RESTRUCTURING THE ROTOR ANALYSIS PROGRAM C-60

P.G. Phelan and F.J. Tarzanin, Jr.

Boeing Vertol Company

Philadelphia, Pennsylvania

Abstract

The continuing evolution of the rotary wing industry demands increasing analytical capabilities. To keep up with this demand, software must be structured to accommodate change. The approach discussed in this paper for meeting this demand is to "restructure" an existing analysis. The motivational factors, basic principles, application techniques, and practical lessons from experience with this restructuring effort are reviewed.

Introduction

As the rotary wing industry matures it is getting increasingly difficult to extract the next significant technological advance. Improved understanding of the physical phenomena of rotary wing aircraft requires more complete analytical representations. Advances in computer technology are allowing larger more sophisticated analyses than were previously practical. As a result, the demand for complex analysis capability is growing rapidly, with increasing emphasis on interdisciplinary analysis. The changing and growing demands of rotary wing analysis necessitate that software be structured to quickly and efficiently accommodate change if it is to take advantage of continuing developments in this dynamic environment.

Software may be designed with structure emphasizing maintainability and modifiability. Software structured toward this goal provides reduced modification and maintenance costs, reduced time delay for adding new capabilities and improved reliability through a reduction in the number of undetected errors. This may be accomplished through judicious partitioning of software into functional modules, provision of well-defined paths of data flow, and adherence to a control hierarchy.

The proposed approach to obtaining a well structured rotor loads program is to "restructure" an existing analysis. Restructuring can be a time-saving and cost-saving alternative to developing new structured software. It consists essentially of reorganizing the code of an existing analysis to fit a structured design, while maintaining the theoretical basis for the analysis.

A Boeing Vertol rotor analysis program (C-60) is currently being restructured. Large complex blocks of multi-functional code are being broken down and reorganized into succinct functional modules. Related pieces of code, previously scattered throughout the program, are being gathered to form functional modules. Standardized formats for variable definitions, input data, output data, and documentation are being implemented. Even though the restructuring process is not yet complete, code performing a given task is now easier to find, understand, isolate, and modify. Variables, input data, and output data are also now easier to identify, understand, and modify as needed. The overall result is that new capabilities may be implemented in less time, at lower cost, and with improved reliability.

Background

- Motivation for More Capable Analyses

Physical understanding is expanding rapidly in areas related to analytical modeling of nonuniform downwash, rotor/fuselage coupling, vibration, noise prediction, rotor airloading, and composite material behavior, among others. There is a corresponding demand for improved analytical capability to reflect these advances More sophisticated designs, such as JVX and LHX, and expanded flight envelopes push many analyses beyond their present bounds of application, into regions where simplifying assumptions such as small angles, linearity, and low coupling breakdown.

Compounding these demands for expanded analytical capabilities is the pressing need for more accurate analytical predictions to facilitate finely tuned multi-variable design benefit trade-off studies. Many of the straight-forward one or two dimensional design problems have been solved. The largest potential rotorcraft improvements require complex trade-offs involving different technologies having a consistent level of complexity. As incremental isolated technology design benefits become smaller, the need for more complete, complex inter-disciplinary models increases. This view is supported by Kerr, Potthast, and Anderson[1]. Eventually, the requirements of more demanding interdisciplinary tradeoffs will increase to the point where specialized analyses in isolated disciplines will become inadequate and possibly even misleading.

Presented at the Second Decennial Specialist's Meeting on Rotorcraft Dynamics, AHS/NASA Ames, Moffett Field, CA, November 7, 1984.

Advances in computer technology and software development add fuel to the movement for more capable analyses. These advances provide expanded resources and/or reduced cost in terms of CPU time, memory usage, and I/O (input/output) capacity and sophistication. Approaches which were previously beyond practical resource limitations are now viable current or near-future options. Current computer systems now allow program structure to be independent of memory constraints, in contrast to the memory overlay structure restrictions of the past. Increasing sophistication of computer tools (such as shown in Table 1) also motivate more capable analyses.

- Practical Limitations of Current Programs

The development of more capable analyses is easier said than done. There is a history of difficulties with the development and upgrading of complex, multi-disciplinary analyses, and the prospect of developing even more sophisticated programs with the requirement for continuous updating, projects an image of long development time, high costs and questionable results.

The primary factor contributing to the difficulty of analysis development today is the tendency of large multi-user multi-analyst programs to evolve and grow in complexity beyond the comprehension of any single user/developer. It is as if such analyses follow a specialized law of entropy, tending toward ever increasing disorder until reaching "saturation of comprehensibility". Program maintenance and modification become increasingly difficult as clarity and understanding are gradually replaced by obscurity and misunderstanding. Bergland[2] presents two observations which summarize this behavior of large programs as "The Law of Continuing Change" and "The Law of Unstructuredness" as shown in Figure 1.

Eventually, at least one too many irreversible or untraceable revisions is made. Correlation and reliability falter suddenly. Previous analytical predictions can no longer be reproduced. Things that "worked" now mysteriously fail and the analysis "dies" suddenly. If a reliable backup version exists the analysis may enjoy a temporary reprieve but eventually it is likely to follow the path to extinction. Therefore, the program with chronic "saturation of comprehensibility" will eventually reach the point at which no further cost-effective growth is possible because revisions can no longer be fully understood or debugged.

The tendency toward disorder causes serious problems long before the analysis actually reaches complete "saturation of comprehensibility", that is, the point at which revisions can no longer be made. Complexity grows with each expansion of capability, as change is added to change without an overall plan or global structure. The analysis evolves gradually via the work of a variety of programmers, analysts, and engineers, with a corresponding variety of individual styles and preferences, (see Table 2). Complexity and apparent disorder result as the natural subtle accumulation of the effects of melding individual styles. These trends are compounded by a lack of adequate, accurate, current documentation and the growing innate complexity of the analysis due to expanding requirements. Excessive complexity reveals itself in highly unpredictable and often excessive person-hour costs for program maintenance and modification, and time delays for new capabilities.

As analytical capability is expanded, additions are made to program input and output. Additions for expansions of capability are often made quickly and without thorough coordination. Additions to input and output are often made more expediently than analytical revisions, and are sometimes left in "temporary version" form. The result is input and output which are not clearly defined, are possibly redundant, are not well organized, and are prone to error. This situation results in the expense of extra user-hours for preparation of program input and interpretation (sometimes deciphering) of program output, and an increased probability of undetected input error and/or output misinterpretation.

Other symptoms of saturation of comprehensibility are less obvious, but have the same root cause. For example, poor correlation may be an indication of undetected errors within the analysis, undetected misuse (such as input error), or undetected misinterpretation of analysis output. Compromised reliability is another warning sign of excessive complexity. An analysis which behaves very well sometimes and very badly other times is providing a warning. Growth of analytical complexity is often accompanied by increased dependence upon the specialized experience of experts associated with specific analyses (described by Kerr, Potthast, and Anderson[1] as "Sam's Program Syndrome"). This development implies poor or nonexistent documentation and very complex code, so that others are unable to understand the analysis.

- A Solution: Structured Program Design

The solution to this problem is the development of analysis programs which will start out and remain clear and understandable throughout a long life-cycle of maintenance and modification. This may be achieved by using "Structured Program Design" techniques to design software for maximum maintainability and modifiability.

Structured program design is a formal methodology for software design which was developed in an attempt to deal with the rapid expansion of software associated costs which began in the 1970's. Bergland[2] provides some historical perspective in "A Guided Tour of Program Design Methodologies" in which the author outlines general trends in software development from "Cottage Industry Programming" of the 1950's through "Heavy Industry Programming" of the 1960's to the birth of "Structured Programming" in the 1970's. At one point, over one percent of the GNP (gross national product) was being spent on software[2]. This stimulated early attempts at formulation of design criteria and programming techniques.

