For Pete and the girls

1

Conceptions and Misconceptions of Knowledge Aided Design

MARC GREEN
Computer Studies Programme, Trent University, Peterborough, Ontario, Canada K9J 7B8

Although knowledge-based systems are having a major impact on most areas of human work, they still play a relatively minor role in engineering design. Research in knowledge-based design tools has developed slowly compared to work in other forms of problem solving, such as diagnosis and planning. However, research in design is growing rapidly. In 1985, Sriram and Leff surveyed the literature and found only 108 articles and books on knowledge-based design. Their later bibliography (Sriram and Leff, 1989) counted 1008 entries. Nevertheless, research in knowledge-based systems for design has lagged far behind work in other forms of human problem solving. Why has this been the case?

The reasons are at least partly due to major misconceptions about design and about knowledge-based design aids. The term "design" often evokes images of beret-clad sculptors engaged in emotional catharsis and divine inspiration. In this view, design is presumably beyond the scope of scientific understanding, analysis and formalization: no computer can ever hope to capture human creativity or aesthetics, and no computer can ever hope to embody human inspiration or intuition. In short, as the characters in old horror movies once said, "There are things that man is not meant to know!"

The articles in this book refute the mystical conceptualization of design. Engineering design, or at least relatively routine design problems, can be formalized, and it is possible to build useful knowledge-based systems. The authors in this book describe both abstract analyses of design as a problem class as well as specific systems that solve particular design problems.

This initial chapter provides a background by placing their work in a

Knowledge Aided Design

broader context. (I will note subsequent articles dealing with the same topic by the authors' name(s)). First, I will discuss both conceptions and misconceptions of design. The goal is to show that at least some types of design problem solving can be analysed, understood and aided by computers. Second, I will argue that many objections to knowledge-based design aids reflect a narrow view of both the design process and of the roles that computers can play in design. Specifically, design aids need not be expert systems which replace human designers. On the contrary, there are a variety of knowledge-based aids that could enhance human creativity and problem solving.

The next section describes the current role of computers in real-world engineering design; drafting and analysis. The subsequent section discusses the design process from three perspectives (cf. Finger and Dixon, 1989): (1) the abstract view of design as a domain-independent problem-solving method; (2) a comparison of design to other problem-solving methods; and (3) the cognitive processes used by engineers during design. Much of the perspective in this chapter derives from studies on robot gear and motor designers in a real-world setting at Spar Aerospace in Toronto. Their expertise is not due to unanalysable intuitions but rather to a set of well-defined problem-solving strategies.

The final section discusses the possible roles that knowledge-based systems could play as design aids. An important conclusion arising from the study of real-world design is that there are two distinct design problems. Most large design problems require that the artefact be decomposed into subcomponents (Tong), which are designed by different individuals or teams. The task of taking a set of requirements and creating a specification for a device component might be termed "design-in-the-small". However, there is also "design-in-the-large", the task of coordinating the activities of the groups of designers working on different subcomponents of the same design. Although design-in-the-large is the costlier and more time-consuming problem in complex systems, it has received relatively little attention. The likely explanation is that the most common model for a knowledge-based system, the expert system, assumes the design-in-the-small problem. Solving design-in-the-large problems will first require some new models for intelligent aids. I will close by suggesting some possibilities.

PRESENT USES OF COMPUTERS IN ENGINEERING

Currently, computers play two roles in design. One set of tools aids in the final drafting of the specifications. They have become so popular that the very term "computer-aided design" (CAD) has come to mean a class of graphics tools for creating drawings of the design topology, i.e. the

physical description of the artefact. Whitney (1990) notes that many people confuse engineering, design and drafting, and are under the false impression that CAD is automated design. As discussed below, engineers use CAD tools only after having made the major design decisions. To use a software analogy, CAD tools only aid the actual "coding" of the design. By the time the designer is ready to code, the artefact has been largely specified.

CAD systems have grown in sophistication since their inception. The earliest CAD products specified only two-dimensional, wire-frame models. Designers could only model 3-D shapes by tediously joining the wire-frames. The advent of solid modelling made direct 3-D object representation possible. The most recent trend has been the addition of capabilities that supplement CAD's graphics facilities. In "feature-based modelling", the user may label parts of the drawing with information required for manufacturing. For example, a cylinder may be labelled as a countersunk hole, so that the machining control system will know the type of tool needed. Another innovation is "geometric dimensioning and tolerancing", which helps the designer to deal with tolerance build-up in assemblies. Lastly, a few advanced CAD tools include "intelligence" that allows the computer to keep track of a few constraints. For example, if the designer changes the size of one dimension, the system uses knowledge of the relative dimensions, maximum and minimum sizes, etc., to maintain automatically correct proportions.

While they are possibly valuable (and the jury is still out), engineers generally use CAD tools after the major design decisions are settled. Moreover, CAD systems do not always support the best or even a desirable problem decomposition (Tong). CAD systems also fail to represent many aspects of a design, such as its rationale or pragmatics (Boose), that may be critical for maintenance, fault diagnosis and redesign (Keller *et al.*). While they have begun to address these issues, CAD tools are still primarily a drafting aid.

The second major class of computer aids is the analysis package. While such analyses may invalidate a candidate design, expert knowledge is still needed to decide why the design failed and what changes are required. Like CAD systems, they are only used long after designers have made their major decisions.

The research described in this book represents a third way that computers could aid engineering design. The notion is to enhance design by having computers make knowledge available to the designer. I have coined the term "knowledge-aided design" (KAD) to contrast these systems with the currect "computer-aided design" (CAD) technology. While designers use CAD only after the major design decisions have been made, KAD systems operate at a much earlier stage in the design process, when engineers make the major—and more expensive—decisions.

CHARACTERIZATIONS OF THE DESIGN PROCESS

The slow development of KAD systems relative to other knowledge-based systems is rooted in the common conceptions of design. Researchers have used three methods in attempting to characterize the design process. Some researchers (e.g. Simon, 1973; Cross, 1984; Clancey, 1985; Smithers and Troxell, 1990) discuss design at an abstract level. They model design as a form of problem solving having intrinsic properties that are independent of any particular application domain. A second, and related, method is to compare design to better-understood problems, such as diagnosis, planning and natural language. Lastly, an alternative approach is to study designers at work and to determine the cognitive processes and strategies used to create designs in a particular domain. General principles may then be induced from particular instances of design.

