NASA Contractor Report 181835

Design for Validation: An Approach to Systems Validation

William C. Carter, Chairman
Janet R. Dunham, Vice-chairman

Jean-Claude Laprie, Thomas Williams, William E. Howden, Brian Smith

Carl M. Lewis, Editor

Center for Digital Systems Research
Research Triangle Institute
Research Triangle Park, NC 27709

Contract NAS1-17964

May 1989

National Aeronautics and Space Administration

Langley Research Center
Hampton, Virginia 23665

DESIGN FOR VALIDATION: AN APPROACH TO SYSTEMS VALIDATION

William C. Carter, Chairman
Janet R. Dunham, Vice-chairman

Jean-Claude Laprie
Thomas Williams
William Howden
Brian Smith

Carl M. Lewis, Editor

Center for Digital Systems Research
Research Triangle Institute
Research Triangle Park, North Carolina 27709

Contract NAS1-17964
Task Assignment No. 7

May 1989

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

Every complex system built is validated in some manner. Computer validation began with one person reviewing the design of the system. As systems became too complicated for one person to review, validation began to rely on the application of *ad hoc* methods by many individuals. As the cost of changes mounted and the expense of failures increased, more organized procedures became essential. Attempts at devising and carrying out those procedures showed that validation is indeed a difficult technical problem.

The successful transformation of the validation process into a systematic series of formally sound, integrated steps is necessary if the liability inherent in future digital-system-based avionic and space systems is to be minimized. This report presents a suggested framework and timetable for that transformation.

In sections two through four, we provide basic working definitions of two pivotal ideas — validation and system life-cycle, and show how the two concepts interact. Many examples are given of past and present validation activities by NASA and others. In section five, we present a conceptual framework for the validation process. Finally, in sections six and seven, we list important areas for ongoing development of the validation process at NASA Langley Research Center (NASA-LaRC).

2. DESIGN FOR VALIDATION

The process which we refer to as *design for validation* is the panel's vision of future systems validation. We define the process as follows:

- *Validation* is the process of using quantitative theory and measurements to provide *enough confidence* in a system's capability to deliver its specified service [1].

  The definition of *enough confidence* will vary with the type of system, its complexity, its mission, its repairability, its intended life-cycle, its maintenance protocols, its capability for self test, its failure modes, its testability, and other factors. A major theme of this report is that the very high levels of confidence required of NASA's mission-critical and life-critical systems cannot be provided only by system tests, no matter how extensive, but must be designed in sound design disciplines and thorough design validation activities.

- *Design for validation* is an integrated approach to realizing verifiably correct systems through the application of validation principles.

  The validation principles enumerated below, and discussed in more detail within the body of this report, are derived from existing commercial practice and from current research in computer science.

- *Design for validation* begins with the system's conception and continues through the entire system life-cycle.

  Design for validation must be an integral part of the systems research and development process.

- *Design for validation* is based upon the use of automated tools.

  Automated tools will provide information capture and the analytical power required for conducting systematic validation activities which are fully integrated with each other and with the design process.

- *Design for validation* must play a central role in the planning and construction of future crucial systems, including flight controls, nuclear power systems, and strategic defense systems.

  Design for validation must convincingly demonstrate that these systems have attained the required level of dependability, as defined in the system specifications. Dependability may be expressed, depending on the particular system, in terms of minimum uptime, maximum downtime, throughput under load, mode of degradation, data error rates, or other parameters.

- *Design for validation* must be quantifiable and provide accountability.

  If it is not quantifiable, then results cannot be measured and improvements cannot be proven. If it does not provide accountability, then design decisions may be made casually and without adequate review or documentation. When system changes occur, their consistency with the previous system must be shown.

- *Design for validation* must be implemented hierarchically from the most abstract conception levels down through system operation.

  Design for validation must be systematic, comprehensive, and consistent within and between design levels. If any deviation from intended behavior occurs, its origin must be able to be determined by tracing the system design process, beginning with the requirements. This method of hierarchical design has been widely explored and is currently in limited use for a few levels. Unfortunately, a comprehensive theory of design for the validation of complete systems does not yet exist.

- *Design for validation* should use information gained from previously validated systems. This information can be used only if it has been incorporated into a validation knowledge base.

- *Design for validation* must be applicable to future systems.

  Advances in system engineering technology will continue to yield more powerful and widely distributed digital systems. The validation process must be capable of extension; that is, it must be applicable to all digital systems, irrespective of size or application.

3. DIMENSIONS OF THE VALIDATION PROCESS

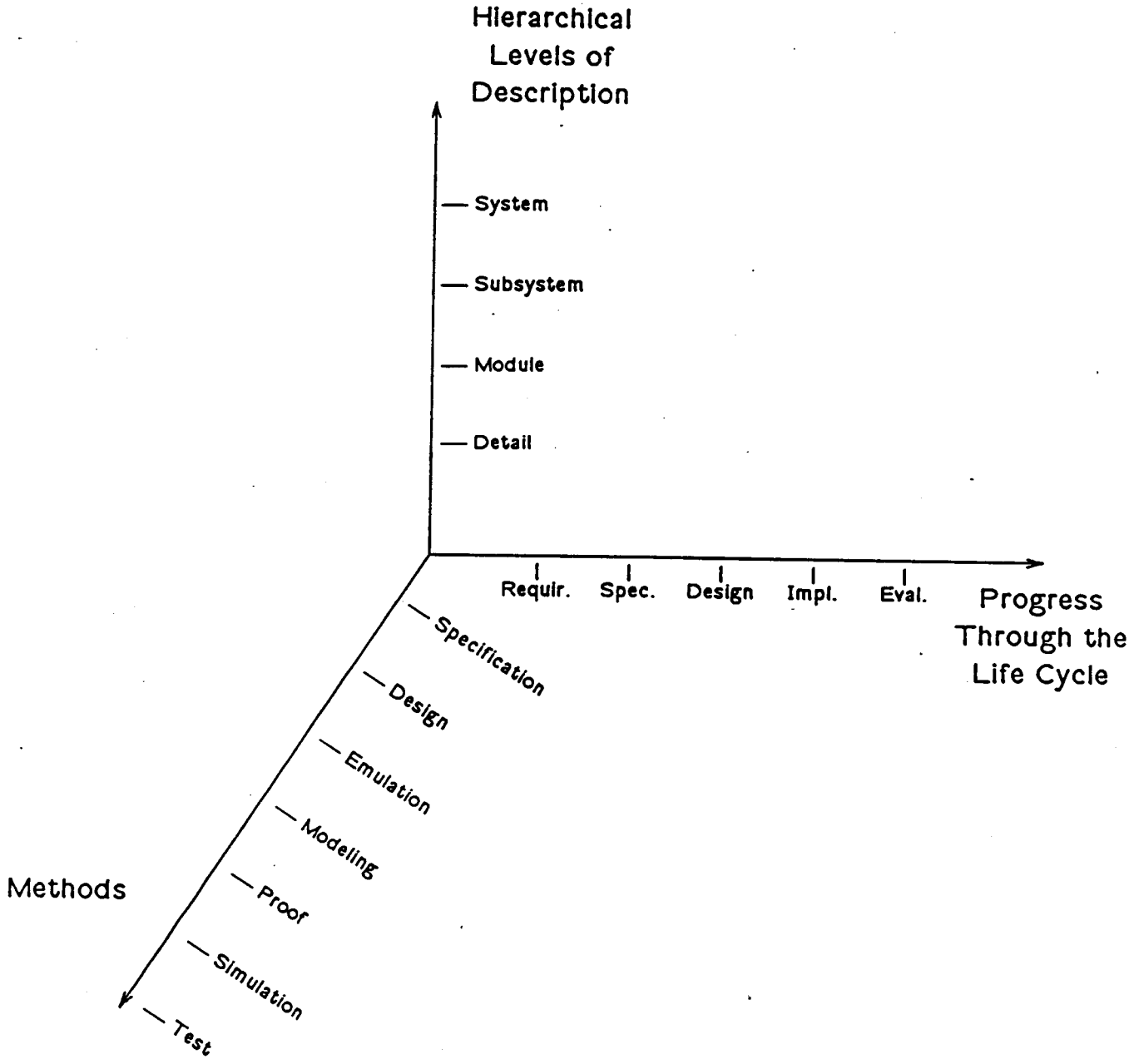


Figure 1. The Validation Space

There are three important dimensions to validation. They can be conceptualized as forming a validation space. The dimensions are *METHODS, PROGRESS,* and hierarchical levels of *DESCRIPTION,* as shown in Figure 1.

The *METHODS* dimension contains the full complement of present and future *design for validation* activities. This dimension includes specification, design, emulation, modeling, proof, simulation, and test.

PROGRESS through the life-cycle is seen as encompassing *design for validation* activities conducted at different stages in the system life-cycle. The stages of progress are requirements, specification, design, implementation, and evaluation. Iteration occurs within and between stages in the life-cycle. Progress is made when the need to revisit a previous stage is eliminated. These stages must be planned and information about them must be traced and audited throughout the life-cycle.

Hierarchical levels of *DESCRIPTION* refer to the levels of abstraction of the system and its components. These levels are determined by constructing a hierarchical set of axiomatic and empirical models which encapsulate various views of the system. These models are used to measure the achievement of relevant system requirements. The ability to abstract is important to validation methods, irrespective of size or application.

This validation space can be viewed as containing several subspaces which are prevalidation, the validation during the initial part of the design cycle; primary validation, the important validation which is done during design before the product is shipped; and post validation, which is the validation and monitoring done in the field in order to ensure the growth of not only dependability but of our belief in the validation and assurance that the system will really do the job it is designed to do.

As any system is designed, its program forms a trace, or a walk, through this validation space. In the beginning, consider the subsystem which consists of the requirements, the specification methods necessary for the validation, and the abstract specification of the system. The next step that the walk goes through will be determined by the abstraction of the specification levels. This process of determining a walk can be continued until the trace of the entire project effort is formed.

