6-1988

Stepwise Refinement and Verification in Box-Structured Systems

Harlan D. Mills

Stepwise Refinement and Verification in Box-Structured Systems

Harlan D. Mills
University of Florida

System design begins in problem domains, usually with considerable informality, and ends in computer domains in completely formal languages for programmers and users. Ideally, the system specification should be defined formally first, and the system designed accordingly. However, there are few systems of any size where this is practical. Even if a designer could do it, the sponsors and users would be hard pressed to understand the formal specification.

I propose that the formality of specifications and designs be developed together in box structures with many sponsor and user interfaces. Box structures allow the stepwise refinement and verification of hierarchical system designs from their specifications at formal and informal levels.

Box structures of data abstractions allow the stepwise refinement and verification of hierarchical system designs from their specifications at formal and informal levels.

Long division in place notation is an example of a stepwise refinement and verification process that produces a quotient and remainder from a dividend and divisor. Each major step produces the next digit of the quotient by a creative estimate of its value followed by an immediate verification of its correctness. If the verification fails, a new estimate is provided and verified. The minor steps are digit-by-digit arithmetic operations with no further invention beyond estimating the next digit of the quotient. Using the fundamental mathematical discoveries that made long division possible, a skilled school child can solve problems beyond the capability of Euclid and Archimedes.

Box-structured system design[1] is a stepwise refinement and verification process that produces a system design from a specification. Such a system design is defined by a hierarchy of small design steps that permit the immediate verification of their correctness, just as the next digit can be verified immediately in long division. Three basic principles underlie the box-structured design process:

(1) All data to be defined and stored in the design is hidden in data abstractions.[2] Even program variables define simple data abstractions with entries for, say, set and query; the set entry with a value ensures that any query entry preceding another set returns that value.

(2) All processing is defined by sequential and concurrent uses of data abstractions. For example, the simple assignment statement $x := y$, where x and y name program-variable data abstractions, can be considered as the sequence of a query of y followed by a set of x using the value returned by y.

(3) Each use of a data abstraction in the system occupies a distinct place in the usage hierarchy.[3] For example, a data abstraction for the assignment $x := x + y$ would use the data abstraction for x in two distinct ways and show the uses in its usage hierarchy.

These three principles unify the distinctions between system design and program design. In particular, Parnas' usage hierarchy[3] provides a powerful place notation for creating, documenting, and inspecting software designs.

0018-9162/88/0600-0023$01.00 © 1988 IEEE

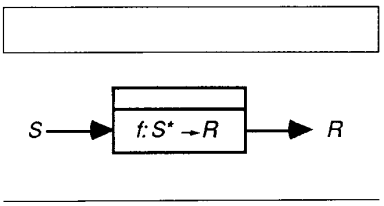

Figure 1. Black box.

Program variables and assignment statements are especially simple data abstractions. Stacks and queues are more elaborate, requiring tens or hundreds of program declarations and statements. Entire systems, such as information or database systems containing thousands or millions of lines of program source code, are also data abstractions, whether consciously developed as such or not.

Defining a data abstraction in three forms reduces the size of steps required to define its structure and use in system design. These forms are called the *black box*, *state box*, and *clear box*. The black box gives an external description of data abstraction behavior in terms of usage. The state box gives an intermediate description in terms of an internal state and the use of an internal data abstraction. The clear box gives an internal description by replacing the state box's internal data abstraction with the sequential or concurrent usage of other data abstractions.

Each major step in box-structure design expands a black-box description into a state box and then into a clear box. An immediate verification of correctness follows each substep. If a verification fails, a new expansion is required. Two points of creative invention are required in each major step: (1) the encapsulation of usage histories into a state box and (2) the decomposition of state-box transitions into a clear-box process. The verification steps are analytic and repeatable without invention.

Usage hierarchies and block diagrams

Block diagrams, such as those found in data-flow diagrams, coalesce the uses of each data abstraction in the usage hierarchy into a single node and coalesce the usage relations among data abstractions into arcs between nodes. Such diagrams irreversibly summarize hierarchical information. Mappings from usage hierarchies to block diagrams are from many to one and mappings from block diagrams to usage hierarchies are from one to many.

Block diagrams used in descriptions summarize system structure to aid general understanding of a system's parts and data flow among the parts. The parts can be decomposed into block diagrams and parts hierarchies for a general perspective of system structure.

Block diagrams used in design are conceptions about data flow among modules to be designed. Since the mapping from block diagram to complete design is from one to many, only one or a few designs that satisfy a block diagram will be correct. Consequently, if the sequential or concurrent use of modules shown in block diagrams is left as programming details for later expansions, then the design process is inherently error-prone because the correctness of use among modules cannot be immediately verified. Correct detailed program design is even more difficult when block diagrams are expanded into hierarchies of more-detailed block diagrams without defining sequential and concurrent uses at each level.

Box structures of data abstractions

There is a direct relationship between object-oriented development and box-structured design in the implementation of data abstractions as "objects."[4] In fact, box-structured design represents a systematic process for creating object-oriented designs.

Object-oriented development also uses inheritance to describe objects or data abstractions. As data abstractions become more complex and usage hierarchies become deeper, inherited properties can dramatically improve the precision of descriptions. Such property hierarchies, combined with formal methods of axiomatic and algebraic description,[2] are needed to deal with substantial systems in an orderly way.

Black-box descriptions. Stepwise refinement and verification of system design describes system behavior entirely in data abstractions. Parnas' principle of information hiding distinguishes between concrete, visible data storage in a system with persistent information and the system's external behavior.[3] Parnas describes system behavior by its traces with no reference to stored data.[5] Hoare also uses traces to define system behavior for networks of communicating processes.[6] These ideas lead to a direct, mathematical description of system behavior.

Any realized data abstraction exists in real time, whether it performs a so-called real-time function or not. Various uses are initiated at various entries, possibly concurrently. It is useful to classify each initiation (defined by the entries and any data required) as a stimulus to the abstraction and each return (defined by the exits and any data produced) as a response by the abstraction. Each response is determined uniquely by the stimuli it has previously received and accepted. That is, for each realized data abstraction, there is a mathematical function from its stimulus histories to its next response.

Many abstractions do not require specific real-time behavior, even though each realization exists in real time. In such cases, the function is defined for sequentially ordered stimuli. For example, a finite-stack data abstraction is usually described as a sequential process rather than a real-time process.