Design in the abstract

Abstract analyses often contrast design with classification. Most problems can, in theory, be solved by either method. For example, if there were enough computer power to store all possible VAX configurations, XCON (McDermott, 1982) could presumably select a solution rather than design one. In fact, one view suggests that "technological advance" means that a problem that people once solved by design could now be solved by classification. This reflects the common idea that classification deals with easier problems, a bias that has severely limited the amount of design research.

Design as a method for solving ill-structured problems

Studies of design usually stress the distinction between well-structured problems, those with definable problem-spaces, starting and goal states and operators, and ill-structured problems, those missing one or more of these attributes. When a problem is well-structured, heuristic classification is the preferred paradigm. Only when problems cannot be structured must the problem solver "resort" to design. Design, therefore, typically deals with ill-structured problems, which are more difficult to represent and solve. Smithers and Troxell (1990) suggest that the major source of ill-structure is that the problem space cannot be specified ahead of time. The very act of design is an exploration that constructs the problem space on the fly.

However, the sharp distinction between well-structured and ill-structured problems is illusory. Most problems have both well-structured

and ill-structured components. For example, Birmingham *et al.* (1989) described a system called CGEN, which designs single-board computers. It handles some subproblems by simply selecting from pre-enumerated sets of alternatives. Similarly, the Spar study reported below also found that designers use the requirements as "symptoms" to classify the problem and select from a limited set of general design categories. Further, close examination of ill-structured problems may suggest a structure which can be exploited for problem solving. In the discussion of motors below, for example, I describe the process by which the goal state (the design) evolves by an iterative process.

Lastly, an intelligent aid can still be useful even if it does not specify particular designs. Much of the pessimism about applying AI to design results from a narrow view of computer tools. The "expert system" is only one of many possible intelligent aid models. One reason that design is a difficult problem is that it seems to resist formalization. Some believe that the correct formalization has simply not been uncovered (Smithers and Troxell, 1990). An alternative possibility, however, is that some problems are simply less amenable to syntactic solutions and require a heavy injection of semantics, which can only come from humans. The basic assumption underlying artificial intelligence is that computers can solve problems by pushing symbols around and without regard to meaning, which in design corresponds to function. There are many (e.g. Rosen, 1987) who doubt that this is possible.

Rather than creating design expert systems that turn requirements into design, a knowledge-based tool could simply support decision making by suggesting alternatives, warning of constraint violations, documenting choices, or could support cooperative work by providing a common knowledge base. This allows room for humans to supply the semantics, not to mention the creativity and inspiration, presumably necessary for producing the design.

Design as an explosive search space

One explanation for the difficulty of design is the size of the problem-space. Even when the problem space can be well-specified, design is usually characterized as search through a very large and complex set of alternatives (Brown and Chandrasekaran, 1989). While heuristic classification, e.g. diagnosis, requires selection from relatively small set of solutions, the search space for design can be all possible designs. The resulting space is explosive and possible solutions are too numerous to list. It is often difficult to evaluate partial solutions or to know when the best solution has been found. Moreover, both heuristic and evaluation functions are likely to be domain-dependent.

However, the situation is not as bad as seems at first glance. First, parts

of design can often, as noted above, be reduced to classification. Second, though the search space is large, candidate solutions are sparse. That is, most solutions are so unreasonable that they need not be considered, so large areas of the space can quickly be pruned. Third, the search space can be greatly reduced in size by heuristics. Discussion with domain experts suggests, in fact, that much of their skill lies in the ability to simplify the problem-space by abstraction. Fourth, designers generally do not require an optimal solution. The problem solver can quit searching when he finds an "acceptable" solution.

Design as language

Several authors (e.g. Coyne, 1988; Mullins and Rinderle, 1991) suggest that design is similar to language. Both possess a syntax consisting of (1) a vocabulary of primitive elements (e.g. gates in a logic circuit) and (2) a grammar that specifies legal configurations of the elements (how the gates may be connected) as well as a semantics refering to the meaning (functional mapping of inputs to outputs). The task in design is to find an intersection of spaces defined by syntax (what is possible) and semantics (what the desired function is).

There are several advantages to this conceptualization of design. First, it provides a means of readily formalizing both designs and design knowledge. A logic circuit, for example, can be viewed as a collection of vocabulary elements combined in a way specified by rewrite rules (Tong). Second, it captures the common observation that requirements evolve during design. There is a search through a requirements space (defined by semantics) as well as a search through a topology space (defined by syntax). Perhaps this parallel search of distinct but intersecting problem-spaces is what makes design unique. Third, the formalisms developed by language researchers might be transferable to design. For example, the classification hierarchy for grammars may suggest a similarly taxonomy for design classes. There are already attempts (e.g. Rinderle, 1991; Mullins and Rinderle, 1991) to find a grammatical formalism for design.

Comparison of design and other problem domains

Design and diagnosis

Although often contrasted (e.g. Clancy, 1985), design and diagnosis are closely related. When a system component fails, the problem of locating the fault is called "diagnosis", but it could equally be considered a design problem: the system has simply redesigned itself, and the faulty behaviour is merely the "normal" behaviour of this new design. The task of fault diagnosis reduces to finding the design variation that behaves in the observed fashion. If this view is accepted, then the same knowledge used to create the design can be used for diagnosis and maintenance. The design representation need only be modified until the fault behaviour is produced.

This idea underlies much of the research in intelligent tutoring programs (e.g. van Lehn, 1982). The system has a student model that specifies the correct procedure for solving a problem. When the student makes an error, the program creates variations in the procedure until it produces an error that matches the student's behaviour. The system then assumes that the student made a similar error in reasoning.

There are a few instances of similar systems in design. Oxman and Gero (1987), for example, have implemented the "diagnosis-from-design strategy" in the PREDIKT system, which performs architectural design. Keller *et al.* (this volume) have taken the notion one step further by creating related design and diagnosis modules from the same device representation.

Design and planning

Design is also very similar to planning because both require construction rather than selection of solutions. Clancey (1985), in fact, divides design problems into "configuration" and planning. In configuration, the problem is to organize components into a functioning whole that will process or transform some internal system, i.e. its input. Planning is merely the opposite viewpoint of the same problem—the transformation of an artefact by interactions with its surrounding system. The similarity is so great that the problem solved by some systems, such as MOLGEN (Stefik, 1981a,b) may be classified as either design or planning.