3.1. PRE-VALIDATION

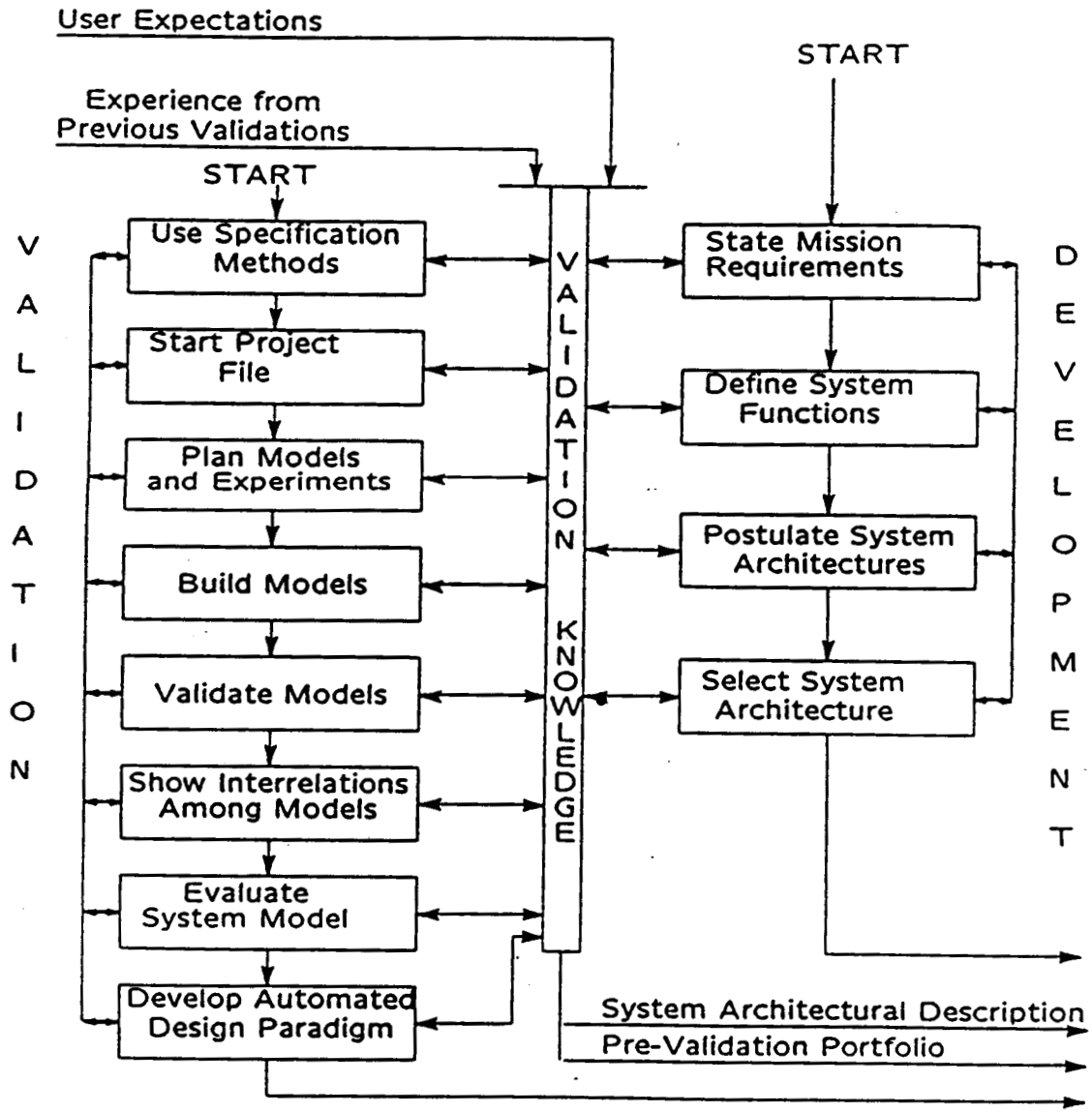


Figure 2. Pre-validation Activities

The left half of Figure 2 depicts the pre-validation stage of the system life-cycle. The right half depicts the development activities conducted which track the system from user expectations to a dependable system architecture. Experience from previous post-validation efforts are used as input to this stage.

Validation methods begin with specifying the system. Specification methods are formal methods, including language and notation, that describe or represent the behavior of a system or subsystem in a hierarchical fashion. Hierarchy is a critical element of successful design and validation.

Specification methods are used to state the mission requirements and define the system functions which meet those requirements. The critical process of defining mission requirements should, of necessity, be an iterative process involving both users and designers. As the system is specified, a project file which captures the salient design and validation decisions is constructed. This file provides traceability for the system being developed and, once analyzed, contributes to the validation data base.

The next pre-validation activities involve planning the analytical and empirical models to be developed and the validation experiments to be conducted. This activity, although performed at the pre-validation stage, pervades the validation and post-validation stages through the refinement of the models and the distillation of the experimental outcomes. This planning activity may well require the development of new validation techniques to demonstrate that the system is meeting special system requirements.

The different models pertain to parts of the system, potential system architectures, and mission requirements. The interrelations among these models are demonstrated and evaluated. This evaluation process results in the selection of a system architecture.

Once the architecture is selected, the initial validation plan is refined. This plan becomes the basis for validation activities during the primary validation and post-validation stages of the system life-cycle.

Output from the pre-validation stage, in the form of validation information and a system architectural description, is passed to the primary validation stage.

3.2. PRIMARY VALIDATION

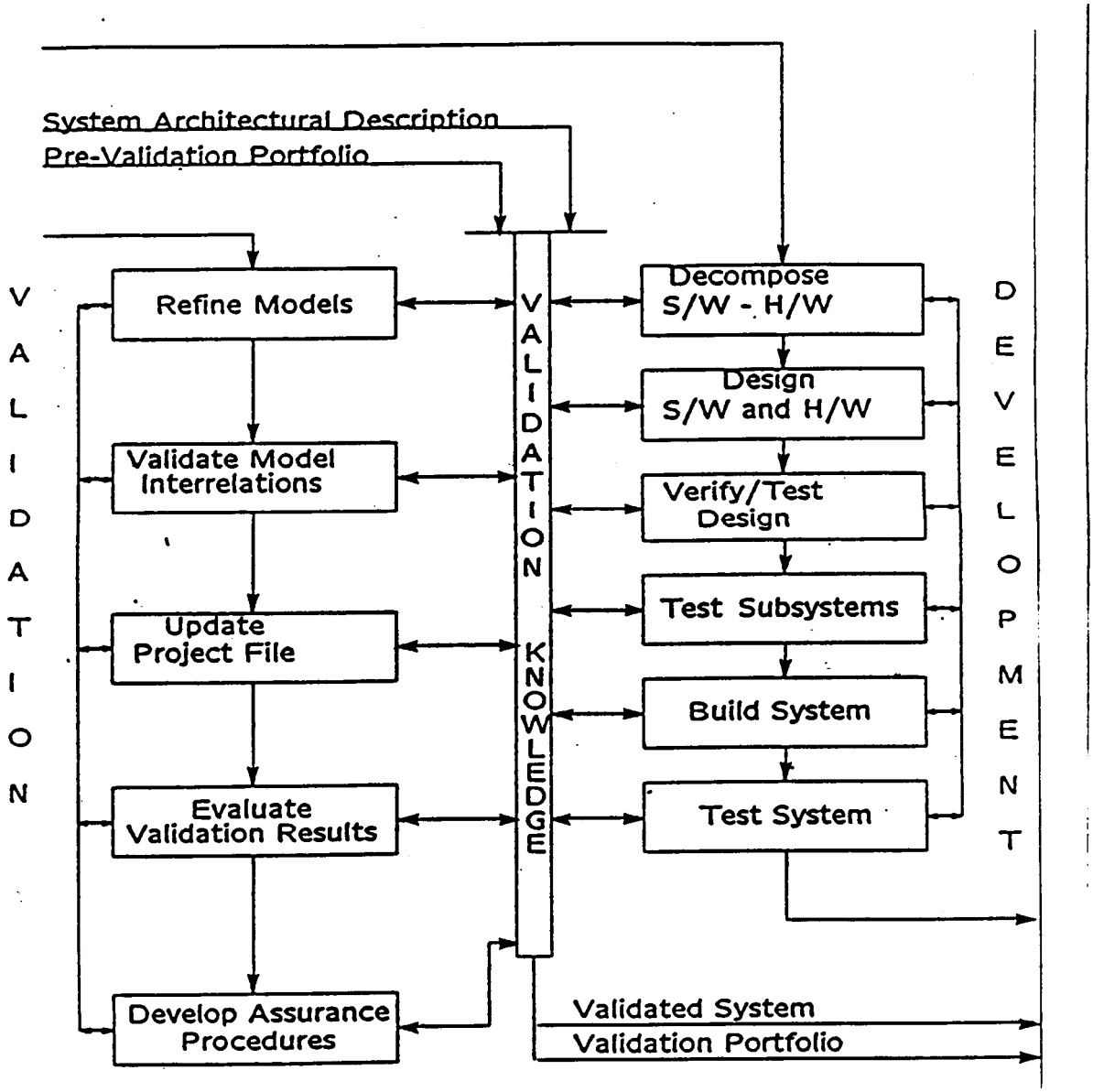


Figure 3. Primary Validation Activities

In the primary design and validation stage (shown in Figure 3), the inputs are the system architectural description and all of the previous work which was done for validation.

First, decompose the system into software and hardware and begin designing the software and the hardware in more detail.

Now there is more information, so the models must be refined. This refinement means that the model interrelationships must be validated again. All interrelationships must be identified, demonstrated to be valid, and evaluated. The principle interrelationships used are the consistency and completeness of the models. Because of the hierarchy, show the consistency and completeness at all levels.

At each level there are different views of the system. They must be reconciled and shown to be consistent. Using many different views improves the completeness of the system validation. The next step is to keep updating the project file and to continue to evaluate the validation results. As the system is being designed, the subsystems must be verified, tested, and the adequacy of their responses noted. The system is then built from the validated subsystems and the final system test completed.

Product assurance meets two needs — first, to ensure that the system will satisfy its requirements and can be shipped to the customer; and second, to develop the procedures required to monitor, maintain, test, and analyze operation of the system during use.

