Alternative Methods of Task Analysis

A Comparison of Three Techniques.

Wellesley R. Foshay
Associate Director, Quality Assurance, Control & Training
Advanced Systems, Inc.
1601 Tonne Road
Elk Grove, IL 60007

Introduction

Task analysis has recently taken on new significance. Instructional developers have long been accustomed to performing a task analysis as part of the front-end work for a project. But recently the term has also gained currency among cognitive psychologists researching human information processing. They tend to use the term in its broadest sense, to refer to the entire analysis performed to specify what is to be learned, after the needs assessment and before the actual design of instruction.

With this broadened definition have come analytical techniques which go considerably beyond the objectives-based behavioral approach with which instructional developers are most familiar. The newer approaches analyze concepts or—most recently—cognitive processes.

These new approaches present a problem for the practicing developer: Should the new techniques be used instead of the established behaviorally based ones, or should an integration be attempted? We will address this question by comparing examples of the three basic approaches and then conclude with some tentative recommendations about when each basic approach is likely to be of greatest use.

Methods Used

To represent behaviorally based analysis, we will use a version of Gagne's learning hierarchy analysis, but we will not employ recently suggested modifications of the technique (Briggs, 1977). Strictly speaking, the learning hierarchy is not inherently behavioral. However, it is perhaps the most widely used technique for determining the structure of a task and an optimal sequence of instruction for that task. The objectives upon which it is based are often derived from behavioral analysis. The more recent additions to Gagne's technique (the five-part objective, analysis of underlying concepts, principles and skills, and information-processing analysis) are omitted here in order to keep the example more "purely" behavioral. These enhancements represent a significant "cognitivization" of the learning hierarchy approach, and thus make it less useful for comparing behaviorally based and cognitively based task analyses.

The concept-based technique we have chosen is concept hierarchy analysis (Tiemann & Markle, 1978; Reigeluth, Merrill, & Bunderson, 1978). Analysis of concepts has taken a number of different forms, ranging from Gilbert's (1962) Mathetics and Thomas, Davies, Openshaw and Bird's (1963) matrix analysis, through M.D. Merrill's (1977) concept elaboration structures. The method of concept hierarchy analysis used here has achieved some currency because it is relatively simple to use, and it highlights the structural characteristics of a concept network in a way that is prescriptively useful.

The information-processing analysis method we have used is P. Merrill's (1978) flowchart adaptation of Scandura's (1976) directed graph technique. This technique attempts to specify with relative precision the cognitive operations involved in solving a particular class of problems. While use of some kind of algorithmic representation is common among information processing cognitive psychologists, the specific cognitive operations and processing parameters incorporated in the analysis vary from one researcher to another depending on the specifics of the cognitive processing model used. These differences generally are not addressed here, in part because the analysis has not been carried to a fine enough level of detail to make the differences relevant.

The task analyzed by these three techniques is an example of complex problem solving: debugging a computer program. This problem was chosen in part because of its familiarity to many developers, and in part because of the belief that it is complex problem-solving tasks such as this that make the strengths and weaknesses of the three approaches most apparent.

Learning Hierarchy Analysis

Following standard practice, the terminal objective has been placed at the top of the hierarchy (see Figure 1). The hierarchy contains four vertical "branches," corresponding to the skills involved in debugging a program under each of the four general conditions in which a program can fail:

1. failure to compile
2. failure to link-edit (if this is done as a separate step), after successful compilation
3. failure to run, after successful link-editing
4. failure to produce expected output at the end of a run (due either to incorrect program structure or incorrect input data)

Note that standard practice for construction of a skill hierarchy calls for branches to occur only when the *skills* required differ, not when the problem structure differs. In this hierarchy, there happens to be a close relationship between causes of failure and skills required for debugging, but the relationship is not perfect. For example, the two types of unexpected output are lumped together because the differences between them usually are not clear at the beginning of the debugging process. Similarly, at the top of the hierarchy, two different causes can require the skill of reading a dump (a listing of the contents of a computer's working memory), so the hierarchy converges at that point.

Note: An earlier version of this paper was presented to The Association for Educational Communications and Technology, 4 May 1982.

At the very bottom of the hierarchy, there are two skills which are among the first applied in the debugging process: determining if there is a problem, and determining the class of the problem (failure to compile, failure to link-edit, failure to run, failure to run correctly). These are placed at the bottom of the hierarchy because they are simplest to learn, although they also happen to come first in the debugging process. This is typical of learning hierarchies: The order in which a task is performed may not necessarily conform to the sequence of instruction, particularly if an instructional strategy such as backward chaining is used.

To simplify the discussion, we will examine in detail only the approach taken in the branch of the hierarchy which deals with failure to produce expected output. The skills involved in this type of debugging are:

1. identifying errors when there is an error message pointing directly to the problem
2. identifying errors when there is an error message which points to a problem only indirectly
3. using trace code when the error message provides only indirect hints, or when there is no error message at all (trace code usually includes extra print statements to print out intermediate values of variables such as loop counters and other state indicators)
4. reading and interpreting a dump

The progression of these skills is from simplest to most complex. Notice that the hierarchy indicates that the instructional sequence includes "bypassing" some of the higher-level skills, such as dump reading. This is because it is possible to debug most problems in this class without using the higher-level skills. For example, a programmer using trace code can debug many programs without also having to read a dump.

