INTELLIGENT SUGGESTIVE CAD SYSTEMS

by
Mark John Jakiela

Technical Report
UM-MEAM-88-07

INTELLIGENT SUGGESTIVE CAD SYSTEMS

by

Mark John Jakiela

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Mechanical Engineering)
in The University of Michigan
1988

Doctoral Committee:

Associate Professor P. Papalambros, Chairperson
Professor L. Conway
Professor J. Eisley
Assistant Professor E. Kannatey-Asibu
Associate Professor D. Kieras
Professor R. Volz

ACKNOWLEDGEMENTS

There are many who deserve my thanks for their support while I completed this work.

First, I thank the dissertation committee, who provided valuable input and ideas that guided and strengthened this effort. In particular, Professor Panos Papalambros, my thesis chairperson, has supported me since my undergraduate days and has brought me to this point. In addition, he had the courage to support the nontraditional research ideas that led to this work. I am extremely grateful.

Second, I thank friends and coworkers. In the early stages two older friends, Mr. Jerry Raski and Mr. Jung-Ho Cheng, "showed me the ropes" and provided much valuable advice. In the later stages, two younger friends, Mr. Nick Tzannetakis and Mr. J.R. Jagannatha Rao, benefitted from my experiences and provided much help and comradeship. Mr. Craig Thams managed the CAD computing system expertly and Mr. Peter Bielby coded many of the small software routines, often from less than adequate specifications.

Third, I thank the students of the winter 1988 "Design Laboratory" course who generously volunteered their time to be experimental subjects.

Finally, I thank my family, and especially my parents. They have provided endless support and encouragement and have seen me through good times and bad. Without them I could have done nothing.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF APPENDICES

Appendix

CHAPTER I

INTRODUCTION

Two Major Issues

This thesis addresses two major issues.

The first is the need for a computational aid for preliminary design. The entire design process can be thought of as an iterative two-phase cycle. *Synthesis* creates a solution to a design problem and *analysis* determines if the solution is satisfactory. If it is not, a new design solution is synthesized and the process starts again.

There is a wide variety of computational aids for the analysis phase of the design cycle. Finite element methods, for example, can determine stresses and strains in a proposed design. Kinematic and dynamic analysis will reveal the motions and attendant forces in a mechanism under consideration. Both of these techniques, however, require that a proposed design exist. They do nothing to help the designer formulate a design solution; they only analyze a completely specified design idea.

It is a reasonable argument that computer-aided design and drafting systems do actually aid the design synthesis process since they facilitate the creation and visualization of a record of a design solution. The important point is that they aid in *archiving* the design solution, not in *creating* it. The human designer is still responsible for generating all design solutions. It seems highly unlikely that attractive,

accurate renderings of designs will cause designers to create different, better designs. (Indeed, they may in fact cause premature satisfaction and unwarranted complacency in designers.)

The second major issue is the need for a computational framework for concurrent engineering. Typically, products are first designed to meet functional requirements and then, once most design decisions have been made, is consideration given to other aspects, such as manufacture, distribution, repair, and disposal. In a concurrent engineering design approach, consideration of these aspects influences the initial design of the product. Ideally, the product and all related processing are designed at the same time. By doing this, costs associated with the "downstream" aspects are minimized. Redesigns and design revisions are less likely because initial design decisions are more informed. This general philosophy goes by several names, including "life-cycle engineering" and "simultaneous engineering," and encompasses many different "design for ..." (assembly, manufacturability, etc.) techniques.

At present, implementing this philosophy is more of an administrative issue than a technical one. The concurrent engineering idea can be implemented by simply soliciting the input of personnel familiar with each life cycle aspect during the initial design stages. We hope for a computational implementation of this process. Experts are not always available and their time is very expensive. Encoding their expertise once in a system tailored to helping with concurrent engineering would faciltate the process and perhaps cause dramatic improvements in design productivity.

What is needed, then, is a system that helps designers obtain better design ideas, and allows the computational implementation of a concurrent engineering design approach. The system developed here attempts to achieve these two goals.

Intelligent Purposeful Suggestions

To address these two issues, the idea is to devise a system that will make intelli-

gent, purposeful suggestions to the designer during the preliminary design phase.

An example demonstrates the idea. Figure (1.1) illustrates how this system would operate in a typical design process. The figure shows four successive screens of a CAD modeling session. In Figure (1.1a), the user has created a block. In Figure (1.1b), the user has subtracted a groove from the block in order to satisfy some design requirement. Assume that the user's concern is to maintain the dimension indicated. The system recognizes that a feature has been added to the block and in Figure (1.1c) graphically suggests that the groove be moved to the center of the block face. The system's concern is to alter the feature to make the entire design 180-degree symmetric about the X-axis, perhaps to facilitate later mechanical handling of the part. If the user does not understand the system's motivation, an explanatory text interactivity could be carried out on a seperate terminal. In Figure (1.1d) is shown the user's design solution. The user has widened the groove in order to both maintain the important dimension and obtain symmetry. There has been an effective mixture of human and artificial intelligence, with the human intelligence creating a design to meet specifications and with the artificial intelligence analyzing each design step and suggesting improvements relative to some previously encoded knowledge domain.

The system in the above example performs two tasks. First, it analyzes the design to determine how well its own design goals are met. In this case the design goal was to obtain X-axis symmetry. Second, based upon the analysis, it alters the design to improve it with respect to its goals. To perform the analysis, the system must access a representation of the evolving design, for this is what is being analyzed. It must also access representations of domain analysis knowledge so it can carry out the analysis. Finally, the system must possess methods to alter the design in order to make the suggestions to the user. The system does not, however, use any representation of the user's design goals. The user clearly had a goal to maintain a particular dimension and the system's suggestion did not meet the goal. The system

has no concept of the user's intentions.

Considering these system capabilities, it is evident that the user also performs two tasks. The first is to perform design steps to meet their design goals. The second task is to revise their design steps to accomodate in some way the suggestions of the system. The user must ensure that their design goals are satisfied while allowing the system to influence the way they achieve their goals. This could result in a range of user behavior, depending on how the user is influenced. At one extreme, the suggestions may differ very little from the user's design step, causing minor changes in the user's design. At the other extreme, the suggestions may cause the user to obtain a completely new idea of how to meet their goal.

It is important to note that the system operation is not influenced by the specifications of what is being designed. The user will begin a design session with some design specifications, most importantly specifications of design function. Assuming that the user is a capable designer, the sum total of their design goals will meet the design specifications. The system only responds to the user's actions; it can do nothing with the requirements that are motivating the user's design steps. Because of this, the system could *not* be considered an automated designer. It could not create a design independently of the user. On the other hand, since the system operation does not depend on the design specifications, it can be applied to a wide range of design problems. It can apply its analysis and suggestive capabilities to many different classes of objects.

An obvious question arises from the description of this type of design aid. Can suggestions favorably influence designers? If they can, then such a system can help in the preliminary design process, thereby achieving one of our goals. Additionally, if the suggestions embody some aspect of the product life cycle normally not considered during preliminary design, then the system will provide a computational method for concurrent engineering, achieving the other goal. Also, it is significant that

the suggestions are not related to the product specifications. If the suggestions help designers create better designs, it will demonstrate that design improvement assistance need not be specific to a particular class of designs, making such a system more broadly applicable.

The research described in this thesis attempts to answer the above question. A system like the one described above is developed and configured to provide suggestions in different ways. Following this, the system is tested with human users to determine how effectively it provides assistance during preliminary design.

Relation to Other Disciplines

This research is highly interdisciplinary in nature and relates to several other very fertile research areas. Here, we briefly describe some of these relationships and the potential contributions to each.

In the general area of *design,* the proposed system would actually help people be better designers by providing design improvement information during the preliminary design process. To date, many have considered it impossible to augment true design ability, since this ability was considered to be an abstract, almost artistic talent.

In the area of *computer-aided design,* the system developed here will actively take part in the design process by autonomously altering the design in response to the user's actions. An architecture for achieving this capability will be specified.

The intelligent CAD system proposed will not require any sophisticated applications of *artificial intelligence* programming techniques, but will result in a production rule system specifically designed to interact with a feature-based CAD model. In addition, heuristics for making design improvement suggestions will be investigated.

In the area of *human-computer interaction,* this work will provide a starting point for the study of suggestion-making CAD interfaces. The results of the user tests will aid in the design of interactivity schemes for systems that actively take

part in design processes. The question of how to motivate and stimulate users to create better solutions to design problems is obviously of great importance.

Organization

This thesis is organized into ten chapters.

Chapter Two discusses background and related work. Chapter Three introduces design for assembly as a design knowledge domain. Chapter Four reviews the author's previous efforts to devise suggestion-making systems. Chapter Five formalizes some pertinent concepts. Chapter Six specifies an architecture for a suggestion-making CAD system. Chapter Seven derives two possible suggestion-making interactivities and provides example design sessions for them. Chapter Eight discusses knowledge engineering issues that arose during a system implementation. Chapter Nine describes some initial user tests that were performed and chapter Ten provides conclusions.

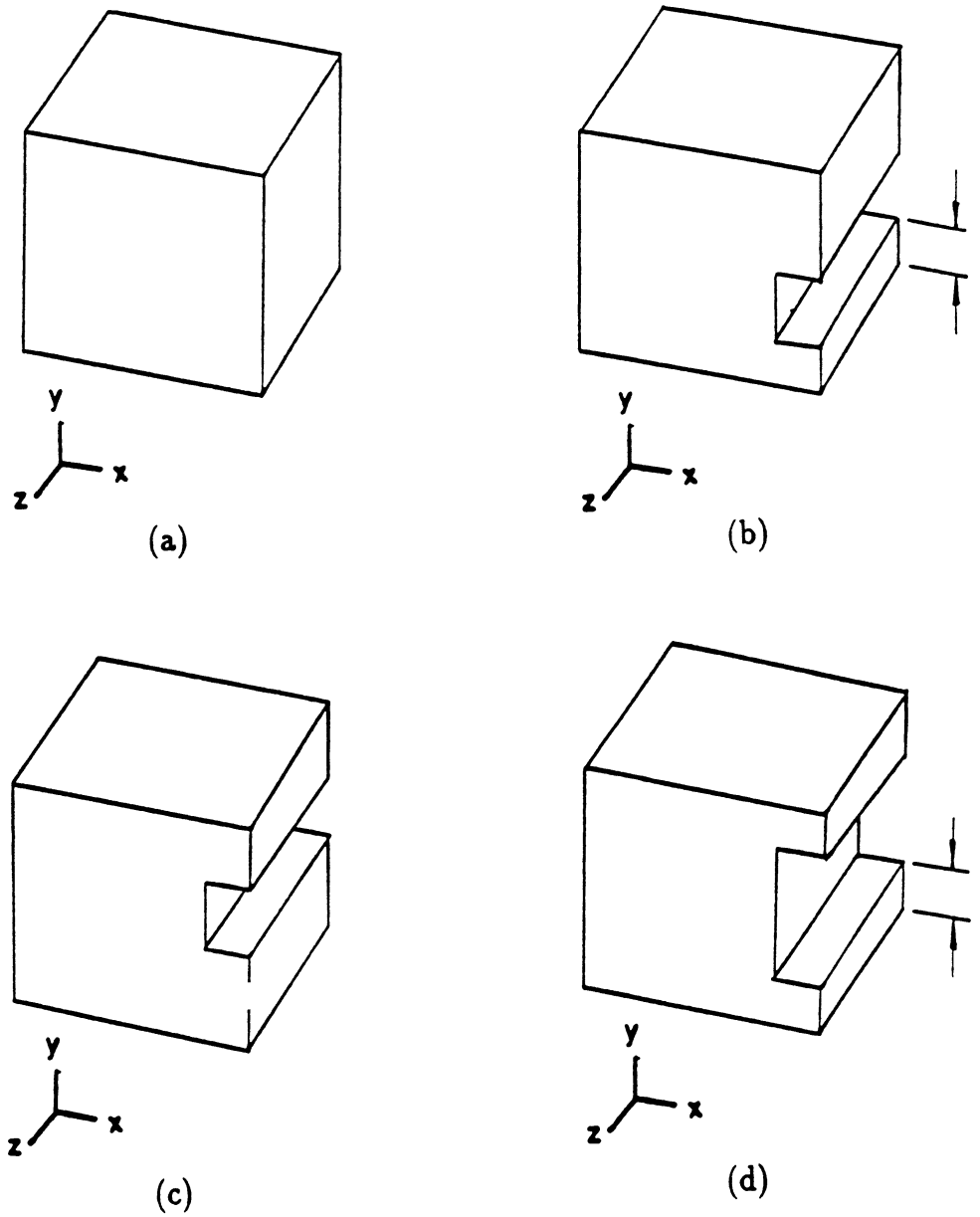


Figure 1.1: Proposed interactivity with the intelligent CAD program. (a) Initial design. (b) User input. (c) System suggestion. (d) Revised user input.

CHAPTER II

BACKGROUND AND RELATED WORK

Introduction

This chapter provides background for the research by reviewing some related work of others. This related work has been grouped into seven general categories, which are represented with sections of this chapter. Six of the categories derive from the discussion of two major issues and four related disciplines found in Chapter One. The other category, *similar systems*, is required to discuss developed systems that are somehow similar in purpose or philosophy to the system described in this thesis. *Similar systems* are discussed first. This is followed by discussion of works related to the two major issues: *preliminary design aids* and *concurrent engineering*. The next four sections discuss related work in the four disciplines of *design, computer-aided design, artificial intelligence,* and *human-computer interaction.* A final *conclusion* section closes the chapter.

These divisions are not very precise. Some of the research reviewed below obviously makes contributions to several of the categories. The categorization of the various research efforts reflects the author's opinions on how the efforts most significantly contribute to the idea of an intelligent suggestive CAD system. Also, with such

a variety and large amount of related work, it is possible that some pertinent efforts have been overlooked. This possibility is regrettable, but seemingly unavoidable.

Similar Systems

Of the similar systems that were found in the literature, one descibed by Runciman and Swift seems to be the closest in purpose [Run.1] (see also [Swi.1]). The system they describe uses a CAD model of a part to estimate the expected costs of automatically handling the part. Additionally, minor graphical recommendations are made directly to the CAD model. The system accesses a CAD representation and uses a production rule system to perform its task.

The system is similar because it addresses the same concurrent engineering aspect as is used in the system developed here (see Chapter Three) and makes recommended changes to the CAD model. The primary differences are in emphasis and sophistication. The emphasis of the work described in [Run.1] is to accurately predict the cost of automatically handling the part, given the CAD model. The authors measure the performance of their system by comparing its predictions with those of a human expert. They make clear that the purpose of their graphical recommendations is to make very minor changes to the part that will yield decreased cost predictions. In the system described in this thesis, the emphasis is on influencing the designer to make better design decisions. These better decisions will, of course, result in lower costs for the subsequent handling of the part. There is no effort in the system we propose, however, to suggest only minor changes that maintain the user's original idea. Gross changes may be suggested. These may cause the designer to obtain a completely different design idea with handling costs lower than any design that would result from a minor change. The system described in the subsequent chapters also seems to be more sophisticated and robust than that described in [Run.1]. In certain situations their system requires significant help from the user to derive the

necessary facts from the design representation. The system described here requires none. Additionally, their system represents the design as a list of corner points of a planar polygon which is the silhouette of the part. This is actually a representation of a *drawing* instead of an object, and it suppresses the three-dimensional nature of the design. As will be seen in Chapters Five and Six, the system developed here employs a three-dimensional feature-based model, a representation that facilitates necessary processing.

Another similar system with a more complete representation of the design is described by Luby et al. [Lub.1]. Their system is a sophisticated CAD modeler that automatically evaluates the suitability of a design for manufacture by casting. The user creates a design with meaningful geometric features. These features are part of a heirarchical object-oriented representation of the design. This representation includes the design features and various levels of CAD entities (planes, lines, points, etc.). Relationships between the design features and the CAD entities are maintained in the representation. This allows a "spreadsheet interface," where altering one of the design features also appropriately alters the features bordering it. In operation, the user first builds a CAD model, and then asks for analysis and recommendations. Recommendations are made without changing the CAD model: the user must modify the design.

The system is similar because it employs a design-with-features interface[1] and provides design analysis and recommendations. The sophisticated features interface seems to be the major thrust of this work. This interface is considerably more capable than the one employed in the system developed here. The design improvement recommendation capabilities, on the other hand, seem to be more limited. The system does not autonomously alter the CAD model, which is an important part of our

---

[1] The design-with-features approach taken in the system developed here is explained in Chapters Five and Six.

approach. Upon request it analyzes the design, notes any problems, and recommends possible improvements, but it is up to the user to incorporate the recommendations. Like the system described by Runciman and Swift, the emphasis seems to be on the automation of a design analysis task rather than the stimulation of new design ideas.

It is conceivable that an additional benefit of using a system that would produce the output shown in Figure (1.1) is to learn more about the encoded concurrent engineering domain. By viewing the suggestions, the user might learn what design characteristics are favorable with respect to the domain. Such a system, therefore, seems to be somewhat similar to some type of intelligent tutoring system. Sleeman and Brown have edited a collection of work in this area [Sle.1]. Existing intelligent tutoring systems often have a typical structure and mode of operation. The structure consists of two models: a model of the student that is derived from the interactivity, and a model of an expert that is encoded beforehand. The system will operate by comparing the student model to the expert model to determine pertinent differences in skill and knowledge. These differences are then used to guide subsequent instruction.

One example is the WEST program described by Burton and Brown [Bur.1]. This program coaches users on how to more effectively play a computer game. The game involves moving along a path of discrete spaces, where the distance of a move is related to an arithmetic expression formed by a player. Players must form these expressions with a set of numbers that result from spinning several dials. The object is to reach the end of the path. There are also several strategic possibilities related to jumping ahead and forcing back the opponent. Usually, the student plays against the tutor. The tutor will recognize deficiencies in the student's play and will suggest alternate improved moves.

Another example is a geometry tutor described by Anderson et al. [And.1]. This system contains production rules that represent the decisions that might be made

during the formulation of a proof in high-school level geometry. One set of productions represents appropriate steps in the proof and can be considered the expert model. Another set, the so called "bug catalog," represents incorrect steps. The system monitors the student and infers which type of production he is using at a given step in the formulation of the proof. If the production is one of those in the bug catalog, the system provides pertinent instructions to the student.

The behavior of these two systems *is* similar to that shown in Figure (1.1). The basic operation of monitoring a user and providing advice when needed is demonstrated in Figure (1.1). A major difference between a tutoring system and the system described in this thesis is, of course, that in the system described here there is no intention to provide tutoring. The intention is instead to provide useful information that may stimulate better design ideas. If this information is properly brought to the design situation, there is really no need for the user to learn it. Another difference is that the system proposed here does not exhibit the level of expertise in its domain that the tutoring systems do. In particular, note that the WEST program can actually play the computer game and that the geometry tutor can actually complete a proof. The analagous capability for a suggestion-making program would be for it to complete a design on its own. This is not to say, of course, that game playing and theorem proving are somehow equivalent to design; the distinction is that the tutoring systems can perform the task the user is trying to learn, while a suggestion-making system cannot perform design. This is due to the complexity of the task. Design requires a large amount of information and knowledge. To date, systems that can carry out a design independently have tended to be very specific to a certain class of objects (see the "Artificial Intelligence" section below). This brings the amount of knowledge required down to a workable level. The system we propose instead limits the amount of knowledge by making it specific to a particular engineering concern. This knowledge can then be applied to a range of different

objects if the user participates in the design process

There are tutoring systems that are more aligned with this approach. One example is the SPADE-0 program developed by Mark Miller [Mill.1]. This system attempts to promote good program planning, development, and debugging practice in the coding of simple graphics application programs. Its basic operation is to prompt the user about what to do next in an orderly software development plan. Importantly, it could not take program specifications and automatically generate the correct program. In this way, it is like the system we propose. It is unlike it in the way it responds to the user's actions. It does not, for example, propose alternate ways of coding a portion of the program that the user has just finished. Its purpose is to encourage the user to follow a good program planning and debugging approach.

A system that does react more to a user's actions is described by Shrager and Finin [Shr.1]. Their program keeps track of a user's interactivity with an operating system and informs the user of more efficient ways to use operating system commands. If, for example, a user simply wants to rename a file but does not know the proper command, the user might copy the file to a file with the desired name and then delete the old file. The Shrager and Finin system will infer their desire and help them with the delete command. The emphasis, therefore, is on understanding what the user is trying to accomplish in order to help them do it more efficiently. The situation is somewhat different with our suggestion-making program, where the emphasis is on altering what the user did in order to prompt a more desirable user action.

A final related system is the SACON program developed by Bennet and Englemore [Ben.1] which provides advice on the use of a structural analysis program. The user first participates with the program in a question and answer session which provides the system with required information. The system then makes recommendations about parameters for a particular run of the structural analysis program.

This system does provide assistance, but does little if anything to provide the user with new ideas.

Preliminary Design Aids

The first major issue was the need for a preliminary design aid. In this section we discuss research related to this issue. Two basic approaches were found in the research that was reviewed. The first was to stress some *idea* that provided an approach to preliminary design. Two of these ideas are discussed below. The second approach was to *develop a system* that actually performs a preliminary design of some type of object. Four of these working systems are described. The research outlined in this section could rightfully be found under one of the other categories. The *ideas*, for example could be found with the other approaches to design. They are described separately here because they are thought to be more distinctly related to preliminary design and somewhat more formalized. The *developed systems* could be grouped with the other artificial intelligence applications. They are described here because they seem to be more focused on issues pertaining to preliminary design. Although these systems are not actually aids to preliminary design, they do emphasize the preliminary design process.

One idea that can be applied in the preliminary design process is the axiomatic approach to design that was proposed by Suh et al. [Suh.1]. The basic theme to an axiomatic approach is to apply axioms when one must make design decisions in uncertain situations. If all pertinent information was at the designer's disposal at the time of the design decision, the decision would be very easy to make. When information is lacking, the decision can still be made by considering some generally true design axiom and applying it to the situation. These axioms are very general ideas for which there seem to be no counterexamples. An example axiom is that

a design's primary functional requirement should be satisfied first and other functional requirements should be satisfied in order of importance [Suh.1]. Rinderle and Suh developed quantitative measures of "functional coupling" which they related to one of these design axioms [Rin.1]. The only real relation of an axiomatic approach to the work described in this thesis is that the suggestions that are output by a suggestion-making program can be thought of as fairly specific axioms for a concurrent engineering concern.

Another idea, put forth by Lansdown, is the idea of design as a modification of prototypes [Lan.1]. A prototype is a generally accepted concept of something. A kitchen chair, for example, is a prototype for all chairs. Prototypes would be represented as a concept with a list of features. Two parameters related to this representation are "c" and "m." "C" is a measure of how close a particular design is to its prototype, and in the chair example would measure the "chairness" of the design. If the prototype is a kitchen chair, an office desk chair has a higher "chairness" than a bean-bag chair. "M" is a measure of how much the list of features of the design differs from the list of features of the prototype. The author proposes that an innovative design is one that is far from its prototype with a very similar list of features. Another idea outlined in the paper is that of "lean" and "robust" designing. "Lean" designs are those that develop with a discontinuous progression of ideas, with many large and abrupt changes made on the way to a solution. "Robust" designs are those that develop with a continuous smooth progression of ideas, with small refining changes made to the original design idea. The author relates these two types of designing to mathematical catastrophe theory (Lansdown cites [Tho.1]). Suggestion-making programs can be thought of as systems that propose alternate prototypes or alterations to prototypes. The resulting design process may be lean if the desiger gets many new ideas, or robust if the designer maintains and refines an original concept.

The first working system we will describe is reported by Murthy and Addanki [Mur.1]. Their PROMPT system modifies prototypes (there is no relation between these prototypes and the prototypes proposed by Lansdown), in order to obtain a problem solution. The system uses a graph of prototype models, with a prototype at each node. Changing the prototype is equivalent to traveling from node to node on the graph. As an example, consider designing a beam cross section. A possible prototype change is to change from a solid cross section to a tubular cross section. This prototype change might be made if the tubular cross section is better suited to the given load conditions. The authors describe "modification operators" that transform one prototype into another. This program changes protoypes autonomously (for a particular type of object, e.g. beams); it is hoped that suggestion-making programs will help the user to change prototypes in an effective way.

Dyer et al. report on a very ambitious project to develop an engineering design invention system that can create novel mechanical devices [Dye.1]. The EDISON program is meant to create innovative solutions in the way a human does. It uses naive physics, qualitative reasoning, planning, various discovery/invention heuristics, and a set of abstract devices that are organized and indexed in an episodic memory. To create an innovative design the system can employ one of several strategies. It might obtain a prior design that could serve as the solution, make some type of generalization or analogy, or mutate an existing abstract device. The implementation described by the authors is limited to the application of mutation heuristics that can manipulate and create doors. In proposing suggestion-making programs, we take a different approach. Instead of modeling what the designer does when creating innovative solutions, we attempt to devise a program that will help the designer be more innovative.

Ulrich and Seering describe two systems that automate the preliminary design of a particular type of object. In [Ulr.1] they describe a system that uses a combinato-

rial approach to design mechanical fasteners given a set of functional requirements. Five fastener subcomponents are found on seven different prototype fasteners. The prototypical fasteners exhibit structure-function relationships that are used to constrain the combinatorial search for solution fasteners. This limits the number of solutions to a reasonable number that meet the functional requirements. A deficiency of the system is that the structure-function relationships preclude some truly novel solutions. The authors suggest that this problem could be alleviated by observing structure-function relationships at a more detailed level: in other words to decrease the "grain size" of the observed relationships. This work has very little relation to programs that are meant to influence designers since the system completes the entire design.

A second effort designs objects in a simple planar blocks domain by using a design-debug strategy [Ulr.2]. The important concept is that the debug step is done from a number of *perspectives*, which are related to different design goals. An initial design is first created. New designs are then created with a debug step for each perspective. All of these designs, created from each debug step, are then tested to see if any satisfy the overall problem goals. If one is satisfactory, then the design process stops. If none are satisfactory, then one is picked to become the new initial design, and the process repeats. The important similarity to the approach we propose is the explicit separation of the perspectives. This allows the debug knowledge to be isolated in several distinct computational modules. Because it is dedicated to a single perspective, a module is probably relatively small and easy to encode and maintain. We similarly separate knowledge about various concurrent engineering perspectives, from knowledge about other concurrent engineering perspectives and from the user's knowledge of the design requirements.

Concurrent Engineering

In this section we discuss some of the literature related to concurrent engineering. Again, the categorization of certain works into this group is a matter of opinion. Some works described here are closely related to computer aided design (articles pertaining to the feature-based approach) and aritificial intelligence (articles about collaborative systems). Their inclusion is due to their relationship to the idea of considering several engineering aspects simultaneously.

Several brief publications can provide an introduction to topics in concurrent engineering [Eva.1, Dwi.1, Sto.1]. In particular, Stoll provides a useful review of several design for manufacturability approaches and compares their various qualities in [Sto.1].

Concurrent engineering has as its most direct ancestor the techniques of value analysis and engineering. Miles provides a thorough introduction with many case study examples [Mil.1]. The purpose of value analysis is to identify and eliminate unnecessary costs in a product. Once these costs are identified, alternative designs are proposed. This is usually not done during preliminary design, when designers are primarily concerned with meeting functional specifications. It is more likely attempted when the product idea is fairly well defined, so that concentration can be focused on reducing costs (i.e. increasing value). With a suggestion-making program, of course, we try to integrate the consideration of value into the preliminary design phase. This is done by inserting the "value knowledge" into the design process with suggestions.

The specific concurrent engineering aspect we have chosen to apply is design for assembly. Boothroyd and Dewhurst have created several useful design for assembly charts that can be found in a "designer's handbook" [Boo.1]. The Boothroyd-Dewhurst system provides the concurrent engineering knowledge for the intelligent CAD system that is developed here. These charts are used to rank a part to determine the ease with which it can be handled and assembled. The information on

these charts is based on theoretical and experimental studies, [Boo.2] for example. Thorough background information is provided in [Boo.3]. It should be noted that computer programs have been developed that aid in the use of the design for assembly charts [Dew.1]. These programs are really only computational representations of the charts: they do not access representations of designs to perform automatic analyses. Use of the Boothroyd-Dewhurst design for assembly system as a concurrent engineering design concern is discussed in more detail in Chapter Three. Other researchers have investigated problems related to design for assembly. Andreasen et al. consider the overall problem of design for assembly [Andr.1]. den Hamer discusses several different kinds of part feeding and orienting tracks and provides some guidelines [deH.1].

To use the Boothroyd-Dewhurst system, one must note the *features* on a part. This implies that a representation of the design should allow some representation of its features. Features can be thought of as meaningful geometric entities that are pertinent to some processing of the design. Most of the interest in part features has been related to the problem of automated process planning, for example the planning of required machining operations [Pra.1]. Others recognize a more general utility of part features [Bri.1]. The idea of *automated* process planning is to derive the required processing operations from a CAD representation of the part. Since the part features are related to the processing operations, it is necessary to determine the features of the part. This motivated attempts to automatically extract features from standard CAD model representations [Hen.1, Woo.1]. This proved to be a very difficult problem. To avoid an extraction of features, others have proposed designing with features from the outset [Dix.1]. In this approach, the user creates the CAD model with feature building blocks. The representation of the model will be organized around the features, thereby eliminating the need for any feature extraction. This is the approach taken in the system developed here. An additional benefit is that an

easy to use human-computer interface results.

