Delegation and Circumvention: Two Faces of Efficiency

Suresh K. Bhavnani
School of Architecture
Carnegie Mellon University
Pittsburgh PA 15213 USA
+1 412 363 8308
suresh@andrew.cmu.edu

Bonnie E. John
HCI Institute
Carnegie Mellon University
Pittsburgh PA 15213 USA
+1 412 268 7182
Bonnie.John@cs.cmu.edu

ABSTRACT
Throughout history, inefficient methods to use devices have been replaced by more efficient ones. This shift typically occurs when users discover how to *delegate* work to the powers of a tool, and to *circumvent* its limitations. Strategies of delegation and circumvention, therefore, appear to be the core of efficient use. To show how this approach can explain the relationship between tools and strategies in complex computer systems, we describe five ways to perform a real-world drawing task with current as well as future tools. We then present five corresponding GOMS models that demonstrate the value of efficient strategies when compared to the observed behavior of a professional CAD user. We conclude by presenting a generalized framework to characterize efficient strategies and discuss its relevance to design and training.

Keywords
Strategies, CAD, GOMS, efficiency, productivity.

INTRODUCTION
Records from early civilizations show that humans throughout history have developed devices and processes to assist in the efficient performance of tasks. The Sumerians, for example, first began to write on clay tablets by scratching marks on their surface. Over 700 years, this method of writing gradually changed into the cuneiform script comprising of wedge-shaped marks pressed into clay with a reed stylus. Historians suggest that this change was largely motivated by scribes who discovered that pressing marks into the clay instead of scratching upon its surface was not only faster, but also more durable over time [11].

While the Sumerians may have had the luxury to gradually develop a more efficient method of writing on wet clay, today's users of computer tools have far less attention and time resources to become efficient. As complex applications such as CAD and word-processors continue to explode with a profusion of new tools, users level off into sufficient usage patterns that tend not to exploit potential efficiencies. Furthermore, several longitudinal and real-world studies suggest that neither good design nor experience can ensure that users move from a sufficient to an efficient use of computer tools [4, 6, 8, 14].

Based on observations of real-world use of complex systems, we have argued that *strategies* hold the key to efficient usage, and identified several that were shown to be not only powerful, but also generally useful across applications [3, 4]. However, we need a more systematic approach to understand the relationship between computer tools and efficient strategies. Similar to frameworks that help us understand and prevent errors through better design [15, 16], this paper defines a framework to help us understand and identify efficient strategies.

We begin by analyzing examples from history to understand the relationship between tools and the strategies to use them efficiently. These examples suggest that as new and more powerful tools evolved in the past to improve product and performance, they often presented new limitations either due to inherent constraints, or due to the new tasks that were attempted. The efficient use of such tools therefore required a combination of *delegation strategies* that exploited the powers of these tools, as well as *circumvention strategies* that dealt with their limitations. The circumvention strategies transformed either the process, or the task, or triggered the development of a more powerful tool, in which case the process repeated.

To show how the relationship between tools and strategy is relevant to computer usage, we analyze a real-world CAD task performed by a user during an ethnographic study [2]. Starting from how such a drawing task would be done using manual tools, we explore the effect of the increasing sophistication in CAD tools on strategies to complete the task. In order to test the hypothesis that these strategies are actually efficient and of value to users, we present the results from five GOMS models. These models range from how the user performed the task on a current version of the system, to how the same task could be done in future versions. The analysis demonstrates that regardless of how powerful computer systems become, their efficient usage requires delegation and circumvention strategies. However, while these strategies can have strong effects on product and performance, they may not be obvious to even experienced computer users.

CHI 98 Los Angeles CA USA
Copyright 1998 0-89791-975-0/98/ 4..$5.00

We then present a framework of strategies focused on iterative tasks, and show how it can be generalized. In conclusion we discuss how such a framework could be used to explain the relationship between tools and strategies to users, as well as to help designers more systematically explore the effects of their designs.

CIRCUMVENTION AND DELEGATION IN HISTORY

There are numerous examples throughout history that show the relationship between tools and efficient methods to use them. The following three illustrate some important concepts about efficient methods.

According to surviving records, most scribes of ancient civilizations either wrote from right to left or from top to bottom. But around the fifth century BC the Greeks began a dramatic reversal; for a period their writing zigzagged horizontally in a style known as boustrophedon (as the ox plows), before they settled down to the modern style of writing in English from left to right [7, 11]. What might have caused this gradual but radical shift?

Historians and calligraphy scholars note that this change coincided with the replacement of the reed brush by the reed pen as the dominant writing tool [11]. The reed pen, whose tip, if cut and used properly, offered a more precise way of making marks on papyrus. However, unlike the reed brush, the reed pen could easily catch in the fibers of rough papyrus. Historians hypothesize that the Greeks, over a period of 200 years, realized that it was more efficient to pull the reed pen across the rough papyrus rather than to push it. For a right-handed scribe, this meant writing from left to right. Therefore the Greeks used new methods to exploit the hard material of the reed pen to be more precise, and overcame its limitation by changing the direction of writing.

