Analytical Design and Assurance of Digital Flight Control System Structure

D. B. Mulcare,* W. G. Ness,† and R.M. Davis‡
Lockheed-Georgia Company, Marietta, Georgia

With the increasing prevalence of flight-critical functions, sophisticated fault tolerance, and functional integration in digital flight control systems (DFCSs), purposeful design of system structure can no longer be neglected. Needed then are criteria and methods to obtain good system structure, which is a prerequisite to good flight software structure. Basically, this calls for a new awareness in the flight controls community of a distinct and coherent body of techniques for the analytical design of structure that is comparable with that now used for the analytical design of control laws. In the software community, it calls for a recognition that the characteristics of redundant, real-time, closed-loop digital systems demand rather specialized methods that transcend strictly software issues. An exacting, practitioner-oriented methodology has been developed that enables the analytical design of DFCS structure essentially independent of control law development. Utilizing control state decomposition, the methodology is based on a design paradigm that facilitates analytical design verification in the form of consistency, completeness, and reachability checks. Heavy emphasis is also placed on quantitative methods to control design and implementation complexity and on tradeoff techniques to enhance the "ilities" from the outset of design.

Introduction

SYSTEMS are composed of structure and function or, in flight controls parlance, system architecture and control laws. As used herein, structure is the arrangement and interrelationships of the system or software elements, actual or conceptual. Thus, structure can be defined independent of an implementation. In typical development projects, unfortunately, system structure is addressed only on a shallow level that evades meaningful consideration and conscious design of important details. As a result, structure usually evolves by default as premature or unsubstantiated hardware and software implementation decisions are made, usually to the detriment of the quality of the design.

Such practice contrasts with analytical control law development, which is normally performed on a detailed, methodical basis. This is possible because the supporting technology, control theory, is a formal and well-developed academic discipline. Furthermore, its practical value is well recognized as necessary to good functional performance of a flight control system. On the other hand, system structure is not so readily perceived or adequately understood as control laws. This results because:

1) Structure is a rather vague term and criteria for good structure are not yet solidified.

2) Inferior structure is not very evident, except where problems of safety criticality, functional intricacy, or maintenance arise.

3) Even then, inferior structure may not be recognized as an actual source of problems.

4) Relevant technology is rather immature, and it is neither unified nor application-oriented.

Unfortunately, the need for the analytical design and verification of system structure has not been generally recognized for several reasons:

1) Intuitive, brute-force approaches have usually sufficed for the limited-scope digital flight control systems (DFCS) implemented to date.

2) The very notion of analytical, quantitative design and verification of structure is generally foreign to flight controls practitioners.

3) An acceptable methodology has not been fully developed or demonstrated for flight control applications.

Although useful methods exist for the design of software structure (e.g., see Refs. 1 and 2), this technology falls short of meeting the demands of real-time, fault-tolerant systems. To begin with, it is difficult to delineate between hardware and software in such systems, so the comparatively tractable task of designing software per se is more limited in scope than might generally be thought. Furthermore, the quality of the DFCS software structure is highly dependent on the prior existence of good system structure.

The intent of this paper is to describe that portion of an integrated development methodology dealing with the analytical design of DFCSs and the verification of such designs and to illustrate the more important aspects of the methodology. Reference 3 describes the application of this methodology in substantial detail, so the greater emphasis here is on the definition and rationale of the analytical design/verification methods.

Problem

Overall, the problem addressed is that of formulating, developing, and validating an analytically based methodology that can dependably yield a readily validatable and maintainable DFCS. As indicated in Fig. 1, a dependable and effective methodology demands an appropriate degree of rigor or exactitude to constructively guide the system development activities. Simultaneously, the methodology needs a certain sense of pragmatism to adapt to the application and its development environment. Overall, an effective methodology should engender user acceptance and warrant increasing confidence with use.

Presented as Paper 82-1626 at the AIAA Guidance and Control Conference, San Diego, Calif., Aug. 9-11, 1982; submitted Aug. 12, 1982; revision received June 22, 1983.

*Staff Engineer, Stability and Flight Controls Department. Member AIAA.

†Specialist Engineer, Reliability Engineering Department.

‡Senior Engineer, Stability and Flight Controls Department. Member AIAA.

Figure 1 also identifies the essential elements of the methodology, such as incremental analytical design verification. Regarding the realization of these elements, and more particularly the attributes or benefits sought through the methodology, the following maxims have provided fundamental guidance:

1) Simplicity: "The price of reliability is the pursuit of utmost *simplicity.*"[4]

2) Visibility: "...graphics-based languages excel character-string languages in both speed of information transfer and overall *clarity.*"[5]

3) Consistency: "...*conceptual integrity* is the most important consideration in system design.... Simplicity and straightforwardness proceed from conceptual integrity."[6]

4) Conclusiveness: "Validation is meaningful only with respect to a statement of needs or requirements... it would be prudent to design systems so that their most critical functions can be analyzed (validated) with the *highest degree of confidence.*"[7]

Software Engineering Methodologies

It has been noted[8] that software engineering continues to be practiced on an *ad hoc* basis with somewhat limited success because of the lack of suitable scientific underpinnings. Among some of the less sucessful methodology experiences, the pursuit of too much generality or the neglect of application considerations have been identified as contributing factors.[9] This is particularly true in the case of embedded computer systems such as DFCSs. Of specific concern is the lack of accommodation of problem peculiarities, practical constraints, system/software distinctions, and exigencies of the development process. It should not be surprising then that none of the publicized software engineering methodologies has been found to be acceptable for flight-critical DFCSs. All have been found to suffer from at least some of the following drawbacks:

1) System issues are subordinated or neglected—software alone is more tractable.

2) Design descriptions are unsatisfactory—representations are bulky, prescriptive, trivial, and/or abstruse.

3) The structure is not suitably and explicitly designed—complexity is not characterized and analyzed.

4) Validation is an afterthought—designed-in assurance considerations are lacking.

5) Methodology is nonresponsive to DFCS needs—only a few partially developed approaches are well suited.

Despite the unacceptability of any publicized methodology, many individual software engineering methods or strategies hold promise for fulfilling the needs of DFCS applications. The problem is initially one of defining DFCS methodology requirements. Second, it is primarily one of selectively adapting or developing available software engineering techniques and of integrating them into a realistic system development process.

