Modelling Cognition
Edited by P. Morris
© 1987 John Wiley & Sons Ltd.

5

Initial skill learning: an analysis of how elaborations facilitate the three components

DAVIDA H. CHARNEY and LYNNE M. REDER
Carnegie-Mellon University

1. INTRODUCTION: SKILL LEARNING AND THEORIES OF COGNITION

Understanding the processes by which people learn is fundamental to any theory of cognition. Accounting for learning adds constraints to theories of cognition; not only must a theory account for adult capacities, but it must also posit mechanisms for acquiring new capabilities as adult learners do. Although the study of how information is acquired has been central in memory research, only a few theories of cognition, problem-solving and the like (e.g. Anderson, 1983; Kieras and Polson, 1985; Hayes and Simon, 1974) have been concerned with specifying how a skill or procedure is initially acquired.

It seems natural and desirable to try to apply what we have discovered about acquiring factual information to the study of how people learn new skills or procedures. However, while the findings from the memory domain are certainly relevant to the study of skill acquisition, they fall short of what is needed. At the very least, the standard performance measures for fact learning (e.g. recognition judgments, binary-choice decision tasks and recall protocols) are inappropriate for measuring skill learning, which requires the learner to *apply* his or her knowlege. Consider a budding scientist who has studied inferential statistics in order to determine the reliability of experimental results. A fair test of how well this student has learned the various statistical tests is not whether she can recall their formulae, but rather whether she can select the appropriate test and use it correctly to analyze the data.

Presented with two samples to contrast, the student must remember that there is a test called the Student t-test and decide that it is an appropriate test for the data at hand. Then she must retrieve the formula for the test, find and plug in values for the variables in the formula correctly, and solve for the correct result.[1]

This example illustrates that although skill learning and fact learning both involve the acquisition of new information, they differ in the types of information that must be learned and the ways in which the learner uses the information. The first obvious difference between fact learning and skill learning is that a skill has an *execution component* that is quite specific to that skill and the requirements for execution vary considerably from skill to skill. The output for many skills comes in the form of complex motor activity, such as pressing a certain sequence of keys or fitting together the parts of a device. In contrast, the output modes for demonstrating mastery of a fact are general, simple and well-learned: the learner vocalizes or writes the fact that is retrieved from memory or signals whether the retrieved representation meets some criterion (e.g. recognition or paraphrase match).

One reason for the variation is that skills are often built up out of component skills, which are called on in the manner of 'subroutines'. For example, performing a t-test involves at least two component skills: finding the mean and the standard error of a sample. A different procedure might involve finding a different mean, but using some of the same values to determine the error term. The relationship between facts in a domain seems qualitatively different from the relationship between subroutines embedded in a procedure. While it is common to 'unpack' a concept by retrieving related concepts, the process is seldom as routine and unvarying as performing a fixed subroutine.

Another difference between skill learning and fact learning is that the context in which a fact is retrieved can facilitate access to the learned information. A learner demonstrates fact learning by retrieving a fact in response to a query. The query not only provides a retrieval cue, but it also provides an appropriate occasion for retrieving that particular fact. In other words, a person can be considered to have mastered a fact if he or she can recall it when specifically queried. But recall of procedures when queried is not sufficient for mastery of a skill. Knowing how to perform a skill requires that the learner understand and appreciate the contexts in which a particular procedure is appropriate. In the case given above, deciding to use the t-test rather than some other test depends on knowing something about the function of the t-test and something about the data to be analyzed. Granted, it is just as important to know when to use a fact as when to use a procedure.

[1] Remembering that the name of the test is 't-test' is not critical in this case, but in other skill tasks (such as using a computer or constructing a proof), remembering the name of a procedure can be quite important.

The difference is, however, that learning a skill means knowing when to use the acquired procedures but learning a fact does not. A test of skill learning should measure the learner's ability to choose procedures appropriately.

In addition to differences in what must be learned, there are also differences in how we view 'mastery' of a fact versus a skill. Although facts can vary in learnability and strength, we usually do not judge 'how well' a fact is recalled when it is recalled. In contrast, procedures not only vary in ease of learning, but most require practice for any degree of competence to be attained. That is why in skill domains, we classify practitioners as experts, novices or intermediates. We do not say that someone is 'skilled' in a particular domain until she can execute the procedures rapidly and rather effortlessly. A person who is slow to execute basic procedures or who rehearses the requisite steps 'declaratively' is usually judged to be a novice.

Because skill learning involves different output requirements and different standards for proficiency, we need new, more sensitive measures in order to study skill learning. But, more importantly, we need to consider closely the cognitive mechanisms of skill learning and how they interact with those of fact learning. If skill learning draws heavily on declarative knowledge, then we might expect factors that affect encoding, retention and retrieval from declarative memory to be important for skill learning. Conversely, if skill learning and fact learning involve largely independent processes, then we might not expect conditions that facilitate fact learning to have much benefit for skill learning.

One model of cognition that carefully considers where declarative knowledge interacts with procedural knowledge is Anderson's (1983) ACT* model. Anderson follows Fitts (1964) is positing that learners initially acquire a skill in declarative form, usually from oral or written instructions. Procedural knowledge of the skill, in the form of a production system, arises only after hands-on practice. At first, in order to approximate the required skill behavior, the learner uses a set of general-purpose productions to retrieve segments of the declarative representation of the instructions, and translate them into a series of actions. With additional practice, the learner gradually constructs a set of skill-specific productions that directly incorporate the relevant declarative knowledge, eliminating the extra step of retrieving this information from declarative memory. As the productions are compiled and tuned, skill performance improves dramatically; performance becomes much more efficient and requires much less conscious attention.

In this model, declarative knowledge becomes increasingly superfluous to skill performance as learners gain expertise, but the declarative representation is critical to the initial stages of skill learning. We might expect, then, that factors influencing the formation of the declarative representation would strongly influence initial skill performance. In particular, the form of the verbal instructions on which the declarative representation is based is clearly quite important. If the learner cannot extract an adequate declarative representation of what to do from the instructions, it is unlikely that he or she will be able to approximate the skill, except perhaps through trial-and-error or some other problem-solving heuristic. Further, since the initial productions are based on the learner's early approximations of the skill performance, the form of the verbal instructions may critically influence what the skill-specific productions look like. Surprisingly the issue of how verbal instructions influence the initial acquisition of procedures is only beginning to receive attention in the literature (e.g. LeFevre, 1985; Kieras and Polson, 1985; Hayes and Simon, 1974).

This chapter, then, is concerned with the initial stage of cognitive skill acquisition; that is, how a novice learns a skill well enough to use it. We mean to distinguish this stage, in which the learner acquires the bare essentials of a cognitive skill, from later stages in which the learner becomes proficient. Theories concerned with proficiency (e.g. the dynamics of speedup with practice) have been developed by a number of researchers (e.g. Anderson, 1982; Newell and Rosenbloom, 1981; Rosenbloom and Newell, in press; Schneider and Shiffrin, 1977; Shiffrin and Dumais, 1981; Shiffrin and Schneider, 1977). Rather than trying to understand the development of cognitive expertise, our goal is to explore more carefully the requisite components of initial cognitive skill acquisition and what features of the initial verbal instruction facilitate each component.

We will begin by outlining three components that we consider crucial for initial skill learning. Then we will describe three types of elaborations that we believe facilitate learning and illustrate the ways in which they facilitate the components of skill learning. Finally, we will summarize our view as to where elaborations are beneficial and where they are not.

2. A TRIPARTITE MODEL OF COGNITIVE SKILL ACQUISITION

We conceive of initial skill learning as consisting of three critical components:

(a) learning novel concepts and the functionality of novel procedures;
(b) learning how to execute the procedures;
(c) learning the conditions under which a procedure is applied; and remembering the best procedure to execute in a given situation.

