Patterns of Change in Design Metaphor
A Case Study

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

ABSTRACT
Although design metaphors play an important role in many software projects, their influence on system functionality, project methodology and the interactions among members of the development team is not well understood. This paper seeks insights into these issues by examining the development of a computer program 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 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, including its influence on program functionality, system architecture and knowledge representation [1-5].

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 [6-8] and scientific discovery [9-11]. 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 [12]. 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 essential structure on the enormous problem spaces found in such target areas as scientific discovery, language development, or, as examined in this paper, design.

Hesse [9] has examined the use of analogies in science, providing a foundation for understanding 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 source properties that are either untrue or irrelevant for the target, and the *neutral* analogy contains those that have not yet been classified as positive or negative. Developing a metaphor or analogy requires evaluating the unclassified components of the neutral analogy.

Design and the Interaction Theory of Metaphor
This initial model 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, both designers and users 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 a document on a printer icon) is not an act of physical stacking, but of invoking some 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* [6] provides a framework for understanding these processes of adapting the positive and repairing the negative components of a 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 Black,[1] 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 their office trash.

The interaction theory's view of metaphor as inducing shifts of meaning in both target and source is 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 feature's intended use, as well as its interactions with other features. Consequently, DFM required much richer evaluation criteria and user interactions than conventional spelling checkers. This, in turn, broadened our understanding of spelling checkers in general.

The Social Function 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 often 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 "square" 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. This was particularly important in enabling us rapidly to outline an initial system design and development plan. It 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 increased.

These benefits were not without cost. 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. These early commitments and the power of the metaphor often made it difficult to correct these problems.

Do Metaphors Ever Die?

A commonly accepted view of the use of metaphor holds that, as a metaphor develops and becomes better understood, its interpretation becomes conventionalized, reducing 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 follow this life cycle, shifting from the broadly suggestive to the more conventionalized as they become reified in an artifact. As DFM neared completion, at least certain aspects of our interpretation of the "spelling checker" 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, development alternates between periods of gradual refinement of a stable design, and radical shifts in the design and the underlying interpretation of the metaphor. This reflects Kuhn's classic model of scientific discovery [13]. The DFM Advisor went through at least three such radical interpretive shifts.

Many theorists [7] 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.

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 our understanding of the metaphor in turn. Our experience supports the following observations:

- Although design metaphors do tend to move from

[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)

being richly suggestive to more fixed as the design matures, the patterns of change are far from steady, 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, including 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 [2, 14]. 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 DFM 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 machine tools, 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 making needed changes. 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. 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 for evaluation.

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 management 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 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.

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

Interactions among the design team were complicated by the fact that it was spread over four different organizations and buildings at Sandia National Laboratories. The feature recognition team was at one site, the primary 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.

A final important aspect of the development milieu is Sandia National 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 advisor.

PROTOTYPE 1: A PURE SPELLING CHECKER

The first prototype system we constructed was an attempt at a direct realization of the "spelling checker" metaphor. The feature recognizer acquired features from a solid model of a part, and sent them to the design advisor for evaluation. 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. A critic was essentially a rule for detecting a specific type of machinability problem. The advisor displayed these "fired" critics one at a time, just as a word processor's spelling checker displays misspelled words. A "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 also 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. This order is determined by the feature recognition algorithms.

The Order of Feature Evaluation

Implementing such a direct translation of the "spelling checker" metaphor required that we repair several potential breakdowns in its interpretation. 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. In contrast, feature recognition exhibited no obvious sequence, although we did highlight the current feature in the feature recognizer's display. In initial evaluations, neither the domain expert nor potential users saw any 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 order of evaluation. I believe this results 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.

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 for use in the Design for Machinability Advisor.

