NASA Conference Publication 2015

(NASA-CP-2015) STANDARDIZATION, CERTIFICATION, MAINTENANCE, AND DISSEMINATION OF LARGE SCALE ENGINEERING SOFTWARE SYSTEMS (NASA) 103 p HC A06/MF A01 CSCL 09B G3/61
N77-33876
Unclas 50892

Standardization, Certification, Maintenance, and Dissemination of Large Scale Engineering Software Systems

A panel discussion held at
The George Washington University
Washington, D.C.
March 29-31, 1976

NASA Conference Publication 2015

Standardization, Certification, Maintenance, and Dissemination of Large Scale Engineering Software Systems

Editors

Theodore G. Toridis, *The George Washington University*
Harvey G. McComb, Jr., *NASA Langley Research Center*
Khalil Khozeimeh, *The George Washington University*

A panel discussion held as a part of the Second National Symposium on Computerized Structural Analysis and Design sponsored by NASA Langley Research Center, Hampton, Virginia, and the National Science Foundation, Washington, D.C. at The George Washington University, Washington, D.C., March 29-31, 1976

NASA
National Aeronautics
and Space Administration

Scientific and Technical
Information Office

1977

FOREWORD

The Second National Symposium on Computerized Structural Analysis and Design was held on the campus of The George Washington University on March 29-31, 1976. The purpose of the Second Symposium was to facilitate the exchange of information among structural analysts and design and research engineers on recent developments in structural engineering and applied mechanics with special emphasis on the utilization of computers in the analysis and design of structures and their components.

As a part of the symposium, a panel discussion was held on Standardization, Certification, Maintenance, and Dissemination of Large Scale Engineering Software Systems. The panel moderator and panel members are listed on the following page. The panel discussion consisted of two basic parts:

PART I - PRESENTATION BY PANEL MEMBERS: Each member of the panel presented a brief discussion on a specific topic associated with the theme of the panel discussion.

PART II - AUDIENCE RESPONSE AND DISCUSSIONS: This consisted of a cross-discussion between the audience (symposium participants) and the panel members on topics that were either discussed earlier by the panel members or that were pertinent to the discussion.

The discussions by panel members and the response by the audience were recorded on magnetic tape. In view of the interest shown by many individuals both during the meeting and subsequent to the symposium, a written documentation of the panel discussion was prepared although the proceedings of the symposium itself will not be published. To a great extent, this report represents a verbatim transcript of the presentations of the panel members and the responses of the audience. Mr. Mehmet I. Basci, Graduate Research Assistant at The George Washington University, helped in the drawing and preparation of the figures appearing in the publication.

Theodore G. Toridis

Theodore G. Toridis
Conference Co-Chairman
Professor, School of Engineering
and Applied Science
George Washington University
Washington, D. C.

PANEL MEMBERS

MODERATOR

Harvey G. McComb, Jr.
Head, Computer Aided Methods Branch
Structures and Dynamics Division
NASA Langley Research Center
Hampton, Virginia

Stanley D. Hansen
Manager, Flexstab System and
Advanced System Development
Boeing Computer Services
Seattle, Washington

Robert E. Nickell
Supervisor, Design Technology
Division, Nuclear Fuel Cycle
Development Department
Sandia Laboratories
Albuquerque, New Mexico

Steven J. Fenves
Professor, Department of Civil
Engineering
Carnegie-Mellon University
Pittsburgh, Pennsylvania

Harry S. White, Jr.
Associate Director for ADP Standards
Institute for Computer Sciences and
Technology
National Bureau of Standards
Washington, DC

James R. Johnson
Aerospace Engineer
Air Force Flight Dynamics Laboratory
Wright Patterson Air Force Base,
Ohio

Michael P. Gaus
Head, Engineering Mechanics
Section
Engineering Division
National Science Foundation
Washington, D. C.

SECOND NATIONAL SYMPOSIUM ORGANIZATION

PART I - PRESENTATION BY PANEL MEMBERS

Harvey McComb, NASA Langley Research Center, Moderator:

Good afternoon ladies and gentlemen. Welcome to our panel discussion on Standardization, Certification, Maintenance, and Dissemination of Large Engineering Software Systems. We are aware that the field of computerized structural analysis and design is growing at a very rapid rate. It has been described as chaotic and undisciplined. This situation is natural, I suppose, in a field so relatively new and where many bright, aggressive, and innovative people are involved. It means that people trying to make effective use of computerized methods have many options available, but it also means that difficult choices must be made both in terms of software and hardware. Many people feel that it is important to bring some sort of order or discipline into the field if it can be done without stifling too much the rich variety and innovation. To try to get some handle on the subject, we have asked the people in the panel to address specific areas in their opening remarks. This breakdown is not clear but rather arbitrary and overlapping. Nevertheless, perhaps, it will help focus our thoughts. I will introduce the panel members now.

Starting on my right we have Stan Hansen, who is Manager of Flexstab System and Advanced System Development for Boeing Computer Services. I would like to ask Stan to address himself to Program Design and Development.

Next is Harry White, who is Associate Director for ADP Standards of the Institute for Computer Sciences and Technology of the National Bureau of Standards. He will be talking about Standards and Documentation.

Robert Nickell, Supervisor of the Design Technology Division of Nuclear Fuel Cycle Development Department at Sandia Laboratory. Bob will be discussing Software Validation and Certification.

Then on my left is James Johnson, Aerospace Engineer at the Air Force Flight Dynamics Laboratory. Jim will address himself to Dissemination, Portability, and Maintenance.

Next is Steve Fenves, who is Professor of Civil Engineering at Carnegie-Mellon University, and I asked Steve to address the subject of Education.

Then on the far left is Mike Gaus, Head of the Engineering Mechanics Section of the Engineering Division of the National Science Foundation.

I want to give the panel members a chance to make opening remarks first. This will take us up to the coffee break. After break we will allow the panel to react to what they have heard from each other and make any other comments they wish. Then we will open up the discussion to members of the audience.

We do hope to put out a publication which captures the essence of this discussion and we are trying to tape the discussion here. (Please keep that in mind when you make your comments.) In addition, if there are things later on, if you would like to send us a written digest of remarks or comments you would like to make, we will be glad to include this material in our published volume if it seems appropriate.

Ok, we will get started. We will start off with Stan Hansen on my right. Stan, as I mentioned, will be talking about Program Design and Development.

Stan Hansen, Boeing Computer Services:

I should place myself in context, that is, I have worked in the aerospace industries since the initiation of my career in this field. We deal with very large systems. Therefore, what I am going to say will seem a little bit different to some of you who are dealing with smaller systems. I keep figure 1 on my bulletin board. When I began my present assignment, the subsystem of which this subroutine is a part was being documented. This and several other flowcharts like it had been prepared for inclusion in the documentation. This flowchart would be humorous if the process that produced it had not been so seriously deficient.

This particular subsystem had been developed on an emergency basis. A programmer had been brought in on loan from another organization, told there were no specifications, no time for design, that he was to take his direction from the other engineers and programmers, and to finish in three months. Nineteen months later, when I came on the project, the programmer was still working. He had generated 15,000 new coding statements in addition to the 10,000 that had existed.

We managed to close out the work in another two months. Today, eighteen months later, we are finding that the code in this subsystem is so much more complex than the functions being computed, in many cases, it is more cost effective for us to reprogram than to attempt to maintain.

And this is the point: complexity. Engineering software systems are complex. They become unmanageable if the complexity is out of control, as illustrated by the above example and by figure 2. Uncontrolled complexity is a major contributor to today's software problems (fig. 3):

- Unsatisfied user requirements
- Unpredictable costs and schedules
- Low technical quality
- Functional unreliability
- Fragile software
- Difficult maintenance

Complexity seems to be produced by three factors (fig. 4):

- The number of components interacting
- The number of interactions between components
- The complexity of the interactions

These three factors apply whether the components are individuals, organizations, engineering specifications, software design, code, etc. If the software system and the organizations and functions which produce and maintain it are to be controlled, they must be kept simple enough for a single man's intelligence to grasp completely that part for which he is responsible.

The objective, as shown in figure 5, is to, first, develop a software system design of minimum complexity and, second, to manage the remaining complexity to achieve system quality and control costs and schedules. This point is further illustrated in figure 6. There will exist a bounded design space within which the system requirements are satisfied. Within this system space, the least complex design will be the better design. Some important factors in achieving this objective are discussed in the following paragraphs.

MANAGEMENT ATTENTION

Figure 7 is a common diagram, but appropriate. The ability of managers and technical developers to influence the outcome of a project is very high at the outset but declines rapidly as commitments are made, schedules and budgets are used up, and code and documents are produced. If ideas, decisions, commitments, etc., are not well defined and communicated, management attention will be required, as shown by the curve dominant on the right side of figure 7. Disproportionate attention will be required at acceptance when the disparities between requirements and system capabilities are found and again during use when the system is applied to problems other than those demonstrated at acceptance. As shown, these are the very times when little ability remains to influence the outcome of the system except by rebuilding the deficient parts.

The goal is to give management attention as shown by the middle curve on the left hand side of figure 7. This curve follows the same form as the curve representing ability to influence outcome (i.e., management attention is given to the system development at the time of greatest effectiveness).

It is important to note that a marked distinction should be made between management, of which there is often too little, and manager presence, of which there is often too much. Management is that work involved in planning, organizing, leading, and controlling the project, regardless of who does it.

INFORMATION REQUIREMENTS

The general information requirements are shown in figure 8. These requirements are shown as isolated islands of "things, knowledge and data" pertaining to:

Problem characteristics

External system characteristics

Internal system characteristics

Generally, a great deal is known about each of these, but this knowledge is, in the general case, diffused among many individuals and organizations. More importantly, it is not organized relative to the new system being developed.

To amplify, new systems are usually developed as a further increment in capability and productivity in an existing environment. Users in the problem solving environment possess a great deal of information essential to the success of a new system. Similarly, those who have advanced the engineering technology will have developed new external system characteristics compatible with that advancement. Further, new advances will have been made in computing hardware and operating systems, software languages, control logic, numerical methods, etc. that will have an important effect on the internal system characteristics. The task in developing a new system is the organization of this information.

TOP DOWN SYSTEM DEVELOPMENTS

The organization of information cannot be achieved until two conditions exist:

1. Communication is established between the isolated islands.
2. A disciplined procedure is imposed for organizing and iterating between and within the isolated islands.

The approach is that shown in figure 9.

1. A problem analysis is performed to establish a model for the process or problem the new system is to support or solve.
2. A requirements definition is made based on the problem characteristics that were organized as a result of the problem analysis. This requirements definition establishes usage procedures, technology, and relationships that form the external system characteristics, or, more conventionally, the engineering specifications.

3. The external system requirements thus organized form the foundation for performing the various levels of system requirements. The system can then be implemented.

This bridging and organizing process is illustrated in figure 9. The bridges shown are two way. They will be much traveled while the system characteristics are iterated to achieve an optimum implementation. This process of organizing and iterating prior to implementation is essential and should consume about fifty percent of the project budget and schedule. A discipline is required, however, for organizing and iterating information in a consistent manner that will converge to the objective stated in figure 5. This discipline is the structured approach.

THE STRUCTURED APPROACH

There are three basic elements in the structured approach (figure 10):

1. A hierarchal grouping, or tree structure, is formed by decomposing general statements into levels of more detailed statements. This grouping must be complete (i.e., all functions or processes to the lowest level must be shown) and it must be self consistent (i.e., the functions or processes shown must be implementable). When complete, this grouping represents the static relationships of data.