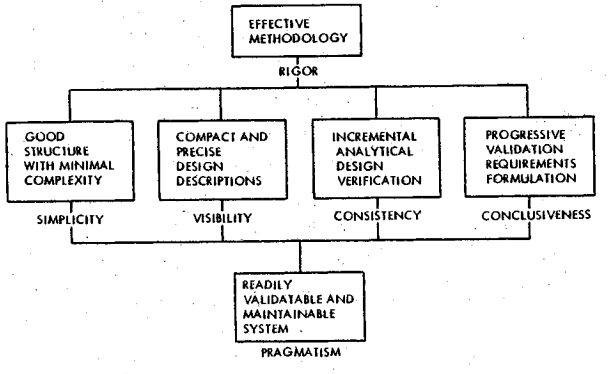

Fig. 1 Elements and attributes of an effective methodology.

DFCS Development Practices

As previously indicated, inferior development practices have yielded deployable DFCSs largely because the requirements addressed have been comparatively modest. Basically, the approach has been to digitize analog-oriented designs. Specific aspects of current practice that impede the DFCS development process or compromise its product include:

1) Allocation into hardware and software elements is done prematurely.

2) Analysis of structure is generally done to justify the design, rather than to determine it.

3) Only failure effects analysis is customarily performed on an in-depth basis.

4) The impact is often restricted to redesign, rather than design, of system structure.

5) Design decisions are often made implicitly or without consideration of all crucial issues.

6) The bases for the design decisions are usually poorly documented.

Considerable research and technology effort has been devoted to direct digital design, but only with regard to control law implementation. This is not where the crucial problems are. From a priority standpoint, a correspondingly

Table 1 General approach to the methodology (see Table 4 for elaboration)

PROPERTY OF PRODUCT / ELEMENT OF METHODOLOGY	SIMPLICITY	VISIBILITY	CONSISTENCY	CONCLUSIVENESS
ANALYTICAL DESIGN OF STRUCTURE	USE COMPLEXITY METRICS TO MAKE DESIGN DECISIONS	Good Structure Greatly Enhances Visibility	Consistent Design Fosters Good Structure	Reduced Complexity Facilitates More Thorough Testing
DESIGN DESCRIPTIONS	Abstraction of Key Characteristics Simplifies View	EMPHASIZE GRAPHIC FORMATS AND ABSTRACTION	Some Description Formats are Amenable to Analysis	Design/Analysis Models can Serve as Execution Monitors
INCREMENTAL DESIGN VERIFICATION	Reduced Structural Complexity Facilitates Interface Assessment	Stepwise Verification Fosters Design Comprehensibility	APPLY STATIC TOOLS AND EXACTING ASSESSMENT CRITERIA*	Key Design Features Can be Designated for System Testing
VALIDATION REQUIREMENTS	Test Observability Simplifies Results Interpretation	Availability of Instrumentation Points Aids Observability	Specification Consistent with User Requirements Facilitates Validation	COMPILE TESTABLE, MULTI-LEVEL REQUIREMENTS

direct and unimpeded approach to the design and verification of digital system structure is needed more. This is where the leverage exists to improve safety and reduce life cycle costs.

DFCS Application Characteristics

Distinctive characteristics of DFCS requirements tend to yield rather specialized hardware and software. Furthermore, the typical development program places significant demands or constraints on design/verification methods. In the case of safety-critical functions especially, the overall result is the desirability, if not the necessity, of a highly specialized, optimized methodology. DFCS design or implementation aspects that warrant particular emphasis are: control logic and structure, hardware/software interactions, timing/sequencing constraints, boundary conditions, latent software errors, and redundancy management in general.

Other crucial determinants of an effective methodology derive from the nature of a representative DFCS development project, which may or may not be a subset of an aircraft design. In either case, the methodology must be compatible with and supportive of the overall project. Associated considerations of importance are:

1) Aircraft/DFCS design iterations introduce unavoidable requirements changes into the development process.

2) Design and procurement specifications must proceed despite such uncertainties.

3) Typically, the entire DFCS development is accomplished by a relatively small group of practitioners.

4) Substantial integration testing and post-integration development are required.

5) In the case of flight-critical functions, system safety and performance assurances may be a prerequisite for any flight testing.

These factors indicate the need for a practitioner-oriented methodology that is quite robust, yet capable of attaining very high assurance levels. Design iterations and system development changes place a premium on modifiability. Those changes relating to control laws tend to be readily accommodated in the flight software, but those relating to structure frequently cannot be. Hence, it is desirable to verify the design of system structure before implementation decisions or commitments are made.

General Approach

The overall approach to the design/verification methodology is the synthesis of software engineering methods and an analytical flight controls mode of thinking, along with the close coupling of various development process activities. The basic strategy is to facilitate validation and modifiability through good structure and appropriate documentation of both the system and the embedded software. In turn, good structure is to obtained through control state decomposition and complexity metrics, and suitable design descriptions through compact and precise formats. In all of these endeavors, the intent is to extend and exploit the training and orientation of analytical flight controls practitioners as applicable to the structure of software systems.

The basic position is taken that DFCS characteristics and practitioner needs must determine the methodology requirements, which in turn dictate the tools. This is in accord with the software engineering guidelines advocated in Refs. 10 (requirements analysis) and 11 (tailoring of tools). With the foregoing background established, the remainder of this paper elaborates, substantiates, and illustrates the tools, methods, and procedures for a DFCS-oriented methodology. In particular, this is done for one that embodies the following features:

1) A priori composition: control state (not functional) decomposition and semiautonomous design of structure and function.

2) Explicitly developed elements: design emphasis on structure and logic and the avoidance of premature or implicit decisions; diverse use of design data base and models; incremental analytical (nor formal) design verification; and propagated, multilevel validation requirements.

Previously introduced in Fig. 1, the four elements explicitly sought as part of the methodology are discussed further in the next section. Table 1 summarizes the major thrusts and rationale for achieving these features and also indicates how the resultant product attributes are fostered in a coherent, highly integrated methodology. The a priori features of the methodology are considered in the following two subsections.

Control State Decomposition

State decomposition for DFCSs has previously been advocated and illustrated[12] and, still earlier, its suitability for real-time systems in general has been described.[13] System state has particular meaning in terms of operational capability,[14] and ultimately substates reflecting selected DFCS modes relate well to software execution path traversals. Additionally, the state tables and transition graphs used to define finite-state machines (FSMs) are familiar to many flight controls practitioners. More importantly, FSMs are compact, analyzable, and capable of modeling hardware and/or software.

