USA

Abstract

Research in IT must address the design tasks faced by practitioners. Real problems must be properly conceptualized and represented, appropriate techniques for their solution must be constructed, and solutions must be implemented and evaluated using appropriate criteria. If significant progress is to be made, IT research must also develop an understanding of how and why IT systems work or do not work. Such an understanding must tie together natural laws governing IT systems with natural laws governing the environments in which they operate. This paper presents a two dimensional framework for research in information technology. The first dimension is based on broad types of design and natural science research activities: build, evaluate, theorize, and justify. The second dimension is based on broad types of outputs produced by design research: representational constructs, models, methods, and instantiations. We argue that both design science and natural science activities are needed to insure that IT research is both relevant and effective.

Keywords: Information system research; Design science; Natural science; Information technology

1. Introduction

Researchers in Information Technology (IT) have defined *information* as "data that has been processed into a form that is meaningful to the recipient and is of real or perceived value in current or prospective actions or decisions" [[14], p. 200]. This definition can be grounded in cognitivist theories of mental representation [67]. Human thinking involves mental representations that intendedly correspond to reality. These representations are commonly called beliefs or, when highly validated, knowledge. They are produced when people pick up sensory inputs or stimuli from their environment. As new information is acquired, one's beliefs are adjusted to better match the perceived reality.

Human knowledge and beliefs inform actions taken in pursuit of goals. Well-informed actions (i.e., those based on true beliefs) are more likely to achieve desired ends. Information is valuable insofar as it helps individuals form true beliefs which, in turn, promote effective, goal-achieving action.

Technology has been defined as "practical implementations of intelligence" [[20], p. 26]. Technology is practical or useful, rather than being an end in itself. It is embodied, as in implements or artifacts, rather than being solely conceptual. It is an expression of intelligence, not a product of

☆ This paper is an extension of ideas originally presented in [46]

* Corresponding author

SSDI 0167-9236(94)00041-7

blind accident. Technology includes the many tools, techniques, materials, and sources of power that humans have developed to achieve their goals. Technologies are often developed in response to specific task requirements using practical reasoning and experiential knowledge.

Information technology is technology used to acquire and process information in support of human purposes. It is typically instantiated as IT systems – complex organizations of hardware, software, procedures, data, and people, developed to address tasks faced by individuals and groups, typically within some organizational setting.

IT is pervasive throughout the industrialized world. Business and government organizations annually spend billions of dollars to develop and maintain such systems. IT affects the work we do and how that work is done [62]. Innovative uses of information technologies have led to significant improvements for some companies (such as American Hospital Supply, Federal Express, Mrs Fields, Frito Lay, etc.) and have defined the competitive marketplace for others (e.g., the airline industry).

IT practice is concerned with the development, implementation, operation, and maintenance of IT systems. Development and maintenance are largely design activities. Systems analysts, programmers, and other professionals construct artifacts that apply information technology to organizational tasks. Applications can be as mundane as keeping track of customer and vendor accounts and as sophisticated as decision making systems exhibiting human-like intelligence. Implementation and operation are processes that utilize designed methods, techniques, and procedures. Not all attempts to exploit information technologies have had such positive results [44]. Even mundane applications of information technology can be overly expensive or have adverse affects on the organization [26].

IT has attracted scientific attention, in part because of its potential for dramatically impacting organizational effectiveness, both positively and negatively. Scientists have also been drawn by the pervasiveness of IT phenomena in our information-based society [18,19]. Scientific interest in IT reflects assumptions that these phenomena can be explained by scientific theories and that scientific research can improve IT practice. Note, however, that there are two kinds of scientific interest in IT, descriptive and prescriptive. Descriptive research aims at understanding the nature of IT. It is a knowledge-producing activity corresponding to natural science [27]. Prescriptive research aims at improving IT performance. It is a knowledge-using activity corresponding to design science [65].

Though not intrinsically harmful, this division of interests has created a dichotomy among IT researchers and disagreement over what constitutes legitimate scientific research in the field. Such disagreements are common in fields that encompass both knowledge-producing and knowledge-using activities. They are fostered in part by the prestige attached to science in modern societies and the belief that the term "science" should be reserved for research that produces theoretical knowledge. The debate in IT research is similar to that between engineering and the physical sciences. Knowledge-producing, "pure" science normally has the upper hand in such debates. In IT, however, the situation is different. It could be argued that research aimed at developing IT systems, at improving IT practice, has been more successful and important than traditional scientific attempts to understand it. With the issue undecided, the field is left in an uneasy standoff.

This article proposes a framework for IT research that reconciles these conflicting points of view. This framework also suggests a research agenda for the scientific study of IT. The next section contains theoretical background material needed to understand the dual nature of IT research. The framework itself is presented in the following section and is applied to IT literature, with examples drawn primarily from the domain of data management. The final section offers prescriptions for future IT research.

2. Theoretical background

IT research studies artificial as opposed to natural phenomena. It deals with human cre-

ations such as organizations and information systems. This has significant implications for IT research which will be discussed later. Of immediate interest is that fact that artificial phenomena can be both created and studied, and that scientists can contribute to each of these activities. This underlies the dual nature of IT research. Rather than being in conflict, however, both activities can be encompassed under a broad notion of science that includes two distinct species, termed natural and design science [65]. Natural science is concerned with explaining how and why things are. Design science is concerned with "devising artifacts to attain goals" [65, p. 133].

Natural science includes traditional research in physical, biological, social, and behavioral domains. Such research is aimed at understanding reality. Natural scientists develop sets of concepts, or specialized language, with which to characterize phenomena. These are used in higher order constructions – laws, models, and theories – that make claims about the nature of reality. Theories – deep, principled explanations of phenomena [1] – are the crowning achievements of natural science research. Products of natural science research are evaluated against norms of truth, or explanatory power. Claims must be consistent with observed facts, the ability to predict future observations being a mark of explanatory success. Progress is achieved as new theories provide deeper, more encompassing, and more accurate explanations.

Natural science is often viewed as consisting of two activities, discovery and justification [35]. Discovery is the process of generating or proposing scientific claims (e.g., theories, laws). Justification includes activities by which such claims are tested for validity. The discovery process is not well understood. Though some have argued that there is a "logic" of scientific discovery [7], mainstream philosophy of science has historically regarded discovery as a creative process that psychologists may or may not be able to understand [4]. Promising insights into this phenomena have recently been offered by AI research [41].