In other words, learning a skill means knowing what procedures exist for accomplishing various goals, knowing how to carry out the procedures, and knowing under what circumstances to apply them (including remembering to use them when the situation warrants). Each of these components can be learned independently and each component can be a 'bottleneck' to acquiring a skill. Furthermore, the relative importance of the components may vary, depending on the type of skill being learned: assembling a piece of equipment, operating a device, using a computer system, solving problems. This section will consider the requirements of each component in more detail.

2.1. Learning novel concepts and the functionality of procedures

For someone learning an entirely new skill, the idea of what kinds of things can be accomplished and what objects the procedures act on may be entirely unfamiliar. Consider someone who is learning to use a computer text-editor (e.g. EMACS) for the first time. A proficient typist who has never used a word-processor will consider it a novel concept to insert a word into a string instead of erasing a line and retyping it. Features such as automatic line wrapping, multiple windows into the same file, keyboard macros and kill buffers are other concepts that will be entirely new. In addition to novel concepts such as these, the novice user must also find out what things can and cannot be done with a text-editor. That is, the learner needs to know what distinct procedures (in this case, what commands) are available in the text-editing system and understand what each one does.

It is important to distinguish between learning a procedure's functionality and acquiring a 'mental model' of the system that uses these procedures. Learning the concepts and the function of individual procedures does not entail acquiring a mental model of a system (i.e. its components and how they interact). Although a *sophisticated* understanding of the concepts and procedures may involve the construction of a mental model, a mental model is not always necessary for proficient skill performance. For example, many experienced drivers have little more than a crude idea of how a car works. The value of a mental model to the novice learner depends on the type of skill being learned. As Kieras (1985) points out, a mental model of how a device works is only likely to improve performance if the procedures for operating or assembling the device can be inferred from knowing how the parts interact. In a computer-operating system, the syntax and the names of commands are oftentimes chosen arbitrarily, so having a mental model of how a computer works is not likely to help people learn and remember how to execute the commands. On the other hand, when the goal is to *trouble-shoot* the system, having a mental model can be very helpful. That is, when the computer is not responding in the expected way, it is much easier to diagnose the trouble if one has a mental model of what the computer does under various circumstances.

2.2. Learning how to execute new procedures

It is often crucial in skill learning to remember fairly arbitrary associations of objects and functions (e.g. which button on a control panel produces the desired state) and to be able to reproduce exact sequences of symbols and actions. In this respect, skill learning is quite different from fact learning. When seeking declarative information, people are usually highly tolerant of gist reporting, paraphrasing and even slips of the tongue. It is irrelevant for most purposes whether a fact presented in one syntactic form (e.g. passive) is stored or retrieved in another (e.g. active). In contrast, there is low tolerance for such variation in most cognitive skill domains. For example, computers cannot commonly recognize a wide range of synonyms or abbreviations of crucial terms. In fact, computers are notoriously 'literal-minded': when a user presses the wrong key, the computer at best responds with an error message, and at worst performs an undesired action. The computer has little or no capacity to infer what key the user intended, even if the key he typed had a similar label or was physically near the correct key. Given the current technology, learners must strictly adhere to the vocabulary and syntactic rules of whatever system, language or program they are using. Equally strict conditions on the sequence of operations and the use of symbols are imposed in many noncomputer skill domains.

The complexity of the execution component depends on the nature of the procedure and the skill domain itself. In many domains, the task of learning to execute a procedure consists of learning three elements:

(a) the name of the procedure;
(b) the rule describing a sequence of operations; and
(c) the method of binding variables in the rule to objects in that problem space.

These three elements can be illustrated most clearly in the case of computer commands. Suppose that a learner is given the task of renaming a file. To execute the procedure, the learner must first remember the name of the command (e.g. RENAME). Then he must reproduce the sequence of arguments that the RENAME command requires (e.g. type the name of the command, then the current name of the file, then the desired name of the file). In this sequence, the name of the command (RENAME) is a constant term, but the present and desired names are variables. The learner must determine the values of these variables for the present situation, and plug the values into the correct places in the sequence.

The first and third elements, learning the name of a procedure and learning to bind variables, are not required in all skills. They are most often required in skills like using a computer that involve a set of general, multi-purpose procedures. Whereas the procedures in a computer manual can be used over and over in novel combinations to accomplish a wide variety of goals, the goal is fairly fixed in an assembly task or a device operation task. For example, when one is learning to put together a stereo phonograph system, there is a specific, known object that a closed set of pieces is going to form. In this case, the descriptions of the procedures can be completely explicit, naming the

exact parts that are involved at each stage. Each step in the assembly is performed only once. Under these circumstances, there are no variables and the procedures need not be named.

The second element, the rule or sequence of operations, can take many different forms. Commonly, the sequence of operations comes in the form of an ordered list such as a cookbook recipe, each step of which may refer to distinct subprocesses (e.g. sautéing vegetables as part of a recipe for making spaghetti sauce). However, a list format is not appropriate to all skills. Mathematical formulaes and computer commands have a formal syntax which may be expressed abstractly in the form of a rule or template. To illustrate the diversity of these abstract rules, we provide three examples below. The first is a formula for the *t*-test for single mean (compared to a specified constant). The second is the syntactic rule for the command to rename files on the IBM-PC. The third is a template for defining functions in the computer language LISP:[2]

$$t = \frac{X - \mu}{s/\sqrt{N}} \qquad (1)$$

where *s* is an estimate of the population standard deviation.

REN[AME] [*d:*][*path*]*filename*[.*ext*] *filename*[.*ext*] (2)

```
(DEFUN ⟨function name⟩                                         (3)
       (⟨parameter 1⟩ ⟨parameter 2⟩ . . . ⟨parameter n⟩)
       ⟨process description⟩
```

To execute the procedures for the *t*-test, one finds values for all of the variables mentioned in the rule, performs the arithmetic calculations signaled by the mathematical symbols and solves for a numeric solution. This procedure is quite different from the use of a syntactic rule or template. The sequence of operations for generating a RENAME command or defining a LISP function from a template is to type a sequence of symbols that matches the template in structure, in which each constant term (e.g. DEFUN) and each symbol (e.g. parentheses, colons, spaces) appears in the appropriate location and each variable is replaced by the appropriate value.

The degree to which the rule for a procedure must be internalized depends on the task. For many skills, executing the procedures should be automatic. For example, once learners know how to use a computer text-editor, they use the manual mainly to learn new features or to solve some unexpected problem or for an occasional reminder. They should not have any difficulty executing the standard set of commands without much conscious attention. On the other

[2]The rule for the rename command was taken from the official Disk Operating System (DOS) manual (Anonymous, 1983). The LISP template is taken from Winston and Horn (1981).

hand, factors such as the number and complexity of the procedures, their importance or their frequency of use may require learners to depend on written instructions each time they perform a task. For example, airplane pilots review printed check-lists each time they fly.

2.3. Learning conditions for application

As argued above, having a skill means knowing when to apply particular procedures. In some cases, the conditions for application are perfectly straightforward. For example, when the procedures come in a strictly ordered sequence, the precondition for any given procedure is the result of correctly executing the previous procedure. The conditions for application may be much more complex, however. In skills such as using a computer, there may be multiple ways to accomplish a goal. Under certain conditions, one procedure might be much more efficient or advantageous than the rest. If novices do not appreciate the conditions under which the procedures are most useful, they might overlook procedures that would be very useful to them. Instead, they might always choose to use some inefficient procedure that they happened to learn first or that may initially be easier to remember.