3.3. POST-VALIDATION

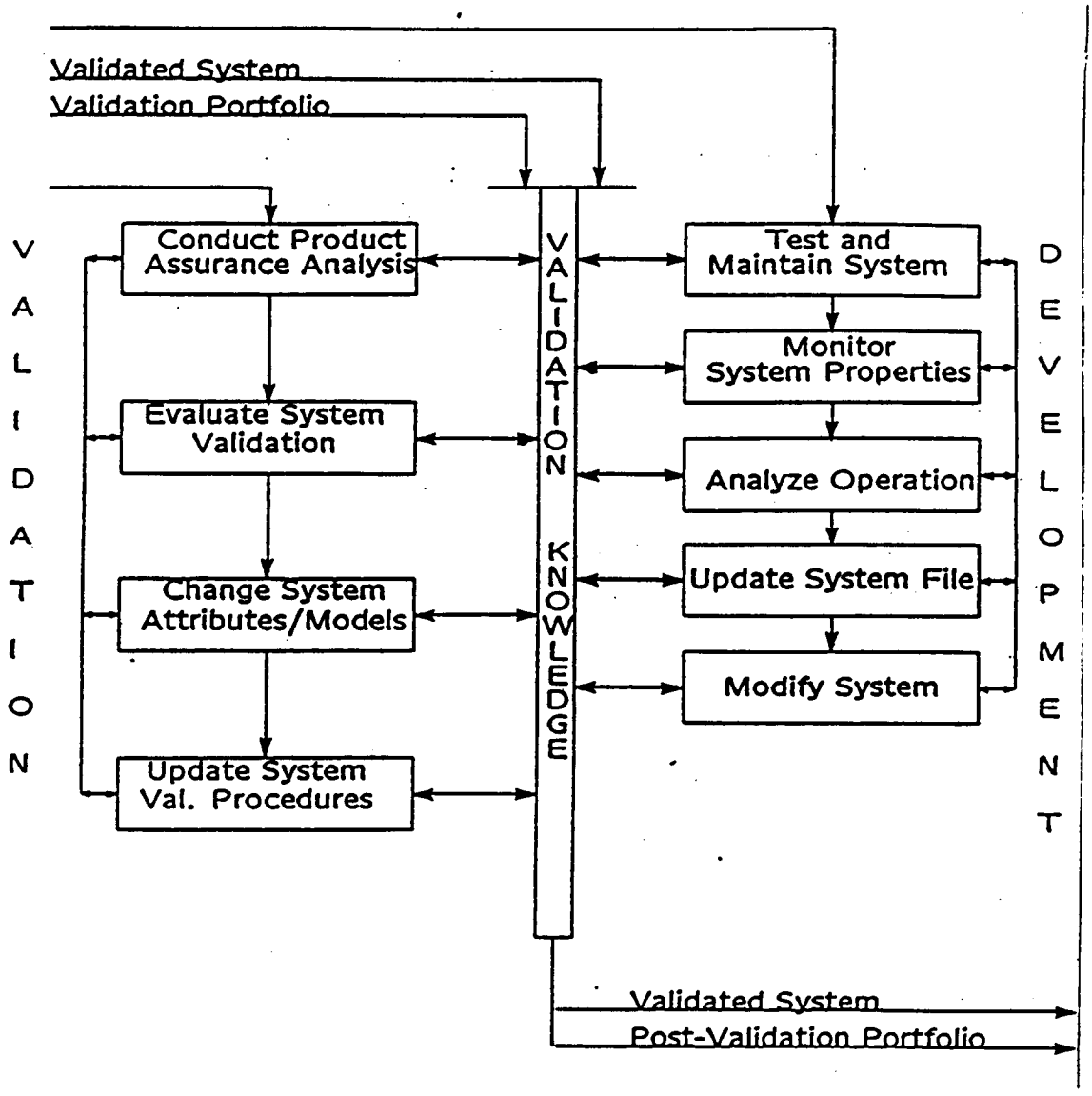


Figure 4. Post-validation Activities

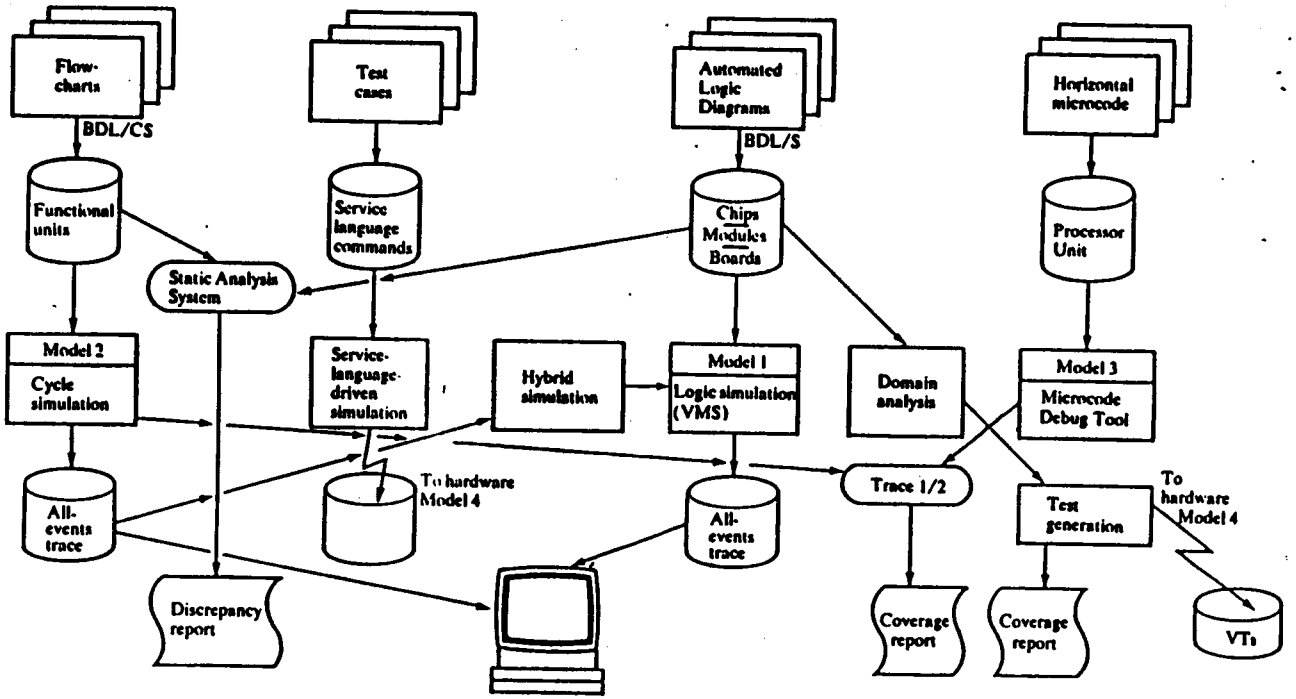
While the system is being operated (see Figure 4), it must be tested and maintained to ensure that its dependability continues. The system properties must be monitored; if system changes are made, the desired product assurance analysis must be redone. The system is re-evaluated to ensure it is operating as desired.

The system properties are also being monitored. If at any time they appear to be unsatisfactory, they must be changed, even though they are not detracting from the system service. As these changes are determined by analyzing the operations, the system file is updated, system models are changed, and the validation is done again to ensure that the system with the proper changes is not only validated, but that its dependability or other attributes are improved by the change.

3.4. CASE STUDIES IN VALIDATION

On the following pages some simple examples of the dramatic intimacy of design and validation are shown.

3.4.1. CASE STUDY ONE



Note: BDL/CS — Basic Design Language for Cycle Simulation
BDL/S — Basic Design Language for Structure
VMS — Variable Mesh Simulator
VTs — Verification Tests

Figure 5. Typical Commercial Design Validation System[2]

The IBM design system[2] (see Figure 5), as used for the 3081 and the 3080X series design and validation, has the following properties:

First, there are multilingual inputs. Each of the inputs to this process uses the language most convenient for it. These inputs are often not in the language used for the initial specifications. For example, the inputs on the left are the flow charts, which are in a language called the Basic Design Language for Cycle Simulation (BDL/CS). This is a moderately high-level design language.

The next set of inputs are the test cases. These inputs use a format which is easiest for test case design and application.

The next inputs are the automated logic diagrams, or ALDs, as they have been called since the days of the IBM Stretch computer in the late 1950's. They are now called the basic design language for simulation. These diagrams show the detailed logic of the complete computer system. Every gate is shown with its logical function, its logic family identification, and its electrical connections.

Finally, the horizontal microcode is shown as an input. These are in the microcode specification language.

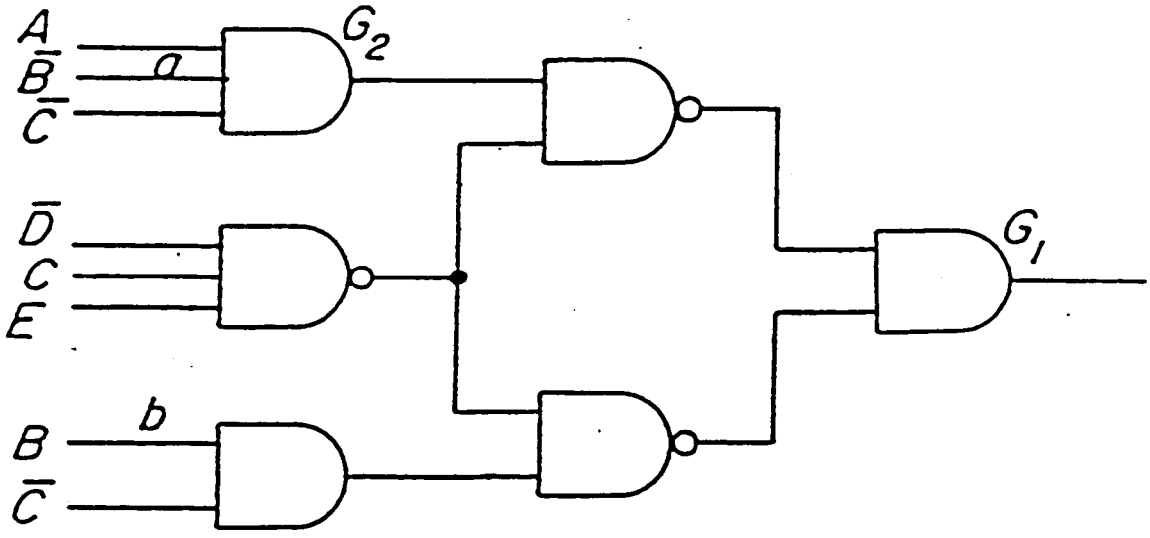
On the next level, several types of simulation and debugging tools are shown. The flow charts describe the functional units and are analyzed by cycle simulation. Cycle simulation does two things — it tests at a fairly high-level using the input test cases, and it also proves that the timing is correct. The automated logic diagrams are also tested by hybrid simulation and by logic simulation using the second model. Finally, the horizontal microcode is tested. Then the information on the "all events" tape is used for automatic comparison of the outputs of the same tests run through these different representations of the same physical system.

The most important box for validation is the box labeled Static Analysis System. Since the basic design language for cycle simulation is a well-formulated language, it can be compiled and its compilation is a set of logic functions whose proper implementation would guarantee that the system would be exactly the one described in the flow charts. These logic functions, after being compiled, are compared with the actual logic functions for the system shown by the ALDs.

Some of these functions were compiled using a different compiler, others were designed manually for maximum throughput. This static analysis system proves that the circuit system implementation agrees with the design language implementation.

The validation of these 500 000 to 600 000 logic circuits, using a formal system description, avoided changes to circuit chips, reduced turnaround time, and saved 66% from the normal system production schedule.

3.4.2. CASE STUDY TWO



The set of tests

$$A\bar{B}CD,\ \overline{ABC},\ A\overline{BCDE},\ BCD,\ AB\overline{CDE}$$

detects any detectable fault in the circuit G_1.
Fault b is detected by $AB\bar{C}$.

Because of redundancy, the fault line a s-a-1 cannot be detected.
If the fault line a s-a-1 exists, then the detectable fault line
b s-a-0 cannot be detected.

Figure 6. Logic Diagram[3]

Many circuits have redundant logical properties, either to fit standard packages or to help eliminate races. Figure 6 contains a simple logic diagram and a set of five tests which will reveal any detectable fault in the circuit. In order to determine the behavior of these circuits, we need the same information which was available during the circuit design. If changes will be made, it is much more efficient to make them during the design process than after the chips have been fabricated.