Justification, on the other hand, has been heavily prescribed for by philosophers of science. An initial commitment to inductive logic, justifying claims by accumulating confirming instances, was overthrown by Popper's falsificationism [59]. Popper argued that scientists should try to disprove claims since a single negative instance could do so, while innumerable confirming instances could not prove a theory true.

Scientific study makes extensive use of the hypothetico-deductive method. That is, theories can be tested insofar as observational hypotheses can be deduced from them and compared to relevant empirical data [4]. Most scientific methodologies used by IT researchers are prescriptions for collecting and assessing data in this way [34].

Whereas natural science tries to understand reality, *design science* attempts to create things that serve human purposes. It is technology-oriented. Its products are assessed against criteria of value or utility – does it work? is it an improvement? Design is a key activity in fields like architecture, engineering, and urban planning [61] that may not be thought of as "sciences" per se. At the same time, design activities are an important part of traditional scientific fields, some scientists conducting both natural and design science investigations. Then too, fields like operations research and management science (OR/MS) are heavily prescriptive in intent, while claiming to be sciences. Rather than producing general theoretical knowledge, design scientists produce and apply knowledge of tasks or situations in order to create effective artifacts. If science is activity that produces "credentialed knowledge" [[51], p. 311], then, following Simon [65], design science is an important part of it.

Design science products are of four types, constructs, models, methods, and implementations. As in natural science, there is a need for a basic language of concepts (i.e., constructs) with which to characterize phenomena. These can be combined in higher order constructions, often termed models, used to describe tasks, situations, or artifacts. Design scientists also develop methods, ways of performing goal-directed activities. Finally, the foregoing can be instantiated in specific products, physical implementations intended to perform certain tasks. Notably absent from this list are theories, the ultimate products of natural science

research. Rather than posing theories, design scientists strive to create models, methods, and implementations that are innovative and valuable.

Design science consists of two basic activities, build and evaluate. These parallel the discovery-justification pair from natural science. Building is the process of constructing an artifact for a specific purpose; evaluation is the process of determining how well the artifact performs. Like the discovery process in natural science, the design science build process is not well understood.

Significant difficulties in design science result from the fact that artifact performance is related to the environment in which it operates. Incomplete understanding of that environment can result in inappropriately designed artifacts or artifacts that result in undesirable side-effects. A critical challenge in building an artifact is anticipating the potential side-effects of its use, and insuring that unwanted side-effects are avoided.

Evaluation is complicated by the fact that performance is related to intended use, and the intended use of an artifact can cover a range of tasks. General problem solving methods, for example, are applicable to many different problems with performance varying considerably over the domain of application. Not only must an artifact be evaluated, but the evaluation criteria themselves must be determined for the artifact in a particular environment. Progress is achieved in design science when existing technologies are replaced by more effective ones.

To further clarify the natural science-design science distinction, it can be compared to others commonly made. Simon's [65] distinction between natural and artificial phenomena, discussed at the beginning of this section, should not be confused with the natural-design science pair. Design science produces artifacts and artificial phenomena. However, natural science can address both natural and artificial phenomena. Natural scientists, for example, try to understand the functioning of organizations, which are artificial phenomena. Likewise, chemists attempt to determine the properties of synthetic compounds, and biologists investigate the behaviours of man-made organisms. These are natural science activities even though they take artificial phenomena as their objects. Rather than being driven by research topic, the natural-design science distinction is based on different research objectives. Natural science aims at understanding and explaining phenomena; design sciences aims at developing ways to achieve human goals.

The distinction between basic and applied science is also relevant. This usually reflects how closely scientific research impinges on practice. While natural science tends to be basic research and design science tends to be applied, the two pairs of concepts are not strictly parallel. A natural science account of information systems failure could be more relevant to practice than the development of a new data modelling formalism. Yet the former is natural science research, whereas the latter is design science research. Again, research intent is critical.

More relevant is the description-prescription distinction frequently employed by decision scientists [5]. Natural science is descriptive and explanatory in intent. Design science offers prescriptions and creates artifacts that embody those prescriptions.

Having differentiated two species of scientific activity, it is important to appreciate their interactions. First, design science creates artifacts, giving rise to phenomena that can be the targets of natural science research. Group decision support systems, for example, foster user behaviours that are the subject of natural science investigations (see for example, [21]).

Second, because artifacts "have no dispensation to ignore or violate natural laws" [65, p. 6], their design can be aided by explicit understanding of natural phenomena. Thus natural scientists create knowledge which design scientists can exploit in their attempts to develop technology. However, frequently the natural laws governing an artifact and its environment are not well understood. Hence, the constructed artifact itself presents a challenge to explain how and why it works. Natural science explanations of how or why an artifact works may lag years behind the application of the artifact. In medicine, for example, the explanation of why a drug is effective in combating a disease may not be known until long after the drug is in common use.

A final interaction concerns the justification of natural science claims. Theories are intended to correspond with or present a true account of reality. However, reality cannot be directly apprehended; we only have perceptions and other representations of such. How then are we to determine if theoretical claims are true? This validation problem is overcome in part by the effectiveness of theories in practical applications [43]. Natural science theories receive confirmatory support from the facts that bridges do not collapse, medical treatments cure diseases, people journey to the moon, and nuclear bombs explode. Indeed, philosophical pragmatists deny the correspondence notion of truth, proposing that truth essentially is what works in practice [60]. Thus, design science provides substantive tests of the claims of natural science research.

3. A research framework in information technology

Prior research frameworks in IT have characterized specific research subjects, identifying sets of variables to be studied (see, e.g., [32,24,49]). Such frameworks facilitate the generation of specific research hypotheses by positing interactions among identified variables. While this provides innumerable research questions it has several weaknesses. First, it fails to provide direction for choosing important interactions to study [74]; any and all interactions among identified variables are treated equally. Second, it fails to account for the large body of design science research being done in the field. Third, it fails to recognize that IT research is concerned with artificial phenomena operating for a purpose within an environment; the nature of the task to which the IT is applied is critical. Fourth, it fails to recognize the adaptive nature of artificial phenomena; the phenomena itself is subject to change, even over the duration of the research study.

Weber recognized that IT research is the study of artifacts as they are adapted to their changing environments and to changes in their underlying components [74]. We further argue that an appropriate framework for IT research lies in the interaction of design and natural sciences. IT research should be concerned both with utility, as a design science, and with theory, as a natural science. The theories must explain how and why IT systems work within their operating environments.