At a later stage in the history of writing, the reed pen and papyrus were replaced by the much more precise quill and smooth parchment, both of which offered a new range of possibilities to the medieval scribe. Illuminated manuscripts of that era show extremely detailed floral and abstract patterns in brilliant colors. But these elaborate decorations greatly increased the time to produce such books as demonstrated by the Book of Kells from Ireland, which was never completed [11]. One of the incomplete pages reveals how its illuminators had sped up the process. This page, like others, contained several independent patterns that had to be painted in many colors. Although the illuminator could have completed each pattern in turn with all its colors, the page shows that all patterns across the page had been painted with yellow. By finishing *all* areas of one color, the painter had reduced tool switching between paint brushes or color. In addition, this procedure may have allowed the yellow paint to dry on the first pattern while subsequent patterns were painted yellow; consequently another color paint could be applied immediately after the last pattern's yellow was complete. Despite this task organization, such processes still remained time-consuming, and the replication of books was laboriously slow.

The production of books was radically changed with the invention of the printing press. With the advent of this new technology, the construction and replication of individual pages could be done far more rapidly compared to the manual process. However, early printers in Germany took time to realize that the way to exploit the iterative power provided by the press was by *not* adding anything to the copies. For instance, many early books tried to copy the style of scribed manuscripts by hand-painting first letters of paragraphs on pages *after* they were printed. Later printers abandoned this approach, making the book production process far more efficient.

The above examples provide several insights into the evolution of efficient methods. First, none of the efficient methods employed were absolutely necessary to complete the tasks but were employed as they improved either the quality of the product, or performance of the tasks. Because such methods are *goal-oriented* and *non-obligatory,* they have been defined as *strategies* [4, 17].

Second, each new tool offered new or increased powers and, by using the tools efficiently, users could exploit these new powers. These methods can therefore be referred to as *delegation strategies.* In the case of the reed pen and quill, by using proper strategies to prepare and use the tools, users could delegate precision to the tools; with the printing press, users could delegate the iterative task of replication to the printing process.

However, new limitations often accompany new powers, either due to inherent constraints in the tool, or when new tasks are attempted with the new tools. When users perceive such limitations, they may make a change to the process (such as the change in writing direction by the Greeks), or move to another tool (such as the printing press). These transformations can therefore be regarded as *circumvention strategies.*

While the above explanation of circumvention and delegation strategies through history is plausible, what relevance does it have to the evolution of computer tools and the strategies to use them? Moreover, are the efficient strategies to use these tools obvious to experienced users?

THE EFFECT OF DRAWING TOOLS ON EFFICIENT STRATEGIES

To demonstrate how drawing tools affect strategies, we present a real-world task performed by a CAD user during an ethnographic study [2]. This task and user behavior is typical of 9 other users currently being analyzed [1] (for example, see [2] and [3] for the analysis of another user's interaction from the same data). One of the users from that study, "L1", had more than 2 years experience in using a CAD system called MicroStation™ (version 4). His task was to edit a drawing containing ceiling panels in an architectural plan that overlapped air-condition vents. Such repetitive tasks are common during the detail drawing stage of a building design. As vents go vertically through ceiling panels, they both cannot occupy the same space. Therefore, as shown in Figure 1, the task is to remove all the line segments representing ceiling panels that overlap the rectangles representing the air-condition vents.

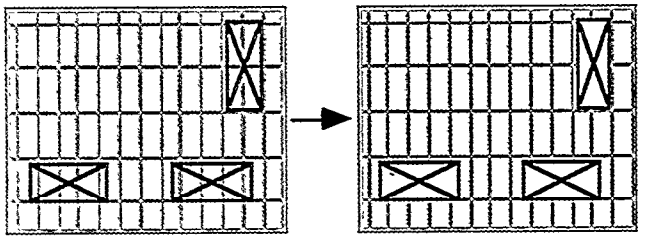

Figure 1. The panel clean-up task requires all ceiling panel lines that overlap the air-condition vents to be modified.

The vents and panels are defined in two different drawing files that are simultaneously displayed on the screen to reveal their overlap. The vents (shown here in black), were placed in a file by an engineer, and are displayed on the screen by L1 as a REFERENCE FILE, where elements can be viewed and selected, but not edited. The panels (shown in gray), were placed by an architect and viewed by L1 as an editable file. L1 can therefore modify the panels with respect to the vents. (This separation of information is necessary to enable architects and engineers to perform tasks independently and at different stages of the design process).

The file has 21 such vents, all of them similar to those shown in Figure 1. L1 zoomed in and panned a single window in order to frame sets of these vents to work on. The rectangle around the vents in Figure 1 represents a typical example of such a window setup, with 3 of the 21 vents showing. The remainder of this paper will refer to editing the panels overlapping these three vents as the *panel clean-up task*.