2. Each node in the hierarchal grouping has the characteristics defined by the basic node. In the illustration of figure 10, the basic node is a process. The process is fed by known

control, data, and devices and produces a known output. Other basic node characteristics may be defined as required.

3. The nodal relationships are established by analyzing control and data communication between the nodes at each level in the hierarchal grouping. When complete, this analysis establishes the dynamic relationships of the system.

The process begins with a very general but complete statement of the output required of the new system. The process by which the output can be obtained is then defined. Necessary control data and device information, again general but complete, are then determined. The process in the top level node is then decomposed into its component parts and each of these is analyzed in precisely the same manner as top level node. In addition, the dynamic relationships between nodes are analyzed to determine control structures, data structures, and device requirements. Decomposition is continued downward until all primitives necessary for implementation of software are defined and designed.

This structured approach then forms the basis for organizing all of the work and products associated with the system development (i.e., management of the system development becomes possible and takes on a meaningful role).

SYSTEM MANAGEMENT

The work of management, as shown in figure 11, is to plan, organize, lead, and control. Through the structured approach, a work breakdown structure can be established for each stage of the system development.

The work breakdown structure forms the basis for estimating schedules and budgets and for making work assignments. Since, in the structured approach, the design process and its documentation are always highly visible, control can be exercised over budgets, schedules, and quality through meaningful reviews as the system is decomposed and its components designed. Since the development proceeds in an orderly controlled manner always within the intellectual grasp of the developers, the system always remains manageable. The complexity that results from plunging immediately into the details of the system has been replaced by a controlled step-wise procedure in which each step is no more complex than the previous one.

In conclusion, this process is not new. It is used routinely under other titles for the organization of complex information and for the development of essentially all fabricated products from lawnmowers to airplanes to satellites. Everyone who has used a dictionary, obtained a part from a Sears store, or used engineering drawings has used the structured approach. Its application to software development is recognition that software is a fabricated product. It has been difficult to think of software as a fabricated product since the medium of expression is not metal or glass or plastic. In fact, there is no parallel to these in the software product. The medium of expression is logic and language and mathematics. The formation of the product consists wholly of organizing this medium of transforming data successively through a series of processes until the output requirements are met.

The use of the structured approach makes possible a fabrication of the software product in which efficiency, modularization, extendability, reliability, and maintenance can be meaningfully designed into the system. Thank you.