Our proposed framework is driven by the distinction between research outputs and research activities (Fig. 1). The first dimension of the

Research Activities

Research Outputs	Build	Evaluate	Theorize	Justify
Constructs				
Model				
Method				
Instantiation				

Fig. 1. A research framework.

framework is based on design science research outputs or artifacts: constructs, models, methods, and instantiations. The second dimension is based on broad types of design science and natural science research activities: build, evaluate, theorize, and justify. IT research builds and evaluates constructs, models, methods, and instantiations. It also theorizes about these artifacts and attempts to justify these theories. Building and evaluating IT artifacts have design science intent. Theorizing and justifying have natural science intent.

3.1. Research outputs

Constructs or concepts form the vocabulary of a domain. They constitute a conceptualization used to describe problems within the domain and to specify their solutions. They form the specialized language and shared knowledge of a discipline or sub-discipline. Such constructs may be highly formalized as in semantic data modelling formalisms (having constructs such as entities, attributes, relationships, identifiers, constraints [31]), or informal as in cooperative work (consensus, participation, satisfaction [39]). Kuhn's notion of paradigm is based on the existence of an agreed upon set of constructs for a domain [40].

Conceptualizations are extremely important in both natural and design science. They define the terms used when describing and thinking about tasks. They can be extremely valuable to designers and researchers. Brooks [6], for example, discusses the transformation in thinking about the software development process when presented with the conceptualization of *growing* rather then *building*. When software is built, one expects to specify the entire plan in advance and then construct the software according to plan. When software is grown, it is developed *incrementally*. Functionality is added as it is needed. The software developer conceives of the product as being dynamic, constantly evolving rather than as a static entity that is "completed" at a point in time.

On the other hand, conceptualizations can blind researchers and practitioners to critical issues. In the database area, the relational data model [11] provided an extremely influential conceptualization with which to describe data. It conceives of data as flat tables and defines concepts such as functional dependency and normal forms with which to evaluate database structures. It removes the consideration of physical file structures and permits an analyst to be concerned with "well-formed" logical structures. Unfortunately, this formalism became so ingrained that researchers lost sight of its shortcomings for physical database design. Research activities focused on how to efficiently process flat tables (see, e.g., [52]), when it is clear that nested record structures and system pointers (disallowed in the relational model but available in prior database representations [45]) are necessary to achieve efficient operations in many applications.

A *model* is a set of propositions or statements expressing relationships among constructs. In design activities, models represent situations as problem and solution statements. In semantic data modelling, the term model has been inappropriately used to mean data modelling formalism. The Entity-Relationship Model [9], for example, is a set of constructs, a data modelling formalism. The representation of an information system's data requirements using the Entity-Relationship constructs is more appropriately termed a model. Such a model is a solution component to an information requirements determination task and a problem definition component to an information system design task.

A model can be viewed simply as a description, that is, as a representation of how things are. Natural scientists often use the term model as a synonym for theory, or propose models as weak or incipient theories, in that they propose that phenomena be understood in terms of certain concepts and relationships among them. In our framework, however, the concern of models is utility, not truth (the concern of theories is truth, as discussed below). A semantic data model, for example, is valuable insofar as it is *useful* for designing an information system. Certain inaccuracies and abstractions are inconsequential for those purposes. In data models, for example, the notion of entity is pragmatically defined as an

"arbitrary" grouping of instances [37,38]. Although theoretical criteria have been proposed for defining entities [57], the key concern is that the entities chosen be useful in representing and communicating information system requirements.

Although silent or inaccurate on the details, a model may need to capture the structure of reality in order to be a useful representation. Simulation and mathematical models, for example, can be of immense practical value, although lacking in many details [66]. On the other hand, unless the inaccuracies and abstractions inherent in models are understood, their use can lead to inappropriate actions.

Modelling database operations as "logical block accesses" [70], for example, calculates the expected number of data blocks required to satisfy a database retrieval request, ignoring the details of physical storage devices and the complexities of the computer operating environment. It is extremely useful for feasibility assessment where the purpose is to obtain "order of magnitude" approximations of system performance. However, it is inappropriate for physical database design where the purpose is to tune the design for efficient overall operations. In this case physical storage devices and the computer operating environment must be more accurately represented or the model will not be able to distinguish the efficiency effects of various design decisions [10].

To further illustrate the utilitarian concern of constructs and models, consider the research in expert systems where knowledge is modeled as a set of production rules or frames [13]. Although proposed as a model of human expertise (i.e., a cognitive science theory), from a design science point of view, it is irrelevant if humans actually represent knowledge in this way. The concern is if this type of representation is useful for the development of artifacts to serve human purposes. While expert systems research has been criticized for abandoning "basic research on intelligence" (Hofstadter, quoted by Weber [74]), its design science impacts are irrefutable.

A *method* is a set of steps (an algorithm or guideline) used to perform a task. Methods are based on a set of underlying constructs (language) and a representation (model) of the solution space [54]. Although they may not be explicitly articulated, representations of tasks and results are intrinsic to methods. Methods can be tied to particular models in that the steps take parts of the model as input. Further, methods are often used to translate from one model or representation to another in the course of solving a problem.

Data structures, for example, combine a representation of computer memory with algorithms to store and retrieve data. The problem statement specifies the existing stored data and the data to be stored or retrieved. The method (algorithm) transforms this into a new specification of stored data (storage) or returns the requested data (retrieval). Many algorithms use tree-structured constructs to model the problem and its solution.

System development methods facilitate the construction of a representation of user needs (expressed, for example, as problems, decisions, critical success factors, socio-technical and implementation factors, etc.). They further facilitate the transformation of user needs into system requirements (expressed in semantic data models, behaviour models, process flow models, etc.) and then into system specifications (expressed in database schemas, software modules, etc.), and finally into an implementation (expressed in physical data structures, programming language statements, etc.). These are further transformed into machine language instructions and bits stored on disks.

The desire to utilize a certain type of method can influence the constructs and models developed for a task. For example, desiring to use a mathematical programming method, a researcher may conceptualize a database design problem using constructs such as decision variable, objective function, and constraint. Developing a model of the problem using these constructs is itself a design task requiring powerful methods and techniques. Depending on the particulars of the model, new mathematical programming solution methods may need to be designed. The solution to the model itself represents a design task for implementation.

Natural science uses but does not produce

methods. Design science creates the methodological tools that natural scientists use. Research methodologies prescribe appropriate ways to gather and analyze evidence to support (or refute) a posited theory [34,42]. They are human-created artifacts that have value insofar as they address this task.

