Design can be approached scientifically. However, before a formal science of design can be developed, several principles must be established, including the continuous nature of the design verification process.

Workshop Report:

The Science of Design

Mario J. Gonzalez, Jr.
University of Texas at San Antonio

"Systems theory appears to be driven by—and follows the development of—technology. For example, the theory of linear systems lagged far behind the actual use of those systems. Similarly, today our theory for the proper use of multidimensional, multivariable, distributed, nonlinear, testable electronics systems lags behind the actual implementation of these systems. The most widely touted system architecture today is distributed functional processing. The main challenges here are how complicated large scale systems can be distributed into optimal functional blocks, how systems which are already broken up into non-optimum and ad hoc functional blocks can nevertheless be used in optimum ways, and how dedicated processors/controllers can be used to synthesize analog and digital functions."

So stated William J. Dejka of the Naval Ocean Systems Center, addressing the Workshop on the Science of Design held last year at the University of Texas at San Antonio. Though he was quoting from a report (unpublished) on a workshop held nearly a year earlier, his remarks were directly relevant to the issue raised in this workshop.

"In the past," Dejka continued, "a system design criterion has been the minimization of the number of components. Entire theories have been erected based on the minimization of hardware components in producing a function. Later, the number of components was dropped and the number of 'states' was instituted as the predominant criterion.

"Today, instead, a dominant criterion is the time required to perform the function. Time has become the principal parameter in systems operation, particularly for military systems which must function in real time and yield correct decisions and actions within fractions of a second. It is necessary to determine measures of effectiveness of different implementations for producing the required functions. These measures should include the hardware involved, the algorithm used, and the system architecture—all interrelated.

"The second important measure is the regularity, or ordered nature, or structure of the implementation. This becomes important in considerations of system verification, testability and fault diagnosis among others.

"The concept of system regularity appears to be quantifiable," he continued, "inasmuch as successful efforts have been made in the past to quantify the regularity of data.

"The expansion of the idea to systems is a giant step, but it is conceivable and useful. Such systems would have superior testability and superior reliability and maintainability. These are also dominating criteria for future systems."

Dejka went on to emphasize the importance of quantifying the architecture selection process. The selection of an architecture to solve a particular problem should be based on an objective, quantifiable methodology instead of on a subjective assessment of problem requirements and architecture capabilities. With respect to the science of design, he said, this means that we need to quantify the "work" in an algorithm in addition to traditional parameters such as time constraints and storage. We need techniques for mapping algorithms to standard building blocks or new architectures. This mapping is an abstraction which should be quantified (formalized) to minimize subjectiveness and to promote automated translation from the requirements space to the solution space. This mapping should occur in the early or conceptual stages of design and should consider factors such as the cost of redundancy, operating and control procedures, I/O requirements, system partitioning, system and interface specifications, queueing associated with the allocation of shared resources, and subsystem integration. These techniques must occur as early as possible in the design process because it is much harder to correct problems and make things better after the design is well into the engineering cycle.

After these introductory remarks individual attendees presented their views on a particular aspect of the science of design based on personal experience, research interests, and the specific demands of an individual's work-related activities.

Major issues

During the first part of the workshop, attendees were asked to make brief presentations on some aspect of the science of design, and during these presentations, discussion and debate were limited. Upon completion of the presentations, key issues were identified and discussed. Because most of the attendees participated in each of these discussions, the following summary does not attribute specific comments to all contributors in each topic area. Instead, the author of the topic area is identified and a summary of the comments made during the discussion of each major issue is given.

0018-9162/79/1200-0112$00.75 © 1979 IEEE

Program generators (C. V. Ramamoorthy). The ultimate objective of program generators is to produce error-free programs automatically instead of manually in a manner characterized by a reduced reliance on software specialists. Even though the emphasis is obviously on a theory of software design, the principles suggested here may be extended to more general application areas. A key element of this objective is the construction of models that enable us to determine how successes and failures in our creative design efforts can be encoded in a knowledge base which becomes available later on in the form of a theory. With the proper theoretical base, knowledge can be compacted and used very effectively to simplify teaching and learning processes. Since this base does not exist, an essential first step is the use of interaction to provide the designer with design aids. That is, we need analysis techniques based on some sort of a theory or experience since that is where the foundations of design may be found. The design process can, therefore, be viewed as the process of conveying creative experiences into a form of symbolism that other people can use to advantage.