Beyond this, state decomposition permits DFCS structural design to proceed in parallel with functional or control law algorithm design. This does not seem possible with the more generally prevalent functional decomposition approach, which additionally tends to yield trivial description tasks and unwieldy structure in DFCS applications. General concerns over functional decomposition, moreover, have been expressed elsewhere.[1]

DFCS Design Paradigm

Following the definition of control states, the design elaboration proceeds through the iterative use of the model in Fig. 2. Since the control structure dominates the DFCS mechanizations, it is the logical point of departure for design; the functional flow for both the system and the software is then readily configured in response to the various control states. Subsequently, implemented functions can be incorporated into the control structure with the substantial flexibility provided. Finally, the data flow through the system and the executing software must be accounted for, especially with regard to the typically involved set of Boolean variables that alter and reflect the total system control state.

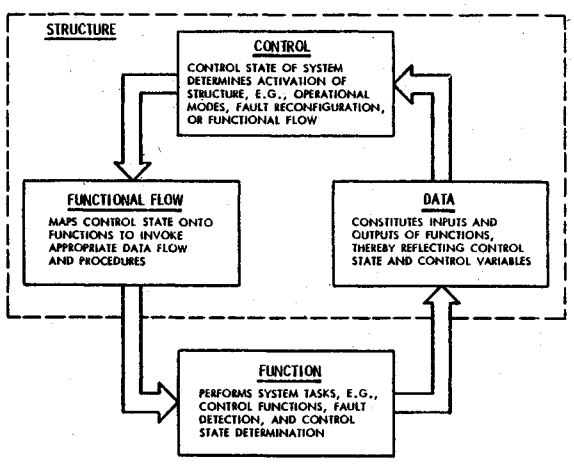

Fig. 2 DFCS design model.

Technical Approach

For the initial stages of development, the methodology is based on successively more detailed iterations of the design model in Fig. 2. As depicted in Fig. 3, the model is translated

into design, analysis, and verification tasks. The associated activity is represented in an input-process-output form wherein the input consists of the design and verification requirements levied by a prior phase, plus all of the then-existing constraints. The internal loops of the process are traversed until all of the design and verification requirements for that stage are shown to be satisifed analytically. At this point, the process outputs are propagated as requirements and constraints for subsequent developement phases.

As described below, the synthesis and analysis of each of the four aspects of the design model in Fig. 2 are accomplished such that documentation and verification are accomplished concurrently with design increments. Since the activities performed to realize the control functions are essentially reflected in current practice, the emphasis is on the control, functional flow, and data aspects. As indicated in Fig. 2, these are the elements that constitute the structure.

The design of DFCS structure is reviewed first to describe the activities encompassed within the rectangular boxes in Fig. 3. The design description methods to capture the resulting information are then discussed, with emphasis on the support of analytical design verification methods. For the most part, development of the latter is described individually for each of the four design model aspects. This then explains the four inner-loop closures through the hexagonal boxes in Fig. 3. The other aspect of design verification, which relates to explicit requirements examined in the decision block, is rather conventional and straightforward. Accordingly, it receives little attention. Last, system validation is examined from the viewpoint of increasing conclusiveness through multilevel testing.

System/Software Structure Design

Following the definition of a functional architecture and the system requirements, the former is expanded to yield a detailed conceptual design of the system structure. This design is largely determined by the system requirements pertaining to the "ilities." For example, the quantitative flight safety and mission reliability requirements of Ref. 14 apply on Air Force projects. In the civil domain, a comparable document[15] designates quantitative limits on the likelihood of system failure according to the criticality of its functions.

Various types of redundancy and reconfiguration options are therefore incorporated into the system structure to enable compliance with such requireements. Functional system requirements determine the types and locations of many system elements. In the early-on stages of design, the representation and analysis of system structure are quite tractable, but the step from high-level conceptual design to implementation design is formidable, even for a well-conceived baseline functional architecture.

The approach here is to extend the abstract definition of structure through iterations of the DFCS design model presented in Fig. 2. This must be done with continual attention to the impact on evolving complexity and the "ilities." All of the above issues are addressed in the following subsections, with the exception of the "ilities." It is important to note, however, that reliability and failure effects analyses are integral design tasks in flight-critical applications.

Complexity Considerations

There are a number of types of complexity, the characteristics and somewhat arbitrary nature of which admit differing definitions. In most cases, however, it is possible to formulate meaningful, quantitative measures of complexity that can be useful in the development process. Although essentially enumerative in nature, such measures yield useful and repeatable results, so the main concern is to devise complexity measures as appropriate to a particular application. Fortunately, the technical literature offers several promising approaches.[16-18]

Table 2 Complexity metrics summary

METRIC	COMPLEXITY: PATH	STRUCTURAL	LOGICAL
INFORMATION FLOW (REF. 16)			
DATA FLOW FAN-IN/FAN-OUT	•	•	
COUPLING MATRIX			
ENTROPY METRIC (REF. 17)			
HIERARCHICAL RELATIONSHIPS		•	
INFORMATION THEORY			
STRUCTUREDNESS MATRIX			
COUPLING & COHESION		•	•
MATRIX NORMS			
CYCLOMATIC NUMBER (REF. 18)			
LOGICAL FLOW			•
CONTROL GRAPHS			