An *instantiation* is the realization of an artifact in its environment. IT research instantiates both specific information systems and tools that address various aspect of designing information systems. Instantiations operationalize constructs, models, and methods. However, an instantiation may actually precede the complete articulation of its underlying constructs, models, and methods. That is, an IT system may be instantiated out of necessity, using intuition and experience. Only as it is studied and used are we able to formalize the constructs, models, and methods on which it is based.

Instantiations demonstrate the feasibility and effectiveness of the models and methods they contain. Newell and Simon [53] emphasize the importance of instantiations in computer science, describing it as "an empirical discipline." They further state, "Each new program that is built is an experiment. It poses a question to nature, and its behaviour offers clues to the answer." Instantiations provide working artifacts, the study of which can lead to significant advancements in both design and natural science.

Group Decision Support System (GDSS) instantiations, for example, were developed in order to study the impacts of automated interventions on group processes. As early GDSSs were studied, constructs (e.g., roles, anonymity, Type I, II, and III GDSSs), models (e.g., situation models representing different problem types), and methods (e.g., idea generation, idea evaluation, automated facilitation) [16] leading to improved instantiations, were developed.

As a further example of the importance of instantiations, consider early research in operating systems. An operating system manages the resources of a computer system. It instantiates constructs such as processes, states, interrupts, and abstract machines and uses methods such as scheduling algorithms and memory management algorithms. The operationalization of such concepts within the UNIX operating system, "led a generation of software designers to new ways of thinking about programming" [[2], p. 757].

3.2. Research activities

Research activities in design science are twofold: build and evaluate. Build refers to the construction of the artifact, demonstrating that such an artifact *can* be constructed. Evaluate refers to the development of criteria and the assessment of artifact performance against those criteria.

Research activities in natural science are parallel: discover and justify. Discover, or more appropriately for IT research, theorize, refers to the construction of theories that explain how or why something happens. In the case of IT research this is primarily an explanation of how or why an artifact works within its environment. Justify refers to theory proving. It requires the gathering of scientific evidence that supports or refutes the theory.

We *build* an artifact to perform a specific task. The basic question is, does it work? Building an artifact demonstrates feasibility. These artifacts then become the object of study. We build constructs, models, methods, and instantiations. Each is a technology that, once built, must be evaluated scientifically.

We *evaluate* artifacts to determine if we have made any progress. The basic question is, how well does it work? Recall that progress is achieved when a technology is replaced by a more effective one. Evaluation requires the development of metrics and the measurement of artifacts according to those metrics. Metrics define what we are trying to accomplish. They are used to assess the performance of an artifact. Lack of metrics and failure to measure artifact performance according to established criteria result in an inability to effectively judge research efforts.

Hopcroft [30], for example, describes early research in algorithm development as being "very unsatisfying" due to a lack of coherent metrics by which to compare performance. A common approach to evaluating an algorithm was to com-

pare the execution time of the algorithm with that of a competing algorithm for a specific task. The difficulty lay in the fact that the two algorithms were likely written in different languages and run on different computers, yielded little information from which to judge their relative performance. This motivated the development of "worst-case asymptotic performance" metrics. These provided objective measures of the relative performance of algorithms independent of their implementations.

However, as noted by Tarjan [68], metrics themselves must be scrutinized by experimental analysis. The algorithm with best "worst-case" performance may not be the best algorithm for a task. The simplex method, an algorithm commonly used to solve linear programming problems, for example, has relatively poor worst-case performance, growing "exponentially with the number of linear inequalities" [[36], p. 108]. The ellipsoid method, on the other hand, is polynomialy bounded, a considerable improvement. In practice, however, "the number of iterations is seldom greater than three or four times the number of linear inequalities" for the simplex algorithm, whereas the ellipsoid method does not fare nearly so well.

Given an artifact whose performance has been evaluated, it is important to determine why and how the artifact worked or did not work within its environment. Such research applies natural science methods to IT artifacts. We *theorize* and then *justify* theories about those artifacts.

Theories explicate the characteristics of the artifact and its interaction with the environment that result in the observed performance. This requires an understanding of the natural laws governing the artifact and those governing the environment in which it operates. Furthermore, the interaction of the artifact with its environment may lead to theorizing about the internal workings of the artifact itself or about the environment.

Among others, Chan, et. al. [8] conclude that, for inexperienced end-users, semantic data models and query languages are more effective for database access than the relational data model and SQL. While this is an important evaluative result, the critical question for designers of database interfaces is, *why* are the semantic data models more effective? Norman [56] theorizes that human performance in man-machine interactions depends on the "gulf" between the human's conceptualization of the task and the presented interface. When the gulf is large, performance is poor. When the gulf is small, performance is good. One could theorize, then, that semantic data models better correspond to an end-user's conceptualization of a database than does the relational model.

Mathematically-based artifacts may lead a researcher to posit mathematical theorems in order to explain behaviour and improve performance. Genetic algorithms, for example, were developed nearly twenty years ago and have been effectively applied to numerous discrete optimization problems [15] including distributed database design [47]. They were initially developed based on the analogy of natural selection resulting in improved living populations [28,29]. Yet only recently has theory been developed to explain how various control parameters affect performance [25].

Theorizing in IT research must explicate those characteristics of the IT artifact operating in its environment that make it unique to IT and require unique explanations. Theorizing that Newton's theory of gravity holds for IT, and testing it by dropping a PC from an office window in the MIS department is obviously not valuable. While this example is extreme, the issue is that researchers should not simply test theories from reference disciplines in an IT context unless there is good reason to believe that IT phenomena are unique in some way that would affect the applicability of that theory. On the other hand, where IT artifacts significantly affect the task or the environment, adapting theories from referent disciplines can be extremely fruitful (e.g., theories of group dynamics in a GDSS environment [58]).

Given a generalization or theory we must *justify* that explanation. That is, we must gather evidence to test the theory. For artifacts based on mathematical formalisms or whose interactions with the environment are represented mathematically (such as in information economics research or automated database design), this can be done

mathematically, i.e., by using mathematics and logic to prove posited theorems. In such cases, we normally prove things about the performance of the artifact based on some metric. Database design algorithms, for example, have been proven to produce "optimal" designs in the sense that they optimize some performance measure (e.g., minimize the operating cost) for the given set of database activities. Of course, these results are only valid within the underlying formal problem representation (model). Furthermore, the performance metrics themselves must be justified (e.g., is response time more significant than operating cost?).

Justification for non-mathematically represented IT artifacts follows the natural science methodologies governing data collection and analysis [34].

