

---

# The Strategic Use of Complex Computer Systems

Suresh K. Bhavnani  
Bonnie E. John

---

## 5.1 Introduction

A dominant goal of the Human-Computer Interaction (HCI) field has been to design facile interfaces that reduce the time to learn computer applications. This approach was expected to enable users to quickly perform simple tasks with the implicit assumption that they would refine their skills through experience. However, several longitudinal and real world studies on the use of complex computer systems such as UNIX (Doane et al. 1990), word processors (Rosson 1983), spreadsheets (Nilsen et al. 1993), and computer-aided drafting (Bhavnani et al. 1996) have shown that despite experience, many users with basic command knowledge do not progress to an efficient use of applications. These studies suggest that knowledge of tasks, and knowledge of tools on their own are insufficient to make users more efficient.

In this paper we argue that, in addition to task and tool knowledge, users must also learn an intermediate layer of knowledge that lies between the layers of tasks and tools. This can be illustrated in even very simple tasks performed with simple tools. Consider the task of driving in a nail with a hammer. The task description (drive in a nail) together with the design of the hammer (designed to afford gripping), leads a user to grasp the handle, hold the nail in position, and hit it with repeated blows. While this method can achieve the goal, it often leads to bent or crooked nails, or fingers being accidentally hit with the hammer. In contrast, master craftsmen know that

a quicker way to avoid these problems is (1) tap the nail to guarantee its proper angle of entry and to hold it in place, (2) remove the fingers holding the nail, and (3) drive in the nail with heavier blows. The knowledge of this efficient method is neither expressed in the task description nor expressed by the design of the handle. Instead, this knowledge lies between the layers of tasks and tools. This intermediate layer of knowledge has to be learned, and the cost of learning is amortized over subsequent use of the hammer to drive in nails.

This paper focuses on efficient strategies to use computer applications that lie in the intermediate layers of knowledge. We will show that these strategies are (1) efficient because they exploit specific capabilities provided by computers, (2) difficult to acquire from tool and task knowledge alone, and (3) general in nature and therefore have wide applicability.

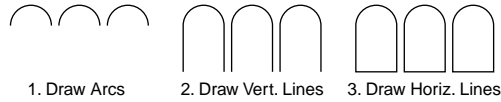
Section 5.2 will introduce the preceding three concepts in the context of *aggregation* strategies that exploit the iterative power of computer applications. Section 5.3 will provide empirical evidence that these strategies are not spontaneously acquired by experienced users but, if used, can reduce task time and errors. Section 5.4 will discuss possible explanations for why such strategies are not easily learned or used. Section 5.5 will expand the notion of strategies beyond those to perform iterative tasks and will briefly discuss our experience of using them to design training. In conclusion, we present some concepts that could lead to a general framework to systematically identify efficient strategies at different levels of generality. The goal is to help designers and trainers identify strategies that make users more efficient in the use of complex computer applications.

---

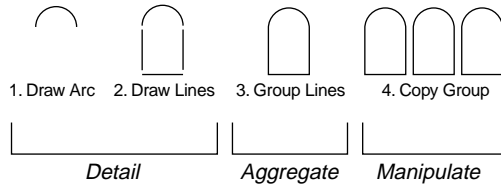
## 5.2 Strategies in the Intermediate Layers of Knowledge

Complex computer applications such as UNIX, CAD, word processors, and spreadsheets often provide more than one way to perform a given task. Consider the task of drawing three identical arched windows in a CAD system. As shown in Figure 5.1A, one way to perform this task is to draw all the arcs across the windows, followed by drawing all the vertical lines, followed by drawing all the horizontal lines. An alternate way to do the same task (as shown in Figure 5.1B) is to draw all the elements of the first shape (Detail), group these elements (Aggregate), and then make multiple copies of the aggregate to create the other shapes (Manipulate). Both these methods allow a user to complete the task. We call such *nonobligatory* and *goal-directed* methods *strategies*. The *Sequence-by-Operation* and *Detail-Aggregate-Manipulate* methods just described are prime examples of strategies that can be used in complex computer systems.

## A. Sequence-by-Operation Strategy



## B. Detail-Aggregate-Manipulate Strategy



(© 1999 ACM, Inc., Included here by permission.)

**FIGURE 5.1** Two strategies to perform the task of drawing three windows in a CAD system

### 5.2.1 Strategies That Exploit the Iterative Power of Computers

The advantage of the *Sequence-by-Operation* strategy is that by drawing all arcs, followed by drawing all lines, the user reduces switching between tools. Although the *Sequence-by-Operation* strategy reduces tool switching, the user still has to perform the iterative task of creating each of the elements. In contrast, the advantage of the *Detail-Aggregate-Manipulate* strategy is that the user draws the elements of only one window, and the computer performs the iterative task of creating copies of the other windows when given their locations. However, a critical part of this strategy is that the user must make sure that all the elements in the original are complete and error-free *before* they are grouped and copied. This avoids having to make corresponding changes in each copy.

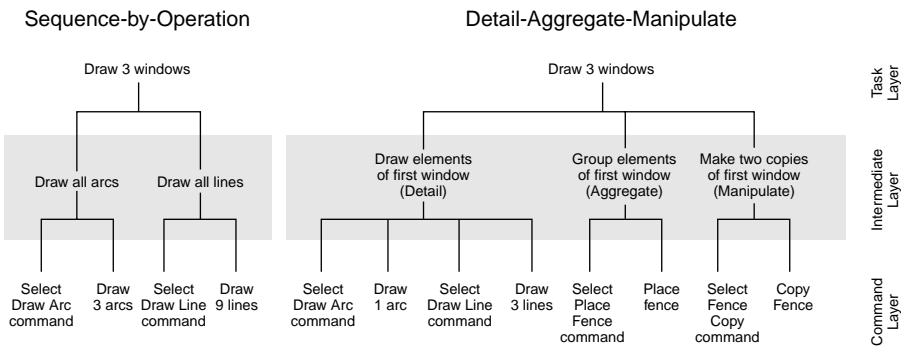
The *Detail-Aggregate-Manipulate* strategy exploits the iterative power of computers through the capability of aggregation provided by most computer applications. For example, most CAD systems, word processors, and spreadsheets allow users to aggregate groups of objects by dragging the cursor over a selection and then applying to this aggregate manipulations or modifications such as copy and delete. By grouping before applying operations, the user exploits the iterative power of the computer because the computer performs the iteration over all the elements in the group. This

notion is captured in the basic strategy *Aggregate-Manipulate/Modify* of which the Detail-Aggregate-Manipulate is just one of several variations. We refer to all of these strategies as *aggregation strategies* (Bhavnani 1998). We will show in Section 5.3 that aggregation strategies are in fact much more efficient in terms of time and errors when compared to Sequence-by-Operation.

Figure 5.2 shows decompositions of the Sequence-by-Operation and Detail-Aggregate-Manipulate strategies for the draw three windows task. These decompositions reveal that the strategies exist in an intermediate layer of knowledge lying between the task description (at the top of the decomposition) and the commands to complete the task (at the bottom). The location of these strategies in the intermediate layers of knowledge profoundly affects their learnability and generalizeability.

### 5.2.2 Acquiring Strategies in the Intermediate Layers of Knowledge

Because strategies such as Detail-Aggregate-Manipulate reside in the intermediate layers of knowledge above commands, they are difficult to infer from command knowledge. For example, in the task to draw three windows, knowledge of how to use commands such as Draw Line and Group Elements in a CAD system is not sufficient to know that it is important to complete all the elements of the first window before grouping and copying. This has led to the general observation that good interface design on its own cannot lead to efficient use (Bhavnani and John 1997). Furthermore, when different strategies can accomplish the same task, the task itself also cannot express this strategic knowledge. This knowledge, therefore, has to be learned by various processes such as through trial and error or through explicit instruction. In



**FIGURE 5.2** Decompositions of the task to draw three windows. The Sequence-by-Operation and Detail-Aggregate-Manipulate strategies lie in the intermediate layers of knowledge below the task, and above the commands.

fact, we will show in Section 5.3 that, despite mastery of basic commands, many users do not spontaneously acquire strategic knowledge to use commands efficiently.











