

Patterns of Change in Design Metaphor

A Case Study

SAND98-0129C

SAND-98-0129C

CONF-980414--

RECEIVED

JAN 29 1998

OSI

ABSTRACT

Design metaphors play an important role in the development of many software projects. However, the influence of metaphors on project functionality, design methodology and the interactions among members of the development team is not well understood. This paper seeks insights into these issues by examining the experiences of a design team in building a system under the influence of a particularly strong design metaphor.

Keywords

Metaphor, software design, user-oriented design.

INTRODUCTION

Metaphor has long been recognized to play an important role in user interface design, with developers systematically exploiting *desktop metaphors*, *agent metaphors* and similar figures to enhance the usability of computer interfaces. Recently, there has been an increased recognition of metaphor's larger role in the design process [1]. This paper examines the effect of a strong design metaphor on the development of an interactive computer system.

The *Design for Machinability Advisor* [DFM] was built to assist mechanical engineers in improving the manufacturability of machined parts. From the beginning, the project was conceived to be a "spelling checker" for machinability problems. This paper examines the influence of the "spelling checker" metaphor on DFM's design, as well as the changes that occurred in our understanding of the metaphor itself as the design matured. It also considers the metaphor's social context, particularly differences in the way team members interpreted it, and the effects of these differences on their interactions.

THE USE OF METAPHOR IN DESIGN

The use of metaphor in design reflects its function in such areas as the development of language [2-4] and scientific discovery [5-7]. Metaphors are figures of the form: "A is B", where B is said to be the *source* of the metaphor, and A is the *target*. Interpretation of a metaphor is a process of discovering which properties of the source may be valid and useful to understanding the target. For example, cognitive science's use of an *information processing* metaphor can be regarded as a process of finding which properties of computers can effectively be used to explain human thought. The metaphor provides a restricted set of hypotheses about human cognition. By focusing our attention on the significant properties of relatively well understood concepts (the source), metaphors impose an

William A. Stubblefield
Sandia National Laboratories
P. O. Box 5800
Albuquerque, New Mexico 87185
505-284-2856
wastubb@sandia.gov

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.

essential structure on the enormous problem spaces found in such areas as scientific discovery and the development of language. This focusing ability also underlies metaphor's contributions to design.

Hesse [5] has examined the use of models and analogies in science, providing a foundation for understanding the nature of design metaphors. She describes three components of analogies: the *positive* analogy consists of those properties of the source that are known to apply to the target, the *negative* analogy includes properties of the source that either untrue or irrelevant for the target, and the *neutral* analogy contains those properties that have not yet been classified as positive or negative. Scientific discovery is a process of evaluating these yet unclassified components of the neutral analogy. A similar analysis can be applied to the development of linguistic and design metaphors.

Design and the interaction theory of metaphor

There are several aspects of this initial model that must be refined if we are to understand the role of metaphor in design. Aspects of a design metaphor can seldom be classified as simply positive or negative: in transferring properties from the source to the target, the designer generally will re-interpret them. For example, direct manipulation interfaces are not so much an instantiation of the ways we manipulate objects in space, as they are a rethinking of those operations. Instead of grasping an object, we select it. Placing one object on top of another (such as placing a document on a printer icon) is not a simple act of stacking, but often invokes some computer operation (such as printing).

Similarly, properties that might initially seem to be part of the negative metaphor are often modified during design to maintain the metaphor's consistency. An example of this is in the use of a "trash can" icon to invoke file deletion. Unlike real trash cans, trash can icons never become full. This negative aspect of the metaphor can cause problems: users who only empty their physical trash cans when they overflow may neglect to empty their computer trash, eventually causing disk storage to fill up with deleted files. Some interfaces (e.g. the Macintosh) ignore this problem, while others attempt to repair the metaphor through such techniques as prompting the user to empty the trash when logging off of the system.

Black's *interaction theory of metaphor* [2] provides a framework for understanding these processes of adapting the positive and repairing the negative components of a

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED



MASTER

19980406 104

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

design metaphor. Black argues that metaphor is not simply a process of transferring properties from the source to the target, but a complex interaction between them in which our knowledge of the target is equally capable of changing our understanding of the source. Metaphor induces complex shifts of meaning across both the target and the source. To paraphrase Black¹, if implementing file deletion with a trash can icon improves usability, we must not forget that this will also change our understanding of "real" trash cans. As an informal example, colleagues have told me that they depend on the ability to retrieve things from their computer trash, treating it as an intermediate stage between long term storage and deletion. This has made them wish for more control over the schedule for emptying the physical trash cans in their offices.