Depending on the tools available to an architect, the panel clean-up task can be performed in several ways[1]. To understand the relationship between tools and efficient ways to perform the task, we begin by describing strategies to use simple drawing tools available in manual drafting as well as in CAD, and progress towards more sophisticated CAD tools that either exist or are being prototyped. At each stage we shall explore the powers and limits of the tools and discuss efficient strategies to deal with them. It is important to remember that more sophisticated tools usually appear *in addition to* less sophisticated tools in complex computer applications. Thus, users always have a choice of strategies, some of which may have been efficient in older versions of the application, but which are inefficient relative to the new tools.

[1] There are at least two ways that obviate the panel clean-up task. The first is to draw the panel lines only *after* the vents have been placed. But this is not a general solution as the locations of vents typically change during the evolution of the design, in which case the architect has no option but to modify all the panel lines again. The second approach is to make the vents opaque and place them *over* the panel lines, therefore masking the overlap. While this approach could produce a correct printed out drawing in some CAD systems, it does not produce an accurate CAD model on which other operations such as automatic dimensioning can be performed. The strategies presented in this paper assume that the panels and vents can be placed in any order, and the task requires an

Powers and Limits of Precision Tools

It is well-known that manual drafting tools provide users with the ability to create precise graphic elements. For example, the T-square provides constraints to the movement of a pencil and enables a user to draw precise horizontal and parallel lines. Therefore, given a T-square and a set-square, a user can *delegate* the act of achieving precision in horizontal and vertical lines to the tools. With such manual drafting tools, one way to perform the panel clean-up task is to erase a segment of each line to an approximate distance, followed by precisely redrawing the lines to meet accurately at the vents.

As shown in Figure 2a, this approach to the panel clean-up task can also be performed on CAD using equivalent precision tools. The DELETE-PART-OF-ELEMENT tool could cut each panel line, and EXTEND-TO-INTERSECTION could extend it precisely to the boundary of the vent. We call cutting and extending each line in turn an instance of the *Sequence-by-Element* strategy.

The Sequence-by-Element strategy is sufficient for a small number of elements. However, when this strategy is used for iterative tasks with many elements, which are typical in the production of architectural designs (e.g., our user had to clean up 21 vents, each with about four overlapping elements), they can offer little assistance beyond precision. The delete and extend tools offer only the ability to apply single operations to single elements, and therefore have to be repeatedly applied to each line overlapping each vent. One way to circumvent this limitation, at least partially, is to reorganize the task by reducing the number of times a tool is switched. Therefore, instead of selecting the DELETE-PART-OF-ELEMENT tool and applying it to a line, then selecting EXTEND-TO-INTERSECTION and applying it to the line, a more efficient method as shown in Figure 2b, is to delete segments of *all* relevant lines, followed by an extension of *all* the erased lines to meet the vent boundary. We call this method an instance of the *Sequence-by-Operation* strategy.

Sequence-by-Operation addresses the problem of repeated tool-switching in the Sequence-by-Element strategy, but no matter how these tasks are reorganized, precision tools just cannot assist much in iterative tasks. A more general approach to circumvent the limitation of precision tools is to use an entirely different set of tools, which allow the delegation of iteration.

Powers and Limits of Iteration Tools

A wide range of applications such as CAD, spreadsheets, and word-processors provide assistance for iterative tasks through *aggregation*. Aggregation refers to the ability to group disjoint elements in various ways and to manipulate these groups with powerful operators [4]. There appear to be three types of aggregation tools: those that allow for the application of single operations on element aggregates, the application of operation aggregates on single elements, and the application of operation aggregates on element aggregates. We call the general class of strategies to use these tools the *Aggregate-Modify* (AM) strategies.