3.4.3. CASE STUDY THREE

The project FAFTRCS (Full-Authority Fault-Tolerant Reactor Control System) is a project at Argonne National Laboratories to develop a full-authority fault-tolerant digital control for a reactor. The control system monitors the flow using indirect sensors. Due to a desire to construct a deliverable program and the difficulty of setting up a model, implementation of the flow sensor of validation software preceded the development of the system models.

Subsequent efforts to model and formally verify the actual software failed. This is not the only case in which such attempts have failed. Finally, a series of hierarchical models were developed for the operation of the hardware and software and proofs of the fault-tolerance of the generic application programs were devised.

To ensure that the reactor system and its control is properly validated, the software is now being rewritten to be consistent with the high level hierarchical models of the system. This software, with its known structure, will be proven to be valid using these models.

4. CURRENT NASA VALIDATION RESEARCH

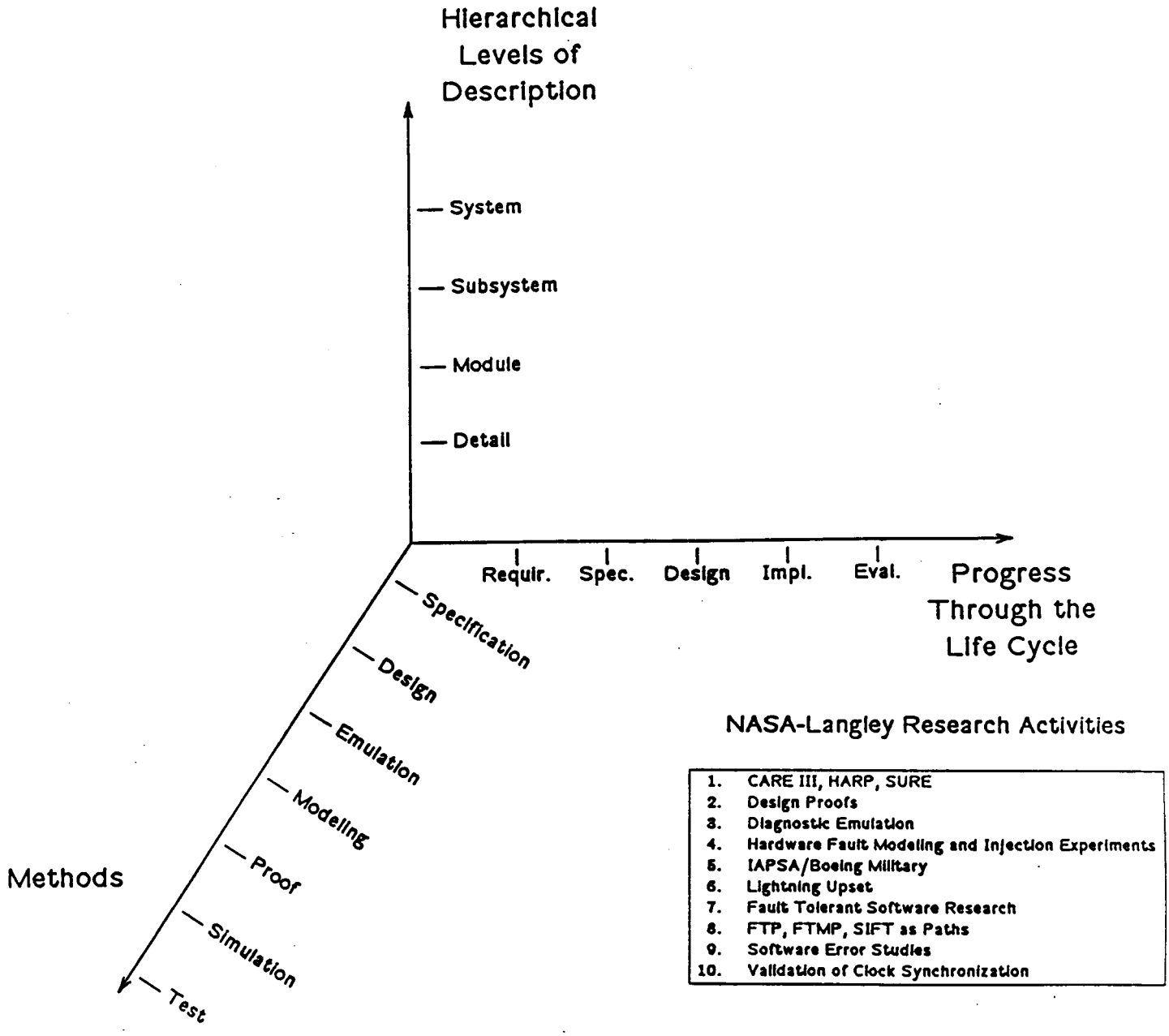


Figure 7. AIRLAB Research in the Validation Space

The current and past research activities in AIRLAB at NASA-LaRC can be viewed as regions in the validation space (see Figure 7). These research and development activities have contributed to the development of methods and to the accumulation of validation knowledge and experience.

4.1. CARE III, HARP, and SURE

The first set of activities to be considered is use of axiomatic models, generally Markov or semi-Markov models, for system reliability evaluation. The first model was CARE III [4] [5], the next was HARP[6] [7], and the most recent model, currently under development, is SURE [8] [9]. CARE III was developed by Raytheon and has been improved and extended by Boeing Computer Services Corporation. HARP embodies new ideas. The most important are the ideas for error handling using a set of Petri nets. Finally, SURE uses more general techniques, semi-Markov processes; it evaluates these quickly. It has been shown that uncertainty in the input parameters, a frequent condition, contributes greatly to uncertainty in any detailed evaluation.

4.2. DESIGN PROOFS

The design proofs[10] were done during the design of SIFT[11]. Six levels were defined and the theory of commutative diagrams relying upon only one language was used for the first five. It was started while the SIFT design was being performed and substantial changes were made in the SIFT physical design in order to produce proofs of consistency between hierarchical levels. The final code level was to use standard program verification techniques. The timing assumptions were verified by a timing analysis program. The language used for all of the first five levels was SPECIAL, which was shown to be complicated and not very user friendly. This language is now being improved.

4.3. DIAGNOSTIC EMULATION

Diagnostic emulation[12] at Langley is one of the examples of analyzing critical properties of gate-level implementations of digital systems, using a hardware model. Such a model supports the primary validation stage and gives information about what should be done during the testing stage. Other models are now being used. The Yorktown Simulation Engine (YSE)[13], which is only one of the types of engines used at IBM, AT&T, and other places, is able to describe up to a million logic functions, so it can easily describe the entire IBM 3081 CPU. YSE then emulates the IBM 3081 instructions at the rate of a thousand instructions per second, which is 3 600 000 instructions per hour. These instructions can be emulated with faults inserted in the system, so that the behavior caused by faults may be analyzed.

4.4. HARDWARE FAULT MODELING AND INJECTION

Hardware fault modeling and injection experiments have been used in many places. They provide research results which are pertinent to the primary validation stage to ensure that reliability predictions are valid.

4.5. IAPSA I AND II

The IAPSA [14] [15] [16] [17] [18] program is presently supporting research in prevalidation activities.

4.6. LIGHTNING-INDUCED TRANSIENT EFFECTS — UPSET

Research provides information and experience about the effects of lightning strikes on fly-by-wire systems. Measurements of lightning effects are extremely important, both during the design stage and for evaluation during the post validation stage.

4.7. FAULT-TOLERANT SOFTWARE

Fault-tolerant software research [19] [20] has been emphasized recently by NASA. The development of the N-version software coincidence errors model yielded much additional insight into the reliability gain of such software or any N-version architecture. This research provides knowledge during the prevalidation stage, which greatly assists in determining which architectures should be chosen.

4.8. FAULT-TOLERANT MULTIPROCESSORS, EXPERIMENTAL SYSTEMS

SIFT[11], FTMP[21], and FTP[22] are complete systems, developed under contract to NASA-LaRC and are now being validated at AIRLAB. Their validation will provide practical knowledge and experience which are applicable to all phases of validation. Validation of the clock synchronization algorithm was done for SIFT. This provided a much better understanding of the interrelationships between the system operational behavior and the abstraction of algorithms which show interactive consistency, often called the Byzantine generals' problem.

4.9. SOFTWARE ERROR STUDIES

Research and experimentation [23] [24] [25] in the measurement of the operational reliability of software provides information about how to post-validate software. The software error experiments, which are still being conducted, provide information about the measurement of the operational reliability of software. These experiments also provide information about how to test software, how to determine that it's ready to ship, and how one is going to most efficiently determine that software should or should not be changed during system operation.

4.10. VALIDATION OF CLOCK SYNCHRONIZATION

The research[26] which culminated in the validation of SIFT's clock synchronization algorithm provided understanding of system operational behavior and thus supports post-validation activities.

5. IMPORTANT IDEAS IN DESIGN FOR VALIDATION

While current validation research has yielded many important results, we considered it profitable to examine some of the important principles embodied in current work to help determine what also needs to be done. The ideas that were discussed fell into five categories:

- Formal Classification of Models
- Design Duality
- Complementary Completeness
- Integration of Design and Validation Tools
- Planned, Traceable, Audited Life Cycle Validation

5.1. FORMAL CLASSIFICATION OF MODELS

The process of designing complex systems is, as it has always been, the implicit or semi-implicit process of creating models of the system. Without classification and unification, it becomes very difficult to determine either that the results achieved are satisfactory or that the reasons for stopping one stage of the modeling and proceeding with the next are met.

A practical example of the importance of models is given by coding theory. There were many codes known before Hamming's work and some of them were extremely complex; however, they were all determined in a hit-or-miss fashion by intuition. How good they were was never explicitly determined except in specific cases. Hamming's achievement was to determine the so-called Hamming measure or Hamming distance, which is based on a mathematical distance property. Using this measure, it was easy to predict and evaluate codes which were designed for error detection or for error correction.

5.1.1. AXIOMATIC MODELS

Axiomatic models satisfy and use axioms from mathematical theory and are based on the application of logic (proof) from general principles to specific conclusions. Examples of axiomatic models are the Markov models of system reliability and the first order logic models of programs.

5.1.2. EMPIRICAL MODELS

Empirical models demonstrate properties based on observations. A system and its properties are normally so complicated that either they do not satisfy a consistent set of mathematical principles, or, if they do, the set is so complicated that it cannot be used to derive further properties.

In these situations, it is easier to construct an empirical model. Examples of this type of model are hardware descriptions, particularly the hardware descriptions used in the simulation of systems.