The interaction theory's view of metaphor as inducing shifts of meaning in both the target and source is particularly important to understanding design metaphors. During DFM's design, we encountered many situations that required modification of the "spelling checker" metaphor. One of the most notable resulted from the complexity involved in detecting machinability problems. Unlike a word processor's spelling checker, which simply looks for words that fail to appear in its dictionary, determining the machinability of a feature in a metal part requires an understanding of the designer's intention, as well as the feature's interactions with other features. Consequently, DFM required much richer interactions with the user than conventional spelling checkers. As will be discussed, this led to significant changes in our understanding of what it means to be a spelling checker.

Do metaphors ever die?

A commonly accepted view of the use of metaphor in language or discovery holds that, as a metaphor develops and becomes better understood, its interpretation becomes conventionalized, and it loses its ability to convey new ideas about the target. According to some views, the metaphor eventually becomes so conventionalized that it loses whatever suggestive power it had as a metaphor and effectively "dies." Linguistic idioms are common examples of *dead metaphors*. For example, "clawing his way to the top" is no longer seen as a metaphor for corporate success, but is simply a common, if not trite, idiom of everyday language. Similarly, as cognitive science has matured, the information processing metaphor has been conventionalized into the accepted problems of representation, search and cognitive architecture.

In general, design metaphors clearly follow this life cycle, shifting from the broadly suggestive to the more conventionalized as they become reified in an artifact. For example, as DFM neared completion, at least certain aspects of our interpretation of the "spelling checker"

¹ "If to call a man a wolf is to put him in a special light, we must not forget that the metaphor makes the wolf seem more human than he otherwise would." [2] (page 44)

metaphor became effectively fixed in the design.

However, the details of this process are far from straightforward. Rather than a process of steadily refining the metaphor, design seems to alternate between periods of gradual refinement of a stable design, and periods of radical shifts in the design and the underlying interpretation of the design metaphor. As will be described below, the Design for Machinability advisor went through at least three such radical interpretive shifts.

Many theorists [3] argue that metaphors never completely die, but retain both the structural properties of metaphor and at least some potential for revealing new meanings. This is also true of design metaphors, in spite of the reifying effect of the design artifact. As the development of DFM reveals, the eventual success of the project's initial stages further stimulated the "spelling checker" metaphor, and led to its expanded use in both talking about DFM and in proposing future projects.

The social lives of metaphors

An important function of linguistic metaphors is in defining the structure and boundaries of social groups, with societies or segments of a society being defined by shared metaphors. Teenagers seek new figures of speech that adults will not share, just as hipsters have always used dense and rapidly changing metaphors to lock out "straight" society. Cognitive science's use of an information processing metaphor virtually defines the field, and distinguishes it from other psychological and biological approaches to understanding thought.

As the development of the DFM advisor showed, this social function is one of the most important aspects of design metaphors. The "spelling checker" metaphor provided a common frame of reference that enhanced communication among members of the design team. The importance of this shared frame of reference was particularly important in enabling us to rapidly outline an initial system design and development plan. This common focus remained important as the project grew in complexity and different team members experienced potentially conflicting pressures from users, customers, technology, schedules, budget and organizational demands. The shared metaphor helped us to maintain a common understanding of the project and made it easier for us to negotiate compromises as the pressures of these competing demands both diverged and increased.

These benefits were not without price. Because the "spelling checker" metaphor enabled such rapid early progress, we made many decisions before we had a complete understanding of the user's needs and assumptions. As this paper will discuss, these early commitments and the power of the metaphor made it difficult to modify the design to correct these problems.

PATTERNS OF CHANGE

The remainder of this paper uses the development of the Design for Machinability Advisor as a case study in the evolution of a strong central metaphor across the design process. It considers both the influence of the metaphor on

the design, and the ways in which the design process changed the team members' understanding of the metaphor in turn. The paper examines the following conjectures:

- Although design metaphors do tend to move from being richly suggestive to more fixed as the design matures, the patterns of change are far from linear, but involve radical shifts in the interpretation and use of the metaphor. Periods of relatively stable interpretation alternate with more dynamic periods where the meaning of the metaphor once again becomes highly plastic and suggestive of multiple design options. Shifts in the metaphor are often reflected in radical changes in the design. Finally, the metaphor never really "dies" but retains its metaphoric foundations and suggestive power even after the system has been completed.
- Strong design metaphors can be both a benefit and an obstacle. Where the metaphor is appropriate, it can quickly lead to a powerful solution; where it is not, it may make it harder to solve design problems.
- Design metaphors do not work only at a semantic level, i.e. in terms of system functionality and user interface. The metaphor also influences system architecture, complexity and algorithm design. These, in turn, can change the interpretation of the metaphor.
- A central function of design metaphors is social, supporting communication and cooperation among members of the design team. This process is far from simple. Different team members arrive at different interpretations of the metaphor, with often surprising results.