As with the other analyses in this paper, this one has been completed at only a relatively gross level. If the analysis of each component skill were completed, conditions and criteria for each objective would be specified. Also, more detailed objectives would demonstrate that the lower-level skills also contain sub-skills which are components of higher-level skills. This would further justify construction of the relationships between the component skills which are shown in the hierarchy.

Concept Hierarchy Analysis

As with the learning hierarchy, the concept hierarchy also is based on the four types of program error situations (see Figure 2). In Reigeluth, et al.'s (1978) terms, this diagram is a "kinds" taxonomy which specifies the four kinds of error conditions which the learner may encounter.

Again to simplify the discussion, only the branch of the hierarchy dealing with unexpected output will be examined. This branch is extended somewhat in Figure 3. Following the taxonomic approach, errors due to incorrect input and errors due to incorrect design are identified as the two kinds of errors resulting in unexpected output. Errors due to incorrect input are further broken down according to each type of incorrect input. If the analysis were complete, this list of types would contain more components. For each type, the analysis would also show the corresponding critical features of the output typically generated: error messages, job log, program output, and dump components.

The branch labeled "how programs process inputs" represents a departure from a pure "kinds" taxonomy. If it were completed, the branch would enumerate principles of program operation which would apply both to programs which operate properly and those which malfunction due to incorrect input. By including such principles, it would be possible to teach the learner to apply them to predict probable outputs under each of the kinds of input error conditions listed in the "kinds" branch of the hierarchy. This, in turn, would make it unnecessary to teach directly the critical features of output generated by each kind of error condition. Instead, it would be necessary to provide practice in applying the principles to predict output generated by each error condition, using an appropriate set of positive and negative examples of outputs. Including principles in a "kinds" hierarchy in this way is unusual, and conventional concept analysis probably would not include it. It has been done here to illustrate one way of simplifying a concept hierarchy by emphasis on principles as well as concepts. It is appropriate to note that use of principles in this way may reflect the influence of an information-processing approach to the analysis.

Information Processing Analysis

The information processing analysis is represented in the form of a flowchart (Figures 4 and 5), following P. Merrill's (1978) suggestion. Representation of cognitive processes by some form of algorithm is common among cognitive psychologists, although other methods also are used. The algorithms are constructed to be consistent with the analyst's model of cognitive processing. Consequently, the set of mental operations and parameters will vary from one analyst to another. At the risk of oversimplifying a complex area of study where consensus among researchers is still a scarce commodity, the following "typical" principles were used in constructing the analysis shown in Figures 4 and 5:

1. Each process shown in the algorithm should be a fairly generic cognitive operation. This analysis presumes operations such as: recalling from long-term memory, holding in short-term memory, perceiving (by each sensory channel), decoding, chunking, discriminating, searching, and scanning.
2. Whatever operations are included, close attention must be paid to the relationships among them. For example, if the model of cognitive processing used postulates a central processor which operates on the contents of short-term memory (STM), then the algorithm must assure the correct information is in STM before each transformation occurs.
3. The algorithm should stay within the limits of human information processors. This includes precautions such as: limit the load on STM to 4 to 7 "chunks," allow for decoding and encoding processes surrounding use of long-term memory, and allow for the relative processing requirements of the various parts of the system. These factors in turn might influence the "chunking strategy" used to accomplish a given sub-task.

Obviously, the algorithm shown in Figures 4 and 5 is not a complete application of these principles. Like the other two analyses, it has been carried only far enough to permit comparison. A fully worked example of information-processing analysis for a skill this complex is beyond the scope of this paper.

The focus of this analysis is on the cognitive processes used by an expert problem-solver. The central thesis of the analysis is that the problem-solver covertly models the program's operation and predicts the program's "normal" output. Discrepancies between this prediction and the actual program output lead to the recognition that a "bug" exists and that the debugging algorithm must be called in. The actual process of debugging then is portrayed as involving