5.2. DESIGN DUALITY

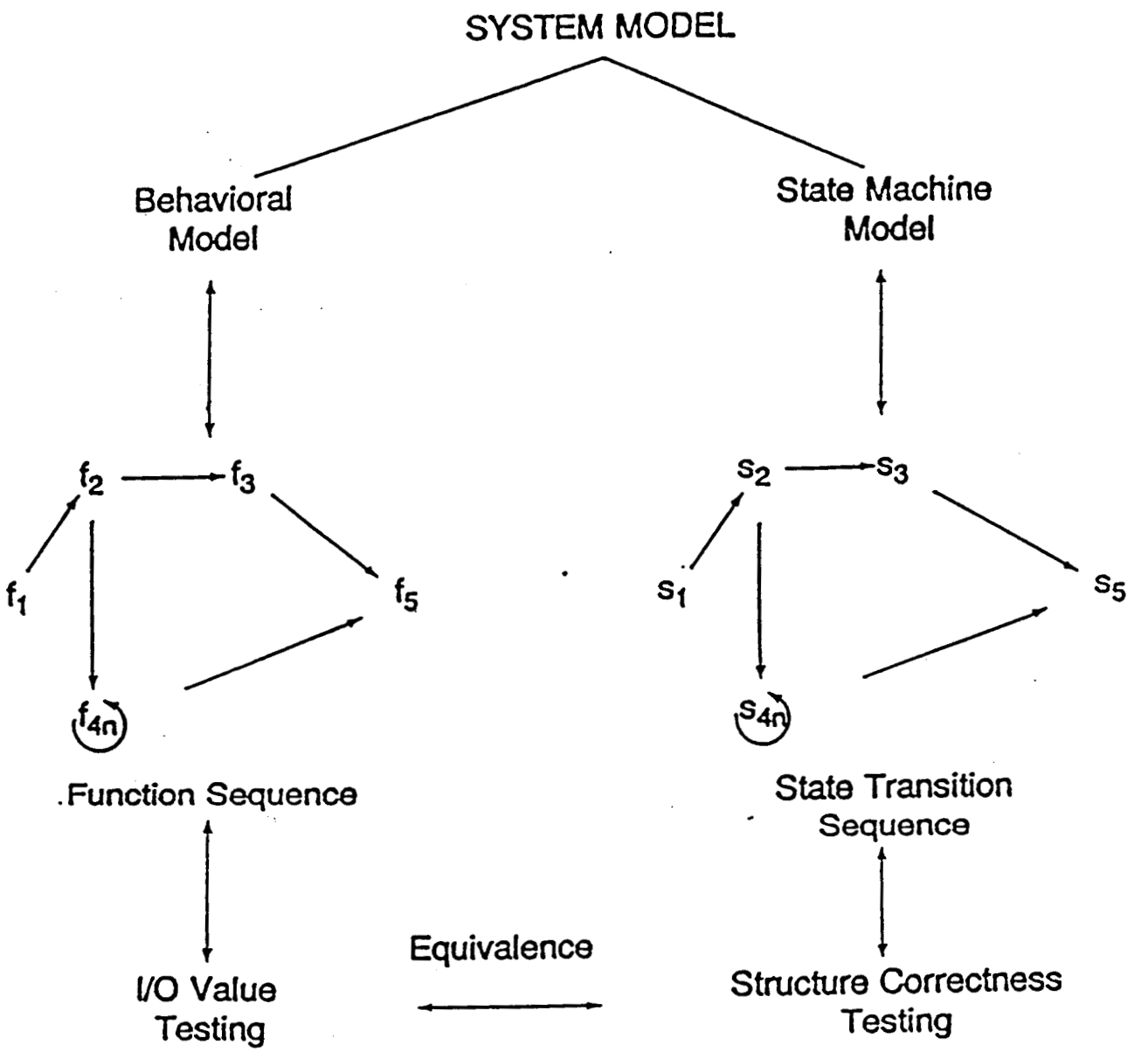


Figure 8. An Example of Design Duality

Behavioral models are frequently used to demonstrate system properties. These models are called black-box models because the only things considered are the functions performed, the inputs, and the outputs. Therefore, they are tested, and their worth is determined, only from sets of their inputs and the resulting outputs.

A second type of model is the object-oriented, or structural model, which emphasizes the manipulation of objects during each step in the model's process. This model is called white box.

Other viewpoints, such as geometric models, showing the details of circuit design, are also used. Using several models for each level is important.

The example of design duality in Figure 8 is derived from one of Howden's papers[27]. Suppose we are given a specification of a system which we wish to model. The system is not yet constructed, so all we have is a set of specifications. This model can be described in several ways.

The first is in terms of a behavioral model. This behavioral model is described as a sequence of functions. Each of these functions operates upon a set of inputs and finally produces a set of output values which can be tested for correctness.

Another model is the state machine model, an object-oriented model. We begin with a state of the system which includes the data. We then determine a set of state transitions; this set is intended to manipulate the data into the various forms that we want. The set is tested by determining that the data structure is transformed correctly and in proper sequence.

Now that we have two tested models of the same system, the next step is to show they are equivalent; for every input pattern for either of the models, both models must give the same output. In other words, while they give two very different views of the same system, they must be shown to be compatible.

Finally, of course, there is the third view of the system, which is the way the system will actually be implemented. This view must also be shown to be equivalent to these models.

The approach of design duality and complementary completeness is to develop a sufficient number of models so that all views of the system have been considered and have been shown to be consistent.

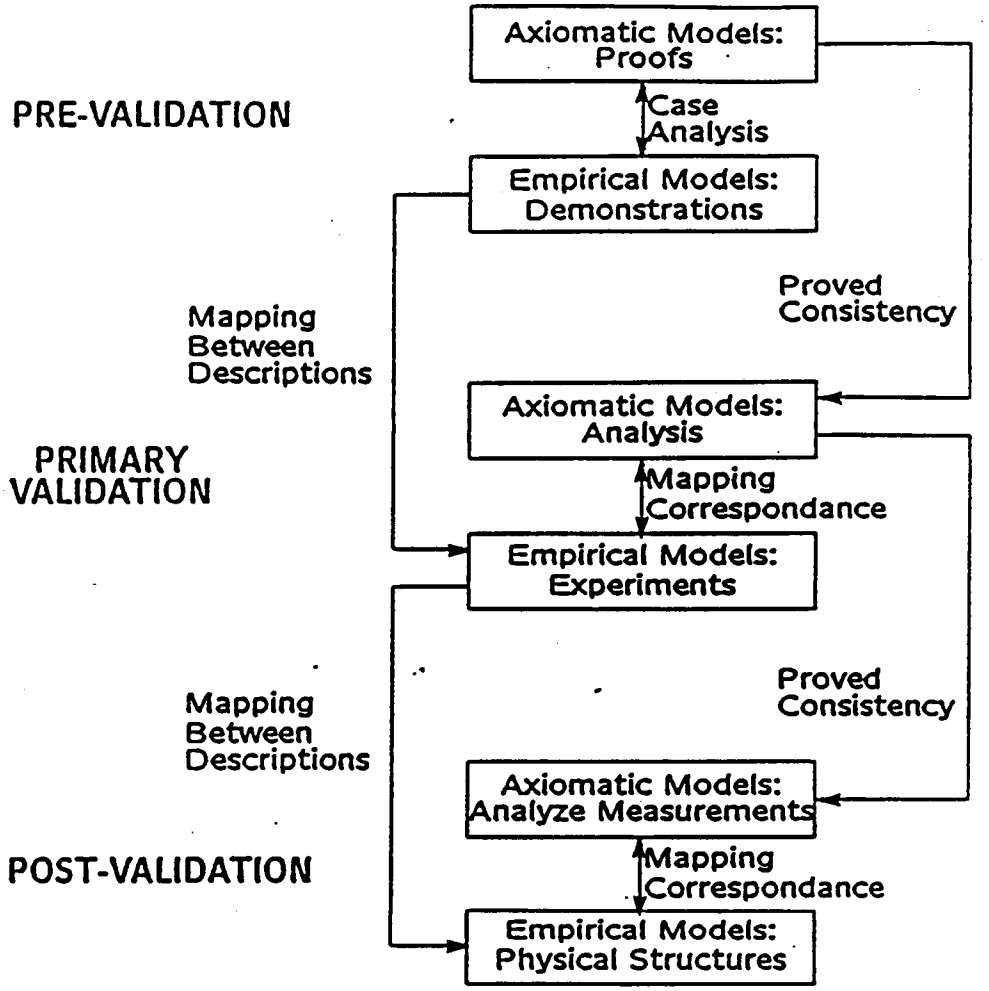


Figure 9. Application of Design Duality and Complementary Completeness

5.3. COMPLEMENTARY COMPLETENESS

If many different models are built using the idea of design duality so that the system at a particular level is considered from many points of view, then complementary completeness says that all of these different points of view must be shown to complement each other. As they complement each other, then designers are more certain that every facet of the design has been considered and validated. Clearly, consistency must be shown between each of the hierarchical levels in the design so that the whole system is integrated and its functions can be audited and traced.

The demonstration of complementary completeness is key to the appropriate use of hierarchical levels of abstraction. During each stage of the system life-cycle, a set of axiomatic models and empirical models are constructed.

5.3.1. PRE-VALIDATION

At the pre-validation stage we have axiomatic models (e.g., proofs or Markov models of system attributes), and empirical models or demonstrations (e.g., a simulation model). The simulation of the system shows the details of the error-handling process, if a fault occurs. The Markov model uses an abstract process of error handling if a fault occurs. The Markov model is evaluated using mathematics to show the predicted reliability. In order to show that these two models have complementary completeness, it must be proven that each one of the empirical models is a proper case analysis of an example of the axiomatic model. If any of the empirical models demonstrate behavior which is contrary to the assumed action in the axiomatic model, then one or both must be changed until they agree.

5.3.2. PRIMARY VALIDATION

During primary validation, the axiomatic models are used for proofs, as well as for more detailed analysis. Also, they are changed because more information is available. The same process can be performed for the empirical models. At this stage, not only do we have demonstrations, but also experiments. Now the axiomatic models at these two levels can be proved consistent since they are both mathematical and satisfy a set of axioms. This proof of consistency must be shown. The empirical models are also supposed to be descriptions of the same system. A mapping between the descriptions must demonstrate that the new and more complicated empirical models are simply an improvement and an extension of the previous models and demonstrations.

5.3.3. POST-VALIDATION

Finally, at the post-validation stage, the axiomatic models are used to analyze the measurements which are obtained from the physical system. These measurements must be proven consistent with the analysis done using mathematical models in the validation stage, which is also consistent with the predictions which were made in the

pre-validation stage. Measurements must be taken from the physical system and must be shown to correspond to the proper predictions. At any stage it must be possible to audit the system properties and attributes, to trace these properties to their initiation, show the decisions which cause them, determine how they are related to the system specifications and requirements, and show that they have been properly validated.

