

The Strategic Use of Complex Computer Systems

Suresh K. Bhavnani and **Bonnie E. John**
Carnegie Mellon University

ABSTRACT

Several studies show that despite experience, many users with basic command knowledge do not progress to an efficient use of complex computer applications. These studies suggest that knowledge of tasks and knowledge of tools are insufficient to lead users to become efficient. To address this problem, we argue that users also need to learn strategies in the intermediate layers of knowledge lying between tasks and tools. These strategies are (a) efficient because they exploit specific powers of computers, (b) difficult to acquire because they are suggested by neither tasks nor tools, and (c) general in nature having wide applicability. The above characteristics are first demonstrated in the context of aggregation strategies that exploit the iterative power of computers. A cognitive analysis of a real-world task reveals that even though such aggregation strategies can have large effects on task time, errors, and on the quality of the final product, they are not often used by even experienced users. We identify other strategies beyond aggregation that can be efficient and useful across computer applications and show how they were used to develop a new approach to train-

Suresh Bhavnani specializes in computational design and human-computer interaction with a research focus on the identification, acquisition, and performance of efficient strategies to use complex computer systems; he is assistant professor at the School of Information in the University of Michigan. **Bonnie John** is an engineer and psychologist researching methods for usable systems design, especially computational models of human performance; she is an associate professor in the Human-Computer Institute, the Computer Science Department, and the Psychology Department of Carnegie Mellon University.

CONTENTS

- 1. INTRODUCTION**
 - 2. STRATEGIES IN THE INTERMEDIATE LAYERS OF KNOWLEDGE**
 - 2.1. Strategies That Exploit the Iterative Power of Computers
 - 2.2. Acquiring Strategies in the Intermediate Layers of Knowledge
 - 2.3. Generality of Strategies in the Intermediate Layers of Knowledge
 - 3. EVIDENCE FOR THE EFFECTS OF AGGREGATION STRATEGIES ON PERFORMANCE**
 - 3.1. The Panel Clean-Up Task
 - 3.2. How L1 Performed the Panel Clean-Up Task
 - 3.3. Cognitive Analysis of the Panel Clean-Up Task
 - 3.4. Inefficient Use Reported in Other Studies
 - 4. POSSIBLE EXPLANATIONS FOR INEFFICIENT COMPUTER USAGE**
 - 4.1. Efficient Strategies Not Known
 - Efficient Strategies Have Not Been Made Explicit
 - Weak Causal Relation Between Method and Quality of Product
 - Office Culture Not Conducive to Learning
 - 4.2. Efficient Strategies Known But Not Used
 - Efficiency Not Valued
 - Strategies Not Really Efficient
 - Prior Knowledge Dominating Performance
 - 4.3. Discussion of Possible Explanations of Inefficient Computer Usage
 - 5. GENERAL COMPUTER STRATEGIES BEYOND AGGREGATION**
 - 5.1. Propagation Strategies
 - 5.2. Organization Strategies
 - 5.3. Visualization Strategies
 - 5.4. Implications for Training
 - 6. SUMMARY AND FUTURE RESEARCH**
-

ing with promising results. We conclude by suggesting that a systematic analysis of strategies in the intermediate layers of knowledge can lead not only to more effective ways to design training but also to more principled approaches to design systems. These advances should lead users to make more efficient use of complex computer systems.

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 com-

plex computer systems such as UNIX[®] (Doane, Pelligrino, & Klatzky, 1990), word processors (Rosson, 1983), spreadsheets (Nilsen, Jong, Olson, Biolsi, & Mutter, 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 article 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 intermediate layer 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. Although 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: First, tap the nail to guarantee its proper angle of entry and to hold it in place. Second, remove the fingers holding the nail. Third, drive in the nail with heavier blows. The knowledge of this efficient method is expressed neither in the task description nor 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.

In this article we focus on efficient strategies to use computer applications that lie in the intermediate layers of knowledge. We show that these strategies are (a) efficient because they exploit specific capabilities provided by computers; (b) difficult to acquire from tool and task knowledge alone; and (c) general in nature, therefore having wide applicability.

Section 2 introduces the three previously mentioned concepts in the context of aggregation strategies that exploit the iterative power of computer applications. Section 3 provides empirical evidence that these strategies are not spontaneously acquired by experienced users but, if used, can reduce task time and errors. Section 4 discusses possible explanations why such strategies are not easily learned or used. Section 5 expands the notion of strategies beyond those to perform iterative tasks and briefly discusses their implications to strategic 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.