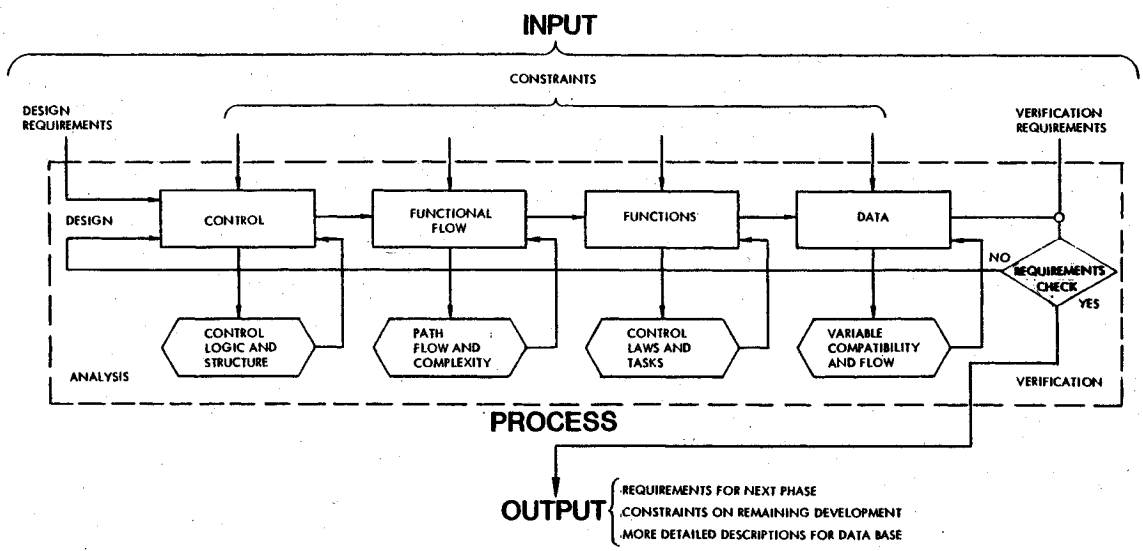

Fig. 3 Task flow diagram of an early development phase.

For clarity, it is appropriate to provide the following definitions:

1) Path complexity on a system level is the extent of the signal flow paths and their interrelationships.

2) Structural complexity is the degree of coupling among the system elements or computer program modules.

3) Logical complexity is the degree of decision logic in a computer program.

4) Psychological complexity is the degree of incomprehensibility exhibited by a system or a computer program or descriptions thereof.[19]

For controlling the complexity of an evolving design, structural complexity is the dominant concern. Therefore, suitable metrics are vital to characterize complexity during the earlier stages of design, for large-scale improvements to structure can then be realized. This can be achieved by using analytical mechanisms to direct design decisions: the first three complexity metrics approaches indicated in Table 2.

The fourth metric in Table 2, also known as the McCabe complexity number, generally applies to an implemented code. Consequently, its design utility is limited, but it remains a useful quality metric. A software verification test case definition is one worthwhile application. All of these four complexity metrics are deemed to be natural and plausible for most flight control practitioners in that these techniques relate well to their typical backgrounds. One example of the impact of a controls concept on a software engineering technique is the structuredness matrix in Fig. 4. In controls practice, strong off-diagonal or coupling terms are usually considered undesirable; such is also the case for the software/system structure.

Here the still developmental structuredness matrix attempts to represent structural complexity in a concise and unified way. Since cohesion and coupling constitute the major aspects of structuredness,[2] it is possible to appraise their combined effect in terms of the dominance of the main diagonal. Qualitatively, the dominance of the cohesion or strength coefficients reflects strong modularity and weakness of the off-diagonal terms indicates lesser interaction among modules. Unfortunately, since a good means of measuring cohesion coefficients remains to be found, the task of calibrating and interpreting matrix norms is impeded.

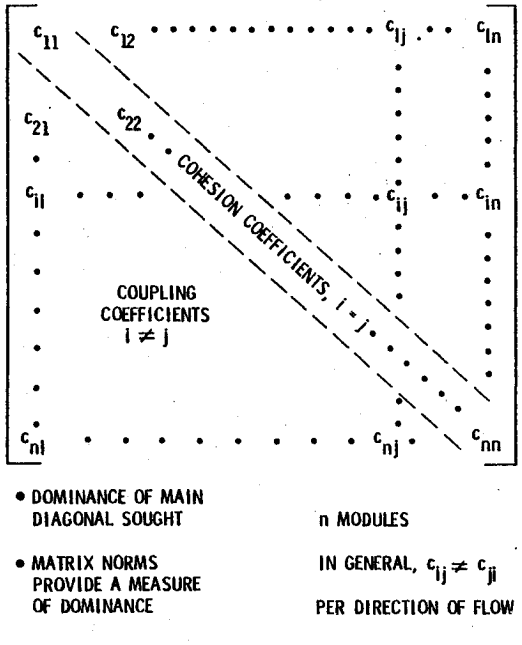

Fig. 4 Structuredness matrix.

Control

Design of the control aspect of the DFCS model begins with the definition of the highest-level system states using an FSM representation of MIL-F-9490D Operational States. The operational state reflects the status of the total system, which is essentially dependent on the failure status of all system components. Issuing from each top-level state is a corresponding FSM whose states reflect status with regard to function engagement and intermediate-level fault assessments. In the authors' experience, two further levels of FSM nesting are appropriate to the detail fault status. The system operational state is, therefore, uniquely defined with regard to component availability by the Boolean variables indicative of the nested FSM states. It is explicitly derived through a consensus among computational channels.

Associated with each FSM is a state table driven by such logic change events such as comparator trips or mode switching. This logic involves a very large number of Boolean variables and expressions, but they are readily obtained in a straightforward manner from the nested FSMs. The functional mode logic is not fully represented in FSM form because the associated fan-in and awkwardness tend to diminish clarity; thus, the lowest levels of the logic expansion are accomplished solely in terms of the Boolean expressions. Combining the high-level operational state information with the low-level mode logic yields the control state. Since different computational channels may be in different functional modes at the same time, the control state may vary among channels. This control state is essentially a static property that activates the system and software structure to yield the desired functional flows.

Functional Flow

Functional flow is a dynamic property reflecting signal path routings, data bus transmissions, software path traversals, etc., in response to the control state. In analytical design, then, the task is to relate specific control states to the resultantly invoked functional operation of the system. In terms of design continuity, this focuses largely on explicit high-level control states in the software architecture. This is not a particularly difficult task, but if neglected, results in an irrevocable breakdown in the analytical design strategy.

On the other hand, the design of the cross-channel control, is rather challenging. It is basically a control task, but it cannot be undertaken until some of the functional flow design is accomplished. This type of dependency underlies the need for the design loop in Fig. 3. The channel coordination problem is complicated by timing constraints and hardware/software interactions that must be modeled analytically.

Function

As previously stated, analytical design of the function or control laws is well established in practice, and the only concern here is to assure that the functions are properly evoked and are serviced with regard to the data input/output.