Card, Moran and Newell's (1983) study of experienced users of text-editing system demonstrated that experts have well-defined rules for selecting between procedures. The text-editor involved offered two basic methods of moving the cursor: searching for a specified string of characters or moving the cursor up or down a line at a time. The subjects had consistent strategies for choosing between these methods (e.g. use the search method if the target location is more than three lines away, use the line-feed method otherwise). Presumably, these computer users developed their strategies themselves, but their early learning was not observed. At least some of the subjects had developed fairly inefficient strategies. For example, one subject never used the string search method; she used some variation of the line-feed method even when the target location was over ten lines away.[3]

If we wish learners to use a repertoire of procedures appropriately, it may be necessary to *motivate* the use of some procedures by demonstrating the advantages they have in particular situations. Acquiring a repertoire goes beyond the ability to decide between specified procedures on demand. Even if a person can, when queried, consistently judge which of two procedures is more efficient in a given context, he may not always select the most efficient one in *real* situations. The learner may fail to ask (or be unwilling to ask) for each subgoal, 'which procedure is optimal here?' Computing the relative costs

[2] Card, Moran and Newell successfully modelled how the experts used selection rules, but they were not interested in the relative efficiency of the rules their subjects had come up with nor in how the subjects had acquired their rules.

of procedures can be time consuming and tedious, and not without its share of the costs. Therefore, unless a procedure easily 'comes to mind', it may remain unused. This means that skilled performance involves not only the ability to recognize the situations in which a given procedure is optimal, but the ability to retrieve the best procedure easily and rapidly when necessary.

It is worth noting that learning sophisticated selection strategies is probably unnecessary for skill tasks such as learning to assemble a device or operate a piece of equipment. Since the procedures in these tasks are less multipurpose, they are also less interchangeable. It is more likely that the conditions for application in these tasks grow out of ordering constraints rather than considerations of efficiency.

2.4. The independence of the three components of skill learning

The three components of skill learning that have just been described are fairly independent of one another. A learner may know that a specific procedure exists for solving a problem without knowing or remembering how to apply it. For example, a child knows what shoe-tying is and when it needs to be done, but lacks the ability to carry out the procedure. Similarly, by rote learning, one may learn to perform a series of steps without knowing what the steps are for. Finally, one may understand what a procedure does and how to carry it out, but not know when or why to use it. This situation often arises when novices consult computer manuals: they finish reading a description of a command, understand more or less what it does and how to issue it, but lack the slightest inkling as to when they would ever want to use it or how it relates to other commands they have learned. Since each of the three components is necessary to skill learning, each can constitute a 'bottleneck' for acquiring a skill.

3. THREE TYPES OF ELABORATIONS FOR FACILITATING SKILL LEARNING

The question we address in the remainder of this chapter is how the presentation of information in an instructional text can facilitate learning in the three components just described. We will focus on a particular aspect of instructional texts, namely the degree to which the main points are elaborated.

The effect of elaborations on the acquisition of information from a text has been the topic of considerable speculation and research (Reder, Charney and Morgan, in press; Anderson and Reder, 1979; Reder, 1976, 1979, in press; Weinstein, 1978; Mandl and Ballstaedt, 1981; Mandl, Schnotz and Tergan, 1984; Bransford, 1979; Chiesi, Spilich and Voss, 1979; Craik and Tulving, 1975). In the view of most researchers, there are several reasons why elaborations should help subjects learn and remember the main ideas of a text. Elaborations provide multiple retrieval routes to the essential information by creating more connections to the learner's prior knowledge. If one set of connections is forgotten, it may be possible to retrieve the desired information another way. Further, if the learner forgets an important point, it may be possible to reconstruct it from the information that is still available. Not all the evidence on elaborations is positive, however. Reder and Anderson (1980, 1982) found that elaborations can impair learning and retention of the main points of a textbook chapter as compared to studying a brief summary of the main points. In contrast, Reder, Charney and Morgan (in press) found that when the goal is to use the information in a skill-learning task, elaborations can improve performance.

This section defines the essential characteristics of three types of elaborations that we think are especially important for skill learning: analogies and two types of examples (simple instantiations and situation examples). After these types of elaborations are defined, the next section will describe how they may specifically contribute to skill learning.

3.1. Analogy

An analogy draws a comparison between a concept that a person wants to learn and concept in a different domain that the learner is already familiar with. In Gentner's (1983) terminology, the former is the 'target concept' and the latter is the 'base concept'. Gentner proposes that the quality of an analogy depends on what type of information can be mapped from the base to the target construct. Good analogies map across relationships between objects rather than specific attributes of objects. For example, in the familiar analogy between the solar system and the structure of the atom, the attributes of the sun (HOT and YELLOW) are not mapped to the nucleus of the atom. What is mapped is the relationship of the sun to the planets (i.e. the sun is MORE MASSIVE THAN the planets and the planets REVOLVE AROUND the sun).

We suspect that analogies reduce the processing load during learning by facilitating chunking of the information in the target domain. That is, the structure of concepts and relations from the base domain can be used to provide temporary labels on components of the target idea while the problem or task is being solved. Since the base labels are well understood, the pointers to the relevant structure(s) in memory are not lost while the learner works through the critical new aspects of the target domain.

3.2. Exemplification

Like analogy, exemplification involves a mapping between two concepts, but the mapping is more tightly constrained. We will follow Hobbs (1978) in

defining exemplification as a relationship between a rule (or generalization) and a specific instance (or example) for which the rule holds true. The rule and the example are related by sharing the same underlying proposition. In other words, the relationships beween objects in the general construct must map across to the specific construct. Unlike analogy, however, exemplification also constrains the mappings of object attributes. The objects in a rule are abstract, general categories. The objects in an example are more specific members of those general categories. That is, an example can be constructed by substituting one or more specific, concrete terms for general terms in a rule. To see how this works, consider the following generalization–example pair (taken from Charney, 1985):

> Lawsuits are now pending which seek to hold handgun manufacturers and distributors liable for the damage caused by their products. (4)

> The family of James Riordan, a Chicago police officer killed by a handgun, is suing Walther, the West German maker of the gun and International Armament Corporation, its American distributor. (5)

Statement (4) is a generalization that asserts the existence of a new type of lawsuit, initiated against manufacturers and distributers of handguns by people who have been hurt by the handguns. Statement (5) is an example of this generalization. It asserts the existence of a particular lawsuit, initiated against a specific manufacturer (Walther), and a specific distributor (the International Armament Corporation), by the Riordan family, who were hurt when James Riordan was killed by a handgun.

The class/member constraint on examples leads to an important difference between analogy and exemplification. The base and target concepts in an analogy come from different domains: a familiar base and an unfamiliar target. In exemplification, on the other hand, the rule and the example are both from the same domain, the domain of the skill to be learned. Since both constructs are relatively unfamiliar, there is the danger that the learner will not understand the rule well enough to make the appropriate mapping to the example.

Despite this danger, exemplification may aid learning in several ways. First, seeing typical objects that the rule might operate on can clarify the general terms in the rule. The general terms are linked to more concrete and specific concepts. Second, seeing a variety of examples and counterexamples can help the learner define the scope of the rule's application (Nitsch, 1977; Tennyson, 1973; Tennyson, Woolley and Merrill, 1972). Third, examples may have an important role for establishing the validity or the utility of a rule (Perelman and Olbrechts-Tyteca, 1969; Schoenfeld, 1979; Mandl, Schnotz and Tergan, 1984; Gilson and Abelson, 1968). Finally, learners can use examples of correctly solved problems as models for solving new problems (e.g. Anderson, Sauers and Farrell, 1984).