3.3. Application to IT research

Constructs, models, methods, and instantiations are each artifacts that address some task. Research activities related to these artifacts are: build, evaluate, theorize, and justify. Build and evaluate are design science research activities aimed at improving performance. Theorize and justify are natural science research activities aimed at extracting general knowledge by proposing and testing theories.

This four by four framework produces sixteen cells describing viable research efforts. Research can build, evaluate, theorize about, or justify theories about constructs, models, methods, or instantiations. Different cells have different objectives and different methods are appropriate in different cells. Research efforts often cover multiple cells. Evaluation of research should be based on the cell or cells in which the research lies.

Research in the build activity should be judged based on value or utility to a community of users. Building the *first* of virtually any set of constructs, model, method, or instantiation is deemed to be research, provided the artifact has utility for an important task. The research contribution lies in the novelty of the artifact and in the persuasiveness of the claims that it is effective. Actual performance evaluation is not required at this stage. The significance of research that builds subsequent constructs, models, methods, and instantiations addressing the same task is judged based on "significant improvement," e.g., more comprehensive, better performance. Research in database design has numerous examples of researchers extending or combining constructs, models, methods, and instantiations developed by other researchers (see, e.g., [47,48]).

Recognizing that there is nothing new under the sun, "first" is usually interpreted to mean, "never done within the discipline." The relational data model [11], for example, was the first attempt to define a formalism in which to describe data and retrieval operations. Relations and relational operators had never before been used to describe data, although they had been used extensively in mathematics to describe sets and set behaviour.

While there is little argument that novel constructs, models, and methods are viable research results, there is less enthusiasm in the information technology literature for novel instantiations. Novel instantiations, it is argued, are simply extensions of novel constructs, models, or methods. Hence, we should value the constructs, models, and methods, but not the instantiations. Reaction is quite different in the computer science literature where a key determinant of the value of constructs, models, and methods is the existence of an implementation (instantiation).

In much of the computer science literature it is realized that constructs, models, and methods that work "on paper" will not necessarily work in real world contexts. Consequently, instantiations provide the real proof. This is evident, for example, in AI where achieving "intelligent behaviour" is a research objective. Exercising instantiations that purport to behave intelligently is the primary means of identifying deficiencies in the constructs, models, and methods underlying the instantiation.

On the other hand, instantiations that apply known constructs, models, and methods to novel tasks may be of little significance. Of primary concern is the level of uncertainty over the viability of the constructs, models, and methods for the task. For example, there is no reason to believe

that expert system technology would not be applicable to the task of choosing stock investments. The task is characterized as having a limited number of choices, reasonably well defined selection criteria, and there are a number of experts currently performing the task. Instantiating an expert system for this task, even the first, would be of marginal scientific significance.

Research in the evaluate activity develops metrics and compares the performance of constructs, models, methods, and instantiations for specific tasks. Metrics define what a research area is trying to accomplish. Since "the second" or subsequent constructs, models, methods, or instantiations for a given task must provide significant performance improvements, evaluation is the key activity for assessing such research.

Evaluation of constructs tends to involve completeness, simplicity, elegance, understandability, and ease of use. Data modelling formalisms, for example, are constructs with which to represent the logical structure of data. Proposing a data modelling formalism falls within the build-constructs cell of the framework. The database literature was subjected to a plethora of data modelling formalisms [31]. As researchers found deficiencies in existing data modelling formalisms, they proposed additional or slightly different constructs to meet their specific tasks (e.g., modelling statistical and scientific data). Finally, reviewers screamed, "not yet another..."

Models are evaluated in terms of their fidelity with real world phenomena, completeness, level of detail, robustness, and internal consistency. For example, numerous mathematical models have been developed for database design problems. Posing a new database design model may be a significant contribution; however, to inform researchers in the field, the new model must be positioned with respect to existing models. Often existing models are extended to capture more of the relevant aspects of the task. March and Rho [47], for example, extended the distributed database model posed by Cornell and Yu [12] to include not only the allocation of data and operations, but data replication and concurrency control as well, significant issues in this area.

Evaluation of methods considers operationality (the ability to perform the intended task or the ability of humans to effectively use the method if it is not algorithmic), efficiency, generality, and ease of use. Associated with the numerous distributed database design models are numerous methods to solve problems represented in those models. While some use mathematical algorithms (e.g., linear or non-linear programming), others develop novel methods (e.g., iterative algorithms). In the former case, the method is not a significant research contribution, although it may be important to evaluate the effectiveness or efficiency of the method for this particular type of problem (e.g., are there characteristics of the model that make one method more or less applicable?). In the latter case, the method itself is a significant research contribution, to be studied and evaluated even apart from the application.

As another example, consider the numerous information system development methods. These can be evaluated for completeness, consistency, ease of use, and the quality of results obtained by analysts applying the method [33,56,64].

Evaluation of instantiations considers the efficiency and effectiveness of the artifact and its impacts on the environment and its users. A difficulty with evaluating instantiations is separating the instantiation from the constructs, models, and methods embodied in it. CASE tools are an example of instantiations. While these embody certain constructs, models, and methods, the developer of the CASE tool must select from among a wide array of available constructs, models, and methods, and must decide how much latitude to afford users [71]. It is these design choices that differentiate CASE tools. Evaluations focus on these differences and how they change the task of system development.

Once metrics are developed, empirical work may be necessary to perform the evaluation. Constructs, models, methods, and instantiations must be exercised within their environments. Often this means obtaining a subject group to do the exercising. Often multiple constructs, models, methods, or instantiations are studied and compared. Issues that must be addressed include comparability, subject selection, training, time, and tasks. Methods for this type of evaluation are

not unlike those for justifying or testing theories. However, the aim is to determine "how well" an artifact works, not to prove anything about how or why the artifact works.

The third and fourth columns of the framework correspond to natural science activities, theorize and justify. The *theorize* activity involves explaining why and how the effects came about, i.e., why and how the constructs, models, methods, and instantiations work. This activity attempts to unify the known data (observations of effects) into viable theory – explanations of how and why things happen. It may involve developing constructs with which to theorize about constructs, models, methods, and instantiations. The *justify* activity performs empirical and/or theoretical research to test the theories posed. Such theories, once justified, can provide direction for the development of additional and better technologies.

As discussed earlier, Norman [55] developed the construct of gulfs when theorizing about how the constructs used in human-computer interaction affect performance. This construct underlies theory posed by Weber and Zhang [75] explaining why analysts had difficulty with a particular data modelling formalism (activity in the theorize-constructs cell). Its value has been substantiated by several studies [3] (activity in the justify-constructs cell).

