Designs Conducive to the Use of Efficient Strategies

Suresh K. Bhavnani
HCI Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213 USA
Tel: +1-412-268-5663
E-mail: suresh@cmu.edu

ABSTRACT

Studies on the widespread inefficient use of complex computer applications have suggested that users need to learn efficient strategies in addition to learning how to use tools. This paper argues that our growing understanding of strategic knowledge can be used to guide designers develop systems which are conducive to the use of efficient strategies. The paper first describes ten general strategies which appear to be useful across three computer application domains. Next, the paper discusses the functionalities required to execute the ten strategies, and what makes them conducive to strategy use. An analysis of six major computer applications in three domains reveals that these functionalities are not consistently offered, and how their absence directly affects the performance of complex tasks. The analysis leads to questions related to the generality of the results, the problem of featurism, and how strategy-conducive systems could facilitate the transfer of knowledge across applications. The paper concludes by briefly describing how we intend to use the strategy framework to develop analysis methods for designers and trainers.

Keywords

Strategy, strategic knowledge, efficiency, design, training.

1. INTRODUCTION

Advances in interaction design such as direct manipulation, and its widespread acceptance, have greatly contributed to the ease with which users begin to use computer applications. Increasingly, users are able to explore and begin to use applications they have never used before, often with little or no training.

However, the knowledge to use direct manipulation tools does not seem to aid users to perform complex tasks efficiently. Several studies have shown that despite training and many years of experience, many users with basic command knowledge do not progress to an efficient use of applications [3, 6, 9, 10, 12]. Recent research has led us to believe that the efficient use of computer applications requires strategic knowledge in addition to tool knowledge [1, 5, 6]. The analysis of strategic knowledge has helped identify a set of strategies which are generally useful across computer applications. For example, computer users can perform many iterative tasks efficiently by using strategies which exploit the aggregation of objects [6] provided by common applications such as word processors, spreadsheets, and computer-aided drafting (CAD) systems. This paper explores how our understanding of general strategies can be used to guide designers to check if their designs are conducive to the use of efficient strategies.

The paper begins by presenting ten general strategies which appear to be useful across three computer domains: word processors, spreadsheets, and CAD systems. Next, the paper describes how these strategies are dependent on specific functionalities being present in an application, and the design issues relevant to making them conducive to strategy use. An analysis of six major applications in the above three domains reveals that even major applications do not consistently provide important functionalities required to execute efficient strategies. The absence of these functionalities directly impacts opportunities to perform tasks efficiently.

The above analysis prompts several questions related to the general applicability of the results, the problem of featurism, and how strategy-conducive systems could facilitate the transfer of strategic knowledge. The paper addresses these questions and concludes by presenting two research directions which we are pursuing. First, the development of an analysis method to guide designers to check if their designs are conducive to the use of efficient strategies. Second, the development of principles, based on strategic knowledge, which could guide the design of more effective training. As both these approaches are based on the common strategic framework developed in this paper, it is hoped that it leads to more collaboration between designers and trainers with the ultimate goal of making users more efficient in the use of complex computer applications.

2. STRATEGIES THAT GENERALIZE

Several empirical studies have aided in our understanding about the efficient use of complex computer applications. These studies include the observation of real-world users [3, 9, 12], cognitive analysis of their interactions [1, 4, 5, 6, 7], the analysis of functionalities provided by complex computer applications [6], and attempts to teach the efficient use of complex computer applications [2, 8].

A key result from the above research is that complex computer applications (such as word processors, spreadsheets, and CAD systems) typically provide many methods to complete the same task. These *non-obligatory* and *goal-directed* methods have been called strategies [4]; the knowledge of these alternate methods and how to choose between them has been referred to as *strategic knowledge*.

Our cognitive analyses of real-world usage revealed that some strategies are more efficient than others in terms of performance variables such as task time, errors, and ease of modifying content [1, 3, 7]. Furthermore, our analysis of complex computer applications led to the identification of three efficient strategies which generalize across computer applications [6, 7]. This paper expands that set to ten general strategies. These strategies were identified through empirical observations, and by exploring the powers and limitations of complex computer applications [4]. We are currently attempting to identify a more systematic approach to discover such strategies.