SUBROUTINE SHRK

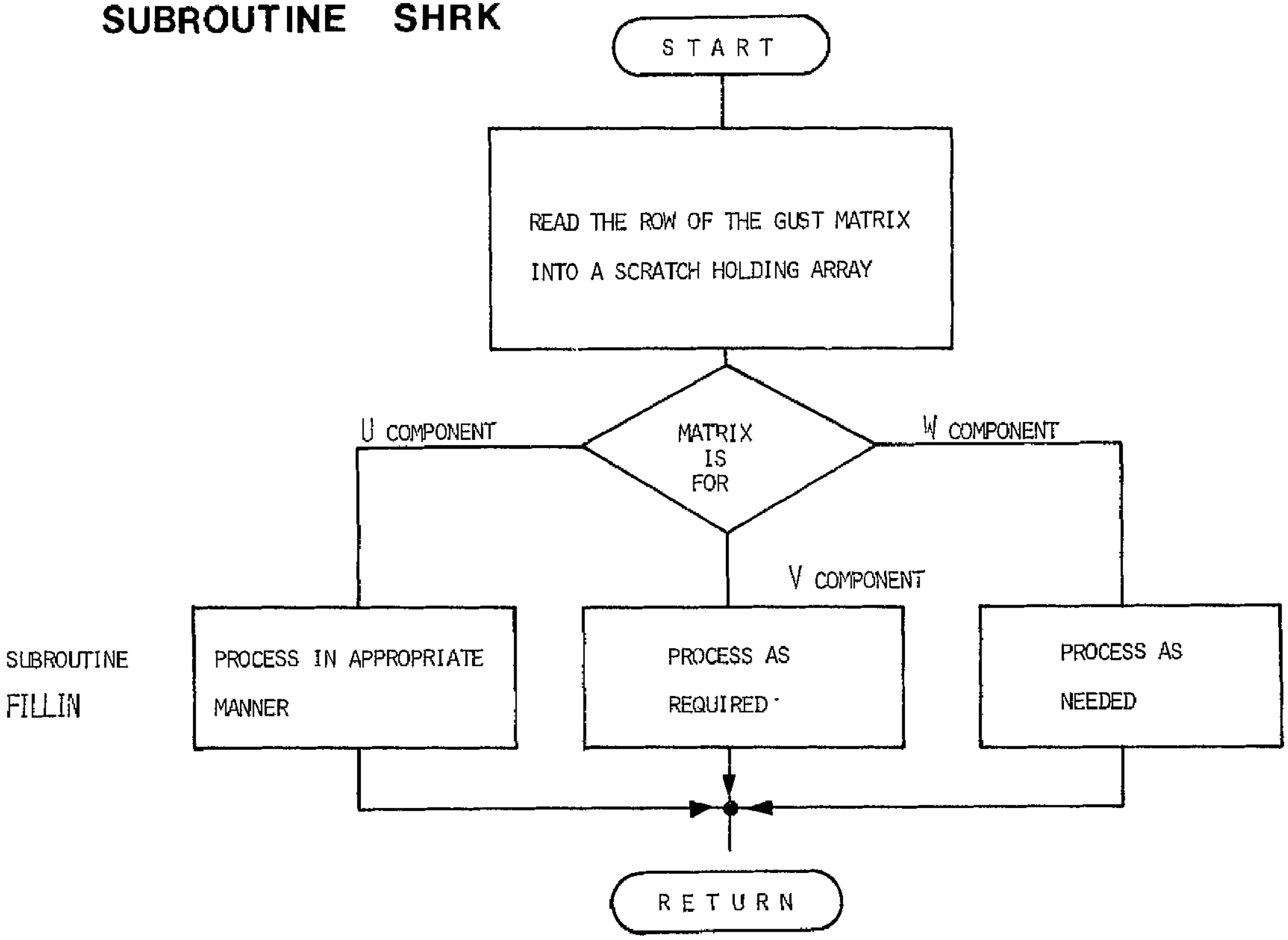


Figure 1

AUTOMATED ENGINEERING SYSTEMS ARE COMPLEX

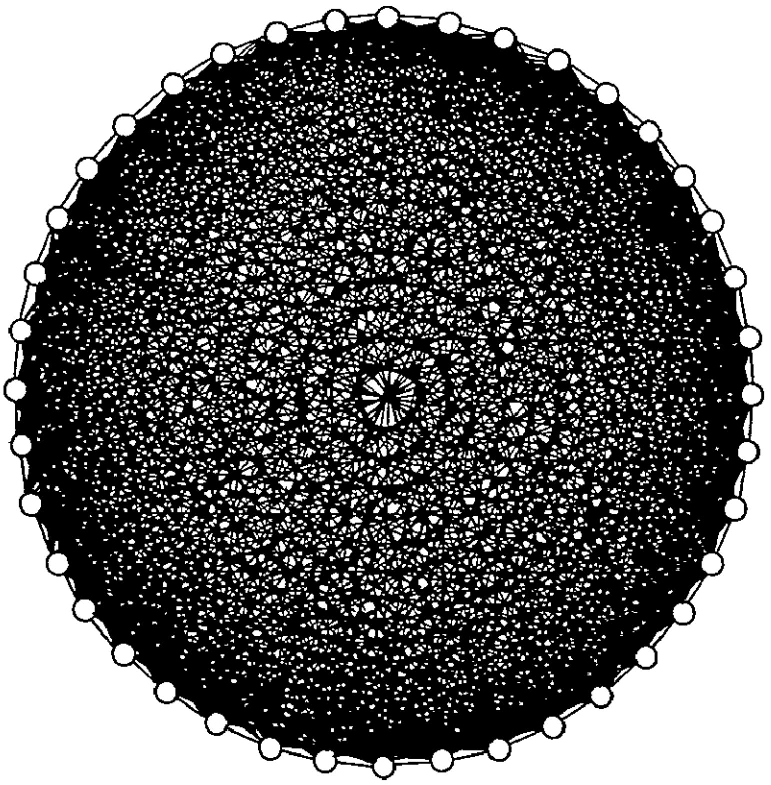


Figure 2

PROBLEMS RESULTING FROM CONTROLLED COMPLEXITY

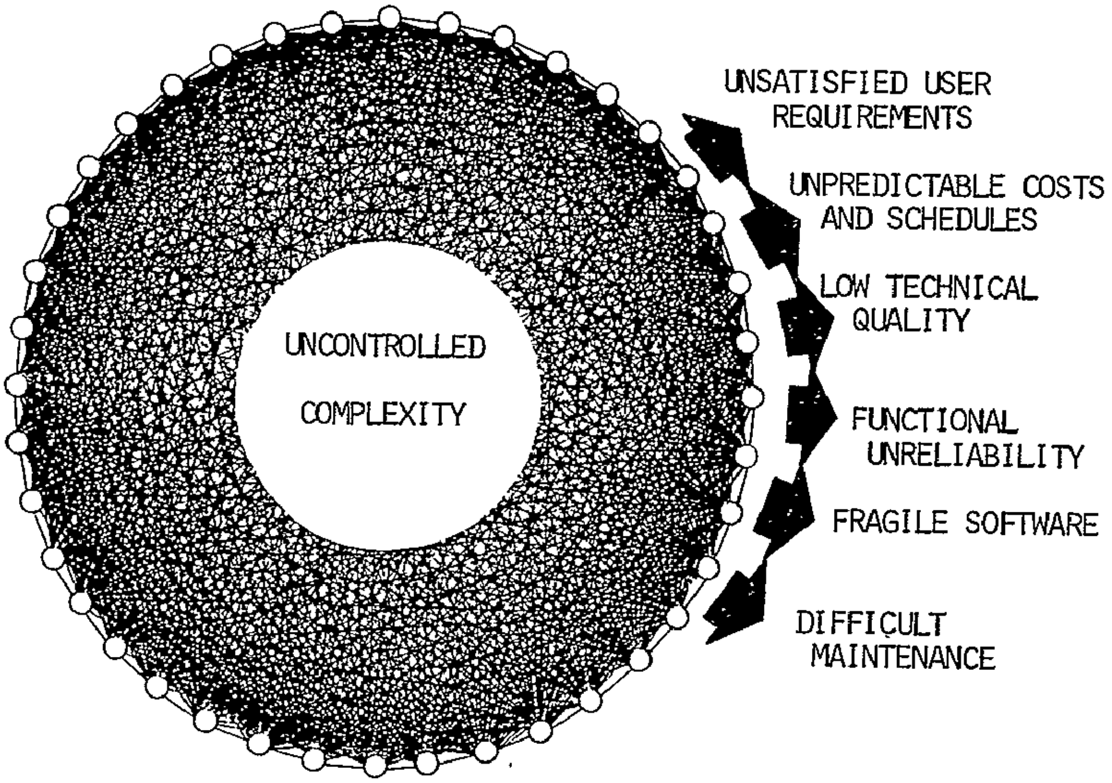


Figure 3

SOURCE OF COMPLEXITY

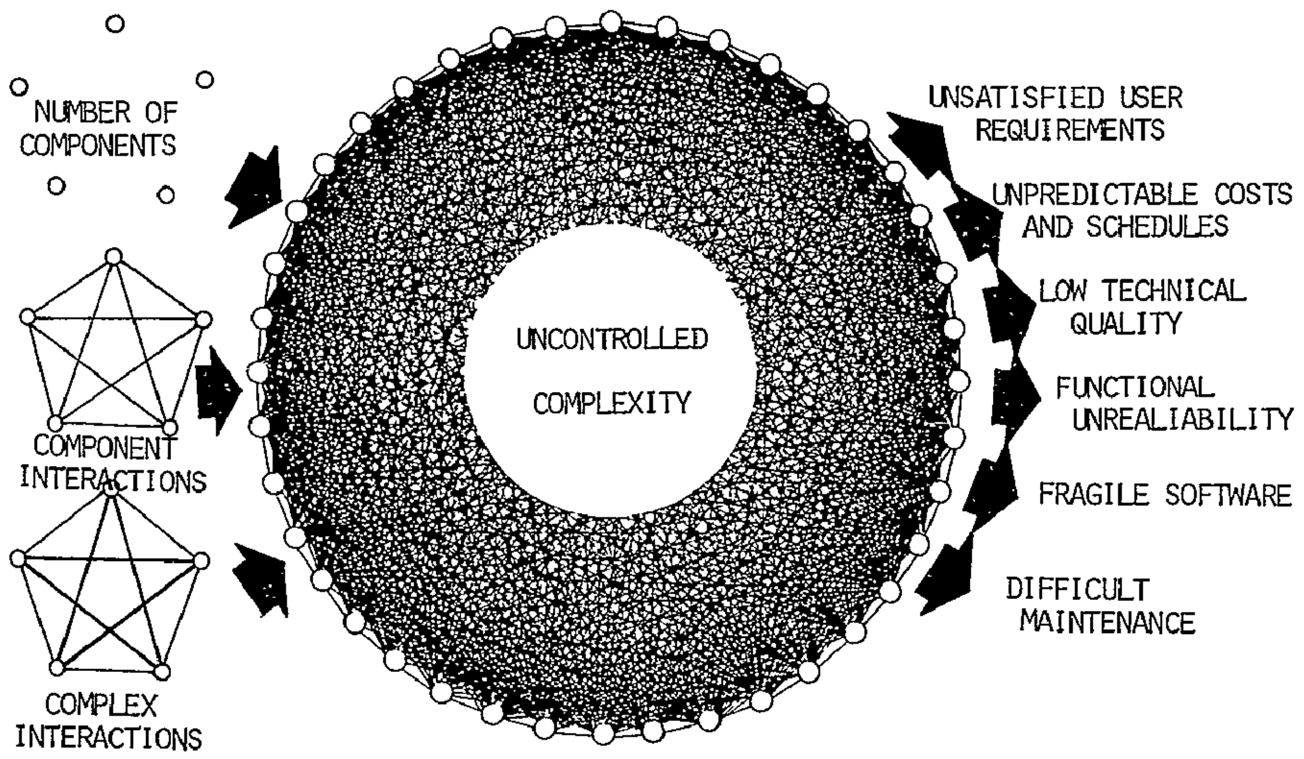


Figure 4

SOFTWARE DEVELOPMENT OBJECTIVES

DEVELOP A SOFTWARE SYSTEM DESIGN OF MINIMUM COMPLEXITY

MANAGE THE REMAINING COMPLEXITY TO ACHIEVE SYSTEM

QUALITY AND CONTROL COSTS AND SCHEDULES

Figure 5

DEVELOP MINIMUM COMPLEXITY DESIGN

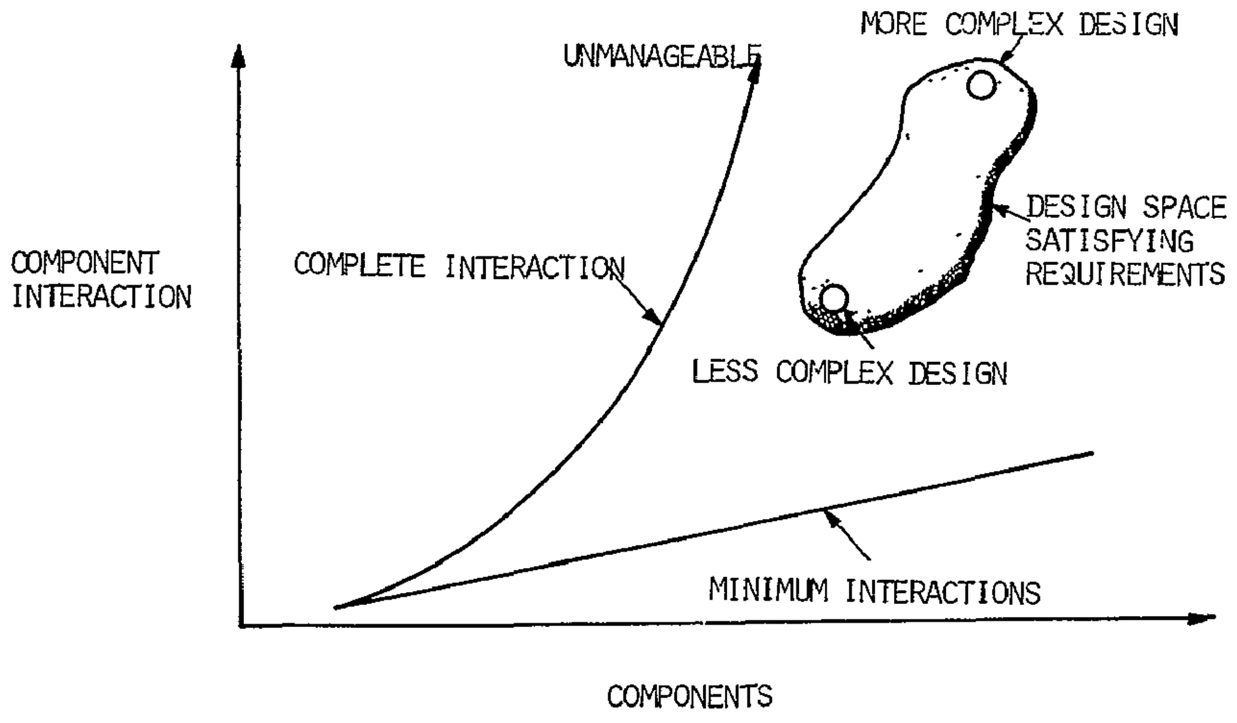


Figure 6

TYPICAL SOFTWARE SYSTEM MANAGEMENT CHARACTERISTICS

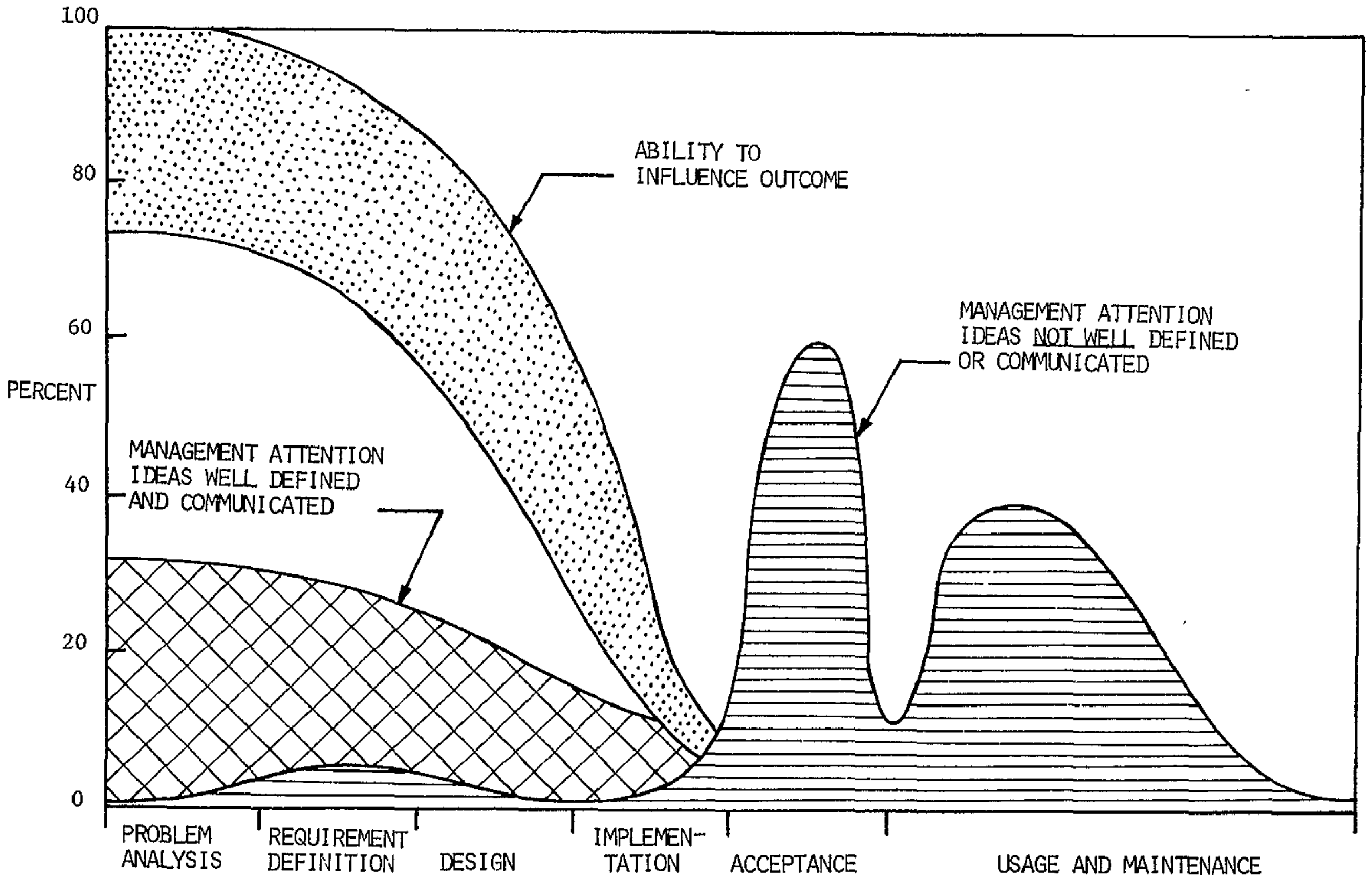
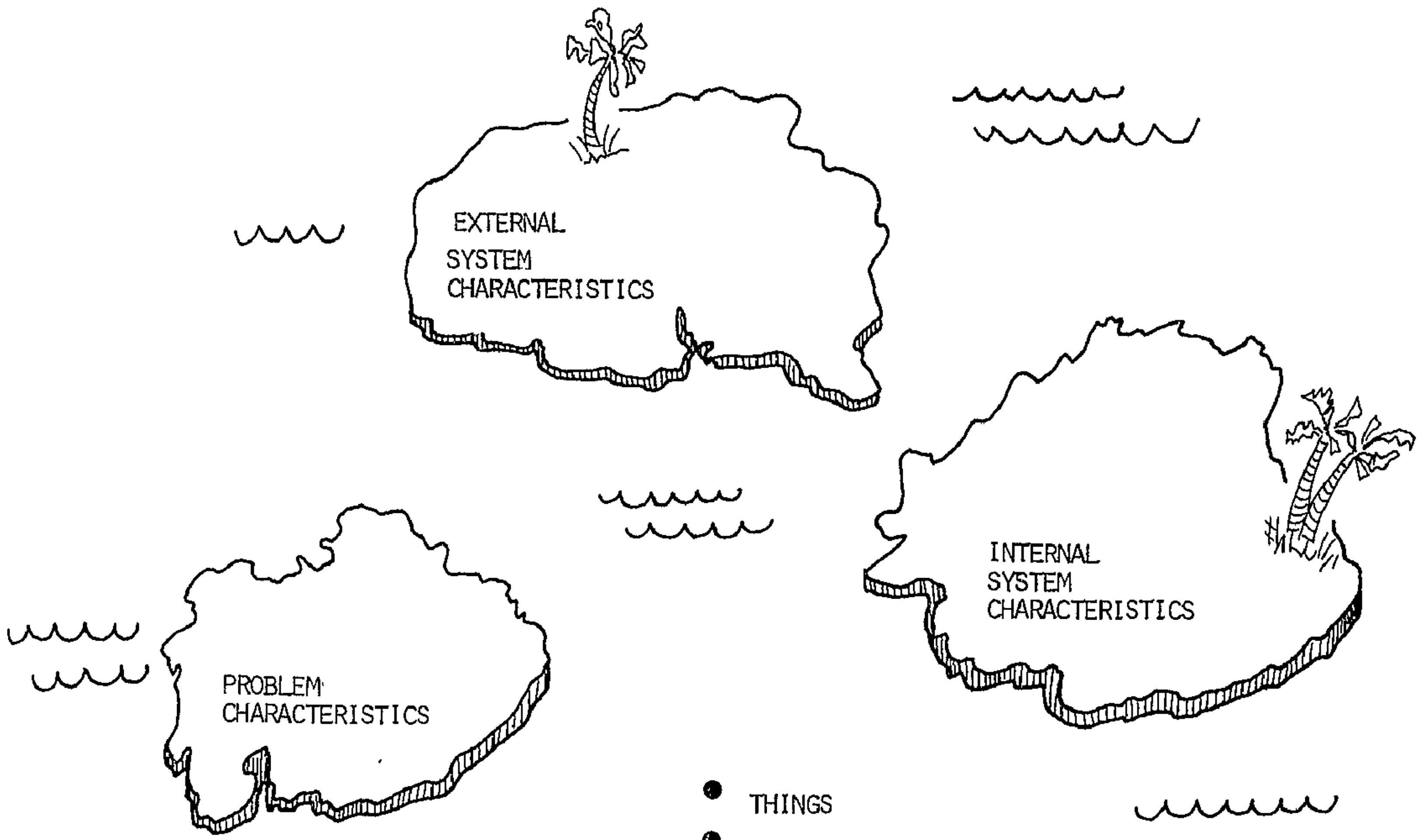