Data

In the design of the system/software structure, the main concern over data relates to interfaces among the hardware and/or software modules. There are other considerations such as scaling, refresh rates, and data types, but these are not strongly related to the structure. However, all are of vital concern in terms of verification, as will be amplified later.

The amount of data exchanged by the modules impacts the structural complexity, as previously discussed. In current flight control computers, such transfer of data is somewhat unconventional because of the nature of the extensively used memory-mapped input/output. There results a profusion of global variables, which is generally undesirable, but this is tolerated to off-load the central processor. The associated data flow is still reflected by the complexity of the coupling

between the various software modules and one of two hardware modules, either a read-only or write-only section of memory. A considerable amount of data is also passed between the computational channels, which must be accounted for as well.

From an analytical design standpoint, the major concern is merely keeping track of the large number of variables in each of the computational channels. Regarding software structure *per se,* the handling of the Boolean variables is an especially important concern. The associated leverage to alleviate complexity constitutes one of the central motivations and determinants of the methodology.

Design Description

Figure 5 depicts the system description process over the life cycle of the DFCS; this is an elaboration on a very apt prior formulation.[20] Basically, each phase of the life cycle affords a different perspective to the system description. Accordingly, each phase contributes a unique aspect of the system description to the data base. Additionally, each phase propagates the requirements for the succeeding phase. The circle is re-initiated when circumstances such as operational deficiencies generate redesign requirements. Moreover, this circularity motivates the "spoked wheel" representation.

A typical development process or life cycle entails many loops about the wheel, without necessarily performing all of the indicated activities. A design data base must facilitate and reflect the associated system modifications. Additionally, the design data base must be constituted to alleviate psychological complexity and to facilitate various analyses such as structural complexity measures. Table 3 summarizes the primary design description methods used in the methodology. Certain analytical aspects of these design descriptions are discussed in the next section. The design data base must also include models or descriptions to support the reliability and failure effects analyses.

Table 3 Design description/analysis methods

DESIGN ASPECT	BASIC APPROACH
CONTROL	• NESTED FINITE-STATE MACHINES (FSMs) • BOOLEAN EXPRESSIONS • PETRI NETS
FUNCTIONAL FLOW	• MULTIRATE EXECUTIVE STRUCTURE • CALL GRAPH • CONTROL GRAPH
FUNCTIONS	• INPUT-PROCESS-OUTPUT DIAGRAMS • ANALYTICAL CONTROL DIAGRAMS • NON-PROCEDURAL TEXT
DATA	• DATA ACCOUNTABILITY LIBRARY • DATA FLOW ANALYSIS

Design Verification

Verification of the evolving conceptual design is the central thrust of the subject methodology. The selected design description methods enable and foster this endeavor. Further along in the development process, these description methods support the verification of the implementation consistency with detailed conceptual design. However, the major emphasis is the analytical verification of the abstract design.

As noted in Table 4, each of the four aspects of the DFCS design model is analyzed for consistency, completeness, and reachability. These types of checks, which are in addition to the establishment of compliance with the stated design and performance requirements, form the basis of analytical rigor. Consequently, an elaboration of Table 4 appears in the next four subsections. Note that the tools and methods to ascertain compliance with the criteria are quite varied in nature and origin. Those used, coupled with their timely, coordinated, and specialized application, are considered to yield a unique methodology.

Control Design Verification

FSMs are inherently analyzable with regard to the three particular criteria indicated, so the dominant concern over

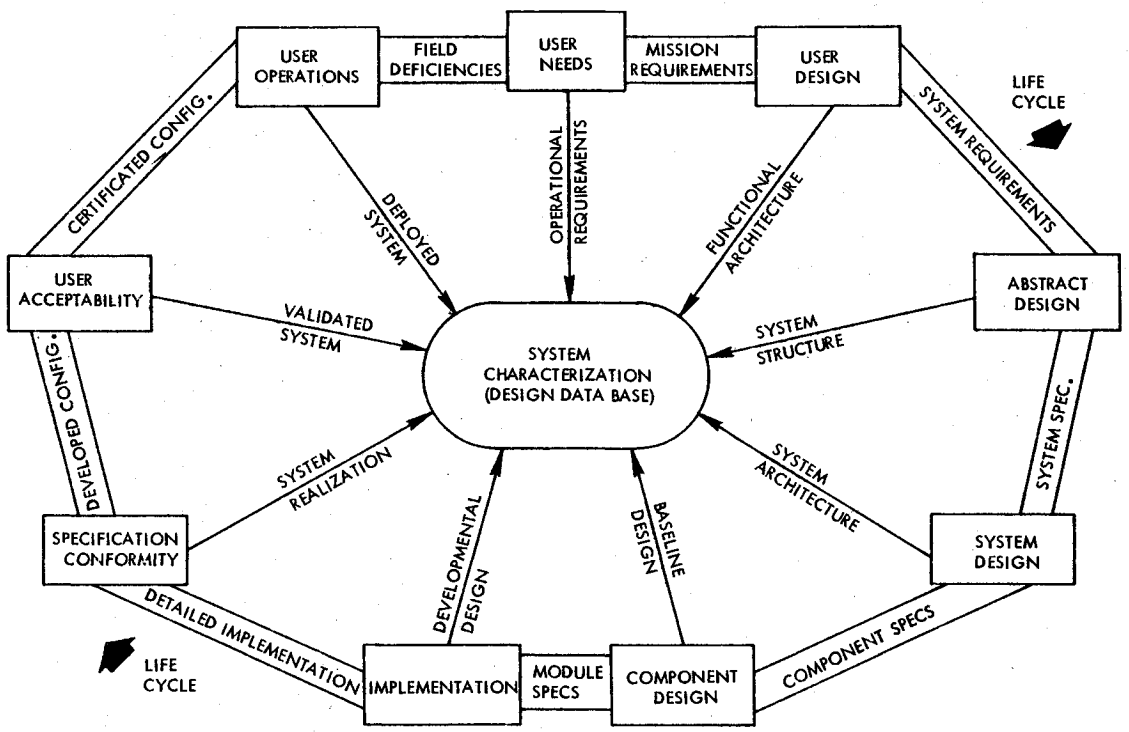

Fig. 5 Life cycle system description.