Figure 1 shows a black box, with a stimulus (possibly through multiple concurrent entries) producing a response (possibly through multiple concurrent exits). Let S be the set of possible stimuli and R be the set of possible responses. The black-box description is a mathematical function f from the set of sequences on S (call it S^*) to R of type

$$f: S^* \rightarrow R$$

For especially simple data abstractions, f can be given in closed mathematical notation. For large abstractions, it may be necessary to give f in the natural language of a problem domain, often a mixture of formal and informal language. Whatever the notation, the black-box description is a function.

A specification for a data abstraction can be given as a set of traces[6] consisting of every acceptable sequence of interleaved stimuli and responses. If two subsequences are identical until a stimulus produces different responses in each, then the specification defines a mathematical relation rather than a function. Mathematical relations can also be used as black boxes to describe data abstractions with nondeterministic behavior.

In programming languages that permit program variables to be declared but not

initialized, data abstractions of the variables have nondeterministic behavior up to the first set stimulus, but deterministic behavior after. In programming languages that require variable initialization, these data abstractions are deterministic.

State-box descriptions. The term "data abstraction" implies the possibility of storing data between stimuli to respond to the effects of previous stimuli. For example, in a finite-stack abstraction, a push provides a data item that may be required as a response for some later copy-top operation. The principle of information hiding requires that any such data be regarded as part of an abstract state of the data abstraction, which may be implemented in various ways but always provides a correct description of behavior.

There is a simple mathematical way to guarantee the existence of such a state and the correct behavior of an abstraction with it. Regard the stimulus history itself as the state. Then, for each stimulus, use the black-box function to compute the response from the stimulus history, including the stimulus just received. Finally, compute the new state by appending that stimulus to the previous state. This construction defines a state machine, but not a finite-state machine because stimulus histories are not bounded. Of course, most interesting data abstractions have a function mapping the unbounded set of stimulus histories to a finite set of new state representations that let a finite-state machine provide the black-box behavior.

However, the classical-state machine, in which the state machine transition is a mathematical function from stimuli and old states to responses and new states, has one serious deficiency in elaborating system behavior in a hierarchical structure. Such a transition function forces all state data into the state machine's state, terminating the hierarchy of state data storage. In fact, the implementations of complex data abstractions typically contain several levels, with information hidden at each level.

A state box—a simple generalization of the idea of a state machine—allows such implementations. A state box uses a data abstraction to determine the next state and response for each stimulus. The abstract state can then be distributed in any way desired between the state and the transition behavior of the state box. A state box is pictured in Figure 2, which shows an internal black box whose stimulus arrives concurrently from the external stimulus and the internal state, and whose response departs concurrently to the external response and the new internal state.

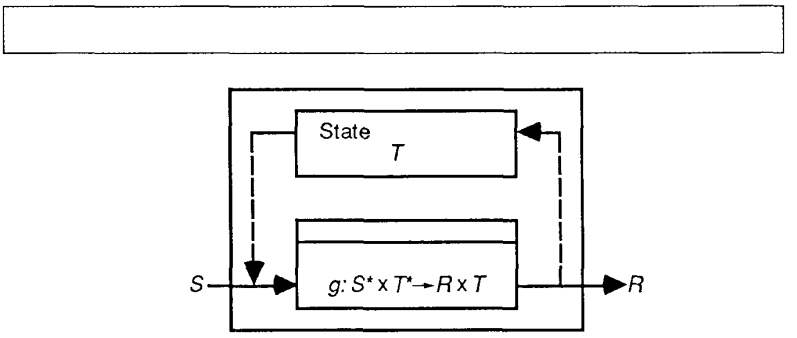

Figure 2. State box.

Let T be a set of states. The behavior of the state box is given by an initial state t in T and function g of the type

$$g\colon S^* \times T^* \rightarrow R \times T$$

that is, a black-box function from stimulus and state histories to the next response and state. Each pair $\langle t, g \rangle$ uniquely defines a black-box function f through the elimination of intermediate states by repeated substitution.

For example, given $\langle t, g \rangle$ with ith stimulus $s.i$, stimulus history $sh.i = \langle s.1, s.2, \ldots, s.i \rangle$, ith response $r.i$, state $t.i-1$, state history $th.i-1 = \langle t, t.1, \ldots, t.i-1 \rangle$, and $g = \langle gr, gt \rangle$ then

$$r.i = gr(sh.i, th.i-1)$$

and

$$t.i = gt(sh.i, th.i-1)$$

Define the state history function gth such that

$$gth(sh.i, th.i-1) = \langle t, gt(sh.1, \langle t \rangle), \ldots, gt(sh.i, th.i-1) \rangle$$

By substitution, if $i > 1$,

$$r.i = gr(sh.i, gth(sh.i-1, th.i-2))$$

If $i > 2$,

$$r.i = gr(sh.i, gth(sh.i-1, gth(sh.i-2, th.i-3)))$$

and, continuing i substitutions,

$$r.i = gr(sh.i, gth(sh.i-1, \ldots gth(sh.1, \langle t \rangle) \ldots))$$

which is a value of a function of type f with parameter t. That is, for each state box there is a unique black box. Given black-box functions of type F, state-box functions of type G, and states of type T, there exists a mathematical function d of type

$$d\colon T \times G \rightarrow F$$

The values of function d are called the black-box derivatives of state boxes. To verify that a state box has been designed correctly to provide black-box behavior, the derived black box need only be compared to the intended black box.

Clear-box descriptions. The theorems and experiences of structured programming lead to a direct definition of four kinds of clear boxes: three sequential forms for sequence, alternation, and iteration, and one concurrent form. In each case, a particular form of sequential or concurrent usage of data abstractions is defined to replace the internal data abstraction of the state box. Alternation and iteration use special data abstractions called *conditions* in which the stimulus is directed out through one of multiple exits. In each use, a regular data abstraction (not a condition) accesses and updates the state.

The definitions in sequential usage are familiar. In concurrent usage, the definitions are novel to ensure referential transparency in concurrent and sequential usage. The definitions in concurrent usage

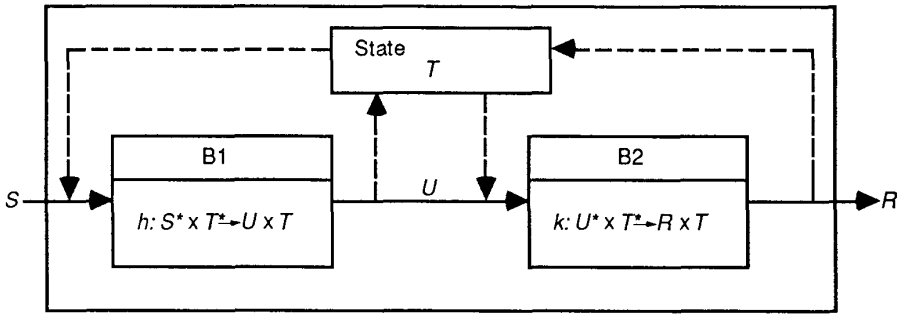

Figure 3. Sequence Clear Box. The response from B1 becomes the stimulus to B2.