Research in collaborative systems is another area that is related to concurrent engineering. As their name implies, these systems allow several users to collaborate on a problem or design. Wrona and Olszynski, for example, describe a system that causes an interaction between an architect and a non-architect during the design of a portion of a city [Wro.1]. Two computer support systems for group meeting situations are described by Stefik et al. [Ste.1]. One system, called COGNOTER, aids the group while they interact to outline a talk or paper. A second system, called ARGNOTER, helps the group conduct arguments concerning research proposals. Stults proposes to use videotechnology in design environments [Stu.1], particularly to provide a real-time connection between designers at different locations and to record the circumstances surrounding design decisions. All of these systems could have some application to a concurrent engineering approach because they facilitate the collaboration of several individuals. These individuals may possess expertise in different concurrent engineering areas. These systems differ from the suggestion-making systems we propose because they assume a human-human collaboration, whereas we assume a human-encoded expertise collaboration.

Design

The purpose of providing suggestions in the interactivity of Figure (1.1) is to help users get better design ideas. Other techniques found in the literature of general design theory and methodology have the same purpose and some of those techniques are briefly explained in this section. These techniques are grouped together here under the general heading of "Design" because they can easily be applied in a number of design situations. Their application is not limited to the preliminary design phase. Jones provides summaries of a wide variety of design methods, including those ex-

plained here [Jon.1]. The explanations below are based on these summaries, and the references are suggested by Jones.

Osborn describes the method of *brainstorming* [Osb.1]. In this technique, the process of developing new ideas is divided into two phases. A group of people are first gathered together for the purpose of finding ideas relevant to a problem at hand. The first phase is an idea generation phase where the participants offer any idea that may occur to them. Other participants may get new ideas from the ideas offered by others. Various idea-spurring questions may be posed. During this first phase it is important that none of the ideas are criticized. The second phase is used for evaluation and organization of the ideas. A similarity between this process and the suggestion-making process is that users may get new ideas from the suggestions output by the system.

Gordon proposes the method of *synectics,* which is another group participation idea generation technique [Gor.1]. A group that is devoted to this approach, gathers to produce ideas relevant to a certain problem. Various analogies are used to stimulate creativity. Participants might, for example, consider how some animal performs the functions required of the design they must create. This technique seems more "free form" than brainstorming, with more and varied interactivity between the participants.

Morphological analysis, discussed by Norris [Nor.1] and Zwicky [Zwi.1], can be done on an individual basis and is more formal than either of the preceding two approaches. In this technique, the necessary functions of a design are noted and possible subsolutions that will provide each function are proposed. A design is created by choosing one subsolution for each functional requirement. The total number of potential designs is therefore the product of the numbers of subsolutions for each function. This can be a very large number. If all possible subsolutions are noted, the technique ensures that no potential designs are overlooked.

All of these methods can be used to generate a large number of design ideas. Unlike the ideas output from a suggestion-making program, there is no indication of the quality of the ideas; the suggestions will be better designs with respect to the concurrent engineering concern. Additionally, with the techniques described above, the generation of ideas can be random and unfocused (particularly with brainstorming and synectics). A suggestion-making program will output design improvement ideas that are dependent on the user's preceding design steps and the encoded concurrent engineering design domain.

Computer-Aided Design

In recent years, there have been impressive advances in the capabilities of CAD systems. Some of these improvements are briefly discussed in this section.

One significant advance was the variational geometry design representation [Lig.1]. In this approach, designs are outwardly represented by a dimensioning scheme. The dimensions correspond to certain geometric constraints, such as distances between points and angles between lines. These constraints are represented with equations containing the variable dimension values. Given a set of dimension values, solution of the constraint equations specifies the location and orientation of the CAD primitives that make up the design model. If the set of dimensions specified would not produce a physically realizable object, the constraint equations can not be solved. This representation allows a "dimension-driven geometry" type of interface. The user makes a rough model of the design and then adds a dimensioning scheme to the model. Values for the dimensions are then input, and the system alters the model automatically to agree with the dimensional values. The commercial Cognition system has this capability [Cog.1].

Note that a variational geometry representation allows the efficient modeling of a family of parts: models of all parts that have the same dimensioning scheme

can be generated automatically by simply changing the dimensional values. This representation does not allow a variable topology. The numbers of faces, edges, vertices, and the connectedness between them does not change. Consider Figure (1.1) again. If the groove is moved far enough across the block face toward one edge, eventually the groove will become a step removed from the corner of the block. A variational geometry representation could not model this change because a block with a step and a block with a groove would require two different dimensioning schemes. Gossard et al. describe a model representation that allows changes in topology [Gos.1]. In this approach, dimensions are represented as "relative position operators" between CAD entities. These relative position operators and standard set operators then become the nodes of a graph that represents the entire model. When a dimension on the model is changed, the graph is reevaluated to produce a new model that may have a different topology.

A very different type of advanced CAD modeler is a *program model* system. With these systems, a design is modeled by writing a program. When the program is compiled or run, the design is "created." This design creation can cause many different types of output, such as a graphical rendering, a manufacturing process plan, or repair information. A design is altered by altering the program model. The commercial ICAD system [ICA.1] and a system described by Premack [Pre.1] are examples of this approach. This type of model is most useful for a design that is a complex prototype, such as a machine with many parts. If new designs are created by altering the prototype program, then all related information (e.g. process plans) can be generated automatically.

All these systems provide advanced techniques for *modeling* the design. None of them autonomously alter the design. In contrast, the system described in this thesis will actively alter the model to try to improve it with respect to a concurrent engineering aspect. In this way, the system we propose is very different from the

CAD modelers described above.

Artificial Intelligence

There has been a considerable amount of research on applying artificial intelligence techniques, especially expert systems, to problems of engineering design. Rychener provides an overview of the major issues [Ryc.1]. In this section we describe some systems that have been developed. The systems outlined below are distinct from the preliminary design systems discussed earlier because their emphasis is more on general design issues. We compare these systems with the idea of a suggestion-making program by noting the specificity of their application and the involvement they require from the user.

Brown and Chandrasekaran have developed a system that does routine design of air cylinders [Bro.1]. The user specifies some parameters of a pre-existing air cylinder configuration and the system completes the detail design of the device. The system is organized as a heirarchical community of design agents and completes the design in four phases: requirements checking, rough design, design, and redesign. The only interactivity with the user is the initial input of parameters. This system is very specific in that it can design only air cylinders with a known basic configuration. The involvement with the user is minimal. This system is clearly more of an automated routine designer than a design aid.

A system that is philosophically similar but seems to be somewhat more sophisticated is the PRIDE system described by Mittal et al. [Mit.1]. PRIDE designs pinch roll paper handling systems for photocopiers. The configuration of the design is variable to some degree (e.g. the system determines the number and location of roll stations in the handling device). The basic operation is that the user inputs the specifications and the system completes the design. The user can, however, change

the specifications at various stages in the design process. PRIDE can also critique designs and suggest modifications. The design problem is represented as a generalized design plan that will make design decisions going from the general to the specific. A graphical interactivity showing the paper path is provided. This allows the user to specify obstacles that might influence the paper path, and to partially construct a paper path that the system will help to complete. Details of the graphical interactivity are found in a report by Morjaria et al. [Mor.1]. This system only designs pinch roll paper handling systems with varying configurations. There is some cooperation possible between the system and the user during design, but the system's main purpose is the design of the paper handling system.

Another system that provides a graphical interface and cooperates with a designer during design is a kitchen design system described by Oxman and Gero [Oxm.1]. This system can fully complete a floor plan design, finish a partially created design, and analyze an existing design. This is a production rule expert system with a graphic display module that performs the transformations between representations used for the graphical display and representations used for the expert system fact base. The user interactivity includes simple digitizing of the graphical floor plan, and text input. This system is similar to the PRIDE system in that it can work with various amounts of user input, that is, it can fully design a kitchen or complete one the user has started. There is no direct attempt, however, to stimulate the users with alternate design ideas. Although only the development of a kitchen design system is reported, it is possible that this system could be extended to other types of rooms since all rooms can be represented with a floor plan layout.

DOMINIC, the final system that is reviewed, is intended to be a domain independent program for mechanical engineering design [How.1]. The system can design in almost any domain provided that the domain meets a few requirements. First, the designs must be represented as a list of variables. This implies some *a priori* knowl-

edge of the structure of the designed object. Dependencies between these variables and the satisfaction of various design goals must also be known. Additionally, there must be some means to measure the overall quality of the design as a function of the degree of satisfaction of the various design goals. The system operates in an evaluate-and-redesign cycle. An initial design is analyzed to determine if it is satisfactory. If it is not, a variable is modified to create a redesign, and the cycle starts again. The user inputs some design specifications and perhaps a starting proposed design. The system then searches for a satisfactory design. The system seems to provide a fairly general method that can be applied to many types of objects. It must be reconfigured for each new domain. The system has been configured to design V-belt drive systems [Dix.2] and heat fins [Kul.1]. The system performs as an automated designer.

To summarize, these systems are usually very specific to a certain type of object or system. This is because they can carry out the entire design process. The user is not purposely stimulated to get better design ideas, although it is possible for the user to get new ideas while using these systems. A suggestion-making program will divide the design concerns between the user and system. The user will design to meet the specifications and the system will respond based on concurrent engineering concerns. The system's actions will hopefully cause the designer to conceive new design solutions that would be overlooked without the system.

Human-Computer Interaction

This final category of reviewed research includes psychological and human-computer interaction studies that have some relation to the idea of a suggestion-making interface. Some of these efforts do not involve CAD systems or computers at all, but are reviewed because they are thought to provide useful findings.

Card, Moran, and Newell, as part of a larger study of various types of human-computer interactivity, studied the VLSI CAD task [Car.1]. Using a CAD program, a user designs the chip layout for an integrated circuit. The authors considered this to be a semi-creative task in the sense that the user often has created the idea of the circuit before using the CAD tool.[2] Less creative tasks, such as compressing the circuit to a minimum area and integrating the circuit with other subcircuits are performed while using the CAD system. The authors had developed a GOMS model (Goals, Operators, Methods, Selection rules) of various instruction following tasks, such as text editing, and they wanted to see how applicable the model was to a semi-creative task where the user followed no prestated instructions. They found that the semi-creative circuit design task was composed of a creative part and a routine part. The routine part was found to be similar to other instruction-following tasks they had studied. This type of result could be useful for the design of the detailed levels of a CAD interface, but it says very little about the predicted efficacy of an interface that makes suggestions.

Eberts et al. studied a different type of CAD interactivity in an attempt to reveal differences in expert and novice use of a CAD system [Ebe.1]. Here, the term "expert" refers to proficient use of the CAD tool, *not* general design skill. Expert and novice users were given the task of creating a model of an existing mechanical object with a typical CAD system. Note that this was not a design task: the object already existed. The users only had to model it. They found that expert users were proficient because they planned their modeling task around the functions of the CAD system. This allowed them to use the system effectively. The novice users, on the other hand, planned their task around logical subportions of the modeled object. This led to a very inefficient use of the CAD system that required significant backtracking. The authors suggest teaching the experts' "functional" strategy during training for the

---

[2] The single user that was studied *started* with a sketch of the circuit.

system. We would instead propose developing a CAD interface that accomodates the novices' "object-oriented" strategy. This could be accomplished by providing design primitives that are like typical subportions of the designs that will be modeled. In short, provide a feature-based interface.

Other researchers have attempted to study the cognitive processes of design. Ullman et al. performed a protocol analysis study of mechanical engineering designers [Ull.1]. In such a study (see also [Eri.1]), subjects are videotaped and asked to verbalize their thoughts as they solve a problem. These records are later examined to hopefully determine their problem solving strategy. In the experiment described in [Ull.1], there were two design problems and six subjects. One problem was a "one of a kind" design and the other was a part design that was intended for mass production. Four of the six subjects were experienced mechanical designers and the other two were inexperienced. Two experienced subjects and one inexperienced subject were assigned to each problem. The experienced subjects were assigned to the problem that was closest to their real-world experience. An important result found by the researchers is that designers tend to pursue a single conceptual design idea. This idea is chosen very early in the design effort, and is maintained and "patched," no matter how bad it proves to be. This is a very significant result because it reveals one reason why bad designs are created. We hope that a suggestion-making CAD interface will cause better initial ideas and promote the abandonment of bad ideas later in the design process.

Other researchers have used different techniques to study designer behavior. Waldron and Waldron, for example, studied the knowledge processing that occurred in a large-scale group design project that took place over a long period of time [Wal.1]. Because of the scale of the project and the time and number of people involved, protocol analysis techniques would have been impractical. They derived information about the design process from retrospective accounts and project records. In con-

trast with the work of Ullman et al. they found that in a group setting, different design ideas are often pursued in parallel, and they report that this was done even by an individual design team member. A study by Davies and Talbot also pertains to the issue of one versus many conceptual design ideas [Dav.1]. In interviews with award winning designers, they found that designers seem to intuitively know when an insightful idea will prove to be the design solution that they will develop. Their interviews indicate that many of the designers don't feel it is necessary to make any effort to confirm the quality of an idea that they *feel* is good. This suppports the single conceptual design idea view because it will naturally be difficult for designers to go against their intuition.

A final study that we will review is reported by Malhotra et al. [Mal.1]. The authors describe an observational study and two experimental studies that were performed. In the observational study, they videotaped client-designer dialogs and later analyzed the vidotapes. The *client* is the person who has a design problem that requires a solution. This problem is often poorly specified. The *designer* is the person who has some experience in a relevant design area, and is considered to be an expert consultant. An interesting finding was that during the period when the client is describing the design requirements to the designer, the designer will often prematurely suggest design solutions *before* all of the requirements are described. The authors feel that the designer does this in order to help the client clarify and elaborate his goals. They use this result to support the idea of a design aid for the goal elaboration process. This aid would output various designs in the domain of interest, allowing clients to familiarize themselves with the domain and obtain prototype design ideas that can be more fully considered. This seems like more than goal elaboration, since some idea generation is also involved ("goal elaboration" and "design generation" are two processes contained in a model of the design process proposed by the authors). The suggestion-making program we propose will behave

somewhat like an expert designer does in the design dialogs. It will not suggest solutions that do not meet the design requirements because it wants to clarify the requirements. It will never "understand" any of the requirements. Its suggestions will provide concurrent engineering information and will hopefully stimulate the client to obtain better design ideas.

One of the experimental studies described in [Mal.1] involved the design of a restaurant. A group of subjects, who were not architects, were given an assignment to design a restaurant that would be located in an old church building. They were given background information about the church building and the restaurant requirements. Half of the subjects also received a list of random words. When their designs were completed, the subjects communicated them to the investigators in any way they wished. One finding of this experiment was that the subjects with the random word lists produced designs that were considered to be more practical, but not more original,[3] than the designs of the other subjects. The authors hypothesize that the words served as memory cues that brought more items from the user's memory into contact with the design problem. They suggest the creation of an unstructured design aid that will provide memory cues during design. The researchers also determined a list of fundamental functional requirements for the restaurant before the experiment. They found that no subject fulfilled more than 70% of these functional requirements, but that all requirements were fullfilled by some subject. From this they propose the creation of a collaboration aid for designers that would be especially useful if the requirements of the design can be partitioned into a few general categories. This idea is very much like the suggestion-making programs we have proposed. Concurrent engineering requirements are handled by the system and other requirements are handled by the user. The system's input to the design process cannot be considered

---

[3] Refer to [Mal.1] for more information about measuring practicality and originality.

as unstructured as a list of random words. After all, the system is motivated by the concurrent engineering domain. It is fair to say, however, that the suggestions are unstructured in the sense that they do not take into account the user's design goals.

Conclusion

To conclude this chapter, it is useful to reiterate the goals of this research project and to summarize how these goals compare with accomplishments described in the literature. The idea of a suggestion-making CAD system should be put in perspective.

The fundamental goal is to provide useful information to designers during the preliminary design process. This information, related to some concurrent engineering aspect, is normally not considered during preliminary design. The system provides this information by autonomously altering the design in response to a user action. The system providing this information will be unable to complete a design independently of the user, as it has no representation of the design requirements. It is hoped that by providing this information, the designer will obtain better design ideas.

Unlike most other AI-based design systems, this system is independent of what is being designed. Its capabilitites can be applied to any object that is pertinent to the encoded concurrent engineering knowledge. Unlike other CAD systems, this system can alter a design in response to a user design step. Unlike most tutoring and advice giving systems, this system does not try to teach and makes no effort to determine the user's intentions. The system's interactivity with the user is somewhat unstructured because the user's design goals are not taken into account. On the other hand, the interactivity is not completely unstructured because it is dependent upon a design knowledge domain and a representation of the design.

CHAPTER III

DESIGN FOR ASSEMBLY

Introduction

In this chapter we describe design for assembly as a design analysis domain. The ideas presented here are pervasive and implicit throughout the remainder of the thesis, as design for assembly information is the sole information encoded to date in the intelligent CAD system described in this thesis. Design for assembly principles are most commonly used for analysis. A part (or assembly of parts) is analyzed to determine the suitability of the part for automated assembly. This information then serves as a starting point for a redesign effort. The intelligent CAD system will instead integrate the principles into the design process, noting bad design decisions and suggesting improvements with respect to design for assembly. Other design analysis domains are possible: design for assembly demonstrates the concept of an intelligent suggestive CAD system.

All the encoded information is derived from the Boothroyd-Dewhurst Charts [Boo.1]. The data on these charts represent the compilation of much empirical and theoretical research (see for example [Boo.2]). Background information on the general area of automated assembly is available in [Boo.3]. Other researchers [Andr.1, deH.1]

have investigated assembly topics but have not derived tables and charts useful for the design task.

Indeed, this is the primary reason the Boothroyd-Dewhurst Charts were chosen as the knowledge source. The design for assembly charts provide a clear and direct way to quantify the suitability of a part for automated assembly. They can, therefore, be used to unambiguously compare two designs with respect to assembly considerations. Additionally, parts can be considered as single entities. The relationship of a part to other parts in an assembly has no bearing on how that single part will behave in an orienting machine, which is our specific interest. This is not to say, of course, that design considerations related to an assembly of many parts are not important. Boothroyd discusses the economic importance of reducing the number of parts in an assembly through elimination and integration [Boo.4]. Other researchers have addressed the difficult problems associated with the design of assemblies. For example, Kroll et al. apply AI techniques to the problem of integration and elimination of parts in an assembly [Kro.1]. Here, we are designing single parts only in the context of their suitability for orienting machinery. Finally, the charts are well suited to an intelligent CAD implementation because they can yield sensible design suggestions that are not obvious to most people. Very few people are familiar with design for assembly ideas.

Design for Vibratory Bowl Feeding

Specifically, the encoded knowledge concerns designing rectangular shaped parts (as opposed to rotational parts) to facilitate the use of vibratory bowl feeders. These devices accept a disorganized bulk of parts and orient them with a series of mechanical filters and orienting devices. The filters and devices are located on a helical track that accomodates a single-file line of parts. An oscillating electromagnetic field vibrates the entire feeder and causes the parts to climb the helix. The filters are used to reject

improperly oriented parts back into the bulk. As the parts move, they slide against the filters which sense their orientation and force them off the track if necessary. The orienting devices, on the other hand, cause a part to change orientations. The parts are reoriented as they slide against the device. Some combination of filters and orienting devices is used to deliver parts in the proper orientation. See [Boo.3] and [deH.1] for example feeder tracks with filters and orienting devices.

The objective is to design the part so as to minimize the cost of engineering and implementing the bowl feeder. The fundamental problem is to orient the parts with certainty and still provide an adequate delivery rate. Since filtering slows the delivery rate by rejecting parts from the track, it is useful to design the part to minimize the amount of filtering required. This can be done by making the part symmetric about one of the axes.[1] Since symmetric parts have different orientations that are identical, less filtering is required. Another useful idea is to make orienting the part very easy. This can be done by making the features of the part very pronounced. Distinct, large features and dimensions facilitate the interaction of the part with the orienting devices.

Figure (3.1) illustrates some of these ideas. Figure (3.1a) shows a very good example part for assembly considerations. Note that the part is 180-degree symmetric about all three axes and the overall dimensions of the part are distinct. This part requires very little orientation and the orientation that is required is facilitated by the distinct side lengths of the original block. Figure (3.1b) shows a part that is much worse than the one shown in Figure (3.1a). The part has no symmetry and two of

---

[1] In the Boothroyd-Dewhurst system, rectangular parts are considered to be symmetric about an axis if they repeat their orientation after a 180-degree rotation about that axis. Parts that repeat their orientation after a smaller angular rotation are considered to be rotational parts. Note that these axes are attached to the part. The X axis is aligned with the longest overall dimension, the Y axis is aligned with the midlength overall dimension, and the Z axis is aligned with the shortest overall dimension.

the overall dimensions are not distinct. There is, however, a very prominent feature that can interact with the filtering devices. Note how orientation of the grooved projection which is parallel to the Z axis orients the entire part. Figure (3.1c) shows a part that is very bad. Although the part appears to have some useful symmetry, note that two pairs of features are symmetric about two different axes (the steps about the Z axis and the holes about the Y axis). Orienting the part with with the steps facing up, for example, does not unambiguously orient the part because the location of the holes is still not known. Both feature pairs must interact with filters and orienting devices.

In the Boothroyd-Dewhurst system, characteristic part shapes are related to the ease of feeding and orienting. These relationships are recorded on matrix-type charts. The indices of the matrix are text descriptions of the part shape and the matrix element specified by the indices is a numerical score of the ease of feeding and orienting. The charts are intended to be used in an analysis mode. A part is designed, ranked with the charts, redesigned, reranked, and so on until the designer is satisfied. The charts themselves are discussed in more detail in the descriptions of the previous suggestion-program efforts found in the next chapter.

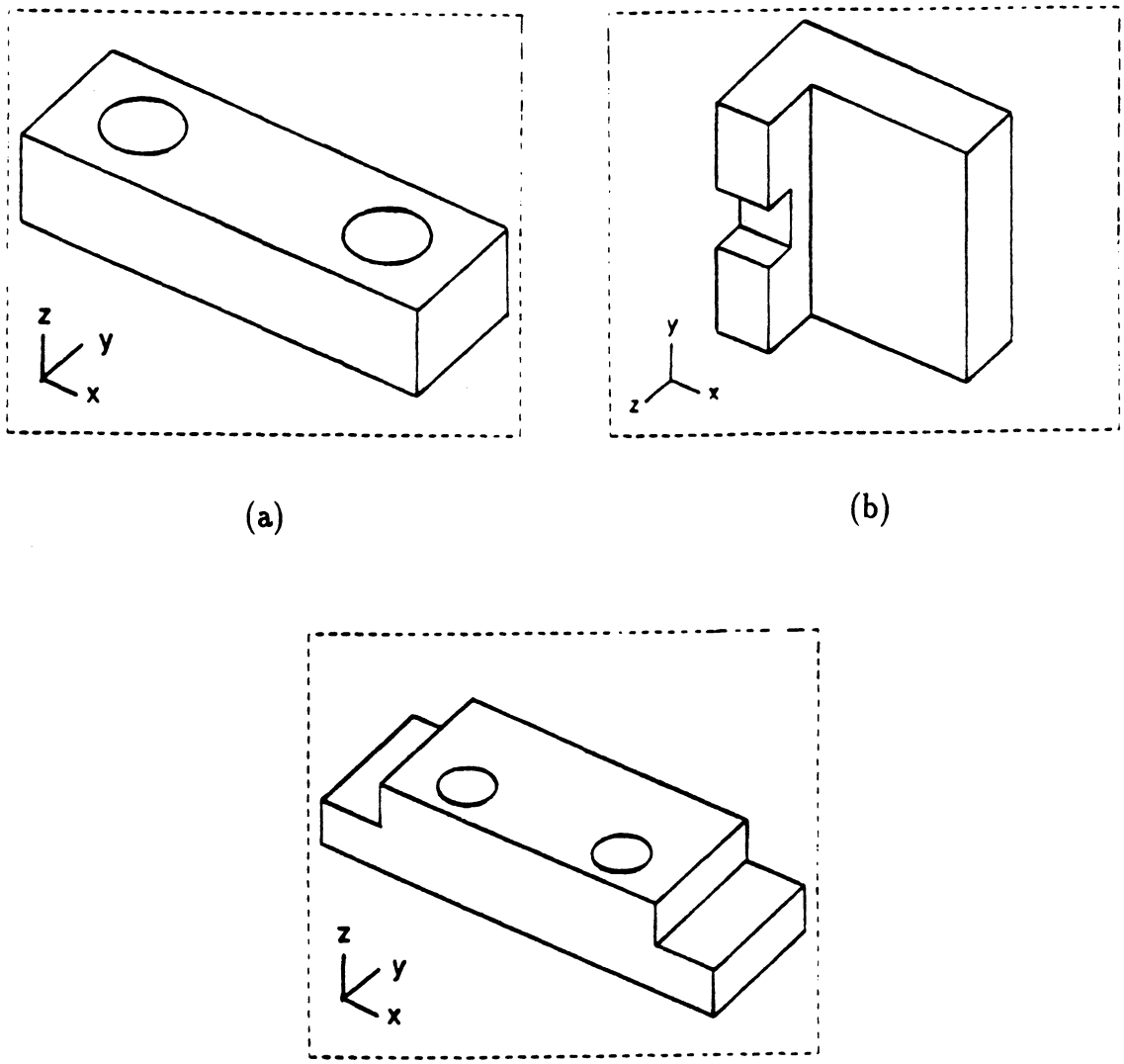


Figure 3.1: Various parts that might be bowl fed. (a) Highly suitable for bowl feeding. (b) Difficult to bowl feed. (c) Very difficult to bowl feed.

CHAPTER IV

SUGGESTIVE SYSTEMS: PREVIOUS EFFORTS

Introduction

This chapter describes previous suggestion-making systems developed by the author. The information encoded into these systems was exclusively design for assembly information. Each major section below describes one of the previous efforts, and begins with a summarizing introductory subsection. This is provided for the reader who seeks only an overview of each system. The remaining detailed description, however, is essential for understanding the research evolution that led to the intelligent CAD system described in this thesis.

Text Suggestions[1]

Introduction

[1] This section is derived from [Jak.1]. Figures (4.1) through (4.7) are also from [Jak.1].

An initial system makes text suggestions to prompt users to design parts that will be ranked more favorably on the Boothroyd-Dewhurst Charts. The inherent structure of the information on the charts was determined and represented with a ranked directed graph structure. At each node of the graph is a group of text suggestions that are output to the user. The system travels a path through this graph, outputting the suggestions at the nodes. The user responds yes or no to a suggestion, thereby determining which subsequent path of the graph to travel to the next node. The graph is structured such that the more suggestions that are rejected, the more the design quality decreases. The user is expected to incorporate the intent of the accepted suggestions into the evolving design, which they are sketching offline. The system has no representation of the evolving design; only the information contained on the Boothroyd-Dewhurst Charts. When several graphs are traversed in this way, the design is adequately specified to determine a cost estimate. Analysis information is thus transformed into suggestive information. It was hoped that the system would aid users unfamiliar with design for feeding and orienting in creating better initial design configurations.

The reactions of initial users, however

miliar with the Boothroyd-Dewhurst system had difficulty understanding the meaning of the brief text suggestions, even though there were clear references back to the charts. After using the system for only a short time, many of them recommended outputting the suggestions as pictures accompanied by liberal explanation and "help" files. This led to the second effort described below ("Text and Pictorial Suggestions").

The emphasis in this first system is specific to the handling of the data used for the suggestions. After a short description of the Boothroyd-Dewhurst data as presented in the design charts, we will explore how the data can be structured to derive suggestion rules for proper interaction with the designer. The actual program-

ming implementation will then be presented with an example. Some experiences of running the program by several users will also be discussed.

A final note should be made here about the programming language. For proper expert systems that are designed to grow and accumulate large amounts of knowledge it is appropriate to use Lisp or an expert system package. For small-size systems that are generally static, a language like FORTRAN or Pascal is quite adequate. In the application presented here, Pascal was used.

The Boothroyd-Dewhurst Data

The classification system proposed by Boothroyd and his co-workers consists mainly of a set of charts which attempt to quantify design characteristics of parts from the assembly viewpoint [Boo.1]. The design features examined are generally of a geometric nature, such as various symmetries, dimensional proportions and presence of grooves or flats. Other qualities such as rigidity, surface type (sticky or not) and complex topology (nesting or tangling when handled in bulk), are included as well. These design features are associated with numerical values assigned to four basic parameters: orienting efficiency (OE), relative feeder cost (FC), additional feeder cost (DC), and relative workhead cost (WC). The latter cost is associated with insertion procedures and means of securing the part in the assembly.

The relation between design properties and numerical values for the parameters is presented by dual-entry table charts. To use the charts, one must first assign a numerical value to a "digit" associated with a particular design feature. For example, the first digit can take values 0, 1 or 2 for a rotational part with ratio of two principal dimensions length/diameter in the ranges (0, 0.8), (0.8, 1.5) and (1.5, Infinity) respectively. Assignment of values to the first three digits leads to a specification of an OE-FC pair. The additional feeder cost DC value is specified by assigning values to the fourth and fifth digits. Two more numbers, referred to here as the sixth and

seventh digits (though Boothroyd and his coworkers do not use these terms), are needed to specify the relative insertion machine cost WC.