A cautionary remark from the audience suggested that full-fledged automatic program synthesis is not likely in the near future, although fairly general program generators are now available in a number of application areas (e.g., assemblers, editors, wire list generators), suggesting that perhaps in time more sophisticated products may be feasible. In the meantime, one thing that we need to do is to consciously incorporate personal tricks and practices into the theory of design, particularly if we seek to automate the design process.

A significant outcome of this discussion is that we need models and structures of design processed. We have many programs and tools available to us today, and in order to integrate these tools we need an overall structure, an overall model of design. One view of a design theory, then, is that of a structure that allows us to integrate all of the tools that are available to us.

Since it is apparent that experience is a vital part of the design process, it was suggested that design theory and the science of design are evolutionary processes whose achievement is best served through human interaction and the exchange of ideas and experiences.

Is there a science of design? (Raymond Yeh). An interesting question raised during this discussion is whether or not the issue of systematic design can be viewed as a science. Is it reasonable, perhaps, to compare the design process to the less well-defined but nonetheless exacting iterative processes employed by a sculptor? One definition of design is that it is the process of creating a form to satisfy a predefined specification. The initial step in design is always an artistic act which consists of a translation from a problem requirement to a form that appears to solve the problem. The next step is to quantify this form in a manner that is as rigorous and methodical as possible.

In software engineering the design process can be viewed as a hierarchical, interconnected structure in which the system structure is defined at the highest level. At the next lower level we find a definition of the design structure and a design evaluation methodology. In the design structure we find the design documentation and a statement of the design intention.

In view of this hierarchical structure the design process can be viewed as a family of designs characterized by a constructive approach in which intermediate decisions are documented and validated before initiation of subsequent steps. The documentation provides a history of decision processes and serves as the basis for changes suggested by the iterative evaluation process. In order to implement the verification objective throughout the entire software design process, at least two goals must be achieved. First, it is necessary to build mock-ups or logical models of the final product in order to give the user techniques for evaluating the reasonableness of a result and for verifying the correctness of the original specification. Second, the evaluation process must be defined in terms of quantitative criteria that can be used to objectively assess a result and thereby minimize any subjectiveness in the evaluation.

In summary, in order to develop a science of design we need to examine the elements of the hierarchical structure closely. Furthermore, model development should borrow from as many disciplines as possible.

Problem-driven systems; design correctness vs. proof of correctness (Franco Preparata). This discussion examined two major issues. The first issue suggests that theoretical but practically motivated studies have not been mapped into the realm of the practitioner. In effect, technology is far ahead of science. We are facing a world of parallel computation that is not well understood. What we need to do in order to design a system is to start with the problem—that is, we need to develop a methodology of problem-driven systems. For a particular problem we need to know which is the best system from the set of available alternatives, and we need to know why particular decisions are being made. If efficiency is reduced in order to achieve other design objectives, for example, then this trade-off should occur only as a result of an awareness of the implications of the trade-off. In order to evaluate the coupling between the problem and the solution space, therefore, it is necessary to consider more than performance issues.

In this area we need research in requirements mapping. That is, we need to identify how we can structurally analyze requirements and translate them into evaluation attributes and into design constraints. We need research in representation schemes that enable us to state requirements that can be analyzed and decomposed into subproblems that permit a more manageable and systematic analysis approach.

The second issue raised in this discussion restates and reaffirms a point made earlier by suggesting that it is better to verify intermediate results than it is to attempt to prove the correctness of a finished product. That is, no matter how elegant proving and testing techniques are, they cannot replace design correctness. This observation is even more valid when applied to parallel and distributed systems. These systems are so much more complex than serial systems, that verification of algorithms after the fact may be impossible. Anything that allows us to do design verification in a less time-consuming manner or produces a design that is inherently more testable is very important. In this regard, design constraints that result in better testability and better verification even though the hardware may be used less efficiently should be encouraged. This can be achieved by limiting the number of combinations of choices available to the designer. A note of caution: we cannot always use procedures that were developed for sequential systems and expect them to apply directly to parallel systems.

Abstractions for virtual architecture synthesis (James Howard). Throughout the workshop a need was expressed for a methodology and associated tools for requirements analysis for a given problem. Among the techniques proposed was the use of abstractions (functions and data abstractions) to obtain the data flow and precedence graphs for the problem from the problem specifications. Once these graphs are obtained, processor and monitor abstractions can be used to configure a series of virtual architectures. A subset of these architectures that satisfy both problem and system specifications can then be selected. If an architecture evaluation scheme is available, then such a scheme can be used to provide a ranking of the members of this subset.