Today, a great deal of effort is being expended in attempts to formalize and standardize software design procedures to yield more maintainable, modifiable, and user-friendly programs. Structured program design is being applied in the rotorcraft field, including a government development program named 2GCHAS (Second Generation Comprehensive Helicopter Analysis System). Documentation has become a large part of most program development efforts. Efforts are being made to standardize, streamline a ! nearly automate the generation of both code and documentation. Some examples of this are specification of FORTRAN coding standards[3], development of a "generic" architecture[4], development of programs which generate diagrams directly from code[5], and the proposal of formal program design procedures such as data flow design or programming calculus[2].

An Approach to Structured Program Design

- Development Strategy

Structured program design is a general term referring to an application-dependent design procedure. It may be defined as "design for the best solution". The key to this approach is a well chosen definition of "best"; one that is well matched to the specific application. The procedure begins with the selection of a general goal, followed by a trade-off of benefits to prioritize different design criteria. A variety of terms have been defined to serve as design criteria. Terminology varies, as illustrated by the list of terms provided as tables 3 and 4, but similar concepts are defines in references 1, 2, 4, and 6. Design criteria used in the present study are efficiency, generality, maintainability, modifiability, reliability, & utility as defined in table 5.

The general goal (or measure of goodness) chosen for the software design discussed here is minimum total lifecycle cost. This lifecycle includes development, checkout, release, operation/maintenance/modification, and maturity, as shown in Figure 2. Yourdon and Constantine[6] describe the ideal program as "cheap to develop, cheap to operate, cheap to maintain, and cheap to modify". The relative cost and importance of the different phases of a program's lifecycle vary from application to application. The resulting prioritization of design criteria should vary accordingly. For example, the strategy for development of a payroll program might differ dramatically from that for a technical analysis in a volatile field because of different prioritizations for different aspects of the program lifecycle (i.e., efficiency for operating costs versus modifiability for modification costs). As another example, execution time and reliability might be most important for a real-time simulation. In each case, the goal of minimization of total lifecycle cost is reflected in the application-specific prioritization of criteria.

For our design, maintainability and modifiability were chosen as the most important design criteria. This prioritization results from heavy weighting applied to person-hour requirements for expansion of analytical capability and very-long-term accumulation of maintenance costs. In addition, coincident improvements in utility (i.e.: user-friendliness) and computational efficiency are anticipated from restructuring of input and output functions and elimination of redundant and repetitious calculations. However, improvements in these attributes are to be achieved only at no expense to maintainability and modifiability.

Two options exist for development of well structured analyses. These are (1) build a new analysis from "scratch", or (2) restructure an existing production analysis. Restructuring is reorganization of an already debugged, validated, correlated, "mature" analysis. It is the imposition of structure on an existing analysis. This approach utilizes current analysis theory, including derivation of equations, and solution method, and keeps current correlation intact. It utilizes information from current documentation in the new documentation. Checkout or validation consists of comparing results of the current poorly structured analysis with the new restructured analysis and implicitly takes advantage of all prior validation and correlation efforts. While restructuring does not, in general, include provision of any new capability it may be coincident with provision of new capability. An additional benefit is that the program remains operational and useful throughout the restructuring process, thereby providing immediate gains. In contrast, development of a new structured analysis from scratch begins with approach development and derivation of equations and thus may include new capability. However, starting from scratch does not utilize results of prior correlation, documentation, etc., and the program does not become useful until the long validation/correlation effort is complete.

Restructuring is the preferred approach if a "mature" analysis is available which is minimally organized and has been validated and correlated. The most difficult, costly, and time-consuming tasks to perform before a large program becomes useful are validation and correlation. Validation is assuring that the analysis program computes what it is supposed to compute (as defined by the equations and method of solution). Correlation is comparison of the analysis results with the "real-world". Validation and correlation require substantial effort for a large sophistocated analysis to exercise multiple-option combinations for multiple configurations. Utilization of prior effort is possible by direct comparison of restructured modular input and output with the same quantities from the poorly structured analysis. The trade-off is the effort required to identify these intermediate values in the poorly structured analysis versus the effort saved in validation and correlation. This potential savings is one of the most significant benefits of the restructuring approach.

Utilization of information in existing documentation provides another potential reduction of effort for the restructuring approach. Since restructuring utilizes the approach and derivation of the current analysis, the documentation pertaining to theoretical development is still valid and may be utilized in generation of new documentation. In addition, use of an existing method allows the development team to concentrate on the program structure. Thus, restructuring is favored if the current analysis approach is relatively well documented.

Another major benefit of restructuring is potential implementation of improvements in the near-term. Incorporation of at least partially restructured modules into the original program prior to completion of restructuring is possible. Small changes are simpler once a restructured module goes into production. Errors in the existing analysis may be uncovered and corrected, and new capabilities may be implemented in the restructured modules prior to completion of restructuring the whole program. Any benefits which may be incorporated into the new module are thus potential near-term benefits as well.

Determination of the suitability of a currently available analysis for restructuring requires an examination of its structure. (All programs have structure, though some have very bad structure in terms of maintainability and modifiability). The current analysis structure must be compared to a desired analysis structure, which of course, requires at least a first-cut design of a "good" structure. The goal of the comparison is a mapping of functions, and the connections between functions, from the poor structure to a good structure. The mapping is not likely to provide a one-to-one correspondence, but will provide an indication of the effort needed and trade-offs required for restructuring to be successful.

A functional mapping exercise indicates that C-60 is a good "target" analysis for restructuring. A first-cut design of "good" structure and a first-cut mapping of components from poorly structured C-60 to "well" structured modules has been performed. The first-cut design of "good" structure with clearly defined functions is shown in Figure 3. The mapping of corresponding functions in the current C-60 is illustrated by Figure 4 as a structure chart with functions distributed throughout the analysis. While the structure of current C-60 is clearly in need of reorganization, the mapping exercise indicated that validation, correlation, and near-term improvement benefits of restructuring should exceed the cost of efforts to identify, isolate, remove, replace, and reconnect pieces of analysis. In addition, the existence of relatively complete and well-written documentation makes program C-60 a good candidate for restructuring.

- The Restructuring Process

The first task in the restructuring process is development of a plan. A general outline of the plan for restructuring program C-60 is provided as Table 6. The initial tasks define desired attributes for overall program structure, data structure, and control structure, and provide coding guidelines and documentation standards. These tasks lay the groundwork which is essential to the restructuring design process, and thus merit long and serious consideration. However, flexibility to allow and in fact plan for review and revision of initial structural attributes, coding guidelines, and documentation standards is advisable to provide a better final result. Initial testing (by example or trial) of guidelines, particularly documentation and coding guidelines, may provide answers to critical questions such as 'Are the restrictions realistic?, Will they be complied with over a long program life-cycle by a variety of programmers/analysts/ users?, Will the documentation be maintained? Is it too cumbersome, or incomplete?, Will this structure still work if we add more coupling considerations?' Though not explicit in the plan outline, flexibility for refinement of guidelines is assumed.

A first-cut "Strawman" design of the overall program structure begins the actual design process. (This step was actually performed, as was required, in the mapping exercise which illustrated suitability of C-60 for restructuring. See Figure 3). A first-cut design of the Main Control Executive is obtained by viewing the analysis as consisting of only its top level "global" functions (e.g.: Velocity, Airloads, Trim, Response). The main design loop may then be executed on a function by function basis until all global functional modules have been designed, built, tested, and documented. The restructuring process is completed by final refinement of the Main Control Executive design, system integration testing, and completion of system documentation. Major stages of the restructuring process as outlined above are described in more detail in the paragraphs which follow.