The assignment of numerical values to all digits gives a "digit code" which specifies values for all the foregoing parameters. With some additional information, such as feed rates, maximum part dimension and number of simultaneous assemblies, the entire assembly cost per part is calculated based on a cost function expression.

It is evident that the optimal design according to this system is the one having maximum OE and minimum FC, DC, and WC. It should be noted, however, that when maximum part dimension dictates a feed rate greater than the user-required feed rate, OE is not used in the cost-function expression. This is a result of under-utilization of the feeding equipment and details can be found in [Boo.1]. The typical use of the charts is an *analysis* of an existing design in order to evaluate it. This can be done in an iterative process, i.e. create a design, rank it according to the charts, redesign, and repeat. Decisions about what to change during redesign are based on familiarity with the charts and some trends implied by them. This will be discussed further in a later subsection.

The procedure of using the charts has been implemented on a microcomputer by Dewhurst and Boothroyd [Dew.1]. This aids the designer to iterate more efficiently on a design, but the main philosophy on the use of the data remains the same, namely as an analysis tool. In the next two subsections we shall discuss how the same data can be used in a suggestive mode as proposed earlier. A full appreciation of the discussion there will require more familiarity with the charts than what was summarized above and the interested reader is urged to consult the cited reference [Boo.1]. It is important to note that the data restructurings described here are very different from the production rule modeling described in Chapter Eight. Also this initial system uses all five of the Boothroyd-Dewhurst Charts, while the intelligent CAD system described in this thesis uses only two of them, those that determine

an OE-FC pair for nonrotational parts. This gives some indication of the added complexity resulting from the integration of an intelligent suggestive program and a CAD modeler.

Binary Tree Representation

The most natural way to represent the data in the charts appears to be one which will aid in assigning one digit at a time. This is amplified by the apparent fact that the digit indexing corresponds to design feature generality; i.e., specification of a first digit value is based on a more general design configuration decision than what is required for the second digit, and so on. This is not strictly true, however, and could be misleading.

Examination of a typical chart, shows that better designs, e.g. better OE-FC pairs, are found in the upper left region of the chart, while less desirable designs are found in the lower right of the chart. So design quality generally decreases along the diagonal. It would then seem appropriate to structure the design property suggestions so that the design is placed as close as possible to the upper left portion of the chart. A natural way to effect this is to recast the row and column numbers (corresponding to digit values) in a binary tree structure.

Such a structure is shown in Figure (4.1). The terminating branches on the left side of the tree correspond to regions near the upper left "optimal" region of the chart. They are accessed by an affirmitive response to the suggested design feature. Any branching to the right will generally suggest successively inferior designs. Note that in the lower right corner of the tree, a common node exists; therefore, a more efficient representation of a directed graph rather than a tree is used there. This is not particularly significant here because of the small size of the tree, but it could be more important in larger problems.

Once the first three digits are determined with binary trees, an OE-FC database is accessed and the values for the parameters are determined. In a similar manner, the fourth and fifth digit determination will lead to a DC database and the sixth and seventh to a WC one.

The binary tree representation appears to be a good way to organize the program in the natural fashion of the charts' organization. There is, however, a major problem. Although in general better design properties are found in the upper left portion of the charts, this is not an absolute rule. In fact there are too many exceptions to make this method practicable. What appeared to be a good decision when assigning the first or second digit, may turn out to be a poor decision when the third digit is assigned. In the present case, the number of possibilities is finite, so a backtracking strategy could be implemented to include better suggestions introduced in the first search. Proper bookkeeping could check for possible "wrong turns" after the design is created and if any were found, they could be brought to the user's attention. Then partial or full redesign would have to be initiated. This approach is plausible due to the small size of the database, but still it is psychologically unattractive because the user would have the feeling of possibly wasting a lot of thinking for solutions that would be discarded after she has completed them.

For the aformentioned reasons the binary tree structure was deemed inadequate and a different representation was sought. This is discussed in the next subsection.

Ranked Directed Graph Representation

The new structure investigated takes advantage of the fact that the state space has a relatively small number of states which can be explicitly enumerated. Recall that the first three digits specify an OE-FC pair, the fourth and fifth specify DC and the sixth and seventh specify WC. Thus we have a triplet and two couplets in the correspondence

(1st, 2nd, 3rd)	(OE, FC)	
(4th, 5th)	DC	(1)
(6th, 7th)	WC	

Furthermore, the goal state is described by the set

maximum	OE	
minimum	FC	
minimum	DC	(2)
minimum	WC	

That is, we have multiple objectives which, however, are ranked in the sense that preference is given to better values for OE than FC.[2] We assume that OE does not drop from the cost function expression (i.e. the feeders are fully utilized) in order to rank designs with respect to OE, a parameter which quantifies part shape rather than economic use of feeders. If OE is dropped, the program may yield false solutions. It should be noted that there are no explicit constraints, but the designer, by responding to the suggestions, implicitly introduces or removes constraints. Thus the goal state (2) is optimal when the constraints allow for it to be reached. If this does not happen, the goal state will not be (2) but another one which can be considered as a constrained optimum, or better as a *satisfying* solution [Sim.1, Wil.1].

A characteristic of the solution space is that it has more than one element. For example, the best triplet will assign (OE, FC) = (0.9, 1.0) for rotational parts, but there are several triplets that can do that, e.g. (2, 0, 0), (2, 1, 5), (2, 0, 5), (2, 1, 0). This leads to including all triplet sets with the same quality, i.e. (OE,

---

[2] In the encoding of information described in Chapter Eight, we instead minimize the ratio FC/OE.

FC) values, into one group. The groups are then ranked according to their quality. Figure (4.2) shows the first two such groups for the rotational triplets. The grouping is done once manually for both rotational and nonrotational parts and also for the couplets corresponding to DC and WC values. Thus the entire database is completely structured and ranked into these groups. The groups are sequentially numbered, the first one having the best (OE, FC) values.

It is important to recognize how the values of (OE, FC) can be achieved within a group. Each group contains a list of states which are connected with OR statements. As mentioned earlier, all the group states have the same (OE, FC) values. Each state now contains three design characteristics connected with AND statements. These three characteristics correspond to specific values of the first three digits. Since each state may often differ from another only by the value of one digit, most states in one group will have similar design features.

To illustrate the aforementioned ideas, let us consider Group 1 in Figure (4.2) and examine the program's interaction with the user. At the point when Group 1 becomes relevant, a decision has already been made that the part will be rotational. Now the program first suggests that the part have the general shape of a long cylinder, a property related to the first digit. If the designer responds that this is possible, the program next suggests, as a single statement, that the part be alpha-symmetric *or* slot-fed with its center of mass below supporting surfaces, properties related to the second digit. Finally, if the designer agrees to this, the program suggests as a single statement that the part be beta-symmetric *or* have a beta-asymmetric groove or flat seen in end view, properties related to the third digit. If the designer agrees again, membership of the design in Group 1 has been determined and the search stops.

The next question to be addressed is how to link the groups together. This is done by a directed graph representation where the groups are examined in a heirarchical serial manner. An example of a typical suggestion structure is shown in Figure (4.3).

Assume that the program is considering a ranking within Group 1 by suggesting that the part be a long cylinder. If the user responds negatively, there is no reason to consider Group 2, since all the states (triplets) in that group have "long cylinder" as the first digit property. Thus the program proceeds to consider the less desirable Group 3 having "short cylinder" as the first digit property. Note that, while still in Group 1, if the user accepts the first digit properties and rejects the second digit properties, then Group 2 will again be bypassed. Only if the third digit property suggestions are rejected will the program consider Group 2.

It is evident now that Figure (4.3) represents essentially an entire rearrangment of the Boothroyd-Dewhurst Chart data. The columns represent different values of the first digit, while the rows represents different states of the same quality, in decreasing quality ranking. Thus, in Figure (4.3) the Groups 3 and 4 have the same quality but different First Digit property, i.e. "short cylinder" versus "disc." If any of the Group 3 suggestions are rejected, consideration of Group 4 begins. If that group is also rejected, the program proceeds to Group 5, a new "long cylinder" group. Thus rejection of the "long cylinder" suggestion in Group 1 does not preclude its consideration later in the search.

At this point an undesirable situation seems to occur because a rejected property is reintroduced. To understand why this is justified, one may first notice that in the discussion of the Group 1 above, the two second digit properties and the two third digit properties were suggested *independently.* This is because all four combinations of these properties are present in this best group. If they were not, after the Third Digit suggestions the program would present a suggestion of the form

$$\begin{array}{ll} & (\text{Second Digit AND Third Digit}) \\ \text{OR} & (\text{Second Digit AND Third Digit}) \end{array} \qquad (3)$$

in order to present all possible combinations of the group. Thus the First Digit

property remains constant within a Group, avoiding the need for a rather unwieldy suggestion of the form:

(First Digit AND Second Digit AND Third Digit)
OR (First Digit AND Second Digit AND Third Digit) (4)

The result is that, as the user tries to evolve and rank the design (in this first group), she has to respond to only three program suggestions. The serial heirarchy described above allows eliminating consideration of a group with a minimum number of suggestions. Moreover, it utilizes the idea that the user should never be given the same suggestion consecutively. Repeating the same suggestion was considered likely to create a degree of frustration and to stifle creativity. If during a search, a rejected property is reintroduced, as in the aforementioned description of Figure (4.3), at a lower value, the designer can rethink the whole situation and perhaps simply restart the search with some fresh ideas.

The serial heirarchy structure of the data described above is summarized as follows. The OE-FC triplets and DC and WC couplets are ranked in the ordered groups. These listings are used by the program for making suggestions to the user about the design being created. The user accepts or rejects the suggestions and synthesizes the design accordingly. When enough suggestions have been accepted, the program assigns the pertinent cost parameters from the set of values for OE, FC, DC, and WC. When all parameters have been assigned, the Boothroyd-Dewhurst cost calculations for the part are performed and presented to the user.

Program Implementation

The ranked directed graph representation was coded in an interactive program written in the OMSI Pascal language on a DEC 11/34 computer using the RSX/11M

operating system. As mentioned in the introduction above, Pascal was selected as the programming language because of its good structure, and the small size of the database. A sample run of the program is shown in Figures (4.4) through (4.7). Figure (4.4) shows the suggestions pertaining to assigning the first three digits, Figure (4.5) shows the suggestions pertaining to assigning the fourth and fifth digits, and Figure (4.6) shows the suggestions pertaining to assigning the sixth and seventh digits. Figure (4.7) shows the results of the session.

The various functions of the program are initiated through responses to a menu. The digits are assigned in the sequence (1st, 2nd, 3rd), (4th, 5th), and (6th, 7th). Each part property suggested is preceded by an ordered pair of two integers within parentheses. The first integer refers to the Boothroyd-Dewhurst digit place and the second integer refers to the number suggested for that place by the associated property. For example, (1,2) refers to assigning "2" as the suggested value of the first digit. Thus an easy reference to the original Boothroyd-Dewhurst Charts is maintained. Numbers larger than nine in these parentheses reflect the fact that some of the Boothroyd-Dewhurst Charts are divided into horizontal blocks, numbered in the charts themselves identically across the horizontal blocks. For instance, (7,19) refers to Chart number 8 in [Boo.1] and means "column number 9" in the middle block. A complete description of these details is not important here. The interested reader could easily understand them by comparing the program with the actual data in [Boo.1].

In a typical design interaction, the user will sit with a sketch pad or a graphics tablet and try to visualize and sketch design configurations as prompted by the program suggestions. To avoid excessive bookkeeping in the program, the user must record and later input the specific Boothroyd-Dewhurst digit values corresponding to an accepted property within a group, because of the inherent ambiguity in an affirmative response to an ... OR ... OR ... OR ... statement. Finally, the menu

allows redoing many choices easily and recalculating the cost results reflecting the changes.

Discussion

A first observation is that, although the entire programming effort is aimed at a synthesis aid, analysis and redesign can be effected using the program in a very straightforward and efficient manner. Note, however, that no concern was given to the elimination of constituent parts of an assembly, or the choice of assembly machinery, issues extensively discussed in [Boo.1]. The goal here has been to demonstrate the feasibility of constructing *suggestion* programs and examining the implications with respect to how existing data are viewed, structured, or manipulated.

There were several principal ideas that were adhered to for the development of the program.

1. The suggestions are made in a truly optimal way, i.e. positive response guarantees the best possible design.

2. Decisions are reached with a minimum number of suggestions.

3. No single suggestion is repeated consecutively.

4. Data organized for design analysis may not be appropriately organized for design synthesis.

Initial informal observations of several users (mechanical engineering senior students) indicate that there are some drawbacks in the program interface. It appeared that the triplet-couplet-couplet suggestion routine was rather irritating. Many commented that they would like to assign one digit at a time, as would be done with the binary tree structure. The reappearance of the same part suggestion, even much

later than the previous time, was also considered irritating. Also, some users would answer the questions without much thinking and complete the design study without having a real design concept outlined. Most important, though, was the need for graphical suggestions. Many users reported that they had trouble envisioning the meaning of the brief text suggestions and recommended that the suggestions be output as pictures along with more complete explanations. These last two observations clearly indicate the need for a system that integrates a graphical design model with the suggestive knowledge.

These observations can be viewed appropriately if we mention that the users had only two hours instruction in both the Boothroyd-Dewhurst design for assembly charts and the program–time clearly too limited. As the motivation behind the program was explained in some detail, the suggestion routine was accepted more readily. It was clearly evident, though, that the program did influence the decisions made during the design synthesis process. The negative attitudes observed may have been a result of poor user preparation rather than the program itself.

Text and Pictorial Suggestions

Introduction

The user recommendations to provide graphical suggestions along with complete text explanations were implemented in a second system. A simplified design knowledge base derived from the binary tree restructuring described above was used. This system ran on two side-by-side computer terminals, one outputting the text and the other outputting the corresponding picture. Note that the graphics of this improved system were only pictures and not models. The system still had no representation of the evolving design and users still had to sketch their evolving design idea. Again,

users were informally observed interacting with the system. Designers found the text and pictorial suggestions helpful and easy to understand. Most users were influenced by the suggestions, but a minority seemed to ignore the suggestions and pursue their own design ideas. These observations seemed to indicate that a graphical suggestive mode program was feasible. The fundamental technical problem was integrating design model representations with design knowledge representations.

The emphasis of this second effort was determining the influence and utilty of graphical suggestions rather than reorganizing data. The goal was to determine if the picture suggestions made any difference in the design process. First, we provide a brief description of the suggestive system developed for this second effort. The suggestive knowledge base is then described. Finally, observations of users are discussed.

The Suggestive System

As mentioned above, the suggestive system ran on two side-by-side computer terminals, one outputting the text and the other outputting the pictorial graphics. The text terminal provided all instructions to the user along with the textual portion of the suggestions. The instructions included commands the user would type at the graphics terminal that would output the picture corresponding to the particular text suggestion. Unfortunately, the hardware and software used prevented any inter-system communication. The users, therefore, would first read the text suggestion, then view the picture, and finally decide if they could work the idea of the suggestion into their design plans.

The program outputting the text suggestions was menu driven and required that the user complete three design phases. Completing a phase would assign one of the Boothroyd-Dewhurst digits for the design: the first phase would assign the first digit, the second phase the second digit, and the third phase the third digit. Therefore,

successfully completing a design session would assign the first three digits, thereby specifying values for the parameters OE and FC. Suggestions related to assigning the parameters DC and WC were not considered here. The suggestions were structured and output as lists, as opposed to tree or ranked directed graph structures. In the list structure used, a text suggestion was placed at each node of the list with a single link pointing to the node from a more desirable suggestion, and with a single link pointing from the node to a less desirable suggestion. In this way, the suggestions were prioritized from best to worst. The users would traverse the list by reading the suggestion and responding yes or no. A "yes" response would terminate the traversal since a particular suggestion had been chosen. A "no" response would cause the next best suggestion to be output. The rationale and derivation of this simplified suggestion structure is described in the next subsection.

The Suggestions

We will describe the derivation first. The suggestions output by this system pertain to rotational parts. As mentioned above, suggestions are made regarding Digits 1, 2, and 3, allowing specification of an OE-FC pair. The list structure of the text suggestions is a simplification of the binary tree structure considered in the first suggestive system effort. Consider, for example, the tree shown in Figure (4.1) since it contains text descriptions for rotational parts. The difference between a list and tree data structure is that a list structure requires only one accepted suggestion to specify a Boothroyd-Dewhurst digit, while a tree may require many. In Figure (4.1) note how it takes two affirmative responses in order to specify Digit 2 = 3. The suggestions are "... BETA symmetric grooves holes or recesses," and "... on both side and end surfaces." For a list structure, one affirmative response to the suggestion "... BETA symmetric grooves holes or recesses on both side and end surfaces" would

be required. The suggestions for the list structure are therefore more detailed and specific.

The suggestions are prioritized on these lists by prioritizing the values chosen for the Boothroyd-Dewhurst digits rather than using the OE and FC values. Digit 1 is considered most important, with a value of 2 (long cylinder basic shape) being better than a value of 0 (disk basic shape), which is in turn better than a value of 1 (short cylinder basic shape). Digit 2, which pertains to symmetries about an axis perpendicular to the rotational axis, is next most important. Digit 3, which pertains to symmetries about the rotational axis is least important of the three. For Digits 2 and 3, lower values are considered better. This means that a design with digits (2,0,5) is better than a design with digits (2,5,4), which is in turn better than a design with digits (0,0,0).

The only motivation for this was simplicity. Our interest was in observing the effect of suggestions on designers rather than measuring the true Boothroyd-Dewhurst quality of the created designs. This structuring of suggestions was straightforward and easy. It was only necessary that the suggestions not seem ridiculous to the designers, which certainly seemed to be true.

Figures (4.8) through (4.13) show a sample run of the program. Figure (4.8) shows the introductory text of the program. Figures (4.9), (4.10), and (4.11) show respectively sample text suggestions for phase 1, phase 2, and phase 3. Figure (4.12) shows the conclusion of the design session. Figure (4.13) shows typical pictures that would appear on the graphics terminal.

Observations of Users

The users were senior mechanical engineering design students. Because they were introduced to design for assembly ideas as part of one of their courses, they had a casual interest in using the program.

Most of the users were influenced by the program. Some commented that a particular suggestion changed their idea about the design they were working on. One user even reported that a conscious effort was made to think only in terms of design function rather than form, allowing the suggestions to provide ideas for the best possible shapes to meet the design requirements.

Other users said that they were not influenced by the program. Quite simply, use of the program did not change their original idea about the design. Some commented that they looked for a suggestion that matched their original idea. Others said the suggestions provided alternatives that were still overruled by the original idea.

In summary, it was clear that the majority of users were favorably influenced by the program. A graphical suggestive mode program was a feasible idea. The next step would be to integrate the suggestive program with a graphical representation of the design. Some of the users hinted at this. One felt that it would be more difficult to consider the meaning of the pictures when working on a more complex design. Another commented that it was difficult to formulate design modification ideas in response to a picture suggestion. An *intelligent* suggestive system with graphical modeling capabilities could formulate some ideas automatically and output them graphically on the evolving design.

Expert System Implementation[3]

Introduction

A limitation of these early systems was that the design aid was structured in a static way. There was no capability for changing the information used or the

---

[3] The system described here is also described in [Jak.2].

sequencing of suggestions without major reprogramming. In order to gain flexibility and allow the user to change the system easily, the data were restructured again and then encoded in a commercial expert system shell (operating with text only) with a control strategy that generated suggestions. Thus the ability to expand and change the knowledge base and the sequence of rule firing was significantly improved. The suggestive mode naturally requires forward-chaining reasoning, while the commercial shell used employs backward chaining. This made the software implementation quite awkward, although this was transparent to the user. The difficulty of not interfacing with graphic representations of the design, however, was not resolved.

The emphasis in this section is on the implementation of a suggestive system with the Teknowledge M.1 expert system shell [Tek.1]. The discussion below begins with a description of a broadly applicable set of suggestions that was formulated. Following this is a brief discussion of how the expert system was used to generate and output suggestions. A short summary closes the section.

The Suggestions

The suggestions result from a drastic simplification of all but one of the Boothroyd-Dewhurst design for assembly charts. Recall that coding a part with the charts results in values for the parameters OE, FC, DC and WC. WC relates to how the part is inserted into an assembly with other parts and therefore depends on the shapes of the other parts and the fastening technique. The other three parameters can be determined by considering *only* the part in question. In this system we did not want to consider parts in the broader context of the entire assembly, so we did not consider the chart that determines a value for WC. The other charts contain some very general design rules that may not be apparent to a novice user. Although there are a significant number of exceptions, following these rules is usually good design practice. These rules were organized into suggestive rules that were encoded.

The idea was that this suggestive system could be used as a preprocessor to the Boothroyd-Dewhurst system. General design principles could be checked before using the charts to quantify the design quality.

The goal was to extract these design rules and encode them in a commercial expert system shell. A suggestive decision graph that represents the general design for assembly ideas was created (see [Jak.2]). The suggestions are at the nodes and are output as a path through the graph is traveled. As in the binary decision tree, better choices are to the left, and affirmative responses cause the traversal to take the left link.

Values are assigned to variables as the graph is traversed. These variables are used by the expert system shell to output the suggestions. First a *basic-shape* is chosen, which is either rotational or rectangular. Once the basic shape is known, it is assigned a size which is called the *enclosure*. Rotational parts can have a long cylinder, disc, or short cylinder enclosure. Rectangular parts can have a long, flat, or cubic enclosure. Four features are then specified, which will be referred to as *feature-1, feature-2, feature-3, and feature-4*. Feature-1 for a rotational part relates to its symmetry about an axis perpendicular to the rotational axis. It is best if it has this symmetry (called "alpha" symmetry), and if not, the best asymmetric shape is similar to a machine screw: the part could ride on two parallel rails with its center of mass hanging below the rails. Feature-1 for rectangular parts specifies the part's symmetry. Either the part is symmetric about all three axes, symmetric about one axis, or has no symmetry at all.[4] Feature-2 for a rotational part specifies if the part is symmetric about its rotational axis (called "beta" symmetry). For a rectangular part, feature-2 specifies if the part has certain features parallel to an axis. For both rectangular and rotational parts, feature-3 specifies if the part is

[4] Recall from Chapter Three that the symmetry we refer to is 180-degree symmetry. Note that if a part is 180-degree symmetric about two axes, it is 180-degree symmetric about three axes.

small and nonabrasive, and feature-4 specifies if the part will not tangle or nest when handled in bulk quantities. In traversing the tree, values are assigned to six variables: *basic-shape, enclosure, feature-1, feature-2, feature-3, feature-4.*

Programming the Expert System

The Teknowledge M.1 expert system package was used to model the suggestion decision graph explained above. This expert system is backward chaining, meaning that the system uses production rules to establish the requirements for known goals. In this implementation, a goal would be the assignment of some value to one of the six variables described above. Different productions would assign different values to each variable. The requirements for each goal were particular properties of the part being designed. Required properties to achieve a desirable goal were output to the designer as suggestions. This was done by using "question" statements that were available with the M.1 system. A rule would output a suggestion question statement if the answer to the question determined the satisfaction of one of its requirements. The user would consider each suggestion and try to work its meaning into the evolving design. If a suggestion could be accepted, the user would inform the system (i.e. answer "yes" to the question) and the requirement would be established. If all of a rule's requirements could be established, the goal of assigning a value to a variable would be achieved. In this way, user responses to suggestions caused values to be assigned to the variables.

Althought this implementation was easy to achieve, it seemed rather awkward. Suggestions were actually generated from the antecedents of the rules. It is more natural to output a suggestion as a consequence of a rule firing. The current state of the design should be checked with the rule antecedents and if a particular suggestion is appropriate, it should be output. This would require production rules of the form:

"If fact-1 and fact-2 and ... fact-N, then suggest suggestion-1." This would require a forward chaining system such as OPS5 [For.1].

Summary

A suggestive mode expert system was implemented with the Teknowledge M.1 expert system package. General design for assembly ideas were derived from tabulated data and placed at the nodes of a suggestive decision graph. These general ideas were related to variables that described the design from an assembly viewpoint. An M.1 rule base was used to provide a representation of the graph. Running M.1 would cause a graph traversal. which would output the general design for assembly ideas as suggestions and assign values to the variables. This system operated with text only.

Conclusion

This chapter has provided a history of the author's previous efforts to devise suggestion-making programs. A first system that made only text suggestions required the complete reorganization of available tabulated data in order to use design analysis information in the design synthesis phase. A second effort investigated the influence of pictorial suggestions on the design process. It was found that pictorial suggestions were preferrable to text suggestions. The tabulated data were again restructured in a third system, which was a true production rule expert system.

The first two systems had no representation of the object being designed, while the third system employed only a very limited list of variables. The next obvious step is to devise a suggestion-making program that contains a representation of the design along with the representation of the design analysis knowledge. The findings of the second effort indicate that these representations should allow a graphical interactivity

with the user. An ideal system would be one in which the system makes suggestions by altering the design in the same way the designer does.

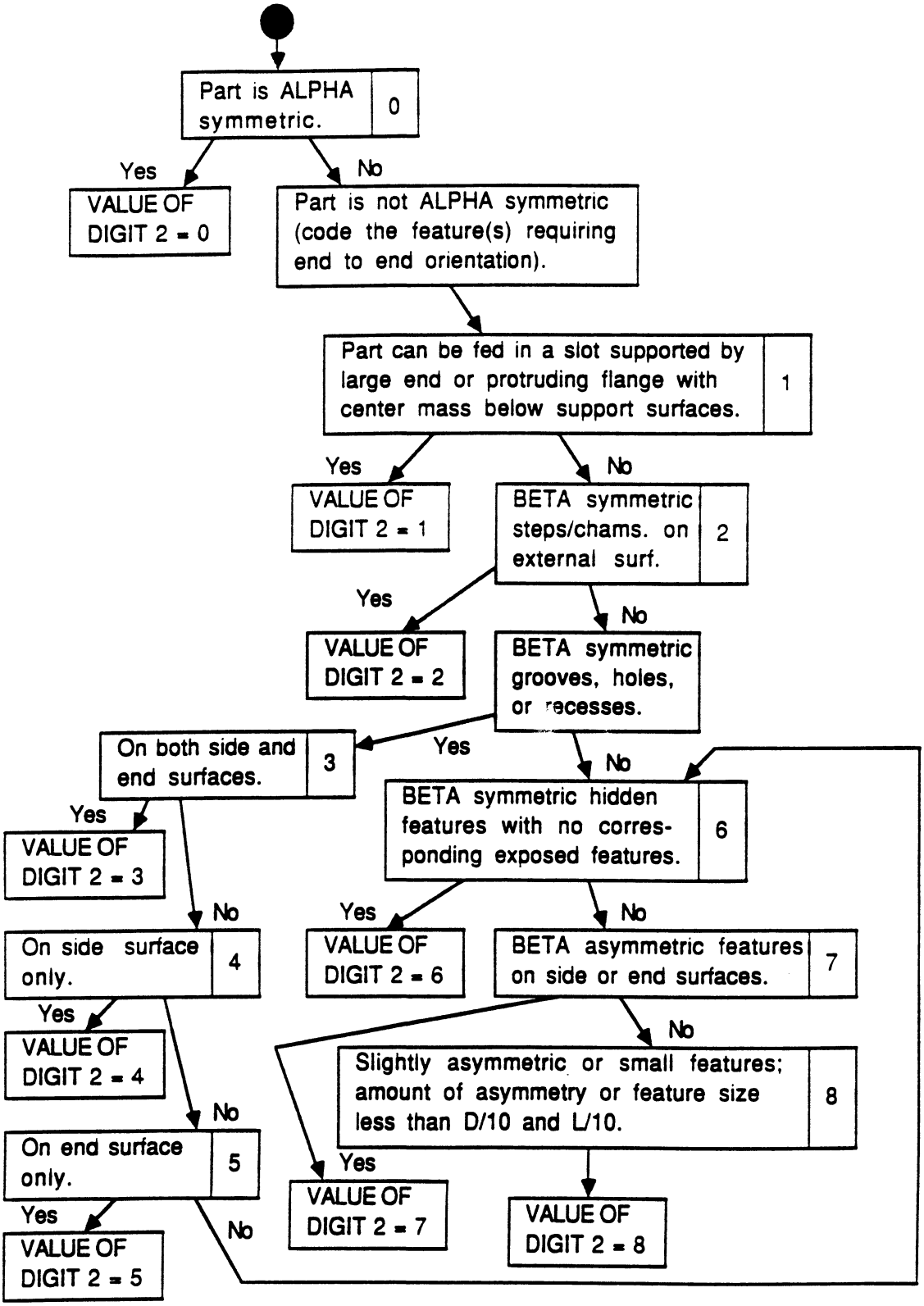


Figure 4.1: Example of a binary suggestion tree.

```
**************************************************
AN ORDERED LISTING OF CONFIGURATION SUGGESTIONS
**************************************************

*** GROUP 1 ***

LONG CYLINDERS L/D > 1.5
PART IS ALPHA SYMMETRIC
PART IS SYMMETRICAL ABOUT ITS PRINCIPLE AXIS (BETA SYMMETRIC)
OE = 0.90  FC = 1.00

LONG CYLINDERS L/D > 1.5
PART CAN BE FED IN A SLOT WITH CENTER OF MASS BELOW SUPPORTING SURFACES
PART IS SYMMETRICAL ABOUT ITS PRINCIPLE AXIS (BETA SYMMETRIC)
OE = 0.90  FC = 1.00

LONG CYLINDERS L/D > 1.5
PART IS ALPHA SYMMETRIC
PART HAS BETA-ASYMMETRIC THROUGH GROOVE OR FLAT SEEN IN END VIEW
OE = 0.90  FC = 1.00

LONG CYLINDERS L/D > 1.5
PART CAN BE FED IN A SLOT WITH CENTER OF MASS BELOW SUPPORTING SURFACES
PART HAS BETA-ASYMMETRIC THROUGH GROOVE OR FLAT SEEN IN END VIEW
OE = 0.90 FC = 1.00

*** GROUP 2 ***

LONG CYLINDERS L/D > 1.5
PART IS ALPHA SYMMETRIC
PART HAS BETA-ASYMMETRIC STEPS, CHAMFERS OR PROJECTIONS ON END SURFACES ONLY
OE = 0.90  FC = 2.00

LONG CYLINDERS-L/D > 1.5
PART CAN BE FED IN A SLOT WITH CENTER OF MASS BELOW SUPPORTING SURFACES
PART HAS BETA-ASYMMETRIC STEPS, CHAMFERS OR PROJECTIONS ON END SURFACES ONLY
OE = 0.90  FC = 2.00

LONG CYLINDERS L/D > 1.5
PART IS ALPHA SYMMETRIC
PART HAS BETA-ASYMMETRIC THROUGH GROOVE SEEN IN SIDE VIEW ON END SURFACE
OE = 0.90  FC = 2.00

LONG CYLINDERS L/D > 1.5
PART CAN BE FED IN A SLOT WITH CENTER OF MASS BELOW SUPPORTING SURFACES
PART HAS BETA-ASYMMETRIC THROUGH GROOVE SEEN IN SIDE VIEW ON END SURFACE
OE = 0.90  FC = 2.00

LONG CYLINDERS L/D > 1.5
PART IS ALPHA SYMMETRIC
PART HAS BETA-ASYMMETRIC THROUGH GROOVE SEEN IN SIDE VIEW ON SIDE SURFACE
OE = 0.90  FC = 2.00

LONG CYLINDERS L/D > 1.5
PART CAN BE FED IN A SLOT WITH CENTER OF MASS BELOW SUPPORTING SURFACES
PART HAS BETA-ASYMMETRIC THROUGH GROOVE SEEN IN SIDE VIEW ON SIDE SURFACE
OE = 0.90  FC = 2.00
```

Figure 4.2: First two ranked groups of rotational triplets.