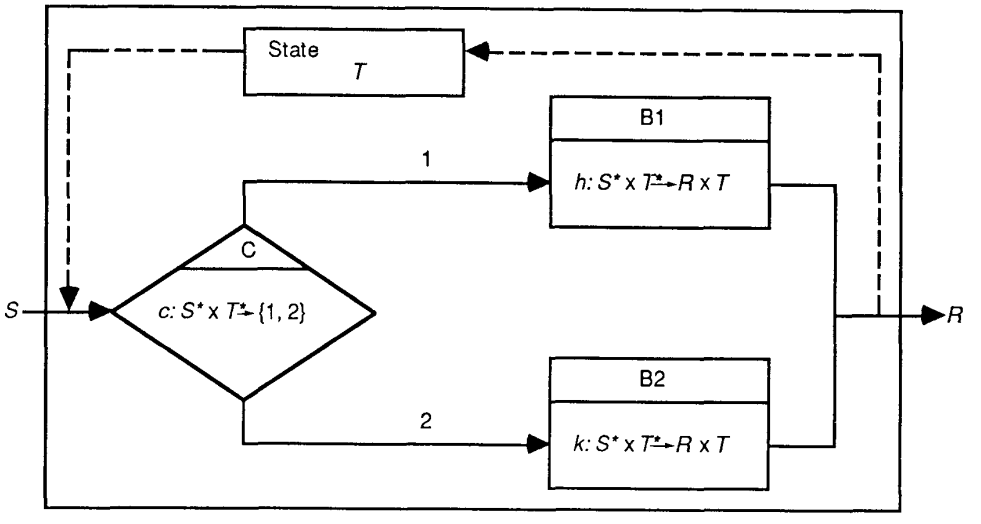

Figure 4. Alternation Clear Box. The condition black box C directs its stimulus to B1 or B2.

provide an ordered set of new states and responses from each concurrent data abstraction, with the requirement that a new data abstraction, called Resolve, be defined to resolve discrepancies among the responses into a single response.

The four kinds of clear boxes are pictured in Figures 3-6. Their function semantics can be derived directly from the semantics of their states and black boxes. As in the case of mapping a state box into a black box, there is a derivative function mapping any clear box into a unique state box. To verify that a clear box has been designed correctly to provide state-box behavior, the derived state box need only be compared with the intended state box. These verifications can be carried out by substitution and case analyses to elimina sequential and concurrent process.

Realism and rigor in software design

When jointly developing formality in specifications and designs, a basic principle is that the final, formal system will have the behavior of a black box function. So the behavioral specification should be a function or relation at every level of for-

mality. The box structures allow any level of formality. As formality increases, fallibilities and ambiguities can be discovered and corrected.

Sponsors and implementers urgently need a coherent account of design activities. However, the unfolding of a design from specifications to computer resources requires considerable learning with much trial and error. In particular, two formidable problems complicate the design trail. Intelligent decision-making at the top level requires that various low-level problems be assessed and solved in detail. Also, any reasonable method must recognize that the specification is almost certain to change during development. To deal with such problems, Parnas advocates a usage hierarchy of modules, each hiding certain secrets, by a joint study of the specification in the problem domain and the available computing resources, with the probability of change explicitly recognized.[3]

Box-structured design leads to the same goal as Parnas' usage hierarchy. Let's assume that the top of a mountain is a formal description and that farther down its slopes the descriptions are more and more informal (with more and more fallibility). I propose a spiral approach to the top through several levels of formality. In fact,

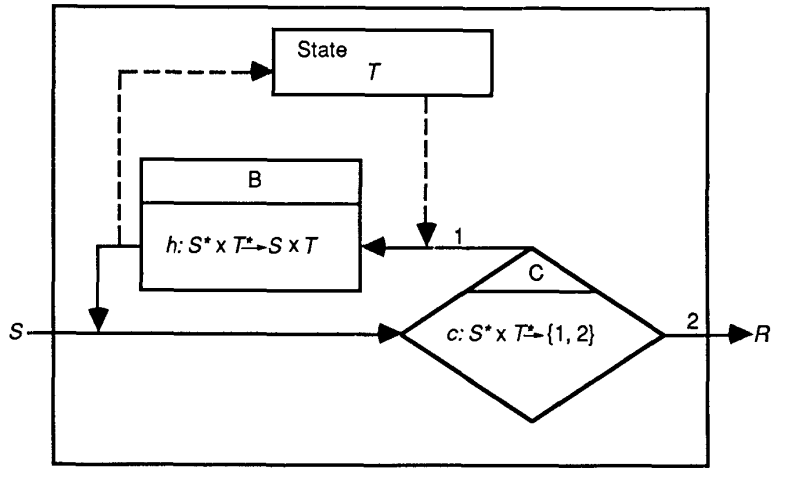

Figure 5. Iteration Clear Box. The condition black box C directs its stimulus to B or the external response of the clear box. The response from B becomes the stimulus to C.

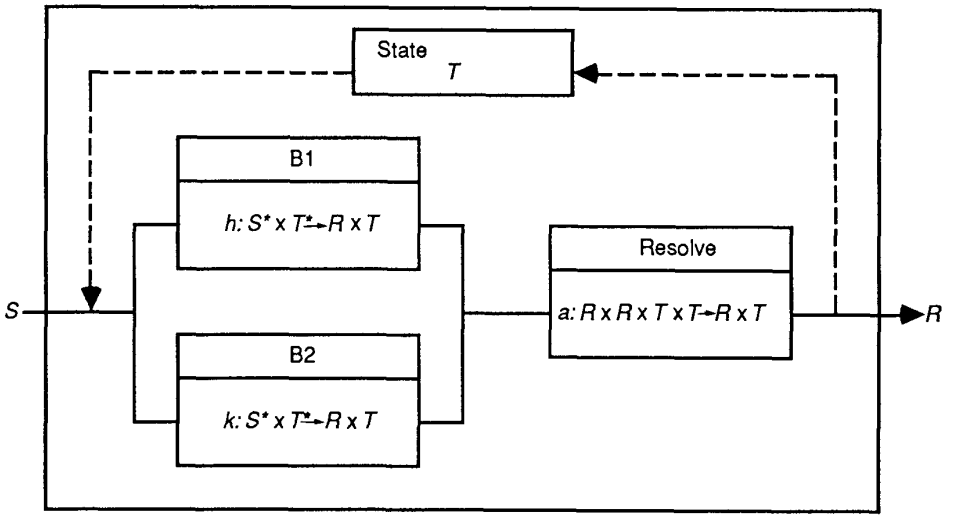

Figure 6. Concurrent Clear Box. The external stimulus is directed to both B1 and B2. The responses from B1 and B2 together become the stimulus to Resolve.

the climb begins in the language of the problem domain—often a lot of English—and ends up in the computer domain of entirely formal code. Box structures, whether formally or informally described, are used on the way up with stepwise refinement and verification. Also, advance scouts must move ahead to assess and solve problems in low-level details with more formality. So, for example, Parnas' ideas about looking for appropriate secrets in upcoming data abstractions are useful.