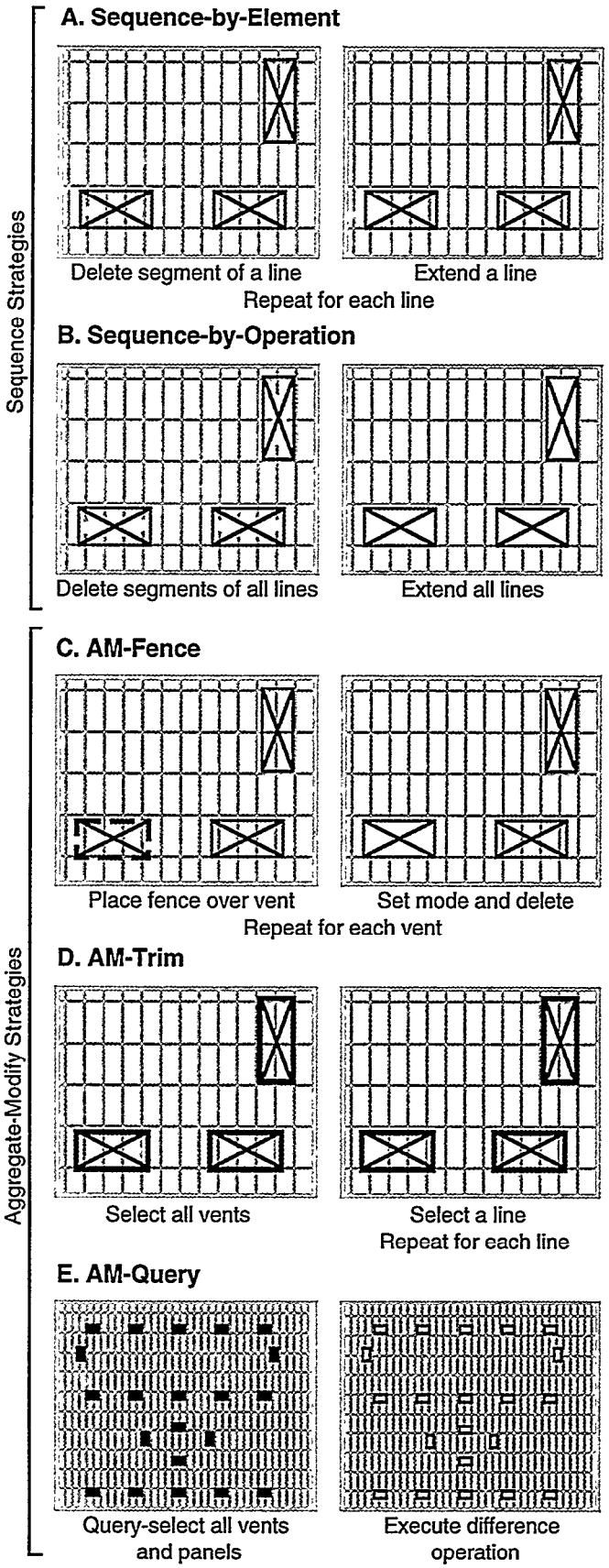

Figure 2. Five strategies to perform the panel clean-up task. The vents and panels are schematic and are not drawn to scale.

Single Operations on Element Aggregates

An example of a tool that allows a user to apply single operations on an aggregate of elements is the PLACE-FENCE command provided by MicroStation™. This command allows a user to place a shape over any number of elements, set a mode of element selection (such as the CLIP-MODE which selects only segments of elements that are inside the fence boundary in the aggregate, therefore replacing the extend operation) and manipulate or modify the resulting set.

As shown in Figure 2c, the FENCE tool could be used to perform the panel clean-up task. To use the FENCE tool, the user must place a rectangular fence on top of the vent by accurately selecting its vertices, setting the CLIP-MODE, and then applying the delete operation to the aggregate. The computer responds by deleting all the element segments within the boundary of the fence. (Since the vents are in a REFERENCE FILE, they will be unaffected by this operation). The crucial aspect of this strategy is to aggregate before modification, which relieves the user from operating on each element. However, the fence tool requires single operations (for example FENCE DELETE) to be independently applied to element aggregates, and there can be only one fence active at a time.

Operation Aggregates on Single Elements

One way to circumvent the single-operation limitation of the FENCE tool is to use the TRIM tool instead. With this tool, a user can select any number of "cutting elements" which define the limits to which intersecting elements will be deleted. Therefore the delete and the extend operation are aggregated for the user over all the vents. However, to disambiguate which segment of an intersecting line is to be deleted, the command requires the user to click on the appropriate segment of each element.

These new powers and limitations allow for a different instantiation of the AM strategy. As shown in Figure 2d, this new tool could be used by first selecting all the vents. When the TRIM command is invoked, the selected vents are interpreted as the limits to which the lines must be modified. As the user selects each line, the aggregated operation is applied to each line by deleting *and* extending each line. Therefore, while this new command addresses the limitation of the fence command which does not aggregate operations, it forces the user to once again iterate over single elements.

Operation Aggregates on Element Aggregates

The fundamental limitation of the FENCE and TRIM tools is that they deal with primitive elements such as lines and shapes, forcing the user in this case to deal with each vent.

Circumventing the limitations of FENCE and TRIM for complex iterative tasks requires a different paradigm of CAD, where users do not place lines and shapes, but *domain objects* such as ceiling panels and vents. As these objects would reside in a database, users could perform sophisticated searches using queries enabling the aggregation and manipulation of information in a much more powerful way [18]. For example, as shown in Figure 2e, one can imagine a command that allows a user to

retrieve *all* occurrences of ceiling panels and vents that overlap. This aggregate of vents and panels can be processed by using another command to perform a difference operation between the geometry of each panel and the vents that it overlaps.

The above strategies can be organized in a matrix as shown in Figure 3.