As shown in Table 1, the ten strategies fall under four categories: iteration, propagation, organization, and visualization. Iteration strategies exploit the power of computers to operate on groups of objects. Such strategies have the potential to reduce the time and errors to perform iterative tasks [1, 4, 5, 7]. Propagation strategies exploit the power of computers to modify objects which are connected through explicit dependencies. These strategies allow users to propagate changes to large numbers of inter-connected 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 [11].

The following discussion will attempt to show how the ten strategies in the above four categories are useful and meaningful in word processing, spreadsheet, and CAD tasks. While many of the strategies may appear obvious, empirical evidence shows that opportunities to use them are often missed by users performing complex tasks [1, 9, 12].

2.1 Iteration Strategies

The first three strategies in Table 1 exploit the power of computers to perform iteration over groups of objects. Strategy 1 encourages the reusing (and modification if appropriate) of existing groups of objects instead of creating new ones from scratch. This strategy is useful regardless of the data objects being created (e.g. paragraphs in word processors, formulas in spreadsheets, or graphic elements in CAD systems). Strategy 2 reminds users to check if an original group of objects is complete and error-free before making many copies, a useful strategy to avoid having to make changes in each of the copies. Once again checking before replication is a general strategy regardless of the data objects involved.

Finally, Strategy 3 specifies that exceptions in a group should be handled (dropped or added) before applying operations on the

Table 1. Ten general strategies and how they are useful in word processing, spreadsheet, and CAD tasks.

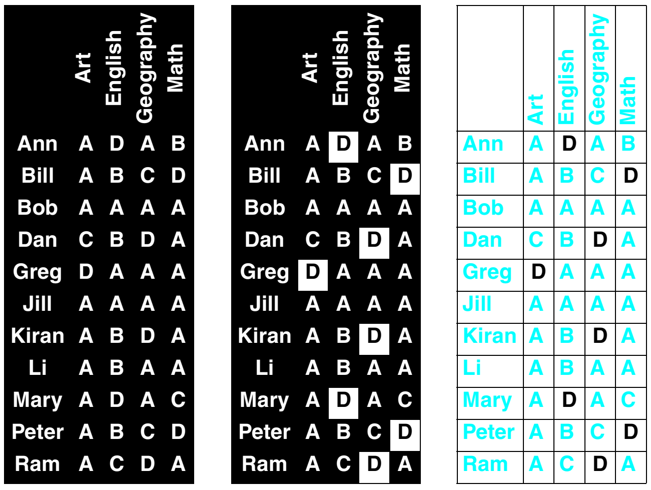
General strategies	Word processing examples	Spreadsheet examples	CAD examples
Iteration			
1. Reuse and modify group of objects	Copy and modify an existing paragraph to create a new one	Copy and modify an existing table and formulas to create a new one	Copy and modify an existing graphic arrangement to create a new one
2. Check original before making copies	Check if paragraph is correct and complete before making many copies	Check if column headings are correct and complete before copying to create new table headings	Check if window in building facade is correct and complete before making many copies
3. Handle exceptions before modification of groups	Group paragraph, drop a sentence, then modify group	Group all information, drop table headings, then modify group	Group graphic elements, drop an element, then modify group
Propagation			
4. Make dependencies known to the computer	Make paragraphs dependent on a format definition	Make formulas dependent on numbers in cells	Make window in building facade dependent on a graphic definition
5. 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			
6. 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
7. Generate new representations from existing ones	Generate table from tabbed words	Generate bar graph from table	Create 3-D model from 2-D floor plan
Visualization			
8. View relevant information, do not view irrelevant information	Magnify document to read small print	View formulas, not results	Do not display patterned elements
9. 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
10. 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

group. This method is more efficient than creating smaller groups to avoid the exceptions. Figure 1 shows how this strategy is useful to modify a group of objects with exceptions in a spreadsheet task. Strategy 3 can be also be used to modify the format of a paragraph with exceptions (such as a sentence), or to modify the color of a group of graphic elements with the exception of one element. Cognitive analyses of real-world CAD usage have shown that users often missed opportunities to use the above aggregation strategies. Use of aggregation strategies could have saved them between 40% to 75% of the time they took to complete their drawing tasks [1]. Real-world users have explicitly stated the importance of saving time while performing repetitious tasks [3]. These aggregation strategies therefore provide efficiencies that are of value to users.