System design by box-structure expansion

A box-structure expansion begins with a black-box specification of a data abstraction. It identifies the black box, a state box with the same behavior as that of the black box, and a clear box with the same behavior as that of the state machine, using data abstractions at the next level with their own black-box behavior. Although the theory can be given entirely in function theoretical terms with abstract states and functions, a practical design process must use concrete design and programming languages to describe data abstractions and their uses. This shift from theory to practice involves only a change in syntax, not in semantics. A design or program is a rule for a function, and data descriptions and program structures are means for facilitating expression of such rules.

Each expansion of a box structure is a step in designing internal state data and internal sequential or concurrent process. These steps can require considerable invention. It helps to break each expansion into smaller steps, leaving design trails that permit more objective engineering inspections. For this purpose, I define the following 11-step box-structure expansion process:

Define the black box

(1) Define black-box stimuli
Determine all possible stimuli for the black box.

(2) Define black-box behavior
For each possible stimulus, determine its complete response in terms of its stimulus history.

Design the state box

(3) Discover state data requirements
For each response to be calculated, encapsulate its stimulus history into a state data requirement.

(4) Define the state
Select a subset of the required state data items to encapsulate stimulus histories.

(5) Design the state box
For the selected state, determine the internal black box required for the state box.

(6) Verify the state box
Verify the correctness of the state box with respect to the required black-box behavior.

Design the clear box

(7) Discover state data accesses
For each item of state data and each possible stimulus, determine all possible accesses of the item.

(8) Define data abstractions
Organize state data into data abstractions for effective access.

(9) Design the clear box
Define sequential or concurrent uses of the data abstractions defined to replace the internal black box of the state box.

(10) Verify the clear box
Verify the correctness of the clear box with respect to the state-box behavior.

Continue the process

(11) Repeat stepwise expansion until design completion
For each new data abstraction, repeat steps 1-10 until suitable data and program specifications are reached.

The inventions required in this process are strictly contained in steps 4, 5, 8, and 9 and labeled there. The other steps are analytic and repeatable. This process isolates and embeds the creative design steps, allowing design reviews in canonical cases and automatically developing relevant information while leading up to each step. For example, step 3 provides enough background to carry out step 4 and review it objectively.

In contrast, heuristic approaches often skip these analytic steps and leap to networks of sources, processes, stores, and sinks. However, in large problems, it is difficult and sometimes painful to determine if a leap was inspired or flawed. The complexity and the number of design alternatives make it risky to leap that discontinuity without a lot of engineering analysis.

A navigation and weather buoy case study

Booch uses the problem of a navigation and weather buoy to illustrate a data-flow approach to object-oriented architecture and design.[4] The problem was redone in box structures as shown below. (I subsequently learned that this problem was originally suggested by Chmura et al.[7] with a solution in terms of a set of information-hiding modules. The solution below was developed without knowledge of Chmura's solution.) Booch gives the following statement of the problem.

The Host at Sea system is a group of free-floating buoys that provide navigation and weather data to air and ship traffic. The buoys collect data on air and water temperature, wind speed, and location through sensors. Each buoy can have a different number of sensors and can be modified to support other types of sensors.

Each buoy is also equipped with a radio transmitter (to broadcast weather and location information as well as an SOS message) and a radio receiver (to receive requests from passing vessels). A sailor can flip a switch on the buoy to initiate an SOS broadcast and some buoys are equipped with a red light that can be activated by a passing vessel during search operations. Software for each buoy must:

- Maintain current average wind, temperature, and location information. Wind speed readings are taken every 30 seconds, and temperature and location readings are taken every 10 seconds. Wind and temperature values are kept as a running average.
- Broadcast wind, temperature, and location information every 60 seconds.
- Broadcast wind, temperature, and location information from the past 24 hours in response to requests from passing vessels. This takes priority over the periodic broadcast.
- Activate or deactivate the red light based on a request from a passing vessel.
- Continuously broadcast an SOS signal after a sailor engages the emergency switch. This signal takes priority over all other broadcasts and continues until reset by a passing vessel.

On the basis of this problem statement, Booch invented a data-flow diagram and identified objects, attributes, and opera-

tions for an object-oriented architecture for such a buoy.[4] However, the 11-step expansion process outlined above yields a different architecture.

Preliminary black-box analysis. The data-flow approach jumps right in with various internal flows and processes to solve the problem. The box-structure approach focuses first on defining the problem as a real-time transformation of stimuli to responses. The problem statement identifies the following general types of stimuli:

- clock data (various references to time)
- wind data
- temperature data
- location data
- request to broadcast 24 hours of weather data
- request to activate or deactivate a red light
- request to start or stop continuous SOS broadcast

Members of these stimulus types may arrive concurrently in various combinations to make up a stimulus.

The problem statement further identifies possible responses to these stimuli as

- start or stop continuous SOS broadcast
- activate or deactivate the red light
- start a broadcast of 24 hours of weather data
- start a broadcast of current weather data

Since there is only one transmitter, starting a broadcast means stopping any other broadcast currently under way.

The rest of the problem statement generally indicates which response should follow any possible stimulus. The response depends on the stimuli accumulated to the moment, through references to running averages. So the buoy's black box will require histories of certain stimuli covering more than 24 hours (for example, a 24-hour-old running average will require data more than 24 hours old).

Data must be encapsulated in any state box for this black box. References to running averages of wind and temperature data suggest that new data abstractions can greatly simplify the necessary computations and data management. However, before rushing into architecture and design decisions on internal data flows and processes, it is worthwhile to focus more attention on the problem.

Questions about the problem statement. This preliminary black-box analysis shows that the problem statement is far from a specification. Many additional decisions are needed to remove ambiguities, ensure completeness and consistency, and provide a solution without unpleasant surprises for the buoy's sponsors and users. Such problems should be tackled at the black-box level before plunging into state-box and clear-box expansions.