Design principles were selected to place the desired emphasis on high priority criteria. To improve maintainability and modifiability, design principles are selected which minimize the human effort required to identify and correct program errors, and to define and implement changes to program requirements. Yourdon and Constantine[6] define desirable characteristics of the parts of a system for maintainability and modifiability:

> "....the cost of maintenance is minimized when parts of the system are:
> - easily related to the application
> - manageably small
> - correctable separately'
>
> "....the cost of modification of a system will be minimized when its parts are:
> - easily related to the problem
> - modifiably separately."

The design procedure should provide partitioning or organization of the analysis into pieces which are manageably small (for human comprehension), clearly defined, well documented, and which reflect the "real-world" partitioning of the problem. Similarly, these pieces should be connected to one another in ways which are clearly defined, well documented, and which reflect only "real-world" relationships without extraneous links. Design principles utilized in the present effort to build such a program structure are functional decomposition, hierachal control structure, and data traceability. These principles, which are described individually in the following paragraphs, are used to promote simplicity and clarity in the way the program models the solution it represents.

Functional decomposition is a top-down approach composed of repeated subdivisions from "big picture" functions to code level functions. It is the judicious partitioning of a problem into cohesive decoupled units or modules. (Detailed discussions of intermodular cohesion and intramodular coupling are presented in References 2 and 6). The trickiest aspect of this concept is determining 'functional decomposition with respect to what?'. "The choice of what to decompose with respect to has a major effect on the goodness of the resulting program and is therefore the subject of much controversy."[2] If the definition of functions is derived from a data flow diagram the result is data flow design. If the design is built on the basis of data structure it is data structure design. Bergland[2] describes data flow design (pseudonyms: transform centered design or composite design), in its simplist form as "nothing more than functional decomposition with respect to data flow. Each block is obtained by successive application of the engineering definition of a black box that transforms an input data stream into an output data stream". Another analogy is an engineering system block diagram with transfer function relationships between input and output for each block. The art or magic of functional decomposition is in definition of a model of the real world as functions.

It is in regard to this task, of judiciously breaking an analysis into functions that model the "real-world", that the role of engineer/analyst and the role of programmer/analyst have a critical interface. Careful partitioning may reflect not only the "state-of-the-art", but also the areas of anticipated expansion of capability. Careful functional partitioning should take care to explicitly represent all functions, including simple approximations of functions. This is necessary if the program structure is to accurately represent the structure of the "real-world" problem, and be easily identified and modified to improve the analytical model. The importance of good functional decomposition makes the engineer/analyst a critical, though often unused link in the software development/maintance/modification chain.

Hierachal control structure attempts to define clear traceable lines of decision-making power. This is accomplished by requiring that decisions be made only once, and by placing decisions immediately above the highest level module effected by the decision, thereby limiting authority of all lower level functions. The result may be likened to a human organization, in which "decision-making" power is graduated from top-level executives to mid-level managers to bottom level "number crunchers", as shown in Figure 5. A similar concept is defined by the term "decision-hiding"[4], in which decision information is accessible on a "need-to-know" basis. Hierarchal control structure tends to minimize extraneous control or decision connections, eliminate control redundancy, and guarantee consistency of all option selections. Involvement of engineering personnel in definition of the Control Hierarchy will provide additional insight into anticipated future analytical options and/or vehicle configuration.

Data traceability is the characteristic of having clearly defined single-source paths of data flow. This characteristic is desirable for maintainability and modifiability because it eliminates redundant and/or inconsistent variable definitions. This characteristic also tends to minimize extraneous connections by eliminating unnecessary and/or misplaced calculation of variables. The debate over transfer of information by argument list versus transfer by common "global" data is an example of an issue of data traceability. Argument list transfers can become cumbersome and may consume extra program execution time, but generally provide better data traceability. This issue is addressed specifically in definition of coding guidelines.

The third step in the restructuring process (outlined in Table 6) is the definition of a data structure. This refers to the organization of data flow from input parameters to output data. It consists of the division of program input data, variable parameters, and output data into categories and subcategories which reflect "real-world" definitions. Data structure is reflected in the organization of documentation, particularly "data dictionaries" which provide symbolic nomenclature, physical definition, units, sign convention, and FORTRAN name. The categories used for restructuring C-60 input data are trim, structural properties, aerodynamics, geometry, downwash parameters, and controls. (Program control parameters are treated with a parallel structure (e.g.: trim controls, input controls, etc.) under the topic of Control Structure). Parameters which are computed as functions of only input data, and which could, in fact, be treated as input data in that they remain fixed throughout the analysis are grouped with input data to form "extended data". Examples of additional parameters included in "extended data" are lumped physical properties derived from distributed physical property curves.

Organization of variable parameters (i.e.: parameters which change in value during the analysis) is based on functional decomposition. Variables are first defined either as "global" results of a specified function (e.g.: the results of the "response module" are deflections, slopes, loads, etc.) or as local internal values appearing only in subordinates of the specified function (e.g.: transfer matrix elements, unsteady stall time delay, etc). The resulting categories of variables thus reflect the top-level global functions. Refinement to repeatedly lower level functions provides similar organization of local variables to parallel data flow through the analysis.

Output data structure duplicates the structure of variable parameters, illustrating the concept that any resulting variable quantity may be considered an output. Highest level function output, or global output is collected for summary information.

The "output" function is otherwise distributed throughout the analysis by activation or "calling" of output utility routines or subfunctions. This treatment provides standardization of formats, simplified revision of standard formats, and simplified identification, tracing, addition or deletion of output parameters.

Definition of control structure refers to the organization of "decision-making" parameters or program controls. The organization of control parameters reflects the levels of decision-making defined by the control hierarchy as well as the functional breakup defined by functional decomposition. Controls are "decomposed" first by global function classifications: Input, Velocity, Airloads, Trim, Response, and Output. Successive subdivision from high-level decision-making to low-level decision-making is performed on a "need-to-know" basis to minimize coupling of program controls.

Coding guidelines are defined as the next step in the restructuring process to provide standardization, to improve clarity, to enforce data traceability requirements, and to maintain data and control structures. Standardization is needed to provide consistency across a variety of programmers/analysts/engineers with a corresponding variety of programming styles, nomenclature preferences, etc. The attributes of this standard will have a significant impact on future maintenance and modification costs. The most tempting tradeoff in coding practices is short-term expediency at the expense of long-term clarity. The provisions made in coding guidelines may deter such practices. Standards of particular interest from the perspective of maintainability and modifiability are those which influence data traceability. These standards are any rules or guidelines which promote the identification and understanding of a source of information. Some examples of standards which promote data traceability are prohibition or restricted use of "EQUIVALENCE" statements, FORTRAN variable naming conventions which reflect the meaning and source of the data and restricted use of FORTRAN COMMON blocks for variable data storage.

Usage of common block data storage is a particularly controversial issue because the trade-off of benefits is significant. Two main drawbacks of COMMON block usage are the difficulty in tracing the origin of values, and the danger of inadvertent and undetected redefinition or "over-writing" of data. A major benefit of COMMON block usage is the ease and simplicity of multi-point access to information. Common blocks of information may best be utilized as single-source, multi-destination vehicles of information transfer and storage to achieve major benefits and avoid main drawbacks of usage, as illustrated by Figure 6.

