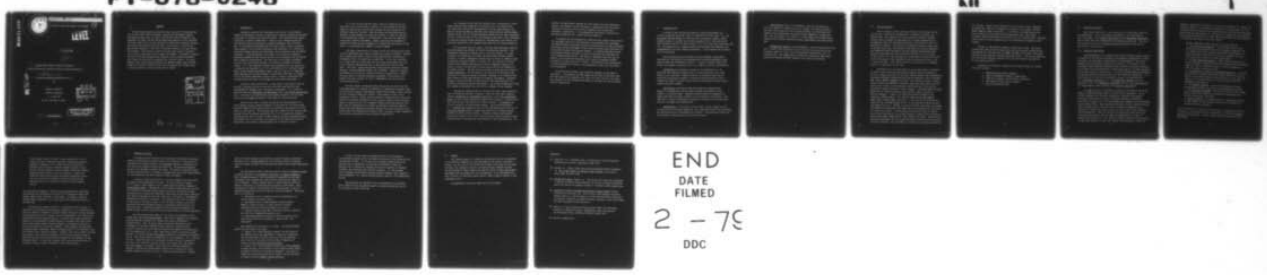
AD-A061 468 PACIFICA TECHNOLOGY DEL MAR CA F/G 9/2
THE INTERAGENCY SOFTWARE EVALUATION GROUP: A CRITICAL STRUCTURA--ETC(U)
AUG 78 R E NICKELL N00014-77-C-0575
UNCLASSIFIED PT-U78-0246

1 OF 1
AD
A061468



END
DATE
FILMED
2 - 79
DDC

PACIFICA TECHNOLOGY

P.O. Box 148 • Del Mar, California 92014 • Tel: (714) 453-2530

LEVEL II

PT-U78-0246

20p.

The Interagency Software Evaluation Group: A Critical Structural Mechanics Software Evaluation Concept

Contract N00014-77-C-0575

by

Robert E. Nickell

Pacifica Technology

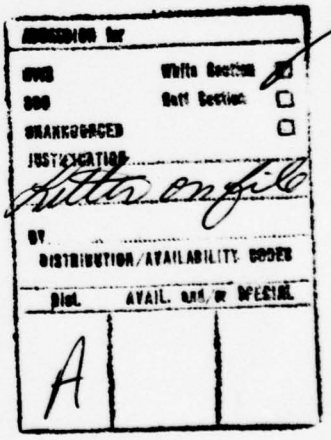
P. O. Box 148

Del Mar, California 92014

14 August 14, 1978

ABSTRACT

The Interagency Software Evaluation Group, composed of representatives from various United States government research and development agencies, including the Army, Navy, Air Force, the National Science Foundation, the Department of Energy, and the Nuclear Regulatory Commission, have initiated an effort aimed at the critical evaluation of applications computer software. With active participation from the three armed services, a list of such applications software in the field of structural mechanics has been derived and screened. The screening criteria were based upon actual or potential use in a multi-laboratory or multi-agency environment. In addition, software selected for further evaluation was required to meet minimum requirements with respect to availability, documentation, and verification. The codes deemed to most typify these conditions were NASTRAN, ADINA, STAGS, and SAP. A family of shell-of-revolution codes was also selected for further study. Critical evaluation criteria are discussed in detail.



I. Introduction

The market structure that encompasses the variety of relationships between the developers and users of applications computer software has received a good deal of scrutiny within the past decade. This marketplace was characterized by a university (developer) - industry (user) dialogue during the evolutionary stages of applications software development some 10-15 years ago. As the pecuniary advantages were realized, however, private industrial firms emerged as the principal developers, many supported either directly or indirectly through research and development (R & D) contracts with the federal government. In the background, small groups of specialists at federally-sponsored laboratories were playing the dual role of developer and user within their respective organizations. The total cost to the federal government for both direct and indirect support is difficult to estimate for several reasons: (1) labor separation between code development and code application is generally not accounted for in R & D organizations; (2) overhead charges often include software development support; and (3) the cost of computing cannot equitably be allocated between an organization that owns its computers (or has the federal government purchase them as a capital expenditure) and an organization that deals in the commercial data center environment.

In spite of these difficulties, a lower bound in the range of tens of millions of dollars per year can be easily derived, based entirely upon extrapolation of manpower costs from known circumstances. It is useful to note that these are development and maintenance costs, not software applications costs. The latter are significantly more computer-intensive, but the labor costs are also staggering because of the number of analysts involved.