The building blocks to be provided by increasingly sophisticated technology will not be totally suitable for distributed systems. Some effort is necessary to identify what the properties of basic building modules should be for the distributed processing environment and to encourage semiconductor manufacturers to incorporate these features into their products. A specific example would be the incorporation of a built-in-test capability into the processor module.

Algebraic theory for system evaluation (Mike Andrews). The theme of this discussion again served to highlight the need for quantitative techniques to evaluate alternative system structures as part of an overall objective of quantifying the science of design. The approach outlined here combines the topology of the structure derived from its connectivity matrix and the information flow between the nodes of the structure derived from a utilization matrix to produce results that optimize one or more measures. The theory proposed here obeys established algebraic laws and the laws of superposition and homogeneity, and permits the application of information theory concepts. This discussion served to point out that the use of mathematical techniques is an essential part of the science of design.

Automation of the design process (Dan Siewiorek). The design process can be said to undergo a series of evolutionary changes. Initially design proceeds in a non-systematic, ad hoc manner. After experience in a discipline is gained, design principles and metrics are obtained. A natural and desirable extension of these steps is the automation of the design process. This discussion dealt with the latter step by means of a data base of tools or building blocks.

The input to this automated process consists of a description of a target system (i.e., its specifications), a description of design trade-offs and constraints, and a description of the technology available. The output of this process is an optimized design in which parallelism is automatically detected and exploited to the fullest extent.

In order to implement this idealized scenario, we need to identify the basic building blocks and specify their I/O characteristics in an unambiguous fashion. In order to accomplish this we need to know what the parameters of these modules should be since we would like them to be reused and reworked later on, possibly to accommodate changes in the data structure, technology, etc.

A relevant point here is that certain design representations are easier to build with than others, and as a result, researchers should be invested to determine what representations should be used in a data base of this type. This topic can be expanded to allow for the fact that different representations may be appropriate at the various levels of the design process.

Other related research topics include the development of techniques for evaluating alternative designs and for implementing efficient search strategies for the building block data base.

An interesting sidelight here is the suggestion that attempts to automate the design process by means of a data base of building blocks or modules should incorporate inputs from a number of disciplines. Initial suggestions limited these disciplines to control theory and digital communications. The reason for suggesting the former is that control theory involves practices that are analogous to testing and design verification and involves control of complex dynamic systems that need a contolling mechanism similar to that of an executive in a computer system. Subsequent remarks expanded the original suggestion to include disciplines outside of the engineering and digital systems areas. The resultant interaction would provide different ideas and perspectives, and it would help us to determine if the science of design really transcends several engineering levels. Such an approach would enable us to characterize the design process by recognizing the common design approaches that people use in different fields. In so doing, we would be able to develop an integrated set of tools by codifying knowledge from several disciplines. We would also be able to consolidate requirements and provide a common method of communication.

Software physics (Leon Traister). This discussion dealt with an ongoing effort designed to quantify the capacity of hardware systems and the demands placed on a system by algorithms—that is, systems and application software units. This effort can be referred to by the term "software physics," defined as the study of the quantitative and measurable properties of executable instructions and their operands and their interactions with computing systems equipment and configurations. In effect, this effort is designed to predict and control performance or software power, defined as the amount of work done per unit time. This effort seeks to influence the science of design by quantifying work demands as a function of software parameters such as program length, levels of nesting, the number of times a module is invoked, and the number of iterations in a repetitive structure. If it is possible to quantify the demands of a workload and the capacity of a hardware structure (by means of a configuration capacity handbook, for example), then it is possible to predict the response of the hardware as a function of the demand.

Notice that the ability to express the demands of an algorithm in a measurable manner suggests that it should be possible to compare the efficiency of different solutions to the same problem. This suggestion is not totally valid, however, since it fails to consider the characteristics of the target machine. A given algorithm cannot be optimized until its impact on a target machine is known.

At the present time this effort is limited in scope since consideration has been limited primarily to issues of performance. In the attempt to influence the science of design, additional elements of the evaluation space such as responsiveness, reliability, availability, and modularity will be considered.

Design aids (Larry Jack). In examining the elements of a science of design, consideration must be given to the nature of the design effort. This discussion suggested that these aids must be user oriented and must focus on what have until recently been secondary issues such as software, testability and fault tolerance, and life cycle cost. A major concern is that in the top-down approach to design it is easy to abstract the requirements of the interfaces of the lower levels. In the actual implementation, however, the details of design must be addressed. In doing this it is often discovered that the available tools incorporate no provisions for the impact of these details in terms of time, cost, and perhaps a basic inability to satisfy design objectives because no provisions were made or no facilities were available for imbedding the necessary capabilities in the original design. What is needed, therefore, is the development of design tools that permit the details of implementation to be incorporated at the interface between levels of a top-down design.