Table 4 Design verification requirements

DESIGN ASPECT \ ANALYSIS CRITERIA	CONSISTENCY	COMPLETENESS	REACHABILITY	OTHER
CONTROL	CONTROL STATE TRANSITIONS ARE DETERMINISTIC	ALL ADMISSABLE CONTROL STATES AND ASSOCIATED TRANSITIONS ARE COMPLETELY DEFINED	EACH CONTROL STATE IS REACHABLE UNDER APPROPRIATE CONDITIONS	THE SAME CONTROL STATE DEFINITIONS AND LOGIC VARIABLE NOMENCLATURE ARE USED IN ALL DOCUMENTATION
FUNCTIONAL FLOW	FUNCTIONAL FLOW PROVIDES THE PROPER SEQUENCE OF FLOW IN ALL CASES	AT LEAST ONE FUNCTIONAL FLOW INVOKES EACH FUNCTION	A DISTINCT FUNCTIONAL FLOW EXISTS FOR EACH CONTROL STATE	CORRECT RECONFIGURATION, IF REQUIRED, OCCURS FOLLOWING ALL FAILURE EVENTS
FUNCTIONS	FUNCTIONS ARE COMPATIBLE WITH DATA TYPES AND SCALING	FUNCTIONS PROPERLY MAP THE SET OF INPUT VARIABLES ONTO THE APPROPRIATE OUTPUTS	EACH FUNCTION IS INVOKED UNDER CERTAIN CONDITIONS	FUNCTIONAL PERFORMANCE IS ACCEPTABLE UNDER ALL ADMISSABLE OPERATIONAL CONDITIONS
DATA	DATA FLOW IS IN ACCORD WITH FUNCTION INPUT AND OUTPUT NEEDS	ALL INTERFACES ARE COMPLETELY DEFINED AND ALL VARIABLES ARE DEFINED AND INITIALIZED	ALL VARIABLES ARE USED UNDER APPROPRIATE CONDITIONS	THE SAME DATA FLOW/USE OCCURS IN THE OPERATIONAL SYSTEM AS IN THE DOCUMENTATION

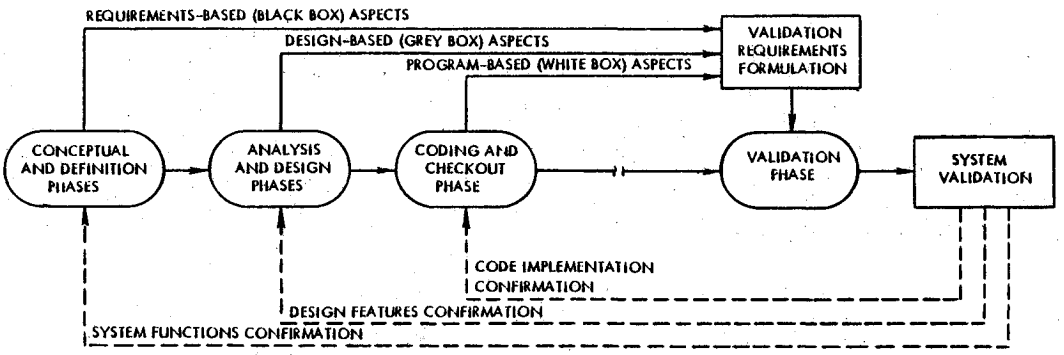

Fig. 6 System validation strategy.

system operational state can be addressed rigorously. In fact, the design of the complete system control state logic can readily be verified analytically. It is much more difficult to confirm the correctness of the cross-channel coordination logic. Petri nets are used for this purpose, but they are somewhat awkward to compose. The approach recommended is to synthesize using control flow diagrams, and then to translate to the Petri net form. This is still a nontrivial task for a full-scale, fault-tolerant system, but the debugging and verification benefits are worth the effort.

Functional Flow Design Verification

The functional flow maps the control states onto the functions, as Table 4 clearly indicates. Basically, the verification task is to confirm the proper evocation and ordering of all functions, without omissions or redundancies. This is straightforward analytically and identifiable in flow diagrams, hierarchy charts, and control graphs. Test case definition for the correct correspondence between the control state and the software path traversal derives from this analysis. This constitutes an important case where the implementation is checked for consistency with the verified design.

Function Design Verification

The design of functions is primarily confirmed analytically by using established methods commonly employed in practice (e.g., examining closed-loop eigenvalues). A major concern here is then to insure that the control law module activation and interfaces are correct within the system/software structure. This is accomplished in part by the functional flow and data verification. There is even greater concern regarding those functions that effect the control state changes, especially those that identify conditions warranting mode or resource changes. Of greatest concern are those functions such as self-test that must detect and isolate discrepant operation in a safety-critical DFCS. The establishment of dependable means to attain the high-level assurance demanded for such functions remains a continuing design verification technology issue.

Data Design Verification

Historically, module interface discrepancies have been manifested far too frequently, which has motivated considerable emphasis on module interface languages and associated static analysis tools.[21] Programming languages such as Ada[R] and Modula promise to rectify the subject problems, and in doing so they basically accomplish the data verification tasks indicated in Table 4. In the meantime, such checks must be done by specially provided tools that are tailored to the design data base description formats.[22] This is largely a bookkeeping task, but it is vital in assuring data consistency and completeness.

System Validation

Following verification of a detailed conceptual design, an implementation must be realized, and its constituent elements must themselves be ascertained to fulfill component specification requirements. Software verification is a foremost example of the latter task. Then comes system integration and validation, which for embedded systems such as DFCSs, are quite challenging because user expectations and nonideal influences must be confronted.