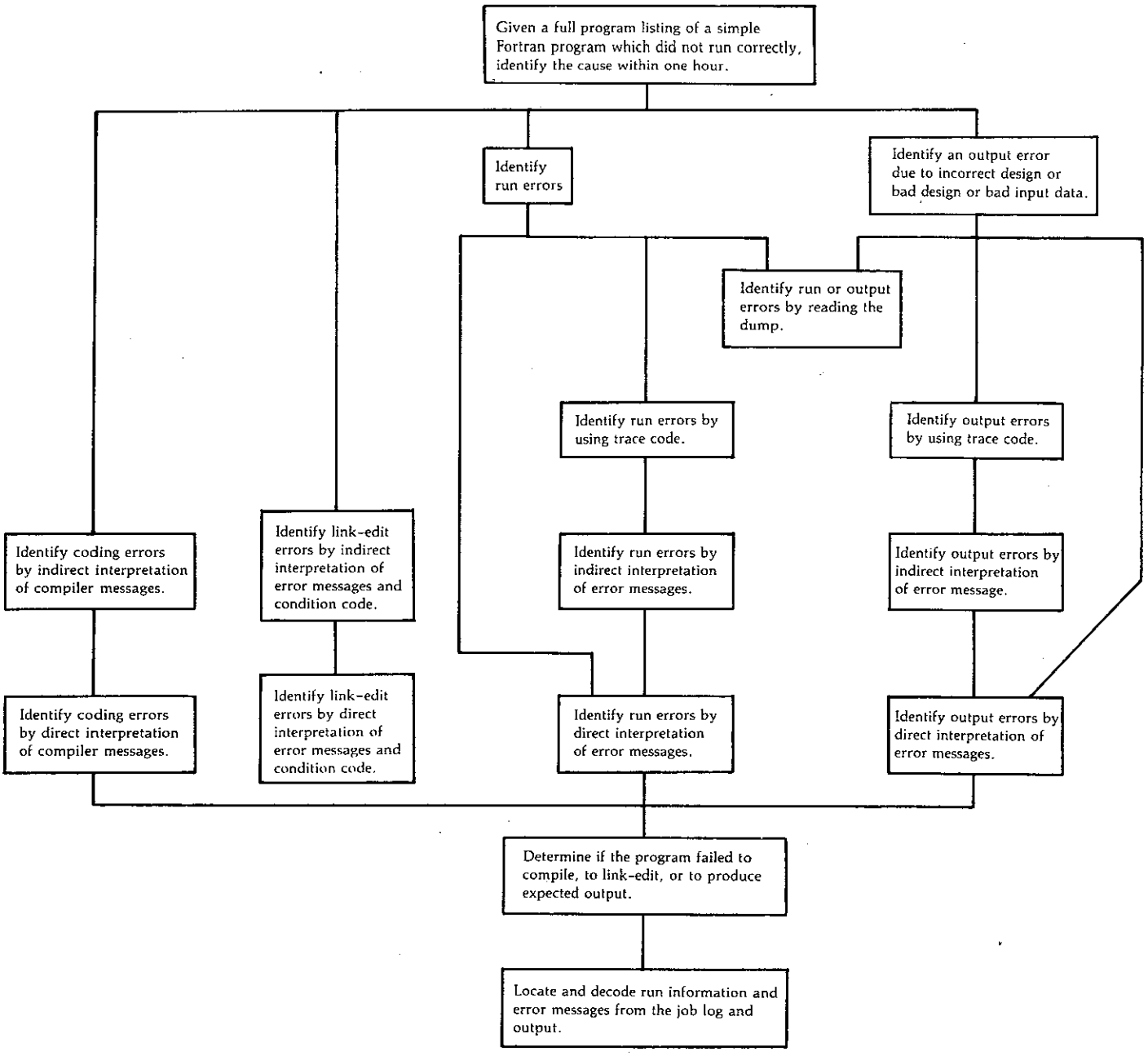

Figure 1: Gagne-style learning hierarchy: Debugging

modification of the learner's internal model of the program by adding hypothetical "bugs" until its predicted operation yields predicted output that matches the actual program output data. This is the general problem-solving model suggested by Braune (1982).

Note that early in the problem-solving process, there is a decision on type of error, as there was in the learning hierarchy analysis. However, the decision occurs early in this analysis because it is done first; in the learning hierarchy, it was included early because it was a simple skill to learn.

Once it has been decided that an error condition exists, the next step is to choose which branch of the debugging algorithm to follow. There is a difference between the branches in this algorithm and those in the two previous analyses. In this analysis, the branches describe different cognitive processes, rather than different problem situations. Thus, unexpected output and failure to run are on the same branch, because the cognitive processes used in debugging them are the same; only the data in short-term memory and from long-term memory change. This kind of simplification of the task analysis is typical of this technique.

To facilitate comparisons with other analyses, only the branch for debugging programs which fail to run or which generate unexpected output will be discussed in detail (see Figure 5).

Once a problem has been encountered, the next step in the debugging process should be to search for error messages. If any are found, they must be decoded. This may involve simple recall of the meaning of the message, or it may involve looking the message up in a reference manual and decoding