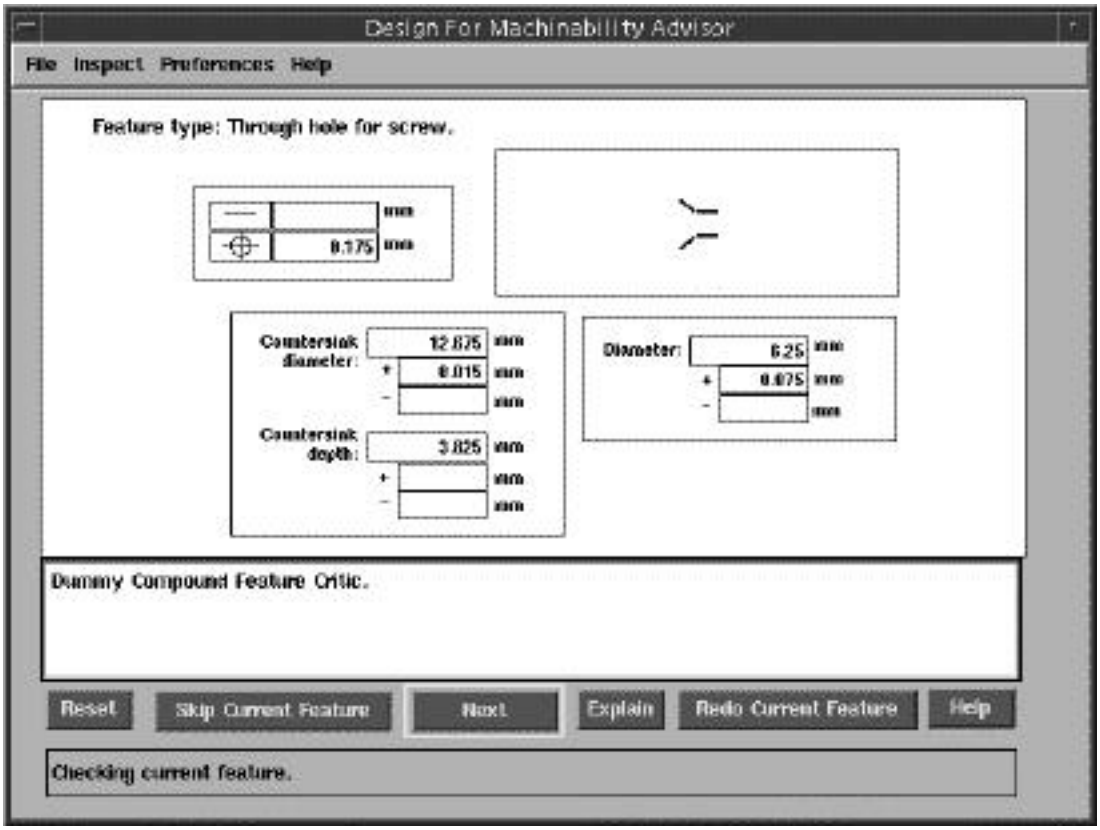

Figure 1

In order to minimize 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 feature's intended function. Our solution to this problem was to construct a database of common features that could be matched with recognized features to obtain reasonable default tolerances. For example, a countersunk hole for a standard screw could be recognized with reasonable reliability, 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 the approach to more complex, novel features. Also, by providingrecommended tolerances rather than those the user had initially entered into the design drawing, we were introducing an unacceptable possibility for confusion and error into their interaction with the system. These problems were among the main reasons for changing the approach taken in this early prototype.

Breakdowns Caused by 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 indicating words that failed to match. Formally, this is a process of matching text strings, 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 simple matching algorithms, and is most likely context sensitive.

Further evidence of the interaction between metaphor and computational complexity can be found in the customer's initial suggestion that we not consider the intended use of features 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 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.

This impact of complexity 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. But, as we discovered, it is possible to implement a semantically valid interpretation of a metaphor, only to see the it break down because of differences in complexity between the source and target situations.

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 relatively late in the development of the first prototype. Similarly, although our customer clearly specified that the advisor would be used to check finished designs, on seeing the prototype, he asked if it couldn't "detect errors as he was working."

PROTOTYPE 2: THE DUAL-USE APPROACH

The second prototype addressed many of these difficulties, but did not abandon the metaphor entirely. 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 chose to implement a "dual-use" strategy, retaining the feature-driven approach as one mode of use for the advisor, while also allowing the advisor's machinability knowledge to be used without feature recognition.

When used as a checker for existing models, prototype 2 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 that allowed the user to select features from the solid model and check them in any desired order. In providing a 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, appropriate critics would fire to alert the user.

In evaluating this "dual-use" approach, 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 value of the “front end design tool” vs. the “spelling checker” modes of use.