In the subject methodology, concern with the conclusiveness of system validation is the overriding issue, especially as it is required to substantiate compliance with the quantitative safety requirements for critical DFCSs. Although far more than validation testing is involved, testing remains a key

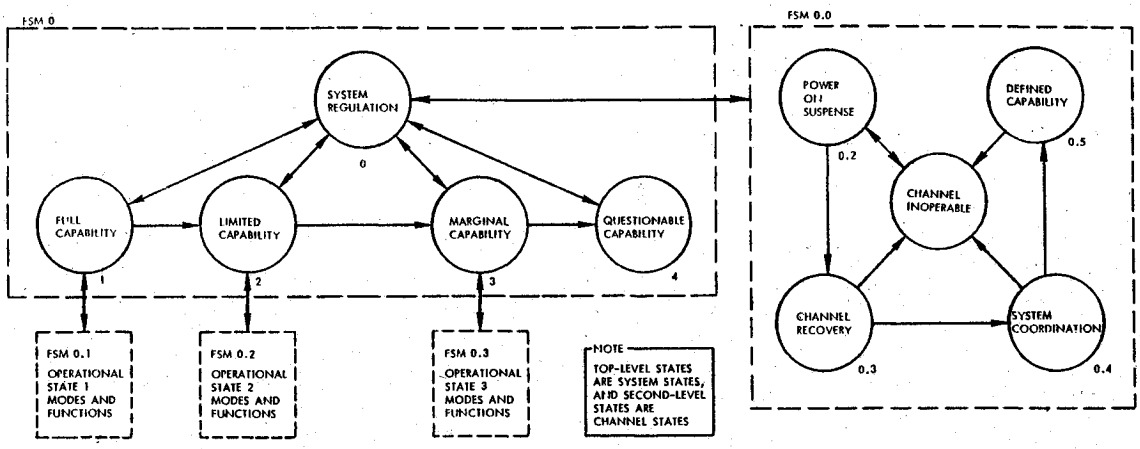

Fig. 7 Finite-state machine representation of higher-level system states.

means of demonstrating compliance because it is concrete and tangible.[2,3] To mitigate the difficulties inherent in attempting "complete" testing,[24] the strategy of the subject methodology is based on the reduced complexity to yield fewer test cases, progressive requirements definition for thoroughness, multilevel testing and instrumentation for conclusiveness, and requirements and design description as test execution monitors for automated accountability.

As indicated in Fig. 6, progressive definition of validation requirements leads naturally to multilevel testing. The really challenging issue then becomes lower-level hardware and software instrumentation. Here the need for observability[25] is compelling because test case design and interpretation can be greatly facilitated by the availability of ample, selected information. Also, the use of design descriptions as test execution monitors to interpret the results yields verified documentation as a by-product.

Methodology Application Results

In Fig. 7, the system operational states are shown in the top-level FSM. Only the system regulation state is expanded out into a nested FSM that defines explicit top-level states within each channel. The system operational state is based in part upon a consensus of these channel states. The design of FSM 0.0 is depicted for a single channel in Fig. 8. This was replicated in each of four channels in the trial application of the methodology in which a quadruplex pitch stability augmentation system was implemented in a system simulator.

Analytical verification of the channel coordination features needed to effectuate the system consensus is achieved through translating the flow chart in Fig. 8 into a quadruplex Petri net format. Automated analysis is then performed to check for deadlocks, ambiguities, inadvertant loops, and unreachable states. Typically, the implementation of this design, once debugged and verified, involves both hardware and software.

The major part of the flight software is concentrated in the foreground applications program, which is elaborated in Fig. 9. This figure concisely illustrates three of the four aspects of the DFCS design model in Fig. 2. Functional flow is depicted by the multirate executive flow chart that insures that the control law functions are selectively invoked at proper intervals. The call graph or hierarchy chart reflects an intermediate-level control structure. Segments of this information are repeated in the module definition listing. However, of primary interest here is the description of the module data interface.

The above information resides in the design data base and certain aspects can be used to characterize structural complexity. For example, the information coupling to and from each pair of modules in the call graph can be extracted from the module definition. Later, both of these design descriptions

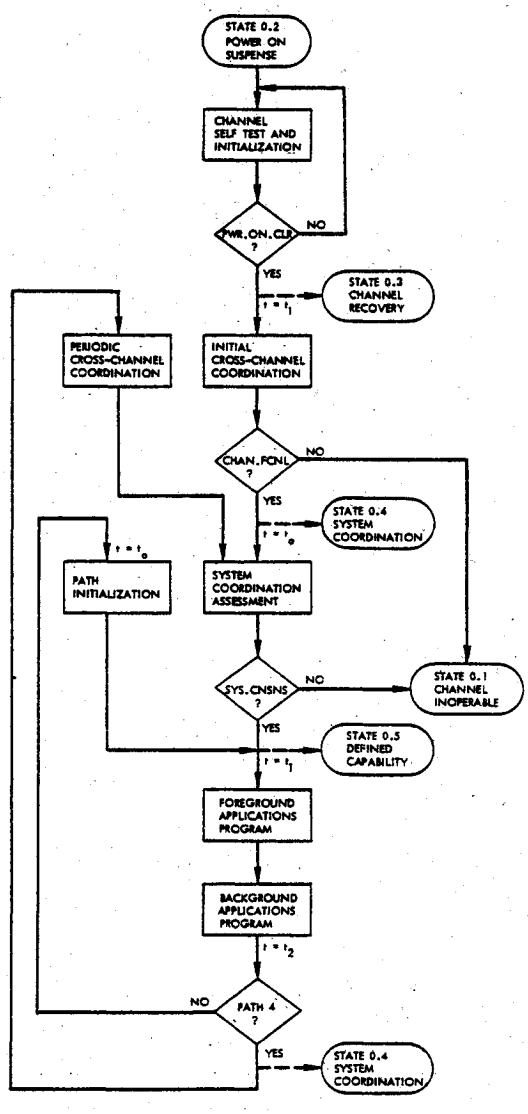

Fig. 8 Simplified flow chart for FSM 0.0.

can be used to ascertain that the actual implementation is in accord with the detailed conceptual design.

As illustrated in Ref. 3, the application of the methodology proceeded well in a limited-scale DFCS development. Application to a complete multiaxis DFCS and calibration of complexity reductions achievable using the methodology are now in progress.