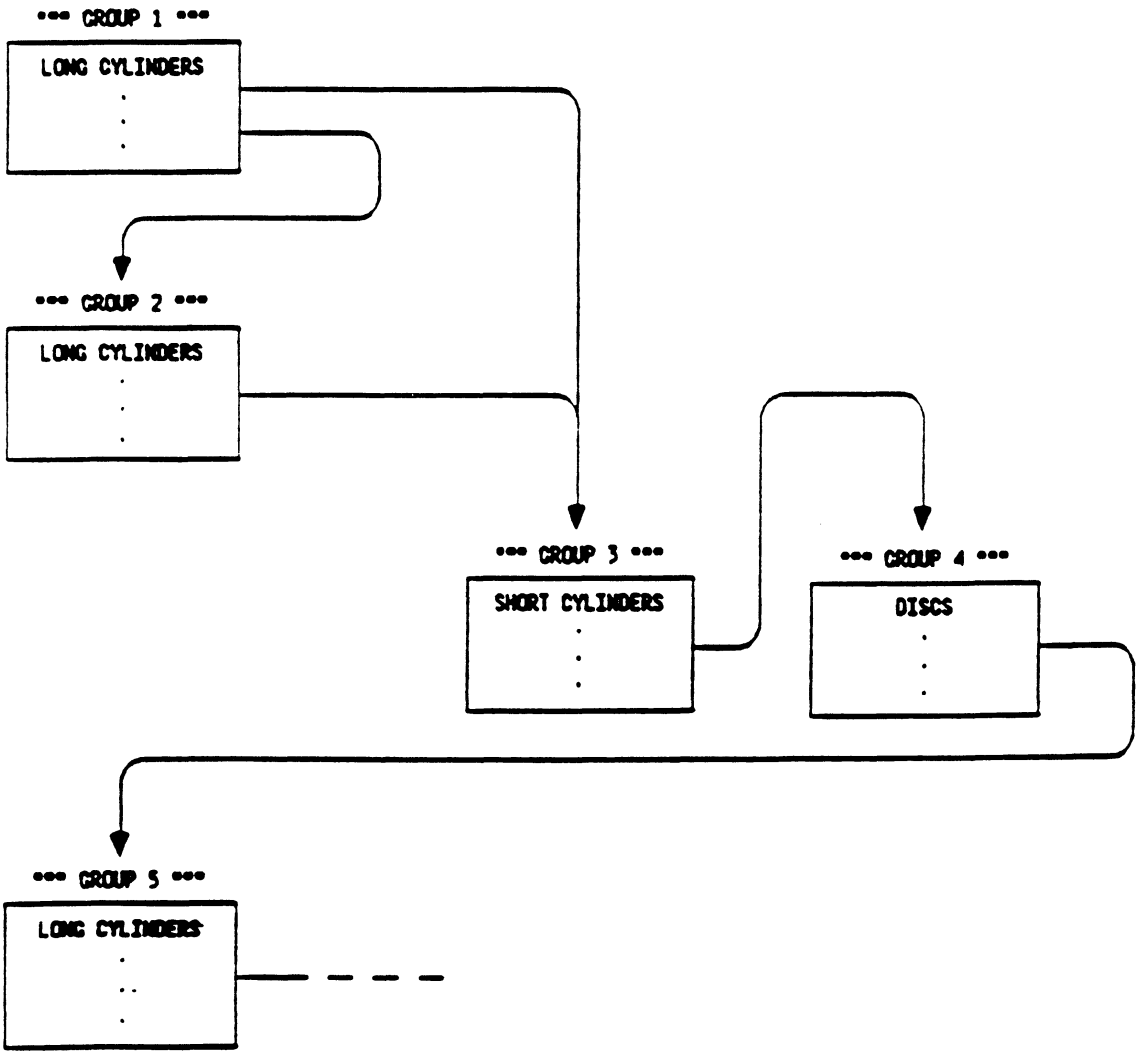


Figure 4.3: Directed graph example for serial heirarchy

```
RUN PROG


--- ENTER ONE OF FOLLOWING COMMANDS ---

1 • REVIEW INSTRUCTIONS
2 • ASSIGN DIGITS 1,2,3 FOR A ROTATIONAL PART
3 • ASSIGN DIGITS 1,2,3 FOR A PRISMATIC PART
4 • ASSIGN DIGITS 4,5 (SURFACE PROPERTIES)
5 • ASSIGN PARAMETER UC (INSERTION)
6 • PERFORM COST CALCULATIONS
8 • END DESIGN SESSION
2

xxx ASSIGN DIGITS 1,2,3 FOR ROTATIONAL PART xxx


CATTEMPTING TO ASSIGN
 OE •    0.90 FC •    1.00]

IT IS RECOMMENDED THAT ...
     (1,2) LONG CYLINDERS L/D > 1.5
IS THIS POSSIBLE ? ...
Y

IT IS RECOMMENDED THAT ...
     (2,0) ALPHA SYM.

     OR

     (2,1) NOT ALPHA SYM., SLOT FED WITH C.M. BELOW SUPPORTING S'FACES
IS THIS POSSIBLE ? ...
Y

IT IS RECOMMENDED THAT ...
     (3,0) BETA SYM.

     OR

     (3,5) BETA ASYM. THRU GROOVE OR FLAT SEEN IN END VIEW
IS THIS POSSIBLE ? ...
Y
xxx OPTIMAL PARAMETERS ASSIGNED xxx


xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
RESULTS OF SEARCH

OE •    0.90     FC •    1.00

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

PLEASE INPUT THE SPECIFIC 1,2,3 BOOTHROYD DIGITS
YOU PICKED, IN THAT ORDER AS INTEGERS ...
2,0,0
```

Figure 4.4: Assigning values to the first three digits with the text-only suggestive system.

```
--- ENTER ONE OF FOLLOWING COMMANDS ---

1 - REVIEW INSTRUCTIONS
2 - ASSIGN DIGITS 1,2,3 FOR A ROTATIONAL PART
3 - ASSIGN DIGITS 1,2,3 FOR A PRISMATIC PART
4 - ASSIGN DIGITS 4,5 (SURFACE PROPERTIES)
5 - ASSIGN PARAMETER UC (INSERTION)
6 - PERFORM COST CALCULATIONS
X - END DESIGN SESSION
4

XXX ASSIGN DIGITS 4,5 FOR PART XXX


[ATTEMPTING TO ASSIGN
 PARAMETER -    0.00]

IT IS RECOMMENDED THAT ...
     (4,0) SMALL, NON-ABRASIVE, DON'T OVERLAP, NOT DELICATE, NON-FLXBLE
IS THIS POSSIBLE ? ...
Y

IT IS RECOMMENDED THAT ...
     (5,0) NOT TANGLE OR NEST, NOT LIGHT, NOT STICKY
IS THIS POSSIBLE ? ...
Y
XXX OPTIMAL PARAMETERS ASSIGNED XXX


XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
RESULTS OF SEARCH

 PARAMETER -    0.00

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

PLEASE INPUT THE SPECIFIC BOOTHROYD 4,5 DIGITS
YOU PICKED, IN THAT ORDER AS INTEGERS ...
0,0
```

Figure 4.5: Assigning values to the fourth and fifth digits with the text-only suggestive system.

```
--- ENTER ONE OF FOLLOWING COMMANDS ---

1 - REVIEW INSTRUCTIONS
2 - ASSIGN DIGITS 1,2,3 FOR A ROTATIONAL PART
3 - ASSIGN DIGITS 1,2,3 FOR A PRISMATIC PART
4 - ASSIGN DIGITS 4,5 (SURFACE PROPERTIES)
5 - ASSIGN PARAMETER UC (INSERTION)
6 - PERFORM COST CALCULATIONS
X - END DESIGN SESSION
5

XXX ASSIGN DIGITS 6,7 FOR PART XXX


[ATTEMPTING TO ASSIGN
 PARAMETER -    0.80]

IT IS RECOMMENDED THAT ...
     (6,3) FINAL SECURING, STRAIGHT INSERT., VERTICALLY ABOVE

     OR

     (6,9) SOLIDS IN PLACE, NON-SOLIDS ADDED OR PARTS MANIPULTD.
IS THIS POSSIBLE ? ...
Y

IT IS RECOMMENDED THAT ...
     (7,18) SCREWING, EASY TO ALIGN & POS'N, NO RESISTANCE

     OR

     (7,22) MECH. FSTNG., NONE OR LOCAL PLASTIC DEF., SCREWING ETC.

     OR

     (7,27) NON-MECH. FSTNG., CHEMICAL FSTNG. (ADHESIVES ETC.)
IS THIS POSSIBLE ? ...
Y

IT IS RECOMMENDED THAT ...
     (6,3) FINAL SECURING, STRAIGHT INSERT., VERTICALLY ABOVE
     AND
     (7,18) SCREWING, EASY TO ALIGN & POS'N, NO RESISTANCE

     OR

     (6,9) SOLIDS IN PLACE, NON-SOLIDS ADDED OR PARTS MANIPULTD.
     AND
     (7,22) MECH. FSTNG., NONE OR LOCAL PLASTIC DEF., SCREWING ETC.

     OR

     (6,9) SOLIDS IN PLACE, NON-SOLIDS ADDED OR PARTS MANIPULTD.
     AND
     (7,27) NON-MECH. FSTNG., CHEMICAL FSTNG. (ADHESIVES ETC.)
IS THIS POSSIBLE ? ...
Y
XXX OPTIMAL PARAMETER ASSIGNED XXX


XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
RESULTS OF SEARCH

 PARAMETER -    0.80

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

PLEASE INPUT THE SPECIFIC BOOTHROYD 6,7 DIGITS
YOU PICKED, IN THAT ORDER AS INTEGERS ...
3,18
```

Figure 4.6: Assigning values to the sixth and seventh digits with the text-only suggestive system.

```
--- ENTER ONE OF FOLLOWING COMMANDS ---

1 - REVIEW INSTRUCTIONS
2 - ASSIGN DIGITS 1,2,3 FOR A ROTATIONAL PART
3 - ASSIGN DIGITS 1,2,3 FOR A PRISMATIC PART
4 - ASSIGN DIGITS 4,5 (SURFACE PROPERTIES)
5 - ASSIGN PARAMETER UC (INSERTION)
6 - PERFORM COST CALCULATIONS
X - END DESIGN SESSION
6

PLEASE INPUT  Y -
MAXIMUM PART DIMENSION IN MILLIMETERS ...
25.

PLEASE INPUT  FR -
REQUIRED RATE OF ASSEMBLY ...
60.

PLEASE INPUT (AS AN INTEGER)  N -
NUMBER OF ASSEMBLIES PERFORMED SIMULTANEOUSLY ...
2




          XXXXXXXXXXXXXXXXXXXXXXXXX

           RESULTS OF DESIGN STUDY

          XXXXXXXXXXXXXXXXXXXXXXXXX

--- ROTATIONAL PART ---

MAXIMUM PART DIMENSION -   25.00 MM.
REQUIRED RATE OF ASSEMBLY -   60.00 PARTS/MIN
NUMBER OF SIMULTANEOUS OPERATIONS - 2

FIVE DIGIT AUTOMATIC HANDLING CODE -
     20000

     ORIENTING EFFICIENCY 'OE' -    0.90
     RELATIVE FEEDER COST  CR - FC + DC -    1.00
     MAXIMUM BASIC FEED RATE  FM -   54.00 PARTS/MIN
     DIFFICULTY RATING FOR AUTOMATIC HANDLING  DF -    1.11
     COST OF AUTOMATIC HANDLING PER PART  CF - 0.03 X DF -    0.03 CENTS

TWO DIGIT AUTOMATIC INSERTION CODE -
     38

     RELATIVE WORKHEAD COST  UC -    0.80
     DIFFICULTY RATING FOR AUTOMATIC INSERTION  DI -    0.80
     COST OF AUTOMATIC INSERTION PER PART  CI - 0.06 X DI -    0.05 CENTS

OPERATION COST -    0.16 CENTS
```

Figure 4.7: Final output results of the design study with the text-only suggestive system.

```
*****************
*               *
*     HELLO !   *
*               *
*****************
```

Welcome to the Design-for-Asssembly suggestion program. The program will make suggestions as you design something. These suggestions will be given both as text and graphical pictures: the graphics will be output on the larger computer screen to your right, and the text will be output on this computer screen. The text of the suggestion will tell you exactly what to do to view the appropriate picture.

Note that a menu is output below. Your design session will be divided into three phases: assigning the first, second, and third Boothroyd digits. Boothroyd is the person who figured out what designs are good and what designs are bad, and how to code part features with his "digits". Each of the three phases will output text/graphic suggestions. You can re-do any of the phases at any time by simply making the menu selection.

The main idea here is to let the suggestions alter the way you are thinking about the design you are creating. Be open-minded! Try to make your design have the properties of the suggestion. If you accept the suggestion the Boothroyd digit will be assigned and the design phase will be completed; if you reject the suggestion, a new suggestion will be output. The process goes on until you accept a suggestion or until the program runs out of suggestions. It is important to note that the suggestions get progressively worse. Try to accept the earlier suggestions in order to get a better design.

Keep a scratch pad handy to keep track of the features that must be on your design configuration. Feel free look at all the suggestions in any phase before deciding what to do, but remember that the earlier ones are best. You don't have to make your design look exactly like the graphic output; the idea is to include similar characteristics in your design. You can modify sizes and proportions as long as you stay with the general idea.

Ok, enough talk. Mark will do an example with you and then you are on your own. Good Luck!

Figure 4.8: Introductory instructions output by the text-and-picture suggestive system.

```
***************
*             *
*     MENU    *
*             *
***************

Please input choice followed by a carriage return:

     1.  Assign first Boothroyd digit.

     2.  Assign second Boothroyd digit.

     3.  Assign third Boothroyd digit.

     ?.  Review introductory instructions.

     *.  End the design session

1

******************
*                *
*     DIGIT 1    *
*                *
******************

The following suggestions pertain to assigning the first Boothroyd
digit.


[ BOOTHROYD INDICES (1,2) ]

IT IS RECOMMENDED THAT ...


LONG CYLINDER

Try to make the part so that its overall shape (envelope) is that of a
long cylinder with length/diameter greater than 1.5.  Type D ROT.1.2 at the
graphics terminal to see an example.  There may be features protuding from
the envelope and there may be features within the envelope.

IS THIS POSSIBLE?  Y/N ...
Y

** BOOTHROYD DIGIT ASSIGNED **
```

Figure 4.9: Assigning a value to the first digit with the text-and-picture suggestive system.

```
**************
*            *
*     MENU   *
*            *
**************

Please input choice followed by a carriage return:

     1.  Assign first Boothroyd digit.

     2.  Assign second Boothroyd digit.

     3.  Assign third Boothroyd digit.

     ?.  Review introductory instructions.

     *.  End the design session

2

******************
*                *
*     DIGIT 2    *
*                *
******************

The following suggestions pertain to assigning the second Boothroyd
digit.


[ BOOTHROYD INDICES (2,0) ]

IT IS RECOMMENDED THAT ...


ALPHA SYMMETRIC

Try to design the part so that it has end to end symmetry.  The part will
have 180 degree symmetry abouut an axis perpendicular to the rotational
axis.  Type D ROT.2.0.1 and D ROT.2.0.2 at the graphics terminal to see
two examples.


IS THIS POSSIBLE?  Y/N ...
N


[ BOOTHROYD INDICES (2,1) ]

IT IS RECOMMENDED THAT ...


CENTER OF MASS BELOW SUPPORTING SURFACES

Design the part so that it could slide down a track with its center of mass
hanging below the supporting surfaces.  The best example is a normal
machine screw configuration (type D ROT.2.1 at the graphics terminal to see
this).  Any type of screw head is allowable.  Note that this and all
following suggestions in this phase of the design are not alpha symmetric.

IS THIS POSSIBLE?  Y/N ...
Y

** BOOTHROYD DIGIT ASSIGNED **
```

Figure 4.10: Assigning a value to the second digit with the text-and-picture suggestive system.

```
***************
*             *
*     MENU    *
*             *
***************

Please input choice followed by a carriage return:

     1.  Assign first Boothroyd digit.

     2.  Assign second Boothroyd digit.

     3.  Assign third Boothroyd digit.

     ?.  Review introductory instructions.

     *.  End the design session

3

******************
*                *
*     DIGIT 3    *
*                *
******************

The follo     sugg     ons pertain to assigning the third Boothroyd
digit.


[ BOOTHROYD INDICES (3,0) ]

IT IS RECOMMENDED THAT ...


BETA SYMMETRIC

Design the part     that it is symmetric about its rotational axis.  This is
known as beta symmetry.  Type D ROT.3.0.1, D ROT.3.0.2, D ROT.3.0.3, at the
graphics terminal to see three examples.  These features are just examples,
any features that provide beta symmetry are allowed.

IS THIS POSSIBLE?  Y/N ...
Y

** BOOTHROYD DIGIT ASSIGNED **
```

Figure 4.11: Assigning a value to the third digit with the text-and-picture suggestive system.

```
***************
*             *
*     MENU    *
*             *
***************

Please input choice followed by a carriage return:

     1.  Assign first Boothroyd digit.

     2.  Assign second Boothroyd digit.

     3.  Assign third Boothroyd digit.

     ?.  Review introductory instructions.

     *.  End the design session


***************
*             *
*     DONE    *
*             *
***************

The design session has been terminated.

Thanks very much for your time and patience.

BOOTHROYD VALUE 1 = 2
BOOTHROYD VALUE 2 = 1
BOOTHROYD VALUE 3 = 0
```

Figure 4.12: Conclusion of a session with the text-and-picture suggestive system.

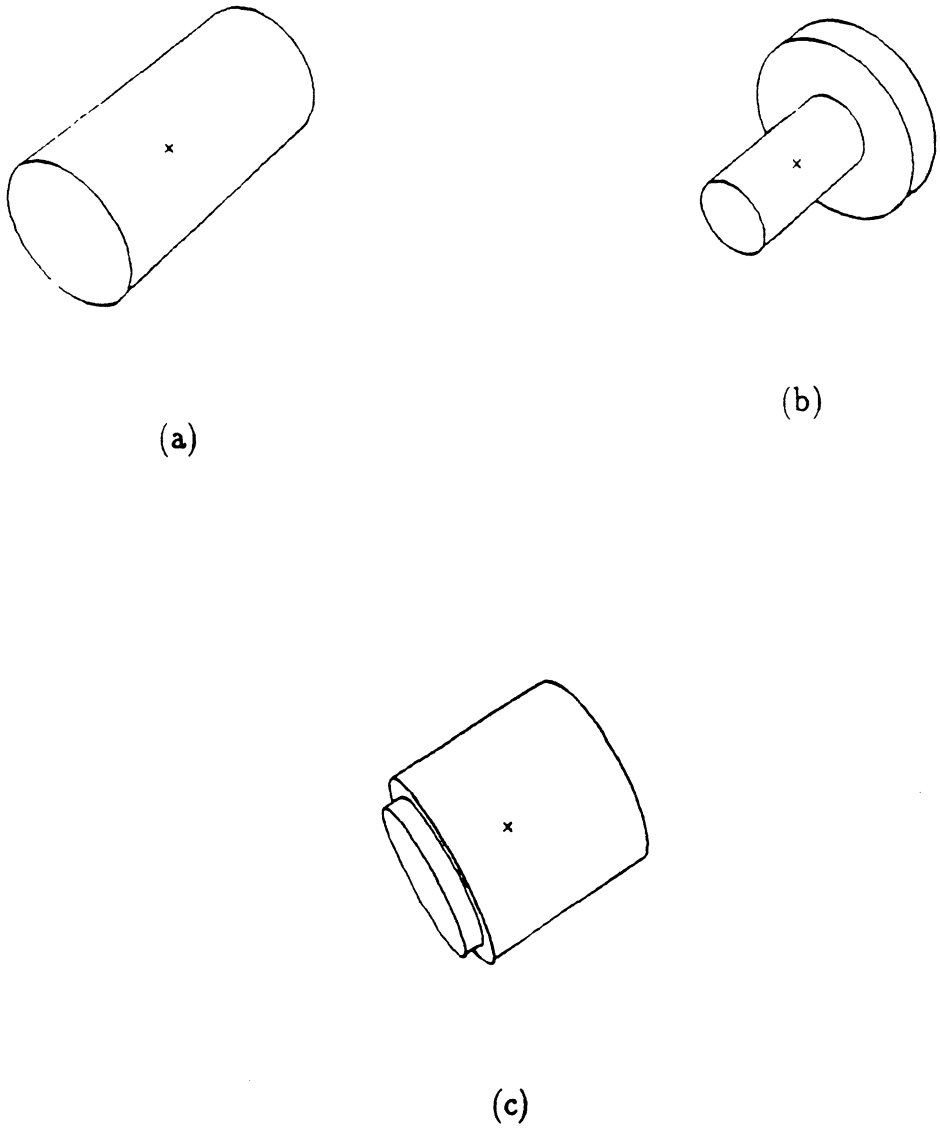


Figure 4.13: Example pictures output by the text-and-picture suggestive system. (a) An example picture from phase 1. Produced with the command "D ROT.1.2". (b) An example picture from phase 2. Produced with the command "D ROT.2.1". (c) An example picture from phase 3. Produced with the command "D ROT.3.0.2".

CHAPTER V

SYSTEM REQUIREMENTS AND DEFINITIONS

Introduction

Previous attempts to devise a suggestion-making program indicate that suggestions should be output on a graphical design model. This is the fundamental system requirement. In addition, it would be useful if the architecture of the system generating the suggestions is somewhat flexible. This would allow suggestions to be output in several different interactivity schemes and the encoding of other applications using the available design knowledge. This chapter first outlines the requirements of such a system and determines the overall methods that will be used to meet these requirements. Then, some terms that are used in subsequent chapters will be defined. The following chapter describes a system architecture employing the methods chosen here.

System Requirements

Several requirements are obviously necessary for the desired system. The system must contain some representation of the designed artifact. This representation is used

for the design analysis. To carry out the analysis, there must be some representation of design analysis knowledge. Additional knowledge is required to generate design improvement suggestions based on the results of the analysis. Here, we choose the methods to satisfy these requirements.

A suitable representation of the design should promote the design analysis. The design representation chosen should be amenable to the processing required to analyze the design with respect to some concurrent engineering aspect. Additionally, the representation should facilitate the creation of a very easy to use human-computer interface. Users should be able to concentrate on the design task, rather than on the use of the CAD system.

These considerations make a *feature* representation seem like the natural choice. Features are meaningful geometric entities, such as grooves, holes, and chamfers, that make up a design. The representation of the design should be based on representations of its constituent features. Features are directly related to later processes and aspects of the product life cycle [Dix.1, Bri.1, Pra.1]. Machining, for example, naturally creates part features as a consequence of the material removal. The Boothroyd-Dewhurst system determines the quality of a part by noting the geometric features. Different aspects of the product life cycle will relate to different features. Portions of a part that are significant to one concern may be unimportant to others.

Features also seem like reasonable building blocks for the designer. Users will naturally construct a design by sequentially building a series of features. Eberts et al. provide evidence of this [Ebe.1]. Recall from the description of their experiment given in Chapter Two that experienced and inexperienced users were given the task of rendering an existing design concept with a CAD modeler. The design was a three dimensional solid object, with several volumetric features. The expert users, with considerable experience using the CAD system, created the model with techniques that optimally used the capabilties of the CAD system. These techniques

also suppressed the geometric nature of the individual features of the design because they required that portions of each feature be created before any one feature was completed. The novice users, following their natural inclinations, tried to create the model with a feature-by-feature approach, completing each feature before moving on to another. This proved to be an inferior technique for the given CAD system. The important point is that the novice users *naturally* thought of creating the model by building its constituent features. A CAD system should accomodate this type of approach.

With the design knowledge representation, we hope to model the decision making processes of a skilled designer. A designer who is an expert in the design analysis domain can determine if a design is good or bad and suggest improvements. These capabilities are based on many decisions that the designer can make about the design and possible modifications to it. A large number of decisions can be modeled appropriately with production rules. The if-then structure naturally represents a decision and a set of production rules can easily expand as more decision processes are understood and encoded. Production rules will be used to analyze the design and determine possible improvements.

The system requirements have led to some basic methods that will be used in the implementation of the intelligent CAD system. Designs will be represented with features and design knowledge will be represented with production rules. The next chapter describes a system architecture that uses these methods to meet the requirements. Before proceeding, however, some useful terminology is introduced.

Definitions

In the context of the user-system interaction, it is useful to think of a feature, as described above, as an *alteration step*. Alteration steps are the primitives that are used to create the design model. A user will build a design with a series of alteration

steps, stopping when they feel the design is completed. An important idea is that the suggestion-making system can build the same alteration steps. The system and the user can modify the design with the same primitives. An alteration step created by the user will be called a *feature* and an alteration step created by the system will be called a *suggestion.* Features and suggestions are tentative in nature and can be easily discarded from the design. When the user chooses to accept a feature or suggestion as part of the design, an *incorporation* process is carried out, making the alteration step a permanent part of the design. Incorporated alteration steps are called *objects.* Finally, a *suggestive system* is a system that makes suggestions during the design process.

To clarify these definitions, consider the design cycle shown in Figure (5.1). In Figure (5.1a) the design is shown before the user's feature input. For the purposes of this example, the design is composed of two objects: a block, and a groove opening in the -Y direction. Figure (5.1b) shows the user's feature input of a groove opening in the +Y direction. Note that the edge lines of the block are not trimmed away with the input of the groove feature, as would be expected. This indicates the tentative nature of the feature input. In Figure (5.1c) the system's suggested improved groove is shown shaded. In the actual implementation, objects, features, and suggestions have different colors. Note how the suggestion makes the entire design 180-degree symmetric about the Z axis. This might be a design for assembly consideration. Figure (5.1d) shows the suggestion accepted and incorporated into the design. Note the trimmed lines to make a geometrically valid object.

The distinction between tentative and incorporated alteration steps is a result of a simplistic treatment of the CAD data associated with the design. As will be shown in the next chapter, the entire design representation is made up of representations of the individual incorporated alteration steps (objects). Very little information concerning the CAD rendering of the objects is represented. In going from Figure

(5.1c) to Figure (5.1d), for example, two edge line of the block are deleted and four new block edge lines are created. These new edge lines are not directly associated to the added groove's representation. If the groove were to be subsequently moved, it could be difficult to determine how to correctly modify the edge lines (particularly if some of the four new edge lines were altered for some reason in the meantime). The system developed here overcomes problems such as this by allowing modification of tentative alteration steps only, for they have not yet altered any other part of the design. This is clearly a deficiency in the design representation and should be corrected by effectively associating geometric entities with each object.

We should be careful, however, to not let relationships among the incorporated objects and various pieces of CAD data overconstrain the suggestion generation process. Consider Figure (5.2), where both the groove and the hole have been incorporated into the design. If the CAD data is properly considered, both objects might be associated with the plane at the bottom of the groove. If the user solicits suggestions regarding the groove, and one of the suggestions is to move the groove in the -X direction, what should be done with the hole? Since the plane at the bottom of the groove is associated with the hole, it is reasonable to move the hole along with the groove. An alternate suggestion is to leave the hole where it is and lengthen it to meet the upper plane of the block. This might be a superior design idea and therefore should be brought to the user's attention.