2.2 Propagation Strategies

The next two strategies in Table 1 (Strategies 4 and 5) exploit the power of computers to propagate modifications to objects which are connected through explicit dependencies. Strategy 4 makes the dependencies between objects "known" to the computer so that (1) new objects inherit properties or receive information from another object, and (2) modifications can propagate through the dependencies. For example, word processor users can make paragraphs which 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 5 exploits such dependencies to generate variations of the same information. For example, the strategy could be used to explore different looks of a document in a word processor, generate different results in a spreadsheet by altering a variable (such as an interest rate), or create several variations of window designs in a building facade while using a CAD system.



Select All

	Art	English	Geography	Math
Ann	A	D	A	B
Bill	A	B	C	D
Bob	A	A	A	A
Dan	C	B	D	A
Greg	D	A	A	A
Jill	A	A	A	A
Kiran	A	B	D	A
Li	A	B	A	A
Mary	A	D	A	C
Peter	A	B	C	D
Ram	A	C	D	A

Drop Exceptions

	Art	English	Geography	Math
Ann	A	**D**	A	B
Bill	A	B	C	**D**
Bob	A	A	A	A
Dan	C	B	**D**	A
Greg	**D**	A	A	A
Jill	A	A	A	A
Kiran	A	B	**D**	A
Li	A	B	A	A
Mary	A	**D**	A	C
Peter	A	B	C	**D**
Ram	A	C	**D**	A

Modify Selection

	Art	English	Geography	Math
Ann	A	**D**	A	B
Bill	A	B	C	**D**
Bob	A	A	A	A
Dan	C	B	**D**	A
Greg	**D**	A	A	A
Jill	A	A	A	A
Kiran	A	B	**D**	A
Li	A	B	A	A
Mary	A	**D**	A	C
Peter	A	B	C	**D**
Ram	A	C	**D**	A

Figure 1. How Strategy 3 is useful to perform a spreadsheet task involving the modification of a table containing exceptions (cells containing "D").

2.3 Organization Strategies

Strategies 6 and 7 exploit the power of computers to construct and maintain organizations of information. Strategy 6 reminds users to make the organization of information known to the computer in order to (1) enhance comprehension, and (2) enable quick modifications. For example, a table constructed with tabs in a word processor is not "known" to the computer as a table, and therefore the tabular structure may not be maintained when the table contents are modified. On the other hand, a table which 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 7 generates new representations from existing ones. For example, tabbed tables in word processors can be mapped to tables and vice versa, numerical data in a spreadsheet can be represented as charts, and 3-D graphic objects can be generated from 2-D representations and vice-versa.

2.4 Visualization Strategies

The last three strategies in Table 1 (Strategies 8-10) exploit the power of computers to view selectively. Strategy 8 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 small to read while using a word processor, this strategy could be used to magnify the view instead of changing the font size. Similarly, in a CAD system, patterned elements can be undisplayed when not needed in order to make the relevant information more salient.

Strategy 9 addresses the limited screen space of most computers. Often, users have tasks which require them to compare or manipulate objects which are difficult to view simultaneously in a single view. For example, a user might need to compare the contents of a table at the beginning of a long word processing document, to the contents of a table in the middle of the same document. In such cases, instead of moving back and forth between the tables, it is more efficient to set up views focused on each table enabling both tables 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. (See [11] for a discussion on how this strategy affects performance).

Strategy 10 extends the notion of selective viewing to tasks involving a combination of navigation and manipulation. For example, a CAD user might need to make many precise changes to different parts of a large floor plan. A magnified view is needed to make precise changes, while a global view is needed for navigation to the next task. One way (as performed by a user in our ethnographic study [4]) is to zoom in to perform the precise modifications, and then to zoom out of the same view to navigate to the next task. A more efficient method is to have one global view of the file for navigation, and one local view to make the changes. The user then selects the location of interest in the global view which automatically updates the local magnified view where the user can make the precise modifications. As shown in Table 1, this strategy is useful when modifying a large word processing document as well as a large spreadsheet.