THE VALUE OF EFFICIENT STRATEGIES

Although Figure 3 allows easy identification and classification of sequence and AM strategies, it remains to be shown that the AM strategies are actually more efficient for real-world tasks, and whether they have value for real users. To rigorously understand the effects of strategies on performance, we conducted an NGOMSL analysis (a variant of GOMS [5] described in [12]) on all five strategies to perform the panel clean-up task. Each model implemented one of the strategies in Figure 2 down to the keystroke level. For example, the goal hierarchy (excluding keystrokes for the lowest subgoal) for the AM-Fence strategy (Figure 2c) is shown below:

Goal: Edit Design
. Goal: Modify Multiple Objects *(Repeat for all vents)*
. . Goal: Determine Strategy *(In this case, AM)*
. . . Goal: Execute Aggregate-Modify Strategy
. . . . Goal: Aggregate Objects
. . . . . Goal: Place Fence
. . . . Goal: Modify Objects
. . . . . Goal: Delete Elements
. . . . . Goal: Set Clip Mode
. . . . . Goal: Execute Delete Command

The execution times predicted by the NGOMSL models were then compared to L1's real-world behavior to determine the potential for improved productivity.

A GOMS Comparison of Five Strategies

The five NGOMSL models developed using GLEAN [19] (a GOMS interpreter), produced estimates for the execution times for each of the strategies on the panel clean-up task for three vents. As shown in Figure 4, the execution time drops as more and more iterations are delegated to the computer. While Sequence-by-Operation saves some time by grouping operations, the AM strategies produce greater time savings by delegating iteration.

The analysis however revealed one unexpected outcome. The AM-Trim model predicts a time of almost 17 seconds less than the AM-Fence model. This was initially puzzling

Operations		Elements		Strategies
Single	Agg.	Single	Agg.	
X		X		Sequence-by-Element Sequence-by-Operation
X			X	AM-Fence
	X	X		AM-Trim
	X		X	AM-Query

Figure 3. Five iteration strategies based on combinations of functionalities.

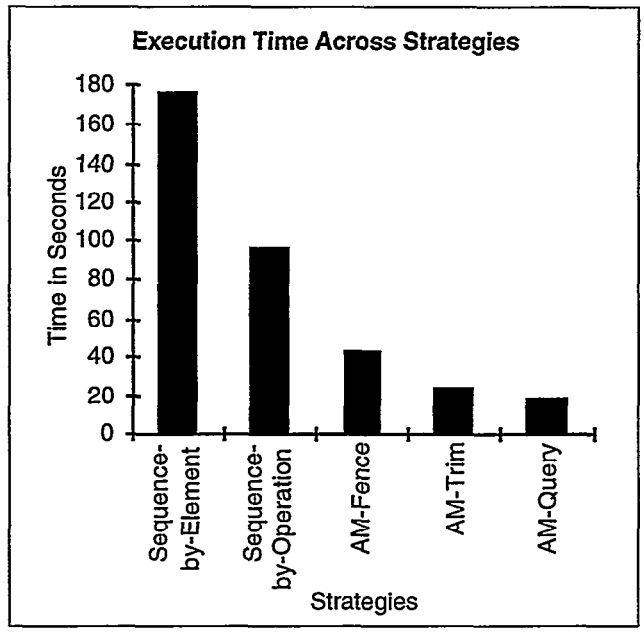

Figure 4. Comparison of the predicted execution times for different strategies to perform the 3-vent panel clean-up task. The AM-Query strategy modifies the panel lines that overlap all 21 vents instead of just 3.

as we expected the selection of individual lines to be more time consuming than using a fence. Investigating the details of this prediction revealed a general principle of these strategies. The models were most sensitive to an increase in the largest undelegated iteration. As shown in Figure 3, the Sequence and AM-Trim models did not support element aggregation and therefore were most sensitive to the number of panel lines. Because the AM-Fence model could not aggregate over operations, it was most sensitive to the number of vents. The AM-Query strategy aggregates over both elements and operations, so its model is neither sensitive to the number of vents nor number of lines, and the time shown in Figure 4 for 3 vents would be the same for the entire 21-vent task. The AM-Trim model and the AM-Fence model are therefore in competition because the number of lines more or less offsets the number of vents in the 3-vent task.

To test this hypothesis, a series of AM-Trim models were executed while keeping the number of vents constant and increasing the number of lines. Figure 5 shows that the predicted execution time for the AM-Trim model is equal to that of the AM-Fence model at around 32 lines, but this time steadily increases as the number of lines increase. Therefore, when the number of lines is small, and the operations are restricted to cut and extend, the TRIM tool is advantageous. But when the number of elements increases, the FENCE tool produces better performance. Our 3-vent example task involves 11 lines, so the TRIM tool wins in Figure 4.