There is a cost to learning strategies such as Detail-Aggregate-Manipulate. Users must learn to recognize opportunities to operate on groups of objects in order to exploit iteration and then know a sequence of actions to execute the strategy. As shown in Figure 5.2, the aggregation strategy requires a very different task decomposition compared to strategies that operate on single elements. However, this learning cost is amortized over the efficiency gains over many invocations of the strategy. This is similar to learning to use any new device efficiently whether it is a hammer or a computer application. Furthermore, we have empirical evidence to show that, when given appropriate instruction, users can easily learn to recognize and use strategies such as Detail-Aggregate-Manipulate (Bhavnani et al. 1999). After a few weeks of class instruction and practice, architectural graduate students learned to decompose complex architectural drawings by using aggregation strategies, in addition to learning commands. One important reason why these strategies were learned easily is that repeated elements are intrinsic to architectural designs (Flemming et al. 1997). Windows, doors, columns, and even entire facades are repeated or mirrored to create designs, and it is typical for an architect to exploit these repetitions while creating drawings. Aggregation strategies such as Detail-Aggregate-Manipulate, therefore, exploit how architects already think about objects in their designs. These results are not unique to teaching CAD strategies to architectural graduate students. Preliminary results from our ongoing research show that strategies can be taught in a short amount of time to a diverse population of freshmen students (Bhavnani et al. 2001).

---

### 5.2.3 Generality of Strategies in the Intermediate Layers of Knowledge

Because strategies such as Detail-Aggregate-Manipulate reside in the layers above the command layer, they are not dependent on specific implementations of commands in an application. For example, the step *Aggregate* in the Detail-Aggregate-Manipulate strategy can be executed by many different commands in different applications. Aggregation strategies, therefore, are generally applicable across computer applications. Figure 5.3 shows three aggregation strategies and how they generalize across computer applications. The first row shows how the Detail-Aggregate-Manipulate strategy can be used in CAD (as already discussed in Figure 5.1B, and in Bhavnani and John 1996) and in other applications. In a spreadsheet application it can be used to create a row of data, aggregate it into a range, and operate on the range using a formula. In a word processor the strategy could be used to copy paragraphs of text across files.

Next, the *Aggregate-ModifyAll-Modify Exception* strategy allows a user to exploit aggregation to handle exceptions. For example, if all except one of a group of elements need to share an attribute, it is better to modify all of them and then change the exception, instead of modifying each on its own. The *Aggregate-ModifyAll-Modify*

	CAD	Spreadsheet	Word Processor											
<b>Detail</b>	1. Draw Lines 	1. Enter Data <table border="1" data-bbox="481 343 632 396"> <tr><th>D</th><th>E</th><th>F</th></tr> <tr><td>22.3</td><td>65.8</td><td>78.4</td></tr> </table>	D	E	F	22.3	65.8	78.4	1. Create Address <table border="1" data-bbox="776 335 969 396"> <tr><td>John Doe</td></tr> <tr><td>100 Main Street, USA</td></tr> </table>	John Doe	100 Main Street, USA			
D	E	F												
22.3	65.8	78.4												
John Doe														
100 Main Street, USA														
<b>Aggregate</b>	2. Fence Elements 	2. Define a Range Named "Set" <table border="1" data-bbox="481 449 632 502"> <tr><th>D</th><th>E</th><th>F</th></tr> <tr><td>22.3</td><td>65.8</td><td>78.4</td></tr> </table>	D	E	F	22.3	65.8	78.4	2. Select Address <table border="1" data-bbox="776 432 969 493"> <tr><td>John Doe</td></tr> <tr><td>100 Main Street, USA</td></tr> </table>	John Doe	100 Main Street, USA			
D	E	F												
22.3	65.8	78.4												
John Doe														
100 Main Street, USA														
<b>Manipulate</b>	3. Copy Fence 	3. Operate on Range <table border="1" data-bbox="463 555 716 608"> <tr><th>D</th><th>E</th><th>F</th><th>G</th></tr> <tr><td>22.3</td><td>65.8</td><td>78.4</td><td>=sum(set)</td></tr> </table>	D	E	F	G	22.3	65.8	78.4	=sum(set)	3. Copy Address to Other Files <table border="1" data-bbox="776 529 1005 608"> <tr><td>John Doe</td></tr> <tr><td>100 Main Street, USA</td></tr> </table> t, USA	John Doe	100 Main Street, USA	
D	E	F	G											
22.3	65.8	78.4	=sum(set)											
John Doe														
100 Main Street, USA														
<b>Aggregate</b>	1. Fence All Elements 	1. Select All Columns <table border="1" data-bbox="481 661 583 723"> <tr><th>D</th><th>E</th><th>F</th><th>G</th></tr> <tr><td></td><td></td><td></td><td></td></tr> </table>	D	E	F	G					1. Select Text <table border="1" data-bbox="776 661 927 714"> <tr><td>xxxx xxxx xxxx</td></tr> <tr><td>xxx xxx xx xx</td></tr> <tr><td>xx xxxxxx xx</td></tr> </table>	xxxx xxxx xxxx	xxx xxx xx xx	xx xxxxxx xx
D	E	F	G											
xxxx xxxx xxxx														
xxx xxx xx xx														
xx xxxxxx xx														
<b>Modify All</b>	2. Modify All Elements 	2. Modify All Columns <table border="1" data-bbox="481 767 680 820"> <tr><th>D</th><th>E</th><th>F</th><th>G</th></tr> <tr><td></td><td></td><td></td><td></td></tr> </table>	D	E	F	G					2. Change Style of All Text <table border="1" data-bbox="776 767 896 820"> <tr><td>xxxx xxxx xxxx</td></tr> <tr><td>xxx xxx xx xx</td></tr> <tr><td>xx xxxxxx xx</td></tr> </table>	xxxx xxxx xxxx	xxx xxx xx xx	xx xxxxxx xx
D	E	F	G											
xxxx xxxx xxxx														
xxx xxx xx xx														
xx xxxxxx xx														
<b>Modify-Exception</b>	3. Modify Exception 	3. Modify Exception <table border="1" data-bbox="481 855 656 908"> <tr><th>D</th><th>E</th><th>F</th><th>G</th></tr> <tr><td></td><td></td><td></td><td></td></tr> </table>	D	E	F	G					3. Modify Exception <table border="1" data-bbox="776 855 896 908"> <tr><td>xxxx xxxx xxxx</td></tr> <tr><td>xxx xxx xx xx</td></tr> <tr><td>xx xxxxxx xx</td></tr> </table>	xxxx xxxx xxxx	xxx xxx xx xx	xx xxxxxx xx
D	E	F	G											
xxxx xxxx xxxx														
xxx xxx xx xx														
xx xxxxxx xx														
<b>Locate</b>	1. Locate Similar Dwg 	1. Locate Similar Set of Formulae <table border="1" data-bbox="481 961 698 1031"> <tr><th colspan="2">A</th></tr> <tr><td>1</td><td>=sum(\$C1:\$C12)</td></tr> <tr><td>2</td><td>=(A1/12)</td></tr> </table>	A		1	=sum(\$C1:\$C12)	2	=(A1/12)	1. Locate Section with Similar Formatting <table border="1" data-bbox="776 961 915 1014"> <tr><td>xxxxxx xxxxx</td></tr> <tr><td>✓ xxx</td></tr> <tr><td>✓ xxxxxxxxx</td></tr> </table>	xxxxxx xxxxx	✓ xxx	✓ xxxxxxxxx		
A														
1	=sum(\$C1:\$C12)													
2	=(A1/12)													
xxxxxx xxxxx														
✓ xxx														
✓ xxxxxxxxx														
<b>Aggregate</b>	2. Fence Elements 	2. Select Formulae <table border="1" data-bbox="481 1067 698 1137"> <tr><th colspan="2">A</th></tr> <tr><td>1</td><td>=sum(\$C1:\$C12)</td></tr> <tr><td>2</td><td>=(A1/12)</td></tr> </table>	A		1	=sum(\$C1:\$C12)	2	=(A1/12)	2. Select Section <table border="1" data-bbox="776 1067 915 1128"> <tr><td>xxxxxx xxxxx</td></tr> <tr><td>✓ xxx</td></tr> <tr><td>✓ xxxxxxxxx</td></tr> </table>	xxxxxx xxxxx	✓ xxx	✓ xxxxxxxxx		
A														
1	=sum(\$C1:\$C12)													
2	=(A1/12)													
xxxxxx xxxxx														
✓ xxx														
✓ xxxxxxxxx														
<b>Manipulate</b>	3. Copy Shape 	3. Copy Formulae <table border="1" data-bbox="481 1181 686 1252"> <tr><th>D</th></tr> <tr><td>=sum(\$C1:\$C12)</td></tr> <tr><td>=(D1/12)</td></tr> </table>	D	=sum(\$C1:\$C12)	=(D1/12)	3. Copy Section <table border="1" data-bbox="776 1181 915 1234"> <tr><td>xxxxxx xxxxx</td></tr> <tr><td>✓ xxx</td></tr> <tr><td>✓ xxxxxxxxx</td></tr> </table>	xxxxxx xxxxx	✓ xxx	✓ xxxxxxxxx					
D														
=sum(\$C1:\$C12)														
=(D1/12)														
xxxxxx xxxxx														
✓ xxx														
✓ xxxxxxxxx														
<b>Modify</b>	4. Modify Shape 	4. Modify Formulae <table border="1" data-bbox="481 1296 686 1367"> <tr><th>D</th></tr> <tr><td>=sum(\$E1:\$E12)</td></tr> <tr><td>=(D1/12)</td></tr> </table>	D	=sum(\$E1:\$E12)	=(D1/12)	4. Modify Section <table border="1" data-bbox="776 1296 957 1349"> <tr><td>xxxxxx xxxxx</td></tr> <tr><td>✓ yyyy yy</td></tr> <tr><td>✓ yyyyyyyyyyyyyyyyyy</td></tr> </table>	xxxxxx xxxxx	✓ yyyy yy	✓ yyyyyyyyyyyyyyyyyy					
D														
=sum(\$E1:\$E12)														
=(D1/12)														
xxxxxx xxxxx														
✓ yyyy yy														
✓ yyyyyyyyyyyyyyyyyy														