This close relationship further suggests that research in one area will benefit the other. While not reviewed here, concepts from the planning literature will probably be useful in design. For example, the notion of abstraction levels (top-down refinement) used in ABSTRIPS (Sacerdoti, 1974) has obvious application to top-down design (Tong, 1987).

STUDIES OF DESIGNERS AT WORK

The design process also may be analysed by watching real designers at work and observing how they solve problems. While there is a large literature studying human problem solving in domains such as diagnosis (see review by van Lehn, 1991), research on human design is significantly smaller. Moreover, most studies (e.g. Carroll *et al.*, 1980; Ullman and Dietterich, 1986; Goel and Pirolli, 1989) have investigated designers in laboratories on toy projects, so studies of designers in their natural habitats are rare.

Below, I describe studies of designers performing real-world mechanical design tasks at Spar Aerospace in Toronto, Canada. The Remote Manipulator Systems Division designs telerobot systems for space and nuclear applications. A typical telerobot consists of a 6-degree-of-freedom manipulator arm, an end effector and a control system. The operator commands the position and orientation of the tip of the arm using joystick controllers. The operator may also use switches, located on a control panel, to select among different modes of operation. In the primary mode, the computer uses special algorithms to transform command inputs into drive signals for the manipulator arm joints. The basic mechanism of each joint is a motor attached to a multistage gear drive.

General approach

A telerobot manipulator arm is a complex device consisting of many interacting pieces. Figure 1 shows a manipulator arm and its decomposition into the major subsystems, each typically designed by a different group of engineers. One goal of the research was to determine how groups of designers performed problem solving on different subsystems. A second, and equally important goal, was to learn whether design expertise has domain-independent components that could be generalized across problems (Birmingham and Tommelein; Chandrasekaran).

The final goal was to study how changes in one design subcomponent propagate to other subcomponents. In robots, as in most large systems, the design task is too complex to be handled by a single engineer or group of engineers. Designs are decomposed and portions are meted out to different teams. As a result, real-world design consists of two interrelated problems. The first is "design-in-the-small", which refers to the problem of specifying individual subcomponents of the design. Most research, including the articles in this volume (but see Fox *et al.*), has focused on design-in-the-small problem solving through creation of an expert system design aide.

However, there is the greater "design-in-the-large" problem, the task of integrating the local solutions into a consistent global design. In the best of all possible worlds, the subcomponents of a device would be independent, so that decisions made by one designer (or team) on one subcomponent would not affect the decisions made by designers working on other subcomponents. However, Simon's (1973) notion of "nearly decomposable" units is rarely applicable to large mechanical systems (cf. Tong). Decisions made in one subcomponent have significant reverberations through the other subcomponents. There is little research into the problem of resolving conflict among subcomponent designers (but see Klein and Lu, 1989) although it is the major source of cost and

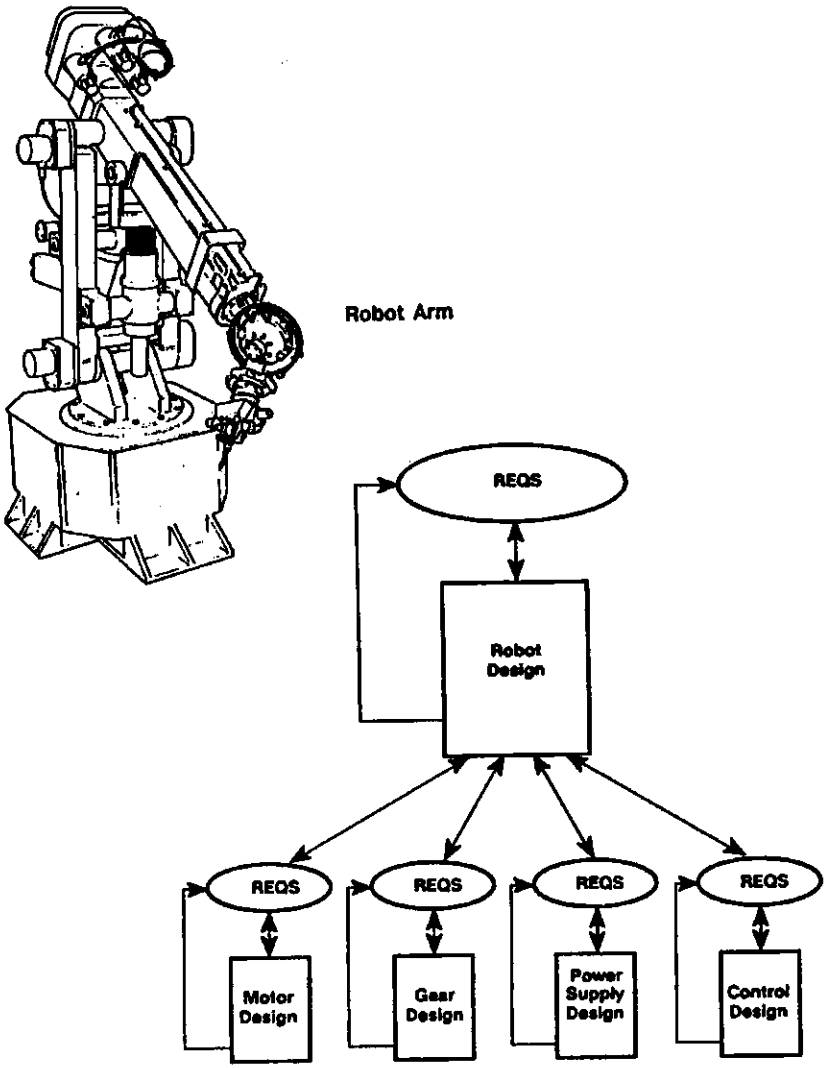

Figure 1. Top: drawing of a robot arm. Bottom: schematic decomposition of the arm into subcomponents. Arrows show the flow of requirements.

inefficiency in real-world design. The distinction between small and large design is similar to Whitney's (1990) dichotomy of design as "a technical process to be accomplished" or "an organizational process to be managed".

The Spar study concentrated on gear and motor design because they are related and important components of robots and because expert designers were available. Figure 2 shows an example of a motor–gear assembly for a robot joint. The two components are tightly coupled, so that each must be designed in the context of the other.

Both gear and motor design are relatively routine design tasks. The engineer may choose from a limited number of design subclasses to create a variation on a theme. Further, each subclass then consists of standardized pieces, and similar sets of laws and principles dictate the interactions of the pieces. As will be clear, there is also similarity in the problem-solving strategy; designers in both domains perform bouts of parameter instantiation (Brown *et al.*) interspersed with heuristic classification. In both, a major problem is to juggle many parameters while meeting constraints and producing a "good" solution. However, gear design is a much more complex task because the number of parameters, constraints and interactions is far greater.