How L1 Performed the Panel Clean-up Task

Figure 4 shows a factor of nine reduction in time from the least efficient strategy to the most efficient strategy. But this theory-based information is only valuable to users if they are not already using efficient strategies. To understand the

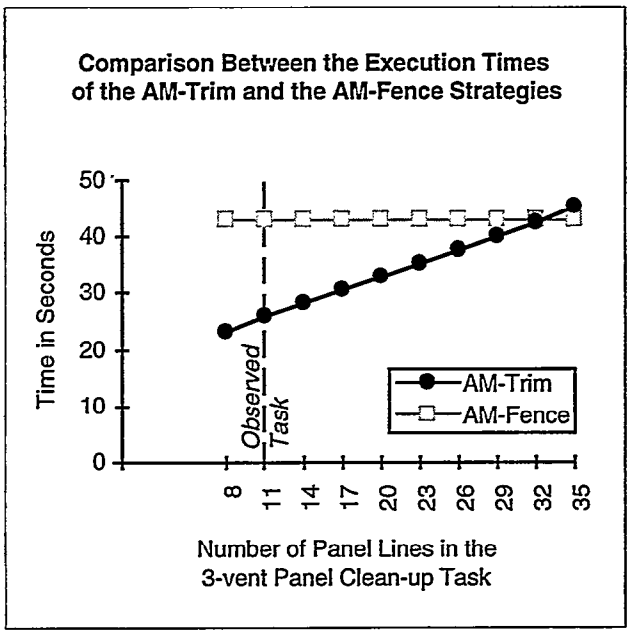

Figure 5. While the execution time for the AM-Fence strategy remains constant, the execution time for the AM-Trim strategy is directly proportionate to the number of panel lines.

potential for productivity improvement, we examined L1's behavior in the context of the specific application he used. MicroStation™ version 4, provided aggregation of elements with the FENCE tool, but no tools for the AM-Trim or AM-Query strategies. However, although L1 had previously used the FENCE tool (based on our ethnographic data), he consistently used the Sequence-by-Operation strategy to delete and extend each line that overlapped all of the 21 vents. The Sequence-by-Operation model is within 11% of the error-free[2] real-world data. If L1 had used the AM-Fence strategy to cut and extend groups of lines, it would have taken him 40% of the time to complete the task.

In addition, L1 committed many errors which added 20.33 seconds to the error-free time. He committed both errors of omission and commission. First, he did not notice that a panel line located very close to the boundary of the upper right-hand vent overlapped the vent; he had to return to it 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 boundary, thereby extending a panel-line to the wrong location. This error of commission went undetected and the drawing was inaccurate after he completed the task. Finally, he committed many slips in the selection of panel and vent lines, which required him to repeatedly re-select to get exactly the line he wanted.

L1 could have used the AM-Fence strategy with a SNAP mouse option (where the cursor jumps to the closest intersection thereby delegating precision to the computer) to accurately place the fence over the vent. With 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 of not noticing lines that had to be cut and extended, would not have occurred. This would have reduced performance time *and* increased accuracy.

The above analyses demonstrate two important points. First, when work is delegated to the computer, there are fewer opportunities for errors. Second, even experienced users such as L1, doing extremely repetitive tasks, tend to miss opportunities to delegate work to the computer.

We are not alone in observing that users do not use efficient strategies to delegate iteration to computers. Other studies in CAD [13], as well as in other domains such as spreadsheet use [6, 14] show similar results where users performing iterative tasks also missed opportunities to delegate iteration to computers. The above analyses of strategies to deal with iterative tasks, their effects on performance, as well as the empirical data, provided the basis to develop a generalized framework for iteration that might be useful across domains.

THE ITERATION FRAMEWORK

As discussed earlier with reference to Figure 3, the different tools to assist in iteration could be characterized in terms of single or aggregate operators applied to single or aggregate elements. It is combinations of these functionalities that produce various designs of tools which, in turn, require particular strategies to exploit them. Figure 6 is an extension of Figure 3 to include these relationships in a more detailed framework. Each row defines a particular combination of operator and element types, which is directly related to what can and cannot be delegated to the computer. The strategies emanate from these powers and limitations. The Aggregate-Modify strategies (shaded gray), exploit the power of iteration tools by delegating iteration. Circumvention strategies can range from transforming a task decomposition as demonstrated by the move to the Sequence-by-Operation strategy, to using an entirely different set of tools (moving to a different row in the framework, if the tools exist). CAD tools are shown as examples.

Because the framework shows the explicit relationship between abstract functionalities, tools, and strategies, it can be applied in four different ways: to identify strategy-instantiations in other domains, to explore the design of tools, to describe behavior, and to train users in the repercussions of strategies.

Identification of Strategy Instantiations

Since the iteration framework is structured around abstract functionalities instead of specific tools, it can be used for any application where these particular sets of functionalities appear. For instance, using a STYLES tool on multiple paragraphs in a word-processor can be seen as instantiation of the fourth row in the matrix. The efficient strategy to perform such a task is to define a style with multiple attributes such as bold and italic (aggregate operations), and

[2] The error-free time (105.7 sec.) was calculated by subtracting the time L1 spent to commit, search for, and recover from errors (20.33 sec.), plus unexplained behavior (0.53 sec.), from the total time (126.56 sec.) he spent editing the 3 vents.