Figure 7

INFORMATION REQUIREMENTS


EXTERNAL SYSTEM CHARACTERISTICS
INTERNAL SYSTEM CHARACTERISTICS
PROBLEM CHARACTERISTICS

- THINGS
- INFORMATION
- DATA

Figure 8

TOP DOWN SYSTEM DEVELOPMENT

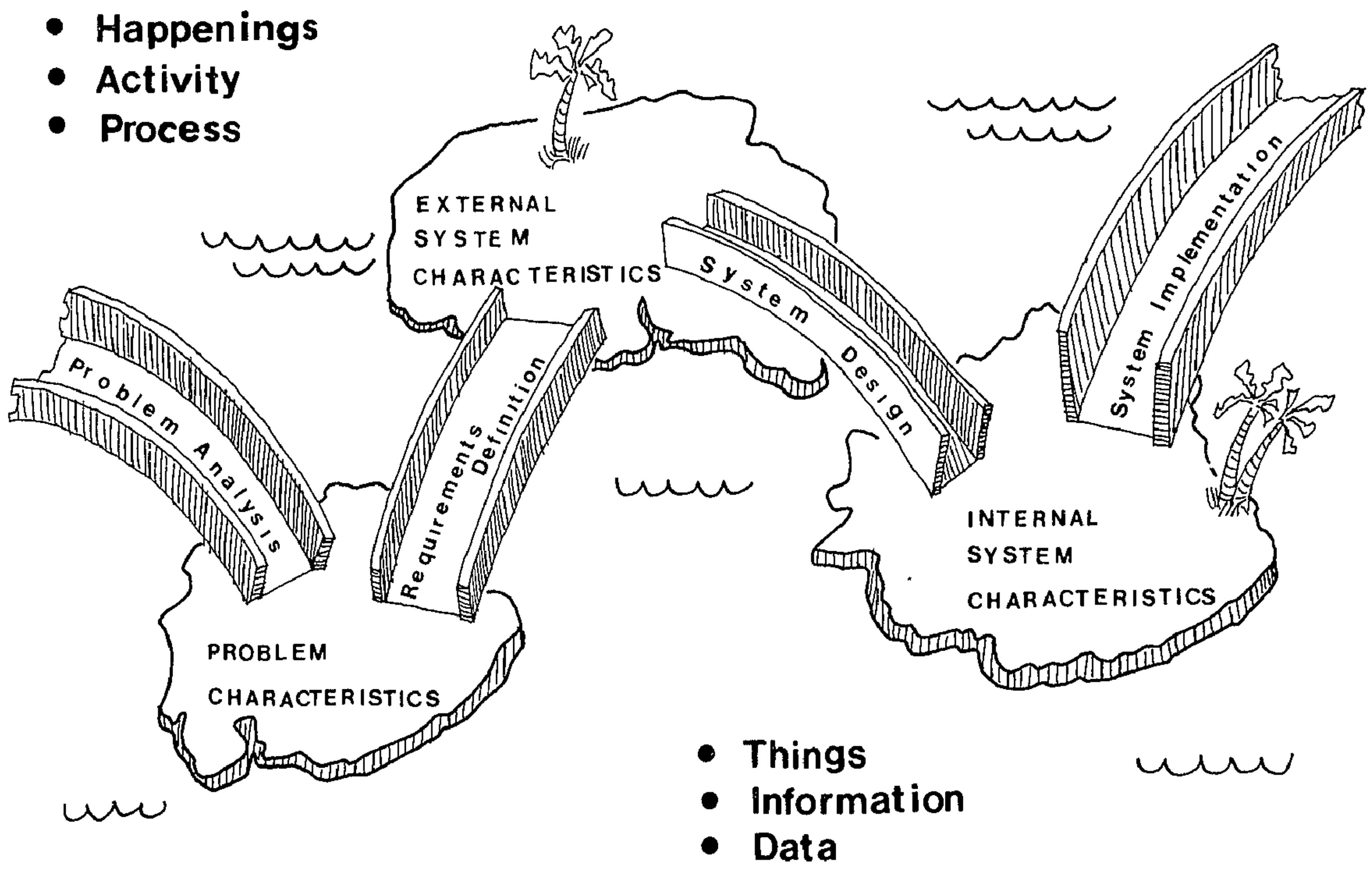
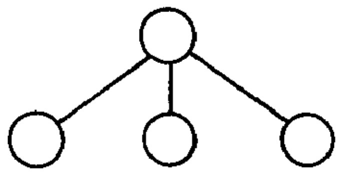


Figure 9

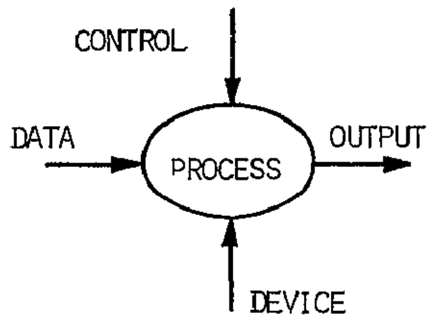
THE STRUCTURED APPROACH A NECESSARY DISCIPLINE

Use Basic Elements

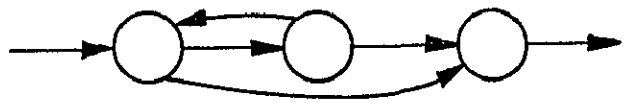
- HIERARCHAL GROUPING



- BASIC NODE



- NODAL RELATIONSHIPS



To Develop Structured:

- DOCUMENTATION
- PROBLEM ANALYSIS
- REQUIREMENTS ANALYSIS
- SYSTEM DESIGN
- SOFTWARE
- TESTING
- ORGANIZATION

Figure 10

THE STRUCTURED APPROACH-MAKES MANAGEMENT POSSIBLE

BASIC MANAGEMENT WORK

●	PLANNING	-	PREDETERMINE COURSE OF ACTION
●	ORGANIZING	-	ARRANGE AND RELATE WORK
●	LEADING	-	CAUSE PEOPLE TO TAKE ACTION
●	CONTROLLING	-	ASSESS AND REGULATE PERFORMANCE

NOW BECOMES POSSIBLE

- WORK BREAKDOWN STRUCTURE
- ESTIMATING
- WORK ASSIGNMENTS
- BUDGET / SCHEDULES / QUALITY
- REVIEWS
- REPORTING

Figure 11

Harvey McComb:

Next speaker is Harry White, from the National Bureau of Standards, to discuss Standards and Documentation.

Harry White, National Bureau of Standards:

I start off first of all conversing. I would like to take an exception to the previous speaker. And sincerely the point is that software is not the dully mess that we found it to be several years ago. But there are still many professional aspects which we have not been able to determine, standardize, or handle that we need to consider in order to adequately address essential software as a tangible resource.

The last couple of years we have been able to more accurately identify the cost of software, which is one of the most essential aspects of computer systems. We have found that software is the most expensive element in computer systems, software and data generation. Unfortunately, from a management standpoint, we concentrated in the early days on the aspect of the hardware. We can count the number of computers and the number of peripheral equipment and the number of tapes that are in the library. But this is not where the real problem is in management of computer systems.

As far as users are concerned, they care less about the hardware as long as they effectively get the job done. And looking at the investment made over the years, it has become clear that the cost of software and cost of maintenance have been grossly underestimated. Therefore, in the last 15 years we have been concentrating upon establishing standards for software development and maintenance. Essentially, that is one of the highlights of the presentation I want to give this afternoon. First of all, there is a little bit of background that I think you might be interested in. What is the Bureau of Standards doing in the computer area, and particularly what do our standards mean? Essentially, our charter for the work at the Bureau of Standards is very recent, recent in terms of 10 years, and it is based upon the legislation enacted back in 1965, referred to as the Brook Act, which placed with the Secretary of Commerce the responsibility for establishing standards for the effective uses of computer services within the federal government. To this end, we work very closely with industry on a voluntary basis, manufacturers, the American Standards Institute, and International Standards Organization on an international basis in arriving at the standards in computer area.

Furthermore, we have a number of federal test groups and public advisory committees that we work with on a day to day basis in arriving and identifying the standard parts in the Federal ADP communities.

The highlights of the presentation today are the role of documentation and standards as it pertains to the aspects of determining compliance with software, the capability of being able to share software and, furthermore, the very hairy question of how do we certify software and what does that mean.

The position of the National Bureau of Standards, and this is arrived at over many years of studies, is that the key element to assuring functional fidelity for computer based information systems is precise documentation. Documentation serves as the standard for testing a system against expected design and performance specifications.

The testing of computer programming language compilers for compliance is always performed using a standard (documentation) as the basis. All deviations are reported in terms of their departure from the standard.

Documentation standards and the capability to precisely describe systems requirements must be recognized, demanded, and practiced before computer technology truly becomes a profession rather than an art. Without documentation standards and practices, it is impossible to validate or certify the performance of a computer system.

Now let us talk about standardization in the computer field and how this relates to users. Essentially, standardization in the computer field is dependent upon user's demand for standards; we cannot assume, based upon our current market place practices, that the vendor, whether it is a hardware or software vendor, is going to provide you with product standards. Our experience currently in the market place is that corporate standards are the market place practice, and if you as users want standards, you are the one that has to demand them. IBM and Sperry are not going to give you voluntary standards. There is a very good marketplace reason why this is being practiced today since this type of situation is being handled more or less on a legal basis, and is a subject of antitrust.