A black box is a formal specification that is complete, unambiguous, and consistent.

Obviously absent from the problem statement is how the buoy is to be operated. While the buoy could be developed as an expendable device that is deployed and left alone, whoever is responsible for its operation might want more control (for example, to monitor system integrity, security, or correctness through testing or diagnostics). If so, additional types of stimuli and responses must be defined.

This example assumes the buoy is expendable. However, note that a focus on immediate users, to the exclusion of secondary users such as operators and maintainers, usually leads to faulty designs that can only be patched up enough to become poor designs.

Also absent from the problem statement are questions involving initialization:

- Is the periodic weather broadcast (every 60 seconds) tied to Greenwich Mean Time (GMT) or to an internal clock that will appear random in real time to its users?
- What is expected if a 24-hour-weather broadcast is requested before enough data have been collected since the start (or restart) of the buoy's operation?
- Can the operators restart the buoy (for example, after moving it to another location in an emergency situation)?
- How are the running averages to be initialized?

The problem statement mentions that SOS broadcasts and 24-hour-weather broadcasts have priority, but that raises more questions:

- Does a lower-priority broadcast abort when a higher-priority one is requested, or does it finish first?
- Can a 24-hour request abort another 24-hour request?

Other questions to be addressed at this stage include

- What is the exact content of a current-weather broadcast? Of a 24-hour-weather broadcast?
- How many samples are needed for running averages, and can that parameter be controlled by the operators?
- Is the running average of wind a vector average?

Completing the problem statement. A black-box analysis forces the identification of every possible stimulus of the buoy and every acceptable response in terms of the stimulus history. The principle of transaction closure[1] requires that any information needed to produce a response be provided by some previous stimulus. For example, a weather broadcast requires previous wind and temperature data.

A black box is a formal specification that is complete, unambiguous, and consistent because it is a mathematical function or relation from all possible stimulus histories to responses, whether it is represented in mathematical notation, English, or a mixture of the two in the problem domain. Completion of a black-box analysis and description usually requires many interactions with sponsors and users. However, getting a good specification is far less expensive over the life cycle than launching into design and implementation without knowing what the sponsors had in mind or the users needed.

In this case study, let us assume the following resolutions to the previous questions:

- The buoy clock and sensors are restarted at the next GMT minute mark after the moment of restart.
- The phrase "24-hour-weather broadcast" means "weather-since-restart broadcast" if restart was less than 24 hours ago.
- A buoy can be restarted or shut down at any time by the use of a password, with restart conditions for its location and running-average parameters. The password can be changed by use of the current password; the initial password is "buoy." The password is unchanged by a restart or shutdown.
- A request for SOS broadcast preempts and aborts any other broadcast.
- A periodic weather broadcast is cancelled if any other broadcast is under way.

• A request for 24-hour-weather broadcast is ignored if an SOS broadcast is under way. It is queued to follow a periodic current-weather broadcast or 24-hour-weather broadcast if one of these is already under way. Otherwise, it is granted immediately.

• The exact content of a current-weather broadcast is the current location, running average of wind, and running average of air and water temperatures. A 24-hour-weather broadcast contains the current-weather broadcasts for each of the previous 24 GMT hour marks (or all GMT hour marks if restart was less than 24 hours ago).

• The number of samples in running averages for wind and temperature is defined by integer parameters set at restart or by default. The term "running average" means "average since restart."

• The running average of wind is the running average of the wind vector.

In practice, these resolutions should be further scrutinized by sponsors, users, and analysts, and the entire problem statement/black box should be cast into a more systematic problem-domain statement for formal review and concurrence by sponsors and users.

The result is a mathematical function or relation from all possible stimulus histories to responses, in which transaction closure is obtained. This function or relation must deal specifically with the response to the first stimulus, the second, the third, and so on, even though it is tempting to focus on steady-state operations far removed from initial conditions.

Stepwise box-structure expansion of the buoy problem

Step 1. Define black-box stimuli. The second round of the problem statement analysis gave the buoy a restart capability that obviates all history except the current password. Such a restart capability is usually needed for operational control, no matter how well the device was thought out.

The physical media for stimulus types include clock and sensor connections, radio receptions, and mechanical switches. Let us suppose the buoy has a basic internal clock that polls the digitized information and is accurate enough to deal with the sensors and the radio transmitter and receiver. Let us also assume that a "start broadcast" command is available and that the transmitter returns a "broadcast terminated" signal.

Sponsors, users, and analysts must determine the response from every possible stimulus history.

Because of the need to maintain GMT, let us suppose the clock is synchronized to GMT by means outside the scope of this study and that the basic periodic clock pulse generated is present in every stimulus. The possible stimulus types are

- clock pulse,
- restart command and data,
- shutdown command,
- change password command and data,
- wind data,
- air temperature data,
- water temperature data,
- location data,
- request to broadcast 24 hours of weather data,
- request to activate a red light,
- request to deactivate a red light,
- request to start continuous SOS broadcast,
- request to stop continuous SOS broadcast, and
- broadcast terminated.

Thus, there are 14 possible stimulus types present in every stimulus. The clock pulse is always present and the others are present independently of each other. Seven of these types contain data, but seven do not.

Some types conflict in their effects. For example, a vessel may send a request to stop continuous SOS broadcast due to its handling of one emergency at the same time a sailor pushes the switch to start an SOS broadcast for another emergency. The black box must describe the response to this presumably unusual case. In fact, the information developed in the interaction between sponsors, users, and analysts must determine the response from every possible stimulus history, whether expected frequently or infrequently.

Step 2. Define black-box behavior. As derived above, the domain for the black box (function) is the set of all possible histories of stimuli with one to 14 members of these stimulus types. That is an infinite domain, but with a simple structure. Fortunately, the interactions between these stimulus types are also simple.

The function mapping from these stimulus histories to responses is simple enough to decompose the effects of stimulus types to a few cases, many quite autonomous for the stimulus type involved. This is shown in Figure 7, where the response required is denoted by *R*. The response required for each stimulus type is given in Box Description Language[1] for readability by sponsors, users, and analysts. The outer syntax is formal (denoted in Figure 7 in uppercase characters), but the inner syntax is informal for now.[8] The informal expressions in inner syntax should be replaced by more formal expressions as the design progresses.

Note that the responses in Figure 7 are described entirely in terms of stimulus histories. The phrase "broadcast is under way" looks suspiciously like a status, but is used as shorthand for "broadcast previously started with no subsequent broadcast termination stimulus." "Broadcast has been requested" is shorthand for "broadcast previously requested with no subsequent broadcast started." It is sometimes convenient to use response history (such as broadcast started) as proper shorthand in a black-box description because any such response can be determined from previous stimulus history.