Theorizing about models can be as simple as positing that a model used for design purposes is true (e.g., that an expert's knowledge of a task is accurately represented by a set of production rules), or as complicated as developing an explanatory model of an IT phenomena (e.g., representing system-task fit as a basis for user perceptions of satisfaction with information technology [22,23]).

The theorize-model cell also includes adapting theories from base disciplines or developing new, general theories to explain how or why IT phenomena work. Nolan's [54] stage theory, for example, posits that the growth pattern of an EDP organization follows a sigmoid shape. Although criticized as a simple application of general systems theory [74]), it does explain important phenomena in the organizational appropriation of information technologies. The theory has been tested by various researchers, albeit with mixed results.

As another example, DeSanctis and Poole [17] proposed Adaptive Structuration Theory (AST) to explain differences in how GDSS technologies were used and in the effectiveness of that use for different group and task characteristics. Although tested by those researchers in a laboratory study, they note that evidence from real groups engaged in a diversity of real tasks over a substantial period of time must be gathered.

For algorithmic methods, theorizing can be formal and mathematical with logical proofs being used for justification (e.g., database design algorithms) or it can be behavioral, explaining why or how a method works in practice (e.g., why and how particular system development methods work). As discussed earlier, formal, mathematical theories are proven only within their defined formalism – they must be tested in practice to validate the formalism.

Theorizing about instantiations may be viewed as a first step toward developing more general theories (e.g., explaining how electronic mail affects communication may lead to more general theories of the effects of technology or more general theories about human communication) or as the specialization of an existing general theory (e.g., applying innovation-diffusion theory to specific information technologies such as spreadsheets and electronic mail can facilitate understanding of the IT phenomena).

4. Discussion and prescriptions for IT research

Key to understanding IT research is the fact that IT deals with artificial phenomena rather than natural phenomena. Brooks [6, p. 12] observes, "Einstein argued that there must be simplified explanations of nature because God is not capricious or arbitrary," but concludes that, "No such faith comforts the software engineer" because the objects of study "were designed by different people, rather than by God." The phenomena studied in IT research are artifacts that

are designed and built by man to accomplish the purposes of man.

Implications for IT research are threefold. First, there may not, in fact, be an underlying deep structure to support a theory of IT. Our theories may need, instead, to be based on theories of the natural phenomena (i.e., people) that are impacted by the technology. Second, our artifacts are perishable, hence our research results are perishable. As needs change, the artifacts produced to meet those needs also change. A theory of how programmers use a now-defunct language, for example, would be of little interest. Third, we are producing IT artifacts at an ever increasing rate, resulting in innumerable phenomena to study. Explicating and evaluating IT artifacts (constructs, models, methods, and instantiations) will facilitate their categorization so that research efforts will not be wasted building and studying artifacts that have already been built and studied "in kind."

Data modelling is an area in which artifacts were not adequately explicated or evaluated. As a result, research efforts proposed data modelling formalisms that, in essence, had already been built – they varied primarily in the names chosen for constructs. Metrics were not developed and substantive comparative evaluations were not done. We still lack an accepted set of metrics and comparative analysis of data modelling formalisms. We further lack a theory of constructs for data modelling formalisms, although work is beginning in each of these areas [3,73].

Similarly in database management, significant amounts of build research have been done without adequate theories to explicate and studies to evaluate its underlying constructs, models, methods, and instantiations. As discussed earlier, the relational data model [11] has been the dominant formalism in both research and practice over the past decade. However, it was not compared with competing formalisms until recently [8]. Had the relational model been compared to formalisms such as DIAM [63], for end-user database interaction, it would have been clear that relational languages were not as effective as the graphical constructs in DIAM (similar to those in current semantic data models), and research in end-user languages could have shifted from relational constructs to semantic constructs.

With the dominance of relational DBMSs in practice, the performance shortcomings for certain types of applications (e.g., CAD and engineering databases) have become apparent. Among other things, the "next generation" of DBMSs, the so-called Object-Oriented DBMSs, have reimplemented complex record structures (at the physical level) and provide efficient system pointers for interfile connections, representations not unlike those described in DIAM and implemented in pre-relational DBMSs!

Current research in system development can be similarly characterized. Significant attention has been given to the build activity. Process, data, and behaviour representations (formalisms) have been build and are in use (see, e.g., [31,56]). Methods to build models using these representations have been proposed, although these are mostly ad hoc (see, e.g., [69]). CASE tools instantiate representations and methods. Thus far there has been only moderate evaluation activity for any of these artifacts. Several researchers compare and evaluate data modelling formalisms (e.g., [3,75]), methods (e.g., [33,64]), and instantiations (e.g., [71]). Additional research is needed to determine what, in fact, actually works in practice. Much of this work should be empirical.

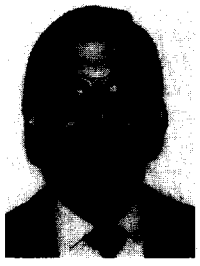
If, indeed, progress is to be made in system development, not only must we determine what works, we must understand why it works. There are virtually no generalizations or theories explaining about why and how (or even if) any of these artifacts work. Wand and Weber's research [72,73] on Ontology may provide a beginning to such a theory, however, significant work is needed before this can be considered to be a theory of system development. Further, any such theory must be justified scientifically before its principles can be introduced back into the artifacts and the cycle repeated.

References

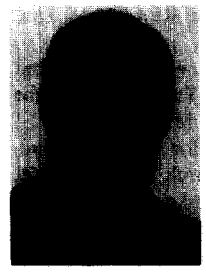
[1] P. Achinstein, Concepts of Science, Johns Hopkins Press, Baltimore, 1968.