Operations		Elements		Example	Powers	Limitations	Efficient Strategies
Single	Agg.	Single	Agg.	CAD Tools	Delegated		
X		X		Line-manip. tools	Precision	Iteration cannot be delegated	Sequence-by-Operation
X			X	Fence, Fence Delete	Iteration over elements	No iteration over operations	AM-Fence
	X	X		Shift-Select, Trim	Iteration over operations	No iteration over elements	AM-Trim
	X		X	Query, Boolean Op.	Iteration over operations and elements	No Propagation	AM-Query

Figure 6. The iteration framework showing the relationship between abstract functionality, tools, and strategies for tasks requiring multiple operations on multiple elements. Precision tools (shown in white) cannot delegate iteration. This limitation can be partially circumvented by reorganizing the task as described by the Sequence-by-Operation strategy, but more fully addressed by using iteration tools with AM strategies (shown in gray). Iteration tools themselves have various limitations which can be circumvented by using increasingly sophisticated iteration tools, or through the use of future propagation tools (not shown).

apply them to multiple paragraphs (aggregate elements), particularly if there are many iterations of this task.

Design of New Tools

To demonstrate the use of this framework to inform design, consider the TRIM tool. The specific operation of this tool was taken from MicroStation™ version 5, and as shown in Figure 6, its limitation is that it does not allow aggregation over elements.

The previous analysis overcame this limitation by going to an entirely new paradigm for CAD where domain-objects are manipulated in a database (a paradigm used by several architectural research projects). However, the framework pinpoints the limitation, which inspires a re-design without changing underlying paradigms. In this re-design, a more specific version of the TRIM tool could allow the selection of closed shapes which could act as cookie cutters on all elements that overlap them. Since the lines to delete would be encompassed by the closed shapes, this would be unambiguous and not require the user to identify each segment. Given this modification, the user could now select all vents and delete all overlapping elements in one step, effectively delegating all iteration to the computer.

Description of Behavior

The framework can also be used to describe how users interact with a system. Since the framework provides a continuum of powers starting from no delegation to the delegation of iteration over operations and elements, one can identify explicitly the level at which a particular user performs a particular task. Instructors could use this information to diagnose lack of knowledge on the part of the user and decide which concepts (e.g., aggregation of operations) and skills to teach (e.g., use of the TRIM tool).

Design of Training

We believe that there is nothing inherently wrong with the way new tools evolve. Often metaphors of older technologies are the only way to start exploring a new technology as its development, usage, and exploration go hand-in-hand. But we do think that the way tools are *introduced* requires a more systematic approach, and when efficient strategies shift, users must be made explicitly aware, not of just *what* the new tools are (as is currently done), but also *how* they directly affect the nature of tasks.

The framework could therefore be used to design training. For instance, users can be taught to recognize opportunities to delegate work to computer powers, as well as to circumvent their limitations Exercises could focus on planning and what we have called *Learning-to-See* (elements and operations to profitably aggregate). In addition, users can do tasks using several different strategies in order to demonstrate the differences in performance these strategies afford. Here, exercises could focus on execution and what we call *Learning-to-Do.* We are currently exploring this approach of Learning-to-See and Learning-to-Do in a course on CAD for architecture graduate students.

TOWARDS A GENERAL FRAMEWORK OF EFFICIENT STRATEGIES

Clearly there are other powers of computer applications beyond precision and iteration. Thus, we are developing a larger framework of which the iteration framework is just a small segment. In addition to precision and iteration, we are currently investigating other powers such as propagation, visualization, and generation as discussed below.

Powers and Limits of Propagation Tools

Although future iteration tools that operate on domain objects with queries are powerful, they also have limitations. For instance, each time a change in the vent layout occurs, the user must remember to make the appropriate changes to panels. Furthermore, these tools can provide little help if the engineer decides to move the vents *after* the panel lines have been modified. Depending on the way the vents are moved, the panel lines could be in complete disarray; some would be partially overlapping the vent, and some not touching them at all. This would force the user into a labor-intensive process to search for, and extend each cut line that did not terminate at a vent.

One way to overcome these limitations is to provide the power of constraint propagation. With tools of propagation, one can imagine future systems where ceiling panels "know" about their relationship to vents and vice versa, and any change in vents can automatically propagate to the

ceiling panels. However one can already expect problems to emerge in such systems. For instance, once ceiling panels are modified, they could violate some other constraint leading to endless cycles of propagation where the user is completely out of control. Tools and strategies to circumvent such limitations will therefore have to be defined.