Definition of documentation standards is one of the most difficult and time-consuming tasks of the restructuring process. Repeated review/revision iterations involving personnel with different perspectives are essential to definition of useful documentation. Insights from perspectives of programming, analytical development, and production usage are essential. Documentation must provide enough information to provide needed understanding without providing so much information that needed information is obscured. In addition, the information must be structured so as to be easily modified and quickly retrieved and to prevent conflicting or redundant information.

First-cut designs of overall structure and the Main Control Executive reflect the interaction of global functions. Refinement of the design is achieved by iterative functional decomposition based primarily on data flow. The top-level second-cut design is shown in Figure 7. An essential element of the design iteration is that it involves individuals representing computer technology, the engineering developer, and the user community. The main design loop is then activated to design, build, test and document functions on a module by module basis. A second-cut detail design of the Input function is shown in Figure 8.

For each module, requirements must be defined in the form of a first-cut module specification or "module spec" including a first cut detail "strawman" design. Discussion of a design is much more fruitful when based on a strawman. It helps to point out the more subtle and obscure requirements and restrictions. It is also useful in establishing, illustrating, and clarifying special nomenclature. However, too large an effort should not be expended in putting together the "strawman" spec, or natural reluctance to "waste" effort may compromise the design effort by discouraging changes to the strawman. The iterative design-change/review process, starting with the strawman design is performed until a satisfactory detail design results.

Steps of module-building, validation, and documentation may begin upon completion of the module design. A mapping of the module function to the current analysis code is required to identify code connections for validation. There generally will not be a one-to-one mapping of new module code and original analysis code. The original analysis will contain duplicate code (possibly inconsistent near-duplicate sections), have some functions unrecoverably distributed (in practical terms), and have some functions which do not exist as functions at all. Coding guidelines define standards for the module-building process, including specification of the format for in-line documentation which should be included as comment statements in the code of the new module. The module is then tested by comparison with the current analysis intermediate and final calculation output. This can be achieved by adding namelist or write statements as temporary modifications to the original program. (The required variables from the original analysis were identified in the mapping step above). Documentation of the global module, including global function executive and all subordinate functions is then finalized.

The Main Control Executive design is finalized after completion of design of all global functions. This is necessary because design revisions of even global level functions may occur during the iterative design loop process. Testing of the main control executive is performed by exercising all combinations of global level control options with "dummy" global functions. Replacement of "dummy" functions with completed functional modules constitutes system integration, which is followed by integration testing, completion of system documentation, and finally, incremental release of the restructured program for production use.

Current Status

We are in the process of executing this restructuring plan for program C-60, and are presently in the global function design loop. Restructured functions which have been implemented in the production program are Downwash (subordinate of Velocity), Coupled Flap-Pitch Response (subordinate of Response), Aerodynamic Coefficient Determination (bottom level subordinate of Airloads), and various standard utilities of Input and Output. Some general observations we have made during this activity are (1) that no design is ever final and therefore flexibility must be built in, (2) review with other interested parties improves the resulting design, (3) definition of documentation standards is at least as difficult as the actual design process, and (4) the potential near-term benefits from implementation of restructured modules into production are extraordinary. A discussion of these points is given below.

The program design structure, even at the top global-function level, evolves during the design process, and beyond. Top-level functions of the current design, as shown in Figure 7, contrast the original design which was shown in Figure 3. Additional functions, such as nonlinear forcing, summary report, graphics, and rotor-airframe coupling, were added as a result of multi-person review, anticipation of future needs and insights from ongoing design refinement. Further revision is anticipated.

Documentation standards evolve in the design process, similar to the actual program structure evolution. Initial elements listed in Table 7 were revised to a scheme employing three documents with items listed in Table 8. The division, though still in work, is similar to that suggested by Kerr, Potthast, and Anderson[1]: model formulation, user's manual, and programming manual. This resulted from trial generation and revision of documentation, which proved to be too cumbersome in its original form. Again, multi-person review provided major insights. Documentation completed thus far, including the input data dictionary and input structure definition, has been very useful in reducing user errors in input preparation.

Restructuring can improve efficiency as a side-effect. Restructuring of the Aerodynamic Coefficient Determination function, a bottom level subordinate of Airloads, revealed inefficient loop structure resulting in unnecessary recalculation of values. Restructuring of the function to improve clarity, maintainability and modifiability also provided more efficient operation. The end result of implementing the restructured function in the production program was reduction of total CPU execution time by 17 percent for a typical case.

Previously undetected program errors may be uncovered in the process of validating restructured modules. During the validation process, differences in answers provided by the restructured module and the original program were found. Usually, this was the result of a coding error in the new module, which illustrates the primary purpose of the validation effort. However, sometimes it was found that the original program was incorrect. Most of the time these corrections were not significant, but in at least one instance, the error correction significantly improved correlation. In the past, the vibratory hub load calculations sometimes depended upon the accuracy of the initial trim guess. After an error was uncovered in the wake update routine, the dependence of the hub loads on initial trim was dramatically reduced. In addition, the correlation with measured pressure data was substantially improved, as shown in Figure 9.

Another restructuring benefit is that it has been significantly easier to incorporate new analytical capability. A number of improvements have been added. including the addition of a nonlinear multi-load path flex-beam capability, even though the Response function was only partially structured. Having a data map, control flow and partial restructuring really reduced the effort required. A number of similar improvements were attempted in the late 70's, but they had to be abandoned since the change could not be checked out with a reasonable effort. Restructuring of the Response function made a similar contribution to revisions to add nonlinear pendulum flap absorbers and consolidate calculations for load and frequency prediction capabilities.

Output capabilities have been expanded and made more user-friendly. Restructuring has facilitated development of flexible, centralized output modules from which the user can select any of 152 output record names representing different output arrays (each array is converted into one of three standard formats). Each of these arrays can be either not output or output to paper, microfiche, on-line printer plots and/or tape. The tape can then be automatically transformed into the required form for a number of display devices (both video and hard copy plots). In addition, all output is completely labeled with a description, units and sign convention.

Reorganization of input data and input processing has reduced input errors and the time required for user preparation of input data. This was accomplished by development of centralized input modules with options for using standard data files established for each aircraft and the ability to input either distributed or lumped physical properties. Scaling factors are provided for each of the physical property characteristics. Finally, the input data is printed out in both standard loader format and logically grouped, annotated and formatted arrays, with description, units and sign convention to allow the user to easily check the input.

Restructuring aids development of interdisciplinary analysis by simplifying the incorporation of analysis components from other technical areas. This capability was demonstrated while performing the restructuring of the Downwash function in the C-60 program. The restructuring of C-60's non-uniform downwash function resulted in the definition of the functional data connections so that any 'generic' downwash function (that used airloads and geometry as inputs and provided velocity distribution as output) could be used. Next, a non-uniform downwash analysis from the L-02 program was restructured and then transplanted into C-60. Currently a third downwash analysis (from the B-65 performance program) is in the process of restructuring for inclusion in C-60 by year's end. It should be noted that this technology interchange works both ways. Since the C-60, B-65 and L-02 wake functions were restructured to the same criteria, these interface boundaries are defined identically. Therefore, it is possible for both B-65 and L-02 to use the C-60 downwash function and each others' as well. The value of this exercise is to show that it is reasonable to transfer technology from discipline to discipline and to demonstrate the value of good (and consistent) program structure in aiding the transfer. Other technologies planned for near-term cross fertilization include unsteady aerodynamics, free flight aircraft trim, and dynamic flight controls.