2. STRATEGIES IN THE INTERMEDIATE LAYERS OF KNOWLEDGE

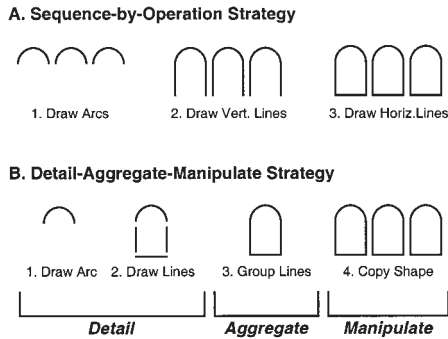
Complex computer applications such as UNIX[®], CAD, word processors, and spreadsheets often provide more than one way to perform a given task. Consider the task of drawing three identical arched windows in a CAD system. As shown in Figure 1A, one way to perform this task is to draw all the arcs across 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 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 of 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 described previously are prime examples of strategies that can be used in complex computer systems.

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 reduces tool switching, the user still must 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 show in Section 3 that aggregation

*Figure 1. Two strategies to perform the task of drawing three windows in a CAD system. From “Exploring the Unrealized Potential of Computer-Aided Drafting,” by S. K. Bhavnani and B. E. John, 1996, *Proceedings of CHI’96*, p. 337. Copyright 1996 ACM, Inc. Reprinted by permission.*



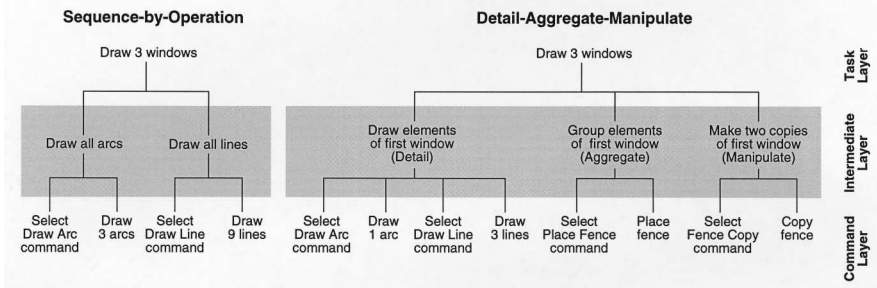
strategies are in fact much more efficient in terms of time and errors when compared to Sequence-by-Operation.

Figure 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 generalizability.

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 & John, 1997). Furthermore, when different strategies can accomplish the same task, the task itself also cannot express this strategic knowledge. This knowledge, therefore, must be learned by various processes such as through trial and error or through explicit instruction. In fact, we show in Section 3 that, despite mastery of basic commands, many users do not spontaneously acquire strategic knowledge to use commands efficiently.

Figure 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.



There is a cost to learning strategies such as Detail–Aggregate–Manipulate. Users must learn to recognize opportunities to operate on groups of objects to exploit iteration and then know a sequence of actions to execute the strategy. As shown in Figure 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, John, & Flemming, 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. An important reason that these strategies were easily learned was because repeated elements are intrinsic to architectural designs (Flemming, Bhavnani, & John, 1997). Windows, doors, columns, and even entire façades 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. Section 5.4 discusses preliminary results from our ongoing research, which shows that strategies can be taught in a short amount of time to a diverse population of freshmen students.

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 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 1B and in Bhavnani & 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–ModifyException 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 than to modify each on its own. The Aggregate–ModifyAll–ModifyException strategy can also be used both to modify the width of columns with an exception and 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 formulas 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 (a) efficient because they exploit the iterative power of computers, (b) difficult to acquire spontaneously from knowledge of commands or tasks, and (c) generalizable across computer applications. In the next section we analyze the first two points in more detail. First, we describe a GOMS analysis of a real-world task to precisely understand how aggregation strategies can affect performance. Second, we provide empirical evidence from other studies to show that aggregation strategies are not spontaneously acquired by even experienced users.

Figure 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. From "From Sufficient to Efficient Usage: An Analysis of Strategic Knowledge," by S. K. Bhavnani and B. E. John, 1997, *Proceedings of CHI'97*, p. 97. Copyright 1997 ACM, Inc.