The study was conducted by an informal methodology. I observed one motor designer and one gear designer at work for approximately 20 hours each. The study concentrated on determining general problem-solving

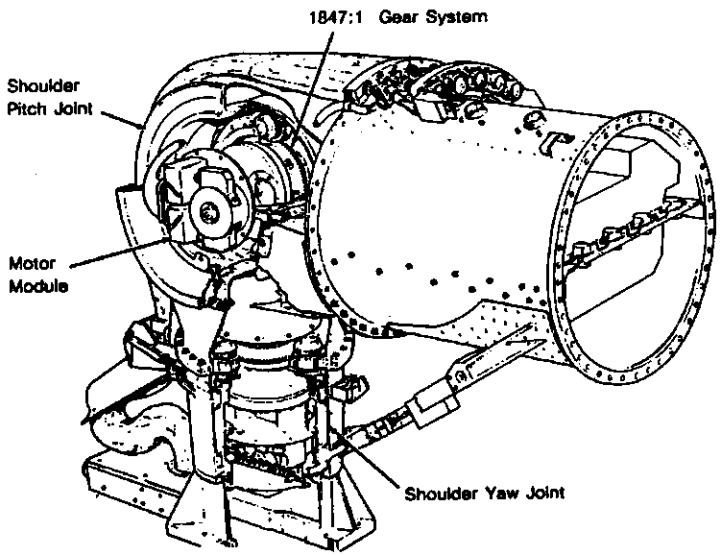

Figure 2. Gear and motor of a robot manipulator.

techniques rather than specifying in detail the knowledge required to solve the problem. The goal was not to learn how to automate gear and motor design, but to uncover the kinds of problem-solving methods and strategies used by experts.

I supplemented the behavioural observations with interviews, in order to obtain explanations and rationales underlying the decision making. The designers had the opportunity to critique my written accounts of their behaviour and thinking and to suggest changes. In general, they found my conclusions accurate.

Motor design

Motor design is interesting because engineers use a problem-solving method that combines the typical design paradigm (solutions must be constructed because the search space is too large to enumerate solutions) and heuristic classification (solutions are selected from a pre-enumerated list). As described below, there are interleaved bouts of "classification" and "design". Motor design also offered a chance to explore one source of ill-structuredness in design—interplay between what is desirable and what is possible. The requirements writer often has only a vague notion of what he really wants or what can be achieved. As a result, the specifications prompt the design, but the possible designs also suggest (changes to) the requirements (Figure 1, bottom). This is consistent with the common notion that design is an ill-structured problem because the goal is not always specified in advance. The final problem becomes clear only after exploration of the problem-space (Smithers and Troxell, 1990).

Figure 3 schematically shows the general method used by the motor designer. The input to motor design is a set of requirements defined by performance variables such as torque, and physical variables such as limits to mass, size, etc. The designer converts the requirements into a set of specifications and then produces a motor design. As shown in Figure 3, the design process has several distinct stages of problem solving and several major branch points, possible loops and iterations.

Upon receiving the requirements, the designer makes a top-level, qualitative design decision by choosing among a limited set of design classes (stepper, induction, etc.). The requirements are then "filtered" by the design class. That is, the designer maps the requirements to a set of variables, some of which are idiosyncratic to the particular design class, some of which occur in all design classes. The mapping, however, is not completely deterministic, so that the designer must apply knowledge and expertise to find the solution. The process is a form of design because the resulting specifications are a description of the desired motor. This initial stage of design culminates in the examination of motor catalogues. The designer scans the catalogue to find a motor meeting his design

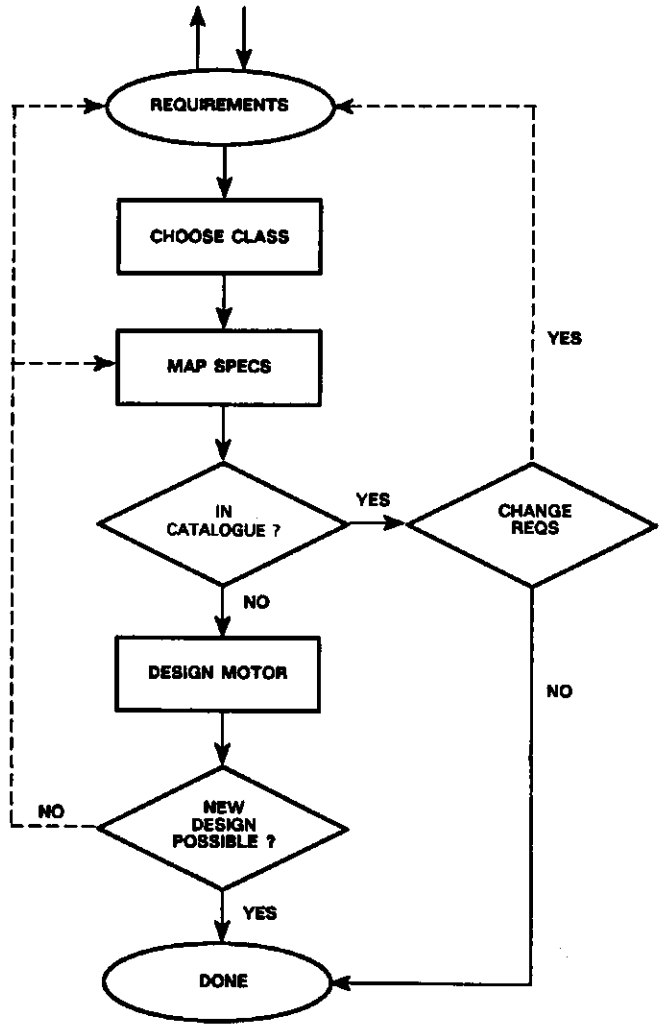

Figure 3. Flow of decisions in motor design.

specifications. Although he can accept one of the motors in the catalogue, one of two other actions is more likely.

First, the designer can find another motor which, although not precisely meeting the requirements, would produce much better performance. The designer than may ask the systems engineer for a change in the requirements. The systems engineer must decide how the requested change would affect the requirements for other subcomponents that are being designed in parallel. This triggers a set of negotiations among designers, who essentially communicate through the requirements. The nature of the negotiations is an important topic but was not covered in the present study (see Klein and Lu, 1989, on this topic). The overall architecture of the process is analogous to a blackboard system, in which each group of designers posts notices with the systems engineer.