(© 1997 ACM, Inc., Included here by permission.)

FIGURE 5.3 Three strategies of aggregation and how they generalize across computer applications. Each cell shows an example of a task that can be performed using a strategy.

Exception strategy can also be used to modify the width of columns with an exception, as well as in a word processor to handle exceptions during the font modification of a paragraph.

Finally, the Locate-Aggregate-Manipulate-Modify strategy in CAD can be used to exploit similarity in a drawing by copying a figure that is already drawn and modifying it. In spreadsheets, this strategy could be used to copy and modify complex sets of formulae. The formulae shown contain absolute and relative referencing of cells that can be modified and reused in another location. In word processors, the strategy could be used to copy and modify a section containing complex formatting.

To summarize, this section described the existence of a set of aggregation strategies that reside in the intermediate layers of knowledge. We argued that these aggregation strategies are (1) efficient because they exploit the iterative power of computers, (2) difficult to acquire spontaneously from knowledge of commands or tasks, and (3) generalizable across computer applications. The next section will analyze the first two points in more detail. First, we will describe a GOMS analysis of a real world task to precisely understand how aggregation strategies can affect performance. Second, we will provide empirical evidence from other studies to show that aggregation strategies are not spontaneously acquired by even experienced users.

---

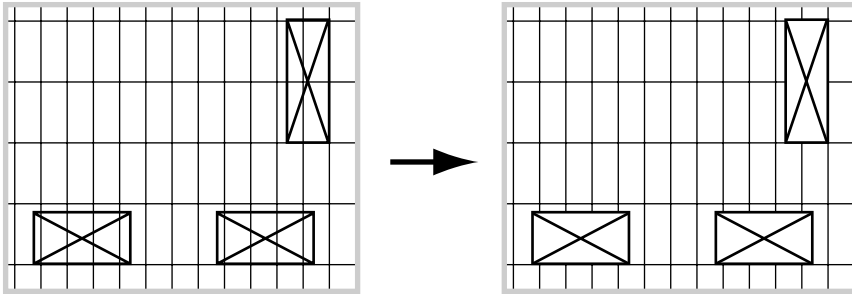
## 5.3 Evidence for the Effects of Aggregation Strategies on Performance

To understand how strategies affect performance, we present a real world task performed by a CAD user during an ethnographic study (Bhavnani et al. 1996). One of the users from the study, “L1,” had more than two years of experience in using a CAD system called MicroStation™ (version 4). His task was to edit a CAD drawing of ceiling panels that overlapped air-condition vents. The task of editing the panels overlapping these vents will be referred to as the *panel clean-up task*. This task is typical of drawing tasks performed by architects during the detail drawing stage of a building design. We observed nine other users who performed similar drawing tasks in our study.

---

### 5.3.1 The Panel Clean-up Task

As vents go vertically through ceiling panels, they both cannot occupy the same space. Therefore, as shown in Figure 5.4, L1 had the task of removing all the line segments (representing ceiling panels) that overlapped the rectangles (representing air-condition vents). The vents and panels were defined in two different drawing files that were simultaneously displayed on the screen to reveal their overlap. This



(© 1998 ACM, Inc., Included here by permission.)

**FIGURE 5.4** The panel clean-up task requires all ceiling panel lines that overlap the air-condition vents to be modified. The drawings are schematic and not to scale.

enabled L1 to modify the panels without affecting the vents. The file had 21 such vents, all of them similar to those shown in Figure 5.4. This meant that L1 had to modify numerous lines that overlapped the vents.

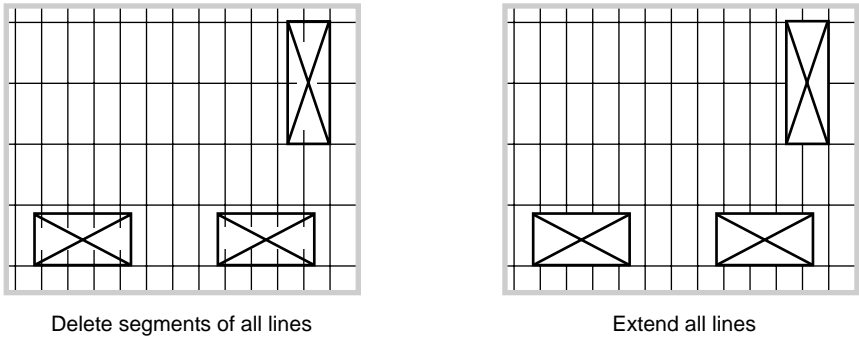
### 5.3.2 How L1 Performed the Panel Clean-up Task

L1 zoomed in and panned a single window in order to view sets of vents to work on. Figure 5.4 represents a typical example of such a window setup, with 3 of the 21 vents displayed. As shown in Figure 5.5, L1 first cut all the panel lines that overlapped the 3 vents by using the Delete Part of Element tool (which deletes a portion of a given line between two specified points). He then cleaned up all the cut lines to the edges of the vent using the Extend to Intersection tool (which extends or shortens a line to the intersection point of any other line).

By sequencing all the cut operations across the vents, followed by all the cleanup operations, L1 is effectively using the Sequence-by-Operation strategy described in Section 5.2. This strategy reduces tool switches between the cutting and cleaning operations but requires the user to perform the iterative task. Furthermore, the task requires high precision, since L1 has to select each panel line in order to cut and extend it to the edge of the vent.

Because of the highly repetitious and precise nature of the task, L1 committed several errors of omission and commission. As shown in Figure 5.6, he did not notice that two panel lines located very close to the boundary of the upper right-hand vent overlapped the vent. He had to return to them after the rest of the lines had been cut and extended. Second, he accidentally selected a panel line just above the lower right-hand vent instead of the actual vent-line, thereby extending a panel-line to the wrong place. This error went undetected, and the drawing was inaccurate after he

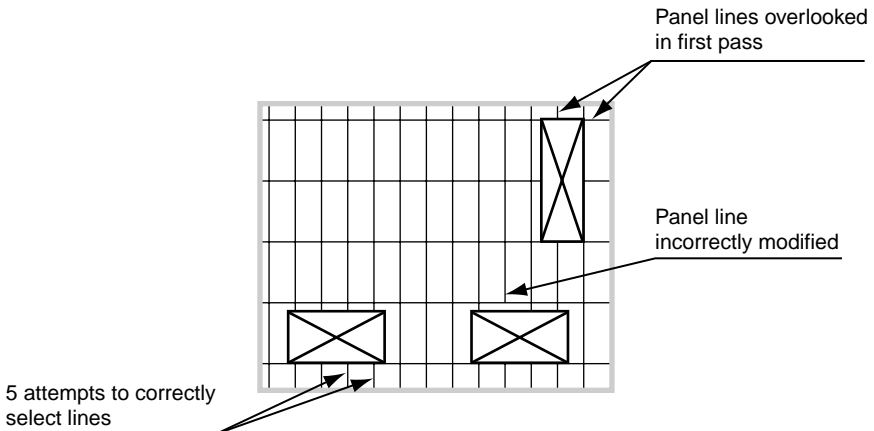




(© 1998 ACM, Inc., Included here by permission.)