	CAD	Spreadsheet	Word Processor												
Detail	1. Draw Lines 	1. Enter Data <table border="1" data-bbox="476 425 602 477"> <tr><td>D</td><td>E</td><td>F</td></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="723 425 878 477"> <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															
Aggregate	2. Fence Elements 	2. Define a Range Named "Set" <table border="1" data-bbox="476 512 602 564"> <tr><td>D</td><td>E</td><td>F</td></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="723 512 878 564"> <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															
Manipulate	3. Copy Fence 	3. Operate on Range <table border="1" data-bbox="459 598 671 651"> <tr><td>D</td><td>E</td><td>F</td><td>G</td></tr> <tr><td>22.3</td><td>65.8</td><td>78.4</td><td>=sum(\$B1:\$D1)</td></tr> </table>	D	E	F	G	22.3	65.8	78.4	=sum(\$B1:\$D1)	3. Copy Address to Other Files <table border="1" data-bbox="723 581 912 651"> <tr><td>John Doe</td></tr> <tr><td>John Doe</td></tr> <tr><td>100 Main Street, USA</td></tr> <tr><td>100 Main Street, USA</td></tr> </table>	John Doe	John Doe	100 Main Street, USA	100 Main Street, USA
D	E	F	G												
22.3	65.8	78.4	=sum(\$B1:\$D1)												
John Doe															
John Doe															
100 Main Street, USA															
100 Main Street, USA															
Aggregate	1. Fence All Elements 	1. Select All Columns <table border="1" data-bbox="476 685 556 737"> <tr><td>D</td><td>E</td><td>F</td><td>G</td></tr> </table>	D	E	F	G	1. Select Text <table border="1" data-bbox="723 685 843 737"> <tr><td>XXXX XXXX XXX X</td></tr> <tr><td>XXX XXX XXX XX XX</td></tr> <tr><td>XX XXXXXX XX</td></tr> </table>	XXXX XXXX XXX X	XXX XXX XXX XX XX	XX XXXXXX XX					
D	E	F	G												
XXXX XXXX XXX X															
XXX XXX XXX XX XX															
XX XXXXXX XX															
Modify (all)	2. Modify All Elements 	2. Modify All Columns <table border="1" data-bbox="476 772 637 824"> <tr><td>D</td><td>E</td><td>F</td><td>G</td></tr> </table>	D	E	F	G	2. Change Style of All Text <table border="1" data-bbox="723 772 843 824"> <tr><td>XXXX XXXX XXX X</td></tr> <tr><td>XXX XXX XX XX XX</td></tr> <tr><td>XX XXXXXX XX</td></tr> </table>	XXXX XXXX XXX X	XXX XXX XX XX XX	XX XXXXXX XX					
D	E	F	G												
XXXX XXXX XXX X															
XXX XXX XX XX XX															
XX XXXXXX XX															
Modify (exception)	3. Modify Exception 	3. Modify Exception <table border="1" data-bbox="476 841 614 894"> <tr><td>D</td><td>E</td><td>F</td><td>G</td></tr> </table>	D	E	F	G	3. Modify Exception <table border="1" data-bbox="723 841 843 894"> <tr><td>XXXX XXXX XXX X</td></tr> <tr><td>XXX XXX XX XX XX</td></tr> <tr><td>XX XXXXXX XX</td></tr> </table>	XXXX XXXX XXX X	XXX XXX XX XX XX	XX XXXXXX XX					
D	E	F	G												
XXXX XXXX XXX X															
XXX XXX XX XX XX															
XX XXXXXX XX															
Locate	1. Locate Similar Dwg. 	1. Locate Similar Set of Formulae <table border="1" data-bbox="476 928 654 980"> <tr><td></td><td>A</td></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="723 928 832 980"> <tr><td>XXXXXXXXXXXXXXXX</td></tr> <tr><td>✓ XXXX</td></tr> <tr><td>✓ XXXXXXXXXXXX</td></tr> </table>	XXXXXXXXXXXXXXXX	✓ XXXX	✓ XXXXXXXXXXXX			
	A														
1	=sum(\$C1:\$C12)														
2	=(A1/12)														
XXXXXXXXXXXXXXXX															
✓ XXXX															
✓ XXXXXXXXXXXX															
Aggregate	2. Fence Elements 	2. Select Formulae <table border="1" data-bbox="476 1015 654 1085"> <tr><td></td><td>A</td></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="723 1015 832 1085"> <tr><td>XXXXXXXXXXXXXXXX</td></tr> <tr><td>✓ XXXX</td></tr> <tr><td>✓ XXXXXXXXXXXX</td></tr> </table>	XXXXXXXXXXXXXXXX	✓ XXXX	✓ XXXXXXXXXXXX			
	A														
1	=sum(\$C1:\$C12)														
2	=(A1/12)														
XXXXXXXXXXXXXXXX															
✓ XXXX															
✓ XXXXXXXXXXXX															
Manipulate	3. Copy Shape 	3. Copy Formulae <table border="1" data-bbox="476 1119 648 1171"> <tr><td></td><td>D</td></tr> <tr><td></td><td>=sum(\$C1:\$C12)</td></tr> <tr><td></td><td>=(D1/12)</td></tr> </table>		D		=sum(\$C1:\$C12)		=(D1/12)	3. Copy Section <table border="1" data-bbox="723 1119 832 1171"> <tr><td>XXXXXXXXXXXXXXXX</td></tr> <tr><td>✓ XXXX</td></tr> <tr><td>✓ XXXXXXXXXXXX</td></tr> </table>	XXXXXXXXXXXXXXXX	✓ XXXX	✓ XXXXXXXXXXXX			
	D														
	=sum(\$C1:\$C12)														
	=(D1/12)														
XXXXXXXXXXXXXXXX															
✓ XXXX															
✓ XXXXXXXXXXXX															
Modify	4. Modify Shape 	4. Modify Formulae <table border="1" data-bbox="476 1206 648 1258"> <tr><td></td><td>D</td></tr> <tr><td></td><td>=sum(\$E1:\$E12)</td></tr> <tr><td></td><td>=(D1/12)</td></tr> </table>		D		=sum(\$E1:\$E12)		=(D1/12)	4. Modify Section <table border="1" data-bbox="723 1206 872 1258"> <tr><td>XXXXXXXXXXXXXXXX</td></tr> <tr><td>✓ VVVV VVVV</td></tr> <tr><td>✓ VVVVVVVVVVVVVVVVV</td></tr> </table>	XXXXXXXXXXXXXXXX	✓ VVVV VVVV	✓ VVVVVVVVVVVVVVVVV			
	D														
	=sum(\$E1:\$E12)														
	=(D1/12)														
XXXXXXXXXXXXXXXX															
✓ VVVV VVVV															
✓ VVVVVVVVVVVVVVVVV															