With this last use of examples (serving as models for solutions to new problems), the boundary between analogy and exemplification begins to blur. On the one hand, using examples as models is analogic in that there is a specific-to-specific mapping between the constructs (the example and the new problem). Furthermore, as we would expect in an analogy, the 'base' example is more familiar than the target problem by virtue of having been seen or worked on before. On the other hand, unlike analogy, the examples that are used as models are often introduced to the learner in the context of a general principle of rule. Further, the model problems come from the same unfamiliar domain as the novel problems they are mapped to. Consider, for example, the problems that are laid out in the course of a mathematics chapter. A general formula or algorithm is exemplified with specific problems that can then be used as models for the chapter-end exercises. Anderson, Farrell and Sauers (1984) explicitly combine general-to-specific and specific-to-specific mappings in their analysis of subjects learning to write LISP functions. The subjects were presented with an abstract template for defining a function in LISP along with an example of a correct function definition. On the basis of protocol analysis, Anderson *et al.* conclude that subjects first make the exemplification mapping between the template and the example, then analogize between the example and the new problem. In spite of the dual nature of this type of learning, we will use the term exemplification whenever a general principle or rule or procedure is instantiated with a specific example within a prescribed domain, even when we assume that the example is later used as a model.

3.3. Situational examples

We will single out *situational examples* as an especially rich type of example. Situational examples differ from other kinds of instantiations in that they illustrate the contexts in which a procedure applies rather than simply illustrating the details of how to execute an abstract procedure. The distinction we are drawing between simple instantiations and situational examples is similar to that drawn by Mandl, Schnotz and Tergan (1984) between 'illustrative examples' and 'application examples'. Both types of examples provide specific instances of the general terms of a rule, but the types differ in what other kinds of information they provide.

To illustrate the two kinds of examples, consider the following three sentences. The first is a rule from a text for teaching students to improve their writing style (Williams, 1981). The second simply instantiates the concepts in

the rule, and the third is a situational example:

> When a nominalization follows an empty verb, change the nominalization to a verb that replaces the empty verb. (6)

> For example, nominalizations such as *investigation*, *inquiry* or *response* often follow empty verbs such as *make* or *conduct*. Use the verbs *investigate*, *inquire* or *respond* instead. (7)

> For example, change the sentence 'The police conducted an extremely thorough investigation into the incident', to 'The police investigated the incident extremely thoroughly.' (8)

The example in statement (7) instantiates the general terms 'empty verb' and 'nominalization' but does not provide a context in which they might occur. The situational example (statement 8) instantiates the general terms within a specific context. The context illustrates something about the situations in which the rule should apply: the nominalization need not follow the empty verb directly. It also illustrates something about how to carry through the solution: changing a noun to a verb can necessitate changes to other parts of the sentence.

If instantiation is the major contribution of an example, then both types of examples should aid performance to the same degree. But if it is important to use the example as a model or to motivate the use of a procedure, then seeing a situational example should improve performance more than seeing a conceptual example. Situational examples may also help people remember a rule when they are working on a task, because seeing the task may remind them of the example (Ross, 1984). Finally, situational examples may be better for demonstrating the utility of the rule, by showing rather than just asserting that following the rule leads to a desirable outcome.

4. THE INTERACTION OF ELABORATION TYPES AND THE COMPONENTS OF SKILL LEARNING

In this section, we recapitulate the three components of skill learning and analyze the potential benefits that specific types of elaborations may have for learning a specific component. In particular, we claim that situation examples are the most useful for skill learning because each example can contribute to learning in all three components. On the other hand, while analogies can be constructed to illustrate each component, they are more likely to help people learn the functionality of a procedure than how to execute it or when to select it. We will begin by briefly discussing the role of analogies in skill learning, describing their benefits and limitations. Subsequent sections will show in more detail how examples can contribute to the three components of skill learning.

4.1. The role of analogies in skill learning

Initially, one might expect analogies to be the most helpful form of elaboration for learners since, unlike examples, they involve constructs in a base domain that are highly familiar to the learner. As such, analogies may be very useful for clarifying unfamiliar concepts. Suppose a novice computer user is learning to use a personal computer with floppy disk drives. The following analogy can help the user anticipate some features of diskettes:

> A diskette is similar to a small, flexible phonograph record, except that instead of storing sounds, it contains information that the computer can read, add to or delete. (9)

By mapping information from his previous knowledge of phonograph records, the user may anticipate that diskettes are used in a horizontal orientation and that it is unwise to let the surface of the diskette become dirty or scratched. The usefulness of the analogy is somewhat limited, however. Learners may draw spurious assumptions from the analogy, e.g. that information is stored on a diskette linearly, as it is on a phonograph record. Or they may assume that, like phonograph records, diskettes must be removed from their protective covering when they are used.

Analogies may also be constructed for motivating the use of some procedures. One procedure that many computer users must learn is how to specify the location of the file or directory they want to work on. The following analogy attempts to motivate the choice between two options for specifying the 'path' through a directory structure. The choice arises in the DOS operating system because paths optionally begin with a backslash symbol (\). The backslash signals that the path is to start at the top-level (or 'root') directory. If the backslash is omitted, the path is assumed to start at the current directory (which may or may not be the top-level directory). This analogy compares the specification of a file in a directory structure to dialling a local or long-distance telephone call.

> When should you specify a path that begins at the root directory? It may be useful to draw an analogy to using the telephone to make long-distance calls. When you are calling a number within the current area code, you do not have to dial or specify your own area code. You just dial the number you want. This is like leaving off the first backslash in a path; the computer assumes you want to stay within the current directory. However, if you want to call someone outside the current area code, you dial '1' and then the new area code and then the number. The '1' that you dial first is analogous to the backslash for the root directory, and the new area code is analogous to th[illegible] of another subdirectory where the files (or phone numbers) are stored. (10)

This analogy clarifies the concept of a path and the appropriate circumstances for starting the path at the root directory. Again, however, the analogy is fairly fragile. For instance, the computer allows you to 'overspecify' the location of a file, but the phone company does not. That is, you can specify a path to any file starting at the root directory, even a file in the current directory. This would be analogous to dialling '1' and your own area code. The analogy also breaks down in a very common circumstance: when the current directory is the root directory and you wish to specify a path to a subdirectory within it, no backslash is needed. If all exceptions have to be explained to the user, the analogy may be more trouble than it is worth.

A more important drawback of the analogy is that it does not help the learner acquire the particular procedures needed for using the operating system. That is, knowing that an area code is analogous to a directory name does not help the learner master the system-specific conventions for specifying a path through a directory structure. So even if the analogy were more robust, learners would still have to rely on other means to learn how to execute the procedures.

As this discussion illustrates, it is possible to construct analogies for various components of skill learning, but the benefit of analogies is limited. Analogies can clarify unfamiliar concepts and procedures, but often cannot hold up at the level of detail to which learners must understand and apply the concepts. Analogies are probably least appropriate for elaborating on the execution component of a skill. We believe that examples drawn from the domain under study, though less familiar than the base domain of an analogy, are more useful to the learner in the long run. If examples are carefully constructed, they may simultaneously clarify the function of a procedure, how to execute it and when to select it.

4.2. The role of examples for conveying functionality and motivating procedures

Below we introduce two new concepts from computing to illustrate how situational examples cán motivate us well as explain a concept or procedure. To give more force to our claim that rich situational examples are best for introducing or teaching these concepts, we will present the concepts first with the impoverished examples and then with richer ones.

4.2.1. Command editing

At the time of this writing, using a command editor within an operating system is a relatively novel concept, even for people familiar with computers, since this feature is not available on many operating systems. Command editing refers to the ability to retrieve commands that were already issued to the system and then use them again, either reissuing them verbatim or issuing a modified (edited) version of the command. The functionality of command editing can be conveyed with rather straightforward examples; however, if the examples do not illustrate the *motivation* for its use, learners will not appreciate the feature and when it is most useful. Consequently, they will not use the procedures regularly and will probably forget all about them fairly rapidly.