Another benefit of this technology interchange is that the three restructured Downwash functions in C-60 will allow the ability to evaluate different analytical formulations for the same function. Different analytical models of such concepts as shed wake, vortex sheet, roll up, lift/wake compatibility and wake convergence, etc. can be investigated with identical external formulations, (a physical impossibility when these routines were located in different programs). Using similar strategy, functional structuring also allows multiple levels of complexity for a single function. For example, C-60 currently has three levels of complexity for blade pendulum absorbers: an idealized absorber (useable for preliminary design), a linear absorber (with couplings and offsets), and a full non-linear absorber (with large angles).

As discussed above, our experience to date shows that restructuring works, and provides extensive near-term benefits with production implementation. It has been possible to develop a hybrid program, partially structured and partially unstructured, that can be utilized as a production analysis in parallel with its restructuring. Of course, after each significant restructuring effort, a new production module is developed, which has the same or improved capability of the previous version. (A modification index is kept in the front of the output to summarize each change, as shown in Figure 10). When substantial differences in the calculated results between versions occur, correlation is performed to show that the new version is at least as good as the original version. If this level of correlation cannot be reached, the old version is not replaced.

Conclusion

Experience in the effort to restructure C-60 indicates that restructuring a "mature" analysis is a viable and worthwhile remedy for the maintenance and modification problems of a current poorly structured rotorcraft analysis. The success and productivity of the restructuring effort in terms of capability-gained for effort-expended depends to a large degree on the organization and documentation of the current poorly structured analysis. Many near-term benefits are possible from incorporation of restructured functional modules into the production program. Near-term benefits achieved thus far for program C-60 are listed below.

- Reduction of program errors
- Reduction of user errors (input and output)
- Reduced run-time
- Simplified incorporation of new capabilities
- Enhanced technology transfer
- Improved correlation

In addition, some important lessons were learned concerning the procedure or process of restructuring, and are outlined below.

- No design is ever final, and therefore flexibility must be built in.

- Reviews of design and documentation standards by representatives of programming, computer technology, analytical development, and user communities provide surprising insights and better "final" results.

- The definition of standards for documentation is at least as difficult and as important as the actual design process.

References

1. A.W. Kerr, A.J. Potthast, and W.D. Anderson, "An Interdisciplinary Approach to integrated Rotor/Body Mathematical Model", AHS Symposium on the Status of Testing and Modeling for V/STOL Aircraft, October 1972.

2. G.D. Bergland, "A Guided Tour of Program Design Methodologies", IEEE, October 1981.

3. FORTRAN Program Coding Standard for BCS Scientific Systems (VSD), Boeing Computer Services, Inc., February 1983.

4. C. Berggren, "Development of a Generic Architecture", AHS 40th Annual Forum Proceedings, pp. 429-437, May 1984.

5. H.H. Hyndman, Jr., "TOTAL FLOW" Computer Program, Boeing Computer Services, Inc., July 31, 1981.

6. E. Yourdon, and L.L. Constantine, Structured Design, Yourdon Press, New York, 1975.

Table 1.

SOPHISTICATED COMPUTER TOOLS

- GRAPHICAL DISPLAY PACKAGES
- OPERATING SYSTEMS
- COMPILERS
- DEBUGGING AIDS
- DATABASE MANAGEMENT SYSTEMS
- FLOW DIAGRAM UTILITIES
- CHART GENERATION PACKAGES
- WORD PROCESSORS

Table 2.

INDIVIDUAL PREFERENCES IN PROGRAM DEVELOPMENT

- NOMENCLATURE
- FORMAT
- VARIABLE NAMES
- UNITS/NONDIMENSIONALITY
- SIGN CONVENTIONS
- IN-CODE COMMENTS
- ARGUMENT OR COMMON TRANSFERS
- IMPLIED OR EXPLICIT LOOPS
- EXPLICIT OR VARIABLE DIMENSIONS
- INTEGER OR REAL PROGRAM CONTROLS
- VARIABLE PRECISION

Table 3.

SOFTWARE ATTRIBUTES AS DESIGN CRITERIA

- CLARITY
- COHESION
- COMPLEXITY
- CONNECTIVITY
- CONSISTENCY
- CONTINUITY
- CORRECTNESS
- CORRESPONDENCE
- COUPLING
- EFFICIENCY
- EXTENDIBILITY
- FLEXIBILITY
- GENERALITY
- INDEPENDENCE
- MAINTAINABILITY
- MODIFIABILITY
- MODULARITY
- PORTABILITY
- RELIABILITY
- REUSEABILITY
- ROLE ADAPTABILITY
- TESTABILITY
- TRANSPARENCY
- UTILITY

Table 4.

SOFTWARE DESIGN PRINCIPLES AND TECHNIQUES

- ABSTRACTION
- BOTTOM-UP DESIGN
- COMPOSITE DESIGN
- DATA ENCAPSULATION
- DATA FLOW DESIGN
- DATA STRUCTURE DESIGN
- DATA TRACEABILITY
- DECISION HIDING
- FUNCTIONAL DECOMPOSITION
- HIERARCHY
- HIERARCHAL CONTROL
- HYBRID (TOP-DOWN/BOTTOM-UP) DESIGN
- INFORMATION HIDING
- PARTITIONING BY OBJECTIVE
- PROGRAMMING CALCULUS
- STEPWISE REFINEMENT
- TOP-DOWN DESIGN
- TRANSFORM-CENTERED DESIGN

Table 5.

DESIGN CRITERIA DEFINITIONS

EFFICIENCY	—	OVERALL COMPUTATIONAL EFFICIENCY
GENERALITY	—	BROADNESS, SCOPE, OR ABSTRACTNESS OF TASK DEFINITION
MAINTAINABILITY	—	EASE OF DETECTION AND CORRECTION OF PROGRAM ERRORS
MODIFIABILITY	—	EASE OF ACCOMMODATING NEW REQUIREMENTS
RELIABILITY	—	CONSISTENCY AND REPEATABILITY OF CORRECT PERFORMANCE
UTILITY	—	EASE OF USE OR USER-FRIENDLINESS

Table 6.

PLAN FOR RESTRUCTURING C-60

1. PRIORITIZE DESIGN CRITERIA
   - MAINTAINABILITY
   - MODIFIABILITY
2. DEFINE DESIGN PRINCIPLES
   - FUNCTIONAL DECOMPOSITION
   - HIERARCHAL CONTROL
   - DATA TRACEABILITY
3. DEFINE DATA STRUCTURE
4. DEFINE CONTROL STRUCTURE
5. DEFINE CODING GUIDELINES
6. DEFINE DOCUMENTATION STANDARDS
7. DESIGN 1st-CUT OVERALL STRUCTURE ("STRAWMAN")
8. DESIGN 1st-CUT MAIN CONTROL EXECUTIVE
9. GLOBAL FUNCTION DESIGN LOOP
   - DESIGN/REVIEW ITERATION
   - BUILD MODULE
   - TEST MODULE
   - COMPLETE MODULE DOCUMENTATION
   - (NEXT GLOBAL FUNCTION)
10. COMPLETE MAIN CONTROL EXECUTIVE
11. SYSTEM INTEGRATION TESTING
12. COMPLETE SYSTEM DOCUMENTATION

Table 7.