It is possible that the user intended that the groove and hole interact in the manner shown, possibly to meet some design goal. This should not prevent the suggestive system from proposing alternate configurations. Recall from Chapter One that the system uses no representation of the user's design goals. Put another way, the system is not concerned with the designer's intent. The system attempts to modify the design in ways that will yield a more favorable design analysis and output these possibilities to the designer. With this information, the designer might

create a different, higher quality design that meets all of their design goals.

Still, the two suggestions described (modifying incorporated objects) might be possible with a better design representation. At the end of the next chapter, a heirarchical representation of objects is proposed as a possible improved representation.

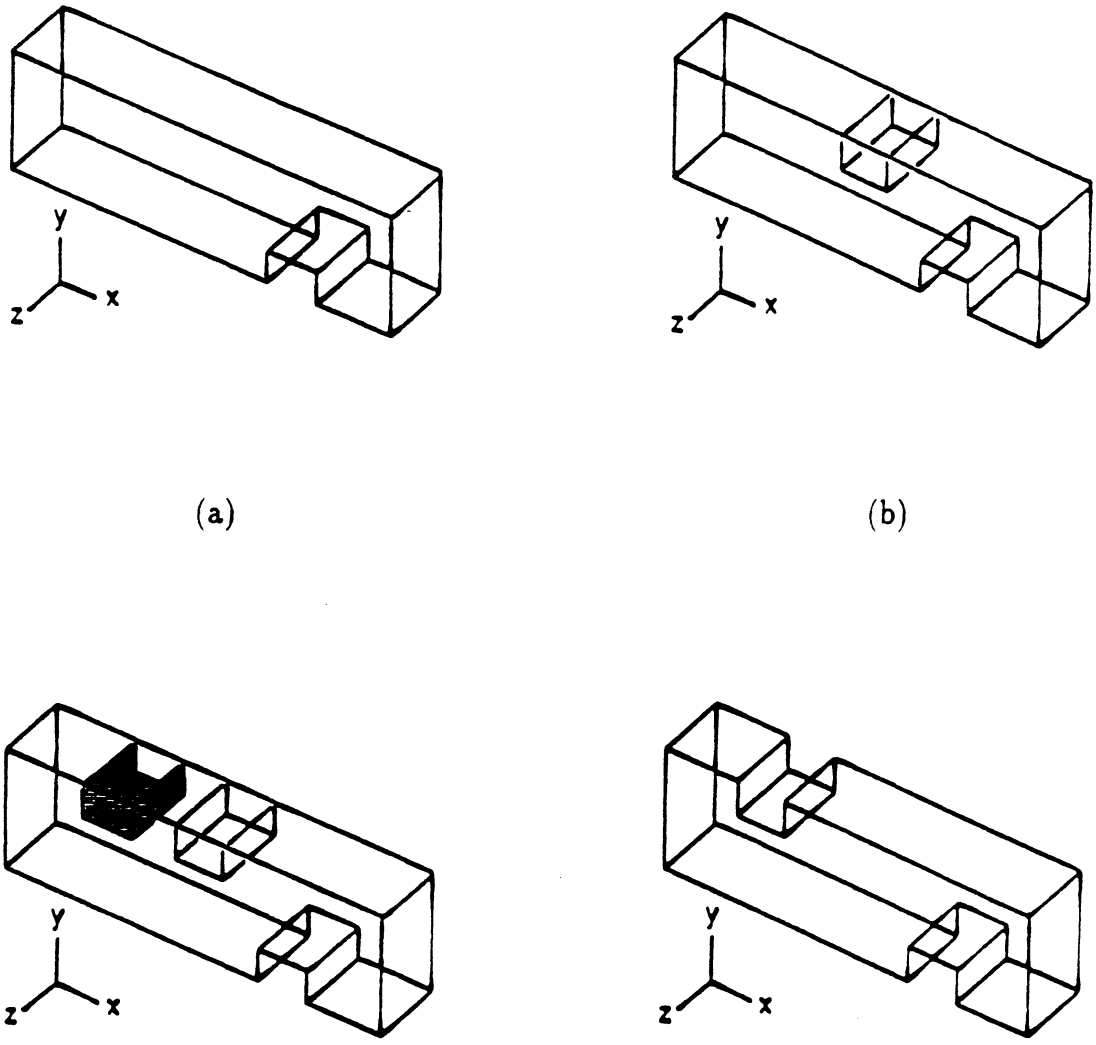


Figure 5.1: Clarification of definitions. (a) Initial design composed of two objects. (b) User's feature input. (c) System's suggestion input. (d) Suggestion incorporated into the design.

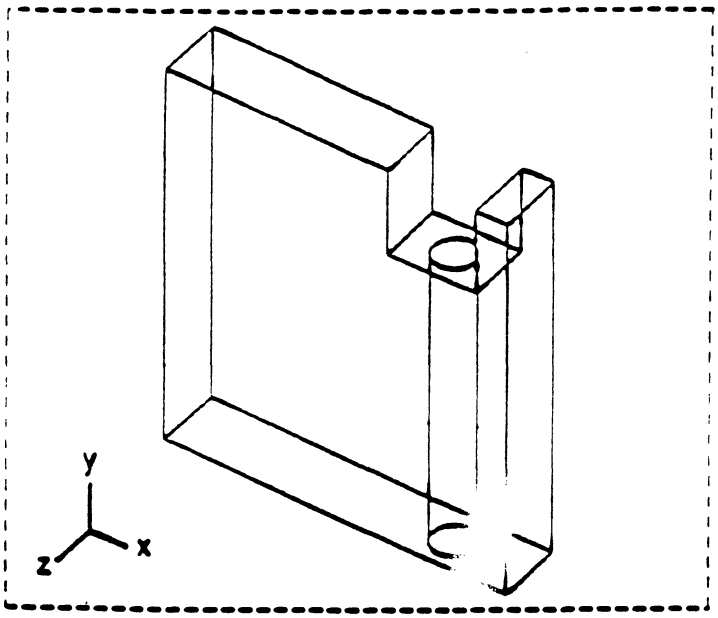


Figure 5.2: Interacting features cause difficulty in generating suggestions.

CHAPTER VI

SYSTEM DESIGN AND IMPLEMENTATION

Introduction

The preceding chapter describes system requirements for an intelligent suggestive CAD system. This chapter specifies an architecture for such a system. This overall architecture is really two architectures. One is the architecture of the production rule system, which is discussed first. This will include a description of the data structures used for the the design alteration steps and the production rules. The inference engine match-resolve-execute algorithm is also explained. This production rule system is invoked by the other architecture, a higher-level intelligent CAD system architecture, which is discussed next. The fundamental program modules are described to explain the operation of the overall system. The goal in specifying these architectures is to provide enough information to allow others to replicate this intelligent CAD system with other hardware and software systems.

To this end, we use a program design language (PDL) to specify the data structures and algorithms of both architectures. A PDL is a structured way to describe the data and operations of a computer program. English language statements describing fundamental operations are presented along with the control constructs of

a programming language. A PDL description, therefore, looks much like a normal program except that some portions are described with natural English language. The PDL used here is adapted from one described by Pressman ([Pres.1], pages 253-261). Progam language keywords will be boldface uppercase (e.g. **IF-THEN**, **WHILE-CONTINUE**), and program module names (i.e. names of programs and procedures) will be normal uppercase. Comments are enclosed in "{ }" brackets. Otherwise, there will be a generally loose syntax. Other specifics of the PDL will be presented in the various explanations that follow. A PDL was chosen over some graphical method of describing the software (e.g. flowcharts) because it facilitates recoding the modules. The important constructs of each program and procedure are already in the specification and one need only devise code for the parts described with natural language. The intelligent CAD system architecture is more precisely described than the production rule system architecture. It is important to specify only what the production rule system does, as it can be coded in a number of ways. The CAD system architecture, on the other hand, is made up of many procedures and it is important to state the exact relationships between them.

A final note concerns the implementation of the system. The approach taken was to add feature and production rule capabilties to an existing CAD system. The CAD package used is a command-driven graphics modeling system that is powerful, but difficult to use [Gen.1]. The system programming language, called DAL, is a Pascal-like language that employs basic list data structures and operators. DAL is tailored to programming graphical output and contains system functions for such tasks as solid modeling, hidden line removal, and plot generation. It was felt that it would be easier to add modest feature and production rule capabilities to a CAD system than to add sophisticated graphics capabilities to an artificial intelligence system. Since list data structures greatly facilitate artificial intelligence programming techniques, and this system was readily available to the author, it was chosen for the implemen-

tation. In the following discussions, the reader should be careful not to confuse the graphics representation of the design model with the feature representation. The graphics representation is maintained by the underlying CAD system and only provides the graphical rendering of the design. The feature representation of the design is maintained by the intelligent suggestive system and is used for design analysis and suggestion generation. As explained in the previous chapter, only minimal links are maintained between the two representations.

Production System Architecture[1]

Introduction

An obvious question is why was a production rule system developed when many commercial production rule systems were already available? The problem with available systems was their inability to easily and intimately communicate with a CAD system. This communication *was* possible, but very difficult to achieve. For this reason, a production rule system was developed that was completely embedded in an existing CAD package.

What follows are data structures and algorithms for the production rule system. A sample production is described to illustrate how the algorithms operate on the data and cause a single suggestion. The organization of groups of productions to perform various tasks is discussed in Chapter Eight.

Data Structures and Algorithms

---

[1] This section is derived from [Jak.3]. Figures (6.1) through (6.4) are also from [Jak.3].

Features, suggestions and objects are represented as association lists (see e.g. [Win.1]) of the form:

```
((property-name-1, property-value-1),
 (property-name-2, property-value-2),
 . . .,
 (property-name-N, property-value-N))
```
(5)

For example, a solid block feature might be represented in the following way:

```
((Type, Block),
 (Name. BLOCK01),
 (X-Dimension, 10.),
 (Y-Dimension, 15.),
 (Z-Dimension, 8.))
```
(6)

This appears to be a natural way of representing entities that have several properties, as most alteration steps likely will.

Production rules are represented as a list of text strings of the form:

```
(Rule-name,
 ((Antecedent-1),
  (Antecedent-2),
  . . .,
  (Antecedent-J)),
 ((Consequent-1),
  (Consequent-2),
  . . .,
  (Consequent-K)),
 Rule-comment)
```
(7)

The rule has a name, a list of antecedent conditions that must be simultaneously true for the rule to fire, a list of consequent actions that are performed, and a place for comments about the rule or other data that may be needed.

Each antecedent or consequent is a list of four text strings of the form:

((Antecedent-program-name or Consequent-program-name),
(Feature-property-name),
(Object-property-name), (8)
(Variable-list-name))

There are several things to note here. First, antecedents and consequents are actually invocations of DAL subprograms. The name of the subprogram is the first string in the antecedent or consequent and the names of the arguments passed to the subprogram are the remaining three strings. These subprograms can be programmed to perform any task desired, giving each individual rule significant processing power and flexibility. Second, one should note the presence of the "Variable-list-name," which is a list of DAL variables that are passed to the antecedent or consequent subprogram. This list is used to hold values that are not related to any feature or object and may be pertinent to the particular rule.

The match-resolve-execute cycle is feature driven. That is, a feature that was just input, or one of the objects incorporated permanently into the design, must be singled out as *the* feature for each iteration of the match-resolve-execute cycle. Consider first the match step as shown in Figure (6.1). For each rule, the antecedents are invoked with the feature and one of the objects. If all the antecedents evaluate to "true," then the object is "attached" to the rule by adding the object to the rule's "attachment list." The first antecedent that evaluates to "false" ends consideration of the rule with the particular object. This test is done for all objects. Note that each object that causes the match to succeed (i.e. all antecedents evaluate to "true") is added to the attachment list. This yields an attachment list of the form (rule,

(object, object, ..., object)). If at least one object causes the rule to succeed, the attachment list is added to a conflict list, which is just a list of attachment lists. This is done for all rules.

The resulting conflict list has the form:

$$\begin{aligned}&((\text{rule, (object, object, . . ., object)}),\\ &(\text{rule, (object, object, . . ., object)}),\\ &\ldots,\\ &(\text{rule, (object, object, . . ., object)}))\end{aligned} \tag{9}$$

To perform conflict resolution, one must choose some sublist of this list of "triggered" rules. There are many possibilities. Different conflict resolution strategies are used in different situations. Often, if a rule base is small, only one rule will match in any given situation. Most commonly, several rules will match and all will later be executed, for example, generating all design improvement suggestions related to a feature input.

The execution part of the cycle is similar to the matching part, except that all consequents are invoked, regardless of what they do. The PDL description of the execution process is shown in Figure (6.2).

A Sample Production

To get an idea of how the production system is used to encode design knowledge, consider an example production that makes improvement suggestions about blocks. When designing something with the basic shape of a block, a heuristic that can be derived from the Boothroyd-Dewhurst Charts is that a flat plate-like block is better than a long bar-like block, which is in turn better than a cube-like block. Therefore, when the user inputs a block feature, if it is not flat, it would be beneficial to suggest ways to make it flat. If a flat block is not possible, it would be beneficial to

suggest ways to make it long. Figure (6.3a) shows a user's input feature of a block with dimensions which, according to the Boothroyd-Dewhurst Charts, make it cubic. Figure (6.3b) shows the system's suggested improved long block, which appears as a different color than the user's input on the graphics screen, and with explanatory text on an alphanumeric screen. The block is made long by lengthening its longest side. Figure (6.3c) shows the system's suggestion incorporated into the design to complete one cycle of the suggestive mode.

Figure (6.4a) shows the actual DAL code rule that made the suggestion above. Figure (6.4b) is an explanatory annotation of the rule. The antecedent routine MATCHSTRING can be used to compare the character string property value of some feature or object property with a character string provided in the variable list. So the first antecedent determines if the feature's "Type" property value is equal to the character string found in the list named "BLOCK." Since the list "BLOCK" is just a single element list containing the string "Block", the first antecedent simply checks if the feature is a block. Note that the object property used is "None." This results in no object property being used in the invocation of MATCHSTRING.

In a similar way, the second antecedent checks if the object's "Name" property value equals the string value found in the list "DUMMY01," which is the string "Dummy01." More simply, it checks if a particular object is being considered by the rule. This second antecedent insures that the rule will match with only one object (i.e. the object with the "Name" property value equal to "Dummy01"). Without it the rule would match with each object and the suggestion shown in Figure (6.3) would be output repeatedly. We call the object named "Dummy01" a dummy object, since its property values are not needed by the rule. The third antecedent determines if the block has a cubic enclosure. Another rule infers and adds this property to the block's association list in an earlier use of the production rule subprograms.

The consequent routine SUGGESTBLK$ copies the side lengths of the feature

block and modifies them with the ratios found in the list "SUGLONG1." It then uses these new side lengths to output the suggestion block. MESSAGE outputs the text messages found in the list "MESSLONG1" on the alphanumeric screen. USERRESPONSE solicits the user for a decision regarding whether to consider incorporating the suggestion, or to view other suggestions. Both MESSAGE and USERRESPONSE require no property values from the feature or object. A brief comment ends the rule.

It is important that many of the antecedent and consequent routines can accept input from the feature and/or the object, and that a list of variables can be used. This makes each routine more powerful and reduces the total number of routines that are required. To determine, for example, if a string property value of the feature is equal to a string property value of the object, MATCHSTRING is invoked with a property name from the feature, a property name from the object, and no variable list (by using the string "None" for the variable list name). This seems more natural than having three string-matching routines: one to match the feature properties with variables, one to match object properties with variables, and one to match feature properties with object properties.

Suggestive CAD System Architecture

Introduction

The production rule system described above is invoked as one of the many procedures of a larger intelligent suggestive CAD system. This system can operate in several different suggestive modes. The only suggestive mode seen thus far (in Figures (1.1), (5.1), and (6.3)) might be called a "before" suggestive mode. A user builds a feature, and suggestions are output *before* the feature or some suggestion is

incorporated into the design. Many other suggestive modes are possible, and some of particular interest are discussed in the next chapter (there, we will see that a "before" mode is a specific type of *during* suggestive mode). Here we describe a software architecture that would effect this *before* mode. Other modes would be coded similarly, with minor changes to the high-level control programs. It will be seen in Chapter Seven how the same suggestive rule bases can be used in different suggestive modes.

We hope to provide enough information to allow others to implement a similar suggestive system with other hardware and software. We cannot describe all the code, however, because the system is far too large. Instead, we recognize sensible limitations on the system description. First, the description must not specify programming that relates to a specific alteration step, since other systems may be designed with other alteration steps. It should not, for example, describe the code that will derive the properties of a groove. It should only specify the software that is used with all the features. Second, the description must not specify programming that relates to specific hardware or software. This is obvious, as our motivation is system generality.

Basic System Design

With these limitations in mind, we provide a hierarchical description of idealized program modules. This description is idealized in the sense that software structures that are due to the specific hardware and software considerations are not presented. This description will go from the general to the specific by describing several different modules at different heirarchical levels. The overall structure and the relationships between the modules is shown in Figure (6.5).

We begin with the highest-level calling program, "SUGGEST1," as shown in Figure (6.6). The structure of this program dictates the suggestive mode. Before

examining the function of the program. we consider some PDL syntax issues. Recall that program and procedure names are upper case and that keywords are upper-case boldface. **EXTERNAL** indicates that certain procedures that will be invoked are written externally (i.e. in different files) to the file containing this program. **DECLARE** is used to specify variable names (the convention here is that variable names are lower case) and to assign a type and scope for each variable. The only type shown here is **ASSOCIATION-LIST**, which was described earlier in this chapter as the primary data structure used with the production rule system. **LIST-OF** is used to form compound data types, here a list of association lists. The scope of all three variables is **LOCAL**, meaning that they are known only to this program and are available to other procedures and programs only if they are passed to those procedures and programs as arguments. **ARGUMENT** is also, then, another possible scope, along with **GLOBAL**, which means that variables are available to all procedures and programs that specify their names as **GLOBAL**.

Considering now the program function, note how the program is functionally divided by the idea of incorporation. First the list of objects is initialized. Then while the user wishes to continue designing, features and suggestions are generated with GENERATE-ALTERATION-STEPS. If a feature was successfully created (it is possible th... feature, and therefore no suggestions were created), then CHOOSE-ACTION ...termines the next course of action, particularly whether or not to incorporate each tentative alteration step. Both of these invoked procedures are primarily control routines and are discussed further below. They divide the architecture into two major branches, a generation branch and an incorporation branch. The generation branch will be described first.

Procedure GENERATE-ALTERATION-STEPS, shown in Figure (6.7) controls the user creation of, and system response to a feature input. Continuing to clarify some syntax, note that now "feature," "suggestion," and "objects" have scope

ARGUMENT since they were passed into this procedure. The **CASE** structure, quite simply, provides a way to choose and act upon one of many cases. The case statement in GENERATE-ALTERATION-STEPS appears to the user as menu on a text screen. Virtually all nongraphical interactivity is accomplished with menus.

Note that for a suggestive system with "N" features in a feature library, the system can invoke any one of N CONSTRUCT-FEATURE and GENERATE-SUGGESTIONS routines. The basic operation is to first allow the user to try to build a feature. If a feature was successfully constructed, then suggestions are generated from it. In each GENERATE-SUGGESTIONS routine, "feature" is used to motivate the generation of suggestions, "suggestion" is used to hold the association list representation of a suggestion, and "objects" is used to represent the entire design. Typical CONSTRUCT-FEATURE and GENERATE-SUGGESTIONS procedures will be considered in more detail.

A generic feature construction routine, which we call CONSTRUCT-FEATURE-x, is shown in figure (6.8). This procedure manages the user input of a particular type of feature and allows it to be repeated or aborted before the invocation of any extensive production rule programs. It invokes three procedures for the construction of each feature. GET-FEATURE-INFORMATION-x allows the user to input information required to build the feature. This information is stored in a local variable called "input-information" and in the case of a CAD suggestive system is a list of some type of geometry. Note that it is possible for the user to not input the information correctly, causing "input-information" to retain the value NIL. If "input-information" is not NIL, then PROCESS-FEATURE-INFORMATION-x creates an association list representation of the feature from the input information list. Finally, RENDER-FEATURE-INFORMATION generates a graphical image of the feature for the user to see. The GET-, PROCESS-, and RENDER- routines invoked are feature-specific and will not be described here.

It is necessary, however, to discuss the general operation of a typical GET-FEATURE-INFORMATION procedure. In the system developed here, the user inputs feature information directly onto the CAD model with a screen digitizing pen. Locations and dimensions are input in a freehand manner, causing a sketchpad type of interactivity that does not allow the input of exact feature parameters. The desire was to allow the quick input of a design idea rather than the creation of an accurate model. If precise input were required, different GET-FEATURE-INFORMATION routines could be easily developed. The available features, shown in Figure (6.9), are initial blocks, grooves, holes, steps, and chamfers. Additionally, symmetric sets of grooves, holes, steps, and chamfers can be created by making a single template feature and specifying the type of symmetry.

GENERATE-SUGGESTIONS-x, shown in Figure (6.10), is a generic suggestion generation procedure which invokes the rule based programming required to generate suggestions. It is convenient to break this code into two parts, PRE-SUGGESTIVE-RULES-x, and SUGGESTIVE-RULES-x. These two parts are actually procedures that contain rules as local variables and invoke the required rule-firing programs. The structures of rule sets that would be found in these procedures are discussed in Chapter Eight.

Jumping back up the calling chain to program SUGGEST1, we now begin to specify the incorporation branch of the architecture. Procedure CHOOSE-ACTION, shown in Figure (6.11), allows the user to choose what to do with the tentative geometry that has been created. If the system was able to generate one or more suggestions (i.e. suggestion not equal to NIL), then the choice is between keeping the feature keeping a particular suggestion (one suggestion is chosen from all possible prior to the invocation of CHOOSE-ACTION), or rejecting both, causing the design to remain unchanged. If no suggestion was generated, then the choice is between keeping and discarding the feature. The procedure INCORPORATE-AND-

PROCESS incorporates (see Chapter Five) the tentative geometry and processes the object list representation of the design by appropriately augmenting the list with the representation of the feature or suggestion. Note that if the user has chosen to incorporate a suggestion, the procedure ENSURE-VALIDITY is invoked to make sure that the entire design representation is valid. This is because the effect of incorporating a suggestion into the design representation is not precisely specified prior to the incorporation process. This issue is discussed in more detail in Chapter Eight.

Finally, procedure INCORPORATE-AND-PROCESS, shown in Figure (6.12), controls the incorporation of tentative alteration steps into the design. An alteration step, either feature or suggestion, enters the procedure as an argument. Procedure GET-PROPERTY-VALUE extracts the value of the "Type" property from the alteration step association list and stores it in the local variable "is-a." GET-PROPERTY-VALUE is an example of one of several *access* routines that facilitate the manipulation of the data structures. "Is-a" is used in the **CASE** statement to choose which feature-specific procedures to execute. POST-ALTERATION-RULES-x correctly modifies the alteration step association list and the list of objects before adding the alteration step as a new object. In a suggestive CAD system, INCORPORATE-ALTERATION-x makes the graphical representation geometrically valid. This is the routine that trims the lines in Figure (5.1). These last two routines are, of course, feature-specific. Finally, procedure AUGMENT adds the association list of the alteration step to the object list design representation. This augmentation process is specific to the way the programming language performs list processing.

Discussion

It is useful and appropriate to point out some deficiencies of this architecture. First, it is evident that many alteration-step-specific procedures exist. An obvious

question, then, is why not use a true "object-oriented" architecture, where the names of all such procedures are attached directly to the representation of the alteration step (see e.g. pages 239-243 of [Win.1]). A procedure like RENDER-FEATURE-INFORMATION-HOLE could be the value of a "Render-procedure" property of a hole association list. This would eliminate the need for procedures such as GENERATE-ALTERATION-STEPS and INCORPORATE-AND-PROCESS, that only choose which alteration-specific procedures to invoke. The user would simply invoke the desired feature and the system would extract and invoke the program property values at the appropriate times.

Another shortcoming of this system is that there is no heirarchical structure in the object list representation of the design. All objects are at the same level of detail: there is no distinction between primitive and complex objects. A consequence of this is that the association lists have no properties that would allow changes to one object to directly affect another object. Consider again Figure (5.2) and assume that there is a primitive feature of type "planar-surface." The groove's association list could have a "Made-of" property with a list of planar surfaces as the property value. The hole, on the other hand, could have an "End-surfaces" property with a list of two planar surfaces as the property value. Moving the groove to accept a suggestion would also move the *single* common planar surface. Simple procedures would be run to determine if this primitive object is related to any high level objects, showing that the hole would be affected. If the strategy is to move the hole with the groove, then the plane movement dictates the translation; if the strategy is to extend the hole to the new upper surface, the common planar surface must be deleted from the hole's "End-surfaces" property value and replaced with the new upper surface. Primitive features, then, could solve some of the geometric problems associated with making suggestions about incorporated objects. Presently, the features of the system only have a property that holds representations of the *points* used to construct the feature.

This is not enough information to determine the interactions between incorporated objects.

These deficiencies are primarily due to the emphasis of the research effort. We were interested in a *production rule* system that could be used to model the decision processes of design creation and improvement. Because the encoded intelligence is not directly linked to any particular alteration step data structure, general purpose intelligent routines that apply to the *entire* design representation can be easily employed (e.g. ENSURE-VALIDITY). Luby et al. describe a feature-oriented CAD system for the design and evaluation of aluminum castings, which suffers from neither of the deficiencies described above since it is a true object-oriented system [Lub.1]. Evaluation procedures are related to each feature and therefore tend to be more narrow in scope. An ideal system would be the integration of characteristics of both the production-rule and object-oriented approaches to create a system that could model the design decision processes and perform geometric reasoning.

The programming effort to create the intelligent CAD system described in this thesis was significant. Approximately 370 source code programs were written, including all rule based programs. Average program length is about 100 lines of source code. Although this might seem very large, it should be noted that there was some amount of "conceptual redundancy." Many of the programs related to a step feature. for example, are very much like the programs for a chamfer feature. Subsequent implementations of the ideas presented here might allow for some streamlining along with architectural improvements.

```
Given a feature:

FOR every rule

    FOR every object

        i := 1

        REPEAT

            Evaluate antecedent(i) of the rule using the:
                antecedent program specified
                feature property specified
                object property specified
                variable list specified

            i := i + 1

        UNTIL (an antecedent evaluates to "false") OR
              (all antecedents evaluate to "true")

        IF all antecedents evaluate to "true"

            THEN attach the object to the rule

    CONTINUE { over all the objects }

    IF any objects attached to the rule

        THEN add the attachment list to the conflict list

CONTINUE { over all the rules }
```

Figure 6.1: The match step of the match-resolve-execute cycle.

```
Given a feature:

FOR every attachment list

    FOR every object

        FOR every consequent

            Evaluate consequent(i) of the rule using the:
                consequent program specified
                feature property specified
                object property specified
                variable list specified

        CONTINUE { over all the consequents }

    CONTINUE { over all the objects }

CONTINUE { over all the attachment lists }
```

Figure 6.2: The execute step of the match-resolve-execute cycle.