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 2 years 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 is 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.

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 4, L1 had the task to remove 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 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 4. This meant that L1 had to modify numerous lines that overlapped the vents.

3.2. How L1 Performed the Panel Clean-Up Task

L1 zoomed in and panned a single window to view sets of vents to work on. Figure 4 represents a typical example of such a window setup, with 3 of the 21 vents displayed. As shown in Figure 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 clean-up operations, L1 is effectively using the Sequence-by-Operation strategy described in Section 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 as L1 has to select each panel line 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 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 completed the task. Finally, he committed five slips in the selection of panel lines that had to be repeatedly reselected to get exactly the line he wanted. Despite these difficulties, L1 con-

Figure 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. From “Delegation and Circumvention: Two Faces of Efficiency,” by S. K. Bhavnani and B. E. John, 1998, *Proceedings of CHI’98*, p. 275. Copyright 1998 ACM, Inc. Reprinted by permission.

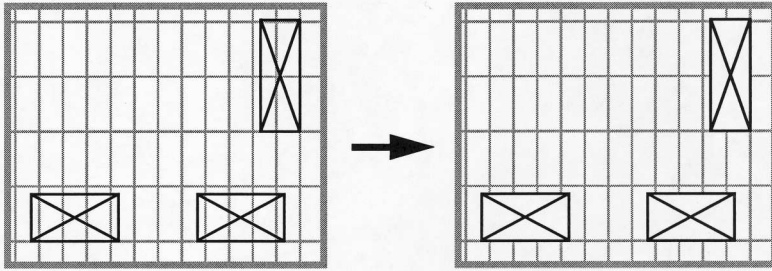
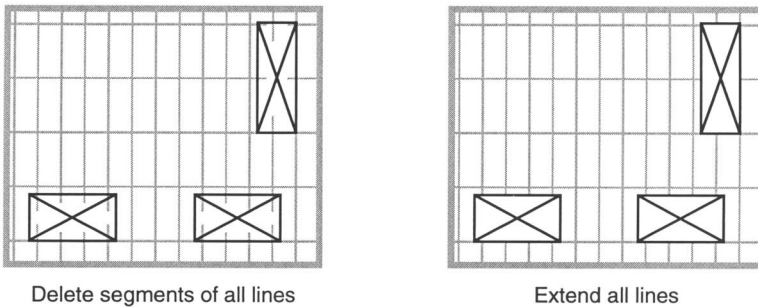


Figure 5. The method used by L1 to perform the panel clean-up task. From “Delegation and Circumvention: Two Faces of Efficiency,” by S. K. Bhavnani and B. E. John, 1998, *Proceedings of CHI’98*, p. 276. Copyright 1998 ACM, Inc. Reprinted by permission.



sistently 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 min to complete the entire task.

To precisely understand the nature of these inefficiencies in terms of time and frequency of errors, the data were transcribed at the keystroke level and quantitatively analyzed. As shown in Figure 7, L1 took more than 2 min to complete the fairly simple task of deleting 11 very similar line segments (these numbers relate to the clean-up of 3 vents—the total task, as described earlier, involved the clean-up of 21 vents). Furthermore, he spent 20 sec to commit and recover from errors that formed 16% 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

Figure 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.