Based on the above description, the set of ten strategies appears to be useful across the three application domains especially when dealing with complex tasks involving large documents. It is pertinent to repeat that none of these strategies is actually necessary to complete complex tasks (therefore making them non-obligatory as described earlier). For example, users can choose to operate on individual elements instead of on groups, scroll up and down large documents which cannot fit on the screen instead of bringing them together, and so on. On the other hand, users can benefit by using these general strategies to perform complex tasks. However, for users to actually execute these strategies, they require specific functionalities to be present in computer applications. What are these functionalities and are they provided in commonly used computer applications?

3. FUNCTIONALITIES CONDUCIVE TO THE USE OF EFFICIENT STRATEGIES

Because users can complete a wide range of tasks without using efficient strategies, applications need to provide only very basic functionality for users to get their tasks done. However, if users need to complete complex tasks efficiently, then applications must address the functionality needed to execute these strategies. In addition, as the following discussion will show, the mere presence of a functionality may not be sufficient to make it conducive to the use of efficient strategies.

3.1 Iteration Functionalities

The first column in Table 2 shows the ten strategies discussed earlier, and the second column shows the functionality required for their use. The strategies of iteration (Strategies 1-3) all require the ability to aggregate objects and to perform various operations (such as duplication and modifications) on those aggregates. In addition, Strategy 2 requires the user to check the original for errors, and for completion before replication; Strategy 3 requires the ability to add and drop objects from an aggregate in order to handle exceptions before operating on the aggregate.

However, just implementing functionalities such as add and drop may not be sufficient to make them *conducive* for strategy use. This is because functionalities interact with many other design issues which could affect strategy use. For instance, a user might attempt to strategically reuse and modify an entire building design (containing hundreds of graphic elements). If the system neither warns the user that a subsequent operation on the aggregate would exceed the undo buffer, nor dynamically extends the size of the undo buffer, the user could make a mistake and have to manually undo the operation. Such experiences may dissuade users from exploring and using efficient strategies despite the fact that they are operational.

In addition to the issue of resource management described above, other issues relevant to making aggregation conducive for strategy use include functionality complexity (Should aggregates be allowed to be nested and persistent, or flat and temporary? If they are persistent, how can they be stored efficiently and later disaggregated?), and level of consistency (Should aggregation functionalities be consistent across all application objects such as words, paragraphs, pages, and files?).

3.2 Propagation Functionalities

The next category of strategies in Table 2 deals with propagation. These strategies require the ability to link objects to other objects, or to an abstract definition, and functionality to maintain the dependencies during modification. Strategy 5, in addition, requires functionality to manage variations generated when exploiting the dependencies in what-if scenarios. Design issues which affect how conducive propagation functionalities are to strategy use include how to deal with circularities, propagation update time (dynamic vs. delayed), and the debugging of links between objects when propagation produces unintended results.

3.3 Organization Functionalities

The strategies of organization require functionalities that allow objects to be related in 2-D representations (such as lists, and trees), or 3-D representations (such as layers) and the maintenance of these representations during modification of data. Strategy 7, in addition, requires the ability to map between representations (such as from tabbed paragraphs to tables). Important design issues include whether the organization representations are predetermined, or whether they can be created by the user.

3.4 Visualization Functionalities

The strategies of visualization require different ways to display information without altering the content of the information. Strategy 8 requires ways to selectively display the quantity of information in addition to magnification. Strategy 9 requires display mechanisms to bring together parts of a document which cannot ordinarily fit simultaneously on a screen. Finally, Strategy 10 requires functionality which allows global and local displays of the same information and which allow users to select areas from a global view to work in the local view. Important design issues include refresh capabilities, and window layout and management.

The above functionalities have been described at a fairly abstract level and can be instantiated in many ways. Do popular commercial applications provide these functionalities, and if so, how are they instantiated?

4. FUNCTIONALITIES INSTANTIATED IN POPULAR COMPUTER APPLICATIONS

Table 1 showed examples of how each of the ten strategies could be used to perform meaningful tasks in three application domains. Columns 3-8 in Table 2 show how the same examples could be executed in six applications, if required functionalities are provided. The examples are broken down into detail steps each of which corresponds to an abstract functionality in column 2. These examples are limited to those required for interacting with basic objects in the applications (words and paragraphs in word processors, cells and formulas in spreadsheets, and primitive graphic elements like arcs and lines in CAD systems).