ORIGINAL DOCUMENTATION REQUIREMENTS

MODULE SPEC (FOR EACH SUBROUTINE)

I. SUMMARY SHEET
(FUNCTION NAME AND DESCRIPTION, SUBORDINATES AND SUPERORDINATES, INPUT AND OUTPUT PARAMETERS)

II. LOCAL MODULE MAP (DIAGRAM)
(SHOWS SUBORDINATES, SUPERORDINATES, ARGUMENT LIST I/O, COMMON I/O)

III. MODULE STRUCTURE DIAGRAM
(SHOWS INTERNAL STRUCTURE SUBFUNCTIONS)

IV. SEQUENCE FLOW DIAGRAM
(CLASSICAL FLOW CHART)

V. CONTROL DECISIONS
(PRESENTS OPTIONS AND TESTS MADE ON CONTROL PARAMETERS)

VI. EQUATIONS

VII. DERIVATION OF EQUATIONS

VIII. FORTRAN VARIABLES
(FORTRAN NAMES, SYMBOLIC REFERENCES, AND DEFINITIONS)

IX. CODE (WITH COMMENTS)
(LISTING OF ACTUAL FORTRAN CODE WITH IN-LINE COMMENTS)

X. ADDITIONAL NOTES
(ANY ADDITIONAL COMMENTS, A SUCH AS NOTING EXCEPTIONS TO CODING GUIDELINES, ASSUMPTIONS OR ANTICIPATION OF FUTURE REVISIONS)

DATA DICTIONARIES - INPUT, VARIABLES, OUTPUT

- ALPHABETIZED AND CROSS-REFERENCED
- PROVIDE SYMBOLIC NOMENCLATURE, PHYSICAL DEFINITION, UNITS, SIGN CONVENTION, FORTRAN NAME, AND DERIVATION REFERENCE

Table 8.

REVISED (2nd CUT) DOCUMENTATION REQUIREMENTS

DERIVATION DOCUMENT

- CONTINUOUS/SEMI-CONTINUOUS DERIVATION (NOT BROKEN INTO SUBROUTINES)
- CROSS-REFERENCED TO SUBROUTINES, INPUT DATA, OUTPUT DATA, AND FORTRAN VARIABLES

PROGRAMMING MANUAL

- MODULE SPEC*
  - I. SUMMARY SHEET
  - II. LOCAL MODULE MAP
  - III. MODULE STRUCTURE DIAGRAM
  - IV. SEQUENCE FLOW DIAGRAM
  - V. CONTROL DECISIONS
  - VI. FORTRAN VARIABLES (WAS VIII)
  - VII. CODE WITH COMMENTS (WAS IX)
  - VIII. ADDITIONAL NOTES (WAS X)
    - REGARDING CODE
- DATA DICTIONARY - VARIABLES

USER'S MANUAL

- MODULE SPEC*
  - I. SUMMARY SHEET
  - II. EQUATIONS (WAS VI)
  - III. ADDITIONAL NOTES (WAS X)
    - REGARDING EQUATIONS
- DATA DICTIONARIES-INPUT AND OUTPUT

*ASSEMBLED FROM PIECES OF ORIGINAL MODULE SPEC STANDARD

Figure 1.

LAW OF CONTINUING CHANGE:

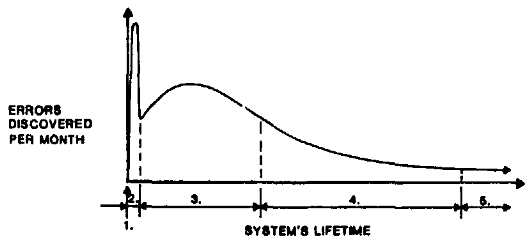
A SYSTEM THAT IS USED UNDERGOES CONTINUING CHANGE UNTIL IT IS JUDGED MORE COST-EFFECTIVE TO FREEZE AND RECREATE IT.

LAW OF INCREASING UNSTRUCTUREDNESS:

THE ENTROPY (DISORDER) OF A SYSTEM INCREASES WITH TIME UNLESS SPECIFIC WORK IS EXECUTED TO MAINTAIN OR REDUCE IT.

Figure 2.

COMPUTER PROGRAM LIFE CYCLE



1. UNDER DEVELOPMENT
2. CHECKOUT
3. INITIAL RELEASE
4. OPERATION/MAINTENANCE/MODIFICATION
5. MATURE PROGRAM

Figure 3.

FIRST CUT FUNCTIONAL DECOMPOSITION

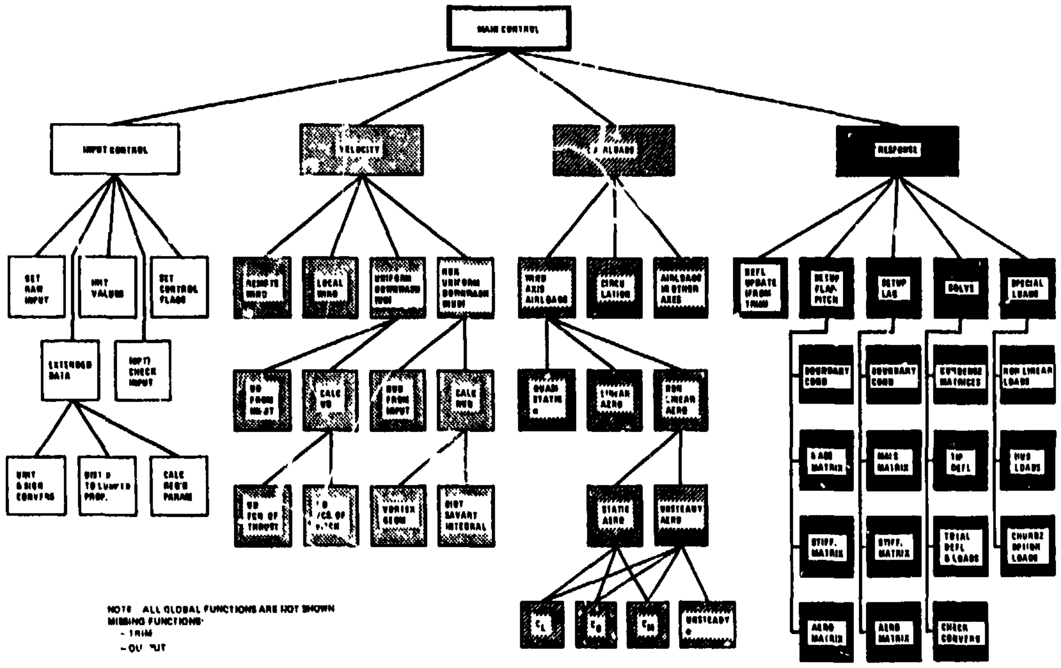


Figure 4.