THE DESIGN FOR MACHINABILITY ADVISOR

The Design for Machinability Advisor was intended to help mechanical designers improve the manufacturability of machined metal parts. Typical problems that add to the difficulty of machining a part include unnecessarily tight tolerance requirements, features that require specialized tools in order to be machined, failure to standardize features across different parts, and use of hard to machine materials. Traditionally, mechanical designers have focused primarily on the functionality of parts, leaving manufacturability concerns until later in the design process. This delay in addressing manufacturability increases the cost and difficulty of changing a design. DFM's goal is to help mechanical designers consider manufacturability early in the process, when designs are still flexible and easy to change.

When launching the project, our customer (a mechanical designer) proposed the metaphor of a "spelling checker" for mechanical designs. Consequently, the initial specification called for a system that used a feature recognizer to find machined features (holes, slots, pockets, etc.) in a solid model of a part. Once found, these features were to be passed on to a knowledge-based system that evaluated their machinability.

The design team was composed of:

- A feature recognition team consisting of two software engineers who were to refine and adapt existing feature recognition software to the needs of the DFM

advisor. One member of this team was also experienced in artificial intelligence, and was extensively involved with the knowledge engineering team.

- A knowledge engineering team consisting of a knowledge engineer (myself) and an experienced manufacturing engineer who was initially intended to serve as a domain expert, but who also contributed significantly and actively to design decisions.
- A project management team consisting of two people, one who took primary responsibility for planning, budget and organizational concerns. This person also participated in the early phases of design, and helped the design team remain synchronized with more global requirements, such as organizational standards for network interactions, and the need for our system to eventually interact with other software also under development. The other member of the design team was primarily concerned with funding and project development, and did not participate actively in design after the early stages.
- The customer for the system was also a mechanical engineer and functioned as our primary source of user input. We also sought out other engineers and machinists for their evaluation of prototype systems.

The organization of the design team was complicated by the fact that it was spread over four different organizations and buildings at Sandia Laboratories. The feature recognition team was at one site, the project manager and I were at a second, the domain expert at a third, and the customer and sample users at several other organizations. Although we met regularly (at least once or twice per week during the early phases of the project), and communicated by phone and e-mail as needed, this organizational scattering did complicate our interactions. One of the benefits of the unusually strong design metaphor was in providing a common focus, one that allowed us to work independently while maintaining a shared but flexible understanding of our common goals.

Because of the ambitious nature of the project, we made an early decision to use an iterative, prototype-based methodology that would allow us to explore the design space more freely. Selecting this exploratory approach in place of a more structured, top-down approach was clearly the right decision, as many of our early assumptions eventually proved mistaken. Three prototypes were implemented and evaluated in the process of refining the design.

It is important to note that a critical implication of the "spelling checker" metaphor, that the DFM advisor should be driven by feature recognition was effectively fixed very early in the project by our choice of the design team.

A final important aspect of the development milieu is Sandia Laboratories commitment to both research and the development of practical, immediately useful tools. As the project developed, these twin goals both supported the exploratory methodology we had selected, and also provided an additional set of design constraints that influenced our development of the machinability advisor.

PROTOTYPE 1: A PURE SPELLING CHECKER

The first prototype system we constructed was an attempt at a direct translation of the “spelling checker” metaphor. The feature recognizer acquired features from a solid model of a part, and sent them to the design advisor. This prototype was only concerned with holes.

The advisor applied a list of critics to each feature, and displayed those critics that detected potential problems. The advisor displayed these “fired” critics one at a time, just as a word processor’s spelling checker displays misspelled words. The “Next” button allowed the user to step through the applicable critics, while additional buttons allowed the user to skip a feature, redo a feature or start over with the first feature.

Figure 1 shows the main screen of this prototype. In order to let the user know which feature was being checked, the feature recognizer displayed the part with the current feature highlighted (not shown). The screen displays the feature in profile (upper right), surrounded by information about its dimensions and tolerances. The box below displays the text of critics (in the figure, it shows a “dummy” test critic). The buttons at the bottom allow the user to move through both features and the critics of a given feature. Note that this order is fixed in prototype 1.