Powers and Limits of Visualization Tools

One of the most important powers that make computer applications useful to architects is the power of visualization. With this power, users can visualize complex objects such as buildings in many different ways without having to alter the underlying representation. However the screen size of most current systems puts a severe constraint on how much information can be viewed at the same time. Users often face the trade-off between visual detail and the scope of information displayed on the screen. One way to circumvent this limitation in CAD is to have two windows: one to always provide an overview of the entire building, and the other zoomed far into the details of a section. Procedures for easily navigating between these views already occur in many CAD systems like MicroStation™. However, as with the iteration strategies, our data showed that L1 did not use this useful circumvention strategy and spent unnecessary time panning and zooming looking for the panel lines that had to be modified.

Powers and Limits of Generation Tools

While iteration and propagation can modify and replicate existing elements, computer applications with powerful algorithms can also generate new kinds of information not explicitly provided by the user. For example, future systems will enable users to explore designs generated by computers based on constraints and rules [10]. However such systems incur huge overheads in their setup and modification, and appear to be useful mainly for recurring problem types (e.g., floor plans of hospitals, dorms, barracks). Therefore they may require a whole new set of strategies that have yet to be encountered.

CONCLUSION

Strategies of delegation and circumvention appear to be the core of efficient use of complex systems. Understanding the relationship between abstract functionality, tools, and strategies can assist us in the development, training, and efficient use of complex applications such as CAD. Our analysis showed that regardless of how sophisticated CAD tools may become in the future, it appears they will always have powers as well as limitations, which users must learn to delegate and circumvent.

ACKNOWLEDGMENTS

This research was supported by the National Science Foundation, Award# IRI-9457628. 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 acknowledge the contributions of O. Akin, U. Flemming, J. Garrett, H. Simon, G. Vallabha; and Bentley Systems, Inc. for the academic edition of MicroStation™.

REFERENCES

1. Bhavnani, S. K. How Architects Draw with Computers: An Analysis of Real-World CAD Usage, *Ph.D. dissertation* (in preparation) Carnegie Mellon University, Pittsburgh.
2. Bhavnani, S.K., Flemming, U., Forsythe, D.E., Garrett, J.H., Shaw, D.S., and Tsai, A. CAD Usage in an Architectural Office: From Observations to Active Assistance. *Automation in Construction* 5 (1996), 243-255.
3. Bhavnani, S.K., and John, B.E. Exploring the Unrealized Potential of Computer-Aided Drafting. *Proceedings of CHI '96* (1996), 332-339.
4. Bhavnani, S.K., and John, B.E. From Sufficient to Efficient Usage: An Analysis of Strategic Knowledge. *Proceedings of CHI '97* (1997), 91-98.
5. Card, S.K., Moran, T.P., and Newell, A. *The Psychology of Human-Computer Interaction.* Hillsdale, NJ: Lawrence Erlbaum Associates, 1983.
6. Cragg, P.B. and King, M. Spreadsheet Modeling Abuse: An Opportunity for OR? *Journal of the Operational Research Society* 44 (1993), 743-752.
7. Denman, F. *The Shaping of our Alphabet.* Alfred A. Knopf, New York, 1955.
8. Doane, S.M., Pellegrino, J.W., Klatzky, R.L. Expertise in a Computer Operating System: Conceptualization and Performance. *Human-Computer Interaction* 5 (1990), 267-304.
9. Flemming, U., Bhavnani, S.K., John, B.E. Mismatched Metaphor: User vs. System Model in Computer-Aided Drafting. *Design Studies* 18 (1997), 349-368.
10. Flemming, U., Woodbury, R. Software environment to support early phases in building design (SEED): Overview. *Journal of Architectural Engineering, ASCE,* 1(4) (1995), 147-152.
11. Jackson, D. *The Story of Writing.* Taplinger Publishing Co., Inc., 1981.
12. Kieras, D. A Guide to GOMS Model Usability Evaluation using NGOMSL. in M. Helander & T. Landauer (eds.) *The handbook of human-computer interaction* (Second Edition). Amsterdam: North-Holland (in press).
13. Lang, G.T., Eberts, R. E., Gabel, M. G., and Barash, M.M. Extracting and Using Procedural Knowledge in a CAD Task. *IEEE Transactions on Engineering Management,* 38 (1991), 257-68.
14. Nilsen, E., Jong H., Olson J., Biolsi, I., Mutter, S. The Growth of Software Skill: A Longitudinal Look at Learning and Performance. *Proceedings of INTERCHI '93.* (1993), 149-156.
15. Norman, D. *The Design of Everyday Things.* Doubleday, New York, 1988.
16. Reason J. *Human Error.* Cambridge University Press, 1994.
17. Siegler, R.S., Jenkins, E. *How Children Discover New Strategies.* Lawrence Erlbaum Associates, New Jersey, 1989.
18. Snyder, J., Aygen, Z., Flemming, U. and Tsai, J. SPROUT - A modeling language for SEED, in *Journal of Architectural Engineering, ASCE,* 1(4) (1995), 195-203.
19. Wood, S. *GLEAN - GOMS Language Evaluation and Analysis.* University of Michigan, 1995.