FUNCTIONAL MAPPING FOR CURRENT C-60 PROGRAM

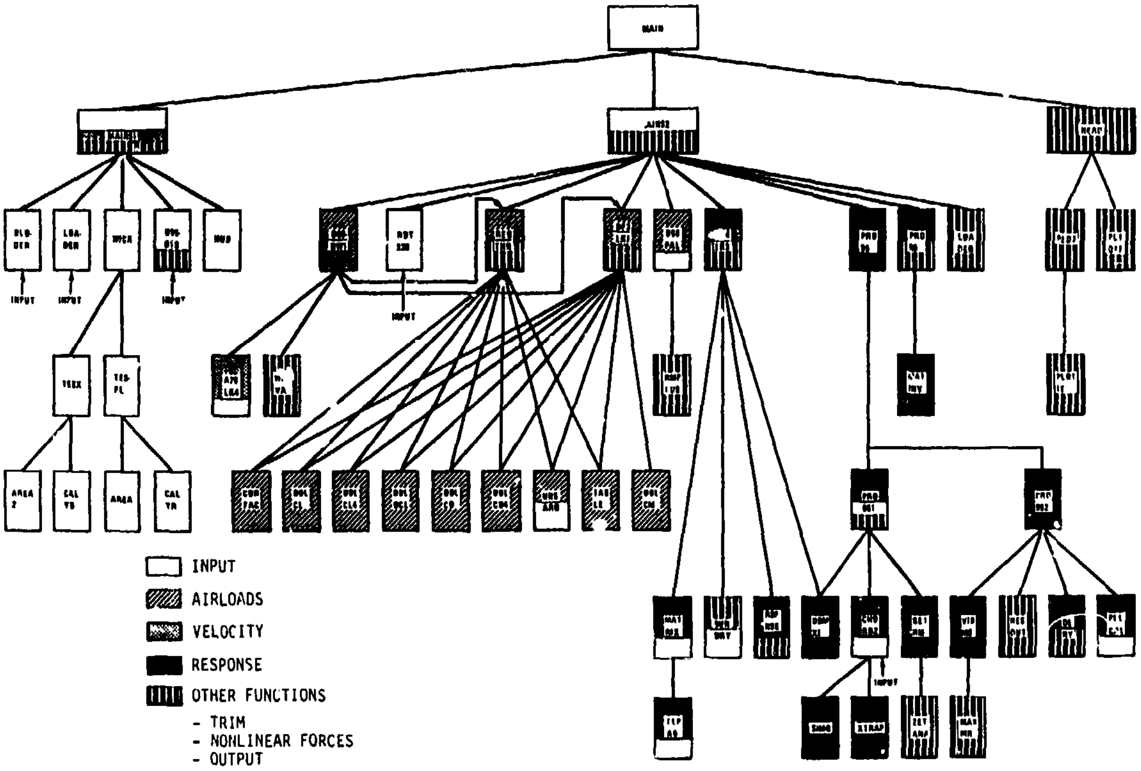


Figure 5.

HIERARCHAL CONTRC

GRADUATION OF DECISION-MAKING

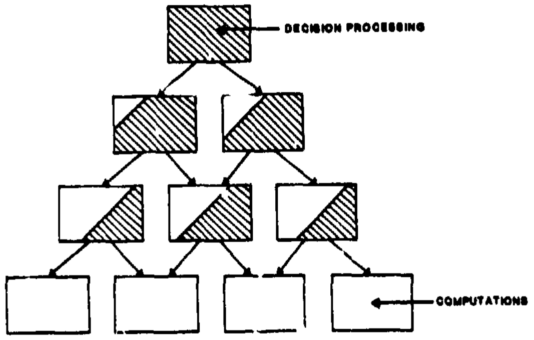
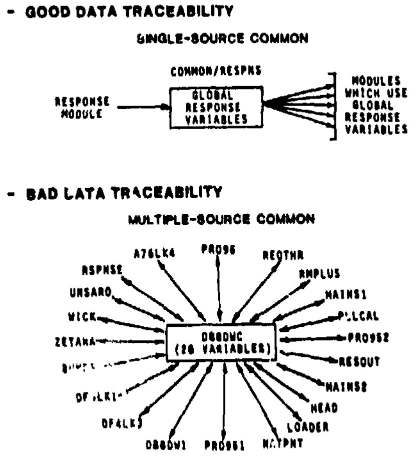


Figure 6.

GOOD AND BAD DATA TRACEABILITY FROM COMMON BLOCK USAGE


- GOOD DATA TRACEABILITY

SINGLE-SOURCE COMMON

COMMON/RESPNS
RESPONSE MODULE
GLOBAL RESPONSE VARIABLES
MODULES WHICH USE GLOBAL RESPONSE VARIABLES

- BAD LATA TRACEABILITY

MULTIPLE-SOURCE COMMON

ORIGINAL PAGE IS
OF POOR QUALITY

Figure 7.

TOP LEVEL OF SECOND CUT DESIGN

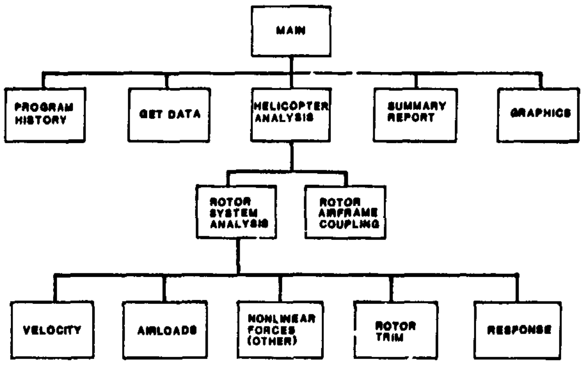


Figure 8.

INPUT FUNCTION - SECOND CUT DESIGN

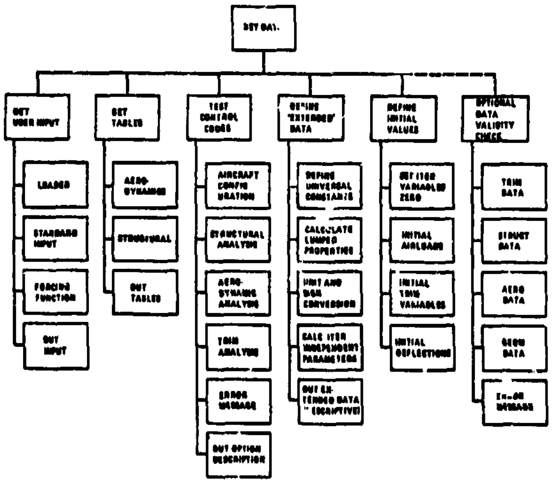


Figure 9.

IMPROVED VIBRATORY AIRLOAD CORRELATION USING RESTRUCTURED C-60 DOWNWASH MODEL

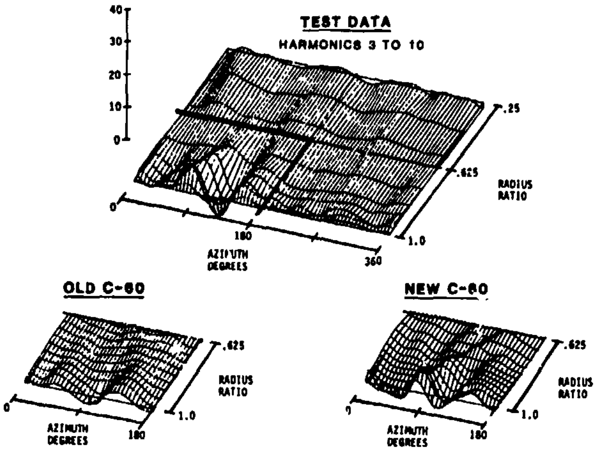
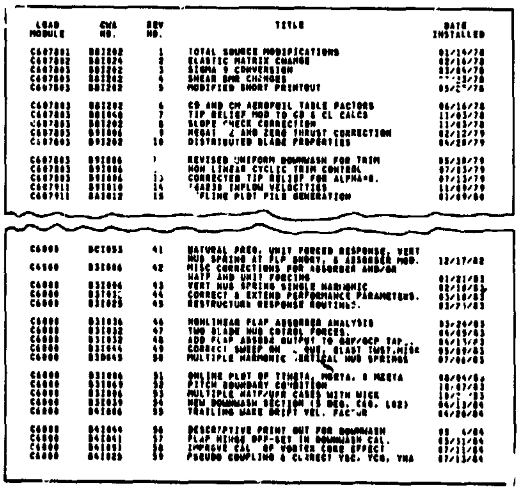


Figure 10.