As an alternative action, the designer may decide that existing motors do not meet the requirements. He then takes the best near-miss in the catalogue, and modifies its design to create a new, idiosyncratic version. This typically requires interpolation between catalogue items or possibly extrapolation beyond existing motors. The actual procedure is a highly complex parameter instantiation problem in which many variables must be simultaneously tuned.

The major problems faced by the designer are (1) to juggle the values of a set of variables until many constraints are satisfied and (2) to reach some performance level ("figure of merit"). A major source of the expert's skill in motor design, as in gear design, is the ordering of decisions (Marcus). Experienced designers apparently know what part of the design to specify first, and what analyses should be performed. When the last component is specified, the designer will typically iterate using an informal hill-climbing procedure. Much of the knowledge lies in the ability to evaluate each design and determine what components should be changed to improve performance (or reduce cost), and how a change in one component will affect others.

During this last phase, computers finally come into use. The designer might use computers to perform analysis on a candidate design. There are, however, no aids to help in creating possible designs.

In some cases, the requirements are so stringent that the designer cannot create an acceptable design. In this case, the motor designer might ask for a relaxation of the requirements. This is possible because the motor requirements assume the existence of sets of requirements for related subcomponents. For example, the motor design requirements assume a specific gear ratio in the gear design. To say this more abstractly, the motor requirements are part of a larger context that contains the design of other components. Because all the requirements are linked, it is possible to trade off the gear ratio against torque, creating a new set of requirements for both designs. The gear designer must create a new design so that the motor designer has a set of requirements he can manage. Of course, changes in the gear requirements will probably trigger changes in the design of controls and power supplies. Since all designs are embedded in a single context, changes to one may affect all others.

Gear design

Gear design is a significantly more complex task than is motor design. Like motor design, however, it can be studied at different levels of complexity. That is, gear designs range from relatively "simple" one-stage, parallel-axis devices, to multiple-stage designs with odd shaft angles. As in motor design, a set of requirements initiates problem solving (Figure 4). The basic function of a gear system is to match a power source to a load. The input, typically from a motor, is a given torque and speed and the output is a desired torque and speed. Gear designs can be grouped into many classes, but the research focused only on relatively simple varieties.

As with motors, the designer starts with a set of performance requirements (power, speed, etc.) as well as physical requirements (size, topology, K factor, etc.). There are also requirements which are implicitly stated, either because they need to be calculated from the givens or because the designer knows that only certain kinds of solutions will fit certain types of problems. It seems clear that the designer has some scheme for taking requirements and identifying them as belonging to a particular problem class. This is similar to the observation (Chi *et al.*, 1981) that an important source of expert knowledge is the ability to classify problems into well-established categories.

The top-level decision is the category of design. One of the important givens is shaft angle. Since designs with non-parallel axes are more complex (there are a variety of types such as bevel gears, worms, spiroids, etc.), the study did not investigate the design of such systems.

The next major decision is the number of stages. If the reduction ratio is great, the design may require several stages to transform the input to the desired output. The simplest designs contain only a single stage. Multiple-stage design is more complex but not too different qualitatively. In addition to the strategy needed to design single-stage gear systems, the engineer must allocate the reduction over the stages, so there are more variables and more interactions.

The general design methodology is to start with the overall structure of the gear system and then work towards the smaller pieces. The first step is to specify the size of gear system, the "frame size". The method used to accomplish this goal typifies the general problem-solving strategy. From the requirements, the designer calculates the torque, which he re-expresses in terms of a set of variables: K factor (contact stress), face width, gear ratio and centre-to-centre distance of the gears. He can juggle these variables within several constraints. The most important is that they result in the needed torque. However, from his experience, the designer also knows that certain combinations are desirable and others are not, so he tries to achieve an optimal set of values. Finally, the choice of

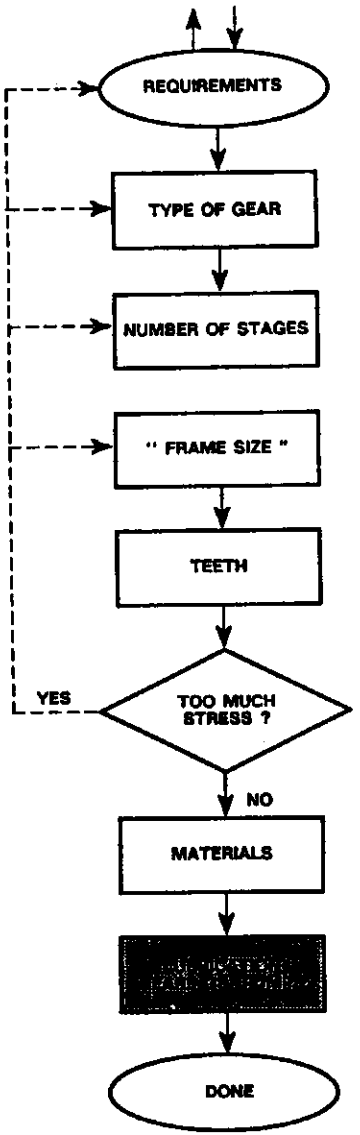

Figure 4. Flow of decisions in gear design.

parameter values is further limited because gear parts are not available in continuous values.

After specifying frame size, the designer specifies the gear's teeth parameters: number, pitch and angle. Again there are many parameters which must be juggled under both explicit and implicit constraints. For example, the designer may know that, although not stated directly, the application requires gears that are not noisy. Another implicit constraint is the available tooling. The designer might pick values simply because he knows that they can be manufactured without retooling.

The next major design step is to choose materials. The designer performs a stress analysis to determine how strong the steel must be. It is possible that the stress on the teeth is so great that the design is not feasible with available materials. This necessitates a return to some earlier stage and a redesign. Depending on severity of the problem, the designer may need to backtrack to the design of the frame size. If the problem is very bad, he may need to change the number of stages or even the type of gear. In some cases, no design is possible under the current requirements. As in motor design, the designer must request a change in the requirements.

What makes an expert designer?

After observing and questioning the gear and motor designers, it became clear that their expertise is based on several distinct abilities. Below, I summarize skills that seem to mark the expert designer. It should be re-emphasized, however, that these traits refer to designers in domains of routine design and may not be applicable to designers of more novel artefacts.