The actions in Figure 7 are limited to responses without presuming internal activity. For example, statement 9, in response to a request to broadcast 24 hours of weather data, responds only if no broadcast is under way. If a broadcast is under way, one might expect some internal action to note the request for later response, but statement 9 takes no such action. However, statement 1 deals with this situation by checking stimulus histories for requests that can be responded to at each clock pulse. The principle is to deal only with responses specified by stimulus histories, not to begin inadvertently inventing internals.

Note that several stimulus types are accepted during shutdown, namely restart command and data, request to activate or deactivate red light, request to start or stop SOS broadcast, all sensor stimuli, and broadcast termination. These are decisions about specifications as well as design if they have not been explicitly defined. In fact, the black box will define a specification that the sponsors and users should

1. clock pulse
   R: IF no shutdown command since last restart command
   THEN
   IF no broadcast is under way AND a 24-hour-weather broadcast has been requested
   THEN form and start 24-hour-weather broadcast
   ELSE
   IF GMT is at the minute mark
   THEN form and start current-weather broadcast.

2. restart command and data
   R: IF password correct
   THEN confirm restart.

3. shutdown command
   R: IF no shutdown command since last restart command
   THEN
   IF password correct and no restart command
   THEN confirm shutdown.

4. change password command and data
   R: IF no shutdown command since last restart command
   THEN
   IF password correct and no restart or shutdown command
   THEN confirm password change.

5. wind data
   R: acknowledge data.

6. air temperature data
   R: acknowledge data.

7. water temperature data
   R: acknowledge data.

8. location data
   R: acknowledge data.

9. request to broadcast 24 hours of weather data
   R: IF no shutdown command since last restart command
   THEN
   IF no broadcast is under way
   THEN form and start 24-hour-weather broadcast.

10. request to activate red light
    R: activate red light.

11. request to deactivate red light
    R: IF no request to activate red light
    THEN deactivate red light.

12. request to start continuous SOS broadcast
    R: start continuous SOS broadcast.

13. request to stop continuous SOS broadcast
    R: IF no request to start SOS broadcast
    THEN stop continuous SOS broadcast.

14. broadcast termination
    R: acknowledge termination.

Figure 7. Black-box responses for buoy.

understand in confirming previous agreements on what is required of the system.

Step 3. Discover state data requirements. The next step is to determine the information needed to encapsulate stimulus histories to be maintained from one stimulus to the next, so no previous stimulus is required to determine the response. There is a simple necessary and sufficient condition for this encapsulation:

- The responses define the necessary information to be maintained in the state box.
- The history of stimuli contains sufficient information for the state box.

That is, a satisfactory encapsulation of history into the state and internal black box of the state box can be derived directly from the black box.

To provide a convenient design trail for engineering inspections of the buoy, I expand the listing of stimulus types and responses in Figure 7 into state data that encapsulates the necessary histories, as shown in Figure 8. The encapsulation follows directly from an examination of the responses and their dependency on stimulus histories.

For example, in the clock pulse stimulus type, the condition "no shutdown command since the last restart command" must be encapsulated because it depends on stimulus history. Let us encapsulate it in "buoy status," an invented term for a derived requirement, and suppose that buoy status is on only if there has been no shutdown command since the last restart command. Similarly, the condition "no broadcast is under way" can be encapsulated in "broadcast status," and "24-hour-weather broadcast has been

1. clock pulse
   R: IF no shutdown command since last restart command
   THEN
   IF no broadcast is under way AND a 24-hour-weather broadcast has been requested
   THEN form and start 24-hour-weather broadcast
   ELSE
   IF GMT is at the minute mark
   THEN form and start current-weather broadcast.
   E: buoy status, broadcast status, broadcast-request status, 24-hour weather history, clock time, location, wind history, air temperature history, water temperature history

2. restart command and data
   R: IF password correct
   THEN confirm restart.
   E: password, restart state.

3. shutdown command
   R: IF no shutdown command since last restart command
   THEN
   IF password correct and no restart command
   THEN confirm shutdown.
   E: password, shutdown state

4. change password command and data
   R: IF no shutdown command since last restart command
   THEN
   IF password correct and no restart or shutdown command
   THEN confirm password change.
   E: password

5. wind data
   R: acknowledge data.
   E: none

6. air temperature data
   R: acknowledge data.
   E: none

7. water temperature data
   R: acknowledge data.
   E: none

8. location data
   R: acknowledge data.
   E: none

9. request to broadcast 24 hours of weather data
   R: IF no shutdown command since last restart command
   THEN
   IF no broadcast is under way
   THEN form and start 24-hour-weather broadcast.
   E: broadcast status, 24-hour weather history

10. request to activate red light
    R: activate red light.
    E: none

11. request to deactivate red light
    R: IF no request to activate red light
    THEN deactivate red light.
    E: none

12. request to start continuous SOS broadcast
    R: start continuous SOS broadcast.
    E: none

13. request to stop continuous SOS broadcast
    R: IF no request to start SOS broadcast
    THEN stop continuous SOS broadcast.
    E: none

14. broadcast termination
    R: acknowledge termination.
    E: none

Figure 8. Derivation of encapsulated data for buoy.

requested'' can be encapsulated in ''broadcast-request status.'' In Figure 8, encapsulated data is denoted by *E*.

Note that these state data requirements are derived from the responses of the black box, not the stimuli. For example, the stimulus types for wind and temperature data require only acknowledgment of such data, not retention. The need to encapsulate wind and temperature data in the state box comes from the response ''form and start current-weather broadcast.'' In summary, the encapsulated data requirements are

- buoy status,
- broadcast status,
- broadcast-request status,
- 24-hour weather history,
- clock time,
- location,
- wind history,

- air temperature history,
- water temperature history,
- password,
- restart state, and
- shutdown state.

Step 4. Define the state. The identification of state data requirements is a first design step. However, all such data are candidates for migration into lower-level box structures. For example, the four history types in the above list appear to be logical candidates for migration, since they will each require considerable storage and processing to meet the buoy's needs. Also, restart state and shutdown state are candidates for migration because each appears in only one statement. The remaining data items are scalar and can make up the state for the buoy state box. Decisions on the migration of encapsulated data are reversible if further analysis uncovers a better strategy.

Step 5. Design the state box. Steps 3 and 4 derive state data from the black box by rewriting the responses in Figure 7 in terms of state data and appending the state transitions required for the state box. These responses and transitions are shown in Figure 9. For each stimulus type, the response and transition is denoted by *RT*. In Box Description Language, CON/NOC brackets concurrent statements separated by commas.