Consider the following example that explains what command editing means, but fails to motivate its use (the example pertains to a modified VMS operating system):

> Suppose you have typed the following sequence of commands into your computer— (11)
>
> $dir
> $finger
> $go .chap
>
> The first command in this sequence produces a listing of the contents of the current directory; the second, a listing of the people currently using the machine, and the last requests that the current directory be changed to a subdirectory called 'chap'.) Now suppose you want to list the contents of the 'chap' subdirectory. Using command editing, you press the up-arrow a few times so that *$go .chap*, then *finger*, then *$dir* appear on the command line. Now you need only press the return key to reissue the *$dir* command for the .chap subdirectory.

The above example is sufficient for explaining the functionality of the command editing procedure, but is poor for motivating its use in that it provides little or no savings in keystrokes over typing a new *dir* command. In such an example, the usefulness of the procedure is obscured and no novice would see the need to spend time learning it. The following example, in contrast, should make much clearer the usefulness of the command editor.

> Suppose, for example, you want to copy a number of files from someone else's account on another system. To copy the first file (called 'draft1.mss'), you must specify a long path to the relevant directory in your friend's account on the other machine. (12)
>
> $copy cmpsyb::[wells.papers.curchap]draft1.mss *.*
>
> Suppose you want to copy another file called 'final.mss' from

> that same location. One way to do so would be to type a new copy command that would look exactly the same as the command above, except that 'final.mss' would appear in place of 'draft1.mss'. However, retyping the command will require 58 keystrokes for each file you want to copy. And if you enter the command with unnoticed typing mistakes in the path or the filename, you will have to type the entire command again. Command editing saves you all of this retyping. Instead of retyping, you simply 'recall' the last copy command and edit it to change the name of the file. Typing the up-arrow key brings back the last command. Then, by striking the left-arrow key, you can move the cursor leftward to the specific characters in the filename that must be changed. When you are finished changing the name of the file, press RETURN to issue the modified command. By editing and reusing your first copy command, you save nearly 40 keystrokes for each file you have to copy and you reduce the chances of error in retyping the whole command.

Both examples clarify the concept of reissuing a command. By specifying the sequence of keys that must be typed, both examples also instantiate the rules for executing the command-editing procedures. Only the situation example, however, illustrates the conditions under which the command-editing procedure is more desirable than the procedure for issuing a new command. In particular, command editing is worthwhile when you must type a number of long, similar commands. We will return to the issue of learning when to use a procedure in section 4.4 below.

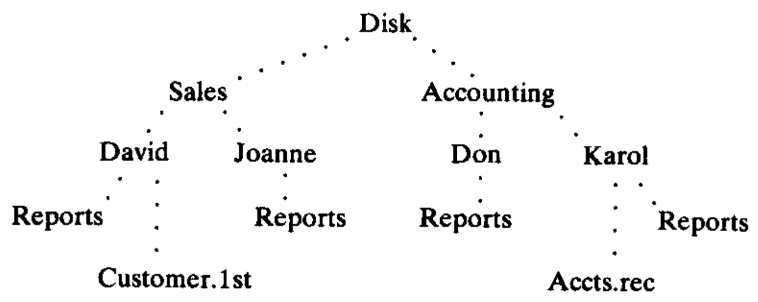
4.2.2. Subdirectories

Consider another example from the same general domain. We have mentioned subdirectories in the course of the preceding example. This construct may also be unfamiliar to many readers of this chapter. Simply stating that a directory can be divided into subdirectories is sufficient for 'explaining' the concept, but it is unlikely that the user will be sufficiently motivated to acquire the cluster of skills needed to make use of such a facility.

The following example, taken from the IBM DOS Manual (Anonymous, 1983), illustrates the concept more fully along with some rationale for the usefulness of subdirectories, but adds little to the reader's sense of how subdirectories might facilitate day-to-day activities on the computer.

> DOS Version 2.00 gives you the ability to better organize your disk by placing groups of related files in their own directories —all on the same disk. For example, let us assume that the XYZ (13)

company has two departments (sales and accounting) that share an IBM Personal Computer. All of the company's files are kept on the computer's fixed disk. The local organization of the file categories would be viewed like this:



With DOS Version 2.00, it is possible to create a directory structure that matches the file organization. With this ability, all of DAVID's report files can be grouped together in a single directory (called REPORTS), separated from all the other files on the disk. Likewise, all of the accounts receivable files can be in a unique directory, and soon.

The example above implies that subdirectories are only useful when different people are using the same disk. Even then, it does not illustrate the advantage subdirectories have over a single directory for any given user. In order to convey the usefulness of subdirectories, the learner might be shown a 'one-level' directory filled with many unrelated files and told to imagine trying to find a file for which the name can be recognized but not easily recalled. The figure below is an example of a listing of files in a flat or one-level directory.

```
Directory _CMPSYB::PSY$USER:[ANON]

2APA.REF;6         2SCS.DAT;1         ABHRTR3.MSS;7      APAREF.LIB;1
CAU.MSS;15         CHP2BIB.AUX;1      CHP2BIB.MAK;3      CHP2BIB.MSS;10
CHP6BIB.MAK;3      CHP6BIB.MSS;12     CHP7BIB.MAK;1      CHP7BIB.MSS;14
CIANCI.MSS;3       COLDSA.DAT;1       COPING1.DAT;1      DEBBIB.AUX;1
DEBBIB.MSS;3       DEBBIB.OTL;1       DOCU.MDR;1         DOCU.MSS;1
EMACSINIT.;1       FILE2.OUT;3        FMC.DAT;1          FORM.ERR;2
FORM.LET;1         FPRO.DAT;2         FRDPRO.LNO;1       FRDPRO.MSS;1
FREUD.LNO;2        FREUD.MSS;13       FREUD2.LNO;1       FREUD2.MSS;1
GPSCIBIB.AUX;3     GPSCIBIB.MSS;4     GRADE.BAS;1        HEARTBIB.AUX;1
HEARTBIB.MAK;10    HEARTBIB.MSS;19    HEARTBIB1.MSS;3    HRTBIBADD.MAK;3
HRTBIBADD.MSS;2    IMP.FRM;2          IMPFRM.MSS;8       INST.MSS;1
INTERR.SPS;1       JPAREV.MSS;1       JUNK.DT;1          JUNK2.DT;1
LMFF.MSS;1         LOGIN.COM;6        LOT.OTL;1          LOT.OU1;1
LOT2.OTL;10        LOT2R.OTL;5        LOTA.MSS;8         LOTREF.AUX;19
LOTREF.MSS;18      LOTREF.OTL;14      LOTTAB4.MSS;13     MAIL.MAI;1
```

```
MALIAS.;4          MBOX.;2            METHOD.MSS;4       MFSBIO.MSS;9
MIKE.MSS;2         MIKE.OU1;1         MYAPA.LIB;2        MYAPA.REF;2
MYREG.CON;1        MYREG2.CON;1       NETLOGIN.COM;2     NETSERVER.LOG;15
NEWSCS.DAT;2       NOTES.MSS;2        NYEAR.;1           P1.MSS;30
P2.MSS;51          P3.MSS;11          PCORR.SPS;1        PFU.MSS;20
PILL.CON;1         PILL.DT;1          PILL2.DT;1         PLAN.;2
POSURG.DAT;1       PRE2.MSS;4         PREFACE.MSS;3      PRETEST.MSS;9
PROO.DAT;1         PROT.DAT;1         QUES.;1            QUEST.;1
QUEST.MSS;6        RA.MSS;1           RALOT.SPS;1        REG.MSS;1
REV1.;1            SA3.DAT;1          SA3.SRT;3          SAMPLE.MSS;1
SAREF.MSS;3        SAREF.OTL;3        SCHBIB.MSS;28      SCORES.DT;1
SCRIBE.LOG;1       SCRIPT.MSS;1       SCS.MSS;4          SECOND.QUE;6
SEVEN.QUE;4        SF2F.SPS;3         SHVITA.MSS;31      SHVITA1.MSS;1
SIG.LET;1          SIGMA1.MSS;1       SIGNUP.;1          SIX.LPT;1
SIX.QUE;1          SOCANX.DAT;1       SOCSUP.MSS;3       SPSS.TXT;1
SPSS1.TXT;1        SPSS2.TXT;1        STATE.;2           SUM.MSS;1
SUM1.MSS;4         SUP.;1             SUPPORT.MSS;9      TA.;3
TAB7.MSS;3         TABLE1.MSS;1       TABLE2.MSS;1       TABLE4.ERR;1
TABLE4.MSS;1       VCR.;1

Total of 134 files.
```