**FIGURE 5.5** The method used by L1 to perform the panel clean-up task.

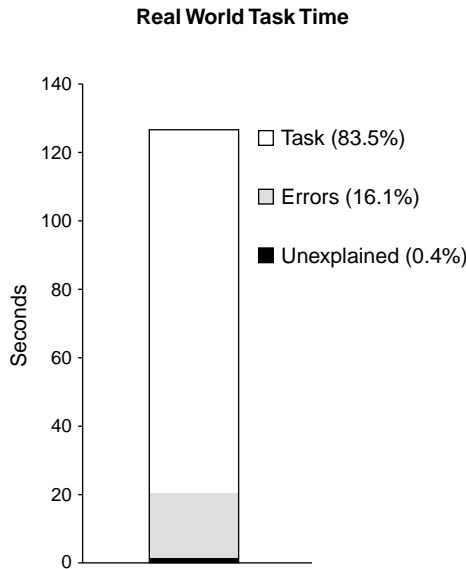
completed the task. Finally, he committed five slips in the selection of panel lines, which had to be repeatedly reselected in order to get exactly the line he wanted. Despite these difficulties, L1 consistently used this time-consuming and error-prone strategy to clean up all 21 vents. In the process, he committed several more omission and commission errors and took approximately 30 minutes to complete the entire task.



**Figure 5.6** Errors in the panel clean-up task leading to inefficiencies and an inaccurate drawing. The figure shows the drawing after L1 completed the task.

To precisely understand the nature of these inefficiencies in terms of time and frequency of errors, the data was transcribed at the keystroke level and quantitatively analyzed. As shown in Figure 5.7, L1 took more than two minutes to complete the fairly simple task of deleting 11 very similar line segments (these numbers relate to the cleanup of 3 vents—the total task, as described earlier involved the cleanup of 21 vents). Furthermore, he spent 20 seconds to commit and recover from errors, which formed 16 percent of the total task time.

Many of these errors could have been avoided if L1 had used the Aggregate-Modify strategy to delegate to the computer the repetitious task of cutting and cleaning many lines. For instance, L1 could have used the Place Fence<sup>1</sup> command (an aggregation command) with a Snap mouse option (where the cursor jumps to the closest intersection) to accurately place a fence over the vent and then delete all the panel lines in one step. By using this procedure, all element segments within the fence, regardless of how visually close they were to the vent boundary, would have been selected. The errors related to precise line selection and of overlooking lines that had to be cut and



**FIGURE 5.7** Total time to complete the 3-vent clean-up task including the time to commit and recover from errors. Errors are all actions where a correct goal was incorrectly executed (slips). Unexplained behavior includes all behavior where it was not obvious what goal was being achieved.

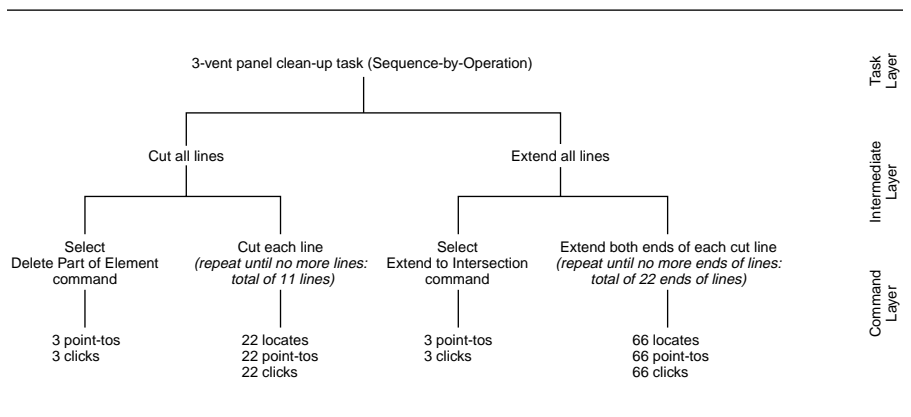
<sup>1</sup>Ethnographic notes revealed that L1 had used the Fence command several times in other tasks to modify groups of objects. The missed opportunity to use the Aggregate-Modify strategy was therefore not due to the lack of knowledge of this command.

extended and could therefore have been avoided. Furthermore, because the iterative task of cleaning up each line would be delegated to the computer, it appears that the strategy could have reduced the time to perform the task.

### 5.3.3 Cognitive Analysis of the Panel Clean-up Task

To understand the differences between the Sequence-by-Operation and Aggregate-Modify strategies to perform the panel clean-up task, we first constructed hierarchical goal decompositions of each approach. Figure 5.8 shows a decomposition of the task as performed by L1 using the Sequence-by-Operation strategy. As shown, he used the Delete Part of Element command to cut each line across the three vents and the Extend to Intersection command to extend each of the cut lines to the boundary of the appropriate vent. The figure shows how L1's strategy choice resulted in many low-level mouse inputs. Figure 5.9 shows a task decomposition of how L1 could have performed the same task using multiple instances of the Aggregate-Modify strategy. When contrasted to the real-world task decomposition, there is a reduction in the number of low-level inputs due to the delegation of iteration to the computer.

To estimate the effect of this reduction in low-level inputs on performance, we developed GOMS (Card et al. 1983) models of both approaches. As shown in Figure 5.10, the model with the Aggregate-Modify strategy predicted a reduction in time of 71 percent. Furthermore, as shown in Figure 5.11, the frequencies of inputs were different between the two models. While there is an increase in the number of command selections (as the Fence and Delete operations have to be applied to three vents), there is a reduction in the number of precision inputs to select lines and intersections, as well as a reduction in the number of overall mouse clicks (command selections, accepts, tentative snaps). The large number of precision inputs may explain why L1 committed many errors, which added 20 seconds to the overall time.



**FIGURE 5.8** A GOMS decomposition of the 3-vent panel clean-up task using the Sequence-by-Operation strategy to clean up each vent.

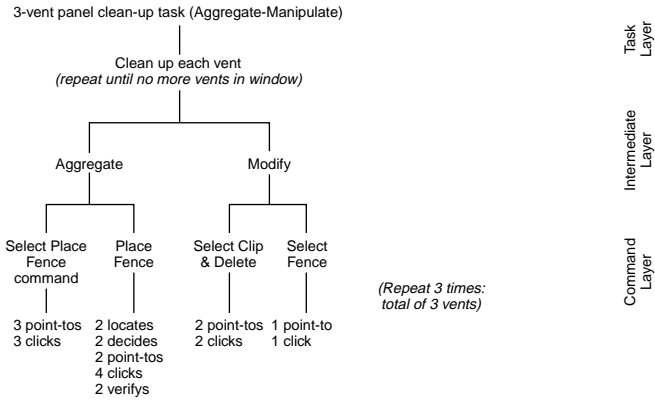


FIGURE 5.9 A GOMS decomposition of the 3-vent panel clean-up task using the Aggregate-Modify strategy to clean-up each vent.

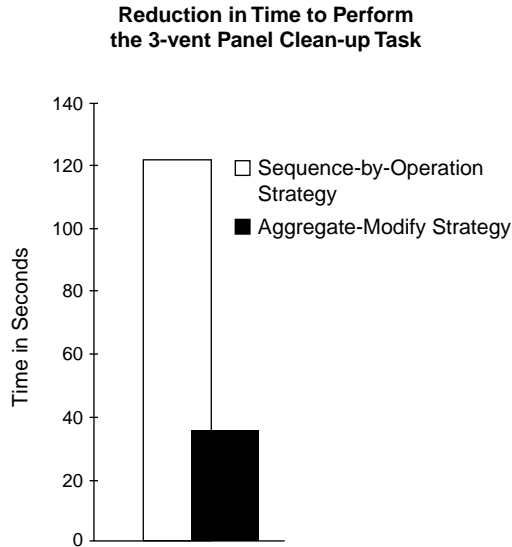
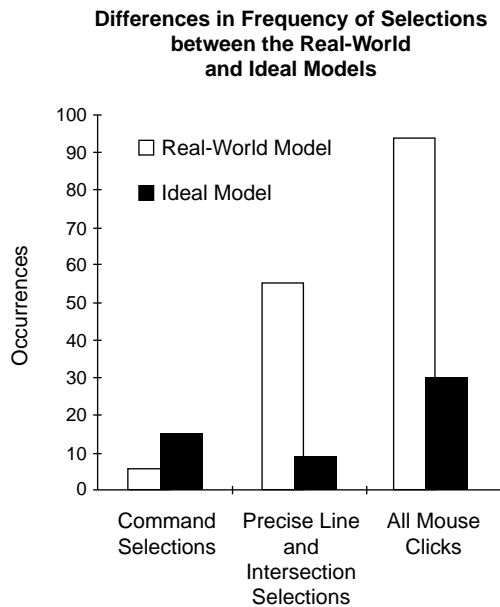


Figure 5.10 The Aggregate-Modify strategy used in the ideal model could reduce the time to do the panel clean-up task by 71 percent.