Since the cost that is usually associated with the development of a fairly general purpose software package is on the order of a million dollars, the total number of such packages that are available to the analyst is on the order of a few hundred. This estimate coincides roughly with the survey, under National Science Foundation sponsorship [1], of civil engineering software, considering that only a fraction of that software represents documented coding. Such numbers serve to illustrate the magnitude of the information-retrieval dimensions of the problem, notwithstanding the cost/benefit aspects.

It is felt by many that the needs of the user community are not being met by the current market structure, not only because of the sheer numbers of choices that the user must examine, but also because of the technical complexity involved. Both the theory and the coding practice employed in these packages are specialty subjects, in which the user has a disadvantage with respect to the developer. Since the user generally relies upon "sampling" procedures - namely, exercising the software on problems such that personal judgement will indicate the correctness of the result - a coordinated user response is usually needed.

The formal coordination of a group of users has resulted from a need to share experiences with respect to particular pieces of software. Periodically, these groups meet to discuss recent applications, elimination of programming errors, and additions to capability. In this way, the whole is, in many respects, greater than the sum of the parts because each user has an increased understanding that is enriched by the views of other users. Such users' groups have evolved around several structural software packages, including NASTRAN, STRUDL, SAP, and ADINA. Perhaps the greatest shortcoming of this form of information sharing is parochialism - namely, the group tends to reinforce favorable perceptions to the exclusion of adverse criticism, through its lack of independence from the developers and pseudo-developers.

The most minimal coordinated effort that would not suffer from this shortcoming would be a sharing of information on applications software, both on the disclosing agency's operational software and on other, related software that the disclosing agency has studied. Some progress on this tack is documented in [2-4]. Perhaps the most ambitious attempt would be the actual sharing of executable software on a host computer, either at the disclosing agency's computing site or at a commonly-funded centralized computing site. Both types of implementation are being considered by individual agencies - namely, NESS (Navy Engineering Software System) [5] and the computing center established at the Department of Energy's (DOE), Lawrence Livermore Laboratory for controlled thermonuclear fusion research computing [6].

As a response to this need for software users' coordination, a multi-agency group has been attempting to formulate a plan for common action. This group has consisted of representatives from the armed services, the National Science Foundation, DOE (the former Energy Research and Development Administration), the U.S. Nuclear Regulatory Commission, and other interested parties. Since the needs of the user community range widely, from information on available software through acquisition, documentation, verification, and production application, the suggested plans have been of varying complexity.

At the present time this group, called the ISEG (Interagency Software Evaluation Group) has defined for itself an intermediate approach. This intermediate approach is necessary for at least two reasons: (1) good intentions are insufficient to render the applications software problem tractable; a careful, manageable, limited scope of effort is needed in order to determine reasonable success or failure; (2) the logistics of shared interest (and funding) among several virtually independent agencies are tenuous, at best, and the scope of effort should reflect the immediacy of the common interest. The plan that has evolved is based upon the concept of software evaluation. For the present, the study will be confined to the evaluation of structural mechanics software, although it is recognized that other technical fields may eventually be added to the program. Structural mechanics software provides an ideal forum for the study because of its diversity and advanced stage of development. The variety of marketing strategies adopted by developers has also led to a plethora of user problems.

The ISEG has operated under the justification that a more significant step in information sharing is required - that this step toward improved software reliability and cost/benefit ratio must rely on some standardized format for the critical evaluation of applications software. In accord with this justification, the ISEG has set itself a purpose - to formalize the critical evaluation of applications software - and two near-term goals: (1) to develop the criteria for the critical evaluation process and (2) to critically evaluate several commonly-used applications software packages according to these uniform criteria. Structural analysis applications software was chosen as the initial area of concern primarily because of the advanced state of developer-user relationships in this specialty area. For

example, few applications programs for heat transfer and fluid mechanical analysis are developed by one individual or organization with the express intention of serving a wide variety of potential users outside the developers' organization. It is expected that this type of activity will eventually become the rule, rather than the exception.

A procedure has been proposed for converting this concept into practice: this procedure consists of the selection of the software to be evaluated, the identification of desirable evaluator characteristics for both individuals and groups, and the development of the criteria by which these individuals or groups would evaluate the selected software. It should be pointed out that volunteered evaluation exercises in the past have been well-intended, but have produced shallow results. The ISEG is a departure from this type of effort, in that qualified contractors will be paid to evaluate software according to a prescribed set of criteria. The possibility of synergism due to shared information, resources, and techniques is also a positive factor to be considered.