In addition, design aids should provide for optimization at the system level. That is, the total interrelationship of all system components should be considered instead of treating these components separately.

Measurement of structural content (Ralph Gonzalez). The objective of this effort is to measure the structural aspects of algorithms with emphasis on the manner in which the structure of the algorithm affects its testability and the fundamental difficulty which is required in order that it may be understood. This approach is in contrast to the analysis of algorithms in which the concrete computational complexity of an algorithm is judged by the amount of resources it uses during its execution, expressed as a function of some parameter(s) of the input data. In analysis of this latter type, structure is viewed as important only in the way it impacts execution time resources.

The effort described here, on the other hand, deals with general systems, of which computer algorithms can be considered a powerful subset. Structural entropy is a function computed on a relational table system description that attempts to quantify the structural complexity for purposes such as comparing one system with another and making initial judgments about the relative ease of testing and maintaining a system.

Structural entropy or structural content is an area that is very important to the theory of design. It is necessary to define exactly what it is and how it can be used in terms of both algorithms and systems.

The influence of data communications (Tilak Agerwala). The central theme here is that we need to incorporate communications and control issues into the science of design. The reason: for many systems communications places tighter bounds on performance than operation dependency considerations. Actual communication time is very dependent on the physical interconnection of the elements of a parallel structure, and the amount of communication can be

ENGINEERING

SOFTWARE ENGINEERS
SYSTEM ARCHITECTS

SYTEK Inc., has immediate opportunities for talented professionals who have the ability to work as:

*Operating System Architects
*Microprocessor System Designers
*Communication Protocol Designers
*Software Development Techniques Specialists

SYTEK, Inc., is a young company of skilled professionals providing high technology systems engineering services to selected government and commercial clients.

We are looking for articulate, academically trained individuals with a strong practical outlook to provide the skills necessary to meet current and projected business demands.

We offer an excellent compensation package which includes incentive bonus, medical/dental insurance, tuition reimbursement, as well as a pleasant, stimulating work environment.

If you are willing to meet the challenge of technological innovation, we invite you to send a resume to:

SYTEK 1153 Bordeaux Dr.
Sunnyvale, CA 94086
Attn: Linda Kyle or

Call (408) 734-9000 for immediate consideration.

SYTEK, An Equal Opportunity Employer

JOB OPPORTUNITIES

Sandia Laboratories is seeking MS and PhD level applied computer scientists and computer related electrical engineers to assist in implementing a major change in architecture to its central computing facilities. Specific areas of interest include:

High Speed Computer Communications
Operating System Design
Computer Security
Computer Systems Simulation
File Access and Control

Preference will be given to those individuals with experience on CDC 6000 and 7000 series computers. Please send resume and salary history to:

John Wheeler
Division 3531
Sandia Laboratories
Box 5800
Albuquerque, N.M. 87185

Sandia Laboratories is located in the heart of the sunny Southwest and offers a liberal fringe benefit package containing 24 days annual vacation. Recreation opportunities in and around Albuquerque abound.

Sandia Laboratories
Equal Opportunity
Affirmative Action Employer M/F
U.S. Citizenship Required

affected significantly by the manner in which data is distributed, by the manner in which the problem is partitioned, and by the number and types of resources in the system. In addition to traditional performance measures, communications also impacts other measures such as availability, reliability, and fault tolerance, and the coupling between parts of a system can affect the fault containment properties of the system. To a significant extent communication can also affect the type of control (that is, operating system) alternatives that can be used in a particular system.

In seeking to incorporate communications into the science of design, however, we find a serious lack of acceptable models. These models will have to consider the communications demands of algorithms and the extent to which communications is likely to be a bottleneck. To restate comments made earlier, we need a method for predicting system performance on the basis of system characteristics.

Additional communication aspects of the science of design suggest that communications bottlenecks in parallel structures may not reduce system performance as much as the contention caused by the sequential access to shared data structures imposed by software locks. An area often overlooked in model design is the problem of getting data into and out of a system. System models often assume that the data is available and concentrate on computational issues. Similarly, data communication bandwidth can be a serious limiting factor in system reconfiguration upon occurrence of a fault.