[2] ACM Turing Award, Dennis Richie and Ken L. Thompson, 1983 ACM Turing Award Recipients, Communications of the ACM, Vol 27, No 8, August, 1984, p. 757.
[3] D. Batra, J.A. Hoffer and R.P. Bostrom, A Comparison of User Performance Between the Relational and the Extended Entity Relationship Models in the Discovery Phase of Database Design, Communications of the ACM, (33, 2), February, 1990, pp 126–139.
[4] W. Bechtel, Philosophy of Science: An Overview for Cognitive Science, Hillsdale, NJ: Lawrence Erlbaum, 1988.
[5] D.E. Bell, H. Raiffa and A. Tversky (eds.), Decision Making: Descriptive, Normative and Prescriptive Interactions, Cambridge, UK: Cambridge University Press, 1988.
[6] F.P. Brooks, Jr., No Silver Bullet: Essence and Accidents of Software Engineering, IEEE Computer, April, 1987, pp 10–19.
[7] P. Caws, The Structure of Discovery, Science, Vol 166, December 12, 1969, pp 1375–1380.
[8] H.C. Chan, K.K. Wei and K.L. Siau, Conceptual Level Versus Logical Level User-Database Interaction, Proceedings of the 12th International Conference on Information Systems, New York, Dec. 1991, pp. 29–40.
[9] P.P. Chen, The Entity-Relationship Model – Toward a Unified View of Data, ACM Transactions on Database Systems, Vol. 1, No. 1, March 1976, pp. 9–36.
[10] S. Christodoulakis, Implications of Certain Assumptions in Database Performance Evaluation, ACM Transactions on Database Systems, Vol 9, No 2, June 1984, pp 163–186.
[11] E.F. Codd, A Relational Model of Data for large Shared Data Banks, Communications of the ACM, vol 13, no 6, June 1970.
[12] D.W. Cornell and P.S. Yu, On Optimal Site Assignment for Relations in the Distributed Database Environment, IEEE Transactions on Software Engineering, Vol 15, No 8, August 1989, pp 1004–1009.
[13] R. Davis and D.B. Lenat, Knowledge-Based Systems in Artificial Intelligence, New York: McGraw-Hill, 1982.
[14] G. Davis and M. Olson, Management Information Systems: Conceptual Foundations, Structure and Development, 2nd Ed., McGraw-Hill, 1985.
[15] K.A. De Jong and W.M. Spears, Using Genetic Algorithms to Solve NP-Complete Problems, Proc 3rd International Conference on Genetic Algorithms, June 4–7, 1989, Morgan Kaufmann, Publishers.
[16] G. DeSanctis and R.B. Gallupe, A Foundation for the Study of Group Decision Support Systems, Management Science, Vol 33, No 5, 1987, pp. 589–609.
[17] G. DeSanctis and M.S. Poole, Capturing the Complexity in Advanced Technology Use: Adaptive Structuration Theory, Organizational Science, 1993, (to appear).
[18] P.F. Drucker, The Coming of the New Organization, Harvard Business Review, Vol 66, No 1, Jan-Feb, 1988, pp 45–53.
[19] P.F. Drucker, The New Productivity Challenge, Harvard Business Review, Vol 69, No 6, Nov-Dec, 1991, pp 45–53.
[20] F. Ferre, Philosophy of Technology, Englewood Cliffs, NJ: Prentice Hall, 1988.
[21] J.F. George, G.K. Easton, J.F. Nunamaker, Jr. and G.B. Northcraft, A Study of Collaborative Group Work With and Without Computer-Based Support, Information Systems Research, Vol 1, No. 4, June 1988, pp. 394–415.
[22] D.L. Goodhue, I/S Attitudes: Toward Theoretical and Definitional Clarity, Database, Fall/Winter 1988.
[23] D.L. Goodhue, User Evaluations of MIS Success: What Are We Really Measuring? Proceedings of the Hawaii International Conference on Systems Sciences, 1992.
[24] G.A. Gorry and M.A. Scott-Morton, A Framework for Management Information Systems, Sloan Management Review, October 1971, pp 55–70.
[25] J.J. Grefenstette, Optimization of Control Parameters for Genetic Algorithms, IEEE Transactions on Systems, Man and Cybernetics, Vol SMC-16, No 1, January/February, 1986, pp 122–128.
[26] J. Hartmanis and H. Lin, (Editors), Computing the Future: A Broader Agenda for Computer Science and Engineering, National Academy Press, Washington, D.C., 1992.
[27] C.G. Hempel, Philosophy of Natural Science, Englewood Cliffs, NJ: Prentice Hall, 1966.
[28] J.H. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, MI, 1975. pp 66–72.
[29] J.H. Holland, Genetic Algorithms, Scientific American, July 1992, pp 66–72.
[30] J.E. Hopcroft, Computer Science: The Emergence of a Discipline, Communications of the ACM, Vol 30, No 3, March 1987, pp 198–202.
[31] R. Hull and R. King, Semantic Database Modelling: Survey, Applications and Research Issues, ACM Computing Surveys, Vol. 19, No. 3, Sept. 1987, pp. 201–260.
[32] B. Ives, S. Hamilton and G.B. Davis, A Framework for Research in Computer-Based Management Information Systems, Management Science, Vol 26, No 9, September 1980, pp 910–934.
[33] S. Jarvenpaa and J. Machesky, End User Learning Behaviour in Data Analysis and Data Modelling Tools, Proceedings of the 7th International Conference on Information Systems, San Diego, 1986, pp. 152–167.
[34] M.A. Jenkins, Research Methodologies and MIS Research, in E. Mumford, et. al. (eds) Research Methodologies in Information Systems, Elsevier Science Publishers B.V. (North Holland), 1985, pp. 103–117.
[35] A. Kaplan, The Conduct of Inquiry, New York: Crowell, 1964.
[36] R.M. Karp, Combinatorics, Complexity and Randomness Communications of the ACM, Vol 29, No 2, February 1986, pp. 98–111.
[37] W. Kent, Data and Reality, North Holland, 1978.
[38] W. Kent, Limitations of Record-based Information Models, ACM Transactions on Database Systems, Vol. 4, No. 1, March 1979, pp. 107–131.