The grayed cells in Table 2 represent steps which cannot be executed in an application, and therefore represent missing functionalities in those applications. As shown, there were missing functionalities for each of the strategy groups across the applications, and each application had at least one missing functionality. To illustrate how missing functionalities affect tasks,

Table 2. Ten general strategies, the abstract functionality they require, and examples of how the abstract functionality is instantiated in six popular applications. The gray cells represents missing functionality required to execute a strategy.

General strategies	Required functionality	Microsoft® Word® (2000)	Adobe® FrameMaker® (v. 5.5)	Microsoft® Excel® (2000)	StarOffice™ (v. 5.1)	MicroStation/J™ (v. 7.01)	AutoCAD® (v. 13)
Iteration							
1. Reuse and modify group of objects	Aggregate objects	Select paragraph	Select paragraph	Select table	Select table	Select many shapes	Select many shapes
	Duplicate aggregate	Copy paragraph	Copy paragraph	Copy table	Copy table	Copy group	Copy group
	Modify aggregate	Modify sentence	Modify sentence	Modify cell content	Modify cell content	Modify radius	Modify radius
2. Check original before making copies	Check aggregate	Spell check parag.	Spell check parag.	Spell check col hd.	Spell check col hd.	Check precision	Check precision
	Aggregate objects	Select paragraph	Select paragraph	Select cells	Select cells	Select shapes	Select shapes
	Duplicate aggregate	Copy paragraph	Copy paragraph	Copy cells	Copy cells	Copy group	Copy group
3. Handle exceptions before modification of groups	Aggregate objects	Select paragraph	Select paragraph	Select table	Select table	Select shapes	Select shapes
	Drop from an aggregate	Unselect word	Unselect word	Unselect cell	Unselect cell	Unselect a shape	Unselect a shape
	Add to an aggregate	Select word	Select word	Select cell	Select cell	Select a shape	Select a shape
	Modify aggregate	Modify font	Modify font	Modify font	Modify font	Modify color	Modify color
Propagation							
4. Make dependencies known to the computer	Link dependent objects	Apply style to parag.	Apply style to parag.	Link formula to cell	Link formula to cell	Instantiate shared cell	Instantiate block
	Maintain consistency over dependents	Redefine style	Redefine style	Data changes update formula	Data changes update formula	Changes in cell updates all cells	Changes in block updates all blocks
5. Exploit dependencies to generate variations	Maintain consistency over dependents	Redefine style	Redefine style	Changes in data update formula	Changes in data update formula	Changes in cell updates instances	Changes in block updates instances
	Manage variations	Variations stored in scenarios	Variations stored in scenarios	Variations stored in scenarios	Variations stored in scenarios	Variations stored in scenarios	Variations stored in scenarios
Organization							
6. Make organization known to the computer	Organize objects through representations	Create table	Create table	Place yearly tables on different sheets	Place yearly tables on different sheets	Organize col. & walls on different levels	Organize col. & walls on different levels
	Maintain org. during modifications	Add row to table	Add row to table	Add sheet	Add sheet	Add new levels	Add new layers
7. Generate new representations from existing ones	Map between representations	Convert tabbed paragraph to table	Convert tabbed paragraph to table	Generate charts from numbers	Generate charts from numbers	Generate 3-D building from 2-D plan	Generate 3-D building from 2-D plan
Visualization							
8. View relevant information, do not view irrelevant information	Selective display of detail	Display para. marks	Display para. marks	Display formulas	Display formulas	Display pattern elem.	Display pattern elem.
	Selective display of magnification	Zoom in	Zoom in	Zoom in	Zoom in	Zoom in	Zoom in
9. View parts of distant information to fit simultaneously on the screen	Simultaneous viewing of distant information	Split window and scroll to appropriate locations in each pane	Split window and scroll to appropriate locations in each pane	Split window and scroll to appropriate locations in each pane	Split window and scroll to appropriate locations in each pane	Open two windows, scroll to appropriate locations in each window	Open two windows, scroll to appropriate locations in each window
10. Navigate and select in global view, manipulate in local view	Global and local views	Open outline windows with normal view	Open outline windows with normal view	Open outline windows with normal view	Open outline windows with normal view	Open two windows, zoom-out in window1 zoom-in in window2	Open two windows, zoom-out in window1 zoom-in in window2
	Global-local view correspondence	Click on outline view, type in normal view	Click on outline view, type in normal view	Click on outline view, type in normal view	Click on outline view, type in normal view	Click on location in global window, operate in local	Click on location in global window, operate in local