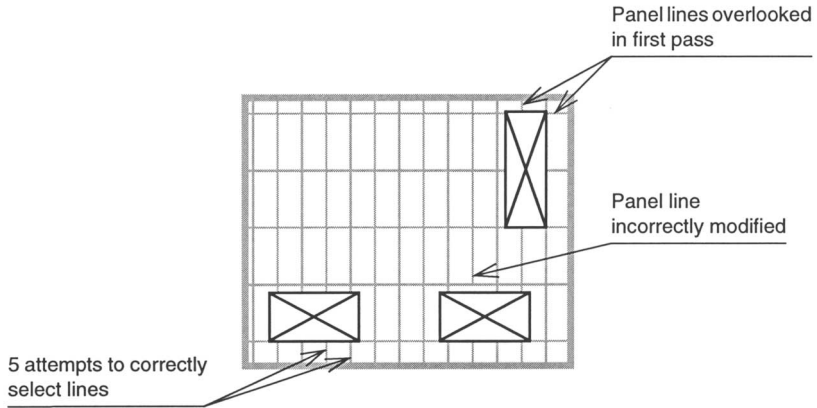
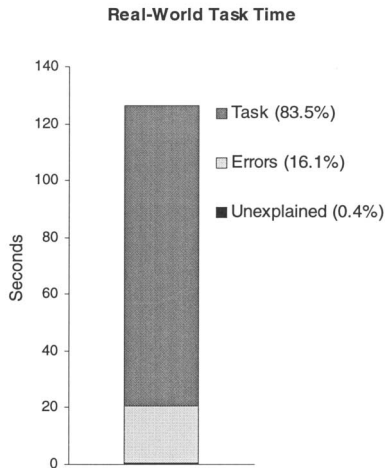


Figure 7. Total time to complete the three-vent clean-up task including the time to commit and recover from errors. Errors are all actions in which a correct goal was incorrectly executed (slips). Unexplained behavior includes all behavior in which it was not obvious what goal was being achieved.



and cleaning many lines. For instance, L1 could have used the place fence¹ 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 those errors of not noticing lines that had to be cut and extended could therefore have been avoided. Furthermore, because the iterative task of cleaning up each line would be delegated to computer, it appears that the strategy could have reduced the time to perform the task.

3.3. Cognitive Analysis of the Panel Clean-Up Task

To understand the differences between the Sequence-by-Operation and Aggregate-Modify strategy to perform the panel clean-up task, we first constructed hierarchical goal decompositions of each approach. Figure 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. Figure 8 shows how L1's strategy choice resulted in many low-level mouse inputs. Figure 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, Moran, & Newell, 1983) models of both approaches. As shown in Figure 10, the model with the Aggregate-Modify strategy predicted a reduction in time of 71%. Furthermore, as shown in Figure 11, the frequencies of inputs were different between the two models. Although there is an increase in the number of command selections (as the place 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 sec to the overall time.

1. Ethnographic notes revealed that L1 had used the place 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.

Figure 8. A GOMS decomposition of the three-vent panel clean-up task using the Sequence-by-Operation strategy to clean up each vent.

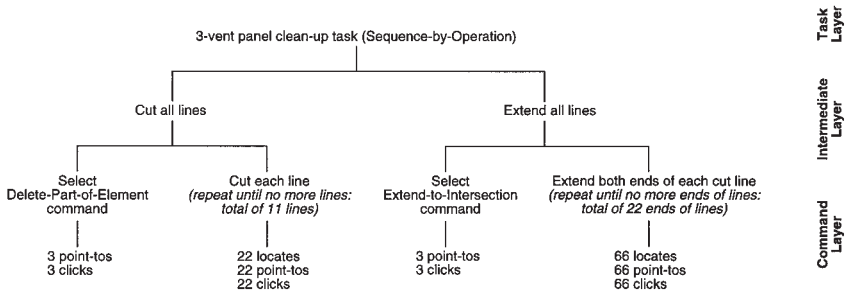
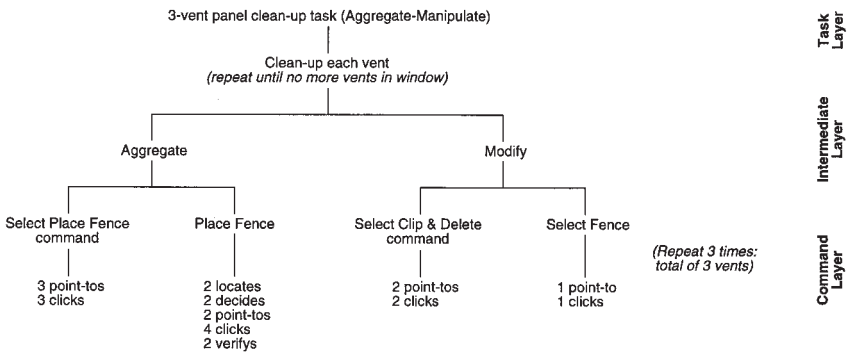


Figure 9. A GOMS decomposition of the three-vent panel clean-up task using the Aggregate-Modify strategy to clean up each vent.



The analysis of the panel clean-up task reveals many issues related to strategy use. First, despite experience and knowledge of the place 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, reduced errors, and led to a more accurate product.

Figure 10. The Aggregate-Modify strategy used in the ideal model could reduce the time to do the panel clean-up task by 71%.