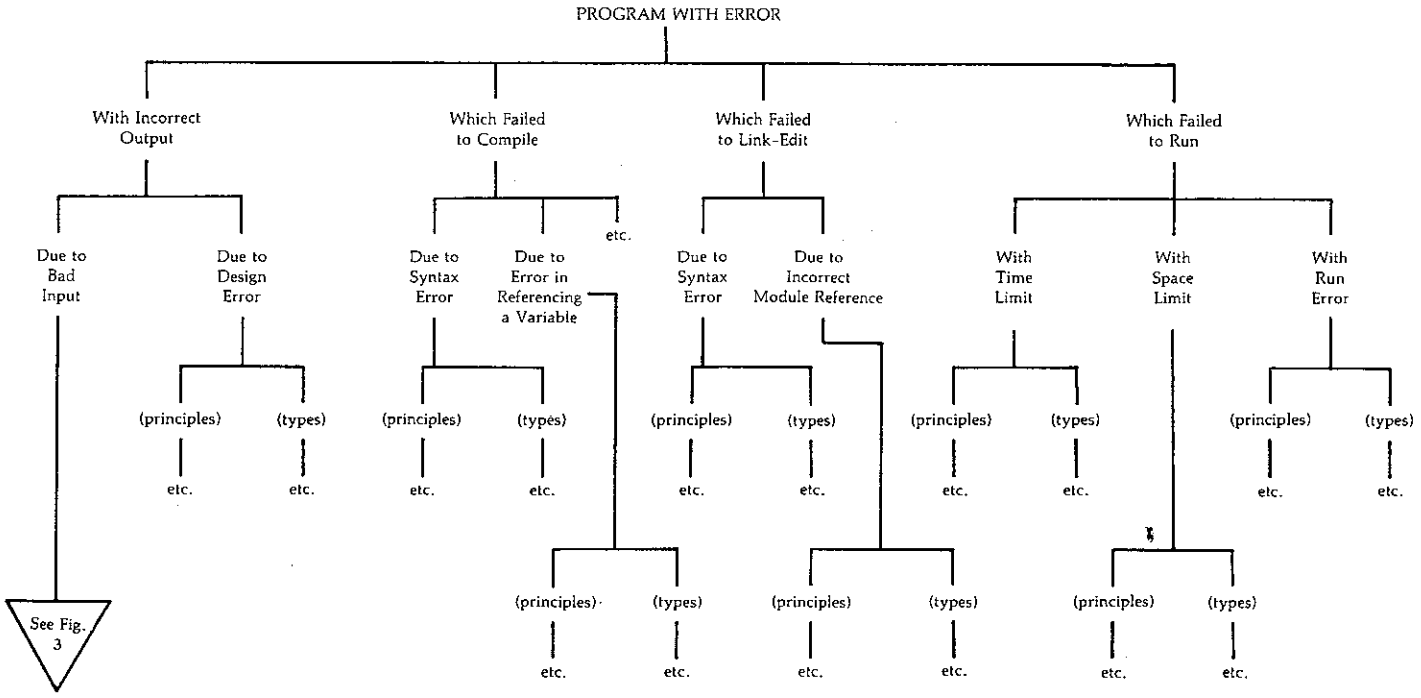

Figure 2: Concept hierarchy (top-level): Debugging

that explanation. Once the meaning of the message has been determined, the next step is for the programmer to internally model the program (including the link-edit structure) by adding bugs to it which are likely to generate the observed error message. When a match occurs between error message(s) predicted from the modified internal program model and the actual error messages observed, then the hypothesis about the cause of the program bug is confirmed.

If no error messages are found, then the debugging process takes a somewhat different course. The actual output generated by the program must be compared in detail with the expected output generated by the programmer's internal model of the program, parts of which must be recalled for the purpose. Discrepancies must be noted and collated. These discrepancies then become the input for the process of revising the learner's internal model. Once the discrepancies have been accounted for, the hypothesis concerning the program bug is confirmed.

With either set of input data (error messages or output discrepancies), the success of the internal modeling process is determined by a number of factors. For example, the degree to which the error messages (or output discrepancies) point to the actual problem significantly regulates the complexity of the task.

Previous experience in modeling for the particular pattern of error messages (or output discrepancies) encountered is also important, because a pattern of cues pointing to a particular error need only be recalled, if it has been previously "solved." An understanding of the structure of the program being debugged has already been mentioned as relevant. In addition, the programmer must know general principles of how programs of that type behave when they function—and when they malfunction. Finally, general problem-solving skills dealing with hypothesis formation and arraying of data to test a hypothesis are also important.

It should be clear that ability to solve problems of this type is controlled by a combination of general problem-solving skills and relatively problem-specific knowledge and experience. This is consistent with research on complex problem solving in other areas, for example, Elstein, Shulman, and Sprafka, 1976. This research also indicates that problem-solvers frequently stop the process of hypothesis generation and testing before all discrepancy data are accounted for. Instead, there is a tendency to systematically ignore or discount data which do not confirm an hypothesis, particularly when a large number of discrepancies exist. The process of developing a hypothesis (by modifying the internal model of the program), predicting its output and comparing that to observed data, is very complex. If this analysis were complete, considerable attention would have to be paid to specifying this process in greater detail.

Comparisons of the Three Approaches

The three analysis approaches differ considerably in techniques for representing the task. The differences of greatest interest relate to type of information identified, macro-level instructional sequence, and micro-level instructional sequence. These differences are summarized in Figure 6.

Type of Information. The type of information made explicit by each technique is different.

The learning hierarchy makes explicit the components of the complete performance, and specifies conditions and criteria for each. Apologists for Gagne might point out that a learning hierarchy analysis could yield much of the same information as the other techniques. For example, it is possible to map concept acquisition tasks into a behavioral analysis by including sub-tasks typical of concept acquisition, using verbs such as "identify," "define," "classify," or "discriminate." Similarly, cognitive processes can be referred to by including tasks with verbs such as "solve," or "predict."