Another problem that we experience essentially is that documentation standards have not become part of the professional standards. In the ADP (Automatic Data Processing) field, since it is essentially a new evolving technology, we have not progressed to the point that we have in other professions of recognizing the importance of standards as part of the professional practices.

And here I challenge you as professionals in your field to consider the aspects and the importance of standards, documentation practices as they pertain to the importance of your profession. And finally professional standards and practices must be user defined and implemented. It i. not the role of vendors to define the type of standards that you need; this is your problem and the solution rests with you.

The National Bureau of Standards has the responsibility for the development of standards to provide for the effective acquisition and use of computer systems within the federal government. This responsibility is derived from the provisions of Public Law 89-306 which was enacted in 1965. One of the highest priorities for standards is for documentation that will facilitate the specification, operation, maintenance, and sharing of computer software. Software costs now far exceed the costs for hardware. Unfortunately, we have not yet learned how to effectively account for and manage software as we have hardware. NBS has a major program that is focused on software management. We have developed and published five federal standards and guidelines that are currently being implemented by federal computer activities.[1] These standards and guidelines were developed based upon active participation and inputs from government, industry, and using ADP (Automatic Data Processing) organizations. A brief description of these standards is provided below:

Vocabulary for Information Processing, Federal Information Processing Standards Publication 11 (FIPS PUB 11). This publication provides an alphabetic listing of over 1200 terms and definitions used in information processing.

Guidelines for Describing Information Interchange Formats (FIPS PUB 20). Also

[1]Information concerning the availability of these Federal Information Processing Standards may be obtained from the Office of ADP Standards Management, National Bureau of Standards, Washington, D. C. 20234. (Phone 301-921-3157.)

American National Standard X10.1-1973. These publications provide documentation guidelines for identifying and describing the physical and logical characteristics of formatted information involved in automated interchange.

Flowchart Symbols and Their Usage in Information Processing (FIPS PUB 24). This publication provides standard flowchart symbols and specifies their use in the preparation of flowchart documentation used to describe computer systems and programs.

Software Summary for Describing Computer Programs and Automated Data Systems (FIPS PUB 30). This publication provides a standard software summary form (Standard Form 185) and instructions for describing computer programs and automated data systems for purposes of identification, reference, and dissemination. This standard is used for documenting software that is developed or acquired for federal government use. This standard is also used by the General Services Administration as a basis for registering and identifying software in the Federal Software Exchange Program.

Guidelines for Documentation of Computer Programs and Automated Data Systems (FIPS PUB 38). These guidelines provide a basis for determining the content and extent of documentation needed for effectively describing computer programs and automated data systems. Ten document types are provided for the various aspects of systems design, operation, and maintenance. These include:'

1. Functional Requirements Document. Provides a basis for the mutual understanding between users and designers of the initial definition of the software, including the requirements, operating environment, and development plan
2. Data Requirements Document. Provides, during the definition state of software development, a data description and technical information

about data collection requirements

3. System/Subsystem Specification. Specifies for analysts and programmers the requirements, operating environment, design characteristics, and program specifications for a system or subsystem

4. Program Specification. Specifies for programmers the requirements, operating environment, and design characteristics of a computer program

5. Data Base Specification. Specifies the identification, logical characteristics, and physical characteristics of a particular data base

6. Users Manual. Describes the functions performed by the software in non-ADP terminology such that the user organization can determine its applicability and when and how to use it. It should serve as a reference document for preparation of input data and interpretation of results

7. Operations Manual. Provides computer operations personnel with a description of the software and of the operational environment so that the software can be run

8. Program Maintenance Manual. Provides the maintenance programmer with information necessary to understand the programs, their operating environment, and their maintenance procedures

9. Test Plan. Provides a plan for the testing of software; detailed specifications, descriptions, and procedures for all tests; and test data reduction and evaluation criteria

10. Test Analysis Report. Documents the test analysis results and findings; presents the demonstrated capabilities and deficiencies for review; and provides a basis for preparing a statement of software readiness for implementation

In summary, there are several key observations concerning documentation and standards that should be noted by users and managers of computer systems and services:

1. Documentation is the key element for the eventual standardization, validation, and maintenance of computer software. Documentation standards and practices must be included as an integral part of systems management, development, and operation.
2. Currently, the computer profession does not adhere to or utilize accepted documentation procedures as do other professions. Standards have been developed and are now available. The responsibility for standards implementation now rests with individual systems managers and computer professionals.
3. Currently, users and managers must explicitly cite the use of standards when acquiring or developing computer systems. Unlike other industries, the computer marketplace is currently dependent upon the individual corporate practices of the major vendors. Unless the users and acquirers of computer services and systems demand standards, this will continue to be the marketplace environment which inhibits compatibility and sharing of computer hardware, software, and data across different product lines.

In conclusion, I hope I have been able to whet your appetite for some of these standards. I have provided you with information about where they can be acquired. As I mentioned, they are available from the National Bureau of Standards and we also have a complete set of all our standards that has been published to date, which is 39 standards included in a specialized three way binder that holds these. Thank you very much.

Harvey McComb:

Next is Bob Nickell, from Sandia Corporation, and his topic is Software Validation and Certification.

Bob Nickell, Sandia Corporation:

I guess I feel like we all have been here before or at some other similar place. It is more like reincarnation, we seem to resurrect this animal periodically every couple of years or so; we examine it to find out whether or not it has obtained the proper purity that we placed alongside the gods; and then we dispense with it and resurrect it again in two years.

The two topics that I am supposed to address today (very briefly by the way and I will save most of the session for the discussion between the audience and the panel members later on this afternoon) are two very different concepts. The idea of validation is a operational requirement on software whereas the concept of certification is a quasi-legal or even possibly legal requirement for software and quite possibly for people that use software substantially. And in the past few years while we have all been talking about the subject, now very quietly down at the grass roots level, a software vendor and software users are reaching a marketplace accommodation on just which of these concepts is important, what kind of certification seems to be in order, and so forth.

Although large software systems have been developed for a variety of engineering disciplines, the title for this symposium suggests that the emphasis should be upon the validation and certification exercises that have been undertaken relative to structural analysis. This discussion, therefore, will attempt to describe the terms validation and certification in their structural analysis context, point out some current efforts in both regards, considering both advantages and limitations, and speculate on the trends of the future.

Two of the components of the validation process - verification and qualification - have been defined by Griffin (ref. 1) and are shown in figures 1 and 2. Verification, which is primarily the responsibility of the software developer, involves the demonstration that the theories and models upon which the program is based are correctly coded, irrespective of their applicability to design. Examples might be a particular constitutive model (say, a dilatational-dependent plasticity theory) or a particular finite element description (say, an arbitrary doubly-curved deep shell element). Qualification, which is primarily the responsibility of the software user, involves the demonstration that the program capabilities will not be exceeded by combinations of elements, boundary conditions, initial conditions, material models, etc., that are typical of the user's design analysis requirements. This exercise should also determine that the logical flow paths and program configurations (core size, backing storage, dimensions, and other critical parameters) are consistent with analytical needs. The developer also maintains a set of qualification problems which serve two purposes: (1) to requalify the program after each set of updates or increment of capability and (2) to provide a mechanism for workshop training of prospective users.

An additional component of validation that has become prominent in recent years is calibration, (see fig. 3) the procedure by which simplified models of complex physical events (sometimes involving experimental correlations) are incorporated into the software, often through free parameters. An example might be the calibration of an uncertain mass distribution through a free vibration survey, followed by transient design analysis.

These three components of validation have been supplemented by occasional thoughts on user qualification (refs. 2 and 3), which deals with the education and training of analysts in such areas as numerical integration, solution algorithms, linear algebra, and the like.

Certification, on the other hand, is not a technical concept but, instead, implies legal standing for the software package (program certification) or for the user (professional certification). The interest of somebody with an ability to provide legal sanctions against uncertified software or users is also required. A somewhat weaker form of certification would accompany an ability to impose economic, or perhaps moral, sanctions. For example, most governmental agencies with large research and development budgets (such as those shown in fig. 4) would be able to exert considerable influence over software and users of software under contract to them although no formal attempts at certification by such agencies have come to my attention. Instead, validation has received major emphasis, as can be seen by examining the ERDA-sponsored (ORNL) High-Temperature Structural Design Methods Program and Validation of High-Temperature Design Methods and Criteria Program (figs. 5-9).

Other official bodies do have implied legal standing with regard to software, should they choose to become involved. Two of these bodies are listed in figure 10. Neither has chosen to venture into the treacherous waters of certification although the NRC position on benchmark calculations often wavers precariously between software qualification (for certain types of safety analysis) and software certification. The ASME Boiler and Pressure Vessel Code is not apt to be extended to treat certification either, with the possible exception of postprocessors (i.e., software that converts stress analysis output into a form that can be compared to Code allowables). A likely course of action would be to address postprocessing procedures but to avoid explicit consideration of the postprocessors themselves.

Professional societies have a vested interest in promoting certification of individuals, both through the legal formalism of professional registration and the quasi-legal accreditation of professional schools or institutions of higher learning. A recent development is the adoption of the concept of recertification through continuing education by the professional societies - a concept that might eventually lead to a form of software user certification.

A more likely candidate description for people certification was offered by Don Griffin at the Winter Annual Meeting at the American Society Mechanical Engineering of 1972. He viewed people certification from a viewpoint similar to the AMA view. At present we require registration which is our legal certification for determining qualification to design structures, buildings, and engineering parts. His view was that registration may be (he did not say it was) but it may be insufficient to protect the public safety. In other words, if it becomes widely accepted that the registration exam and the registration certification are no longer sufficient to guarantee the public safety (as it is not in the medical field) then we must have another level of certification. In the case of AMA, they specify the certification requirement and also require an oral exam and written examination in a submedical specialty.

Now it may be that in the software area such sublevel certification is a requirement because most registered engineers are assigned complex tasks these days and need to use complex algorithms, probably without sufficient understanding of the algorithm and its limitations, even though it is used to perform complicated stress analysis. If in fact that turns out to be the case, we can make an argument, a serious argument, for some type of certification of software users.

One last concept I would like to discuss, the Nuclear Regulatory Commission, does in fact have a certification or a quasi-certification document on the acceptability of computer programs for the design analysis of mechanical systems, and it put these programs in a precategory and I would like to just read to you three short sentences. Either of the programs must fall into these categories:

1. The computer program is a recognized program in the public domain and has had sufficient history of use of justified complexity covering the entire domain.
2. The computer program's solution to a series of test problems with accepted results has been demonstrated to be substantially identical to those obtained by similar independently written programs in the

public domain. In other words, you may wish to use a proprietary piece of software but you must demonstrate that it has the same general procedure, plus the same solution, as a program in a public domain.

3. The program solution to a series of test problems is substantially identical to those obtained by hand calculation or accepted experiments or analysis results published in the Technical Literature. Thus, you have a choice.

The situation, both today and in the future, could be summarized as follows: (a) the verification process is well understood since it involves a one-to-one correspondence between concepts of structural mechanics and the programmer's ability to replicate these concepts; (b) the qualification process is not well understood since the interaction between program logic, solution algorithms, and user-controlled factors (e.g., program configuration, space and time discretization, etc.) does not lend itself to formal academic presentation; those engineers with expertise in program qualification tend to acquire such skill piecemeal, over a period of years; and (c) the certification of software, in addition to its troublesome legal implications, does not appear to offer any relief to the natural tension between the owner of a structure, the designer/analyst, and the software developer. Thank you.