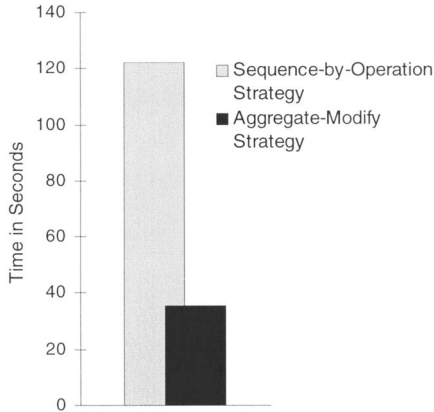
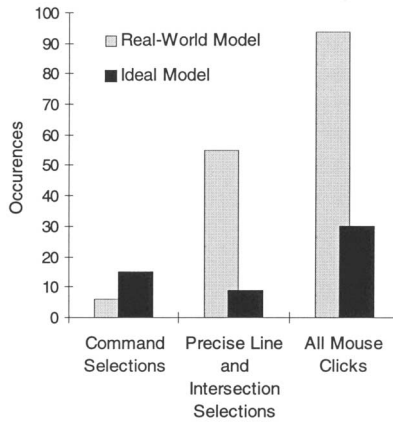


Figure 11. Change of input frequencies between the real-world data and ideal model for the three-vent panel clean-up task.



3.4. Inefficient Use Reported in Other Studies

The above 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% of their time to complete their tasks if they had used various forms of

the aggregation strategies as shown in Figure 3. The above results are also not unique to our study of CAD usage. Lang, Eberts, Gabel, and Barash (1991) reported 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% 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 previous results generalize even outside the domain of CAD. Nilsen et al. (1993) studied the development of 26 graduate students of business 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 the second 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 3.

In a different study on spreadsheet use, Cragg and King (1993) showed that 55% 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 3.

The above 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.

Although 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. In the next section we explore possible explanations why many users do not acquire and use efficient strategies.

4. POSSIBLE EXPLANATIONS FOR INEFFICIENT COMPUTER USAGE

Why do experienced users not learn and not use efficient strategies, and why do these inefficient behaviors persist? This section presents possible explanations under two broad categories: (a) efficient strategies not known, and (b) efficient strategies known but not used. These explanations are derived from empirical studies done on computer applications in which efficient strategies were not used, from 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.

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. Although 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.

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) described the importance of “thinking CAD.” Crosley stated, “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 added that “the advantages of CAD are not free; they come at the expense of having to actually design the drawing” (p. 11). Although this author stressed the importance of rethinking the drawing process, he did not present explicit strategies to design the drawing, leaving the readers to discover and implement the strategies themselves.

Weak Causal Relation Between Method and Quality of Product

Although 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 from getting smudged and drawings getting dirty is to always begin work at the upper left-hand corner of the sheet of drafting paper and to finish at the lower right-hand corner of the sheet (Beakley, Autore, & Patterson, 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 relation between technique and quality, and because the flaws are publicly visible, drafters tend to be highly motivated to improve their technique.

This strong causal relation 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) noted that “productions² 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).

Office Culture Not Conducive to Learning

The above explanations focus on an individual's interaction with a CAD system. However, real-world CAD usage typically occurs in a group environment in which information is exchanged. This exchange can strongly affect the usage of a CAD system. For example, Gantt and Nardi (1992) recommended that CAD managers encourage gurus to develop expert knowledge and to act as disseminators of this information within an organization. Majchrzak, Chang, Barfield, Eberts, & Salvendy (1987) provided several recommendations: Managers should be well-trained in the technology, CAD

2. Singley and Anderson (1989) discussed a model of cognition in which knowledge is encoded as if-then rules called *productions*.

training should focus on presenting a general education in CAD concepts thereby moving away from teaching only commands, and users should have biweekly meetings in which 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. In addition, 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 with 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 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 exact situation occurred 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.

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.

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 do not care for the savings in time and 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) predicted 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 & 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.

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 to not use them. Although 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 clean-up 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 (Figures 8 and 9), the small amount 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.³ 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 (e.g., for a discussion of such domain knowledge known by architects, see Flemming et al., 1997). 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 4.1 for a discussion of learning costs) 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 to 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

3. 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, Moran, & Newell, 1983), Soar (Newell, 1990), and ACT-R (Anderson & 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 & 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 three-vent panel clean-up task (Figure 9) has a deeper goal stack than the Sequence-by-Operation strategy for the same task (Figure 8). Whether a deeper goal stack adds to performance time for skilled use is an open research question (John & Kieras, 1996). Card et al. tried both approaches and found no additional predictive power from assigning time to goal decomposition; therefore they 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 msec to 100 msec. Again, because 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.

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 the above reasons, we believe that in simple task situations similar to those we and others have observed (Doane et al., 1990; Nilsen et al., 1993; Rosson, 1983), 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 trade-off 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 trade-offs.

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 & Thomas, 1982; Douglas & Moran, 1983; Halasz & Moran, 1982; Lewis & Mack, 1982; Mack, Lewis, & Carroll, 1983; 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 & Anderson, 1989).

The strong effects of prior knowledge may explain LI's interactions. Prior to using CAD, LI 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 LI. Because LI 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 LI had been cued to a better way, he may have switched to the better strategy.