Note the internal action in statement 9 that turns broadcast-request status on if the response is to be handled later, in contrast with statement 9 of the black box.

Step 6. Verify the state box. To verify the state box, the state data must be eliminated to obtain a derived black box, which then must be compared with the intended black box. The derivation for this state box is quite direct at the level of description given.

For example, the clock pulse *RT* statement in Figure 9 begins

IF buoy status on

while the clock pulse *R* statement in Figure 7 begins

IF no shutdown command since last restart command

which must be verified as equivalent. In this case, Figure 9 shows that buoy status is set only by the restart command and shutdown command. Since the restart command only sets buoy status on and the shutdown command only sets buoy status off, eliminating the state data in the condition "buoy status on" reduces to any stimulus history in which "no shutdown command since last restart command" holds. Therefore, the two IF statements from the black box and state box begin with equivalent conditions.

The outline of the verification can be a reminder and guide for the formal design and verification.

Broadcast status and broadcast-request status can be treated similarly to buoy status. The systematic elimination of state data in Figure 9 to derive a black box to compare with Figure 7 may seem like a rather detailed effort at this point, but it builds a solid foundation for continuing the design, even on an informal basis such as this.

The alternative to this detailed analysis is to leave the high-level control properties defined by these three state items to later programming details, which cannot be verified as design decisions, and leave the actual design to people who may not comprehensively understand the system requirements. However, in system design, every level of decomposition must be controlled by a few details that should be identified and verified immediately.

Figure 9 should contain enough information to verify the correct use of state data to meet the requirements of black-box behavior in Figure 7. If this verification cannot be carried out, even informally, the state box is not completely defined.

In a completed design in a formal language, the derivation will take on the character of a formal engineering analysis of the designed state box to determine the derivative black box for comparison with a formal black-box specification. This engineering analysis is defined in the function theoretical proof that a state box has a unique black-box derivative. But even at the informal level described here, the outline of the verification can be a reminder and guide for the formal design and verification.

Step 7. Discover state data accesses. The previous lists and Figures 7, 8, and 9 can be used to cross reference all possible accesses to this data in various stimulus types. For example, buoy status data will be captured in certain stimuli and the analysis shows the necessity of their retention in state data. These cross references are given in Figure 10. For each state data requirement item, every stimulus type that could or should access it is listed. For each such type, every type of action related to the items is also listed. For convenience, I identify each access as an update or use. Data must be updated before being used, so further study is indicated if analysis shows no update.

Step 8. Define data abstractions. Access and storage of the 12 data items listed in Step 3 have been represented explicitly in the state or in data abstractions at lower levels. Figure 8 shows every access by every stimulus type, providing a basis to derive the black boxes required for a clear-box design at this level. Six of these objects represent scalar variables in the state:

- buoy status
- broadcast status
- broadcast-request status
- clock time
- location
- password

while four represent histories to be migrated as common services to new data abstractions:

- 24-hour weather history
- wind history
- air temperature history
- water temperature history

and two are complete buoy states to be migrated down in the clear box to be designed:

- restart state
- shutdown state

The response requirements on these common data abstractions determine their forms. For example, "24-hour weather history response" is a sequence of current-weather records, each consisting of a GMT hour mark, location, wind average, air temperature average, and water temperature average, with a maximum of 24 elements in the list. However, the only use of wind average and air and water temperatures (in current-weather broadcast) calls for a running average, so these histories can be migrated and encapsulated into abstractions whose only data responses are running averages.

1. clock pulse
   RT: IF buoy status on
   THEN
   CON
   update clock,
   IF broadcast status off AND
   broadcast-request status on
   THEN
   CON
   form and start 24-hour-
   weather broadcast,
   set broadcast status on,
   set broadcast-request status
   off
   NOC
   ELSE
   IF clock time is at the minute mark
   THEN
   CON
   form and start current-
   weather broadcast,
   set broadcast status on
   NOC
   NOC

2. restart command and data
   RT: IF password in stimulus = password
   THEN confirm restart

3. shutdown command
   RT: IF buoy status on
   THEN
   IF password in stimulus = password
   AND no restart command
   THEN confirm shutdown

4. change password command and data
   RT: IF buoy status on
   THEN
   IF password in stimulus = password
   AND no restart or shutdown
   command
   THEN
   CON
   confirm password change,
   update password
   NOC

5. wind data
   RT: acknowledge data

6. air temperature data
   RT: acknowledge data

7. water temperature data
   RT: acknowledge data

8. location data
   RT: CON
   acknowledge data,
   update location
   NOC

9. request to broadcast 24 hours of weather data
   RT: IF buoy status on
   THEN
   IF broadcast status off
   THEN
   CON
   form and start broadcast,
   set broadcast status on
   NOC
   ELSE
   set broadcast-request status on

10. request to activate red light
    RT: activate red light

11. request to deactivate red light
    RT: IF no request to activate red light
    THEN deactivate red light

12. request to start continuous SOS broadcast
    RT: IF broadcast status not SOS
    THEN
    CON
    start continuous broadcast,
    set broadcast status on
    NOC

13. request to stop continuous SOS broadcast
    RT: IF no request to start SOS broadcast
    THEN
    CON
    stop SOS broadcast,
    set broadcast status off
    NOC

14. broadcast termination
    RT: CON
    acknowledge termination,
    set broadcast status off
    NOC

Figure 9. State-box responses and transitions for buoy.

1. buoy status
   - in clock pulse
     use to test if buoy on
   - in restart command and data
     update as part of restart state
   - in shutdown command
     use to test if buoy on
     update as part of shutdown state
   - in change password command and data
     use to test if buoy on
   - in request to broadcast 24 hours of weather data
     use to test if buoy on
   - in broadcast termination
     use to test if buoy on

2. broadcast status
   - in clock pulse
     use to test if no broadcast under way
     update at start 24-hour-weather broadcast
     update at start current-weather broadcast
   - in restart command and data
     update as part of restart state
   - in request to broadcast 24 hours of weather data
     update at start 24-hour-weather broadcast
   - in request to start continuous SOS broadcast
     update at start continuous broadcast
   - in request to stop continuous SOS broadcast
     update at stop continuous broadcast
   - in broadcast termination
     update at broadcast termination

3. broadcast-request status
   - in clock pulse
     use to test if 24-hour broadcast has been requested
     update at start 24-hour-weather broadcast
   - in restart command and data
     update as part of restart state
   - in request to broadcast 24 hours of weather data
     use to test if broadcast is under way
     update at start 24-hour-weather broadcast

4. clock time
   - in clock pulse
     update every clock pulse
     use to test if GMT is at the minute mark
     use to test if GMT is at the hour mark
   - in restart command and data
     update as part of restart state

5. location
   - in clock pulse
     use in current-weather broadcast
   - in restart command and data
     update as part of restart state
   - in location data
     update with location data

6. password
   - in start command and data
     use to test if password correct
   - in shutdown command
     use to test if password correct
   - in change password command and data
     use to test if password correct
     update with new password

Figure 10. Encapsulated data analysis table for buoy.