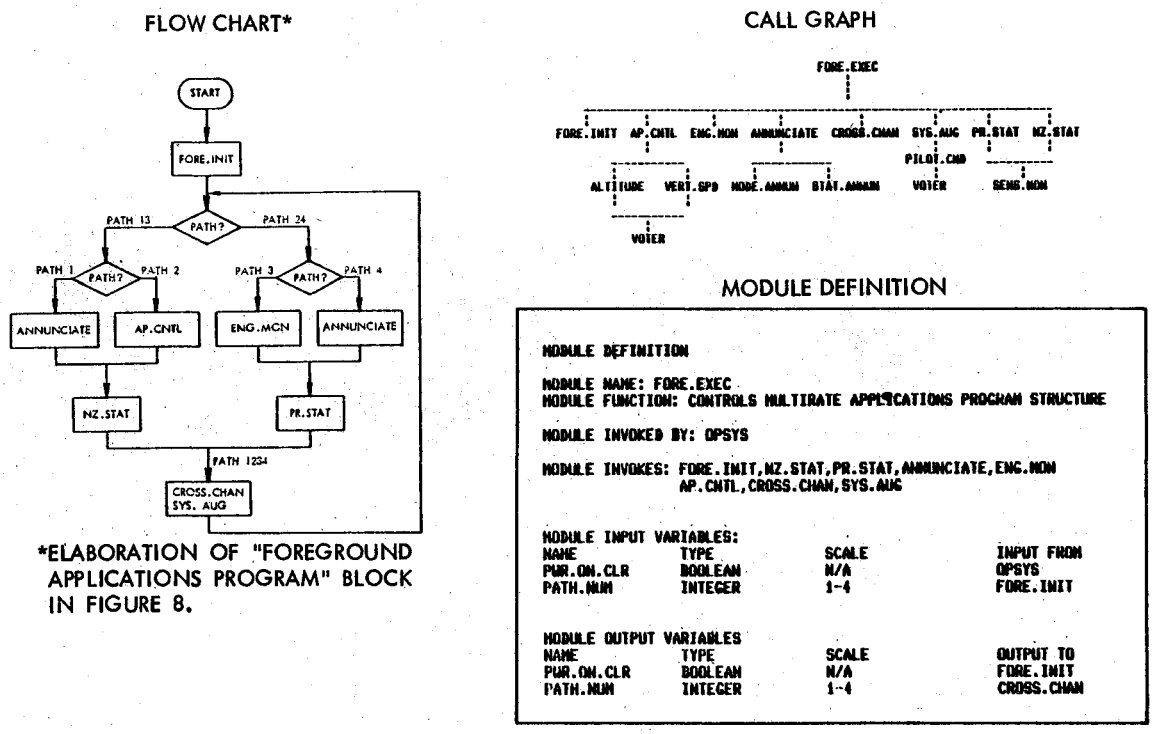

Fig. 9 Foreground executive program design description.

Conclusions

Results derived from the described methodology definition project, coupled with practitioner perspectives regarding actual DFCS development, project the following:

1) Anaytical design/verification of detailed system/software structure will eventually be incorporated into general practice.

2) Such practice will subordinate the early partitioning of hardware and software to a delineation of structure and function.

3) Such practice will be based on methodologies quite similar to the one described here because of its suitability for DFCS applications and practitioners.

4) Inherent flexibility in the described methodology will be exploited in continued extentions or refinements of: improved measures of structuredness; Ada[R] for specification, design, and interface verification; and formal specification and verification methods.

References

[1] Bergland, G. D., "A Guided Tour of Program Design Methodologies," *Computer*, Oct. 1981.

[2] Yourdan, E. and Constantine, L. L., *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*, Prentice-Hall, Englewod Cliffs, N.J. 1979.

[3] Davis, R. M., Mulcare, D. B., and Ness, W. G., "Validation-Oriented Development of a Quadruplex Digital Flight Control Systems," NAECON, May 1983.

[4] Hoare, C. A. R., "The Emperor's Old Clothes," 1980 ACM Turing Award Lecture, *Communications of the ACM*, Feb. 1981.

[5] Jones, C., "A Survey of Programming Design and Specification Techniques," *Specifications of Reliable Software Proceedings*. IEEE Computer Society, 1979.

[6] Brooks, F. B., *The Mythical Man-Month*, Addison-Wesley, Reading, Mass., 1975.

[7] Goldberg, J., "General Concepts of Validation," *Validation Methods for Fault-Tolerant Avionics and Control Systems*, Working Group Meeting No. 1, NASA Langley Research Center, Hampton, Va., March 1979.

[8] Browne, N. C. and Shaw, M., "Towards a Scientific Basis for Software Evaluation," *Software Metrics*, edited by A. Perlis et al., MIT Press, Cambridge Mass., 1981.

[9] Roman, G., Letters to the Editor, *Computer*, July 1982.

[10] Zelkowitz, M. V., "Perspectives on Software Engineering," *Computing Surveys*, June 1978.

[11] Riddle, W. E., "Software Tools—Are We on the Right Track?," Paper presented at 4th International Conference on Software Engineering, Sept. 1979.

[12] Range, E.R., "The Use of Finite-State Machines for Describing and Validating Flight Control Systems," NAECON, May 1980.

[13] Salter, K. G., "A Methodology for Decomposing System Requirements into Data Processing Requirements," Paper presented at 2nd International Conference on Software Engineering, Oct. 1976.

[14] "Flight Control Systems—Design, Installation and Test of Piloted Aircraft, General Specification for," MIL-F-9490D, USAF, June 1975.

[15] "System Design Analysis," U.S. Department of Transportation, Federal Aviation Administration Advisory Circular 25-1309-1, Sept. 7, 1982.

[16] Henry, S. and Kafuras, D., "Software Structure Metrics Based on Information Flow," *IEEE Transactions on Software Engineering*, Sept., 1981.

[17] Mohanty, S. N., "Models and Measurements for Quality Assessment of Software," *Computing Surveys*, Sept. 1979.

[18] McCabe, T. J., "A Complexity Measure," *IEEE Transactions on Software Engineering*, Dec. 1976.

[19] Shneiderman, B., *Software Psychology*, Winthrop, 1980.

[20] Callender, E. D. et. al., "Relations Between Information System Engineering and Software Engineering," *AIAA Computers in Aerospace III Conference*, AIAA, New York, Oct., 1981.

[21] Tichy, W. F., "Software Development Control Based on Module Interconnection," Paper presented at 4th International Conference on Software Engineering, Sept. 1979.

[22] Briggs, P., and H. McGlynn: "Evaluation of Front End Software Development Tools," AIAA/IEEE 4th Digital Avionics Systems Conference, Nov. 1981.

[23] Mulcare, D. B., Ness, W. G., and Davis, R. M., "Digital Flight Control System Validations Technology Assessment," DOT/FAA/CT-82/140 or NASA CR 166374, July, 1982.

[24] Goodenough, J. B., and S. L. Gerhart: "Toward a Theory of Test Data Selection," *IEEE Transactions on Software Engineering*, June 1975.

[25] Weinberg, G. M.: *An Introduction to General System Thinking*, Wiley-Interscience, New York, 1975.