The order of feature evaluation

Implementing such a direct translation of the “spelling checker” metaphor required that we repair several potential breakdowns of the metaphor. The first of these was lack of any obvious sequence in feature recognition. A conventional spelling checker scans a document from start to finish, highlighting potentially misspelled words as it checks. Although feature recognition exhibited no obvious sequence, we did highlight the current feature in the feature recognizer’s display. In initial discussions of the design with the domain expert and potential users, this solution was accepted; the users saw no difficulty in the essentially unpredictable order with which features were evaluated.

On reflection, the design team decided that there was no reason to be bound to an arbitrary order of evaluation and, in prototype 2, developed a “feature browser” that allowed the user to check features in any desired order. This browser displayed a list of features. Clicking on a feature caused it to be highlighted in the display of the part drawing, and sent to the advisor to be evaluated. The users appreciated this improved flexibility, although it is interesting to note that no one complained about the original, more rigid interaction. I believe this to result from the power of the original metaphor, and illustrates the way a strong design metaphor can lead a user to accept a less than optimal solution to their problem.

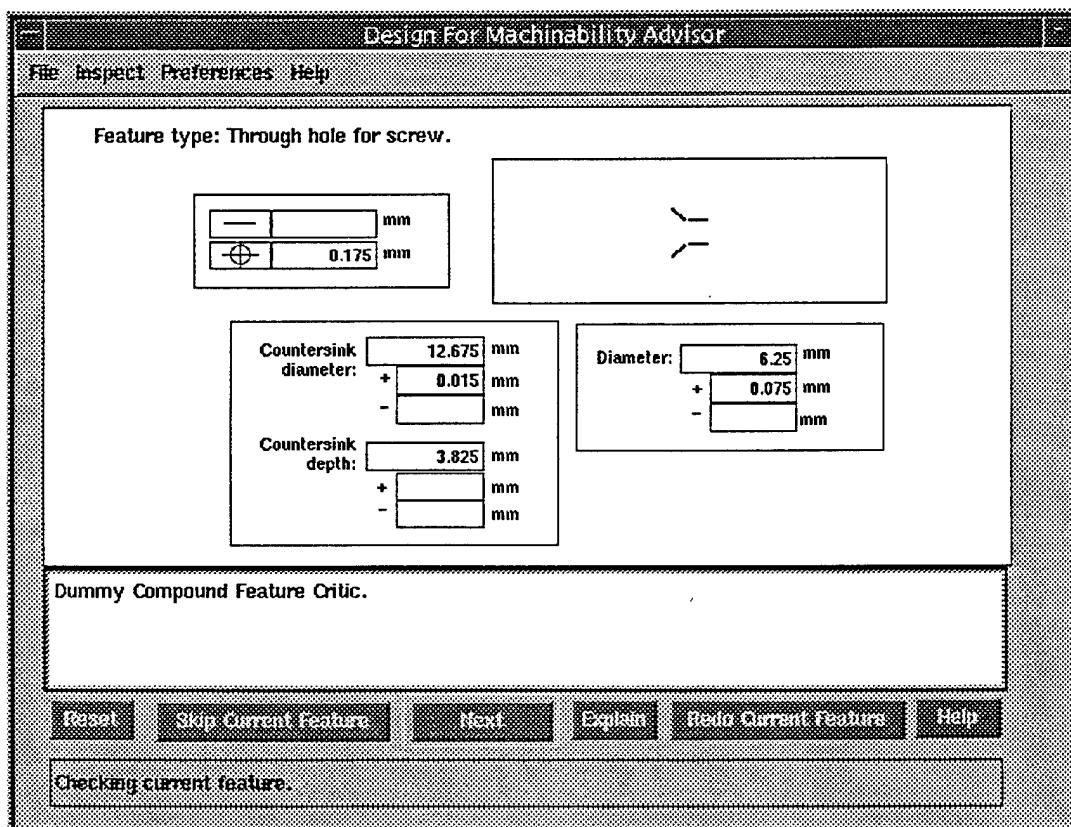


Figure 1

The problem of missing information

A more difficult problem was in the lack of complete information from the feature recognizer. Due to

limitations of the solid modeling software the feature recognizer was built upon, it was unable to recover information about tolerances from the original engineering

drawing. This was a severe limitation, since nearly all machinability problems involve tolerances at some level: if you don't care about tolerances, practically anything can be machined. Although a related project was exploring the possibility of adding this capability to the feature recognizer, it was not available to us as our project developed.

In order to minimize the data entry requirements on the user, we felt it important to provide useful default values for tolerances on feature dimensions. This proved difficult, since "reasonable" tolerances depend upon the intended use of the feature. Our solution to this problem was to construct a database of typical features that could be matched with new features in order to provide default tolerance values. For example, a countersunk hole for a standard screw could be easily recognized, providing the user with recommended tolerance values for that feature.