Software science (Maurice Halstead). In order for a set of systematic activities to be referred to as a science, it is necessary that these activities possess certain characteristics. For example, they should be experimental in nature—that is, they should provide metrics by means of which objective evaluations can be made. In addition, these metrics should be supported by a theoretical foundation, and in order to achieve the desired objectiveness the metrics should be quantitative. Finally, a science should be predictive, and experiments in that science should be reproducible.

Software science possesses these characteristics and, therefore, qualifies as a branch of natural science. Software science seeks to predict software properties solely as a function of some static properties on the software itself, such as the number of operands and the number of operators. A significant outcome of ongoing effort in this area has been to show that programming effort grows as the square of the volume (i.e., the size of the problem). Using the techniques of this emerging science it has been possible to quan-

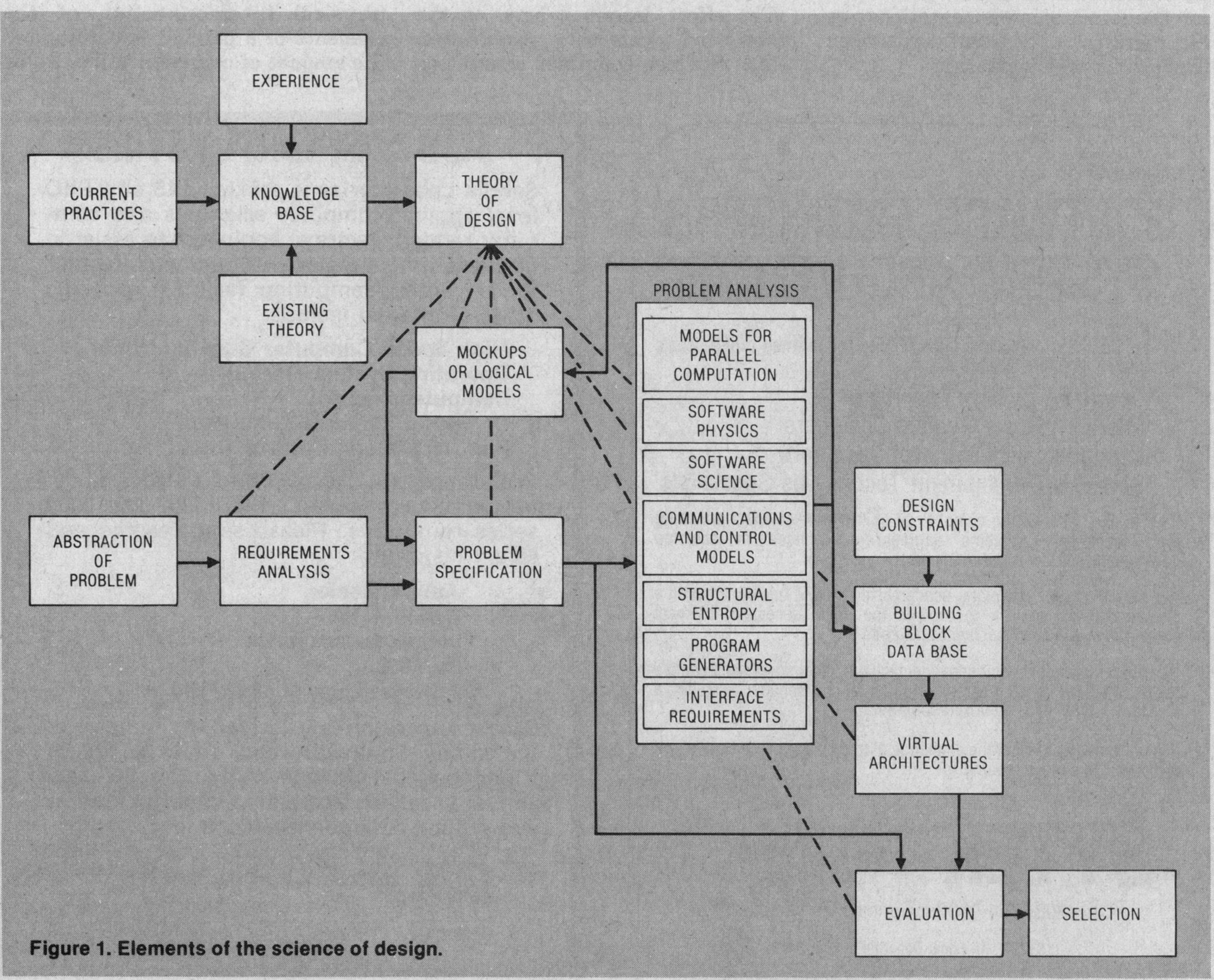

Figure 1. Elements of the science of design.

titatively confirm established approaches that suggest that a large program should be broken up into modules. Another outcome suggests that the size of an operating system can be predicted as a function of the number of allocatable resources. The addition of one more allocatable resource, for example, can more than double the size of an operating system.