Often people forget the exact name they gave to a file. They correctly believe that they can often recognize the name from a complete listing of the file names; however, in a directory such as the one above, as many as 134 filenames might have to be inspected. Contrast the directory listing above with the one below:

```
Directory _CMPSYB::PSY$USER:[NEAT-NIK]

COURSES.DIR;1      EMACSINIT.;1       EXPER.DIR;1        INFO.DIR;1
LOGIN.COM;57       MALIAS.;6          MAIL.;17           MISC.DIR;1
MSS.DIR;1          PLAN.;1

Total of 10 files.
```

Each of the subdirectories (i.e. the entries with dir suffixes) may contain files and still deeper subdirectories (e.g. under 'courses' are subdirectories for specific courses, and under 'exper' are subdirectories with the data and materials for specific experiments.) Searching for a particular file may still take some looking around; however, assuming the user knows the category of the file, the number of individual filenames that must be inspected is much smaller.

By contrasting the situations for finding a file with and without subdirectories, the pair of example directories above clarifies what subdirectories are, as well as motivating the circumstances for their creation (e.g. whenever a user has a large number of files that an be categorized fairly easily.) Carefully constructed situational examples can thus illustrate both why the feature is useful and when it is most appropriate or efficient to use.

4.3. The role of examples for learning how to execute procedures

We believe that examples can give the learner the most concrete, most specific picture of exactly what to do while executing a procedure and making the necessary adjustments to specific task situations. The procedures that benefit most from exemplification are those that involve the interpretation of a general rule. An unelaborated rule, with its special notation, variables, symbols, general terms, etc., is usually too abstract for learners to comprehend. In this section, we will describe various ways in which examples can facilitate execution, including clarifying the spirit of a rule and providing a model for future solutions.

4.3.1. Learning to integrate a collection of operations: command editing

Command editing, a procedure introduced in the last section, is not a difficult concept to grasp; however, whether or not it is easy to execute depends on the nature of the system implementation. Both the DOS and VMS operating systems have command-editing features, but the implementations differ in several important respects, such as providing external cues to relevant operations. For example, on the IBM-PC, the key that recalls the previously issued DOS command is the F3 button. Of the ten function keys on the keyboard, there is no obvious reason why the desired key should be F3. Once the previous command is recalled, the user may edit it using a key labeled 'Ins' that toggles the system between insert and overwrite modes, and a key labeled 'Del' that puts the machine into delete mode. In contrast, it may be easier to remember how to initiate command editing in VMS because there are up-arrow and down-arrow keys that are uniquely associated with going back over a buffer of previously issued commands. Once a command is recalled, however, it may be harder to remember how to toggle from insert mode to overwrite mode because there are no overt function keys; the relevant sequence of keystrokes is the non-mnemonic control-a.

Command editing thus consists of a collection of operations for viewing and retrieving items in a buffer and changing the mode or state of the computer. The operations themselves are fairly simple: usually consisting of a single keystroke. As a result, the execution component of command editing is not easily described with a general rule; rather, learners must remember all the component operations and determine how to sequence them. Situational examples that show learners a complete interaction should be very valuable.[4]

The most difficult aspect of learning to execute this type of procedure may be remembering the arbitrary association of a key and a function. An additional benefit of examples may be to strengthen memory traces for such associations through repetition in a concrete context.

[4]The situational example about command editing presented earlier (example 12) provides much of the necessary description. To have fullest effect, the editing operations should probably be described more fully and should be set off from the body of the text.

4.3.2. Learning to generate instances of a rule: renaming files

Among the types of procedures that are hardest to learn to execute are those that require the learner to generate a particular instance of an abstract rule. Examples of these procedures were provided earlier: learning to perform a *t*-test, learning to issue computer commands, or defining a function in a programming language. As discussed above, executing this type of procedure requires that the learner remember the name(s) of the procedure, the details of the rule or sequence of operations and how to assign values to any variables that appear in the rule.

Examples can help people learn this type of procedure in several ways. Consider the following typical example that was intended to help learners parse a rule, in this case, a rule for renaming files on the IBM-PC. In the manual, the example in (15) follows the general rule in (14).

REN[AME] [*d*:][*path*]*filename*[.*ext*] *filename*[.*ext*] (14)

For example, the command:

REN B:ABODE HOME (15)
renames the file ABODE on drive B to HOME.

The example clarifies some notational aspects of the rule. Elements that appear in square brackets in the rule are optional; in the example, the last three letters of the name of the command and the path are omitted. One problem with the example is that it does not clarify under what conditions the optional elements can be omitted. A series of situational examples that contain different combinations of optional elements might be necessary to illustrate these points. The example does begin to illustrate the distinction between constant terms and variables. Elements in the rule that are printed in italics are variables. In the example, the italicized elements have been replaced. The *d*: is replaced by B: and the first instance of *filename* is replaced by ABODE.

One serious problem with this example is that the filenames ABODE and HOME do not seem very typical of real filenames and, more importantly, they do not signal which is the *old* name and which is the *new* name of the file. If learners have trouble figuring out and remembering the order of the arguments in the rule, remembering this example is unlikely to help them. Some manuals attempt to solve this problem with examples like the following:

RENAME OLDFILE NEWFILE (16)

While the ‘filenames’ in this example do signal the function of the arguments, they are far from typical examples of filenames. Since real filenames do not typically refer to functions in rules, using this type of example may ultimately confuse the learner. The filenames are poor illustrations of what the ‘fillers’ of the argument slots may look like. The following situational example is better:

> Suppose you have a file called BUDGET that contains your budget for 1986. Now you want to create a new budget for 1987, but you need a way to keep the files for the two years distinct. The command: (17)
>
> RENAME BUDGET BUDGET.86
>
> changes the name of the existing file BUDGET to BUDGET.86. Now you can create a file for the new budget called BUDGET.87, and it will be easy to distinguish the two files.