This solution never proved fully satisfactory. Although we were able to classify many simple features, such as holes, in a reasonable manner, it was clear that we would have difficulty in extending this approach to more complex, novel features. Also, by providing recommended tolerances rather than those the user had initially entered into the design drawing, we were introducing an unacceptable possibility for error into the interaction with the system. These problems were among the main reasons for changing the approach taken in this early prototype.

Metaphor breakdown due to complexity mismatch
A deeper problem was in the complexity of recognizing machinability problems. Spell checking a textual document is a relatively straightforward process of matching words in the document against those in a dictionary and indicting words that failed to match. Formally, this is a process of matching *regular expressions*, and algorithms exist for doing this efficiently. As we worked with our domain experts in acquiring knowledge of machinability problems it became clear that evaluating machinability problems was significantly more complex than finding a word in a spelling checker's dictionary. For example, determining whether a 0.002" diameter tolerance on a 0.25" hole is excessive or not depends entirely on the intended use of the hole. This, in turn, required asking extensive questions of the user, a further violation of the "spelling checker" metaphor. Although we did not formally characterize the complexity of evaluating features, it clearly cannot be done by matching regular expressions and is most likely context sensitive.

This impact of complexity problems on a design metaphor is both unexpected and significant. None of the literature I have encountered on the use of metaphor in discovery or design mentions the impact of complexity issues on metaphor interpretation. Generally, metaphors are assumed to fail if they make predictions that prove to be false. As we discovered, it is possible to implement the general patterns of a metaphor, only to see the metaphor break down because of differences in complexity between the source and target situations.

It is interesting to note that our customer had initially specified that we should not consider design intent in our evaluations, as this was too difficult. On subsequent conversations with the customer, our domain expert and other engineers and machinists, everyone acknowledged the importance of this type of information. Although I can only speculate, it seems reasonable that the "spelling checker" metaphor may have led the customer to add this limitation to our initial requirements in an effort to fit the problem to the metaphor.

Finding vs. preventing mistakes

The final difficulty encountered in prototype 1 was in its emphasis on error detection, rather than error prevention. As a "spelling checker" for designs, it was natural to apply the advisor to existing engineering drawings. It was only as we made progress in knowledge acquisition that we recognized that machinability knowledge could be more easily and more effectively applied if we offered it to the engineer as he or she was creating a design, rather than after the design already existed. It is interesting to note that although the maxim, "it is better to prevent errors than to detect them," is part of every designer's knowledge, the influence of the "spelling checker" metaphor led us to ignore this valuable rule of thumb until empirical evaluations of the first prototype led us to reconsider our assumptions. Similarly, although our customer clearly specified that the advisor would be used to check finished designs, on seeing the prototype, he asked it couldn't "detect errors as he was working."

PROTOTYPE 2: THE MIXED USE APPROACH

The second prototype addressed these difficulties we encountered in applying the "spelling checker" metaphor to the design of the machinability advisor. We did not, however, abandon the metaphor entirely. There were several reasons for this. Although the domain expert and I had doubts about the metaphor by this time, the feature recognition team and the customer still found it to be useful. Our discussions revealed a number of effective arguments for the benefits of a feature driven approach. Among these was the ability of the advisor to serve as a final check before manufacture, the learnability and usability gains provided by the metaphor, the ease of fitting a "spelling checker" into the engineering development process, and the technical benefits of providing the feature recognition team with a challenging test of their capabilities. Consequently, we decided to retain at least a feature-driven approach as at least part of the tool.

In order to address these limitations, we implemented a dual use advisor. When used as a checker for existing models, it took an approach that was similar to prototype 1. The only real difference was in replacing the one-at-a-time approach to checking features with a more flexible feature browser. This allowed the user to select features from the solid model and check them in any order.

In providing the second, non-feature driven interface, we made the database of typical features directly available to the user. They could browse this list, selecting, for example recommended configurations for common features

such as countersunk holes or holes for a rotating pin or shaft. The user could then edit these recommended configurations, changing either dimensions or tolerances. Where these changes violated any of the machinability checks, the advisor warned the user. Although not implemented in this prototype, we felt that completing this approach would require some way of "pasting" the resulting feature into the engineering drawing.

In evaluating this prototype, it was clear that we had moved in the right direction. Although everyone agreed on the merits of a dual use strategy, the team remained divided over the relative merits of the "front end design tool" vs. the "spelling checker" approach.