Software science suggests that given the right inputs, we can predict things like the time required to generate a particular program, the number of errors that the resultant product is likely to have, the effort (in a quantitative sense) required to generate a program, and the "complexity" of a program.

Like the software work concept introduced earlier, software science seeks to provide a better understanding of some key issues that we encounter as part of a larger science of design.

Attributes for evaluation (Mario Gonzalez). This discussion addressed the previously raised issue concerning the quantitative assessment of architectures with respect to the demands of a particular problem. This need is becoming more acute in view of the emergence of distributed architectures and classification schemes or taxonomies. The objective of the approach outlined here is to produce a figure of merit to provide a ranking for architectures that are considered as candidates for the solution of a particular problem.

The first step in the approach is to identify a set of evaluation attributes that can be used to characterize system behavior in a problem-independent manner. The second step is to objectively and quantitatively indicate to what extent each distributed structure possesses these attributes and then to indicate the importance of these attributes to the problem at hand. In the final step these quantitative parameters are combined to produce the desired ranking.

Before this approach can be implemented, two major difficulties must be overcome. First, the size of the attribute space must be reduced to manageable proportions. Second, it will be necessary to divide the workload or problem space into classes or intervals since the ranking of attributes is not constant for all variations of the problem. Instead, it varies as a function of the demand or workload.

Summary

The following are the key issues or findings identified in this workshop. Where appropriate, related research topics are also given.

1. We need to develop program generators to produce error-free programs automatically. Research item: develop models and structures of design processes.

2. Design verification must be implemented throughout the entire design processes. Research items: (1) Develop logical models to permit verification of a specification. (2) Develop quantitative criteria to permit system evaluation.

3. We need to develop a methodology for the design of problem-driven systems. Research items: (1) Requirements mapping—that is, structurally identify requirements and translate them into evaluation attributes and into design constraints. (2) Develop realistic models for parallel systems.

4. It is better to verify intermediate results than it is to prove the correctness of a finished product.

5. We need a design methodology that gives us testable designs.

6. Correctness and an understanding of the design process must be addressed at an early stage in the life cycle of a system.

7. The building blocks provided by advanced technology will not be totally suitable for distributed systems. Research item: Identify the properties of basic building modules for distributed systems.

8. A natural and desirable extension of traditional design approaches is the automation of the design process. Research items: (1) Determine what representations should be employed in a data base used to automate the design process. (2) Develop techniques for evaluating alternative designs.

9. The science of design should be based on common design approaches from a number of different disciplines.

10. The efficiency of an algorithm cannot be maximized unless the characteristics of a target machine are identified. Research item: Develop techniques for identifying and matching algorithm and architecture characteristics.

11. We need to develop design tools that permit the details of implementation to be incorporated at the interface between levels of a top-down design.

12. Design aids should provide for optimization at the system level.

13. Structural entropy or content must be defined exactly in order for it to be used in terms of both algorithms and systems.

14. We need to incorporate communications and control issues into the science of design. Research item: Develop models that consider the communications demands of algorithms and the extent to which communications is likely to be a bottleneck.

15. We need to determine the extent to which the techniques and practices of software science are applicable on a larger scale in the science of design.

16. We need to quantitatively assess the suitability of individual distributed structures to the demands of a particular problem. Research item: Identify key evaluation attributes and quantify the extent of their presence in a system and the extent of their importance to a problem.

17. We need to explore the relationship between reliability and fault tolerance. Research items: (1) Develop a theory for testing. (2) Determine how a network of computers can be configured to be fault tolerant.

The ideas proposed in this workshop can, in a very preliminary manner, be used to map a requirement into a final design in the manner shown in Figure 1. ■

Mario J. Gonzalez, Jr., has been on the faculty of the University of Texas at San Antonio since 1977, where he is an associate professor of computer science. From 1973 to 1977 he was on the computer science faculty at Northwestern University. From 1972-1973 he was a member of the technical staff in the Advanced Systems Group at Texas Instruments. Gonzalez is a past co-chairman of the Technical Committee on Computer Architecture. His primary interests are in the general area of computer architecture with emphasis on distributed computer systems. Gonzalez received the BS, MS, and PhD in electrical engineering from the University of Texas at Austin in 1964, 1969, and 1971, respectively.