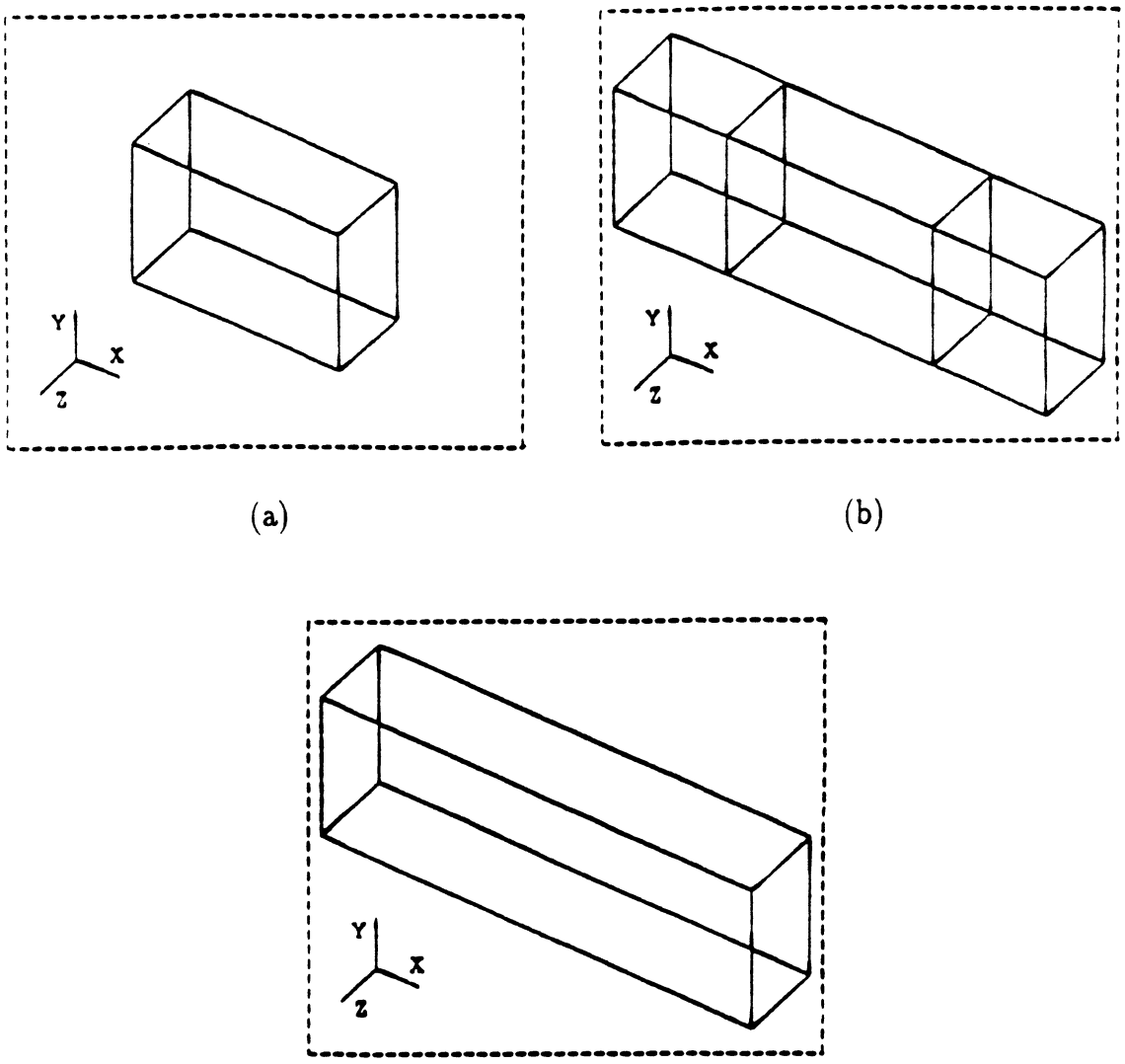


Figure 6.3: The effect of an example production. (a) User's initial feature input. (b) System's improved suggestion. (c) Suggestion incorporated into the design.

```
( "Suggest_long_from_cubic", ^
  ( ( "MATCHSTRING", "Type", "None", "BLOCK" ), ^
    ( "MATCHSTRING", "None", "Name", "DUMMY01" ), ^
    ( "MATCHSTRING", "Enclosure", "None", "CUBIC" ) ), ^
  ( ( "SUGGESTBLKS", "Side lengths", "None", "SUGLONG1" ), ^
    ( "MESSAGE", "None", "None", "MESSLONG1" ), ^
    ( "USERRESPONSE", "None", "None", "USERVAR" ) ), ^
  "Because   A long block is better than a cubic block " ), ^
```

(a)

If the feature type is 'block',
and the feature is compared to a dummy object,
and the enclosure of the feature is 'cubic',
Then suggest a block with 'long' side lengths,
and output a text message,
and obtain a response from the user,
Because a long block is better than a cubic block.

(b)

Figure 6.4: An example suggestive rule. (a) The DAL code rule that made the long-block suggestion. (b) Annotation of the rule.

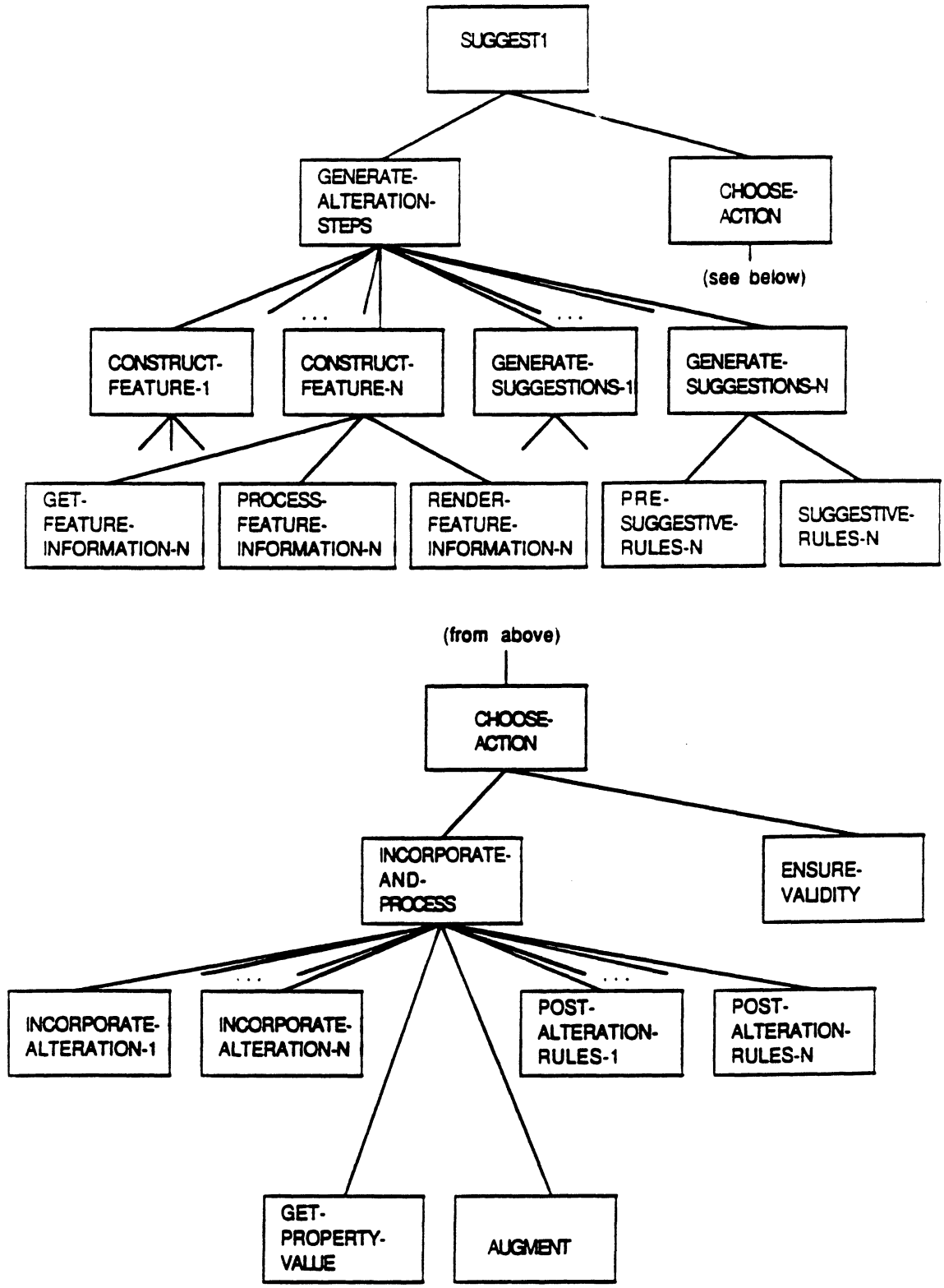


Figure 6.5: The structure of the intelligent suggestive CAD system.

```
PROGRAM SUGGEST1

    EXTERNAL    GENERATE-ALTERATION-STEPS
                CHOOSE-ACTION

    DECLARE     feature AS ASSOCIATION-LIST LOCAL
                suggestion AS ASSOCIATION-LIST LOCAL
                objects AS LIST-OF ASSOCIATION-LIST LOCAL

    BEGIN { SUGGEST1 }

        objects := NIL

        WHILE user wishes to continue DO

            feature := NIL
            suggestion := NIL

            BEGIN

                GENERATE-ALTERATION-STEPS ( feature, suggestion,
                                            objects )

                IF feature <> NIL

                    THEN CHOOSE-ACTION (feature, suggestion,
                                        objects )

            END

        CONTINUE

    END { SUGGEST1 }
```

Figure 6.6: The PDL description of program "SUGGEST1."

```
PROCEDURE GENERATE-ALTERATION-STEPS ( feature, suggestion, objects )

    EXTERNAL    CONSTRUCT-FEATURE-1
                CONSTRUCT-FEATURE-2
                    ...
                CONSTRUCT-FEATURE-N
                GENERATE-SUGGESTIONS-1
                GENERATE-SUGGESTIONS-2
                    ...
                GENERATE-SUGGESTIONS-N

    DECLARE     feature AS ASSOCIATION-LIST ARGUMENT
                suggestion AS ASSOCIATION-LIST ARGUMENT
                objects AS LIST-OF ASSOCIATION-LIST ARGUMENT

    BEGIN { GENERATE-ALTERATION-STEPS }

        CASE
            WHEN user desires feature-1
                BEGIN
                    CONSTRUCT-FEATURE-1 ( feature )
                    IF feature <> NIL
                        THEN GENERATE-SUGGESTIONS-1
                            ( feature, suggestion, objects )
                END
            WHEN user desires feature-2
                BEGIN
                    CONSTRUCT-FEATURE-2 ( feature )
                    IF feature <> NIL
                        THEN GENERATE-SUGGESTIONS-2
                            ( feature, suggestion, objects )
                END
            ...
            WHEN user desires feature-N
                BEGIN
                    CONSTRUCT-FEATURE-N ( feature )
                    IF feature <> NIL
                        THEN GENERATE-SUGGESTIONS-N
                            ( feature, suggestion, objects )
                END
        ENDCASE

    END { GENERATE-ALTERATION-STEPS }
```

Figure 6.7: The PDL description of procedure "GENERATE-ALTERATION-STEPS."

```
PROCEDURE CONSTRUCT-FEATURE-x ( feature )

    EXTERNAL    GET-FEATURE-INFORMATION-x
                PROCESS-FEATURE-INFORMATION-x
                RENDER-FEATURE-INFORMATION-x

    DECLARE     feature AS ASSOCIATION-LIST ARGUMENT
                input-information AS LIST-OF GEOMETRY LOCAL

    BEGIN { CONSTRUCT-FEATURE-x }

        input-information := NIL

        REPEAT

            GET-FEATURE-INFORMATION-x ( input-information )

            IF input-information <> NIL

                THEN

                    BEGIN

                        PROCESS-FEATURE-INFORMATION-x
                            ( input-information, feature )
                        RENDER-FEATURE-INFORMATION-x
                            ( input-information )

                        CASE
                            WHEN user desires to keep feature
                                Do nothing
                            WHEN user desires to redo feature
                                Delete tentative geometry
                            WHEN user desires to abort feature
                                Delete tentative geometry
                        ENDCASE

                    END

        UNTIL ( user is satisfied with feature input ) OR
              ( user has aborted feature input efforts )

    END { CONSTRUCT-FEATURE-x }
```

Figure 6.8: The PDL description of procedure "CONSTRUCT-FEATURE-x."

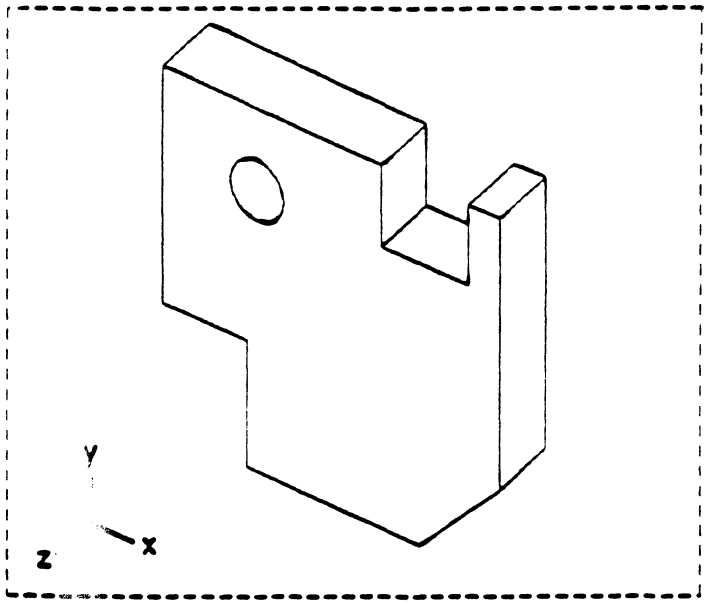


Figure 6.9: Available features: block, groove, hole, step, chamfer.

```
PROCEDURE GENERATE-SUGGESTIONS-x ( feature, suggestion, objects )

    EXTERNAL    PRE-SUGGESTIVE-RULES-x
                SUGGESTIVE-RULES-x

    DECLARE     feature AS ASSOCIATION-LIST ARGUMENT
                suggestion AS ASSOCIATION-LIST ARGUMENT
                objects AS LIST-OF ASSOCIATION-LIST ARGUMENT

    BEGIN { GENERATE-SUGGESTIONS-x }

        PRE-SUGGESTIVE-RULES-x ( feature, objects )
        SUGGESTIVE-RULES-x ( feature, suggestion, objects )

    END { GENERATE-SUGGESTIONS-x }
```

Figure 6.10: The PDL description of procedure "GENERATE-SUGGESTIONS-x."

```
PROCEDURE CHOOSE-ACTION ( feature, suggestion, objects )

    EXTERNAL    INCORPORATE-AND-PROCESS
                ENSURE-VALIDITY

    DECLARE     feature AS ASSOCIATION-LIST ARGUMENT
                suggestion AS ASSOCIATION-LIST ARGUMENT
                objects AS LIST-OF ASSOCIATION-LIST ARGUMENT

    BEGIN { CHOOSE-ACTION }

        IF suggestion <> NIL
            THEN
                CASE
                    WHEN user desires to incorporate suggestion
                        BEGIN
                            Delete feature geometry
                            INCORPORATE-AND-PROCESS
                                ( suggestion, objects )
                            ENSURE-VALIDITY ( objects )
                        END
                    WHEN user desires to incorporate feature
                        BEGIN
                            Delete suggestion geometry
                            INCORPORATE-AND-PROCESS
                                ( feature, objects )
                        END
                    WHEN user desires to discard both feat and sug
                        BEGIN
                            Delete suggestion geometry
                            Delete feature geometry
                        END
                ENDCASE
            ELSE
                CASE
                    WHEN user desires to incorporate feature
                            INCORPORATE-AND-PROCESS
                                ( feature, objects )
                    WHEN user desires to discard feature
                            Delete feature geometry
                ENDCASE

    END { C   OSE-ACTION }
```

Figure 6.11: The PDL description of procedure "CHOOSE-ACTION."

```
PROCEDURE INCORPORATE-AND-PROCESS ( alteration-step, objects )

    EXTERNAL    GET-PROPERTY-VALUE
                AUGMENT
                INCORPORATE-ALTERATION-1
                INCORPORATE-ALTERATION-2
                    . . .
                INCORPORATE-ALTERATION-N
                POST-ALTERATION-RULES-1
                POST-ALTERATION-RULES-2
                    . . .
                POST-ALTERATION-RULES-N
    DECLARE     alteration-step AS ASSOCIATION-LIST ARGUMENT
                objects AS LIST-OF ASSOCIATION-LIST ARGUMENT
                is-a AS CHARACTER-STRING LOCAL

    BEGIN { INCORPORATE-AND-PROCESS }

        is-a := GET-PROPERTY-VALUE ( alteration-step, "Type" )
        CASE
            WHEN is-a = "Feature-1"
                BEGIN
                    POST-ALTERATION-RULES-1
                        ( alteration-step, objects )
                    INCORPORATE-ALTERATION-1
                END
            WHEN is-a = "Feature-2"
                BEGIN
                    POST-ALTERATION-RULES-2
                        ( alteration-step, objects )
                    INCORPORATE-ALTERATION-2
                END
            . . .
            WHEN is-a = "Feature-N"
                BEGIN
                    POST-ALTERATION-RULES-N
                        ( alteration-step, objects )
                    INCORPORATE-ALTERATION-N
                END
        ENDCASE
        objects := AUGMENT ( objects, alteration-step )

    END { INCORPORATE-AND-PROCESS }
```

Figure 6.12: The PDL description of procedure "INCORPORATE-AND-PROCESS."

CHAPTER VII

SUGGESTIVE INTERACTIVITIES

Introduction

With an intelligent suggestive CAD system architecture specified, it is now appropriate to examine some of the human-computer interactivity issues. There are many ways to make suggestions. In this chapter we will choose two suggestive interactivities (or suggestive modes) to use in the experiment discussed in Chapter Nine. The rationale for this decision will be discussed and examples of each interactivity will be presented.

Two Interactivities

Our goal in providing suggestions during preliminary design is to provide information to designers that will cause them to obtain better design ideas. These better ideas will result in designs that are superior with respect to some concurrent engineering concern. Recall that one of the major findings of a study done by Ullman et al. ([Ull.1]) was that designers will often pursue a single initial design idea, refusing to abandon it regardless of how poor it proves to be. By providing suggestions we

hope to stimulate the creation of better ideas and promote the abandonment of bad ones. An important issue is the manner in which suggestions are presented to the designer. What type of suggestive mode is most helpful in preliminary design?

Consider first a nonsuggestive mode, shown as a flowchart in Figure (7.1). While the user wants to continue designing, they input a feature. If they are satisfied with the feature, they incorporate it, and if they are not, they discard it. The addition of each feature constitutes a "design cycle," and features are added until the design is complete. Recall from Chapter One that the system we propose will provide improvement suggestions only *in response* to user actions. The nonsuggestive mode, therefore, can be augmented by providing suggestions either during or after the design process. If the suggestions are provided *during* the design process, they will be generated at some point in the feature input loop of Figure (7.1). If they are provided *after* the design process, they will be generated after the user has decided to stop designing. Suggestions cannot be output before the design process commences because the user has completed no design steps. During versus after will be the major distinction between the two suggestive modes used in the experiment.

An implementation of either type of mode raises interesting issues. A *during* mode would generate suggestions at each design cycle, bringing improvement ideas to the user while they are designing. One might expect that this would result in better designs with fewer redesigns and design revisions since designers can alter their design ideas before the design is complete. On the other hand, because later suggestions may stimulate ideas that were not evident from earlier suggestions, redesigns may be necessary. An additional question is whether or not designers will find a *during* mode distracting. Providing suggestions during the design process might actually be detrimental if the suggestions interrupt the designers' train of thought and prevent them from concentrating on the design task.

An *after* mode would not generate suggestions until after the designer is finished.

Once an initial effort is completed, the system would provide suggestions that would motivate an improved redesign. A redesign step is an expected part of the overall design process. An interesting question is whether or not designers will be willing to undertake a redesign effort. In the process of completely specifying a design with the CAD system, they may become committed to the design idea. Redesign suggestions may not be welcome, and therefore might be ignored.

The following two sections describe the *during* and *after* interactivities that were implemented with the architecture described in Chapter Six. Two limitations should be noted. First, both interactivities make suggestions with respect to one alteration step at a time. A suggestion, for example, would not propose to alter a feature input *and* move one of the incorporated objects. The rule-based suggestive programs described in Chapter Eight are based on the idea of generating suggestions from one alteration step only. Second, suggestions may be generated that alter incorporated objects (particularly in the *after* mode), but these suggestions can only be viewed. Incorporated objects cannot be altered.

Suggestions During

Introduction

The particular *during* mode implemented is shown in flowchart form in Figure (7.2). Suggestions are generated from each feature input. Features and suggestions are therefore both present on the CAD screen. These suggestions will propose various alterations to the feature input and will not attempt to modify any incorporated objects. Because of this, the suggestions themselves can be incorporated into the design.

Example

We present an extended example to give some idea of typical suggestions that would be generated. The figures in this example and the *after* example below are from actual system output. For both the *during* and *after* cases, the suggestions appear on a graphics screen and accompanying text explanations appear on an alphanumeric screen.

Figures (7.3) through (7.5) show the first design cycle. Assume the user begins with the block shown in (7.3a) and then builds the groove feature in (7.3b). The system will analyze the design quality with the feature input and will then formulate a list of suggestions. In this case, since the unaltered block was 180-degree symmetric about all three axes, the suggestions will attempt to maintain some symmetry. Figure (7.3c) shows how three-axis symmetry is obtained by duplicating the groove in three other locations; (7.3d) obtains three-axis symmetry by first moving the groove and then duplicating it in one other location; (7.4e) obtains symmetry about the X axis by duplicating in one other location; (7.4f) obtains Y-axis symmetry by duplicating once, and (7.4g) obtains Y-axis symmetry by moving the groove; (7.4h) obtains Z-axis symmetry by duplicating the groove once. None of the suggestions are impossible, so all will be created. Heuristic analysis of the design quality with each of the suggestions produces the interesting result that symmetry about the X axis is best, followed by symmetry about the Y axis, followed by three-axis symmetry, followed by symmetry about the Z axis.[1] This makes (7.4e) the best suggestion, going against the common intuition that maintaining three-axis symmetry is optimal. This unexpected Boothroyd-Dewhurst ranking is caused by the dimensions of the overall block and illustrates the type of design suggestion that could help even a more experienced designer. Figure (7.5i) shows the suggestion (7.4e) incorporated into the design.

---

[1] Heuristic analysis of design quality and the ordering of the suggestions is discussed in Chapter Eight.

Figures (7.6) and (7.7) illustrate the second design cycle. Figure (7.6a) shows the user's feature input of a hole. Design quality analysis yields a low quality score because the orientation of the entire design is defined by the orientation of the hole, and the hole is fairly inaccessible to standard filtering and orienting devices. In comparison to a groove, for example, it only borders two of the original block faces while a groove borders three. Since the design prior to the hole feature input was symmetric only about the X axis, only suggestions that are symmetric about the X axis or all three axes are necessary. Indeed, note that deriving a suggestion that is Y- or Z- axis symmetric, when the prior total design was X symmetric, will yield a very poor design because two main features will be required to orient the part. The suggestions depicted in (7.6b) and (7.6c) cause an X-axis symmetric hole: (7.6b) by duplicating the hole and (7.6c) by moving the hole; (7.6d) provides a three-axis symmetric hole by duplicating the hole three times, and (7.7e) by moving and duplicating once. The suggestion of (7.6d) is eliminated because two of the holes required clash with two of the grooves. When analyzing the design quality and ranking the suggestions in situations where single-axis symmetry existed prior to the feature input, the following heuristic is used: suggestions that cause the existing symmetry are preferrable to suggestions that offer three-axis symmetry. A suggestion that is three-axis symmetric within itself is a neutral addition to the design. Other incorporated objects that are not three-axis symmetric must be used to orient the design, thereby preventing the suggestion from improving the design. A three-axis symmetric suggestion, on the other hand, cannot decrease the design quality. Assume the user rejects all suggestions and incorporates the initial feature input. This is shown in figure (7.7f).

Figure (7.8) shows a third and final design cycle. Figure (7.8a) shows that the user has input another hole feature, parallel to the earlier hole but larger. Analyzing the quality of the design indicates no decrease in quality, since the design was already

asymmetric due to a hole. In cases such as this where the design was previously asymmetric, the suggestions generated will show efforts to reduce the number of asymmetric incorporated objects. Figure (7.8b) shows the only example possible here, moving the hole feature and decreasing its size.[2] Figure (7.8c) shows the suggestion incorporated into the design.

Discussion

Other *during* suggestive modes are possible. Figure (7.9) shows a mode in which suggestions are generated after the incorporation process. The idea here is that the system makes suggestions that give the user some idea of what to do with the next feature input. If some of these suggestions propose altering objects, however, they cannot be incorporated. The mode shown in Figure (7.2) was implemented because it only alters the feature input, thereby creating suggestions that can be incorporated.

It should be noted that the implemented *during* mode emphasizes the order of feature input. The early features and suggestions derived from them can influence later suggestions. This was demonstrated by the suggestions that maintain existing symmetry. If there is any relative importance among the features, the more important ones should be created earlier.

Suggestions After

Introduction

---

[2] The system will actually suggest several different modifications to the hole feature that will cause several different symmetries for the hole. The system warns the user to make sure that the symmetry suggested does not cause a design that will require two features for orientation (as in Figure (3.1c)). The system does not analyze the potential design that would result from acceptance of one of these suggestions.

An *after* interactivity, as shown in Figure (7.10), can be thought of as a design postprocess. After an initial design is completed, it is analyzed and improvement suggestions are generated. The user will then view and consider the suggestions. Since the suggestions involve incorporated objects, they cannot be incorporated. Using ideas obtained from the suggestions, the user performs a redesign.

The extended example of the *during* mode presented above shows three basic suggestion strategies related to optimizing the symmetry of a design. Prior to a feature input, if a design was symmetric about all three axes, the system will generate suggestions for all symmetries (three-axis, X-, Y-, or Z-) and rank them for the user. Thus, the system tries to maintain *some* symmetry. If the design was symmetric about one axis only, the system makes suggestions that maintain the *same* symmetry. If the design was asymmetric, the system will try to *reduce asymmetry.* These are not the only strategies that were encoded. Others, for example, propose altering the dimensions of the initial block, and others warn the designer if the design will require two features for orientation. The strategies related to optimizing the symmetry, however, were the most frequently used in a typical design session. The desire was to use these same three strategies in the *after* mode. This would cause both modes to generate similar suggestions, which would be useful in the experiment. Our desire in the experiment was to only compare differences due to *when* the suggestions are output. Other differences between the two modes should be minimized. An added convenience is that the same suggestive rule bases could be used for each mode.

A problem arises, however, when suggestions are generated from a single alteration step in the *after* mode. To use the same rules in the *after* mode, a single incorporated object is "extracted" from the design and treated as if it were an added feature in the *during* mode described above. The remainder of the design is treated as the total set of incorporated objects. Often, however, when a design is completed, it is difficult for the suggestive strategies described above to generate any suggestions

from the extracted object. Consider the design shown in Figure (7.11). Extracting any object still leaves a design with no symmetry, so no symmetric suggestions of any kind will be made. If one of the grooves is extracted, it can be "matched" with the other groove in an effort to reduce asymmetry. The remaining objects, however, will still cause an asymmetric design after the match. In a design that was completed with no consideration given to assembly issues, this type of "gridlock" situation is not uncommon.

To get around this problem, the object representation is limited to the original block only. Suggestions are made with respect to each object, in the order they were incorporated into the design. If the original block is being considered, then suggestions are made about the block. If one of the other objects is being considered, then suggestions are made as if that object were the only object in the block. The following example will show the effect of this method.

Example

Figure (7.12a) shows the completed design prior to the generation of suggestions. This is a cubic block with an asymmetric groove and hole. Suggestions are first made with respect to the initial block. Figure (7.12b) shows that the first and best block suggestion, as was mentioned in Chapter Six, is to flatten the block. If that is not possible, figure (7.12c) is a suggestion to lengthen the block. Note how the block is modified as if the other features were not there.

Figures (7.13) and (7.14) show suggestions generated from the groove, similar to the way they were for the *during* case described above. In this situation, however, there is a different order for the suggestion quality. Again, X-axis symmetry is best, followed by three-axis symmetry, followed by Y-axis symmetry, followed by Z-axis symmetry. The difference in order is due to the different block dimensions. Some

groove suggestions naturally interfere with the hole because the hole is not considered to be part of the design.

Figures (7.15) and (7.16) show suggestions generated from the hole. These are similar to the suggestions generated from the groove, except that there are two suggestions symmetric about the X axis and only one symmetric about the Y axis. In this case, the suggestion quality order is: three-axis symmetry is best, followed by Y-axis symmetry, followed by Z-axis symmetry, followed by X-axis symmetry.

Discussion

In contrast to the *during* mode, this mode emphasizes all objects of the design, regardless of the order in which they were input. All objects are treated as if they were the first to be built. Additionally the suggestions are displayed along with the objects that are considered not to exist. This has the effect of displaying fundamental suggestions on a detailed design.

Discussion

We have chosen to vary *when* the suggestions are made to derive two different suggestive modes for the later experiment. Other variations are certainly possible.

One possibility is to consider mandatory versus optional suggestions. Should users be forced to see suggestions? Since we are interested in the influence of suggestions upon designers, all initial test interactivities should have mandatory suggestions. Users should not be able to bypass suggestions, even if they believe they need no design assistance.

Another possibility is to allow the use of a subset of the design representation when making suggestions. It was found that this was necessary for the *after* case: the design representation was limited to the block only. More generally, some subset

of the objects could be chosen prior to generating suggestions. Consider the design and design representation shown in Figure (7.17), which is composed of five objects: a block, groove, hole, step, and chamfer. If we omit the hole, step, and chamfer, the design might appear as shown by the bold lines in Figure (7.4a). Additionally, the design representation would be given by the objects in bold type in Figure (7.4b). Turning off some objects could allow an artificial reordering of the feature input that created the design. Omitting objects that were incorporated earlier would cause suggestions that are influenced only by objects that were incorporated later. In such situations it might be beneficial to remove the omitted objects from view or deemphasize them somehow (e.g. as shown in Figure (7.17a)). A system that allowed this capability would be more difficult for users to learn since not only must features be built, but some objects must be omitted, requiring an even more complicated human-computer interface. This is perhaps best done when a group of users gains experience with an initial implementation.