References

(1) Griffin, D. S.: The Verification and Acceptance of Computer Programs for Design Analysis. On General Purpose Finite Element Computer Programs, P. V. Marcal, ed., American Soc. Mech. Eng., c.1970, pp. 143-150.

(2) Nickell, R. E.: Qualification — People, as Well as Programs. Engineering Computer Software — Verification, Qualification, Certification, I. Berman, ed., American Soc. Mech. Eng., c.1971, pp. 51-59.

(3) Nickell, R. E.: User Education: The Finite Element Short Course. The Software User: Education and Qualification, H. Kraus, ed., American Soc. Mech. Eng., c.1972, pp. 9-15.

V E R I F I C A T I O N

VERIFICATION IS DEFINED AS THE DEMONSTRATION THAT A COMPUTER PROGRAM CORRECTLY SOLVES THE MODEL THAT WAS PROGRAMMED, INDEPENDENTLY OF WHETHER OR NOT THE MODEL IS A VALID REPRESENTATION OF ANY PARTICULAR SYSTEM.

Figure 1

Q U A L I F I C A T I O N

QUALIFICATION IS CONCERNED WITH THE USE OF COMPUTER PROGRAMS FOR SOLVING THE REAL PROBLEMS ENCOUNTERED IN DESIGN. GIVEN THAT A COMPUTER PROGRAM CORRECTLY SOLVES THE MODEL PROGRAMMED, DOES THE COMBINATION OF MATHEMATICAL MODEL, DISCRETIZATION, DESCRIPTION OF MATERIAL PROPERTIES, REPRESENTATION OF THE LOADING AND TEMPERATURE HISTORIES, AND BOUNDARY CONDITIONS, ALL CONSISTENT WITH THE PROGRAM LIMITATIONS, GIVE AN ACCEPTABLE SOLUTION TO THE PHYSICAL PROBLEM ?

Figure 2

COMPUTER PROGRAM VALIDATION

VERIFICATION	-	DEVELOPERS' RESPONSIBILITY
QUALIFICATION	-	COMBINED DEVELOPER AND USER RESPONSIBILITY
(CALIBRATION)		

Figure 3

ECONOMIC AUTHORITY

DEPARTMENT OF DEFENSE

ENERGY RESEARCH AND DEVELOPMENT ADMINISTRATION

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

Figure 4

Table 1. Typical Set of Verification Benchmark Problems

Problem No.	Description	Comments
1.1	Uniaxial specimen subjected to reversed cyclic loading into plastic range	Analysis should produce cyclic stress-strain curve. Good for studying load incrementing choices.
1.2	Uniaxial relaxation specimen under isothermal conditions	Analysis should reproduce simple calculations or actual relaxation test results. Good for studying time increment choices in creep analyses.
1.3	Uniaxial specimen subjected to stepped and reversed creep loading	Analysis should produce simple creep curves with hardening rules considered. Good for time-increment selection.
1.4	Thick-walled cylinder subjected to transient radial temperature gradient producing plastic behavior	Results can be checked analytically. Good for studying choice of temperature increments for plastic loading.
1.5	Biaxially loaded plate or cylindrical shell subjected to specialized nonproportional loadings into plastic range	Results can be checked analytically or, in the case of cylindrical shells subjected to pressure, axial, and torsional loads, by experimental results. Good for checking program treatment of nonproportional loading.

Figure 5

Table II. Typical Set of Qualification Benchmark Problems

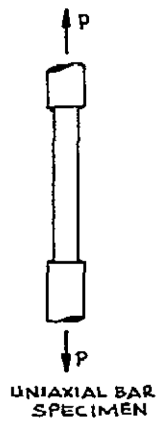
Problem No.	Description	Comments
2.1	Simply-supported beams subjected to various center loading histories producing elastic-plastic creep behavior	For plane 2D-solid and beam analyses. Test results being obtained under ORNL High-Temperature Structural Design Methods Program.
2.2	Simply-supported circular plates subjected to various center loading histories producing elastic-plastic creep behavior	For axisymmetric 2D-solid and flat plate analyses. Test results being obtained under ORNL High-Temperature Structural Design Methods Program.
2.3	Capped circular cylindrical shells subjected to varying internal pressure loading producing elastic-plastic creep behavior	For axisymmetric 2D-solid and shell analyses. Test results being obtained under ORNL High-Temperature Structural Design Methods Program.
2.4	Finite-width flat plate with circular hole subjected to cyclic axial load producing elastic-plastic creep behavior	For plane 2D-solid analyses. Results being obtained under ORNL High-Temperature Structural Design Methods Program.
2.5	Stiffened shear lag panel subjected to cyclic external loading producing elastic-plastic creep behavior	For plane 2D-solid analyses. Results being obtained under ORNL High-Temperature Structural Design Methods Program.

Figure 6

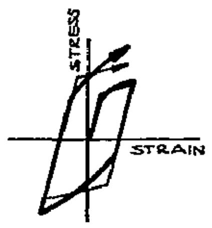
(Table II cont.)

Problem No.		
2.6	Pipe thermal ratchetting specimens (straight pipe subjected to internal pressure and to periodic thermal down-shocks in contained fluids)	For axisymmetric 2D-solid or shell analyses. Results being obtained under ORNL High-Temperature Structural Design Methods Program and by LMEC.
2.7	Piping elbows subjected to external loadings producing elastic-plastic creep behavior	For shell and special piping analyses. Test results being obtained under ORNL High-Temperature Structural Design Methods Program and Validation of High-Temperature Design Methods and Criteria Program.
2.8	Nozzle attachment thermal ratchetting specimen (conditions as above)	For shell analyses. Results being obtained under Validation of High-Temperature Design Methods and Criteria Program.
2.9	Nozzle-to-spherical shell attachments subjected to internal pressure and external loadings causing elastic-plastic creep behavior	For axisymmetric 2D-solid or shell analyses. Results being obtained under ORNL High-Temperature Structural Design Methods Program.
2.10	Intersecting cylindrical shells or piping tees subjected to external loadings producing elastic-plastic creep behavior	For shell analyses. Test results being obtained under ORNL High-Temperature Structural Design Methods Program and Validation of High-Temperature Design Methods

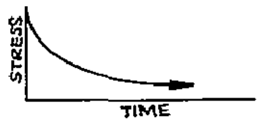
Figure 7



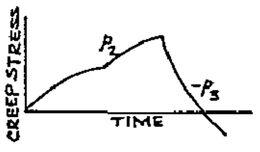
(1) Reversed Cyclic Loading Into Plastic Range

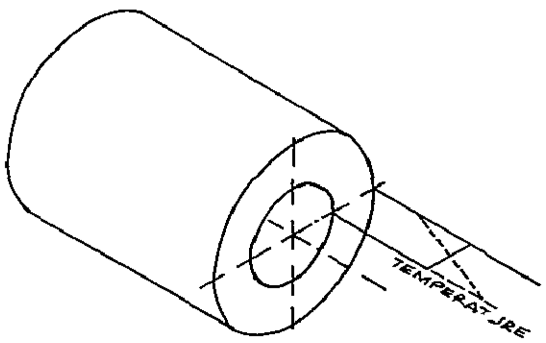


(2) Relaxation (Fixed Total Strain) Loading

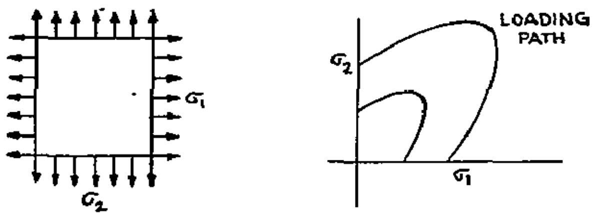


(3) Stepped and Reversed Creep Loading



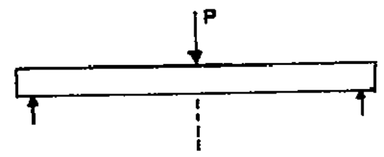


(4) Thick-Walled Cylinder Subjected To Radial Temperature Gradient Producing Plastic Behavior

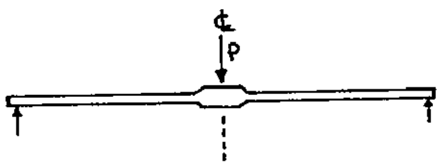


(5) Flat Plate Or Cylindrical Shell Subjected To Special Nonproportional Biaxial Loadings Producing Plastic Behavior

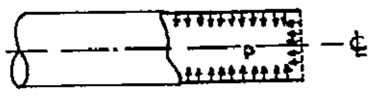
Figure 8. Simple Benchmark Problems for User Training and Basic Program Verification.



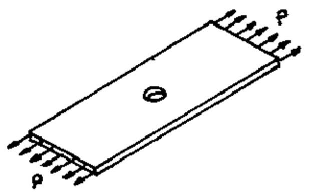
(1) Simply-Supported Beams



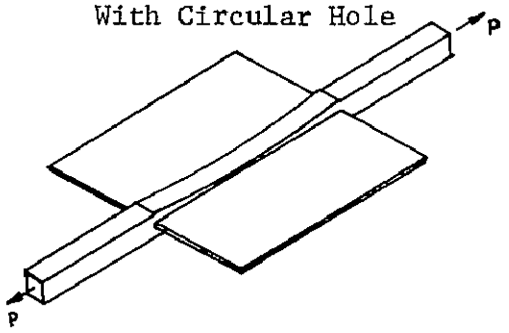
(2) Simply-Supported Flat Circular Plates



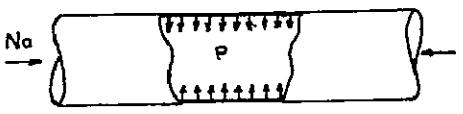
(3) Capped Circular-Cylindrical Shells



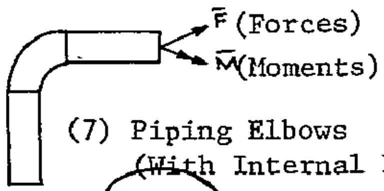
(4) Finite-Width Flat Plate With Circular Hole



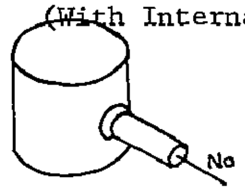
(5) Stiffened Shear Lag Panel



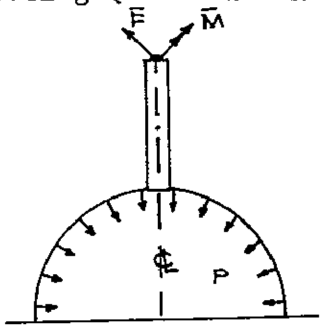
(6) Pipe Thermal Ratchetting



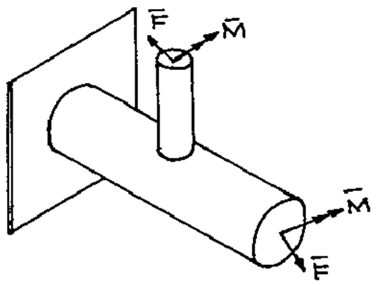
(7) Piping Elbows (With Internal Pressure)



(8) Nozzle Attachment Thermal Ratchetting (With Internal Pressure)



(9) Nozzle-To-Sphere Attachments



(10) Intersecting Cylindrical Shells (With Internal Pressure)

Figure 9. Advanced Benchmark Problems Qualification of Program and User.

LEGAL AUTHORITY

ASME BOILER AND PRESSURE VESSEL CODE

U. S. NUCLEAR REGULATORY COMMISSION

Figure 10

Harvey McComb:

Next is Jim Johnson, from AFFDL, Dissemination, Portability, and Maintenance.