Supplying context. The requirements seldom provide enough information to create a design. This occurs in part because the client himself does not know precisely what he wants. However, another problem is that the stated requirements imply several other, unstated, requirements. The expert can "read between the lines" and supply context that reduces the search space. In one case, as noted above, the gear designer chose a tooth angle that would minimize noise. The requirements made no mention of this variable, but the designer said that the particular application called for a low-noise design. This comment obviously suggests that case-based reasoning (Gero and Rosenman, 1990) and reasoning by analogy (Brown *et al.*) are important in routine design.

Decision ordering. Strategic knowledge was a major part of the designers' expertise. The gear designer was on the verge of retirement and had been attempting to train younger engineers in the art of gear design. Several times he spontaneously noted that the major error made by novices was incorrect decision ordering. While they could learn to make the correct individual decisions, they usually failed to grasp the general outside-in strategy. By making decisions in the wrong sequence, the novices spent much more time backtracking and revising. Marcus (this volume) similarly notes that designers are much better at specifying the nature of their decisions than their strategic knowledge.

It seems likely that decision ordering is important because it ranks constraints. In most designs, some constraints are "design drivers" (Whitney, 1990) because they are more important or more difficult to satisfy than others. The expert's decision ordering may implicitly set constraint values in some optimal sequence. This hypothesis will require further study for validation.

Heuristic classification. Although the overall design problem may be termed ill-structured, it contained some well-structured components. In all cases, some design decisions fell into the heuristic classification paradigm. Each designer began by using requirements, both stated and unstated, to choose initially a class of design from a limited set. This top-level decision greatly constrained the search space.

The problem-solving technique was similar to diagnosis, where the expert maps from a set of symptoms to a pre-enumerated set of diagnostic categories. In design, the requirements acted as "symptoms" and guided the designer to choose from a pre-enumerated set of design classes. Many subsequent design decisions also involved mapping requirements, as well as constraints discovered by exploration to further subcategories of design.

Parameter abstraction. Much of routine design requires the simultaneous instantiation of a large collection of variable values. This can be a very complex cognitive task because it requires the expert to maintain a large amount of information in working memory (Brown *et al.*).

Experts seem to reduce the complexity of this problem, in part, by abstracting the parameters: where the novice sees numerous separate parameters, the expert sees a few "clumps" of related parameters that can be informally treated as single entities. The initial parameter instantiation problem reduces to trade-offs among the clumps. This strategy is consistent with the notion (Pearl, 1986; Chandrasekaran and Goel, 1988; Green and Eshelman, 1989; Green, 1991) that a common problem-solving strategy is creation of "intermediate abstractions".

ALTERNATIVE MODELS OF KNOWLEDGE-BASED AIDS

The study of real-world design revealed the necessity to distinguish between design-in-the-small and design-in-the-large. Often, the term "design" is restricted to mean the task of mapping requirements to a device specification. This design-in-the-small, however, is only a frac-

tional part of a much larger process. Design usually begins with a vague notion of desired functionality and the general requirements for achieving it. The requirements are then decomposed into sets corresponding to subsystems, which may in turn be further subdivided into increasingly detailed levels. In theory, the process ends with individual designers turning the requirements into artefact specifications.

In reality, however, life is seldom this simple. It may prove impossible to meet the requirements, so requests for changes move back up the hierarchy. Because designs are seldom truly decomposable, requirement changes in one part of the design will drive changes in requirements for other parts. Design involves the simultaneous evolution of both the requirements and the artefact specification. Design-in-the-large therefore consists of many additional problems such as requirements analysis, negotiation, communication and conflict resolution.

The importance of design-in-the-large is reinforced by the tendency of designers to concentrate on their local solution and ignore how decisions will impact on others. A knowledge-based tool that considers only isolated design problems would be likely to create a solution which may be locally correct but which cannot be integrated into a global whole. (In fact, it may be impossible even to create local solutions out of context.) As a result, there is a need for knowledge-based tools beyond those following the expert system model (see Whitney, 1990, for a similar discussion).

Expert tools

I somewhat arbitrarily divide the KAD tools into two groups. The first is traditional in the sense that they are derived from the expert system model. This class of tools varies primarily on the detail of the knowledge and locus of control between the computer and user.

Dialogue control is one of the major technical issues in development of any interactive computer tool. In beginning a research programme, it seems logical to start with "weaker" systems, where the user makes most decisions. If successful, it may be then feasible to build towards more autonomous computer tools.

The traditional expert system, the "Design Expert", has autonomous control after the user enters requirements. Other aids in this group are weaker because more control is allocated to the user and less to the computer. The degree of computer autonomy is directly related to the importance of context. Computers work well in context-free environments, such as number crunching. To the extent that the importance of context is limited, computers may be permitted more authority. When context becomes complex, it is better to let users have the final say.

Design Expert; a traditional expert system. Such a system might be

feasible when context is less important. It is possible that, if the domain is nearly decomposable, routine parts of the design (parameter instantiation, a simple classification subtask, etc.) might be completely automated (Tong).

Design Aide: the computer makes no final decisions but suggests alternatives (see, for example, the VEXED system described by Mitchell *et al.*, 1984). To use a spatial analogy, think of a design as a point in multidimensional space, where each axis is a design decision. The design assistant could suggest parts of the space in which to look for the solution. It might suggest possible operators, components, old designs which are available, etc. One likely function would be the suggestion of possible analyses and analysis tools. At Spar, the engineers complained that there were so many analysis tools in the company that they did not know what programs actually existed, or how to apply the programs they found.

Design Informer: this is a weaker version of the Design Aide; the computer does not suggest possibilities, but merely shows what alternatives are available. Although merely a memory aid, it would still be valuable, since oversight is a major source of inefficiency.

Design Demon: the computer makes no final decision but warns of constraint violations. It is common for even experienced designers to overlook relatively simple constraints. (Spar engineers confessed to missing designs that had violated Ohm's law!) As the numbers of variables and constraints grow, this type of error becomes more common. In terms of the spatial analogy, this tool "ropes off" parts of the solution space in which acceptable solutions cannot be found.

Design Strategist: a class of computer tool that makes general decisions, but not detailed ones. For example, it might operate in an ABSTRIPS-like mode, in which it makes decisions on an abstract level but allows the user to supply the details. One good possibility is that the computer might suggest the sequence of decisions (Marcus, Chandrasekaran). As noted, research to date suggests that much of an expert's skill lies not just in making final decisions but also in knowing what decisions to make next. Inexperienced designers frequently err because they fail to order the design decisions properly.