4.3. Discussion of Possible Explanations of Inefficient Computer Usage

The preceding sections 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.

Although 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 in which this may not hold (e.g., 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 in which time is important to users.

The cost of acquiring an efficient strategic level of knowledge is currently very high—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 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 articles, 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. In Section 5.4 we present evidence that if strategic knowledge is presented explicitly

and in a carefully designed educational context, the cost of attaining such knowledge can be negligible when compared to the cost of learning the command-level knowledge required to use a new application.

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 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.⁴ As shown in Figure 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 2, the following section discusses how the seven strategies in the above 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.⁵ All these strategies appear to be intuitively efficient but need to be rigorously tested through future research.

4. We do not yet have a principle to generate these powers. We therefore do not claim that this list is complete.

5. Green (1989) analyzed similar concepts such as hidden–explicit dependencies and viscosity–fluidity in the framework of cognitive dimensions.

Figure 12. Seven general strategies beyond aggregation strategies and how they are useful in word processing, spreadsheet, and CAD tasks.

General Strategies	Word-Processing Examples	Spreadsheet Examples	CAD Examples
Propagation			
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
Organization			
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 3D model from 2D floor plan
Visualization			
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

5.1. Propagation Strategies

The first two strategies in Figure 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 (a) new objects inherit properties or receive information from another object, and (b) modifications can propagate through the dependencies. For 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 façade while using a CAD system.

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 (a) enhance comprehension and (b) 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 three-dimensional graphic objects can be generated from two-dimensional representations, and vice-versa.

5.3. Visualization Strategies

The last three strategies in Figure 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 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 may 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 may need to make many precise changes to different parts of a large floor plan. A magnified view is needed to make the precision changes, whereas 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 that automatically updates the local magnified view where the user can make the precise modifications. As shown in Figure 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, and we do not understand how to systematically identify efficient strategies from these powers. However, we are convinced that teaching such strategies would benefit users. The following section describes how we developed a new approach to training called the strategic approach to computer literacy, based on the strategies of aggregation, propagation, organization, and visualization that we have been able to identify.

5.4. Implications for Training

To address the difficulty that users have in acquiring efficient strategies, we used the previously mentioned general strategies to design a new computer literacy course. The focus of our approach is to teach strategies in addition to commands. We hypothesized that this combination would not only make users more efficient (compared to those who only learned commands in the con-

text of simple tasks) but also enable users to transfer the knowledge across applications.

The method of instruction in the strategic approach was suggested by our GOMS representation of a strategy (Bhavnani et al., 1999). For example, the Detail–Aggregate–Manipulate strategy was modeled as a combination of a selection rule and a method. The selection rule connects the nature of the task (replication) to a strategy label (Detail–Aggregate–Manipulate); the method decomposes the label into subgoals (Detail, Aggregate, Manipulate). The selection rule suggested that a student must “learn to see” when a task offers an opportunity to use a particular strategy. The method component suggested that a student must “learn to do” the strategy by decomposing the task into temporally ordered subgoals.

The above approach was used to design a 7-week computer literacy course at Carnegie Mellon University to teach freshman how to strategically use UNIX[®], Microsoft[®] Word[®], and Microsoft[®] Excel[®]. For example, in the Learning to See step, students were taught the Operate-on-Groups-of-Objects strategy (an aggregation strategy) in UNIX[®]. They were first shown two ways to move many files sharing the same extension: (a) Move one file at a time, and (b) move multiple files with the wild-card operator (e.g., mv*.jpg images). The first method was shown to be repetitious, time consuming, and error prone compared to the second method. They were then explicitly taught when a wild card could be used to operate on groups of files sharing the same extension. This example was then generalized to the Operate-on-Groups-of-Objects strategy. In the Learning to Do step, students executed the same strategy on their own for a similar task. Later in the course, the same strategy was taught in Microsoft[®] Word[®] and in Microsoft[®] Excel[®], with different commands to emphasize its general nature.

The strategic version of the course was compared to the traditional version of the course that taught the same commands as the strategic approach, but without the strategies. Preliminary results (Bhavnani, 2000b; Bhavnani, Reif, & John, in press) show that strategies could be taught effectively in the same amount of time as teaching just commands. Furthermore, there was no statistical difference between the mean scores of both groups (96.07 control, 95.54 experimental) in regular exams that tested command knowledge. The results also showed evidence that the students could transfer the strategies across applications. Extensive analysis of data is being conducted to understand the effects of the strategic approach on a wide range of variables such as gender, major, class attendance, task time, and errors. The analysis of strategies has therefore led to the reexamination of the content and delivery of computer literacy courses with promising results.