In the following sections, the evaluation procedure is described in some detail. The procedure has been adopted by the ISEG to the extent that the three armed service agencies, U.S. Army, U.S. Navy, and U.S. Air Force, have agreed to use the following description as the basis for the evaluation contract negotiations.

II. Screening Tests

In order to meet the near-term goals of criteria development and critical evaluation, the total effort is subdivided into four tasks: (i) the development and application of screening tests to determine whether a given software package is suitable for critical evaluation; (ii) the selection of ideal evaluator profiles for these studies; (iii) the evaluations themselves; and (iv) the preparation of a summary document that provides both highlights and recommendations for future software procurement. In this section the screening tests are discussed.

The screening tests were divided into three mandatory requirements (availability, verifiability, and documentation) and two non-mandatory items (qualification and configuration control). A piece of software which fails to pass the mandatory screening tests is not subject to further consideration.

Availability was defined to include such considerations as: no inordinate time should be needed to make the source and object files operational on the contractor's computing system; no laborious conversion of specialized input/output routines or high-level plotting routines; and no incompatible loader or special overlay structure that requires active system software specialist support.

Verification is defined to mean those numerical examples whose purpose is to demonstrate that the theory alleged to be present in the software is correctly coded. Candidate software should be accompanied by sample input data and output solutions for the purpose of verification. Any code that is unable to reproduce the developer's solutions at the contractor's site will be dropped from consideration.

Documentation is to include, at the minimum, a users' manual for describing the input and output adequately, and a theoretical manual that describes the underlying physics and mathematical treatment. The theoretical manual may be replaced by reference material from the literature.

Qualification, which is non-mandatory, describes the process by which the software is demonstrated to be reliable for production analysis, and takes into account the size of actual problems, any limitations with regard to file manipulation, and other features demanded in a production mode. For this reason, it is useful if a large number of different organizations with different production analysis requirements have exercised the code.

Configuration control, also non-mandatory, concerns the ability of the user to specify the configuration of the code (in-core versus out-of-core resources, number of files, size of the unlabeled common block, etc.) at execution time. This capability will enable the evaluator to make proper comparisons, between different software, with regard to efficiency.

III. Screening Results

The five general criteria that were listed in the previous section - availability, verification, documentation, configuration control, and qualification - were applied to a large variety of structural software packages that are subject to widespread use, or where widespread use is contemplated, within the laboratory/contractor infrastructure of the U.S. Department of Defense. The results from documented, formal surveys, such as the NESS report [5]; from personal discussions with service laboratory and information center personnel; and from telephone conversations with cognizant individuals were used to establish the initial list of codes. Of this initial list, the vast majority of codes were in active use by only one service branch, or one laboratory within a particular service, or by a single individual. Such limited interest was perceived as an immediate reason for elimination from the list.

Continuing on to the second phase of the screening were a variety of structural codes, which were then grouped together into five categories. The first category consisted of the large, general purpose codes representing tremendous development investment and extensive capability, such as NASTRAN, MARC, ANSYS, NEPSAP, STARDYNE, and others. Even the most cursory of examinations revealed that NASTRAN was the most widely used code within the three services and, since all five screening criteria were met, this code was selected for further evaluation in this category. A second category included those codes of limited capability which primarily address the area of nonlinear dynamics - a small, but technology-intensive segment of the structural software market. This category comprised the codes NONSAP, ADINA, AGGIE, PETROS, HONDO, and others. It was felt by the ISEG steering committee that the ADINA package met all of the screening criteria and best represented this class. However, in terms of further evaluation, the related code NONSAP and the emerging code AGGIE would be factored into the evaluation in an unspecific manner. A third category consisted of codes of limited generality aimed primarily at nonlinear analysis of general shell structures, and included STAGS, ADINA (again), PLANS, NEPSAP, and others. Because of its continued development, and its present and potential application to aerospace and naval structures, the code STAGS was chosen by the steering committee for

this category. Another group included shell-of-revolution codes, regardless of whether the loading was axisymmetric or general; this category included BOSOR, DYNAPLAS, KSHELL, and others. It was felt by the steering committee that virtually all of the major shell-of-revolution codes were readily available, and that evaluation of the entire list was a feasible task. Therefore, several codes of similar capability will be simultaneously evaluated in this category.