However, it seems unlikely that a conventional behavioral analysis would

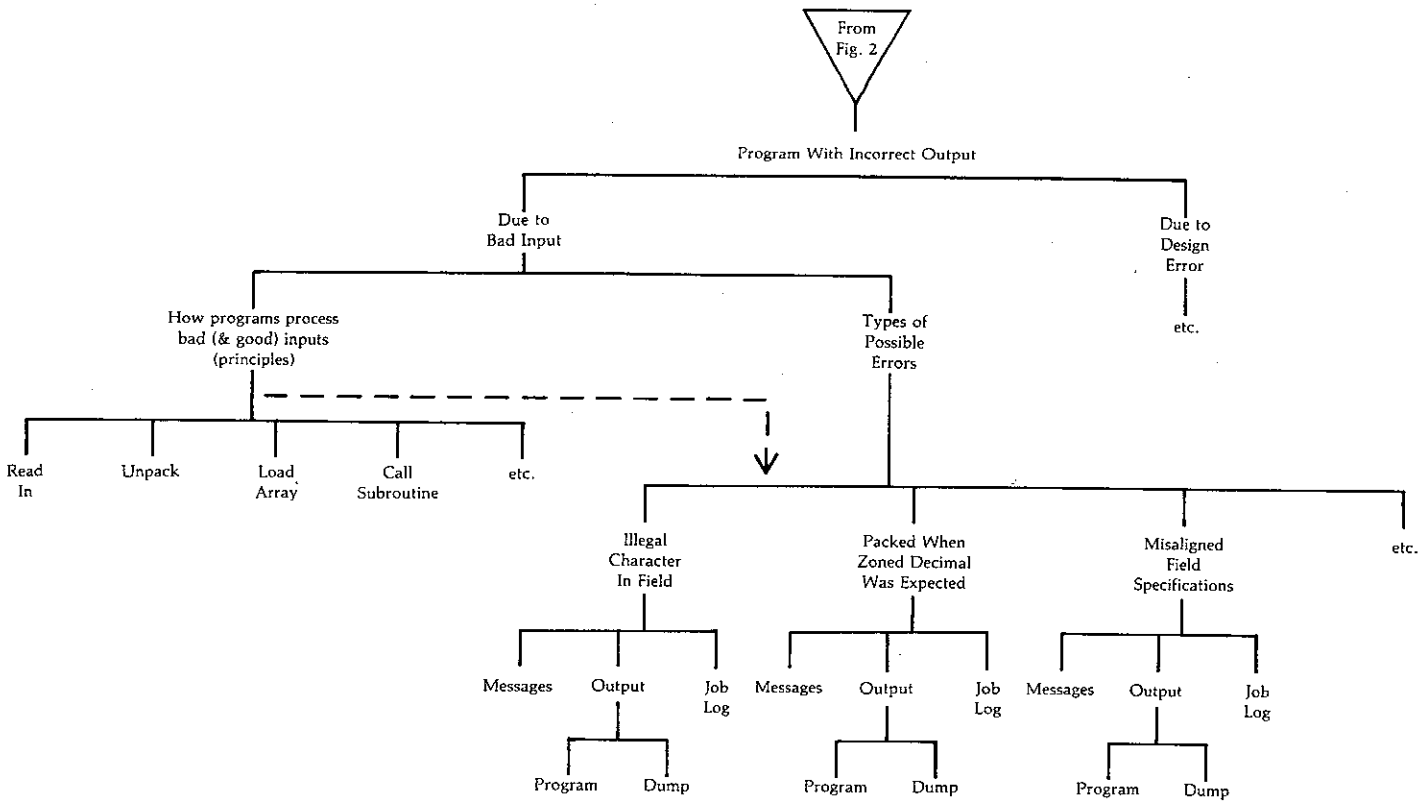

Figure 3: Concept hierarchy (detail: programs with bad input)

identify underlying concept structures or cognitive processes with the precision provided by the other two analyses. When conventional condition-behavior-criterion objectives are used, there is nothing inherent in the learning hierarchy technique that would lead the task analyst to consider explicitly the concepts and cognitive operations involved. Thus, when doing a behaviorally based learning hierarchy analysis, it is *possible* to include behavioral objectives dealing with concept acquisition and cognitive operations, but it is *unlikely* that most task analysts would do this. Instead, the analysis would probably be of the sort done here, with a focus on identifying a set of sub-skills which combine more or less additively into complete skills. The descriptions of skills and sub-skills would probably be in strictly behavioral terms, without examination of cognitive underpinnings.

By contrast, the concept hierarchy analysis explicitly identifies the concepts and the relationships among them (note that this is not true of concept analyses which do not specify a concept structure diagram such as the hierarchy). However, conditions, criteria, observable behaviors, and cognitive skills are implicit, rather than being stated explicitly.

On the other hand, the information processing analysis focuses only on the cognitive skills, although relevant *bodies* of concept knowledge frequently are identified, but not structurally analyzed. Conditions for performance of the skill are identified very carefully, since they constitute the set of input data for the cognitive operations mapped. However, criteria of performance typically are not specified, except for the implication that successful completion of the algorithm is the desired outcome.

Macro-Level Sequence. Additional differences can be identified at the "macro" level of instruction sequencing, that is, from the student's entry level to full attainment of the global objective. The learning hierarchy is the only one of the three methods of analysis which directly suggests an instructional sequence at this level. Indeed, the principal reason for constructing a learning hierarchy is to determine the instructional sequence, which is always from the bottom of the hierarchy to the top.