Jim Johnson, AFFDL:

Yesterday I managed to get soaking wet; it rained in Washington, yet I had a raincoat in the Motel where I was staying. Sure enough I was unable to maintain all in ,good health; consequently my throat is a little raw and I may lose my voice before 5:00 p.m. Also, the Department of Defense is suffering from an economic recession. Consequently we are a little unable to get you visual aids for this presentation. While that was going on we had a reorganization in the laboratory, and I got a physical relocation. Perhaps I should avoid any such remarks and focus primarily on Dissemination, Portability, and Maintenance of Large Engineering Software Systems.

My discussion does not address the many subtle problems of our panel theme. Nor will I attempt to define a workable solution to every aspect of our panel theme. Instead, I shall focus primarily on Dissemination, Portability, and Maintenance of Large Engineering Software Systems.

The complexities and problems in connection with the development, distribution, and maintenance of large engineering software systems are widespread. Within the past decade, most of us have been bombarded with large computer programs such as ASKA, COSMOS, ASTRAL, SAS, STAIR, STARDYNE, FORMAT, MAGIC, ASOP, NASTRAN, FLEXSTAB, etc. for the solution of our engineering problems. Upon receipt of one of the large general purpose structural analysis computing systems, we attempt to load the program and execute a check case. At this point, we usually face the harsh realities of life. In most instances, it is not possible to successfully load the program on the first try. Other abortions result from tape/card processing problems, inadequate definition of minimum machine resources required, etc. Yet we continue to obtain other people's software in spite of the many frustrations to be expected because of our need to utilize the latest technological developments of other engineering organizations. Thus, the most important aspect of dissemination, portability, and maintenance is technology transfer. To date, I believe that too much emphasis is placed on the costs of doing this job rather than the objective of technology transfer. Since the large general purpose programs are developed to serve the scientific and engineering community, it is only necessary to maintain a proper balance between the needs of the practitioners and the costs of computer program maintenance and distribution. Enough for costs. Let us examine software design and development procedures to facilitate maintenance and those procedures and organizations to facilitate dissemination in the light of our objective to effect good technology transfer.

For large engineering software systems, good technology transfer is accomplished via:

1. Documentation
2. Users' Meetings
3. Education and Application Seminars

Inasmuch as documentation and user education will be discussed in detail by other panelists, I shall focus primarily on those procedures and practices that must be considered as an integral part of the aforementioned areas to effect technology transfer.

When a computer program cannot be understood by a potential user, the would-be recipient may write his own program, which results in an unnecessary duplication of effort, or the problem to which the program would have been applied will remain unsolved. First, and foremost then, the dissemination and portability of large computer programs require "good communication". Although good criteria for the dissemination of large computer programs have been around for more than a decade, for completeness I shall restate a lis of requirements that insure good communication when disseminating large general purpose computer programs:

1. Complete statement of the program, its problem formulation, and complete equipment description
2. Functional flow charts
3. Fully documented source program listings
4. A complete user's manual containing a clear description of the inputs and outputs, operating instructions, and symbol definitions
5. Adequate test cases
6. All assembly (machine) language shall be easily identifiable and emphasis will be placed on localizing assembly language coding
7. Master code media

Most of us are quite familiar with all of the above items. We at WPAFB have contractually required the satisfaction of this list since about 1962 or 1963. In spite of our familiarity, let us amplify a few of the items.

Functional flow charts: A complete description of each function must be included. This definition includes the title, purpose and use, system inputs, expected output and results, relationship to other functions, and summary of function operation.

User manual: A complete description of the operating instructions and philosophy. The operating instructions must include step-by-step procedures required to initiate (or implement) the program, maintain computer program operation, terminate and restart the program (normal and unscheduled termination), software system modification/generation ("octal-in" and creation of a new version of the master code media), symbolic updating procedures, and minimum machine resources required.

Assembly (machine) language: Programs containing machine language code are strongly dependent upon the characteristics of the computer itself, and programs containing assembly language code are generally not portable to any machine other than the one for which they were written.

Master code media: The usual mode for dissemination of large programs is by magnetic tape. The master tape description is a must, and this description must define the format, density, blocking, mode, contents, and processing procedures. The normal (or recommended) procedure for dissemination of master code is to copy the source program and test cases onto a magnetic tape, then compile the program from the copy, and exercise this compiled program on the test cases from the copy. This procedure insures that no malfunction occurred during copying and, since this output is also sent to the recipient, it aids him in putting the program up on his system.

Now let us turn our attention to maintenance procedures. Maintenance, at most, implies the requirement to "stay current". Technology is dynamic and maintenance of large engineering software systems necessitates a "constant process" of modification/

updating to provide for new capability and error fixes. Is this new information automatically distributed? Who makes the necessary modifications? When? Why? Is there feedback from users? These are the questions posed by the engineering community, however, program maintenance involves these simple procedures:

Program code revision/modification: Anyone can perform this task. It requires some knowledge of the program to be changed. Changes (enhancements and errors) must be documented (symptoms and conditions), must work and not adversely effect the rest of the program, must be verified and validated, and must be disseminated.

Reference program version: Must be clearly identified and used for maintenance work.

Release of new version: 24 to 48 months during which time a significant number (possibly 50) of code modifications have been made.

Up-to-date user identity: Addresses and identities of users (old and new) are required to maintain currency.

Dissemination: Changes, modifications, error fixes, and the like have no real significance unless they are available and distributed in a timely manner.

The foregoing remarks outlined software design and development procedures to facilitate dissemination, portability, and maintenance. Let us briefly turn our attention to procedures and organizations to facilitate dissemination.

Large engineering software systems are disseminated in one of two ways. Firstly, the program is distributed to a customer (requestor) via master code media. This procedure requires customer (local) implementation and maintenance. Secondly, the program is available to the customer via some network system. This procedure requires no customer (local) implementation and maintenance. Programs available via network systems only require dial-up procedures. For this reason, the following information pertains to procedures of organizations for direct distribution of large programs via master code media.

1. Programs are stored in a central depository.
2. Information defining the availability of these programs is published via computer program abstracts, newsletters, and technical reports.
3. The costs for obtaining a program from one of the depositories vary from zero, to exchange programs, to annual leases (or purchases), up to $40,000.00.
4. Several program distribution centers have defined, or are in the process of defining, criteria for program inclusion in the depositories.
5. Checkout procedures for requested programs consist of compilation and execution of test cases, at most.
6. Available documentation is supplied. Implementation instructions vary from no information to "customizing".
7. Very limited maintenance, if any, is attempted.

The Air Force Flight Dynamics Laboratory at Wright-Patterson Air Force Base has distributed engineering software since the early sixties. The Aerospace Structures Information and Analysis Center (ASIAC) was established in 1974 to collect and disseminate information on aerospace structures. Products and services available from ASIAC include:

1. Technical Inquiry Services on Aerospace Structures Problems
2. Bibliography Service
3. Monographs and Reference Books

4. Current Awareness Publications.

5. Computer Program Distribution.

Approximately 40 structural computer programs are currently available from ASIAC. These programs are distributed, if not generally available at other computer program dissemination centers. In some instances, ASIAC has permission to distribute proprietary codes to government agencies only. To date, the AFFDL and ASIAC have distributed programs at no charge. Our future plans are to recover the handling costs.

AFFDL and ASIAC perform a significant amount of maintenance and documentation for the computer codes in our depository. This procedure is the notable exception to the generally widespread practices noted above. Thank you.

Harvey McComb:

Next speaker is Professor Steven Fenves, from Carnegie-Mellon University, representing the Educator's viewpoint.

Steven Fenves, Carnegie-Mellon University:

I was not quite sure what aspects of education to cover so I decided to put in some personal thoughts. There are three sets of considerations that one has to keep in mind in talking about educational aspects of standardization: certification, maintenance, and dissemination of large engineering software systems.

First is the distribution of the population to be served by educational programs. The vast majority of engineers are program users; their only contact with computer programs is through preparation of input and interpretation of output for programs written by someone else. A much smaller number of engineers are ad hoc program developers; they write, modify, or maintain programs for their own use, or at most, for use by small groups with which they are intimately connected. A miniscule number of engineers are software system developers.

Second, essentially all large engineering software systems are proprietary in nature. Many of the aspects of software systems addressed by this panel constitute the principal competitive advantage of such systems.

Third, there is a serious generation gap between engineering software system developers on one hand and software engineers on the other. Most of the application system developers, myself included, are graduates of the old ad hoc school of programming and are not fully aware of recent revolutionary developments in software engineering such as structured programming, formal modularization, and program assembly testing and verification techniques.

I must call upon a past experience in that category. Exactly 12 years ago we made a presentation to IBM of the program we were developing at that time, STRESS, and one person asked: "What procedure did you use to resolve conflicts at the interfaces?" To me this was an unexpected question, and not knowing how to respond, I said: "Well, let's see: there is this group of five of us working on the project, my office is in the middle, and I only have four chairs, so we borrow a chair to sit down and resolve the conflicts of the interfaces." But that was 12 years ago. I think a lot of people have learned something since then.

In an attempt to put large-scale software development into some focus, the writer has proposed a model of the development process (refs. 1 and 2). An extended version of the model used in a recent study (ref. 3) is shown in figure 1.

It can be seen that education is shown as a professional activity, reflecting the above mentioned need to service the largest number of people involved (i.e., the end users). Figures 2, 3, and 4 taken from the same survey, in fact, classify the available educational activities on the basis of application areas covered. Such educational efforts, in general, accomplish their major objective of making end users aware of, or even proficient in the use of, existing programs without directly exposing the proprietary, competitive aspects of these programs.

Although comparable survey data are not available, it is fair to state that most computer-based courses in formal engineering curricula are geared to the second group of engineers (that is, future ad hoc programmers (either graduate students who will have to write ad hoc programs to satisfy their engineering thesis requirements or students who aspire to use their programming skills to secure a competitive advantage in employment)). Needless to say, courses aimed at this group are generally taught by people who are themselves ad hoc programmers.

Education specifically geared to the small number of software system developers is practically nonexistent in the engineering disciplines. From an educational productivity standpoint, it is unrealistic to expect a major effort in this rather narrow area.

In summary, while we all may want more and better educational activities, the first two groups of engineers are relatively well served by existing educational activities. In order to provide the proper education for the third group, two things must be done:

1. An organizational structure must be developed whereby the technical activities shown in figure 1 are open for scrutiny and evaluation without destroying the valid proprietary, competitive aspects of the software systems

2. The educational horizons of software system developers must be considerably broadened so that they can apply the relevant aspects of software engineering to the development of engineering software.

Thank you.

References

1. Fenves, Steven J.: Current Attempts at Software Coordination in Civil Engineering. Papers Prepared for the Special Workshop on Engineering Software Coordination, Robert L. Schiffman, compiler, Rep. No. 72-4, Univ. of Colorado, Apr. 1972, pp. 53-65.

2. Fenves, S. J.: Software Products for Engineering Applications. Automated Procedures Engineering Consultants Journal, vol. 6, no. 3, May 1971, pp. 9-12.

3. Civil Engineering Programming Applications, Inc.: A Proposal for a National Institute for Computers in Engineering, Oct. 1975.