6. SUMMARY AND FUTURE RESEARCH

To counteract the widespread inefficient use of computer applications, we identified and analyzed efficient strategies in the intermediate layers of knowledge. These strategies have three characteristics: (a) They are efficient because they exploit powers offered by computer applications such as iteration, propagation, organization, and visualization; (b) they need to be made explicit to users because the knowledge to use them is suggested neither by tools nor by task descriptions; and (c) they are generally useful across computer applications. The above characteristics inspired the design and testing of a strategic approach to computer literacy 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 following 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 screen size, memory size, and processing speed. When task requirements exceed such resources, users may benefit by efficient strategies to circumvent the limitations (Bhavnani & 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 be 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 that we have focused on in this article. 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 therefore can lead not only 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.

NOTES

Background. This article is based on Suresh K. Bhavnani's Ph.D. thesis and subsequent postdoctoral research done at the Human-Computer Interaction Institute in Carnegie Mellon University.

Acknowledgments. The views and conclusions contained in this document are ours and should not be interpreted as representing the official policies, either expressed or implied, of the National Science Foundation or the U.S. Government. We thank P. Polson, F. Reif, G. Vallabha, R. Young, and the reviewers for their contributions.

Support. This research was supported by the National Science Foundation, Award# IRI-9457628 and EIA-9812607.

Authors' Present Addresses. Suresh K. Bhavnani, School of Information, University of Michigan, Ann Arbor, MI 48109. E-mail: bhavnani@umich.edu. Bonnie E. John, Human-Computer Interaction Institute, School of Computer Science, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213-3890. E-mail: Bonnie.John@cs.cmu.edu.

HCI Editorial Record. First manuscript received March 15, 1999. Revision received November 15, 1999. Accepted by Peter Polson, Clayton Lewis, and Wendy Kellogg. Final manuscript received May 2000. – *Editor*

REFERENCES

- Anderson, J., & Lebiere, C. (1998). *The atomic components of thought*. Mahwah, NJ: Lawrence Erlbaum Associates, Inc.
- Beakley, G., Autore, D., & Patterson, T. (1984). *Architectural drawing and design*. New York: Macmillan.
- Bhavnani, S. (1998). *How architects draw with computers: A cognitive analysis of real-world CAD interactions*. Unpublished doctoral 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., & Tsai, A. (1996). CAD usage in an architectural office: From observations to active assistance. *Automation in Construction*, 5, 243-255.
- Bhavnani, S., & 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., & 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., & 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., John, B., & 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.
- Bhavnani, S., Reif, F., & John, B. (in press). 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*. New York: ACM.
- Card, S., Moran, T., & Newell, A. (1983). *The psychology of human-computer interaction*. Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.
- Carroll, J., & Thomas, J. (1982). Metaphor and the cognitive representations of computing systems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-12, 107–116.
- Cragg, P., & 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: Wiley.
- Doane, S., Pellegrino, J., & Klatzky, R. (1990). Expertise in a computer operating system: Conceptualization and performance. *Human-Computer Interaction*, 5, 267–304.
- Douglas, S., & 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., & John, B. (1997). Mismatched metaphor: User vs. system model in computer-aided drafting. *Design Studies*, 18, 349–368.
- Gantt, M., & 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, England: Cambridge University Press.
- Halasz, F., & Moran, T. (1982). T. P. analogy considered harmful. *Proceedings of Human Factors in Computer Systems*, 383–386. Washington, DC: ACM.
- John, B., & Kieras, D. (1996) The GOMS family of user interface analysis techniques: Comparison and contrast. *Transactions of Computer-Human Interaction*, 3, 320–351.
- Kieras, D. (1997). A guide to GOMS model usability evaluation using NGOMSL. In M. Helander, T. Landauer, & P. Prabhu (Eds.), *The handbook of human-computer interaction* (2nd ed., pp. 733–766). Amsterdam: Elsevier.
- Lang, G., Eberts, R., Gabel, M., & Barash, M. (1991). Extracting and using procedural knowledge in a CAD task. *IEEE Transactions on Engineering Management*, 38, 257–268.

- Lemaire, P., & Siegler, R. (1995). Four aspects of strategic change: Contributions to childrens learning of multiplication. *Journal of Experimental Psychology: General*, 124(1), 83–97.
- Lewis, C., & 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., & Carroll, J. (1983). Learning to use word processors: Problems and prospects. *ACM Transactions on Office Information Systems*, 1, 245–271.
- Majchrzak, A., Chang, T., Barfield, W., Eberts, R., & Salvendy, G. (1987). *Human aspects of computer-aided design*. London: Taylor & Francis.
- Marchionini, G. (1989). Information seeking in electronic encyclopedias. *Machine-Mediated Learning*, 3(3), 21–26.
- Newell, A. (1990). *Unified theories of cognition*. Cambridge, MA: Harvard University Press.
- Nilsen, E., Jong, H., Olson J., Biolsi, I., & 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., & 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., & Shipley, C. (1995). Variation, selection, and cognitive change. In G. Halford & T. Simon (Eds.), *Developing cognitive competence: New approaches to process modeling* (pp. 31–76). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.
- Singley, M., & 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.