5.4. INTEGRATION OF DESIGN AND VALIDATION TOOLS

Many of the tools currently used for system design can be used for validation with only small changes. For example, tools used for test generation first generate a test, then use a simulation procedure to show that the test not only tests the desired fault, but determines how many other patterns will detect the same fault and, conversely, how many faults will be detected by this single test pattern. The latter is of extreme interest in trying to validate the system. A test can be extended to look at the error-detection circuits to determine which one of the faults will be detected by those circuits, given a particular pattern. This is but one example of the possible integration of design and validation tools.

5.5. PLANNED, TRACEABLE, AUDITED LIFE CYCLE VALIDATION

We must be able, at any stage, to audit the system properties and attributes, to trace these properties to their initiation, show the design decisions which cause them, determine how they are related to their specifications and requirements, and show that they have been properly validated.

6. FUTURE RESEARCH NEEDS

The panel has determined eleven research areas which it believes are necessary. These must be implemented and developed in order to achieve the goal of system validation. They are complementary to the ideas of the past and current research in system validation at NASA-LaRC.

These areas fall into three main categories:

- theory
- design methods
- experiments

Theoretical foundations need to be devised to support the development and use of the new ideas. Design aspects need to be addressed and developed in order to prepare these ideas for experiments and test them against reality. Experiments need to be undertaken to test these ideas and to gain knowledge not only about the new ideas, but about the entire problem of system validation.

New methods of designing systems, so that they can be properly validated, are urgently needed. Systems are built from interactive collections of concepts and ideas.

These concepts and ideas should be clearly stated in their requirements. However, as the system is implemented, there is a conflict of requirements — for example, between performance and reliability. Many of the ideas which were clearly stated in the requirements are not easily seen in the implemented system. They have become anonymous. These concepts do not correspond to individual variables or signals from modules, but rather to a combination of them, many of which do not occur at the same time. Other ideas will be identifiable, but they will be integrated in such a way that they are not easily available for inspection. The designs must identify all the concepts and ideas used and result in the construction of the system which allows their monitoring for the purposes of evaluation and validation.

A practical example of this kind of functional integrity is what IBM calls a processor controller, and what other companies call monitoring processors or maintenance processors. These processors constantly monitor the system status, including electrical and thermal properties, and they handle error recovery. When an error signal is activated, the local state must be gathered and put into the monitoring device. When recovery succeeds, the main system proceeds. The processor controller then analyzes the local state, determines if this error is becoming recurrent (so that the system must be fixed), and locates which of the field replaceable units should be replaced.

As another example, consider the IBM research system, RP3,[28]. (This reference is the first in a series of related papers presented at this conference.) This research prototype will have 256 processors operating in parallel on the same problem. Several proposals have been made to coordinate the parallel action of such processors, ranging from message passing between processors working on relatively independent parts of the problem (such as on the cosmic cube developed for physical simulations at Cal Tech) to the ULTRA machine, which uses a large, shared global memory. The RP3 has been designed so that it can operate in the memory mode of either the ULTRA or the Cosmic Cube or in a combination of the two modes. In addition, there is special hardware which can monitor not only the occurrence of error signals, as discussed previously, but also the progress of the program, and determine various quantities which normally would not be easily available.

6.1. THEORY

6.1.1. SYSTEM SPECIFICATION

As stated earlier, specification methods are formal methods, languages, or notations that describe and represent the behavior of a system or subsystem in a hierarchical fashion. The purpose of a specification method is to provide an analyzable model of the system.

Identification of critical properties of the system components are needed both to support modeling the intended behavior of the system and to provide criteria or conditions that can be validated by analysis. While recognized as serious, this is a major unsolved problem in computer science. Good specification methods do not exist. They

should fully identify the assumptions and critical properties of the system and be mathematically sound and easy to use. The research envisioned here is not the development of a new, comprehensive model. Rather, it is to consider the specification methods which now exist, consider formal and informal extensions of these models, and evaluate the appropriateness of competing methods. The result should be a set of methods which are useful and mathematically valid. An important feature of specification methods is that they must facilitate the use of automatic tools and techniques to aid in building and verifying models. To be useful for realistic systems, the techniques must permit hierarchical specifications and multilingual specifications, because the language which is good for designing a computer circuit is not one which is good for microprogramming, nor is it good for expressing more abstract algorithms.

6.1.2. IDENTIFICATION AND ANALYSIS OF SYSTEM PROPERTIES

Large complex systems contain many interacting and integrated subsystems, modules, data, and functional objects. To develop methods for formally building models of systems, it is necessary to be able to identify the system properties and attributes which must be examined and then to determine the hypothesis and mathematics which will clearly express these and which will permit a recursive definition. In model building, one of the basic problems, which is not well solved at all, is specifying components with dependent properties. For example, it is well known that if a component is connected to the rest of the system by a bus, then if the bus fails, that component is also unusable by the system. Its reliability and availability is dependent upon the reliability properties of the bus. Techniques must be developed to determine the correctness of the solutions given by the models. For design duality, different aspects of the same system are defined. How are we going to show that the results from these aspects are properly related? One way is by saying that the results are related in the same way that the different functional properties of the system are related. Finally, these models must be extended so that they are applicable to large and complicated systems.

6.1.3. DETERMINATION AND QUANTIFICATION OF INTERACTIONS BETWEEN COMPLEMENTARY TECHNIQUES

The main reason for doing this is to show that the process of validation is complete and that nothing has been missed. Secondly, it is to avoid the wastefulness of performing a set of validation activities which lead to unrelated and confusing results. It is impossible to get quantitative measures of validation if activities yield unrelated and confusing results. Basic to this work is the realization that attributes of the different aspects of the same system are related in a way similar to the relationship between these aspects.

6.1.4. VALIDATING HIERARCHICAL LEVELS

The only way to deal with the complexity of a large system is to divide it into hierarchical levels. Work has been done on this; however, it is still a major unsolved problem. The past work has implicitly assumed a static system. Little attention has been given to the validation of complex software systems as a set of layers whose interaction is dynamic.

While some work has been done in protocol validation, this often depends on temporal logic to take care of its dynamic action over a set of time intervals. This work has not been extended to other aspects of a complete system. The use of temporal logic has been extremely difficult, and various improvements which will simplify it have been proposed. Thus, we can say that validating the hierarchical levels of the system is an unsolved problem.

6.1.5. DEVISING EXPERIMENTAL METHODS

Simply putting together a system and running various demonstrations on it is not a satisfactory way to prove that the system has the correct properties. New theoretical ideas will not be very useful unless their validity can be shown, and the best ones determined by experiments.

It is necessary to devise methods based upon the standard paradigm of experimental physics. This paradigm has seldom been used for computer science. To use this paradigm, we must first state the hypothesis to be tested. Secondly, we must determine how the results will be measured, and how they will determine if the attributes of this hypothesis are true. Then, we must plan the experiment, show that the proposed results can be analyzed and collected, and do the experiment. As the experiment is performed, we must collect and analyze the results. This analysis will confirm or deny the hypothesis.

6.2. DESIGN METHODS

6.2.1. STRUCTURING AND PARTITIONING SYSTEMS

As indicated by the earlier examples, methods for structuring and partitioning systems for validation is of primary importance and difficulty. Partitioning has always been done, but doing it well is still an unsolved problem. There is very little data to indicate that any of the proposed solutions are good or valid.

6.2.2. DESIGN FOR TESTABILITY AND VERIFIABILITY

The major problem of design for testability and verifiability is still an important unsolved problem. One of the advances which needs to be made is the one, mentioned earlier, of using the standard recursive ideas for definition. As an example, consider the

6.3.3. VALIDATION TOOLS

Finally, to conduct experiments in system validation, we must have easy-to-use validation tools and methods. These do not exist, except perhaps as a proprietary collection in some large companies. Development of this set of tools should begin with the acquisition of current state-of-the-art design, testing, modeling, simulation, and analysis tools and continue with the acquisition and incorporation of new tools.

These new tools will be determined by the results of the validation research and the results of using the previous set of tools. Tools should be integrated by a data base so that experimental results can be recorded, and information audited and traced.

Much of the work which has been done on validation has been done by human beings without the necessary validation tools. For example, inserting faults into the system is slow, arduous, and extremely complicated, and then analyzing the results of such an experiment is difficult. Proper simulation techniques can show which tests are necessary and which tests are dependent, so that the use of a few tests will be far more profitable.

6.1.4. VALIDATING HIERARCHICAL LEVELS

The only way to deal with the complexity of a large system is to divide it into hierarchical levels. Work has been done on this; however, it is still a major unsolved problem. The past work has implicitly assumed a static system. Little attention has been given to the validation of complex software systems as a set of layers whose interaction is dynamic.

While some work has been done in protocol validation, this often depends on temporal logic to take care of its dynamic action over a set of time intervals. This work has not been extended to other aspects of a complete system. The use of temporal logic has been extremely difficult, and various improvements which will simplify it have been proposed. Thus, we can say that validating the hierarchical levels of the system is an unsolved problem.

6.1.5. DEVISING EXPERIMENTAL METHODS

Simply putting together a system and running various demonstrations on it is not a satisfactory way to prove that the system has the correct properties. New theoretical ideas will not be very useful unless their validity can be shown, and the best ones determined by experiments.

It is necessary to devise methods based upon the standard paradigm of experimental physics. This paradigm has seldom been used for computer science. To use this paradigm, we must first state the hypothesis to be tested. Secondly, we must determine how the results will be measured, and how they will determine if the attributes of this hypothesis are true. Then, we must plan the experiment, show that the proposed results can be analyzed and collected, and do the experiment. As the experiment is performed, we must collect and analyze the results. This analysis will confirm or deny the hypothesis.

6.2. DESIGN METHODS

6.2.1. STRUCTURING AND PARTITIONING SYSTEMS

As indicated by the earlier examples, methods for structuring and partitioning systems for validation is of primary importance and difficulty. Partitioning has always been done, but doing it well is still an unsolved problem. There is very little data to indicate that any of the proposed solutions are good or valid.