Software Activity Model

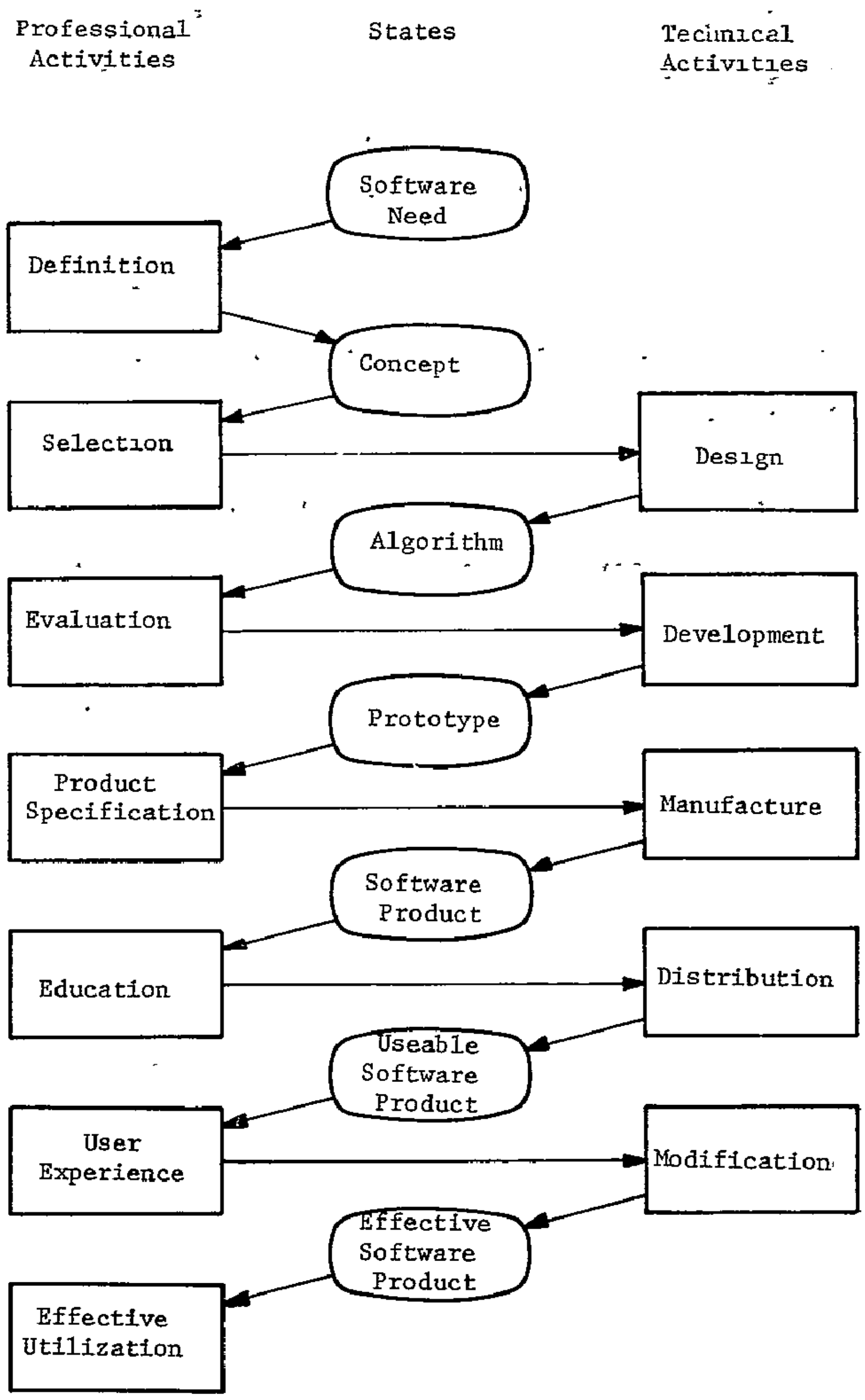


Figure 1

User Group Based Continuing Education

Courses Offered to Practicing Engineers by Computer User Groups

25-Total Number of User Groups Identified (Base)

27-Total Number of Courses Identified (Base)

KEY ☐ Courses

■ User Groups

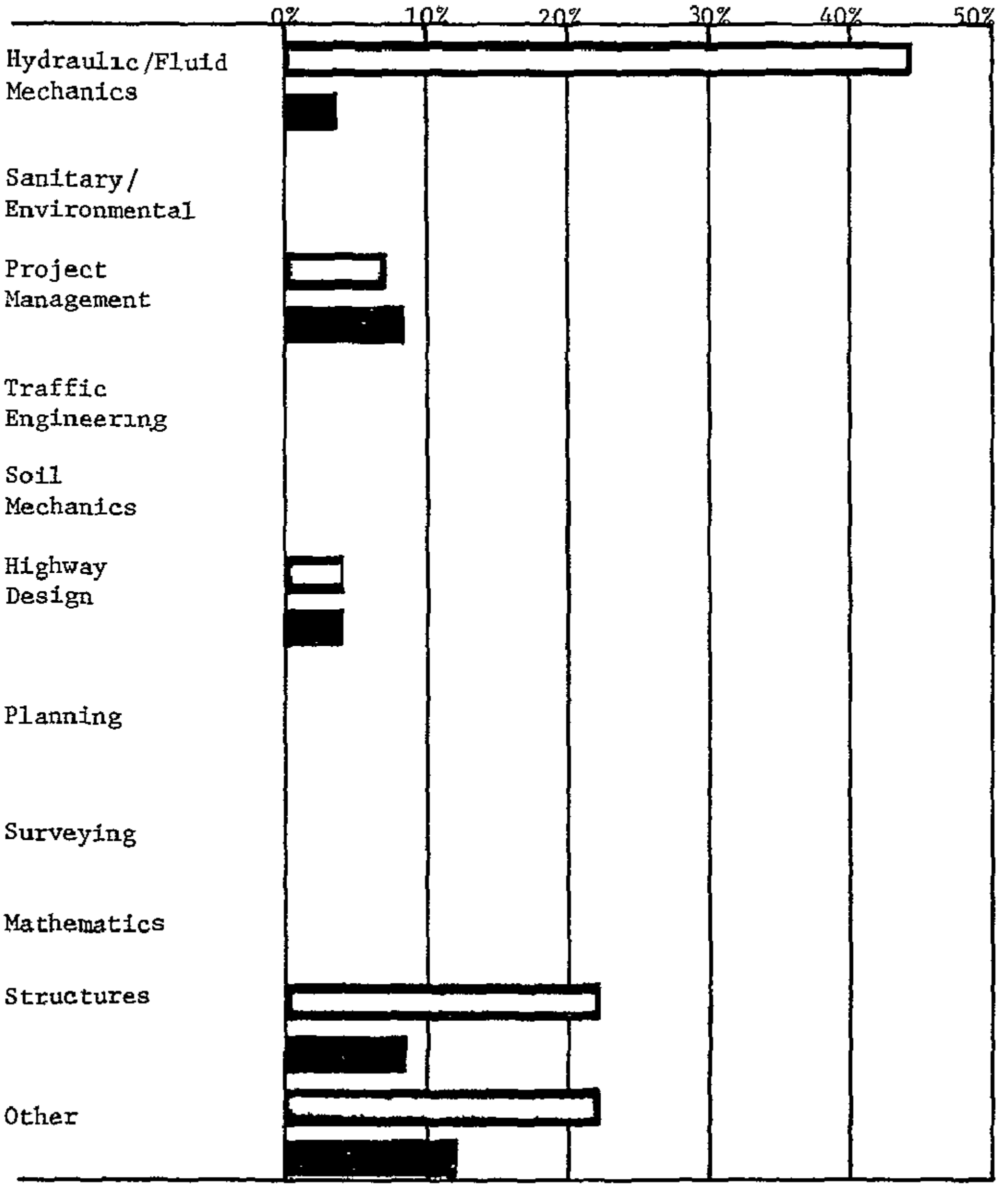


Figure 2

University Based Continuing Education

Courses Offered to Practicing Engineers by Civil Engineering Depts.

111-Total Number of Departments Responding (Base)
164-Total Number of Courses Offered (Base)

KEY ☐ Courses
■ Departments

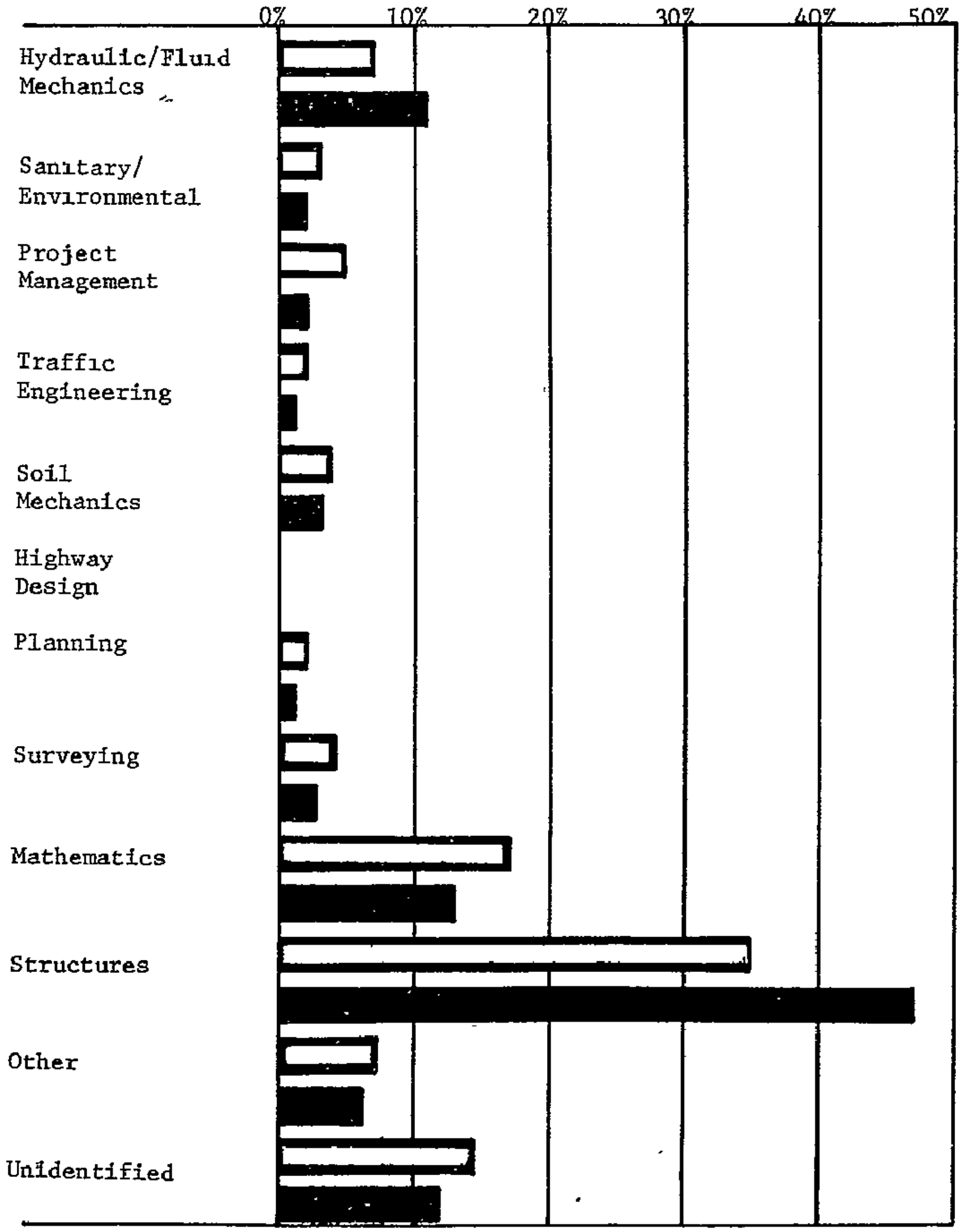


Figure 3