PROTOTYPE 3: TOOLS AND CRITICS

The final prototype built on its predecessor's dual-use approach, but made two notable additions to it. The first of these was in recognizing that certain types of knowledge could be useful for either front-end design or feature checking, but not both. We supported this by dividing our knowledge base into "tools" and "critics." Critics were used exclusively to evaluate existing features, whereas tools could be used for either evaluation of existing features, or to design features from scratch. A key difference between tools and critics was that tools involved more extensive user interaction, while critics simply

detected potential problems. The second addition to prototype 2 was in rewriting the system into a more flexible, modular architecture.

Evaluating solid models

When checking a model, the user browsed through features, and selected those to be checked. Each tool or critic included a condition test to determine its relevance to a given feature. When the user requested a feature be checked, DFM applied each test to the selected feature. The advisor then displayed a list of tools and critics whose conditions applied. Selecting an entry either displayed its text (for the critics) or initialized the tool with the feature's dimensions and launched it as a separate application.

An example of a tool was the PIN FIT ADVISOR (Figure 2) which advised the user on dimensions and tolerances of holes that were to fit a pin or moving shaft. Even when initialized with the diameter of a recognized hole, the pin fit advisor still asked the user a number of questions about the use of a hole (e.g. "is the hole intended to allow a shaft to rotate, or is it intended to fit tightly over a pin to accurately position a part?"), and recommended the tolerances for the pin and hole. Note from the screen in Figure 2 that this evaluation involved extensive

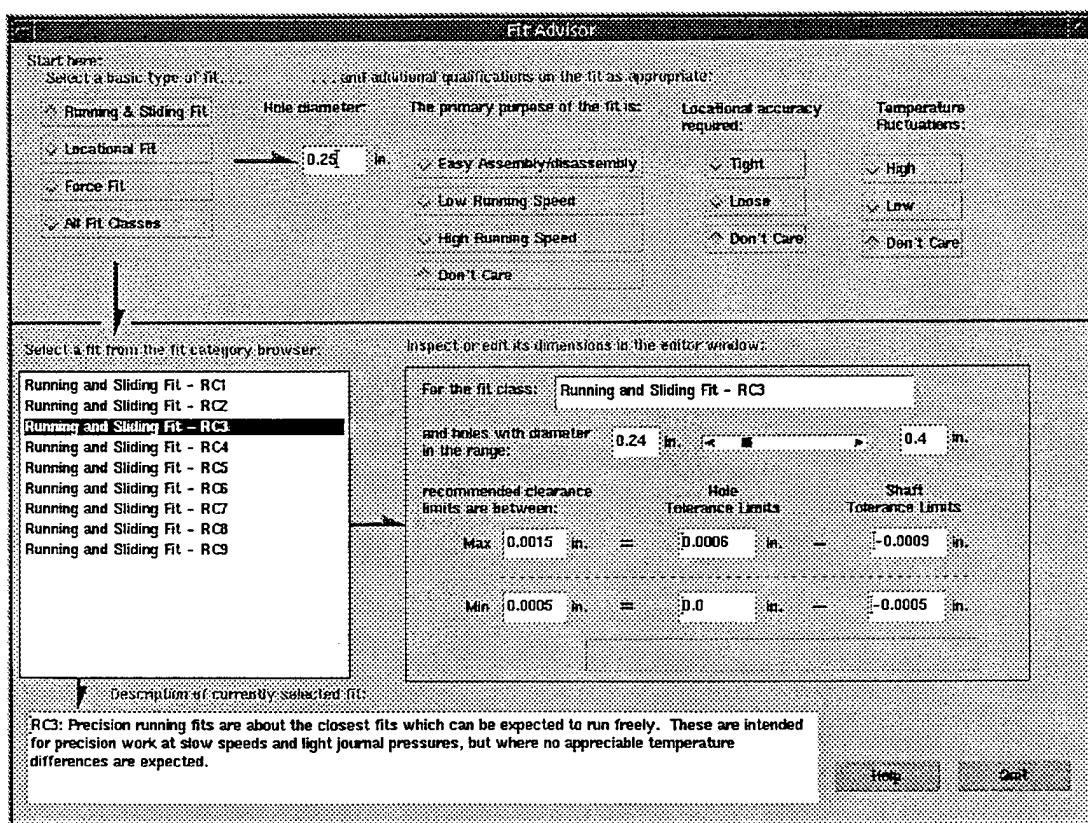


Figure 2

interaction with the user, and moves considerably beyond the "spelling checker" metaphor.