6.2.2. DESIGN FOR TESTABILITY AND VERIFIABILITY

The major problem of design for testability and verifiability is still an important unsolved problem. One of the advances which needs to be made is the one, mentioned earlier, of using the standard recursive ideas for definition. As an example, consider the

7. A TIMETABLE FOR FUTURE RESEARCH

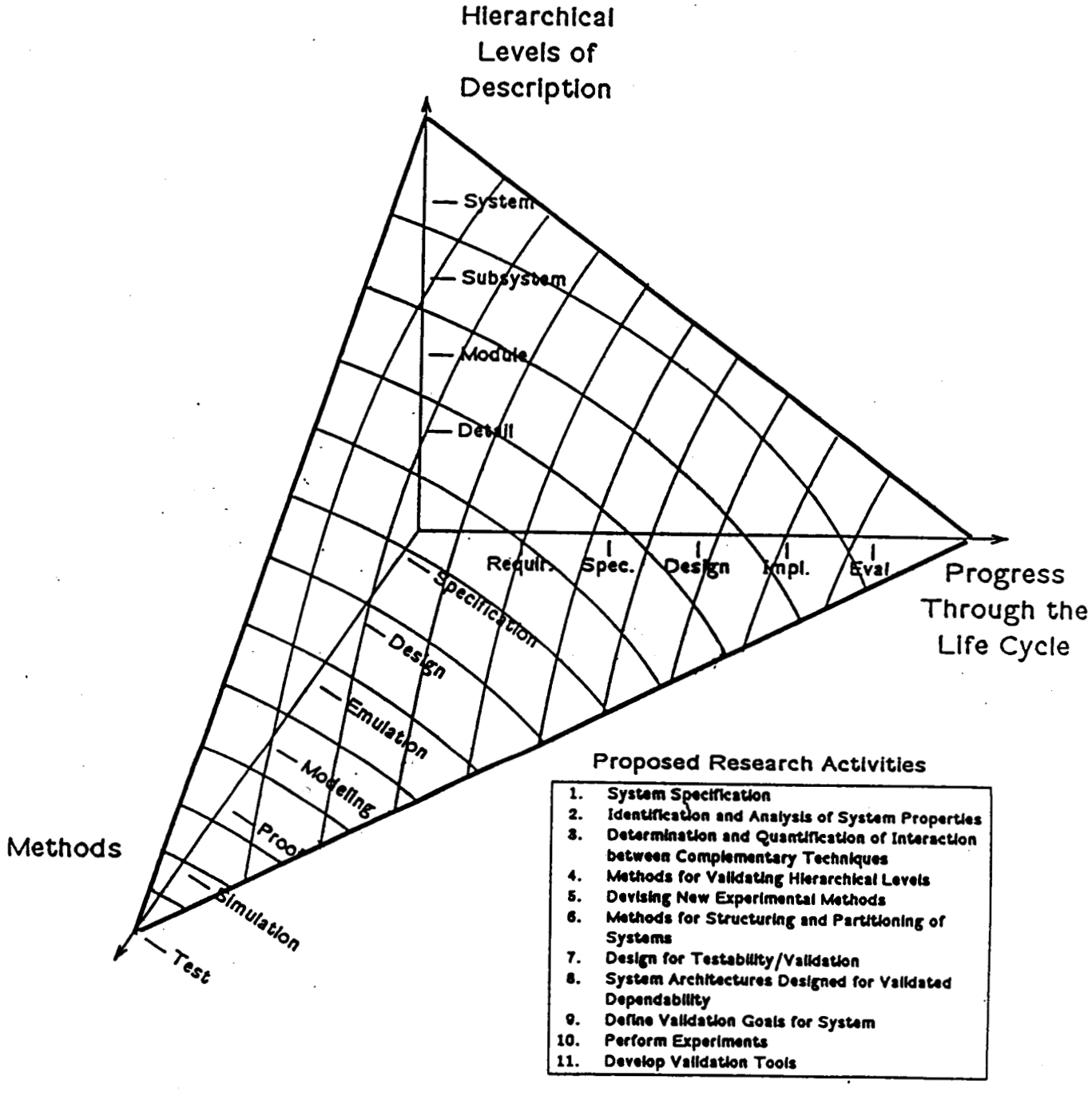


Figure 10. Research Needs in the Validation Space

If the identified tasks are mapped into the validation space (see Figure 10), it is apparent that most of the important ones are abstract; they apply to systems and subsystems. Detail design is discussed only when talking about design detail.

If the ideas are systematically carried out, then instead of small, separated areas, there would be a consistently large area with continuous ideas so that instead of simply saying, "We would like to jump from one area to another," it could be said, "These ideas lead coherently to the one we wish from the one that we have."

A proposed schedule for research in design for validation has been drawn up and each of the eleven research activities has been considered, but not in detail because the panel did not have time to do this, but primarily in terms of the expected results of such a study and when you could expect the results to be available. We divided the times at which the results could be available into three categories: near-term, which are two years or less; medium, which are two to five years (which means, of course, that less is known about them, but we have some idea on how to proceed); and long-term, which are greater than five years.

The research activities listed are known now. Yet, many can be considered only in the long term. Some of these activities will probably be changed so much that they will not be easily recognized.

	Research Activity	Deliverables
1.	System Specification	Near, Long
2.	Identification and Analysis of System Properties	Near, Medium
3.	Determination and Quantification of Interactions between Complementary Techniques	Medium, Long
4.	Methods for Validating Hierarchical Levels	Near, Medium, Long
5.	Devising New Experimental Methods	Near, Medium
6.	Methods for Structuring and Partitioning Systems for Validation	Near, Medium
7.	Design for Testability/Verifiability	Medium
8.	System Architectures Designed for Validated Dependability	Near, Medium, Long
9.	Define Validation Goals for Systems	Near, Medium
10.	Perform Experiments	Near, Medium, Long
11.	Develop and Integrate Validation Tools	Near, Medium, Long

Key: Near: < 2 years
Medium: 2-5 years
Long: > 5 years

Figure 11. A Proposed Research Schedule

7.1. SYSTEM SPECIFICATION

The first research activity is system specification. We believe that the best idea for NASA is to get some results in the near term. This means surveying what has been done, comparing the answers, and making minor modifications to develop a coherent method of doing system specification.

Any other work in system specification should have its deliverables in the long term. There is so little known now and there are so many other people working in the field that it does not appear to be justified for NASA-LaRC to have activity in this field, except on a very long-term basis for the occasional areas that are deemed critical.

7.2. IDENTIFICATION AND ANALYSIS OF SYSTEM PROPERTIES

The second activity is identification and analysis of system properties. This must be started immediately. Until some good methods of identification and analysis are known, much of the work on system validation cannot be done. However, this problem does not appear to be too difficult. We can get some results not only in the very near future but in the medium range. By that time, some specified systems should be built and analyzed using coherent plans, we can learn a great deal more about the identification and analysis, and the problem will undoubtedly have changed.

7.3. DETERMINATION OF INTERACTIONS

The third activity is the determination and quantification of interaction between complementary techniques. This research can begin only after more is known about the identification and analysis of system properties. Therefore, we cannot expect results in the very near term. However, we must get results as soon as possible to have them available in the medium term. This task is one of the most important problems, and one of the most complicated. Results will still be necessary after five years.

7.4. METHODS FOR VALIDATING HIERARCHICAL LEVELS

Results are needed now. NASA needs to collect information (particularly from the improvements in SIFT[11] and the improvements in GYPSY[30] and determine how they can be used. Once this information is collected, more results will be necessary in the medium range to be useful for the FTP[22] system. It is not likely that the method of using hierarchical levels will be solved within five years.

7.5. DEVISING NEW EXPERIMENTAL METHODS

Devising new experimental methods must be started immediately. The status of experiments in computer science is weak.

7.6. STRUCTURING AND PARTITIONING

The next proposed activity is studying methods for structuring and partitioning systems for validation. Once again, near-term projects will consist more of learning what others are doing, and deciding how these design methods can be modified for validation. Again, the long-term problem is vague.

7.7. DESIGNING FOR TESTABILITY AND VERIFICATION

This is a current field of great interest. As mentioned earlier, there are at least two workshops in the area of design for test. There is a major symposium on design for testability every year, and there are papers given on designing for testability and verifiability at the design automation conference and at the fault-tolerant computing symposium. NASA-LaRC should monitor these activities and, after the first flush of papers and experiments have been described, codify them and make these results coherent in the medium term.

7.8. SYSTEM ARCHITECTURES DESIGNED FOR VALIDATED DEPENDABILITY

In the near range we need to set up the research data collection system, determine what we would like to monitor, and start monitoring it. In the medium range, we have to decide how we are going to do the classification and begin collecting data. In addition to collecting data and analyzing it, there probably will be a need (unique to NASA) for advanced system architectures for very reliable systems which operate in real time. Work done previously for NASA-LaRC has been extremely fruitful and is being copied throughout the world. In the long term, new architectures will probably be required.

7.9. DEFINING VALIDATION GOALS FOR SYSTEMS

We do not know enough about how complicated non-sequential systems perform. Experiments are extremely important; to conduct a good experiment we will have to define the validation goals. The methods we use for simple serial systems will not work for real-time parallel systems. Some of the problems which come up are the rescheduling of operations; research has been supported by NASA with the work of Professor Kang Shin [31] [32] [33] [34] at the University of Michigan. This work should be extended.

7.10. EXPERIMENTS

Validation of reliable systems must be recognized as an experimental branch of computer science. It is imperative that a suite of automated tools be developed and integrated so that the results of experiments can be easily gathered and it can be determined if progress is being made.

8. CONCLUSION

The fundamental conclusion of the panel is that there is a crucial need for system validation to become fully integrated into the product life-cycle, from initial specification of system requirements to the operation and maintenance phases of the life-cycle. This conclusion translates directly to the use of formal specification methods, automated aides for the creation and maintenance of specifications, and extensive data base systems that can help capture system performance information completely through to the end of its operational lifetime. At each of the intermediate stages of the system life-cycle, corresponding validation activities have been identified, all of which are deemed essential if the ultrahigh levels of system dependability required for critical missions are to be achieved. The panel was charged with developing a research plan for validating fault-tolerant digital systems used in flight-critical situations. It is expected that this plan will help the NASA Langley Research Center to advance the goal of attaining and accurately estimating the reliability of systems built as part of its AIRLAB research project and as components of future flight-critical and life-critical environments.

Several working groups on validation and fault tolerance have been convened by NASA- LaRC, beginning in March 1979, and have contributed many important ideas. This panel first met in April 1986 and held its final meeting on August 1, 1986; it included experts from the commercial computer industry, the aerospace industry, and members of the NASA Langley research staff.

The panel examined recent research reports, the experience of NASA-LaRC with AIRLAB, and the industry experience and current research findings of members of the panel.

The recommendations of the panel are expressed in a five-year phased development plan consisting of eleven major research areas, as summarized in Figure 11. Each of the research areas is detailed in the body of the report, along with the rationale for its development. The activity areas are categorized within three implementation time frames: immediate to two years, two to five years, and beyond five years.

The panel found that no satisfactory integrated validation approach exists that is likely to meet NASA's mission needs in the near future. While it is believed that most major validation research needs have been identified, many of the problem areas must still be categorized in the domain of basic research. These include:

- formal classification of models
- application of design duality
- application of complementary completeness
- integration of design and validation tools
- planned, traceable, audited life-cycle validation

Further, many of the needs perceived for NASA programs are unique and are not likely to be completely addressed by university or industry researchers in the near term unless specific opportunities are provided to do so.

SUMMARY

As part of its ongoing validation research programs, NASA Langley Research Center formed a panel to plan future research directions. The panel was charged with developing a research plan for validating fault-tolerant digital systems used in flight-critical situations. The panel was comprised of the following members:

- William C. Carter, Chair
- Janet R. Dunham, Vice-chair — Research Triangle Institute
- William E. Howden, Dept. of Electrical Engineering and Computer Science — University of California
- Jean-Claude Laprie — LAAS-CNRS, Toulousse, France
- Brian Smith — Argonne National Laboratory
- Thomas Williams — IBM Corporation, General Technology Div.