By contrast, concept hierarchies present no such clear picture. Some authors (Reigeluth, Merrill, Wilson, & Spillner, 1980; Engelmann, 1980) argue for a top-down instructional sequence in all cases. However, there is not a universal consensus. Tiemann and Markle (1978), for example, make no such recommendation. In the program debugging example, a top-down sequence would be one alternative, but other sequences are also possible. For example, a "middle-out" sequence might be used, in which the designer would start by teaching debugging in one class of problems and then generalize progressively to other classes of problems, concluding with an overview of the taxonomy of problem situations. Conceivably, the sequence in which the concepts are presented might also be influenced by individual differences variables such as previous knowledge or cognitive style.

The information-processing analysis also presents no clear guidelines for instructional sequence. Very little research has been done on how problem-solving structures as complex as the debugging example are acquired. If we apply Scandura's (1976) framework to this example, we probably would *not* attempt to teach debugging in procedural order, using either forward or backward chaining. Scandura's argument is that complex problem solving rests on acquisition of a series of insights about the structure of the problem space which can not be comprehended by the novice. In this ex-

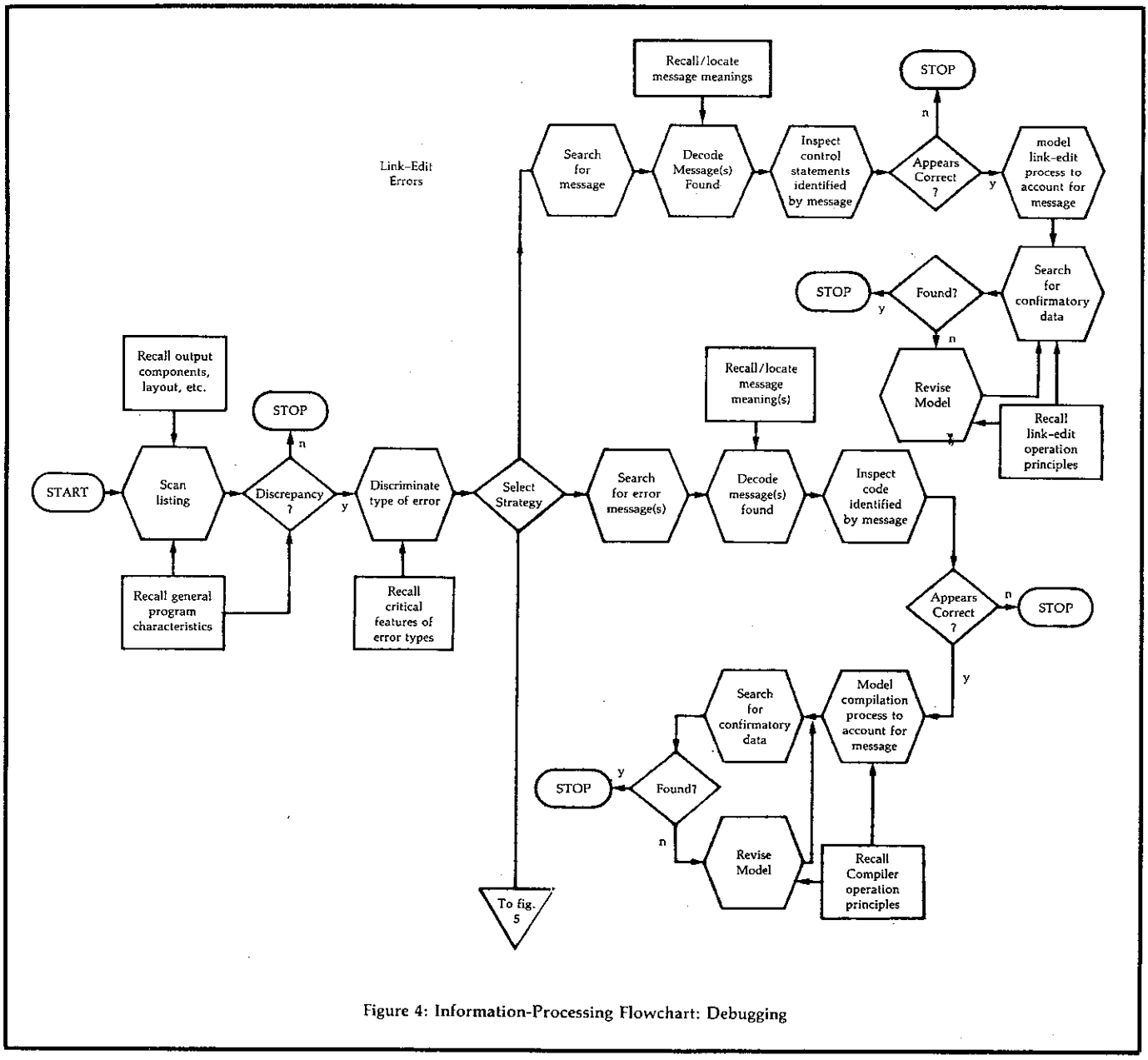

Figure 4: Information-Processing Flowchart: Debugging

ample, such insights probably surround the ability to predict program function and malfunction. Scandura's principles might suggest a carefully-structured sequence of debugging problems which grow in complexity. Each increment in complexity would be designed to cause the learner to acquire more sophisticated insights about how to predict the function and malfunction of programs with different structural characteristics. Then the procedure used by the novice would differ substantially from that used by the expert. Instructional tasks would take this into consideration.