Non-traditional tools

The next set of KAD tools is not built on an expert system model. Rather than expressing the skill of a single expert operating in isolation, these tools enhance design-in-the-large (Fox *et al.*). They are as much concerned about collecting, communicating and storing expertise as about regurgitating on demand.

Design Secretary: the computer represents the current state of the design in a shared knowledge base. Each decision would be added to the representation to provide a public record of the design (Boose). The

major value of a Design Secretary is that different people or groups could see how their decisions would influence the global situation. This also provides a "what if" capability, since the effects of local changes on overall design could be readily tracked.

Further, anyone who has attended an engineering design review has probably noticed the confusion produced by the different terminology used by different groups, or worse yet, by different groups using the same words to mean different things. A Design Secretary might also smooth intergroup communication by performing some translation of terms and jargon. Lastly, the design can be stored, indexed and possibly recalled for redesign. Machine learning techniques (e.g. Tong, 1987) could also use previous designs to induce general principles.

Design Translator: engineers working on different subproblems are not the only parties whose concerns must be harmonized in the design. Designers typically view the artefact in terms of functionality—what it does. However, the design also must meet the needs of those involved in manufacturing and maintenance (Sriram *et al.*, Fox *et al.*). Constraints imposed by these additional perspectives are often left to the very end of the design process, when changes are difficult and expensive. It would be valuable to create a design representation which could be viewed from these different perspectives before the design is completed.

Design Documenter: if the design is represented in the computer, then the system could have an attached tool which would prompt the user for documentation information when the new portion of the design is entered. The documentation would include the design's teleology, which is not usually recorded (Boose). This includes information such as the reasons why the design was chosen, what alternatives were rejected, the user's satisfaction with the design, etc.

Some documentation would be primarily for humans while other parts would be available for computer processing. For example, ratings of design satisfaction could be used later to suggest the likely location of a fault. The documentation could be accessed in various ways, such as (visual or auditory) hypertext. The documentation could be invoked by clicking on a location in the graphic representation of the design, an "object" in an object-oriented implementation, or a rule in a rule base.

Design Simulator: unlike the other aids, the Design Simulator is a first-principles representation of the design (Keller *et al.*). Once completed, it could serve as the basis of future designs or in maintenance and diagnosis of the manufactured artefact.

This list is not meant to be exclusive or exhaustive. Some models overlap, and more than one model might be applied to any particular application. However, any tool is likely to involve many of the same central research problems: representation, knowledge acquisition, human–machine interfaces, etc.

DIMENSIONS IN KNOWLEDGE-BASED DESIGN AIDS

One way to develop new models is to create a model space. First, I suggest a set of primitive dimensions forming the space. Next, each current model is located in the space. Lastly, new models can be generated by looking into unexplored areas of the space.

An examination of current knowledge-based systems suggests that the following dimensions characterize systems.

(1) *Dialogue initiative*. In the archetypal expert system model, the user enters the symptoms or requirements, and then the computer controls the dialogue. The computer initiates all interactions by asking questions and providing answers. In tools at the other end of the continuum, such as database query systems, the computer waits for the user to initiate any dialogue. Mixed initiative programs could provide various degrees of user control.

(2) *Autonomy*. A related dimension is the degree to which the computer can make decisions without human intervention or approval. At one extreme, a computer could act autonomously without human–machine dialogue. A less autonomous tool might make decisions but ask the user for confirmation, suggest likely possibilities, or simply remind the user of his available options.

(3) *Depth*. Expert systems are typically "shallow" because they represent only an expert's heuristic knowledge. This contrasts with "deep" or "first-principles" systems which aim to represent and simulate the physical world. Shallow systems are often brittle and fail disastrously when the problem falls outside the bounds or between the cracks of expert heuristics. Deep systems are presumably more flexible and can deal better with unexpected contingencies (see Keller *et al.*).

(4) *Individual–shared use*. A computer aid may be designed for use by an individual or a group. Expert systems exemplify single-user systems; a person enters requirements of symptoms and waits for the answer. However, design of complex artefacts is a highly social function with interactions among many people. Since the overall design is decomposed into subsections, local solutions must be integrated. A useful tool is likely to need a shared, public knowledge base which could be viewed differently by users from various disciplines (mechanical, electrical) and perspectives (Sriram *et al.*, Fox *et al.*). The recent research in computer-supported cooperative work (CSCW) should be of direct relevance to development in design aids.

(5) *Integration of knowledge acquisition and regurgitation*. At one end of this continuum the user supplies knowledge, while at the other end he receives it. In expert systems, knowledge acquisition and regurgitation are distinct stages. A typical application will require an initial phase of

knowledge engineering to build the knowledge base. After debugging, the system is put to work regurgitating the stored knowledge on demand. However, an alternative model views knowledge acquisition and regurgitation as interleaved processes. When a design representation is constructed, the users are alternately entering new knowledge and drawing on the old knowledge stored in computer. The balance between acquiring and supplying knowledge can be altered to provide a spectrum of systems.

(6) *Enhance–replace the designer*. The term "expert system" is based on the notion of a computer expert replacing a human expert. The alternative view, that computers can enhance but not replace human talents, is more applicable to design. Creativity is a case in point, since it is probably not realistic to expert a computer to be very creative. By performing tedious tasks, however, a computer tool might promote exploration and creativity in human designers.

(7) *Type of explanation*. A computer can provide explanations at different levels, including form, function and teleology. A design decision may be made from form considerations, as, for example, when limited space restricts size. Explanations may be in terms of syntax, semantics or pragmatics.

Expert systems models cluster tightly into a single corner of the "tool" space: they control dialogue: they are autonomous, shallow, and single-user; they separate knowledge acquisition and regurgitation; they replace experts and employ syntactic levels of explanation. The vast majority of the space is unexplored.

CONCLUSION

Pessimism about the creation of KAD tools is based on misconceptions of both the design process itself and the role of knowledge-based tools. Much "routine" design work has structure and is amenable to many of the techniques used in diagnosis and planning. Studies of design engineers performing design-in-the-small showed that their expertise incorporates bouts of heuristic classification and parameter instantiation, both problem-solving methods used in other types of knowledge-based systems.

There are also misconceptions about the role played by knowledge-based systems in design. A computer system does not have to replace human creativity or inspiration to be useful. It can aid design-in-the-small by offering suggestions, supplying constraints, etc. KAD systems also can be useful in design-in-the-large, by providing a shared design representation and documentation.