**FIGURE 5.11** Change of input frequencies between the real-world data and ideal model for the 3-vent panel clean-up task.

The analysis of the panel clean-up task reveals many issues related to strategy use. First, despite experience and knowledge of the Fence command, L1 did not use an efficient strategy to perform a highly repetitious task requiring high precision. Second, despite making many errors, L1 was persistent in using his strategy over the course of the entire task. Third, the use of an aggregation strategy could have reduced time and errors, and could have led to a more accurate product.

### 5.3.4 Inefficient Use Reported in Other Studies

The preceding results are not unique to L1 performing the panel clean-up task. Our analysis of nine other experienced CAD users in the same office revealed a similar pattern of behavior (Bhavnani 1998). Users could have saved between 40 to 75 percent of their time to complete their tasks if they had used various forms of the aggregation strategies as shown in Figure 5.3. These results are also not unique to our study of CAD usage. Lang et al. (1991) report an experienced user who missed an opportunity to use the Detail-Aggregate-Manipulate strategy in a CAD task. When the task was redone after a brief discussion with an expert CAD user, it was completed in 67.5 percent less

time. This study provides more evidence that although aggregation strategies need to be explicitly taught, they are easily learned through instruction and successfully executed.

The preceding results generalize even outside the domain of CAD. Nilsen et al. (1993) studied the development of 26 graduate business students learning how to use Lotus 1-2-3™ over a period of 16 months. Their results showed that even after 16 months of using the application in enrolled courses, the students did not use efficient strategies. For example, a task required five columns to be set to a particular width X and one to be set to a different width Y. The efficient method to perform this task involves two commands: one to set all the columns to width X and one to set the width of the exception to Y. Only 2 of the 14 students used this method. The other 12 students changed the width of each column individually. The authors make the observation that experience does not guarantee that users change their strategies to more efficient ones. It is important to note that the efficient strategy suggested by the authors is in fact the Aggregate-ModifyAll-ModifyException strategy described in Figure 5.3.

In a different study on spreadsheet use, Cragg and King (1993) have shown that 55 percent of users did not use the range option, an aggregation command to group and name many cells in Microsoft Excel. Once a range is created and named, it can be manipulated in other formulae merely by reference to the range name. This is in fact an instantiation of the Detail-Aggregate-Manipulate strategy in the use of a spreadsheet application, also shown in Figure 5.3.

This cognitive analysis of the panel clean-up task, together with the other empirical studies, suggest two basic points. First, despite experience, users do not easily acquire aggregation strategies to perform iterative tasks. The users tend to master the use of commands but do not appear to progress toward using them in an efficient way to complete complex tasks. Second, when used, aggregation strategies can in fact reduce time and errors and lead to a more accurate product.

While the GOMS analyses provide a rigorous account of the observed behavior, in addition to the improvements that could be achieved through the use of aggregation strategies, it cannot explain how the knowledge and behavior of the users got to be that way. The next section will explore possible explanations of why many users do not acquire and use efficient strategies.

---

## 5.4 Possible Explanations for Inefficient Computer Usage

Why don't experienced users learn and use efficient strategies, and why do these inefficient behaviors persist? This section will present possible explanations under two broad categories: (1) efficient strategies not known and (2) efficient strategies known but not used. These explanations will be derived from empirical studies done on computer applications where efficient strategies were not used, from both existing theories of knowledge acquisition, and from emerging theories of strategy choice

and usage. Many of our explanations come directly from our experience studying CAD usage in detail. However, these results generalize to other complex computer applications. The goal of discussing these explanations is to identify approaches to improve the use of complex computer applications.

---

#### **5.4.1 Efficient Strategies Not Known**

The simplest explanation for the inefficient use of computer systems is that some users, despite many years of computer experience, had not yet acquired knowledge of efficient strategies. While it is well known that the acquisition of expertise is time-consuming, the following reasons explore why users of complex systems persist in not acquiring efficient strategies.

##### ***5.4.1.1 Efficient Strategies Have Not Been Made Explicit***

One possible reason that efficient strategies are not known is that they are neither explicitly provided in instructional manuals nor explicitly taught in vendor-provided training. In a systematic search of libraries, publishers, and CAD vendors, we found that only 2 out of 26 books (randomly selected from the entire population of 49 books) went beyond the description of commands to perform simple tasks. One of the books (Crosley 1988) describes the importance of “thinking CAD.” He states, “It’s possible to use computer-aided drawing without really taking advantage of its capabilities. Even some experienced CAD users have simply transferred all their manual-drawing habits over to the computer” (p. 6). Later he adds, “The advantages of CAD are not free; they come at the expense of having to actually design the drawing” (p. 11). While this author stresses the importance of rethinking the drawing process, he does not present explicit strategies to “design the drawing,” leaving the readers to discover and implement the strategies themselves.

##### ***5.4.1.2 Weak Causal Relationship between Method and Quality of Product***

While the absence of strategic knowledge in books and manuals makes it difficult for users to obtain it directly, it cannot explain why CAD users do not discover the strategies while using their systems. An analysis of efficient manual drafting strategies provided some clues as to why strategy discovery in computer usage may be difficult. For instance, a well-known manual drafting strategy to prevent lines getting smudged and drawings getting dirty is to always “begin work at the upper left corner of the sheet of drafting paper and to finish at the lower right corner of the sheet” (Beakley et al. 1984, p. 47). In most cases, if such strategies are not followed, it is very hard to produce a quality drawing. A wrong strategy invariably leads to a visibly low-quality drawing. Because there is such a strong causal relationship between technique and quality, and because the flaws are publicly visible, drafters tend to be highly motivated to improve their technique.

This strong causal relationship between technique and drawing quality is absent in CAD. The drawing produced by LI, when printed, is clean. Therefore, there is no

visible indication that the drawing was produced by an inefficient strategy. As the flaws in the technique are not publicly visible, the users neither notice their inefficient techniques nor have motivation to change them. This phenomenon has also been observed in controlled studies. For example, Singley and Anderson (1989) note that “productions which produce clearly inappropriate actions contribute to poor initial performance on a transfer task but are quickly weeded out. Productions which produce actions which are merely nonoptimal, however, are more difficult to detect and persist for longer periods” (p. 137).<sup>2</sup>

### 5.4.1.3 Office Culture Not Conducive to Learning

The preceding explanations focus on an individual’s interaction with a CAD system. However, real-world CAD usage typically occurs in a group environment where information is exchanged. This exchange can strongly affect the usage of a CAD system. For example, Gantt and Nardi (1992) recommend that CAD managers encourage “gurus” to develop expert knowledge and to act as disseminators of this information within an organization. Majchrzak et al. (1987) provide several recommendations: Managers should be well trained in the technology; CAD training should focus on presenting a general education in CAD concepts, thereby moving away from teaching only commands; and users should have biweekly meetings where they can discuss specific problems and keep abreast of changes.

However, as described in our ethnographic study of an architectural office (Bhavnani et al. 1996), such ideal conditions do not always occur in realistic office settings. The manager of the architectural section we observed was not trained in the use of CAD and did not use it to create drawings. Furthermore, training was perceived as a once-in-a-lifetime requirement, and the users were not encouraged to get follow-up training. As a result, the system had undergone many changes that were unknown to the users.

The lack of training was exacerbated by the absence of any regular discussions on system usage. Most discussions were confined to issues concerning design, and architects rarely discussed drawing strategies or looked over each other’s shoulders during the drawing process. Additionally, there was an internal rule that prevented users from contacting the vendor phone support directly for help. The questions had to be routed through a system coordinator, who did not have a clear understanding of the problems faced by the architectural group and therefore was ineffectual in solving problems. These conditions severely inhibited the flow of CAD-related information within the group.

In cases when drawings are shared and modified within a group working on the same project, a poorly constructed CAD drawing can cause irritations and problems to other users. For example, a user might expect to move a shape by grabbing a side and, when that side moves away from the rest of the shape, realize the shape was

---

<sup>2</sup>Singley and Anderson are discussing a model of cognition where knowledge is encoded as if-then rules called *productions*.



constructed with single lines instead of as a polygon. In such cases, the drawing strategy becomes public and therefore presents opportunities for critical appraisal of inefficiencies. However, if all the users in a group share a flawed mental model of the CAD system, the inefficient strategy can remain undetected despite shared drawings. This was exactly the situation at the office where our data were collected. Therefore, the realities and complications in realistic office environments can make the dissemination of CAD-related information difficult and unreliable.