Step 9. Design the clear box. The clear-box expansion of the state machine is quite direct at this point. The responses and transitions in Figure 9 lead directly to a clear box of 14 concurrent black boxes—one for each stimulus type—in which each black box recognizes its own stimulus type in the current complex stimulus and responds accordingly. Certain black boxes must also recognize other stimulus types. For example, the shutdown-command black box must check for the absence of a restart command before shutting down the system. Also, the clock-pulse black box must identify the stimulus type "request to broadcast 24 hours of weather data." These 14 concurrent black boxes are shown as part of Figure 9.

The Resolve black box required for this concurrent clear box must resolve possible conflicts in the broadcast responses and the values set for buoy status, broadcast status, and broadcast-request status. In this case, the conflicts can be resolved as follows:

R: accept any response of change in state data except for response broadcasts, buoy status, broadcast status, and broadcast-request status, which are to be resolved as follows:

response broadcast:
  select in priority order—
    SOS broadcast,
    24-hour-weather broadcast, current-weather broadcast, no broadcast
buoy status: on
broadcast status: based on response broadcast
broadcast-request status: on

Step 10. Verify the clear box. To verify the clear box, the sequential and concurrent process must be eliminated to obtain the derived state box, which then must be compared with the intended state box. The

derivation is quite direct for this clear box because its concurrent black boxes respond to different stimulus type values. The Resolve black box defines the priorities and conflict resolutions among the concurrent black boxes as already identified. Immediate verification again provides direct control over the eventual behavior of the system.

One possible issue here is the responses to other stimulus types at restart or shutdown. The derived state box will provide responses to sensors and various requests for service that may be counter to the spirit of the problem. For example, if a shutdown command and a request to broadcast 24 hours of weather data arrive concurrently, the derived state box may both confirm a shutdown and form and start the broadcast, possibly a questionable response. A review of the intended state box shows that this clear-box behavior meets the state-box requirements. So, the state box itself should be questioned, which leads back to the black box from which the expansion began. In fact, this may be a desirable way to shut down, but it should be resolved and documented in black-box behavior. The stepwise refinement and verification process leaves a design trail for such reconsiderations, with enough documentation to maintain consistency between specifications and design.

Step 11. Repeat stepwise expansion until design is complete. The new common services—24-Hour Weather History, Wind History, Water Temperature History, and Air Temperature History—are also subject to systematic design with the stepwise box-structure expansion process.

For example, in order to relocate weather data into a new abstraction called 24-Hour Weather History, its black-box stimuli and responses must be determined. The abstraction's main purpose is to return a 24-hour weather history on demand for the 24-hour-weather broadcast. Consequently, a query stimulus is needed. Also, weather data including GMT, location, wind average, and air and water temperature averages must be acquired hourly. Call this a data stimulus. And, because the buoy can be restarted, a restart stimulus is also needed. This information is captured formally as follows:

Black-box stimulus types.
(1) restart
(2) data (GMT, location, wind average, air temperature average, water temperature average)
(3) query

Black-box responses.
(1) restart
R: acknowledge restart.
(2) data (GMT, location, wind average, air temperature average, water temperature average)
R: acknowledge data.
(3) query
R: last 24 or fewer records of weather data received since last restart stimulus.

If this expansion were continued, the state data required for the state box would be derived using the necessary and sufficient condition for the encapsulation of history into state. As before, the listing of stimulus types and responses would be expanded one more step. The expansion is simple in this case, but it provides a design trail for engineering inspections as part of the overall design of the buoy.

Stepwise refinement and verification in the box structures of data abstractions provides a systematic discipline for complex system design at any level of formality. Once the black box is understood as a mathematical function from stimulus histories to responses, the derivation of state data requirements becomes a very direct analysis process subject to rigorous engineering inspections. The identification of state boxes to encapsulate state data and processes at the next level is also a very direct process. Since data abstractions are used at the next level, their restatement as black boxes defines their behavior, from which state data and even lower-level box structures can be derived and inspected systematically. Unlike heuristic invention, this derivation is repeatable, allowing engineering inspections because the products and the steps in deriving them are familiar to the inspectors.□

Acknowledgments

It is a pleasure to acknowledge stimulating discussions with Richard Cobb, Alan Hevner, Richard Linger, and David Weiss in the preparation of this paper. The reviewers of this paper also contributed significantly to its quality.

References

1. H.D. Mills, R.C. Linger, and A.R. Hevner, "Box-Structured Information Systems," *IBM Systems J.*, Vol. 26, No. 4, 1987, pp. 395-413.
2. J. Guttag, J. Horning, and J. Wing, *Larch in Five Easy Pieces*, tech. report, DEC Systems Research Center, Palo Alto, 1985.
3. D.L. Parnas, "On a 'Buzzword': Hierarchical Structure," *Proc. IFIP Congress 74*, North Holland, 1974, pp. 336-339.
4. G. Booch, *Software Components with Ada*, Benjamin/Cummings, 1987.
5. D.L. Parnas and W. Bartussek, *Using Traces to Write Abstract Specifications for Software Modules*, University of North Carolina technical report, UNC TR77-012, 1977.
6. C.A.R. Hoare, *Communicating Sequential Processes*, Prentice Hall, 1985.
7. L. Chmura et al., *Software Engineering Principles*, course notebook, Naval Research Laboratory, 1981.
8. R.C. Linger, H.D. Mills, and B.I. Witt, *Structured Programming: Theory and Practice*, Addison-Wesley, 1979.

Harlan D. Mills is a professor in the Computer and Information Sciences Dept. at the University of Florida. He is also director of the Information Systems Institute in Vero Beach, Fla. His current research interests are systems engineering and the mathematical foundations of computer science.

Mills is a member of the US Air Force Scientific Advisory Board, and was a governor of the Computer Society from 1982-84, a regent of the DPMA Education Foundation, and recipient of the DPMA Distinguished Information Science Award in 1985 and the Warnier Prize in 1988.

Mills received his bachelor's, master's, and PhD degrees in mathematics from Iowa State University in 1948, 1950, and 1952, respectively.

Readers may write to Mills at ISI Information Systems, 2770 Indian River Boulevard, Vero Beach, FL 32960.