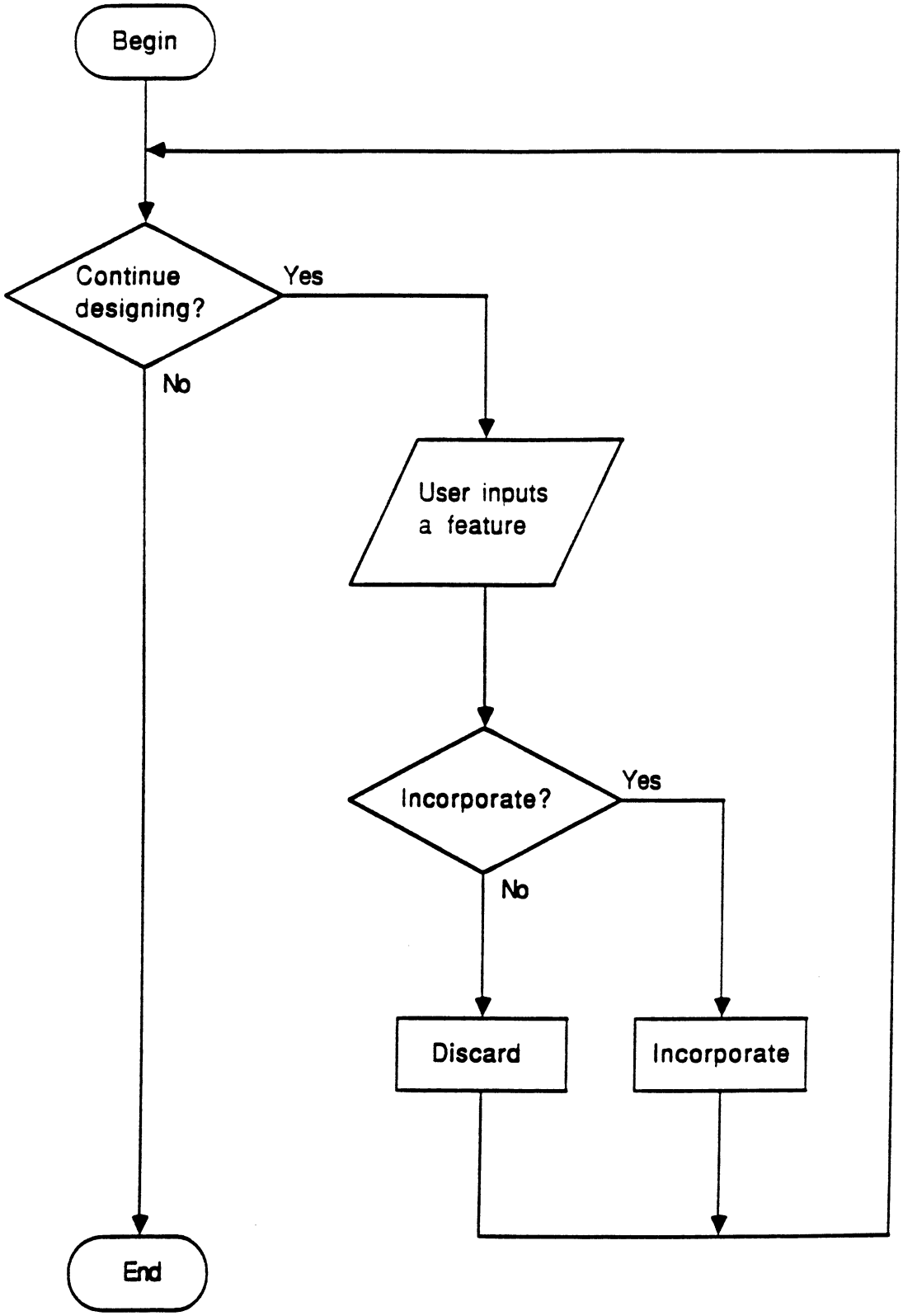


Figure 7.1: The nonsuggestive design process.

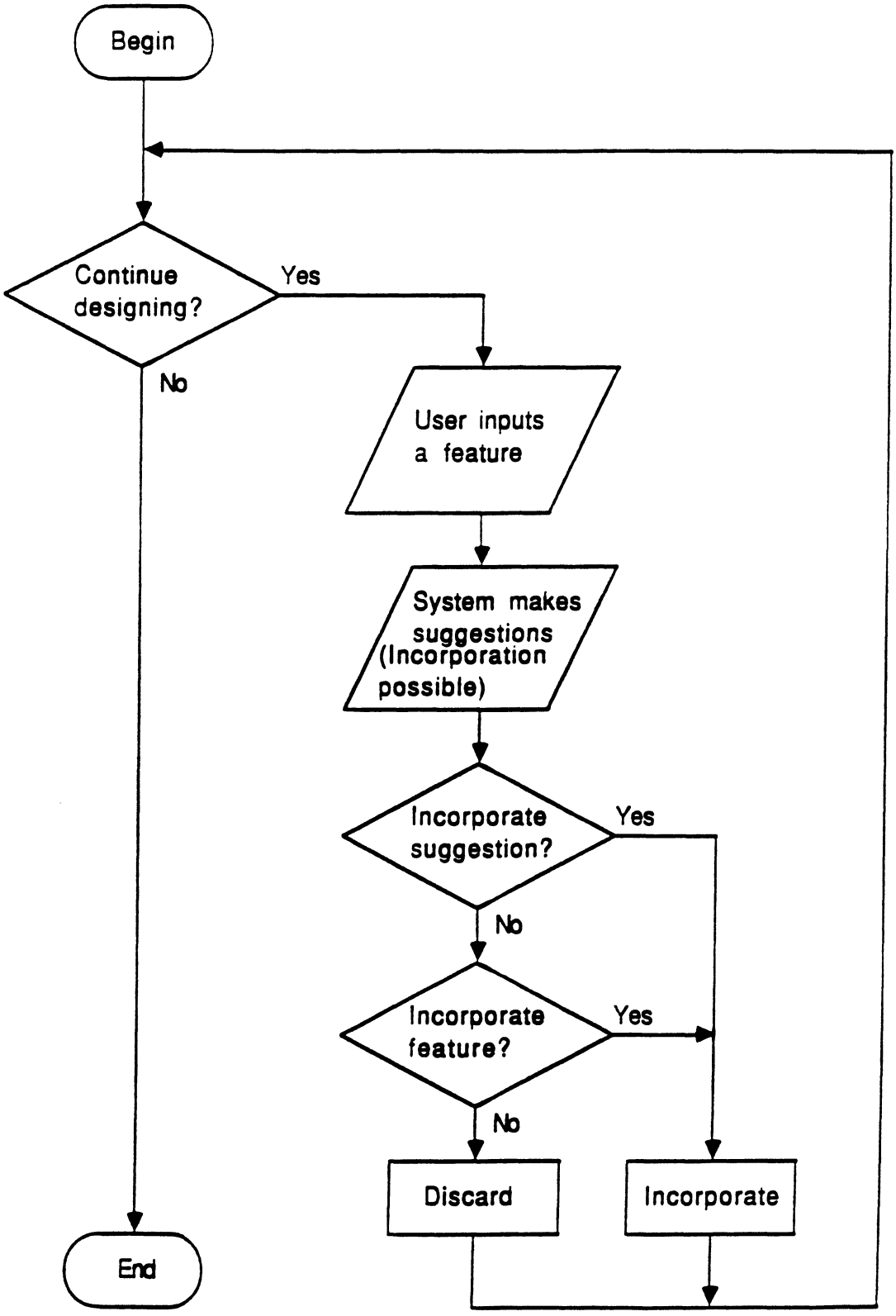


Figure 7.2: The implemented *during* suggestive mode.

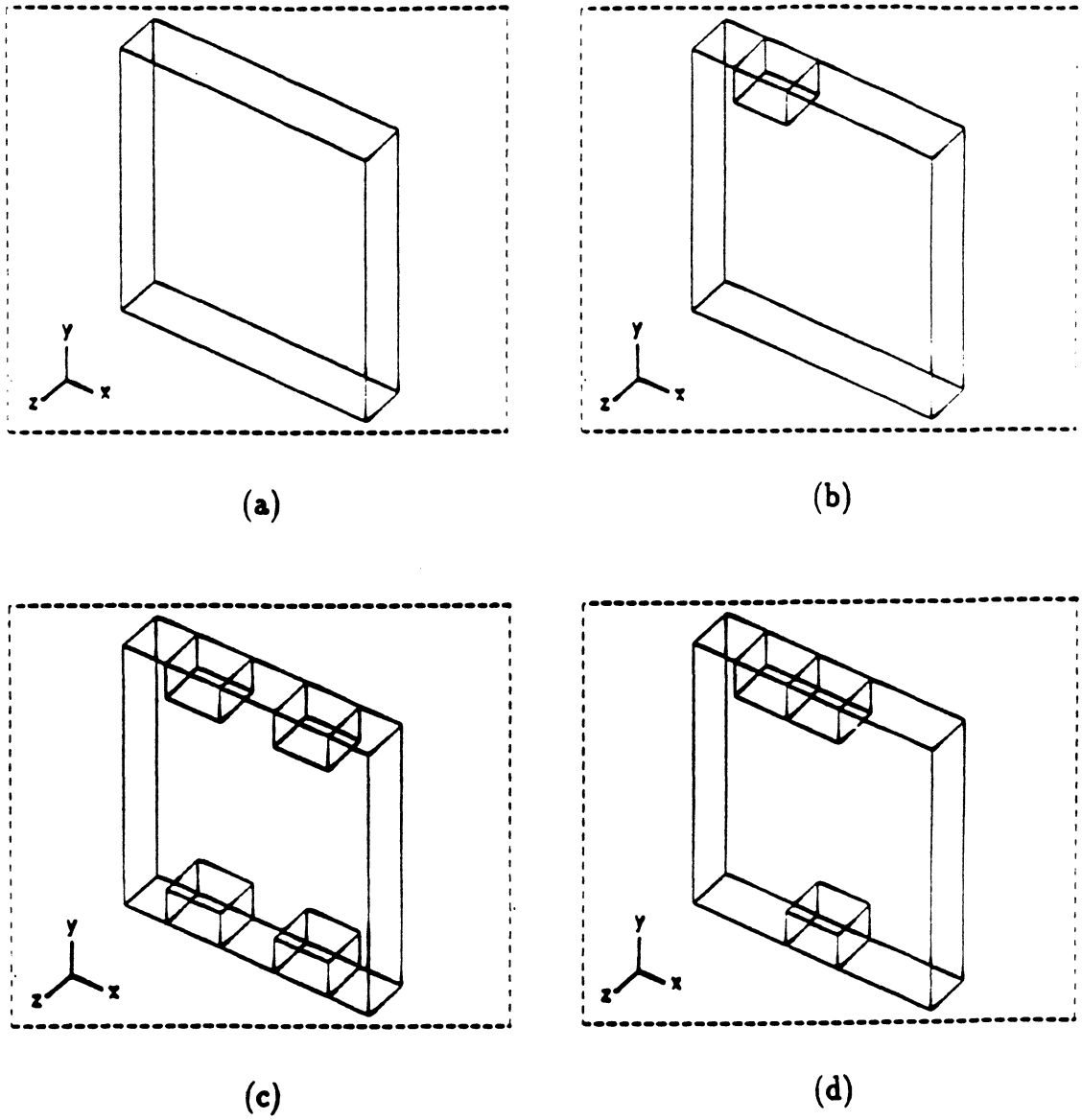
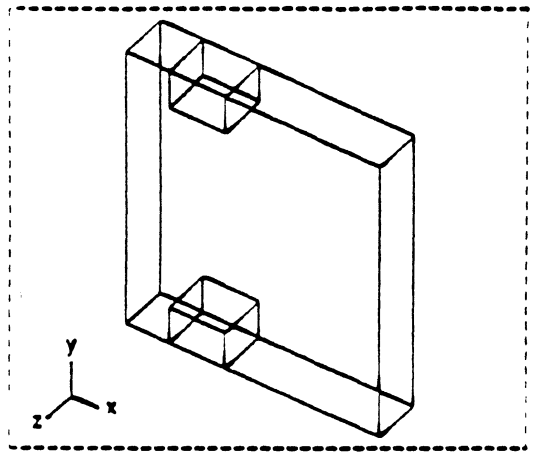
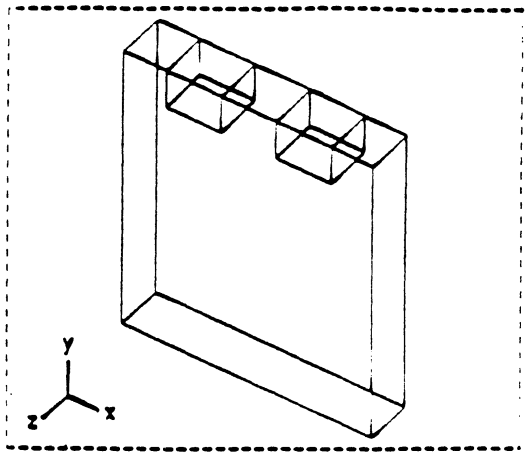


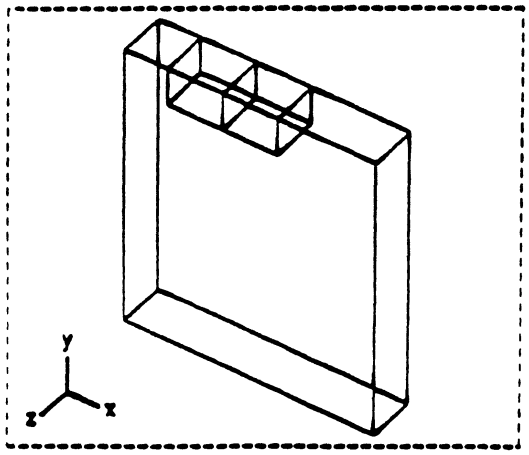
Figure 7.3: The first design cycle of the *during* case. (a) The design prior to the feature input. (b) A groove feature input. (c) First suggestion to obtain three-axis symmetry. (d) Second suggestion to obtain three-axis symmetry.



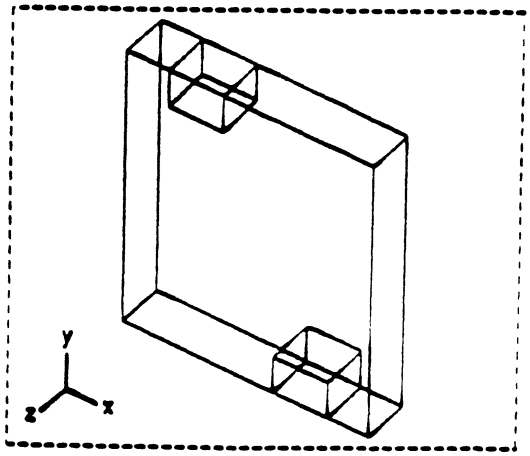
(e)



(f)

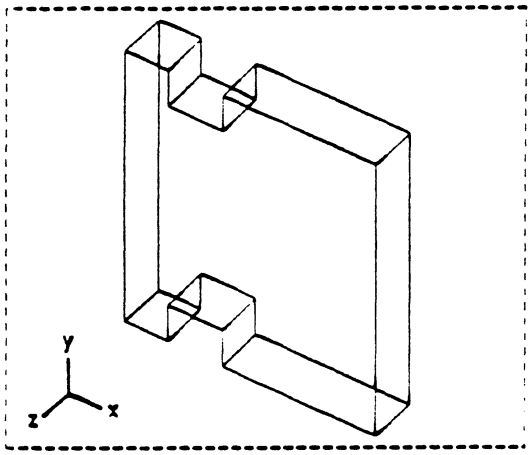


(g)



(h)

Figure 7.4: The first design cycle of the *during* case continued. (e) Suggestion to obtain X-axis symmetry. (f) First suggestion to obtain Y-axis symmetry. (g) Second suggestion to obtain Y-axis symmetry. (h) Suggestion to obtain Z-axis symmetry.



(i)

Figure 7.5: The first design cycle of the *during* case concluded. Suggestion incorporated into the design.

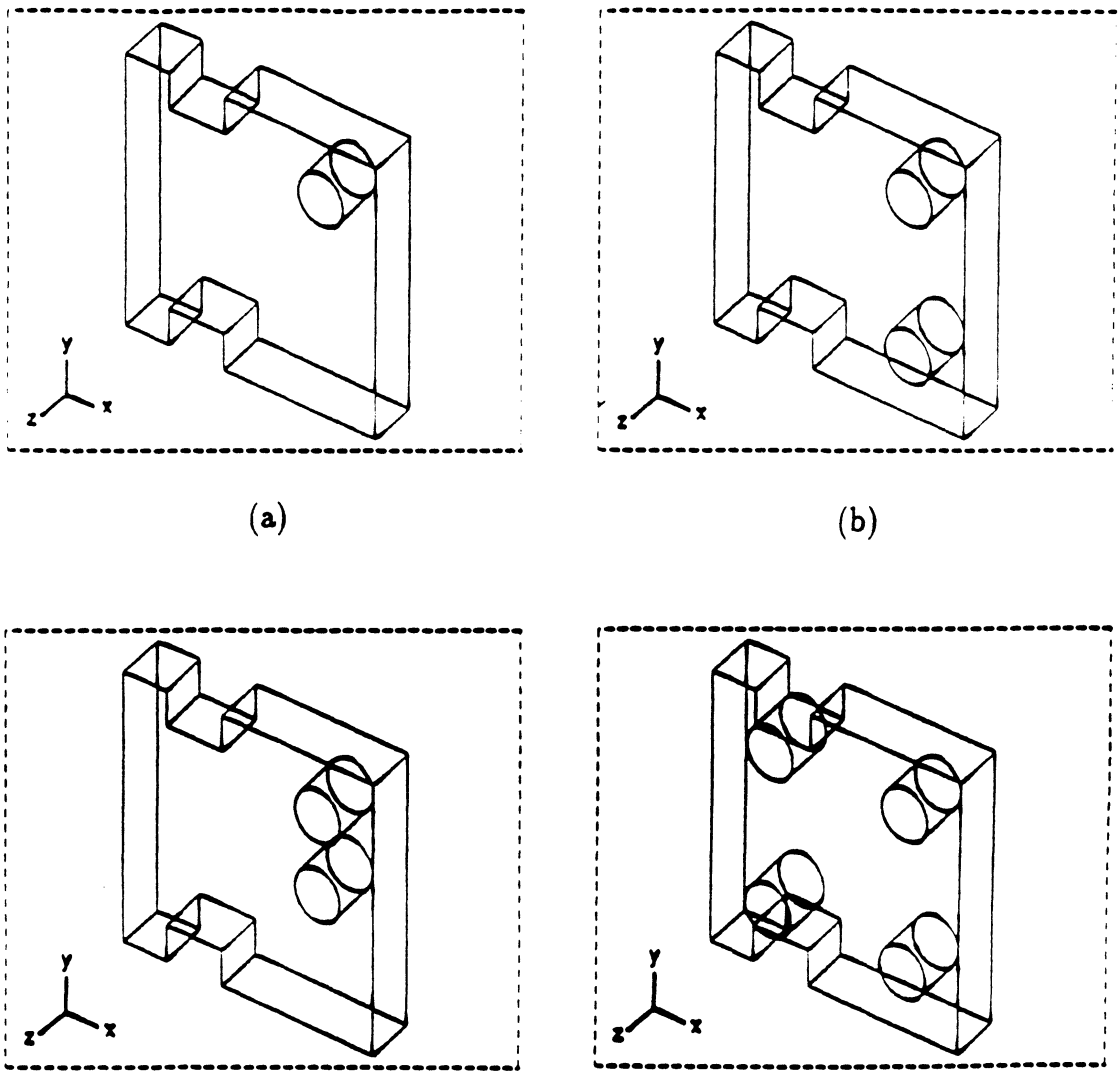
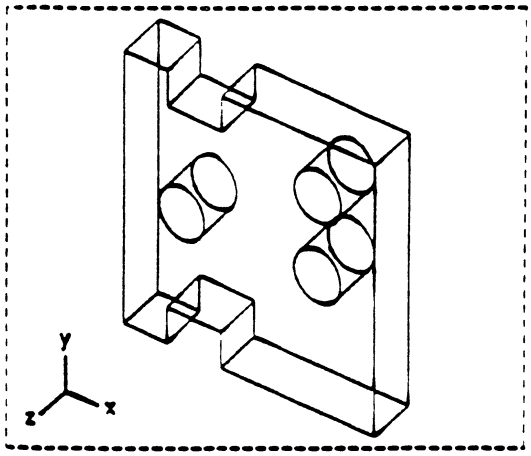
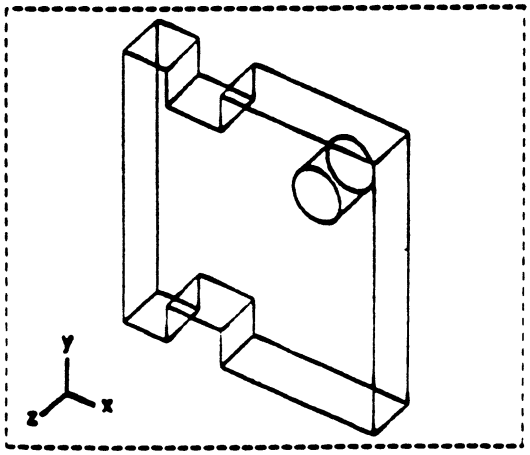


Figure 7.6: The second design cycle of the *during* case. (a) A hole feature input. (b) First suggestion to produce and X-axis symmetric hole. (c) Second suggestion to produce an X-axis symmetric hole. (d) First suggestion to produce a three-axis symmetric hole.



(e)



(f)

Figure 7.7: The second design cycle of the *during* case concluded. (e) Second suggestion to produce a three-axis symmetric hole. (f) Feature incorporated into the design.

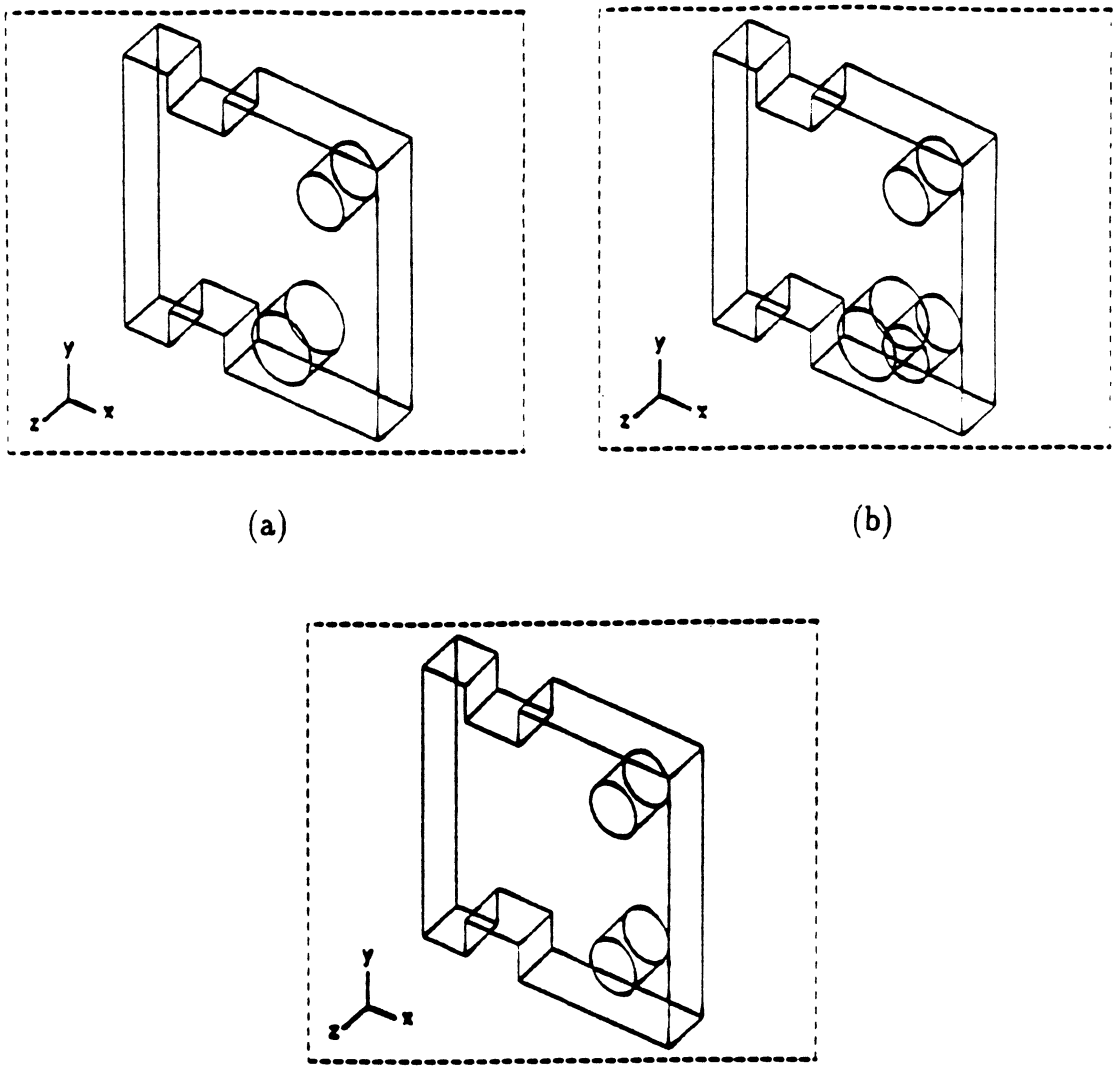


Figure 7.8: The third design cycle of the *during* case. (a) A hole feature input. (b) A suggestion to reduce asymmetry. (c) Suggestion incorporated into the design.

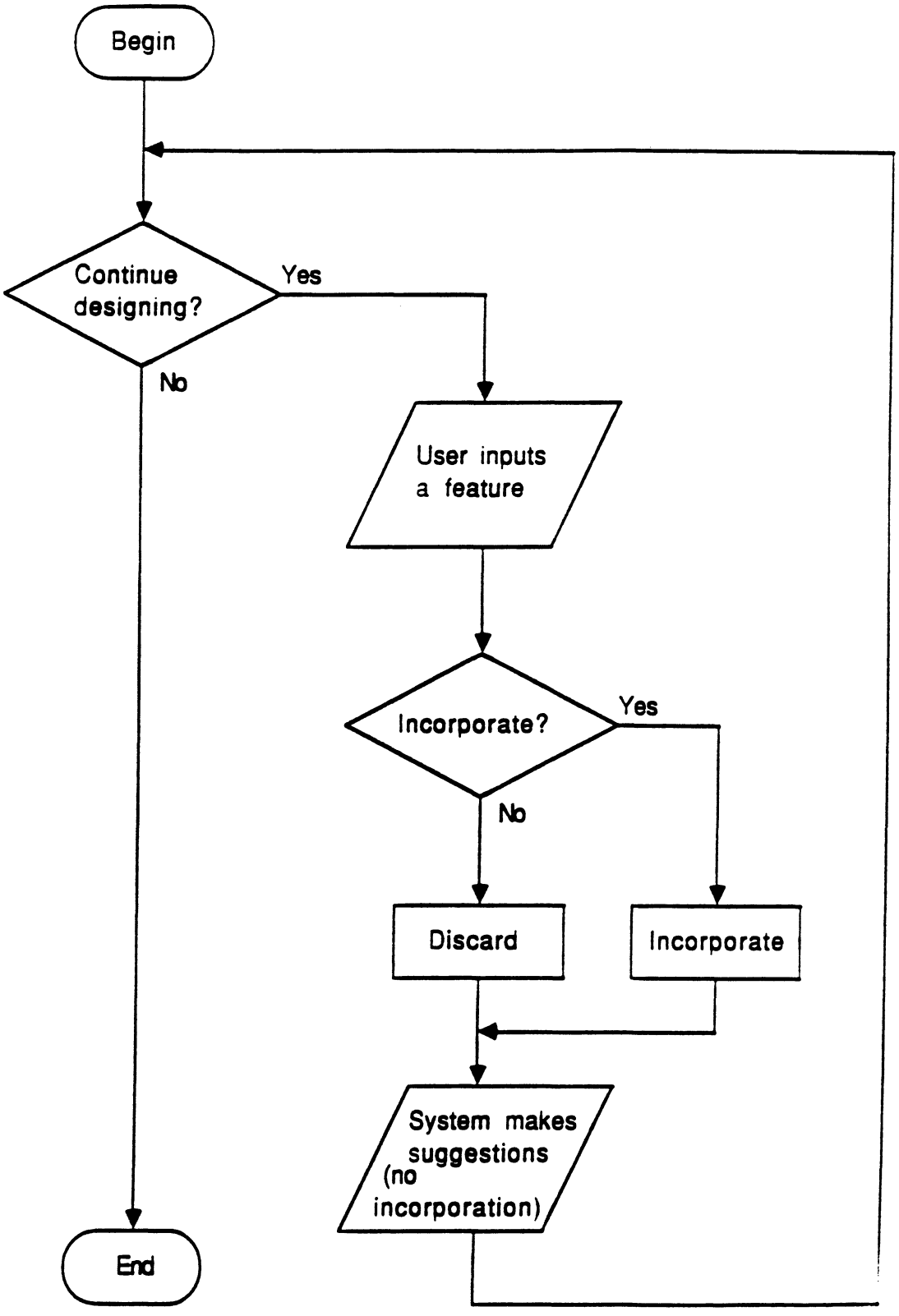


Figure 7.9: Another possible *during* suggestive mode.