three instances will be discussed: The missing functionality for viewing information simultaneously on the screen in Adobe® FrameMaker® (required for Strategy 9), the missing functionality for dropping elements from an aggregate in Microsoft® Excel® (required for Strategy 3), and the missing functionality of managing variations in CAD systems (required for Strategy 6).

4.1 Missing Functionality: Viewing Distant Information

Figure 2 shows a word processing task where many elements of an ordered list (the source) located at the end of a long document must be moved to an ordered list (the target) located in the middle of the file. Depending on the functionality provided in an application, there are at least four ways to perform this manipulation task. The first way is to select an element in the source list, cut it, scroll to the target list and paste it in; then repeat the operations for each element to be moved. The second way is to copy the entire source list to a location close to the target location, and then delete the elements not needed. This avoids the extra back and forth scrolling but still requires going back to the original list and removing the elements. A third way is to open a copy of the same file in another window, scroll to the appropriate source and target locations in each window and then cut and paste across the windows. A fourth way is to split the current window into two panes, and then perform the cut and paste across the panes similar to the method across windows described above.

While the ability to see different parts of documents using duplicate windows and split windows has been implemented in applications such as Microsoft® Word® and Microsoft® Excel®, Adobe® FrameMaker® does not support these features. Adobe® FrameMaker® users are therefore constrained to use the first or second method described above. These methods are time consuming and error prone. The inability to view different parts of a document simultaneously is not limited to manipulation tasks. It also affects tasks where information has to be compared, such as the task of comparing numbers in tables which are far apart in a document.

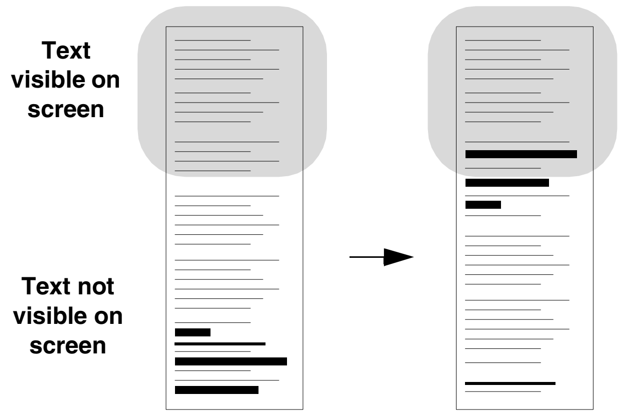


Figure 2. A word processing task requiring the movement of information (shown as bolded lines) across distant parts of a document.

4.2 Missing Functionality: Dropping from an Aggregate

Figure 1 showed a spreadsheet task of modifying all the cells in a large spreadsheet to be gray, and left aligned with the exception of cells containing “D” (which have their own complex formatting). A quick way to perform this task (using Strategy 3) is to select the entire document with a single selection, unselect the eight cells containing “D”, and then apply the two modifications. This requires nine selections, and two modifications.

However, as shown in Table 2, Microsoft® Excel® does not allow cells to be unselected from an aggregate. In the absence of this functionality, the user must select the entire document as before, apply the two modifications, then select the eight cells and again apply modifications to restore the exceptions to their original state. This requires nine selections, two modifications for the entire document, and the additional modifications to restore the cells to their original state. Besides the extra steps of applying the restoring modifications, the restoration process itself can also be error prone as the user may forget the original state of the exceptions.

It is not clear why despite countless design revisions, Microsoft® Excel® does not support dropping[1] an element from an aggregate, even though it supports adding to an aggregate. As StarOffice™ (a spreadsheet application[2] with equivalent capabilities) supports the add *and* drop functionality, it is unlikely that the absence of the drop feature in Microsoft® Excel® is due to a basic design constraint.