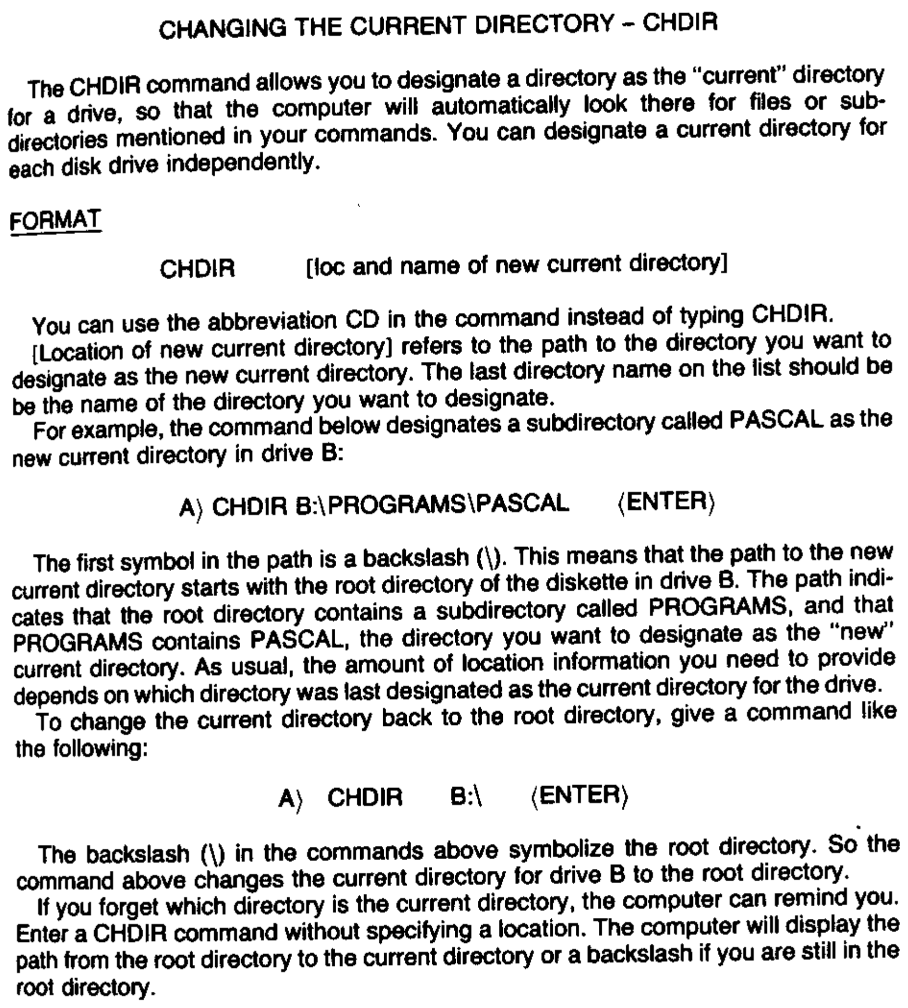
In addition to clarifying aspects of notation, this example also clarifies the functions of the two ordered arguments or parameters in the rule: the first is the old name and the second is the new name. The example also helps to motivate the use of the RENAME command, by presenting a situation in which renaming a file makes sense. As mentioned above, additional examples of the same sort may be needed to illustrate other aspects of rule.[5]

The results of Reder, Charney and Morgan (in press) suggest that rich examples of correct commands help people learn to generate their own commands. Indeed, elaborations on the execution of procedures proved to be more important to learners than elaborations on the function and motivation of the commands. We systematically varied whether or not a computer manual contained *syntactic* elaborations (e.g. examples of syntactically correct commands to illustrate more abstract rules for the commands) or *conceptual* elaborations (e.g. analogies illustrating the basic concepts, examples of situations in which a command would be useful). Factorially combining the two types of elaborations produced four versions of the manual. Figures 1 and 2 are corresponding excerpts from two of the manuals, describing the CHDIR (‘Change Directory’) command; Figure 1 contains just conceptual elaborations and Figure 2 contains just syntactic elaborations.

[5]The rule itself, taken from the DOS manual, is not very informative about the function of the arguments (or parameters). The following statement of the rule might be better:

RENAME *[location and current name of file]* *[new name of file]*

Research suggests, however, that even this form of the rule benefits from exemplification (Reder, Charney and Morgan, in press).



CHANGING THE CURRENT DIRECTORY - CHDIR

The CHDIR command allows you to designate a directory as the "current" directory for a drive, so that the computer will automatically look there for files or subdirectories mentioned in your commands. You can designate a current directory for each disk drive independently.

FORMAT

CHDIR [loc and name of new current directory]

You can use the abbreviation CD in the command instead of typing CHDIR.
[Location of new current directory] refers to the path to the directory you want to designate as the new current directory. The last directory name on the list should be be the name of the directory you want to designate.
For example, the command below designates a subdirectory called PASCAL as the new current directory in drive B:

```
A) CHDIR B:\PROGRAMS\PASCAL     (ENTER)
```

The first symbol in the path is a backslash (\). This means that the path to the new current directory starts with the root directory of the diskette in drive B. The path indicates that the root directory contains a subdirectory called PROGRAMS, and that PROGRAMS contains PASCAL, the directory you want to designate as the "new" current directory. As usual, the amount of location information you need to provide depends on which directory was last designated as the current directory for the drive.
To change the current directory back to the root directory, give a command like the following:

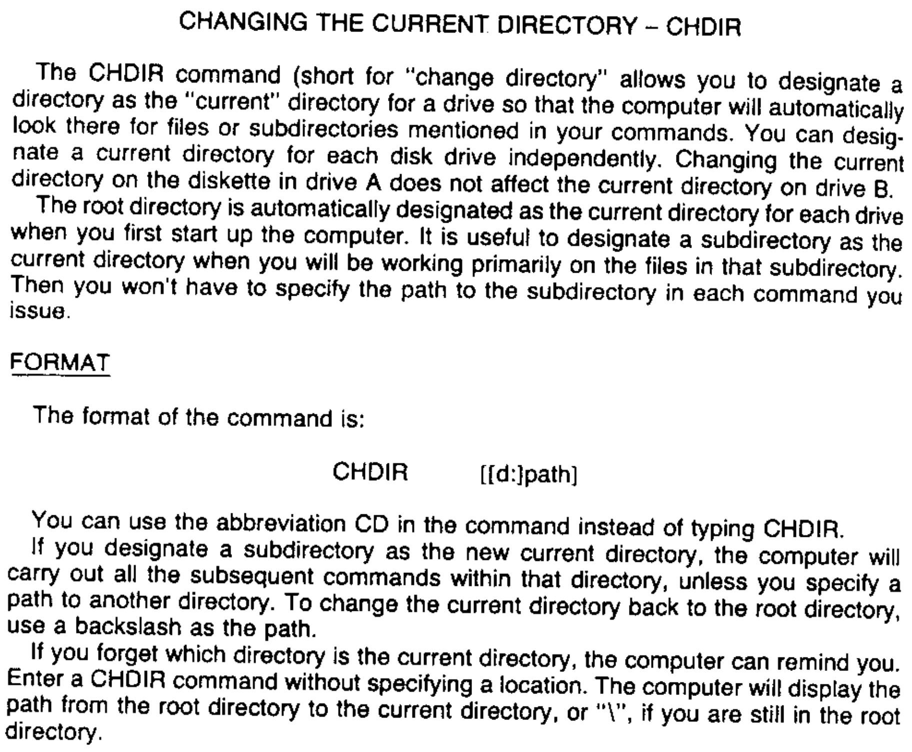
```
A)  CHDIR    B:\     (ENTER)
```

The backslash (\) in the commands above symbolize the root directory. So the command above changes the current directory for drive B to the root directory.
If you forget which directory is the current directory, the computer can remind you. Enter a CHDIR command without specifying a location. The computer will display the path from the root directory to the current directory or a backslash if you are still in the root directory.

Figure 1. Excerpt of manual illustrating RICH SYNTAX elaborations.

After they studied a version of the manual, subjects were asked to carry out a set of ordinary tasks on the computer, without referring back to the documentation. The subjects who had studied manuals containing syntactic elaborations worked significantly more quickly and issued significantly fewer commands. The conceptual elaborations did not significantly improve performance, perhaps because the selection of appropriate commands was fairly obvious for this particular set of tasks.