Finally, a rather general category of codes were listed. This group included such codes as SAP and SAAS, which have had enormously widespread (but diminishing) use, and codes such as TEXGAP and ORACLE, which have limited but intensive use at scattered locations around the country. The steering committee felt that the SAP code best typified this class while meeting the majority of the screening criteria.

Therefore, the structural software that has been selected for further evaluation includes:

1. NASTRAN (general purpose category)
2. ADINA (with NONSAP and AGGIE as comparisons)
3. STAGS (for aerospace and naval shell structures)
4. SAP (general linear elastic category)
5. Shell-of-Revolution Codes.

IV. Evaluation Procedure

The critical evaluation is to be carried out in three steps, which can run concurrently. First, the program architecture is to be described by two of the steps - the functional description and the programming description. Both of these steps will depend heavily on adequate documentation. The third step consists of a series of advanced evaluation exercises, which will be described in a later section.

IV.A. Functional Description

Good documentation for an applications analysis computer program requires a discussion of the physical principles, the mathematical embodiments of these principles, the algorithmic representations of the mathematics, and the programming practices employed to effect the algorithms. In contrast to this requirement, most program documentation refers the user to selected literature, in the form of archived manuscripts, institutional reports, and other documents with varying stages of retrievability. The self-contained program documentation may include only the briefest discussion of the theoretical foundations. As a significant part of the critical evaluation process, the description of: (i) the physical principles; (ii) the mathematical statements of these principles; (iii) the mathematical algorithms, and (iv) the coding practice are lumped together under the heading of functional description.

An example serves to illustrate the intent of this type of description. Suppose a program developer claims an elastic-plastic capability. If the critical evaluator has sufficient documentation available, the physical principles (conservation of linear and angular momentum or equilibrium, conservation of energy, etc.) and the mathematical statements of the principles may be stated clearly. Special attention should be given to reduced continuum theories (beams, plates, and shells) and to kinematic assumptions (e.g., separation of elastic and plastic strains or strain rates in the presence of finite deformation). Then, the critical evaluator must examine the actual coding practice together with the remaining documentation, in order to determine such features as: (1) whether a tangent stiffness, initial stress, or some other approach is taken; (2) whether the convergence criteria are

adequate, considering the particular iterative method chosen; (3) whether the plasticity theory is an incremental, deformation, or other theory; (4) how accurately the rate equations are integrated within the load step (mean normal, equilibrium load correction); (5) the flow rules allowed; (6) the hardening rules allowed; and many others. After these descriptions have been completed, the algorithms and coding practice will emerge.

The most important of these functions to be described are:

1. The discretization approach - finite element, with its weak satisfaction of equilibrium and traction boundary conditions, and its strong satisfaction of energy conservation and kinematics; or finite difference, which differs in that the kinematic relations also possess weak solutions; or some other method.
2. The time integration approach - whether modal superposition or direct integration, whether the particular approach has unconditional stability, built-in stability controls, artificial damping, its order of accuracy, and other factors.
3. The approach for solution of simultaneous equations - (including procedures for treatment of nonlinear terms) whether by iterative, direct, or semi-direct methods; whether pivoting is used; what the measures of ill-conditioning are; storage limitations and file manipulation required; etc.
4. The kinematic approach - the strain-displacement relations incorporated; kinematic constraints and transformations allowed;
5. The constitutive approach - the stress-strain relations allowed; whether strain-rate effects are included; any limitations with respect to anisotropy; and others.
6. Special Features - an example might be a simultaneous solution of heat transfer or fluid mechanical fields, or the ability to interface with such solutions.

These functions are not meant to be all-inclusive. Instead, the critical evaluator has the responsibility for determining the functions alleged or implied by the developer, in order to focus upon the four parts of the description.

IV.B. Programming Description

Irrespective of the structural modeling basis of a software package - finite element, finite difference, exact solution, or some combination - the design of the modern packages have evolved into a characteristic form. The most coarse description of this characteristic form consists of a pre-processing module, an analysis module, and a post-processing module. These segments can be simple or sophisticated and, in the early years of software development, the operations of the current individual modules were intermingled. As the true, personnel-intensive costs of structural computing have become more well recognized, however, the worth of pre- and post-processors as instruments of economy has become evident. It is reasonable, in fact, to estimate that the software development costs associated with pre- and post-processors for production analysis packages far exceed the costs of analysis module development.