ACKNOWLEDGEMENT

Support was provided by PRECARN Associates, Spar Aerospace and NSERC research grant no. 46297 to M.G.

REFERENCES

Birmingham, W. P., Gupta, A. P. and Siewiorek, D. (1989) The Micon System for Computer Design. *IEEE Micro*, **9**, 61–67.

Brown, D. and Chandrasekaran, B. (1989). Design problem solving structures and control strategies. *Research Notes in Artificial Intelligence*. London: Pitman.

Carroll, J., Thomas, J. and Malhotea, A. (1980). Presentation and representation in design problem-solving. *British Journal of Psychology*, **71**, 143–153.

Chandrasekaran, B. and Goel, A. (1988). From numbers to symbols to knowledge structures: Artificial intelligence perspectives on the classification task. *IEEE Transactions on Systems, Man and Cybernetics*, **18**, 415–424.

Chi, M., Feltovich, P. and Glaser, R. (1981). Categorization and representation of physics problems by experts and novices. *Cognitive Science*, **5**, 121–152.

Clancy, W. (1985). Heuristic classification. *Artificial Intelligence*, **27**, 289–350.

Coyne, R. (1988). *Logic Models of Design*. Sydney: Pitman.

Cross, N. (1984). *Developments in Design Methodology*. New York: Wiley.

Finger, S. and Dixon, J. (1989). A review of research in mechanical engineering design. Part I: Descriptive, prescriptive and computer-based models of design processes. *Research in Engineering Design*, **1**, 51–67.

Gero, J. and Rosenman, M. (1990). A conceptual framework for knowledge based design research at Sydney University design computing unit. *Journal of Artificial Intelligence in Engineering*, **5**, 65–77.

Goel, V. and Pirolli, P. (1989). Motivating the notion of generic design within information-processing theory: the design problem space. *AI Magazine*, Spring, 18–35.

Green, M. (1991). Acquiring and representing knowledge: Causal relations vs. metric formalism. *IEEE Transactions on Systems, Man and Cybernetics*, submitted.

Green, M. and Eshelman, L. (1989). Knowledge acquisition for causal reasoning. *Joint International Conference on Artificial Intelligence Workshop on Knowledge Acquisition*, 68–70.

Klein, M., and Lu, S. (1989). Conflict resolution in cooperative design. *AI in Engineering*, **4**, 168–180.
McDermott, J. (1982). R1: A rule-based configurer of computer systems. *Artificial Intelligence*, **19**, 39–88.
Mitchell, T., Steinberg, L. and Shulman, J. (1984). Knowledge based approach to design. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **PAMI-7**, 502–510.
Mullins, S. and Rinderle, J. (1991). Grammatical approaches to engineering design, Part II: An introduction and commentary, *Research in Engineering Design*, **2**, 121–135.
Oxman, R. and Gero, J. (1987). Using an expert system for design diagnosis and design synthesis. *Expert Systems*, **4**, 4–8.
Pearl, J. (1986). Fusion, propagation and structuring in belief networks. *Artificial Intelligence*, **29**, 241–288.
Rosen, R. (1987). On the scope of syntactics in mathematics and science: the machine metaphor. In J. Casti and A. Karlqvist (eds) *Real Brains, Artificial Minds*. New York: North-Holland, pp. 1–23.
Rinderle, J. (1991). Grammatical approaches to engineering design, Part II: Melding configuration and parametric design using attribute grammars. *Research in Engineering Design*, **2**, 137–146.
Sacerdoti, E. (1974). Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, **5**, 115–135.
Simon, H. (1973). The structure of ill-structured problems. *Artificial Intelligence*, **4**, 181–200.
Smithers, T. and Troxell, W. (1990). Design is intelligent behavior, but what's the formalism? *AI-EDAM*, **4**, 89–98.
Sriram, D. and Leff, L. (1989). Knowledge-Based Systems in engineering. An annotated bibliography. *SIGART Newsletter*, **109**, 38–97.
Stefik (1981a). Planning with constraints (MOLGEN: Part II). *Artificial Intelligence*, **16**, 111–140.
Stefik (1981b). Planning with constraints (MOLGEN: Part II). *Artificial Intelligence*, **16**, 141–169.
Tong, C. (1987). Toward an engineering science of knowledge-based design. *Artificial Intelligence in Engineering*, **2**, 133–166.
Ullman, D. G. and Dietterich, T. A. (1986). Mechanical design methodology. In G. Gupta (ed.) *Computers in Engineering*. New York: American Society of Mechanical Engineers, pp. 173–180.
van Lehn, K. (1982). Bugs are not enough: Empirical studies of bugs, impasses and repairs in procedural skills. *Journal of Mathematical Behavior*, **3**, 3–71.
van Lehn, K. (1991). Problem solving and cognitive skill acquisition. In M. I. Posner (ed.) *Foundations of Cognitive Science*. Cambridge, MA: MIT Press, pp. 527–579.
Whitney, D. (1990). Designing the design process. *Research in Engineering Design*, **2**, 3–13.

2

Design Problem Solving: A Task Analysis[1]

B. CHANDRASEKARAN
...tory for AI Research, Department of Computer & Information ...ce, The Ohio State University, Columbus, OH 43210, USA

...K-STRUCTURE METHODOLOGY

...oblem solving is a complex activity involving a number of ... and a number of alternative methods potentially available for ...sk. The structure of tasks has been a key concern of recent ... in task-oriented methodologies for knowledge-based systems ... 1985; Chandrasekaran, 1986; McDermott, 1988; Steels, 1990). ... to conduct a task analysis is to develop a *task-structure* ...ekaran, 1989) that lays out the relation between a task, ... methods for it, the knowledge requirements for the methods, ...btasks set up by them. The major goal of this chapter is to ...sk structure for design as a knowledge-based problem-solving

...earch in a space of subassemblies

...artefacts that are meant to achieve some functions within some ... is an important class of design with characteristic properties ...Pirolli, 1989). We concentrate on this class of design problems ...er.

...iently complex versions of the design problem, a common ...ges for design as a process: it involves mappings from the

...evolved over a number of years. Earlier versions have appeared as Chapter ...d Chandrasekaran (1989) and in *Research in Engineering Design* (1989), **1**, ...lon previously appeared in *AI Magazine* (1990), **11**, 59–71.

...d Design ...0269