There is other evidence that examples strongly influence subjects’ interpretation of procedural rules. LeFevre and Dixon (1984) and LeFevre (1985)



CHANGING THE CURRENT DIRECTORY – CHDIR

The CHDIR command (short for "change directory" allows you to designate a directory as the "current" directory for a drive so that the computer will automatically look there for files or subdirectories mentioned in your commands. You can designate a current directory for each disk drive independently. Changing the current directory on the diskette in drive A does not affect the current directory on drive B.

The root directory is automatically designated as the current directory for each drive when you first start up the computer. It is useful to designate a subdirectory as the current directory when you will be working primarily on the files in that subdirectory. Then you won't have to specify the path to the subdirectory in each command you issue.

FORMAT

The format of the command is:

```
CHDIR        [[d:]path]
```

You can use the abbreviation CD in the command instead of typing CHDIR.

If you designate a subdirectory as the new current directory, the computer will carry out all the subsequent commands within that directory, unless you specify a path to another directory. To change the current directory back to the root directory, use a backslash as the path.

If you forget which directory is the current directory, the computer can remind you. Enter a CHDIR command without specifying a location. The computer will display the path from the root directory to the current directory, or "\", if you are still in the root directory.

Figure 2. Excerpt of manual illustrating RICH CONCEPT elaborations.

conducted research on instructions for solving analogy problems. They found that when verbal instructions for how to solve a problem (i.e. rules) were *contradicted* by a situational example, subjects tended to execute a procedure that was consistent with the example rather than one consistent with the rule. We suspect that because the examples were concrete and specific, subjects mistrusted their intepretation of the more abstract rule and reinterpreted the rule to conform to the operations illustrated in the example. In any case, the results underscore the importance of choosing examples carefully.

4.4. The role of examples for learning and remembering to select the best procedure

Various factors may cause a learner to select a less than optimal procedure for solving some problem. The learner may know that one procedure is more appropriate for a problem than another, but if she only remembers how to execute the suboptimal one, that is the one she will end up using. In this sort

of situation, the learner has mastered the selection problem; she simply needs more help with the execution of the procedures she has learned. In contrast, the situations we are mainly interested in concern people who do not think of using a procedure that they would acknowledge to be more appropriate and people who have not learned to judge between alternative procedures.

We believe that *situational examples* can play a dual role in procedural selection. First, they can provide the relevant stimulus cues to help the user 'think of' the right procedure for a specific situation, and second, they can help people learn or induce a generalization for when a given procedure is better than some alternative.

4.4.1. Increasing the salience of alternative procedures: command editing

Consider again the *command editing* procedure. It is quite possible for a user who knows what command editing is and who remembers how to bring back previous commands, to type in a long command instead of modifying and reissuing a similar command that he recently issued. The procedure simply may not have been sufficiently salient that it occurred to the user at the appropriate time. If a person only thinks of one procedure to achieve a specific goal then the problem of *selection* does not arise.

It is interesting to speculate on what aspects of a hypothetical situation example are most likely to increase salience and facilitate retrieval of the procedure in a real problem situation. Retrieval is probably most likely when the example and the problem are identical or very similar. Under these circumstances, many elements in the problem situation may remind the learner of the example, and hence the procedure used in the example. Unfortunately, since the procedures in a computer-operating system can be used in such a wide variety of contexts, it is highly unlikely that the problems users face will be exactly like the examples in the manual, even if the examples are carefully chosen.

What happens, then, when the actual problem situation does not perfectly match the example? In part, this will depend on how the example is represented in memory and how good the example is at illustrating the relevant dimensions of the situation to encode. We believe that the same examples that best motivate *why* a person should want to use the command-editing facility will also be best for reminding a person to use it because these examples highlight those elements of the problem situation that make the command most appropriate. As long as a problem situation matches the example on those dimensions, the example may serve as a good retrieval cue, in spite of other differences between the situations. On the other hand, it is possible that a learner will only store a *superficial* representation of the example; in this case, examples that *literally* match aspects of the current situation will be better memory cues.

To illustrate these two possibilities, consider again the two examples on command editing that were provided earlier (examples 11 and 12), in the light of the following 'real-world' problem. Suppose your friend Smith has a subdirectory called 'upkeep' on his computer that contains files with helpful information about maintenance and repair people. Smith has given you permission to browse through his files from your account on another computer. To see what files are available in the subdirectly, you type a relatively long command, such as the following:

$dir onion::[Smith.home.upkeep]

To read the contents of any file in the 'upkeep' directory, you will have to issue commands of the form:

$type onion::[Smith.home.upkeep]plumbers.mss.

The question is, will you be more likely to remember to use command editing in this situation (i.e. changing the *dir* command into the needed *type* command and reissuing it), if you had previously seen example (11) or example (12) in the manual? Example (11) superficially resembles the problem situation, in that both involve typing and later reissuing a *dir* command. Example (12) showed how to avoid retyping a different command, the *copy* command. However, the point of this example was to motivate *why* one would want to bother with command editing: to save keystrokes by avoiding retyping a very long file specification or 'path' to a file. So despite the greater superficial similarity of example (11) to the problem situation, example (12) should be a better cue to a user's memory. This assumes that users represent tasks and goals at a deeper level, e.g. 'My goal is to save keystrokes and reuse the expression that I already typed.' It is obviously an empirical question how users tend to encode the examples they read in a manual.

4.4.2. Learning to judge when one procedure is better than another: command editing

Skill learners should be familiar with a repertoire of procedures and should be able to select the most appropriate procedure for any given situation. An obvious question is what makes a procedure most appropriate? At various points in this chapter, we have described situations in which one procedure is 'better' than another. In most cases, we have justified this valuation in terms of efficiency: one procedure saves the user keystrokes (e.g. command editing) or reduces the size of the search space (e.g. creating subdirectories). Often, of course, the choice between procedures is not so clear-cut. Consider the choice between procedures for writing a computer program: e.g. should the programmer write a recursive program or an iterative one? It may be easier to write a computer program one way, but the program may be more computa-

tionally efficient another way. To the extent that the considerations for choosing between procedures can be specified, we are interested in how instructional texts can help learners acquire such selection strategies.

As we argued in the preceding section, situational examples highlight those characteristics of a situation that make a particular procedure highly appropriate. Consider again the two examples used to illustrate the command-editing procedure. Example (11) illustrated a situation in which command editing was no more efficient than typing a new *dir* command. In example (12), however, command editing was by far the more efficient procedure, because the user could avoid typing a number of *copy* commands that all contained the same long path specification. We can use this characteristic of the task situation to formulate a generalization about when to choose command editing:

If you must issue a number of very long, similar commands, it is more efficient to edit the commands than to type new ones.

One way to convey a selection principle such as this would be to state it explicitly in a computer manual or instructional text. As with any generalization, a selection principle can be illustrated with examples, in this case, situation examples. Since, as we have seen, situation examples can also be used to exemplify syntactic rules for issuing computer commands, situation examples have the potential of simultaneously illustrating two sorts of rules: rules for executing procedures and generalizations for when to select the procedures. Of course, it may not always be necessary to state the selection principle explicitly. We speculate that learners who see a number of situation examples can often induce the generalization independently.

The extent to which learners need help choosing between alternative procedures is a question which requires further research. We have found that learners can often choose the most efficient computer command without the benefit of explicit advice, whether exemplified or not (Reder, Charney and Morgan, in press; Charney, 1985). It may be that instruction is needed more in more complex skill domains, such as programming.

4.5. Some potential drawbacks to situational examples

Our analysis suggests that situational examples must be chosen carefully to illustrate the conditions under which a procedure should be applied. The example must make salient the underlying *goal* of the procedure as well as illustrating a situation where it is used. If the examples are poorly constructed, they can actually interfere with good performance. When subjects only see impoverished examples, that fail to emphasize the conditions that make one procedure more appropriate than another, the subjects may draw spurious conclusions about when to use the procedures.