---

## 5.4.2 Efficient Strategies Known But Not Used

Another possible reason for the inefficient use of complex computer systems is that users know efficient strategies but choose not to use them. The following are some of the possible reasons and our evidence for and against those reasons.

### 5.4.2.1 Efficiency Not Valued

There is a possibility that users may know aggregation strategies but decide not to use them because they do not value the benefits they provide. That is, the users neither care for the savings in time nor for the accuracy that the strategies could produce.

This possibility is in fact *not* supported by our ethnographic data. Users explicitly stated the importance of saving time while performing drafting tasks. For example, in a discussion on advanced commands during the ethnographic study (Bhavnani et al. 1996), an architect explicitly stated, “Anything that saves time is of value to us.” This observation is further substantiated by current research in the acquisition of strategic knowledge. For example, the adaptive strategy choice model (ASCM) developed by Siegler and Shipley (1995) predicts how children select strategies to solve problems in arithmetic. One of the predictions provided by ASCM, verified through empirical analysis, states that “when children can choose among alternative ways of executing a given strategy, they should increasingly choose the ones that are fastest and that yield the most accurate results” (Lemaire and Siegler 1995, p. 86). Although these predictions have to be verified with adults using computer applications, the aggregation strategies fit exactly into this category of strategy as they are predicted to be faster than the ones the users had and to produce more accurate results.

### 5.4.2.2 Strategies Not Really Efficient

It can be argued that the strategies we have identified as efficient require additional cognitive costs that are not taken into account in our GOMS models. If this were true, the strategies may not really be efficient, and users may therefore choose not to use them. While this argument may be potentially true for more complex tasks, we do not believe it to be true for the tasks we observed and modeled.

The tasks we observed and modeled were so simple that they did not involve time-consuming problem-solving or planning. For example, the panel cleanup task was

simple and regular. There were many vents, all the vents had to be clear of ceiling panel lines, and the architect knew this at the start of the task. That was the only knowledge necessary to invoke the Aggregate-Modify strategy. There was nothing additional to figure out or plan. The user needed only to select a strategy and execute it. Such tasks are well modeled in the GOMS framework. In our models (Figure 5.8 and 5.9), the small amounts of perception and cognition needed to recognize the task situation are subsumed in the selection rules to pick the strategy and in the traversal of the goal hierarchy. Only perceptual operators (locate, verify), cognitive operators (decide), and motor operators (point-to, click) combine to give the time predictions because the theory and practice of GOMS does not assign time to selection rules or goal manipulation.<sup>3</sup> Therefore, we believe our models reflect any cognitive costs associated with using the strategies we identified, and they truly are efficient during the performance of simple tasks.

More generally, for users skilled in their task domain, the recognition of features like repetition, symmetry, and similarity are likely to be central to their task (for example, see Flemming et al. 1997 for a discussion of such domain knowledge known by architects). Therefore, users who are skilled in their domains need only learn the connection between these task concepts and the strategies that exploit them (see Section 5.4.1 for a discussion of learning costs) in order to invoke this knowledge in simple task situations.

However, there exist more complex tasks that may require problem solving and planning to recognize a structure and exploit it with efficient strategies. For example, given a cathedral with recursive symmetries, an architect, despite his or her domain experience, must first look for the recursive structure in the task, decompose it to the lowest level of symmetry, and then build up the drawing through the successive levels of symmetry using an aggregation strategy. This is what Crosley (1988) meant by “design the drawing” (p. 11). The more complex the structure in a drawing, the more mental effort is required to identify how best to decompose the drawing in order to

---

<sup>3</sup>It is true that adding each new strategy to a user’s knowledge necessarily also adds at least one new selection rule to choose that strategy in the appropriate task situation. However, many cognitive modeling theories with good fit to empirical data assume no extra performance cost to having more selection rules that are not applicable to the task situation. For instance, GOMS (Card et al. 1983), Soar (Newell 1990), and ACT-R (Anderson and Lebiere 1998) all have this characteristic. Although some empirical evidence exists for the mere existence of different methods increasing decision time for skilled users (Olson and Olson 1990), it is small compared to the savings in execution time these strategies would provide.

It is also true that task decompositions using strategies often have slightly deeper goal-stacks than simpler strategies. For example, the Aggregate-Modify strategy for the 3-vent panel cleanup task (Figure 5.9) has a deeper goal stack than the Sequence-by-Operation strategy for the same task (Figure 5.8). Whether a deeper goal stack adds to performance time for skilled use is an open research question (John and Kieras 1996). Card et al. (1983) tried both approaches and found no additional predictive power from assigning time to goal decomposition and therefore left it out of the original GOMS formulation for simplicity’s sake. On the other hand, Kieras (1997) included 100 msec per push or pop of a goal in GLEAN, and both Soar and ACT-R also include time on the order of 50–100 msec. Again, since the difference in depth is typically one or two levels at most, even this potential cost is small compared to the usually more substantial cost in keystrokes and mouse movements.

use an aggregation strategy. These are not the tasks we have modeled, and more research is required to understand how the aggregation strategies play out in such situations. (See Bhavnani et al. 1999 for how we taught students to decompose complex drawings and to use aggregation strategies.) Given the huge savings in execution time predicted by our GOMS models of simple tasks, it is likely that the more complex the drawing, the greater the cost of not using appropriate aggregation strategies. Therefore, we expect that the extra mental effort required to decompose complex tasks will be more than compensated by the overall savings in time that aggregation strategies provide.

For these reasons, we believe that in simple task situations similar to those we and others have observed (Doane et al. 1990; Rosson 1983; and Nilsen et al. 1993), the benefits of using aggregation strategies far outweigh the negligible performance costs. Therefore, if they had been known, they would have been used. In contrast, during the performance of more complex tasks, a tradeoff may arise between the cost of planning the task decomposition and the benefits of executing the appropriate aggregation strategies. Further research would be needed to understand such tradeoffs.

#### **5.4.2.3 Prior Knowledge Dominating Performance**

Several studies have shown how prior experience of manual tasks has a strong effect on performing computerized tasks. For example, many researchers have shown that the difficulties expert typists encounter when they first learn to use a text editor can be explained by their prior knowledge of using typewriters (Carroll and Thomas 1982; Douglas and Moran 1983; Halasz and Moran 1982; Lewis and Mack 1982; Mack et al. 1983; and Waern 1985). Marchionini (1989) found that many high school students, even after being trained to use online encyclopedias with sophisticated query searches, tended to use simple index-based searches similar to manual searches of printed encyclopedias. It may be the case that users know most efficient strategies but fail to use them because they are dominated by prior knowledge. The difficulty of breaking previously learned habits has been explored by cognitive theories such as ACT\* (Singley and Anderson 1989).

The strong effects of prior knowledge may explain L1's interactions. Prior to using CAD, L1 had spent many years using manual drafting tools to create architectural drawings. The tools of manual drafting (such as the T-square, triangle, pencil, and eraser) are precision tools that assist users in creating accurate drawings. They are obviously not designed to assist users in iterative tasks. When using such tools, the user performs all the iteration. If 10 lines have to be drawn, then each line has to be individually drawn. Often, iterative drawing tasks require more than one tool such as the task of shortening 10 lines that requires each line to be erased and then redrawn accurately. For such tasks, it makes sense to use the Sequence-by-Operation strategy where all the 10 lines are erased, followed by redrawing all the 10 lines because it saves switching between the eraser and the pencil. This, of course, is exactly the strategy used by L1. Because L1 had spent many years using manual drafting tools,

the well-learned Sequence-by-Operation strategy (efficient in manual drafting but inefficient in CAD) may in fact have blocked the use of the Aggregate-Modify strategy even though he knew it. It seems possible that if L1 had been cued to a better way, he might have switched to the better strategy.

---

### 5.4.3 Discussion of Possible Explanations of Inefficient Computer Usage

The preceding sections have presented several reasons that conspire against users employing strategic knowledge. Our evidence suggests that the more compelling reasons involve the difficulty of acquiring strategic knowledge or that this knowledge is insufficiently strong to routinely come into play in real-world tasks. Furthermore, users do seem to value the benefits provided by efficient strategies, and those benefits seem to be real.

While we do not deny that cognitive cost will be incurred in *learning* efficient strategies, we believe this cost does not extend in any meaningful way to skilled performance. There are situations where this may not hold—for example, when users are under the effects of fatigue, boredom, or low motivation. Neither present-day cognitive theory in HCI nor our data speak to this issue, and it should be investigated further. However, under the normal, goal-directed, skilled performance often studied in HCI, the aggregation strategies posited here are efficient at performance time and do add value to those task situations where time is important to users.