In contrast, a typical critic was one that detected holes

whose diameter failed to match any standard size drill bit, and called this problem to the user's attention. This critic required no user input, and could be used in a standard "spelling checker" mode. Note that, because of their

- simplicity, critics were able to fit the “spelling checker” metaphor more directly than tools like the pin fit advisor. In effect, prototype #3’s critic facility implemented an almost pure “spelling checker” for designs within the context of the more general tool.

An initial design toolkit

When a tool was used during initial design (i.e. without a solid model), the user selected tools from a browser. Critics were not available in this mode of use. Because the advisor did not use the feature recognizer to initialize tools with feature dimensions, the user entered the required dimensions and tolerance information directly.

Because the toolkit no longer depended upon the feature recognizer, we had more freedom in designing tools that were not feature oriented. One of these tools, the SURFACE FINISH EVALUATOR, determined which machining processes would be needed to achieve a specified surface finish, and evaluated the relative cost of these processes. This tool bore no direct relationship to the feature recognizer.

This toolkit approach represents a fairly drastic break with the “spelling checker” metaphor. The metaphor no longer was the dominant interpretation of the design, although it “re-appeared” when the advisor was applied to existing solid models.

Architectural implications

A significant change in the third prototype was architectural. The advisor was decomposed into much smaller, specialized objects than the earlier versions. For example, we recognized that the feature browser was a potentially valuable addition to the feature recognizer that could stand alone from the Design for Machinability advisor. Consequently, it was separated from DFM, and implemented as a pure feature browser that communicated with DFM through message passing. Similarly, each tool was an independent object that could be either invoked as an independent application, or first initialized with data from the feature recognizer. These were deliberately designed as independent objects to enable later migration to a distributed object environment, as is consistent with broader corporate software strategy.

This change in architecture was another unanticipated, and somewhat paradoxical, result of the maturation of the metaphor. Although it was written in an object-oriented style, Prototype 1 was essentially a “monolithic” program. Most objects were closely coupled and could not be realistically described as independent. As we came to understand the design (and the “spelling checker” metaphor) better, we were able to focus more clearly on architectural issues. This is somewhat paradoxical in that highly modular designs are conventionally regarded as a benefit to exploratory prototyping. However, in the early stages of development, our lack of understanding of the design and the dominance of the design metaphor made it hard to address these purely architectural concerns.

CONCLUSION

The development of the Design for Machinability Advisor was a valuable lesson in the use of design metaphors. In

summary, lessons learned from the project include:

Design metaphors have complex life cycles

The development of design metaphors closely follows the interaction theory [2], being characterized by shifts in the interpretation of both the design (the metaphoric target) and the source metaphor. As the DFM advisor developed, it extended our understanding of “spelling checkers” to include more complex interactions with the user and greater user control over the order with which items are checked.

Although design metaphors do tend to move from the broadly suggestive to a more static interpretation as the design becomes fixed, this is far from a steady progress, but alternates between periods where the interpretation of the metaphor remains relatively stable, and times of radical shift in the understanding of the metaphor. In the development of DFM, there were three such shifts:

1. Recognizing the implications of the feature recognizer’s lack of obvious order, and the rejection of a fixed order of analysis in favor of a more flexible feature browser.
2. The adoption of a dual use strategy that allowed the user to access system knowledge either to check features, or as an initial design tool.
3. Prototype 3’s division of machinability knowledge into distinct “tools” and “critics”, where only critics were required to be applicable to geometric features. This allowed us to create tools that had no direct link to the feature recognizer. Essentially, this step pushed the “spelling checker” metaphor back into part of the DFM Advisor (the feature recognizer/critic combination), both returning to a “purer” instantiation of the metaphor and allowing other parts of the tool to function in a less constrained manner.

In addition to exhibiting the shifts of meaning predicted by the interaction theory, the development of the “spelling checker” metaphor also supports Lakoff and Johnson’s [3] contention that metaphors never really die. Although the third prototype of the advisor both reduces the metaphor’s importance and fixes its interpretation in an artifact, both the customer and the project managers continue to use the metaphor for marketing DFM and planning future projects. The Design for Machinability Advisor has become one instance of a larger metaphor that will continue to be explored through future projects.

Strong design metaphors are a benefit and an obstacle

Because the “spelling checker” metaphor was so strong, it enabled us to quickly agree on a basic design and rapidly develop the first prototype. Prototype 1 was completed in about 4 months, which was fairly impressive given the fact that it involved a separate feature recognizer and knowledge based system interacting over a computer network. However, as the design matured, the metaphor also interfered with our ability to respond to an emerging understanding of the user’s needs and abilities. In particular, the “spelling checker” metaphor made it difficult to move into a fully dual-use approach.