C-60 AUTO-HISTORY



LOAD MODULE	CNA NO.	REV NO.	TITLE	DATE INSTALLED
C607801	B01202	1	TOTAL SOURCE MODIFICATIONS	01/14/78
C607802	B01024	2	ELASTIC MATRIX CHANGE	02/14/78
C607803	B01202	3	SIGMA 9 CONVERSION	03/04/78
C607803	B01202	4	SHEAR BMR CHANGES	[illegible]/23/78
C607803	B01202	5	MODIFIED SHORT PRINTOUT	05/[illegible]/78
C607803	B01202	6	CD AND CM AEROFOIL TABLE FACTORS	06/16/78
C607803	B01040	7	TIP RELIEF MOD TO CD & CL CALCS	11/03/78
C607803	B01202	8	SLOPE CHECK CORRECTION	11/03/78
C607803	B91006	9	NEGAT Z AND ZERO THRUST CORRECTION	02/12/79
C607803	B91202	10	DISTRIBUTED BLADE PROPERTIES	04/20/79
C607803	B91006	1	REVISED UNIFORM DOWNWASH FOR TRIM	05/30/79
C607803	B91006	.	NON LINEAR CYCLIC TRIM CONTROL	07/13/79
C607803	B91006	13	CORRECTED TIP RELIEF FOR ALPHA=0.	07/13/79
C607911	B91010	14	F4A23D INFLOW VELOCITIES	11/09/79
C607911	BA1012	15	FLINE PLOT FILE GENERATION	01/09/80
C6000	BC1053	41	NATURAL FREQ. UNIT FORCED RESPONSE, VERT HUB SPRING AT FLP SHORT, & ABSORBER MOD.	12/17/82
C6100	B31006	42	MISC CORRECTIONS FOR ABSORBER AND/OR HATP AND UNIT FORCING	01/21/83
C6000	B31006	43	VERT HUB SPRING SINGLE HARMONIC	02/10/83
C6000	B31015	44	CORRECT & EXTEND PERFORMANCE PARAMETERS.	03/10/83
C6000	B31025	45	RESTRUCTURE RESPONSE ROUTINES.	03/21/83
C6000	B31036	46	NONLINEAR FLAP ABSORBER ANALYSIS	03/24/83
C6000	B31032	47	TWO BLADE HUB COTROL FORCES.	04/05/83
C6000	B31032	48	ADD FLAP ABSORB OUTPUT TO OGP/OCP TAP..	04/13/83
C6000	B31044	49	CORRECT SWEEP ON . OUT, ELAST TWST,MISC	05/07/83
C6000	B3D043	50	MULTIPLE HARMONIC VERTICAL HUB SPRINGS	07/06/83
C6000	B31006	51	ONLINE PLOT OF THETA, [illegible], & ZETA	08/04/83
C6000	B31069	52	PITCH BOUNDARY CONDITION	10/07/83
C6000	B31006	53	MULTIPLE HATP/UPR CASES WITH WICK	10/[illegible]/83
C6000	B3E039	54	NEW DOWNWASH SECTION (5 DEG. CAL. LOZ)	04/13/84
C6000	B41006	55	TRAILING WAKE DRIFT VEL. FACTOR	04/20/84
C6000	B41040	56	DESCRIPTIVE PRINT OUT FOR DOWNWASH	05 [illegible]/84
C6000	B41041	57	FLAP HINGE OFF-SET IN DOWNWASH CAL.	05/31/84
C6000	B41051	58	IMPROVE CAL. OF VORTEX CORE EFFECT	07/11/84
C6000	B41025	59	PSEUDO COUPLING & CORRECT YBC, YCG, YHA	07/13/84

DISCUSSION
Paper No. 12

RESTRUCTURING THE ROTOR ANALYSIS PROGRAM C-60
P. G. Phelan
and
F. J. Tarzanin, Jr.

Wayne Mantay, U.S. Army Structures Laboratory: I have two questions that I think are related, I hope you do too. When you restructured you said you uncovered at least one major problem. Was it in fact that error in the downwash, was that the one?

Phelan: That showed one of them, yes. Part of the improved correlation was that. That wasn't the total improvement in correlation; we also had improved capability.

Mantay: No, I understand about the correlation, but the error that you uncovered in C-60, was that the major error you alluded [to]?

Phelan: Yes.

Mantay: That was in the downwash?

Phelan: Yes, that was in the downwash.

Mantay: I didn't pick it up on your slide for that correlation, but was that the high speed case from Euan Hooper's data base that he had trouble with; was that the problem child that you set straight?

Phelan: Yes.

Marty Schroeder, Solar Energy Institute: Your presentation was [very good] and I think the work in structured programming is sorely needed. I'm not familiar with C-60 though. What language is it written in?

Phelan: FORTRAN.

Shroeder: Have you considered looking at other languages like PASCAL or C for a structured program?

Phelan: We are working on restructuring an existing analysis for a lot of reasons that I didn't really get into, but are in the paper. We haven't looked at also changing languages, but you could do that. We haven't looked at doing that.

Wendell Stephens, U.S. Army Aeromechanics Laboratory: I wanted to thank you for your paper, also. I have noticed that your paper, the one previous to it by Bob Sopher, a related paper [on] DYSCO involving Kaman, [and] perhaps, even the paper by Gangwani at Hughes when he spoke about a new program all have tended to go this direction which I applaud. My specific question to you is when you begin restructuring this program have you come across any executive-type utilities that you have had to build in FORTRAN that have helped your ability to transfer data from module to module. It sort of relates to the previous question back here of perhaps going to a different language for certain structures for your executive functions. I was wondering if you found that you had to develop any utilities for data transfers?

Phelan: We haven't yet, but I think part of that is maybe that our final main control executive design is . . . the first cut comes at the beginning of the process--the last final design comes at the end after you have decided and really finalized what your global top level functions are. So we've discussed different ways to implement a main control executive quite a bit, but we have not implemented it as of yet. So, we will see.

Ed Austin, U.S. Army Applied Technology Laboratory: I think it's very interesting the approach you have taken to conceive of restructuring an entire program and then to work from kind of the bottom up. Do you have any speculations about what will happen on that day when suddenly the master program is the only thing left to change and you have all these pieces and you haven't designed apparently the final master program? It doesn't sound to me like you have anyway.

Phelan: You mean the main control as far as the . . .

Austin: I saw this incredible diagram of the program the way it was before you started and wires go every which way. I saw your final version which is a very simple wiring diagram. I guess I don't see how you plan to go from this complicated set down to just a few wires. It's kind of like the time my wife decided to rewire our car and she took all the wires off the distributor at the same time.

Phelan: I don't see it as a big problem because the reason why the design is . . . we have done a lot of work on laying out a first cut executive, but the problem is you cannot implement the executive in a piecemeal way because that defines . . . one of the very last things is how you are going to make these common blocks all well structured. Well, when you start out with one that is connected everywhere, you can't do that first; you have to wait until you have consolidated things. I can't take the connections from airloads that go everywhere and eliminate the connections until I've brought all the pieces together and once they are in one piece then you have identified your single source. Do you see what I mean? Do you have a better explanation?

Frank Tarzanin: We have done some partial module restructuring and then we have done, actually completing the downwash module restructuring and once you connect those wires you can isolate the downwash as a kind of structured subprogram. We're going to slowly build structured subprograms then build the total program on top. In fact we are learning and what we started to do was take one thing in airloads, in fact that routine that saved a lot of time was an experiment. Can we grab a routine out of the middle of this mess and restructure it and put it back in and have it work. And it did. You've got to find the connections, and define the interface.

Phelan: I think that another thing that is important too is when you do that the connections are reduced Because what started out as a ball with a whole bunch of strings attached--you can eliminate the strings. You have identified a single source that can go everywhere as you consolidate all through; they kind of connect.

Tarzanin: Each step makes it simpler.