The pre-processor may include any or all of the following features:

1. Reading and printing of input data, with possible format conversion, depending upon the freedom or rigidity of the input format;
2. Automatic mesh generation, with possible interpolants and smoothing algorithms designed to provide some control over mesh condition number (vertex angle acuity) and aspect ratio (in matter of fact, the combination of mesh aspect ratio and deformation gradients are needed in order to properly allocate mesh spacing; the deformation gradients are, of course, unknown at the pre-processing stage);
3. Creation of data bases for use in the analysis module, such as material properties or element data subsets;
4. Creation of special file structure required by the input data, such as restarting, plotting, geometric data base, material data base, coefficient assembly, and/or equation-solving files; files for communication between two or more distinct analysis modules would be included in this item;
5. Initialization of constants, flags, logical variables, and arrays;

6. Pre-analysis plotting, aimed at verifying the mesh geometry, boundary conditions, material properties (constitutive representations, such as creep laws or elastic-plastic stress-strain laws, can be plotted in order to expose any regions of error or omission), or other input feature.

Many modern pre-processors are interactive, interrogative, and are self-contained with respect to documentation. For example, the user may be creating a portion of the input data base interactively, with the pre-processor interrogating the user intermittently and providing a display of the necessary documentation at decision points in the preparatory process.

The analysis module or modules can be generically described as a series of libraries with connective coding. These libraries include, but are not limited to, the following:

1. The geometric library, which consists of the element strain-displacement library (both linear and nonlinear representations), the kinematic boundary condition library, coordinate transformations, and generalized kinematic constraint conditions; additional features, such as a coordinate update option that enables the program to handle both full Lagrangian and updated Lagrangian reference systems, might be offered; it should be pointed out that an essential feature of any general purpose structural code is the ability to incorporate generalized kinematic constraints between different portions of the element library (e.g., beam, plate, or shell elements enforcing a plane section constraint on a series of continuum elements);
2. The constitutive library, which consists of the various material property models available to the user, ranging from linear, isotropic elasticity to nonlinear, anisotropic elastic-plastic-creep behavior; if the constitutive and geometric libraries are kept properly independent, each constitutive capability will be available for any geometric configuration; one of the underlying principles of general purpose coding is to maintain this independence;
3. The procedures library, which contains the range of analysis options open to the user; typical examples are linear static analysis,

linear dynamic analysis (either by mode superposition or by a variety of direct integration operators), eigenvalue extraction (whether for dynamics or for linearized buckling analysis), nonlinear static analysis, nonlinear dynamic analysis, linear elastic fracture mechanics, fluid-structure interaction analysis, thermostructural analysis, and limit analysis; it should be noted that solution and accuracy-preserving strategies, such as in-core or out-of-core solution, iteration and equilibrium load correction, are considered to be part of the procedures library.

The input/output strategies, including those associated with restarting and plotting, are considered to be a part of the connective coding that relates these various libraries to each other. In addition, blank or labeled common, subroutine calls, and logical tests should be treated in the same vein.

Post-processing modules remain in a relatively primitive state, in comparison to pre-processing and analysis capabilities. As a convenience to the user, almost all commercial and many research structural software packages provide plotting files of stresses, strains, displacements, etc. that can be accessed by the local plotting devices. In recent years, there has been an upsurge in the development of post-processors to evaluate the results from one or more analyses against design allowables, providing the user with a direct confirmation of structural integrity. Some attention has been paid to printed output formatting, aimed at producing results in a form that can be put directly into a reporting document. However, very little progress has been made toward a self-contained evaluation of the analysis results, aimed at providing the user with a quantitative assessment of the "goodness" of the structural model. In some cases, however, this type of post-processing is available and merits special attention in the code evaluation.

V. Advanced Evaluation

The highly-skilled evaluator with an extensive background in numerical methods and applications software analysis will discover, in the process of describing the software functions and programming features, characteristics that need to be explored further and quantified. The advanced evaluation exercises are designed to address this need. The extent to which this phase of the evaluation effort is pursued is dependant upon the actual funding levels for each contractor and their respective rates of progress.

In spite of these uncertainties, a series of examples will be offered to point out the methodology to be used. For the pre-processor module (if it exists), these examples will be lumped under the general category of discretization checks. Although the use of automatic mesh generation has led to an easing of the burden of data preparation and an elimination of many associated errors, modern mesh generators should have an additional capability - they should be designed so that the "condition" of the mesh is evaluated and automatically altered, if necessary. The condition of the mesh is related to the acuity of the vertex angles of the individual elements, which then determines the possible energy states and convergence characteristics of a particular mesh. Pathological examples involving reentrant geometry and graded boundary mesh are available to test these features of the mesh generator.