The cost of acquiring an efficient strategic level of knowledge is currently very high. It is so high, in fact, that it is not surprising that many studies of “regular” users report this lack of knowledge. There do exist subpopulations of users who enjoy experimenting with different methods in order to push the edge of their computer knowledge or other groups who experiment and compete with friends to find the fastest ways to perform tasks. Such users are motivated to invest the time necessary to acquire efficient strategies. However, as evidenced by the studies presented in this and other papers, such users are not universal.

Many approaches can be taken to alleviate this situation ranging from making strategic knowledge explicit through training, manuals, help systems, and tutorials, to making organizational changes to encourage exploration, feedback, and sharing of knowledge. However, we believe all these approaches depend on the central fact that the strategic knowledge must first be identified before it is disseminated. In the next section, we describe other general strategies that are important in the use of complex computer applications.

---

## 5.5 General Computer Strategies beyond Aggregation

The basic notion underlying all aggregation strategies is that an efficient way to deal with the iterative task of operating on many objects lies in the ability to aggregate the objects and to apply operations on that aggregate. As we discussed in Section 5.2, this ability shifts the task of iterating over each object from the user to the computer. Such strategies are possible because computers have the power to iterate over many objects in an aggregate. Aggregation strategies therefore exploit the power of iteration provided by computers. This insight motivated us to look for other powers provided by computer applications and to explore whether these powers could help identify other efficient strategies.

Our explorations led us to identify three other powers that were generally provided across computer applications: propagation, organization, and visualization.<sup>4</sup> As shown in Figure 5.12, each of these powers requires a set of strategies to exploit it. Propagation strategies exploit the power of computers to modify objects that are connected through explicit dependencies. These strategies allow users to propagate changes to large numbers of interconnected objects. Organization strategies exploit the power of computers to construct and maintain organizations of information. Such strategies allow for quick modifications of related data. Finally, visualization strategies exploit the power of computers to display information selectively without altering its content. Strategies of visualization can reduce visual overload and navigation time. Similar to the general aggregation strategies presented in Section 5.2, the following section will discuss how the seven strategies in the previous three categories are useful and meaningful in word processing, spreadsheet, and CAD tasks. These strategies also begin to extend our definition of efficiency from task time and errors to include other important variables such as modifiability of content, and visual overload.<sup>5</sup> All these strategies appear to be intuitively efficient but need to be rigorously tested through future research.

---

### 5.5.1 Propagation Strategies

The first two strategies in Figure 5.12 (Strategies 1 and 2) exploit the power of computers to propagate modifications to objects that are connected through explicit dependencies. Strategy 1 makes the dependencies between objects “known” to the computer so that (1) new objects inherit properties or receive information from another object, and (2) modifications can propagate through the dependencies. For

---

<sup>4</sup>We do not yet have a principle to generate these powers. We therefore do not claim this list is complete.

<sup>5</sup>Green (1989) has analyzed similar concepts such as *hidden/explicit dependencies* and *viscosity/fluidity* in the framework of “cognitive dimensions.”

General Strategies	Word Processing Examples	Spreadsheet Examples	CAD Examples
<b>Propagation</b>			
1. Make dependencies known to the computer	Make paragraphs dependent on a format definition	Make formulas dependent on numbers in cells	Make window design dependent on a graphic definition
2. Exploit dependencies to generate variations	Modify style definitions to generate variations of the same document	Modify formula dependencies to generate different results for the same data set	Modify graphic definitions to generate variations of a building facade
<b>Organization</b>			
3. Make organizations known to the computer	Organize information using lists and tables	Organize yearly data in different sheets	Organize columns and walls on different layers
4. Generate new representations from existing ones	Generate table from tabbed words	Generate bar graph from table	Create 3-D model from 2-D floor plan
<b>Visualization</b>			
5. View relevant information, do not view irrelevant information	Magnify document to read fine print	View formulas, not results	Do not display patterned elements
6. View parts of spread-out information to fit simultaneously on the screen	Use different views of the same document to bring two tables together on the screen for comparison	Use different views of the same document to view column headings and data at the end of a long table	Use two views focused at the ends of a long building facade to make comparisons
7. Navigate in global view, manipulate in local view	Use outline view to view entire document and specify location of interest, use local view to make modification	Use outline view to view entire spreadsheet and specify location of interest, use local view to make modification	Use global view to view entire building and specify location of interest, use local view to make modifications

**FIGURE 5.12** Seven general strategies beyond aggregation strategies and how they are useful in word processing, spreadsheet, and CAD tasks

example, word processor users can create paragraphs that need to share a common format to be dependent on a common definition. When the definition is modified, all the dependent paragraphs are automatically changed. Similarly, formulas in a spreadsheet can be linked to dependent data, or graphic elements in a CAD system can be linked to a common graphic definition of objects.

Strategy 2 exploits such dependencies to generate variations of the same information. For example, the strategy could be used to explore different looks of a document

in a word processor, generate different results in a spreadsheet by altering a variable (such as an interest rate), or create several variations of window designs in a building facade while using a CAD system.

---

### 5.5.2 Organization Strategies

Strategies 3 and 4 exploit the power of computers to construct and maintain organizations of information. Strategy 3 reminds users to make the organization of information known to the computer to (1) enhance comprehension and (2) enable quick modifications. For example, a table constructed with tabs in a word processor is not “known” to the computer as a table, and therefore the tabular structure may not be maintained when the table contents are modified. On the other hand, a table that is known to the computer will be maintained under any modification of its contents. Similarly, data for different years in a spreadsheet can be organized in separate sheets for easy access, and different building elements such as columns and walls can be separated in different layers. Strategy 4 generates new representations from existing ones. For example, tabbed tables in word processors can be converted to tables and vice versa, data in a spreadsheet can be represented as charts, and 3-D graphic objects can be generated from 2-D representations and vice versa.

---

### 5.5.3 Visualization Strategies

The last three strategies in Figure 5.12 (Strategies 5–7) exploit the power of computers to view information selectively. Strategy 5 can be used to alter the amount of information displayed by viewing relevant information and not viewing irrelevant information. For example, when text is too fine to read while using a word processor, this strategy could be used to magnify the view instead of changing the font size. Similarly, in a CAD system, patterned elements can be undisplayed when not needed in order to make the relevant information more salient.

Strategy 6 addresses the limited screen space of most computer terminals. Often, users have tasks that require them to compare or manipulate objects that are difficult to view simultaneously in a single view. For example, a user might need to compare the contents of a table at the beginning of a long word processing document to the contents of a table in the middle of the same document. In such cases, instead of moving back and forth between the tables, it is more efficient to set up views that focus on each table to enable both to be viewed simultaneously on the screen. This strategy is clearly useful in large documents containing text, numbers, or graphic elements and therefore generally useful across applications using such objects.

Strategy 7 extends the notion of selective viewing to tasks involving a combination of navigation and manipulation. For example, a CAD user might need to make many precise changes to different parts of a large floor plan. A magnified view is needed to make the precision changes, while a global view is needed for navigation to the next task. One way is to zoom in to perform the precise modifications and then

to zoom out of the same view to navigate to the next task. A more efficient method is to have one global view of the file for navigation and one local view to make the changes. The user then selects the location of interest in the global view, which automatically updates the local magnified view where the user can make the precise modifications. As shown in Figure 5.12, this strategy is useful when modifying a large word processing document as well as a large spreadsheet.

Currently we do not have a systematic way to identify powers of computers nor do we understand how to systematically identify efficient strategies from these powers. However, we are convinced that teaching such strategies would benefit users. To test this hypothesis, we taught one group of freshman students to use UNIX, Microsoft Word, and Microsoft Excel using commands and strategies, and compared them to another group of students who were taught only commands (Bhavnani 2000b; Bhavnani and John 2000; Bhavnani et al. 2001). Preliminary results from the experiment show that strategies could be taught effectively in the same amount of time as teaching just commands. Furthermore, the results also indicate that the students could transfer the strategies across applications. The analysis of strategies has therefore led to the reexamination of the content and delivery of computer literacy courses with promising results.

---

## 5.6 Summary and Future Research

To counteract the widespread inefficient use of computer applications, this paper identified and analyzed efficient strategies in the intermediate layers of knowledge. These strategies have three characteristics: (1) They are efficient because they exploit powers offered by computer applications such as iteration, propagation, organization, and visualization; (2) they need to be made explicit to users because the knowledge to use them is neither suggested by tools nor by task descriptions; and (3) they are generally useful across computer applications. The preceding characteristics inspired the design and testing of a strategic approach to training with promising results. These results suggest that the cost of learning and applying efficient strategies can be easily addressed by proper strategic instruction.