"Micro" Level Sequence. None of the three methods of analysis directly represent prescriptions for design and sequencing of instruction for a single concept or sub-skill. However, strong prescriptive principles are commonly applied to the learning hierarchy and to the concept hierarchy analyses. Prescriptive principles for information-processing analyses are less well-defined, but the analysis does make possible some useful insights in design.

For learning hierarchy analyses, the most familiar prescriptions are summarized in Gagne's Events of Instruction model (Gagne and Briggs, 1979), although practically all of the literature on instructional design could be applied to a task analysis done in this framework.

The concept hierarchy also is associated with strong prescriptive principles. At least three procedures for concept teaching have been published (Tiemann & Markle, 1978; Merrill, Richards, Schmidt, & Wood, 1977; Engelmann, 1980). All three procedures provide highly specific guidelines for construction and use of concept definitions, sequences of positive and negative examples, and testing sequences. The level of precision in these procedures is much greater than those derived from typical behavioral analyses.

Prescriptive use of information processing analysis is not far advanced. Development projects using this tech-

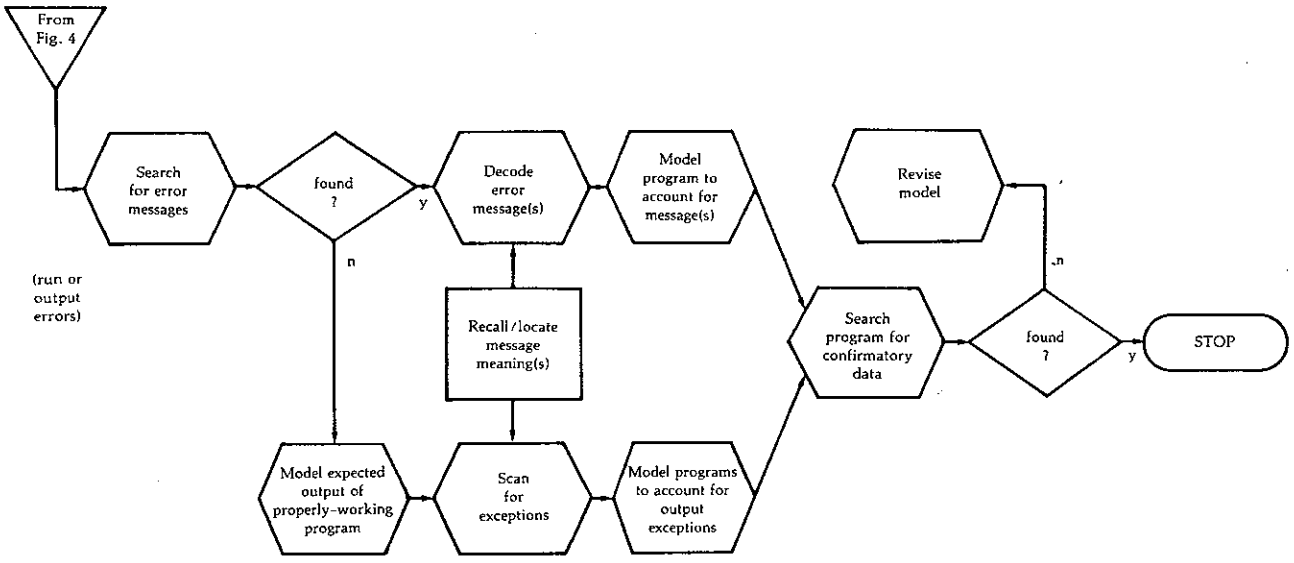

Figure 5: Information-Processing Analysis (output or run errors)

nique are becoming increasingly common, but a set of coherent design principles which are grounded in research has yet to emerge. This does not mean that information processing analysis has no prescriptive utility. The algorithm derived from the analysis may be directly presented to the student in some form. Furthermore, the analysis appears to have great potential in providing detailed guidelines for design of test items, instructional problems and dialogs. In particular, the ability to make inferences about the state of the student's learning from errors committed, is attractive (Scandura, 1976).

Conclusions

The three techniques chosen for this comparison are representative of three major approaches to task analysis: behaviorally based, concept-based, and cognitive process-based. It should not be concluded that the three analysis approaches represent incompatible extremes of a theoretical or ideological continuum. Instead, this paper has demonstrated that they differ only in what is made explicit and what is left implicit in the task being analyzed. If this is the case, the techniques associated with each basic approach are probably better viewed as variety of "lenses," each with a different kind of "filter" through which to view the task. The "lens" and "filter" the developer chooses will depend on the nature of the subject matter and the effect desired.

The techniques vary in the precision with which they can be applied. However, all the methods involve considerable intuitive judgment on the part of the analyst, particularly in projects involving real instructional problems. Practical application of the techniques must, therefore, constantly raise the question of the cost of the analysis versus prescriptive benefits gained. The outcome of this consideration will vary considerably by situation, and it further precludes universal application of any one technique.

Recommendations

The example analyses presented here may be representative of a number of instructional situations involving troubleshooting complex systems, such as electronic devices, or living organisms. Such troubleshooting is one general class of complex problem-solving. It may be that in situations involving complex problem-solving the differences between the techniques become clearest. In any case, the examples presented here lead to four tentative recommendations for use of the techniques. Such recommendations can only be validated, of course, by considerable experience with each technique.