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 without requiring additional user input. This allowed critics to function more like "pure" textual spelling checkers, while letting the "tools" provide richer forms of advice to the user.

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. This more extensive interaction with the user moves considerably beyond the “spelling checker” metaphor.

In contrast, critics were simple if/then tests and required no user input. A typical critic detected holes whose diameter failed to match any standard size drill bit, and called this problem to the user’s attention. 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 returned to an almost pure implementation of a “spelling checker” for designs, although in doing so, it became only one component of a larger machinability tool kit.

Evaluating Solid Models

When checking an existing mechanical design, the user browsed the features, and selected those to be checked. Each tool or critic included a condition test to determine its applicability to a given feature. When the user requested a feature be checked, DFM displayed both critics that detected problems with the feature and tools that might be useful in its refinement. Selecting an entry either displayed a description of a problem (for the critics), or initialized the tool with the feature’s dimensions and launched it as a separate application.

An Initial Design Tool Kit

When the tool kit was used during initial design (i.e. without a solid model or feature recognizer), the user selected tools from a browser. Critics were not available

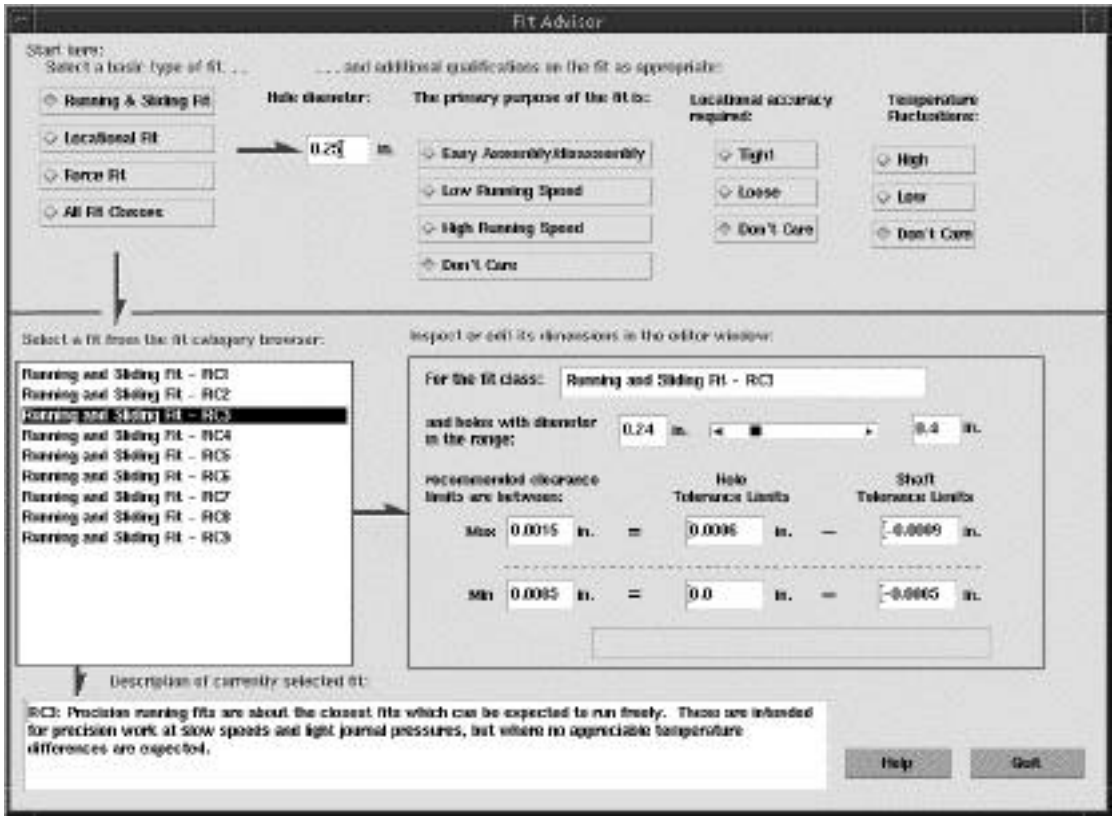

Figure 2

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.

Freed from this tight coupling with the feature recognizer, we had more freedom in designing tools that were not feature oriented. One of these tools, the SURFACEFINISH 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 particular geometric features.