Based on our experience in teaching strategies, we believe that the identification of efficient strategies should be a key research goal. Therefore, we pose the question: Is there a framework that can systematically identify efficient strategies? There are several tantalizing clues that such a framework does exist. For example, we have observed that in addition to powers, computers also have limitations, such as in screen size, memory size, and processing speed. When task requirements exceed such resources, users may benefit by efficient strategies to circumvent the limitations (Bhavnani and John 1998). Therefore, powers, limitations, and their interactions could be the source of many strategies. A systematic identification of powers and limitations of computers could be an important step toward building the framework.



Another clue toward the framework is that efficient strategies in the intermediate layers could contain strategies at different levels of generality. For example, at one level, strategies could be relevant only to a particular application such as Microsoft Word. These strategies deal with eccentricities of the package but are generally useful for many tasks in that application. At another level, strategies could relate to an entire domain such as CAD but not outside. For example, strategies to precisely locate points using snap locks are generally useful across all CAD packages but not relevant to word processors. At yet another level of generality, strategies apply across domains, such as those on which we have focused in this paper. These levels could structure the search for efficient strategies.

Besides exploring a framework to identify efficient strategies, we are also exploring how strategies can guide the design of functionality. Designers could systematically check whether their designs provide the functionality to execute efficient strategies and test whether that functionality actually helps users become more efficient. Research on the systematic identification of strategies in the intermediate layers of knowledge can therefore not only lead to more effective ways of training but also to more principled methods to design functionality (Bhavnani 2000a). Both of these approaches should counteract the persistence of inefficient usage, which has plagued modern computers for many years.

---

## Acknowledgments

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of NSF or the U.S. government. The authors thank P. Polson, F. Reif, G. Vallabha, R. Young, and the reviewers for their contributions.

---

## References

- Anderson, J., and Lebiere, C. (1998). *The atomic components of thought*. Mahway, NJ: Lawrence Erlbaum Associates.
- Beakley, G., Autore, D., and Patterson, T. (1984). *Architectural drawing and design*. New York: Macmillan Publishing Company.
- Bhavnani, S. (1998). *How architects draw with computers: A cognitive analysis of real-world CAD interactions*. Unpublished Ph.D. dissertation. Carnegie Mellon University, Pittsburgh.
- Bhavnani, S. (2000a). Designs conducive to the use of efficient strategies. *Proceedings of the DIS'00 Conference*, 338–345. New York: ACM.

- Bhavnani, S. (2000b). Strategic approach to computer literacy. *Proceedings of the CHI'00 Conference on Human Factors in Computing Systems*, 161–162. New York: ACM.
- Bhavnani, S., Flemming, U., Forsythe, D., Garrett, J., Shaw, D., and Tsai, A. (1996). CAD usage in an architectural office: From observations to active assistance. *Automation in Construction* 5, 243–255.
- Bhavnani S., and John, B. (1996). Exploring the unrealized potential of computer-aided drafting. *Proceedings of the CHI'96 Conference on Human Factors in Computing Systems*, 332–339. New York: ACM.
- Bhavnani, S., and John, B. (1997). From sufficient to efficient usage: An analysis of strategic knowledge. *Proceedings of the CHI'97 Conference on Human Factors in Computing Systems*, 91–98. New York: ACM.
- Bhavnani, S., and John, B. (1998). Delegation and circumvention: Two faces of efficiency. *Proceedings of the CHI'98 Conference on Human Factors in Computing Systems*, 273–280. New York: ACM.
- Bhavnani, S., Reif, F., and John, B. (2001). Beyond Command Knowledge: Identifying and Teaching Strategic Knowledge for Using Complex Computer Applications. *Proceedings of the CHI'01 Conference on Human Factors in Computing Systems*. 229–236. New York: ACM.
- Bhavnani, S., and John, B. (2000). The strategic use of complex computer systems. *Human-Computer Interaction*, 15, 107–137.
- Bhavnani, S., John, B., and Flemming, U. (1999). The strategic use of CAD: An empirically inspired, theory-based course. *Proceedings of the CHI'99 Conference on Human Factors in Computing Systems*, 42–49. New York: ACM.
- Card, S., Moran, T., and Newell, A. (1983). *The Psychology of human-computer interaction*. Hillsdale, NJ: Lawrence Erlbaum Associates.
- Carroll, J., and Thomas, J. (1982). Metaphor and the cognitive representations of computing systems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-12, 107–116.
- Cragg, P., and King, M. (1993). Spreadsheet modeling abuse: An opportunity for OR? *Journal of the Operational Research Society*, 44, 743–752.
- Crosley, L. (1988). *The architect's guide to computer-aided design*. New York: John Wiley and Sons.
- Doane, S., Pellegrino, J., and Klatzky, R. (1990). Expertise in a computer operating system: Conceptualization and performance. *Human-Computer Interaction*, 5, 267–304.
- Douglas, S., and Moran, T. (1983). Learning text editor semantics by analogy. *Proceedings of the CHI'83 Conference on Human Factors in Computing Systems*, 207–211. New York: ACM.
- Flemming, U., Bhavnani, S., and John, B. (1997). Mismatched metaphor: User vs. system model in computer-aided drafting. *Design Studies* 18, 349–368.

- Gantt, M., and Nardi, B. (1992). Gardeners and gurus: Patterns of cooperation among CAD users. *Proceedings of the CHI'92 Conference on Human Factors in Computing Systems*, 107–117. New York: ACM.
- Green, T. (1989). Cognitive dimensions of notations. *People and Computers V: Proceedings of the Fifth Conference of the British Computer Society Human-Computer Interaction Specialist Group*, 443–460. Cambridge: Cambridge University Press.
- Halasz, F., and Moran, T. (1982). T.P. analogy considered harmful. *Proceedings of Human Factors in Computer Systems*, 383–386. Washington, DC: ACM.
- John, B., and Kieras, D. (1996). The GOMS family of user interface analysis techniques: Comparison and contrast. *Transactions of Computer-Human Interaction*, 3, 4, 320–351.
- Kieras, D. (1997). A guide to GOMS model usability evaluation using NGOMSL. In M. Helander, T. Landauer, and P. Prabhu (Eds.), *The Handbook of Human-Computer Interaction* (Second Edition), 733–766. Amsterdam: Elsevier Science B.V.
- Lang, G., Eberts, R., Gabel, M., and Barash, M. (1991). Extracting and using procedural knowledge in a CAD task. *IEEE Transactions on Engineering Management*, 38, 257–68.
- Lemaire P., and Siegler, R. (1995). Four aspects of strategic change: Contributions to children's learning of multiplication. *Journal of Experimental Psychology: General*, 124, 1, 83–97.
- Lewis, C., and Mack, R. (1982). Learning to use a text processing system: Evidence from “thinking aloud” protocols. *Proceedings of Human Factors in Computer Systems Conference*, 387–392. Washington, DC: ACM.
- Mack, R., Lewis, C., and Carroll, J. (1983). Learning to use word processors: Problems and prospects. *ACM Transactions on Office Information Systems*, 1, 245–271.
- Marchionini, G. (1989). Information seeking in electronic encyclopedias, *Machine-Mediated Learning*, 3, 3, 21–26.
- Majchrzak, A, Chang, T., Barfield, W., Eberts, R., and Salvendy, G. (1987). *Human Aspects of Computer-Aided Design*. London: Taylor Francis.
- Newell, A. (1990). *Unified theories of cognition*. Cambridge: Harvard University Press.
- Nilsen, E., Jong, H., Olson J., Biolsi, I., and Mutter, S. (1993). The growth of software skill: A longitudinal look at learning and performance. *Proceedings of INTERCHI'93*, 149–156. New York: ACM.
- Olson, J., and Olson, G. (1990). The growth of cognitive modeling in human-computer interaction since GOMS. *Human-Computer Interaction*, 5, 221–265.
- Rosson, M. (1983). Patterns of experience in text editing. *Proceedings of the CHI'93 Conference on Human Factors in Computing Systems*, 171–175. New York: ACM.

- Siegler, R., and Shipley, C. (1995). Variation, selection, and cognitive change. In G. Halford and T. Simon (Eds.), *Developing cognitive competence: New Approaches to process modeling*, 31–76. Hillsdale, NJ: Erlbaum.
- Singley, M., and Anderson, J. (1989). *The transfer of cognitive skill*. Cambridge, MA: Harvard University Press.
- Waern, Y. (1985). Learning computerized tasks as related to prior task knowledge. *International Journal of Man-Machine Studies*, 22, 441–455.