[39] K.L. Kraemer and J.L. King, Computer-Based Systems for Cooperative Work and Group Decision Making, ACM Computing Surveys, vol 20, no 2, June 1988, pp 115–146.
[40] T.S. Kuhn, The Structure of Scientific Revolutions, University of Chicago Press, 1970.
[41] P. Langley, H.A. Simon, G.L. Bradshaw and J.M. Zytkow, Scientific Discovery: Computational Explorations of the Creative Processes, Cambridge, MA: MIT Press, 1987.
[42] A.S. Lee, A Scientific Methodology for MIS Case Studies, MIS Quarterly, Vol 13, No 1, March 1989, pp 33–50.
[43] J. Leplin (ed.), Scientific Realism, Berkeley, CA: University of California Press, 1984.
[44] S.E. Madnick, The Challenge: To Be Part of the Solution Instead of Being Part of the Problem, Proceedings of the Second Annual Workshop on Information Technology, Dallas Texas, December 12-13, 1992.
[45] S.T. March, Techniques for Structuring Database Records, ACM Computing Surveys, Vol 15, No 1, March, 1983, pp 45–79.
[46] S.T. March, Research Issues in Information Technology, Keynote Address, Proceedings of the Second Annual Workshop on Information Technology Systems, Dallas, TX, Dec. 12-13, 1992, pp. 10–16.
[47] S.T. March and S. Rho, Allocating Data and Operations to Nodes in Distributed Database Design, IEEE Transactions on Knowledge and Data Engineering (to appear).
[48] S.T. March and G.D. Scudder, On the Selection of Efficient Record Segmentations and Backup Strategies for Large Shared Databases, ACM Transactions on Database Systems, Vol. 9, No. 3, September 1984, pp. 409–438.
[49] R.O. Mason and I.I. Mitroff, A Program for Research on Management Information Systems, Management Science, January 1973, pp 475–485. [50] P.E. Meehl, What Social Scientists Don't Understand, in D.W. Fiske and R.A. Shweder (eds.), Metatheory in Social Science, Chicago: University of Chicago Press, 1986, pp. 315–338.
[51] P. Mishra and M.H. Eich, Join Processing in Relational Databases, ACM Computing Surveys, Vol 24, No 1, March, 1992, pp 63–113.
[52] A. Newell and H.A. Simon, Computer Science as Empirical Inquiry: Symbols and Search, Communications of the ACM, vol 19, no 3, March 1976, pp 113–126.
[53] A. Newell and H.A. Simon, Human Problem Solving, Prentice-Hall, 1972
[54] R.L. Nolan, Managing the Computer Resources: A Stage Hypothesis, Communications of the ACM, vol 16, no 7, July 1973, pp 399–405.
[55] D.A. Norman, Cognitive Engineering, in D.A. Norman and S.W. Draper (eds), User Centred System Design, Lawrence Erlbaum Associates, Hillsdale, NJ, 1986, pp 31–61.
[56] T.W. Olle, J. Hagelstein, I.G. Macdonald, C. Rolland, H.G. Sol, F.J.M. Van Assche and A.A. Verrijn-Stuart, Information Systems Methodologies: A Framework for Understanding, Addison-Wesley Publishing Company, Wokingham, England, 1988.
[57] J. Parsons and Y. Wand, Guidelines for Evaluating Classes in Data Modelling, Proceedings of the Thirteenth International Conference on Information Systems, December 13–16, 1992, Dallas, TX, pp 1–8.
[58] M.S. Poole and G. DeSanctis, Microlevel Structuration in Computer-Supported Group Decision-Making, Human Communication Research, vol 19, No 1, 1992, pp. 5–49.
[59] K.R. Popper, Conjectures and Refutations: The Growth of Scientific Knowledge, New York: Harper and Row, 1963.
[60] R. Rorty, Consequences of Pragmatism, Minneapolis, MN: University of Minnesota Press, 1982.
[61] D.A. Schon, The Reflective Practitioner: How Professionals Think in Action, Basic Books, New York, 1993.
[62] M.A. Scott-Morton, (Editor), The Corporation of the 1990s: Information Technology and Organizational Transformation, Oxford University Press, 1990.
[63] M.E. Senko, E.B. Altman, M.M. Astrahan and P.L. Fehder, Data Structures and accessing in Data-Base Systems, IBM Systems Journal, Vol 12, No 1, 1973, pp 30–93.
[64] P. Shoval and M. Even-Chaime, Database Schema Design: An Experimental Comparison Between Normalization and Information Analysis, Database, Vol. 18, No. 3, Spring 1987-a, pp. 30–39.
[65] H.A. Simon, The Sciences of the Artificial (2nd ed.), Cambridge, MA: MIT Press, 1981.
[66] A.M. Starfield and A.L. Bleloch, Building Models for Conservation and Wildlife Management, New York: Macmillan, 1986.
[67] N.A. Stillings, M.H. Feinstein, J. L Garfield, E.L. Rissland, D.A. Rosenbaum, S.E. Weisler and L. Baker-Ward, Cognitive Science, Cambridge, MA: MIT Press, 1987.
[68] R.E. Tarjan, Algorithm Design, Communications of the ACM, Vol 30, No 3, March 1987, pp 205–212.
[69] T.J. Teorey, Database Modelling and Design, Morgan Kaufmann Publishers, Inc., San Mateo, CA., 1990
[70] T.J. Teorey and J.P. Fry, Design of Database Structures, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1982.
[71] I. Vessey, S. Jarvenpaa and N. Tractinsky, Evaluation of Vendor Products: CASE Tools as Methodology Companions, Communications of the ACM, Vol 35, No 4, April 1992, pp 90–105.
[72] Y. Wand and R. Weber, An Ontological Analysis of Some Fundamental Information Systems Concepts, Proceedings of the Ninth International Conference on Information Systems, Minneapolis, MN, Nov 30-Dec 3, 1988.
[73] Y. Wand and R. Weber, Toward a Theory of the Deep Structure of Information Systems, Proceedings of the Eleventh International Conference on Information Systems, Copenhagen, Denmark, December 16–19, 1990.
[74] R. Weber, Toward a Theory of Artifacts: A Paradigmatic Base for Information Systems Research, Journal of Information Systems, Vol 1, Spring, 1987, p 3–19.
[75] R. Weber and Y. Zhang, An Ontological Evaluation of NIAM's Grammar for Conceptual Schema Design, Proceedings of the Twelfth International Conference on Information Systems, New York, Dec. 1991, pp. 75–82.



Salvatore T. March is a Professor in the Information and Decision Sciences Department, Carlson School of Management, University of Minnesota. He received his BS, MS, and PhD degrees in Operations research from Cornell University. His primary research interests are Information System Development, Logical and Physical Database Design, and Information Resource Management. His research has appeared in *Information* and Management, *ACM Computing Surveys*, *ACM Transactions on Database Systems*, *Communications of the ACM*, *IEEE Transactions on Knowledge and Data Engineering*, *The journal of MIS*, *Information Systems Research*, *Information Science*, *Decision Sciences*, and *Management Scienc* . He has served as the Editor-in-chief of ACM Computing Surveys and is currently an Assiciate Editor for *MIS Quarterly*.



Gerald F. Smith is an Assistant Professor in the Department of Management, College of Business Administration, University of Northern Iowa, Cedar Falls, IA 50614-0125, USA. His primary area of research is managerial problem solving, with special interests in problem identification and definition, problem structures, and quality problem solving. Past research has appeared in *Decision Support Systems*, *Management Science*, *Organizational* Behavior and Human Decision Processes, *Omega*, and the *IEEE Transactions on Systems*, *Man*, and *Cybernetics*.