1. Use learning hierarchy analysis if "macro"-level sequencing and measurement problems are paramount, and supplementary analyses will suffice for identifying underlying concepts, principles and skills.

This recommendation is based on the strength of the learning hierarchy as a tool for sequencing instruction, and the strength of the instructional objective as a means of identifying conditions and criteria.

2. Use concept analysis if discriminations and relations among concepts are the most difficult part of problem-solving, but the cognitive processes involved are fairly simple.

This recommendation is based on the strength of concept hierarchies as a means of identifying concepts and concept/example sets for teaching relational structures among concepts.

3. Use concept hierarchy analysis if specification of conditions and criteria of performance are impractical.

In many situations, a behavioral analysis requires strong assumptions about conditions and criteria, because the analyst cannot predict specifically how learners will use the knowledge, or because the range of application situations is too broad or complex to serve as a useful guideline for task analysis. In these cases, concept hierarchy analysis may be a more suitable way of achieving precision in the analysis.

For example, in a recent analysis of diagnostic tasks in a medical subspecial-

ty (Foshay & Hatch, Note 1), a concept hierarchy provided the potential for elegant and precise prescriptions for instructional design while bypassing the problem of developing objectives for the entire range of possible diagnostic problems present in the subspecialty.

4. Use information-processing analysis if cognitive operations involved in problem-solving are complex, especially if detailed design of feedback or error analysis is a design goal.

This recommendation is based on the strength of information-processing analysis as a means of determining the details of the problem-solving process. It has particular importance for designers of simulations, games, and computer-based instruction. At this time there is, however, little consensus on guidelines for taking an information-processing analysis to a finer level of detail than that shown here. For the instructional developer, this means that information-processing analysis is probably more intuitive than would be desirable.

Of course, the techniques may be used in combination, depending on the analyst's judgment of the instructional problem. Given the state of the art, practitioners who work beyond the laboratory cannot afford the luxury of an ideological commitment to either behavioral or cognitive analysis. When dealing with complex instructional problems, it is probably best to retain an eclectic orientation, and learn to combine specific analytic techniques based upon the information generated. For this reason, case studies of such applications are needed if guidelines are to be derived for using the techniques together.

	Type of Information	Macro-Level Instructional Sequence	Micro-Level Instructional Sequence
Learning Hierarchy	condition, observable behaviors, criteria	bottom-up	events of instruction model
Concept Hierarchy	concepts & relationships	none (or top-down)	concept presentation
Information Processing	cognitive skills	progressive complexity	algorithms; error analysis

Figure 6: Summary of features of the three approaches to task analysis

Reference Note

1. Foshay, W. and Hatch, T. Memorandum on Application of a Concept Analysis of Gastroenterology. University of Illinois, School of Clinical Medicine at Urbana-Champaign, April, 1981.

References

Braune, R. *Towards an Internal Model of Pilot Training*. Masters Thesis, University of Illinois, Champaign-Urbana, 1982.

Briggs, L.J. (ed.) *Instructional Design*. Englewood Cliffs, N.J.: Educational Technology Press, 1977.

Elstein, A., Shulman, L., & Sprafka, S. *Medical Problem-Solving*. Cambridge, Mass.: Harvard University Press, 1976.

Engelmann, S. "Toward the Design of Faultless Instruction: The Theoretical Basis of Concept Analysis" *Educational Technology*, February, 1980, 28-36.

Gagne, R. and Briggs, L. *Principles of Instructional Design*, 2nd. ed. New York: Holt, Rinehart and Winston, 1979.

Gilbert, T. "Mathetics: The Technology of Education." *Journal of Mathetics, 1*, 1, 1962, 7-73.

Merrill, M. "Content Analysis via Concept Elaboration Theory." *Journal of Instructional Development, 1*,1, Fall, 1977, 10-12.

Merrill, M., Richards, R., Schmidt, R., & Wood, N. *The Instructional Strategy Diagnostic Profile Training Manual*. San Diego, Calif.: Courseware, Inc., 1977.

Merrill, P. "Hierarchical and Information Processing Task Analysis: A Comparison." *Journal of Instructional Development, 1*,2, Spring, 1978, 35-40.

Reigeluth, C., Merrill, M., & Bunderson, C. "The Structure of Subject Matter Content and its Instructional Design Implications." *Instructional Science, 7*, 2, 1978, 107-126.

Reigeluth, C., Merrill, M., Wilson, B., & Spillner, R. "The Elaboration Theory of Instruction: A Model for Sequencing and Synthesizing Instruction." *Instructional Science, 9*, 3, 1980, 195-219.

Scandura, J. *Problem Solving*. New York: Academic Press, 1976.

Thomas, C., Davies, I., Openshaw, D., & Bird J. "Programmed Learning in Perspective: A guide to Program Writing." Chicago, Ill.: Educational Methods, Inc. 1963. Cited in Davies, I. *Competency Based Learning: Technology, Management, and Design*. New York: McGraw-Hill, 1973.

Tiemann, P., & Markle, S. *Analyzing Instructional Content: A Guide to Instruction and Evaluation*. Champaign, Ill.: Stipes Publishing Co., 1978.