4.3 Missing Functionality: Managing Variations

As discussed earlier, Strategy 5 used dependencies to quickly generate variations in a document. In a CAD system, this approach can be used to quickly generate variations for design reviews by a client. For example, a high-rise building facade may contain 200 arched windows which are all dependent on a common definition. When the definition of the arched window is changed to a rectangular window, the change is propagated to all 200 windows. The floor plan now has 200 rectangular windows. As each of such variations has different repercussions on other parts of the design such as the building entrance (which may now also require a change from arched to rectangular), it is important that the variations be available for quick inspection before deciding on any one variation. One way is to save the variations in different files. However, this approach is expensive as floor plans tend to be huge. Another way is to store only the key parameters (e.g. height, width, and shape) upon which the variations depend. This enables the entire variation to be generated when needed without having to store the entire file.

As shown in Table 2, this functionality is not available in MicroStation/J™ and AutoCAD® even though both the systems are extensively used to design complex buildings in many design

1. When a user shift-clicks on a cell which is already included in a selection, it turns white and appears like it has been dropped, but this is not the case.
2. Available from: http://www.sun.com/staroffice

firms. This feature has been implemented in Microsoft® Excel® where different variations of the calculations can be stored and regenerated by using scenarios.

5. IMPLICATIONS FOR DESIGN AND TRAINING

The above analysis revealed missing functionality pertinent to the use of efficient strategies in six applications. While such information could be useful to designers of the respective products, the question arises:

Can this analysis of functionality be useful to designers of new products beyond the scope of the applications analyzed?

The answer to that question lies in the nature of tasks addressed by the strategies. For example, consider iterative tasks. Such tasks are so general in nature that they have been addressed long before computers became useful (e.g. printing presses, dish washers, and photocopying machines) and will continue to be addressed in any computer application which deals with operations on many objects. These include the manipulation of files in an operating system, the manipulation of words and paragraphs in an e-mailing system, the creation of files in a web authoring application, or the creation of slides in a presentation application. The generality of the analysis lies in the basic insight 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. This ability shifts the task of iterating over each object from the user to the tool. As this method is relevant irrespective of the nature of the objects being operated upon (dishes, paragraphs, shapes, or files), the analysis is useful regardless of the application as long as its users have the task of operating on many objects.

The same generality holds true for the ability to link dependent objects, organize objects, or view objects selectively. Each of these categories of tasks (and therefore strategies) is necessary in a wide range of applications currently existing or yet to be developed. These general categories of tasks and strategies will undoubtedly grow as developers continue to push the envelope of computer applications, and as users begin to explore complex tasks made possible with the new tools.

While this systematic inclusion of functionality may be useful, it raises the troubling issue of users having to deal with an explosion of features that they seldom understand or use. This leads to the next question:

Does the approach suggested in this paper only exacerbate the problem of featurism already rampant in computer applications?

One of the most troubling aspects of featurism appears to be its random nature. New features flood application upgrades for reasons often not clear to users. While the approach discussed in this paper does focus on checking for the presence of features, it provides a principled approach to add such features based on families of functionality. Based on such an analysis, a designer can make the argument that functionalities which directly support efficient strategies are more important than others to include in an application release. Furthermore, if families of features are consistently available in different applications, the features will no longer be random as users could expect them to be there whenever they need to perform classes of tasks. The notion of consistency across applications leads to a third question:

Can strategy-conducive designs be leveraged to enable the transfer of strategic knowledge across applications?

Research on the transfer of cognitive skill [13] suggests that systems which are consistent could enable the transfer of methods and therefore reduce learning time. This presents the opportunity for teaching general strategies and leveraging their generality to reduce the learning time of new applications. We have been experimenting with this approach in undergraduate and graduate classes at Carnegie Mellon University [2, 8]. The hope of such training is that users will not only learn to use current systems efficiently, but will also transfer them to new applications for which they do not receive training.