Metaphors and complexity

Although my understanding of metaphors in language and science had prepared for the likelihood that certain implications of the "spelling checker" metaphor would prove wrong for the design advisor, I had assumed that these would be strictly semantic in nature; that is, I assumed that the metaphor would fail if it led to the wrong functionality or a difficult to understand interface. My experience with the Design for Machinability advisor demonstrated that a metaphor could also break down if the underlying computational complexity of the resulting design was drastically different from that of the source.

The social life of design metaphors

The most unexpected lesson learned was the importance of the design metaphor's social role. Because the design team was separated both geographically and organizationally, and because of the short development times required for a prototyping methodology, we often had to make design decisions without adequately consulting other members of the design team. This was particularly true of the feature recognizer and the knowledge based system parts of the tool. Although these often resulted in minor inconsistencies, none of these involved deeper semantic problems and all were easily repaired. I believe that this was due to the shared semantic understanding engendered by the shared metaphor.

Another surprising discovery was that different interpretations of the design metaphor coexisted quite harmoniously within the design team. As the project progressed, the interpretations of the metaphor seemed to grow along multiple lines: the machining expert and I came to reduce the importance of the "spelling checker" metaphor, while the customer and one of the management team retained a more direct interpretation. We were concerned that this might cause misunderstanding or rejection of the system. Surprisingly, they were quite happy to continue characterizing the advisor as a spelling checker, and accepted the final prototype as a faithful rendering of the metaphor. Ironically, the finalization of the design did not detract from the flexibility with which different team members interpreted the metaphor.

Similarly, the feature recognition team continued to regard the advisor as a "spelling checker" for designs, which is not surprising, since the use of the feature recognizer was closely tied to this metaphor. However, they were quick to accept such changes in the metaphor's interpretation as the shift to a feature browser driven approach. It is also interesting to mention that as the feature recognition team has started to explore other applications for their software, they have largely ignored the "spelling checker" metaphor. Although they continue to use it in discussing DFM, the metaphor plays little role in their continuing work. Because it serves no useful function, it is essentially dead for this portion of our team.

Design metaphor as a necessary fiction

As our experience with the Design for Machinability Advisor indicates, the use of a strong metaphor in design is far from a simple affair. It is characterized by radical shifts in interpretation of the metaphor and resulting

changes in the design. In spite of the metaphor's ability to bring the design team to a common focus in a rapid manner, it often hindered our understanding of user reactions and a flexible response to problems in the developing design.

On balance, I believe that these are not so much problems that need to be fixed as they are essential features of an inherently complex process. I do not believe that we could have started, let alone completed, this project without the contributions of this very strong central design metaphor. Although the system largely moved beyond it, the "spelling checker" metaphor was, and remains an essential component of our understanding of the design. I believe that the lessons to be learned are not that we should avoid strong metaphors, but that we should be prepared for the shifts in meaning that accompany their development, and their tendency to blind us to design possibilities that do not fit the metaphors obvious interpretations.

ACKNOWLEDGMENTS

I would like to thank Sandia National Laboratories for their generous support of the Design for Machinability Project. Ken Washington, Steve Kleban and Dwight Miller provided valuable comments on early ideas and drafts of this paper. I also thank the members of the Design for Machinability team: John Mitchiner, Kim Mahin, Jill Rivera, Lothar Bieg, Robert LaFarge, David Plummer and Marcus Craig. Their technical skill, design sophistication and friendship made this project unusually rewarding.

REFERENCES

1. Madsen, K.H., *A Guide to Metaphorical Design*. Communications of the ACM, 1994. 37(12).
2. Black, M., *Models and Metaphors*. 1962, Ithaca, NY: Cornell University Press.
3. Lakoff, G. and M. Johnson, *Metaphors We Live By*. 1980, Chicago: University of Chicago Press.
4. Gibbs, R.W.J., *The Poetics of Mind*. 1994, Cambridge: Cambridge University Press.
5. Hesse, M., *Models and Analogies in Science*. 1966, Notre Dame, Indiana: University of Notre Dame Press.
6. Gentner, D., *Flowing waters or teeming crowds: Mental models of electricity*, in *Mental Models*, D. Gentner and A.L. Stevens, Editor. 1983, Lawrence Elbaum Associates: Hillsdale, N. J.
7. Stubblefield, W.A., *Source Selection for Analogical Reasoning: An Interactionist Approach*. 1995. PhD Dissertation, Department of Computer Science, University of New Mexico.

M98003144



Report Number (14) SAND--98-0129C

CONF-980414--

Publ. Date (11) 199804

Sponsor Code (18) DOE/DP, XF

UC Category (19) UC-700, DOE/ER

DOE