Architectural Implications

A significant change in the third prototype was architectural. The advisor was decomposed into much more specialized, independent 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 clients (such as DFM) through message passing. Similarly, each feature analysis tool, such as the Pin Fit Advisor could be either invoked as an independent application, or 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. As we came to understand the design (and the "spelling checker" metaphor) better, we were able to focus more clearly on these architectural issues.

CONCLUSION

The development of the Design for Machinability Advisor was a valuable lesson in the use of design metaphors. Lessons learned from the project include:

Design Metaphors have Complex Life Cycles

The development of design metaphors closely follows the interaction theory, 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 process, 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 independently of the feature recognizer.

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 [7] 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.

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 that allowed both evaluation of existing designs and the provision of front-end design tools.

Metaphors and Computational Complexity

Although my understanding of metaphors in language and science had prepared me 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 confusing interface. 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 Function 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 decisions without adequately consulting other members of the team. This was particularly true of the feature recognition and the knowledge engineering groups. 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 common understanding engendered by the shared metaphor.

Another surprising discovery was that different interpretations of the design metaphor coexisted harmoniously within the design team. As the project progressed, the interpretations of the metaphor developed 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. Although reliance on the "spelling checker" metaphor initially caused some confusion for the customer in using the final system, they quickly adapted, and did so without feeling a need to abandon the metaphor. 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 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 strong central design metaphor. By leading us to develop an initial prototype quickly, we were able to impose a structure on what would have otherwise been an enormous and ill-defined design space. Although the system has largely moved beyond it, the "spelling checker" metaphor was, and remains an essential component of our understanding of the DFM Advisor.

The lessons to be learned are not that we should avoid metaphor in design, but that we should be prepared for the

shifts in meaning that accompany their development, and their tendency to blind us to possibilities that do not fit the metaphor's obvious interpretations. I hope that the observations made in this paper will help designers use metaphors with the flexibility and insight they require.

ACKNOWLEDGMENTS

I would like to thank Sandia National Laboratories for their generous support of the Design for Machinability Project, and Ken Washington, Steve Kleban, Dwight Miller and John Linebarger for comments on early 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 knowledge, design sophistication and friendship made this project unusually rewarding.

REFERENCES

1. Carroll, J., R. Mack, and W. Kellogg, *Interface metaphors and user interface design,* in *Handbook of Human Computer Interaction,* Hellander, Editor. 1988, Elsevier: Amsterdam.
2. Erickson, T., *Working with interface metaphors,* in *The Art of Human-Computer Interface Design,* B. Laurel, Editor. 1990, Addison-Wesley: Reading, Mass.
3. Maclean, A., *et al. Reaching through analogy: A design rationale perspective on roles of analogy*. in *CHI-91*. 1991. New Orleans: Addison Wesley.
4. Madsen, K.H., *A Guide to Metaphorical Design.* Communications of the ACM, 1994. **37**(12): p. 57-62.
5. Coyne, R., *Designing Information Technology in the Postmodern Age: From Method to Metaphor*. 1995, Cambridge, Mass.: The MIT Press.
6. Black, M., *Models and Metaphors.* 1962, Ithaca, NY: Cornell University Press.
7. Lakoff, G. and M. Johnson, *Metaphors We Live By*. 1980, Chicago: University of Chicago Press.
8. Gibbs, R.W.J., *The Poetics of Mind*. 1994, Cambridge: Cambridge University Press.
9. Hesse, M., *Models and Analogies in Science*. 1966, Notre Dame, Indiana: University of Notre Dame Press.
10. Gentner, D., *Flowing waters or teaming crowds: Mental models of electricity,* in *Mental Models,* D. Gentner and A.L. Stevens, Editor. 1983, Lawrence Elbaum Associates: Hillsdale, N. J.
11. Stubblefield, W.A., *Source Selection for Analogical Reasoning: An Interactionist Approach*. 1995, University of New Mexico:
12. Gentner, D., *Structure mapping: A theoretical framework for analogy.* Cognitive Science, 1983. (7): p. 155-170.
13. Kuhn, T.S., *The Structure of Scientific Revolutions*. 1962, Chicago: University of Chicago Press.
14. Laurel, B., *Computers as Theater*. 1991, Reading, MA: Addison-Wesley.