6. CONCLUSIONS AND FUTURE RESEARCH

This paper has led to four conclusions. (1) There is a set of strategies which appears generally useful across computer applications. (2) These strategies not only require specific functionalities for their execution, but also require the functionalities to be carefully designed before they can be conducive for strategy use. (3) A systematic analysis based on these strategies can reveal the absence of functionalities even in major applications. (4) This approach could guide designers to develop new systems which are conducive to the use of efficient strategies, and lead to systems which allow the transfer of knowledge across applications.

Based on these conclusions, we are currently exploring the development of a method which critiques whether a design is Conducive to the Use of Efficient Strategies (Design-CUES). This method would be based on the set of ten general strategies discussed in this paper, in addition to others we may discover through future research. For each of the strategies, the critique would include questions like: Does the design provide tools necessary to execute the strategy? If it is not provided, is there an alternate way to perform the task efficiently?

Because efficient strategies are often not discovered, or opportunities to use them not recognized [1, 3, 9, 12], we are experimenting how such knowledge can be incorporated into training [2]. These experiences have led us to explore the development of a method which critiques whether a training design teaches knowledge which is Conducive to the Use of Efficient Strategies (Training-CUES). The critique would include questions like: Does the training include exercises which teach users to recognize opportunities to use the strategies? Does the training include exercises which make users execute the strategies in meaningful contexts?

The goal of the Design-CUES, and Training-CUES analysis methods is to operationalize our experience in identifying, designing for, and teaching efficient strategies. The common framework of efficient strategies for these methods provides the opportunity for designers and trainers to collaborate with the ultimate goal of making users more efficient in the use of complex computer applications.

7. ACKNOWLEDGMENTS

This research was supported by the National Science Foundation, Award# IRI-9457628 and EIA-9812607. The views and conclusions contained in this document are those of the author, and should not be interpreted as representing the official policies, either expressed or implied, of NSF or the U. S. Government. The author thanks B. John, F. Reif, R. Sun, and G. Vallabha for their contributions.

8. REFERENCES

[1] Bhavnani, S. K. *How Architects Draw with Computers: A Cognitive Analysis of Real-World CAD Interactions*, unpublished Ph.D. dissertation, Carnegie Mellon University, Pittsburgh (1998).

[2] Bhavnani, S.K. “Strategic Approach to Computer Literacy,” *Proceedings of CHI’00,* 161-162 (2000).

[3] 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, 243-255 (1996).

[4] Bhavnani, S.K., and John, B.E. “Delegation and Circumvention: Two Faces of Efficiency,” *Proceedings of CHI’98*, 273-280 (1998).

[5] Bhavnani, S.K., and John, B.E. “Exploring the Unrealized Potential of Computer-Aided Drafting,” *Proceedings of CHI’96*, 332-339 (1996).

[6] Bhavnani, S.K., and John, B.E. “From Sufficient to Efficient Usage: An Analysis of Strategic Knowledge,” *Proceedings of CHI’97*, 91-98 (1997).

[7] Bhavnani, S.K., and John, B.E. “The Strategic Use of Complex Computer Systems,” *Human-Computer Interaction*, (in press).

[8] Bhavnani, S.K., John, B.E., and Flemming, U. “The Strategic Use of CAD: An Empirically Inspired, Theory-Based Course,” *Proceedings of CHI’99,* 42-49 (1999).

[9] Cragg, P.B., and King, M. “Spreadsheet Modeling Abuse: An Opportunity for OR?,” *Journal of the Operational Research Society* 44, 743-752 (1993).

[10] Doane, S.M., Pellegrino, J.W., and Klatzky, R.L. “Expertise in a Computer Operating System: Conceptualization and Performance,” *Human-Computer Interaction* 5, 267-304 (1990).

[11] Lee, W.O., and Barnard, P.J. “Precipitating Change in System Usage by Function Revelation and Problem Reformulation,” *Proceedings of HCI ‘93*, 35-47 (1993).

[12] Nilsen, E., Jong H., Olson J., Biolsi, I., and Mutter, S. “The Growth of Software Skill: A Longitudinal Look at Learning and Performance,” *Proceedings of INTERCHI'93*, 149-156, (1993).

[13] Singley, M.K., and Anderson, J. R. *The Transfer of Cognitive Skill*. Harvard University Press, Cambridge, Massachusetts (1989).