Beginning in March 1979, NASA-LaRC convened several working groups on fault-tolerance; they have contributed many important ideas that were precursors to those explored by the current panel. The current panel first met in April 1986 and held its final meeting on August 1, 1986; meeting participants included experts from the commercial computer industry, the aerospace industry, and members of the NASA-LaRC research staff.

The panel examined current research: the experience of NASA-LaRC with AIRLAB, computer industry experience, research findings of members of the panel, and the experience of the aerospace industry.

The recommendations of the panel are expressed in a five-year, phased development plan consisting of eleven major research areas. Each of the research areas is detailed in the body of this report, along with the rationale for its development.

This report is a synthesis by the authors of the panel's conclusions. Additional information on the topics addressed by the panel can be found in the list of references at the end of this report.

We gratefully acknowledge the editing assistance provided by Dr. Brian Smith and the valuable contributions made to the panel's investigations and discussions by the following participants:

- James Clary — Research Triangle Institute
- Bill Dove — NASA Langley Research Center
- Brian Lupton — NASA Langley Research Center
- John Pierce — Research Triangle Institute

References

1. Jean-Claude Laprie, "Dependable Computing and Fault Tolerance: Concepts and Terminology," *Presented at the 15th Annual International Symposium on Fault-Tolerant Computing*, (June 19-21, 1985).
2. Michael Monachino, "Design Verification System for Large-Scale LSI Designs," *IBM Journal of Research and Development* **26, no.1** pp. 89-99 (January 1982).
3. M. A. Breuer and A. D. Friedman, *Diagnosis & Reliable Design of Digital Systems*, Computer Science Press, Inc., Potomac, Maryland (1976).
4. S. J. Bavuso and P. L. Petersen, "CARE III Model Overview and User's Guide," *NASA* **TM-86404**(April 1985).
5. S. J. Bavuso, "Advanced Reliability Modeling of Fault-Tolerant Computer-Aided Reliability Analysis - CARE III," *Presented at AIAA on Los Angeles, California*, (February 11, 1986).
6. R. Geist, K. Trivedi, J. Dugan, and M. Smotherman, "Design of the Hybrid Automated Reliability Predictor," *Proceedings of the 5th IEEE/AIAA Digital Avionics Systems Conference*, (November 1983).
7. R. Geist and K. Trivedi, "Hybrid Modeling Techniques and Their Applications to Computers Systems," *15th Annual Pittsburgh Conference on Modeling and Simulation*, (April 1984).
8. Ricky W. Butler, "The Semi-Markov Unreliability Range Evaluator (SURE) Program," *NASA* **TM-86261**(May 1984).
9. Ricky W. Butler, "The SURE Reliability Analysis Program," *NASA* **TM-87593 (February 1986).**
10. "Peer Review of a Formal Verification/Design Proof Methodology," *NASA* **Conference Publication 2377 (1985).**
11. J. Wensley et al, "SIFT: Design and Analysis of a Fault-Tolerant Computer for Aircraft Control," *Proceedings of the IEEE* **60** pp. 1240-1254 (October 1978).
12. G. E. Migneault, "The Application of Emulation Techniques in the Analysis of Highly Reliable, Guidance and Control Computer Systems," *AGARDOGRAPH*, (1986).
13. G. F. Pfister et al, "Papers on the Yorktown Simulation Engine," *Proceedings of the 19th DA Conference*, pp. 51-64 (1982).
14. L. H. Bangert, K. R. Henke, R. J. Grommes, and W. B. Kerr, "Study of Integrated Airframe/Propulsion Control System Architectures," *NASA* **CR-172167**(November 1983).
15. G. C. Cohen and C. W. Lee, "Design/Validation Concept for an Integrated Airframe/Propulsion Control System Architecture (IAPSA II)," *NASA* **CR-178084**(June 1986).
16. A. D. Stern and C. M. Carlin, "Study of Integrated Airframe/Propulsion Control System Architectures, (IAPSA)Volume I - Executive Summary," *NASA* **CR-**

172173(October 1983).

17. A. D. Stern and C. M. Carlin, "Study of Integrated Airframe/Propulsion Control System Architectures, (IAPSA) Volume II - System Requirements and Development," *NASA* **CR-172174**(October 1983).
18. A. D. Stern and C. M. Carlin, "Study of Integrated Airframe/Propulsion Control System Architectures, (IAPSA) Volume III - AIRLAB Experiment Definition," *NASA* **CR-172175**(October 1983).
19. D. E. Eckhardt, Jr. and L. D. Lee, "A Theoretical Basis for the Analysis of Redundant Software Subject to Coincident Errors," *NASA* **TM-86369**(January 1985).
20. D. E. Eckhardt, Jr. and L. D. Lee, "A Theoretical Basis for the Analysis of Multiversion Software Subject to Coincident Errors," *IEEE Transactions on Software Engineering*, (December 1985).
21. A. L. Hopkins, T. B. Smith, and J. H. Lala, "FTMP - A Highly Reliable Fault-Tolerant Multiprocessor for Aircraft," *Proceedings of the IEEE* **60** pp. 1221-1239 (October 1978).
22. J. Lala, "Advanced Information Processing System," *AIAA/IEEE 6TH Digital Avionics Conference Proceedings*, (December 1984).
23. J. R. Dunham, "Experiments in Software Reliability: Life-Critical Applications," *IEEE Transactions on Software Engineering*, (January 1986).
24. D R. Miller, "Exponential Order Statistic Models of Software Reliability Growth," *IEEE Transactions on Software Engineering*, (January 1986).
25. F. W. Scholz, "Software Reliability Modeling and Analysis," *IEEE Transactions on Software Engineering*, (January 1986).
26. Ricky W. Butler and Sally C. Johnson, "Validation of Fault-Tolerant Clock Synchronization System," *NASA* **TP-2346**(1984).
27. W. E. Howden, "Functional Program Testing," *IEEE Transaction on Software Engineering*, pp. 162-170 (1980).
28. G. F. Pfister et al, "The IBM Research Parallel Processor Prototype (RP3): Introduction and Architecture," *Proceedings of the 1985 International Conference on Parallel Processing*, (1985).
29. T. Anderson and P. A. Lee, "Fault-tolerance terminology proposals," *Proceedings of the 12th International Symposium on Fault-Tolerant Computing*, pp. 29-33 (June 1982).
30. D. I. Good, R. L. London, and W. W. Bledsoe, "An Interactive Program Verification System," *IEEE Transactions on Software Engineering* **1(1)** pp. 59-67 (April 1975).
31. C. M. Krishna, K. G. Shin, and Y. H. Lee, "Optimization Criteria for Checkpoint Placement," *Communications of the ACM* **27** p. No. 10 (October 1984).
32. K. G. Shin, M. H. Woodbury, and Y. H. Lee, "Modeling and Measurement of Fault-Tolerant Multiprocessors," *NASA* **CR-3920**(1985).

33. K. G. Shin and C. M. Krishna, "The Characterisation of Real-Time Computers," *NASA* **CR-3807**(1984).

34. C. M. Krishna, K. G. Shin, and R. W. Butler, "Synchronization and Fault-Masking on Redundant Real-Time Systems," *Preesented at the 14th International Conference on Fault-Tolerant Computing*, (June 1984).

PANEL MEMBERS AND PARTICIPANTS

MEMBERS

CHAIR
Dr. William C. Carter
P.O. Box 301
Bailey Island, ME 00403
(207) 833-5241

VICE-CHAIR
Ms. Janet R. Dunham
Research Triangle Institute
P.O. Box 12194
Research Triangle Park, NC 27709
(919) 541-6562

Dr. Jean-Claude Laprie
LAAS-CNRS
7, Avenue du Colonel Roche
31077 Toulouse Cedex
France
33-61-33-62-39

Dr. Thomas Williams
IBM Corp.
General Technology Division
6300 Diagonal Highway
Boulder, CO 80302
(303) 924-7692

Prof. William E. Howden
Dept. of EE & CS
University of California
San Diego, CA 92093
(619) 534-2723

Dr. Brian Smith
Bldg. 221, Room C-227
Math & Computer Science Div.
Argonne National Lab
9700 South Cass Avenue
Argonne, IL 60439
(312) 972-7232

PARTICIPANTS

Mr. Bill Dove
Information Systems Div.
NASA-LaRC
Mail Stop 469
Hampton, VA 23665
(804) 865-2233

Mr. James Clary
Research Triangle Institute
Center for Digital Systems Research
Herbert Building
Research Triangle Park, NC 27709
(919) 541-6951

Mr. Brian Lupton
NASA-LaRC
Mail Stop 130
Hampton, VA 23665
(804) 865-3681

Mr. John Pierce
Research Triangle Institute
Center for Digital Systems Research
Research Triangle Park, NC 27709
(919) 541-7443

NASA National Aeronautics and Space Administration

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA CR-181835		

4. Title and Subtitle	5. Report Date
Design for Validation: An Approach to Systems Validation	May 1989
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
William C. Carter, Janet R. Dunham, Jean-Claude Laprie, Thomas Williams, William E. Howden, Brian Smith, and Carl M. Lewis	
	10. Work Unit No. 505-66-21-01

9. Performing Organization Name and Address	11. Contract or Grant No.
Research Triangle Institute Research Triangle Park, NC 27709	NAS1-17964
	13. Type of Report and Period Covered
12. Sponsoring Agency Name and Address	Contractor Report
National Aeronautics and Space Administration Langley Research Center Hampton, VA 23665-5225	14. Sponsoring Agency Code

15. Supplementary Notes

Technical Monitor: Charles W. Meissner, Jr., Langley Research Center
Task 7 Final Report

16. Abstract

Every complex system built is validated in some manner. Computer validation began with one person reviewing the design of the system. As systems became too complicated for one person to review, validation began to rely on the application of adhoc methods by many individuals. As the cost of changes mounted and the expense of failures increased, more organized procedures became essential. Attempts at devising and carrying out those procedures showed that validation is indeed a difficult technical problem.

The successful transformation of the validation process into a systematic series of formally sound, integrated steps is necessary if the liability inherent in future digital-system-based avionic and space systems is to be minimized. This report presents a suggested framework and timetable for that transformation.

In sections two through four, we provide basic working definitions of two pivotal ideas--validation and system life-cycle, and show how the two concepts interact. Many examples are given of past and present validation activities by NASA and others. In section five, we present a conceptual framework for the validation process. Finally, in sections six and seven, we list important areas for ongoing development of the validation process at NASA Langley Research Center (NASA-LaRC).

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Computer Systems Validation Design Duality Complementary Completeness	Unclassified-Unlimited Subject Category 62

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price
Unclassified	Unclassified	46	A03