For the post-processor module (if it exists), the examples will be lumped under the heading energy checks. Such checks, together with carefully selected benchmark problems, will expose errors due to deformation incompatibility. Also, an energy check can provide information on the convergence of nonlinear problems (stress states not satisfying flow criteria in plasticity, out-of-balance forces due to geometric nonlinearities or creep deformation, etc.). Also, these energy checks can provide a global measure of the effort expended by the mesh to deform in accordance with applied forces. These energy checks are broken up into two categories: (1) internal/external - where the total (or incremental) internal energy is compared to the total (or incremental) external energy; for elements that lose energy (but often appear to converge to exact solutions with small numbers of elements), selected two-element models should serve to expose such incompatible deformation behavior; and (2) internal energy hierarchy - through

selective volume integration (centroidal, two-point Gaussian, three-point Gaussian, etc.), the mesh effort can be determined, based on the energy partition between constant-straining modes and higher-order element deformation modes.

For the analysis module, there are several classes of examples designed to address questions of convergence, efficiency, and general capability. After "pseudo-convergence" due to deformation incompatibility has been eliminated, those programs that remain should be tested with respect to real convergence. Similar element libraries should converge similarly; however, some errors in formulation may be exposed at this level. Also, convergence of transient and nonlinear solution algorithms, as well as eigenvalue/eigenvector extraction routines, should be examined. Particular topics of concern are:

1. Convergence rates of elements should be determined to be correct (bounds are known a priori),
2. Convergence rates of Newton-Raphson, modified Newton-Raphson, Picard iteration, or other nonlinear solution algorithm should be tested and deemed to be correct;
3. Transient solution algorithms should be tested for stability and artificial propagation properties; and
4. Eigenvalue/eigenvector extraction routines should be evaluated for multiplicity, separability, deterioration, and convergence.

Some comparison of efficiency is in order. The most meaningful comparison would appear to be dual:

1. Compare, for a given benchmark problem, the efficiency of two programs having the same number of degrees of freedom and the same bandwidth; the point accuracy of the two solutions may or may not be the same, but this type of comparison is aimed at testing equation-solving efficiency;
2. Compare, for a given benchmark problem and a given required solution accuracy, the efficiency of two programs without regard to numbers of degrees of freedom and bandwidth. This type of comparison (having eliminated faulty element libraries earlier) is aimed at testing element library efficiency.

In order for any study of program efficiency to be meaningful, the individual programs must be evaluated with respect to any limitations on configuration control. In this way, one or another program need not be penalized due to thoughtless user-defined configuration, unless the developer has so limited the configuration that this penalty is intended. Therefore, the following items should be examined: (1) core storage versus extended core storage or backing storage parameters; (2) equation-solver parameters; (3) equation-solving options, and (4) file manipulation options.

The milestones for evaluators will be determined on an individual basis, especially for any general capability advanced evaluation, due to the intricacy of some of the tasks.

VI. Summary

The summary document is intended to provide the ISEG and the remainder of the technical community with the highlights of the critical evaluation process, plus the views of the various participants with respect to the scope of effort, the evaluation criteria, and the particular software selected. The implications of these studies with regard to future software and hardware development and procurement will also be addressed. It is anticipated that these criteria, or some modification thereof, might be made an adjunct to the procurement cycle.

Recommendations for future study are also anticipated.

References

[1] Schelling, D.R., "CEPA/NSF Study", Presentation at the Sixth National Conference on Electronic Computation, ASCE, 1974.

[2] Perrone, N., "Project STORE (STructures-Oriented Retrievable Exchange)", in: Use of the Computer in Pressure Vessel Analysis, ed. by Harry Kraus, ASME, New York, 1969.

[3] Argonne Code Center, Argonne, IL. The Argonne Code Center was established in 1960 by the Atomic Energy Commission to serve as a central information agency and library for computer programs of relevance to AEC program areas.

[4] Aerospace Structures Information and Analysis Center (ASIAC), Wright-Patterson Air Force Base, OH. The Air Force Flight Dynamics Laboratory sponsors ASIAC as a central agency to collect and disseminate information on aerospace structures, including structural software, to Air Force-funded and other government funded contractors.

[5] Matula, P., "Navy Engineering Software System (NESS) and Preliminary Selection of Computer Programs", TM-184-77-01, Naval Ship Research and Development Center, Bethesda, Maryland, October 1976.

[6] Personal Communication.