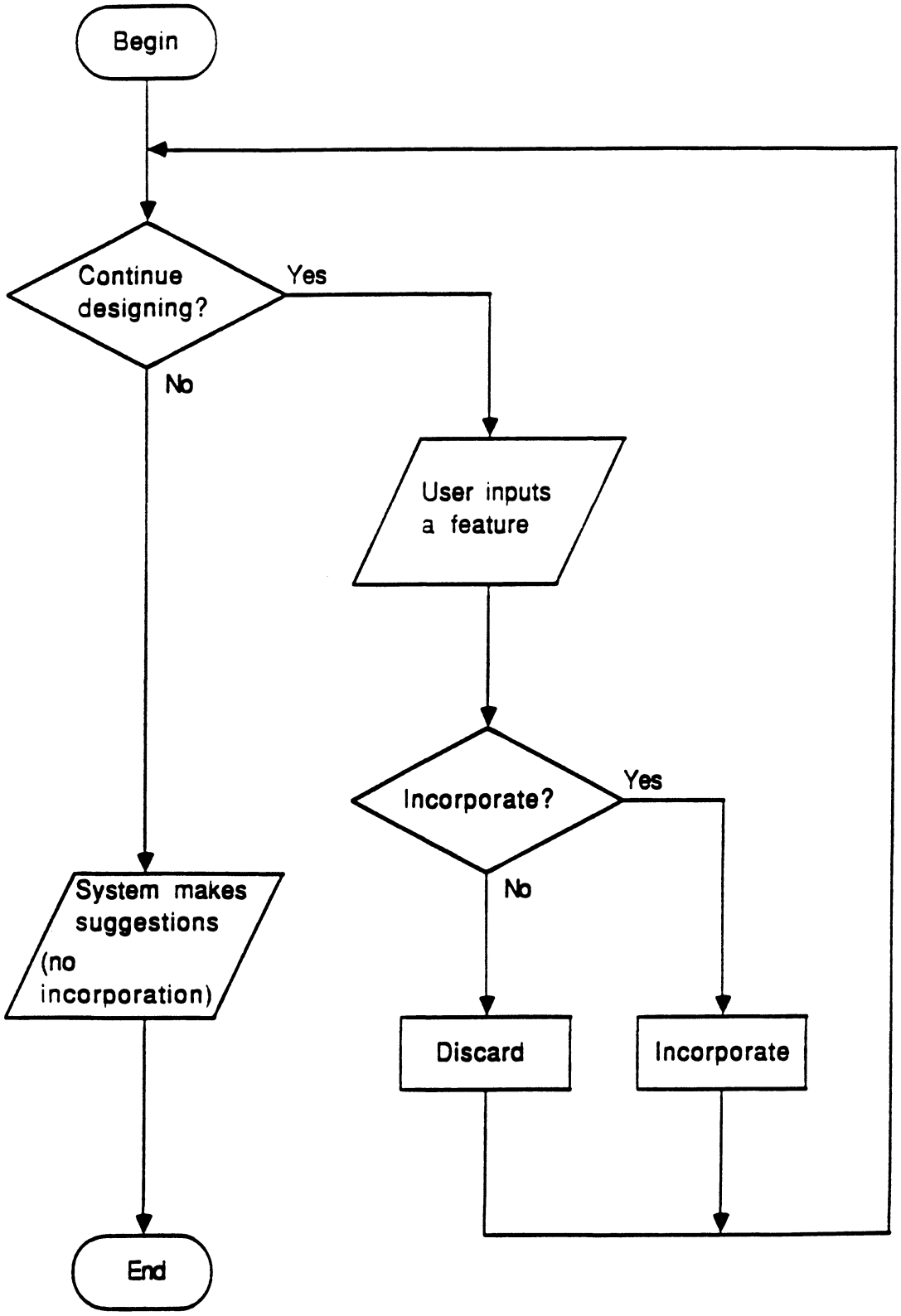


Figure 7.10: The *after* suggestive mode.

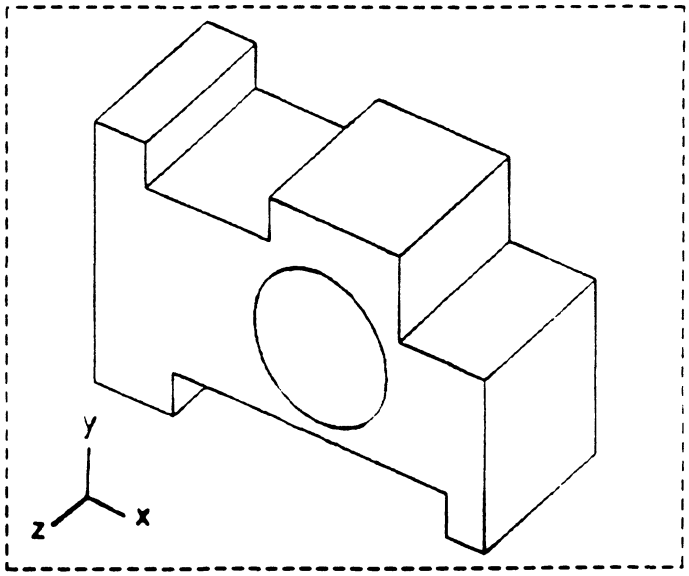


Figure 7.11: A completed design that that would allow generation of very few suggestions in an *after* suggestive mode.

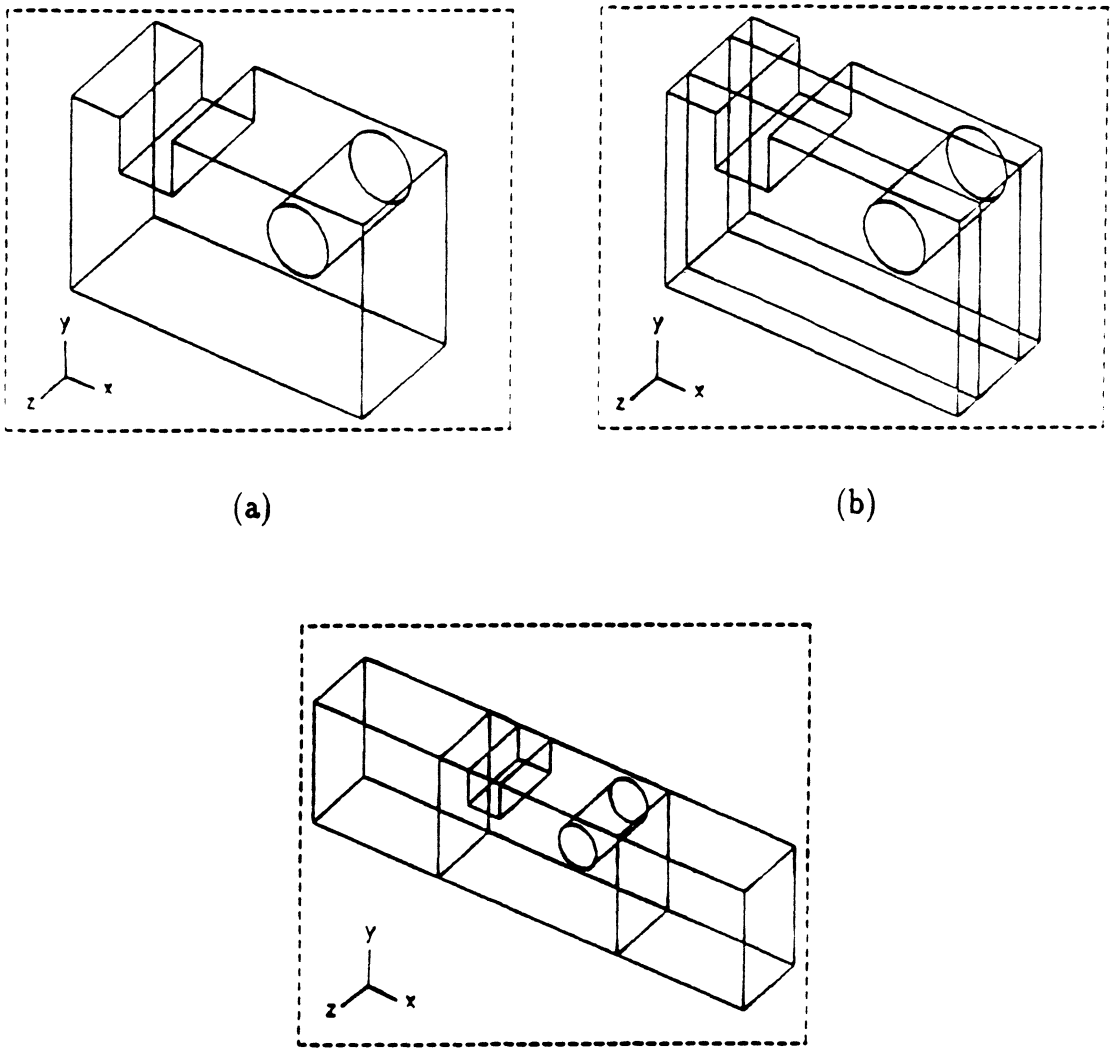


Figure 7.12: Suggestions for the block in the *after* case. (a) The completed design before suggestions. (b) Suggestion to make the block flat. (c) Suggestion to make the block long.

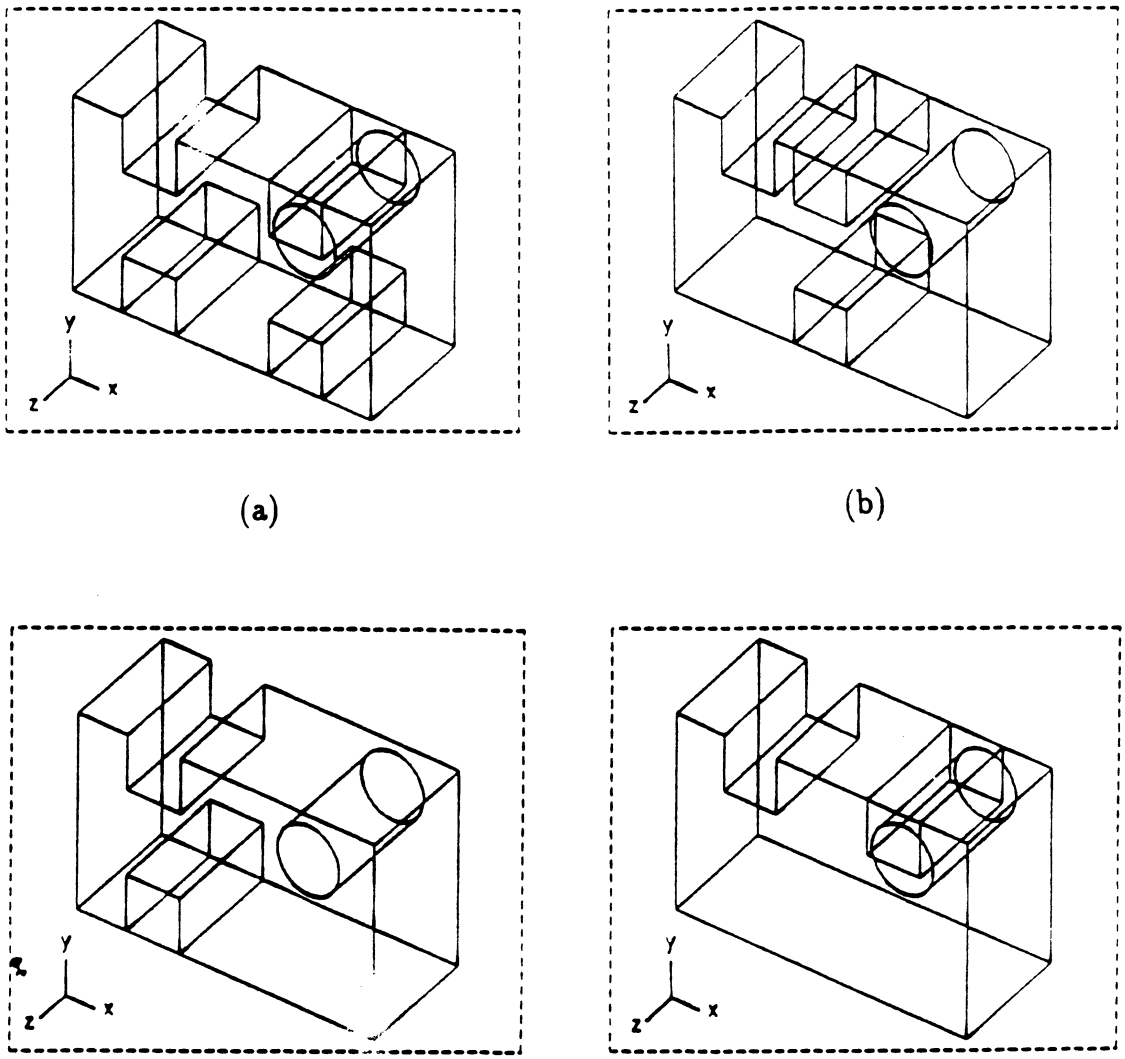
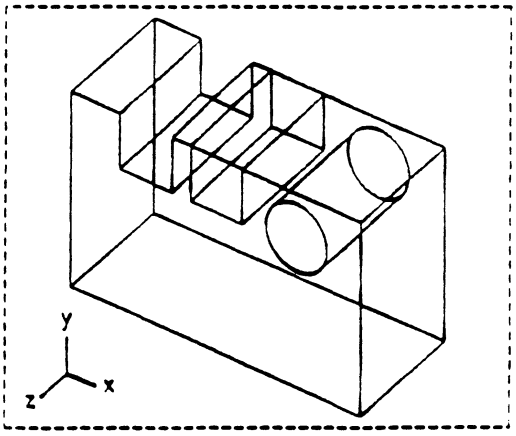
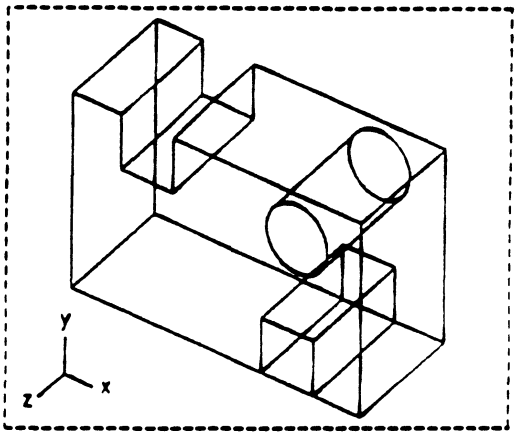


Figure 7.13: Suggestions for the groove in the *after* case. (a) First suggestion to obtain three-axis symmetry. (b) Second suggestion to obtain three-axis symmetry. (c) Suggestion to obtain X-axis symmetry. (d) First suggestion to obtain Y-axis symmetry.



(e)



(f)

Figure 7.14: Suggestions for the groove in the *after* case concluded. (e) Second suggestion to obtain Y-axis symmetry. (f) Suggestion to obtain Z-axis symmetry.

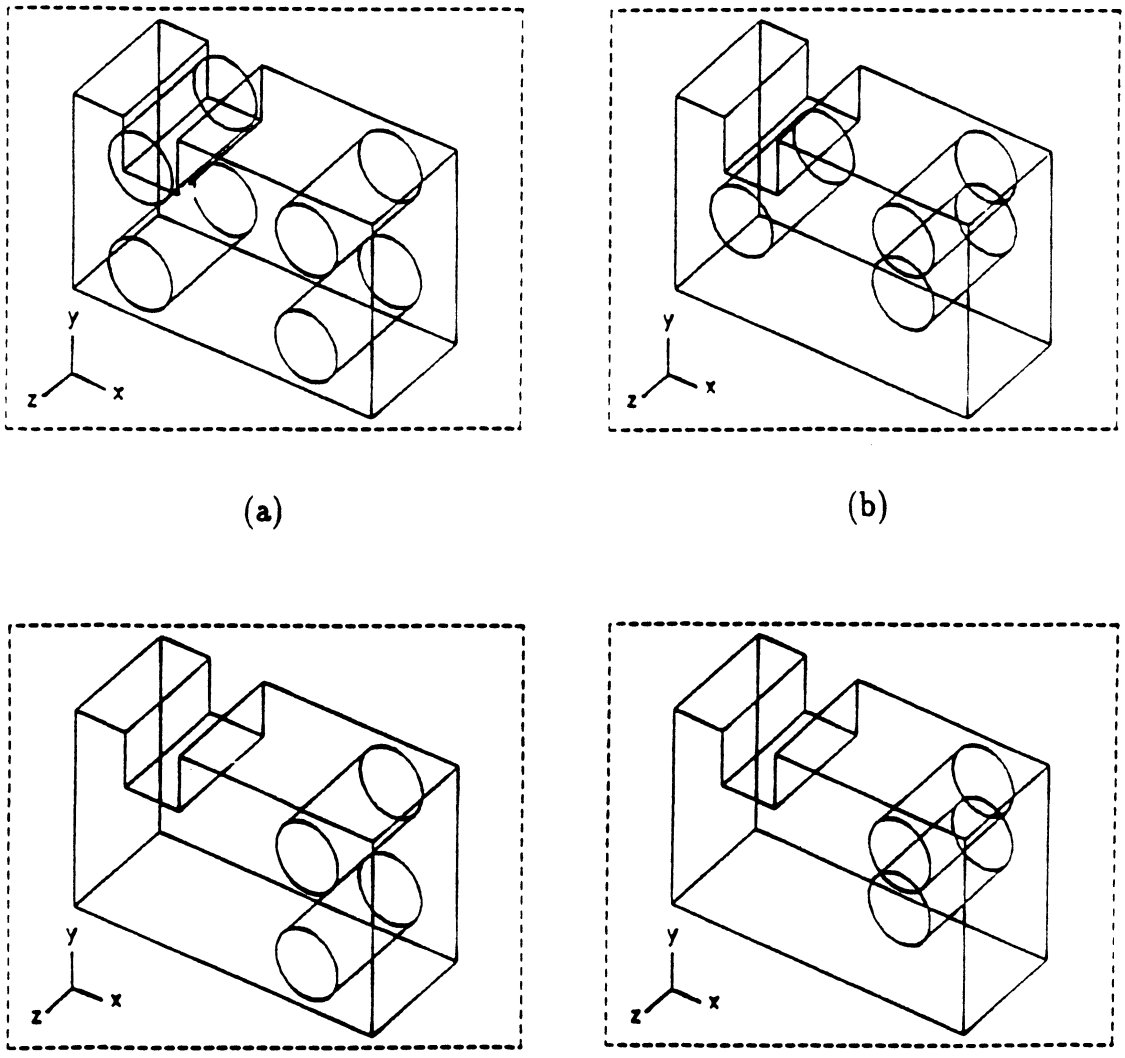
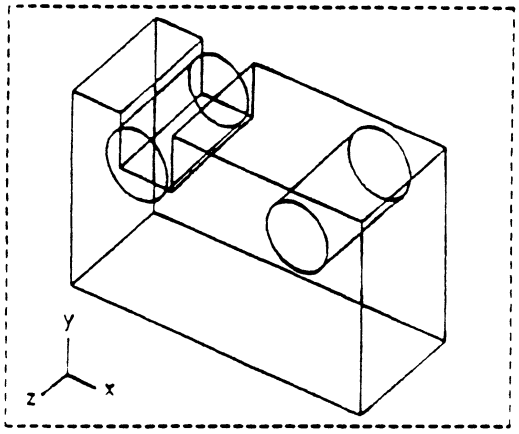
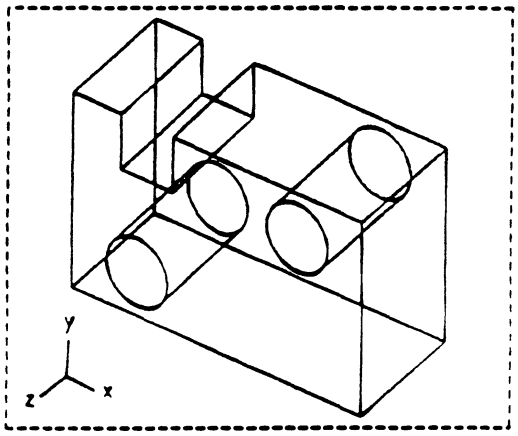


Figure 7.15: Suggestions for the hole in the *after* case. (a) First suggestion to obtain three-axis symmetry. (b) Second suggestion to obtain three-axis symmetry. (c) First suggestion to obtain X-axis symmetry. (d) Second suggestion to obtain X-axis symmetry.

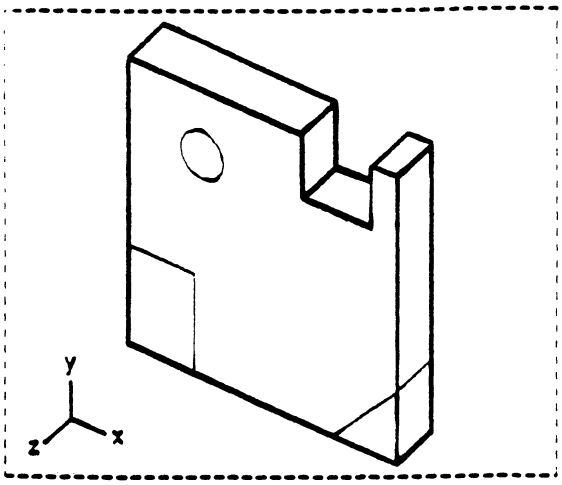


(e)



(f)

Figure 7.16: Suggestions for the hole in the *after* case concluded. (e) Suggestion to obtain Y-axis symmetry. (f) Suggestion to obtain Z-axis symmetry.



(a)

```
(  (Type, Block      ),
   (                 ),
   ...
   (                 )  ),
(  (Type, Hole       ),
   (                 ),
   ...
   (                 )  ),
(  (Type, Groove     ),
   (                 ),
   ...
   (                 )  ),
(  (Type, Step       ),
   (                 ),
   ...
   (                 )  ),
(  (Type, Chamfer    ),
   (                 ),
   ...
   (                 )  )
```

(b)

Figure 7.17: A design with three objects omitted. (a) The design. (b) The corresponding representation.

CHAPTER VIII

KNOWLEDGE ENGINEERING ISSUES

Introduction

In Chapter Six, architectures for a CAD production rule system and an intelligent suggestive CAD system were presented. Chapter Seven described two suggestive interactivities that will be used in a system-user test. This chapter will discuss issues arising from using rule-based programming to cause the suggestive modes. This will be an explanation of what the rules do in the broader framework of the overall system. We will begin by examining the decision making that is performed when a designer uses the existing charts. This will motivate a description of the rule-based processing that is required to generate suggestions from one input feature. The chapter will close with a discussion of characteristics of design domains that are suitable for suggestive systems.

Required Decision Making

The knowledge encoded in the system is the knowledge that allows a human to effectively use the Boothroyd-Dewhurst Charts. This is simply the capability to

rank a part with the charts *and* intelligently redesign it so it will rank more favorably. Encoding the knowledge with the production rule system developed here allows the automation of this decision making task.

Before describing how a human designer typically uses the charts, it is necessary to describe the pertinent charts in more detail. Suggestions are provided with reference to the first three Boothroyd-Dewhurst digits for nonrotational parts. Assigning values to these first three digits specifies values for the parameters OE and FC (see Chapter Four). The first digit is found by using Chart Four, found in [Boo.1]. This chart is is not a dual-entry matrix, but a list of descriptions of the overall shape of the part. There are three possible overall shapes for a nonrotational part: plate-like, bar-like, and cube-like. Aspect ratios of the nonrotational part in question are used to determine the overall shape and specify a first digit. The second and third digits are found by using Chart Six, found in [Boo.1]. This chart is a two dimensional matrix, with the row number specifying the second digit and the column number specifying the third digit. The rows and columns relate to symmetries and features of the part (see further discussion below). Additionally, each element of the matrix contains three possible OE-FC pairs, one for each possible value of the first digit. So Chart Six could be thought of as a three dimensional matrix. Suggestions derived from Chart Four are output when changes to the intial block are being considered. Since there are only three different overall shapes, the related suggestions are rather obvious (see the discussion of Figure (6.3)).When suggestions derived from Chart Six are output, the overall shape (i.e. the first digit) is considered fixed. The remainder of the discussion in this chapter pertains to deriving suggestions from Chart Six.

Although the actual Boothroyd-Dewhurst Chart Six is not reproduced here (see [Boo.1]), it is beneficial to describe how one might analyze an existing design, as this will show some of the decision processes automated by the suggestive system. Figure (8.1) shows an example part. The first step is to determine which feature (here,

"feature" is used in the general sense, not the specific sense defined in Chapter Five) is most likely to interact with the filters and orienting devices, and note it as the "main feature." In Figure (8.1) it is the groove since it borders three planes of the convex hull of the part (while the hole only borders two planes), facilitating the application of orienting moments and forces. Next, several properties of the main feature and properties of the entire design related to the main feature are noted. In this case, the width and depth of the groove are noted together with the fact that the orientation of the entire part is defined by the orientation of the groove. These notations are matched against the descriptions on the charts to determine the second and third digits. Using all three digits, an OE-FC pair is specified. Design quality is defined as the ratio FC/OE. Recall from Chapter Four that the abbreviations "FC" and "OE" are shorthand for "relative feeder cost" and "orienting efficiency" respectively. Naturally, we would hope to minimize FC and maximize OE. In the cost equations given in [Boo.1], if the feeding equipment is underutilized, cost is proportional to FC. If the equipment is fully utilized, cost is proportional to the ratio FC/OE. It is desirable to retain the influence of OE in the quality figures because OE relates primarily to the geometric shape of the part. Therefore, in the quality ranking of designs, we assume full utilization and use the value FC/OE. This value can vary from zero for the best designs to infinity for the worst designs. The part in Figure (8.1) is found to be a relatively poor design because it has no symmetry.

One way to determine improvements to the design is to first generate alterations, and then reanalyze the design with each alteration to determine if in fact the design has been improved. This method will always work, but it can be very computationally costly to repeatedly perform the design analysis for each potential improvement suggestion. An alternate way to determine improvements is to generate alterations and *predict* the design quality that will result. This is the approach taken here.

This is done by assuming that the suggestions will not cause a different feature of

the design to become the main feature. An interesting property of Chart Six is that one index of the matrix is specified by a description of the main feature and the other index is specified by some overall design property related to the main feature. This is shown in Figure (8.2) which is a simplification of Chart Six. The description of the main feature includes the feature type and other properties of the feature, such as orientation and size. The design can be improved by altering it so it is ranked with a more favorable matrix element of the chart. Suggestions, therefore, might cause a ranking with a different row *and* different column of the chart. Since the choice of main feature is assumed to not change, however, the predicted suggestion rankings will be in the same column of the chart, but in a different row. The following section explains how suggestions are generated and how this simplification is used to predict the resulting design quality. In some cases, the suggestions *will* cause a different feature to become the main feature, so these quality predictions are sometimes not accurate. In almost all cases, however, the suggestions will increase or maintain the actual quality of the design.

Generation of Suggestions

Introduction

The description of the high-level system architecture in Chapter Six made reference to the procedures PRE-SUGGESTIVE-RULES-x, SUGGESTIVE-RULES-x, and POST-ALTERATION-RULES-x without describing their structure or precise function. In this section we describe the structures of the various rule sets that make up these three procedures and discuss how the rules change the state of the production rule system. This is done with the example of a groove feature being input to an evolving design. Other features are handled similarly. The discussion

will emphasize techniques to effect the suggestions shown in Figures (7.3) through (7.5) (i.e. suggestions to maintain some symmetry), since the suggestions in Figures (7.6) and (7.7) (maintain same symmetry), and Figure (7.8) (reduce the amount of asymmetry) are achieved with similar methods.

The state of the production rule system is recorded with values assigned to local variables of procedures and values linked to property names of the the various association-list representations. Recall from Chapter Six that the rule sets are actually local variables in procedures that receive features, suggestions, and objects as arguments and also contain other local variables. These other local variables can be used to hold information related to the rule-based processing. This information is held, however, only as long as the rule-based procedure is running: once the procedure stops, the values of local variables are lost. Association-list property values, on the other hand, survive the termination of the procedure since they are part of the arguments that are passed into and out of the procedure. Property values, then, can be used to hold information that is significant to many different procedures.

It is useful to think of two distinct classes of property values for any alteration step that is used to generate suggestions. *Primary* property values are those that are immediately evident and recorded when the feature is input by the user. These properties are normally geometric in nature, such as various sizes, locations, and orientations and are generated by the PROCESS-FEATURE-INFORMATION routines (see Chapter Six). A groove, for example, would have the following primary property values. Note that the *type* of value required for each property is given instead of actual example values.

((Type, **STRING**),
(Name, **STRING**),
(Normal, **LIST-OF-3 SCALAR**),
(Direction, **LIST-OF-3 SCALAR**),

(Origin-point, **LIST-OF-3 SCALAR**),
(Length, **SCALAR**), (10)
(Width, **SCALAR**),
(Depth, **SCALAR**),
(Geometry, **LIST-OF-8 POINTS**),
(Volume, **SCALAR**),
(Extents, **LIST-OF-3 (LIST-OF-2 SCALAR)**))

Figure (8.3) shows the geometric meaning of some of these properties. "Type" and "Name" are identifying string properties, with "Type" being assigned the value "Groove." "Normal" and "Direction" are unit vectors that are specified with a string of three scalars. "Origin-point" is the center point of the normal face of the groove and is specified with a list of three scalars. "Length," "Width," and "Depth" are scalars as would be expected. "Geometry" is a list of eight points found at the corners of the "box" of the groove. The CAD system programming language used here had a *POINTS* data type. This "Geometry" property is an example of how the feature-based representation is related to the underlying CAD representation. Other languages might require storing the numerical space coordinates of the point list. "Volume" is, as expected, a scalar. "Extents" is a list of the form ((Xmin, Xmax), (Ymin, Ymax), (Zmin, Zmax)), where "min" and "max" refer to the minimum and maximum coordinate values over the entire list of eight points. "Extents," therefore, specifies the portion of three-dimensional space taken up by the groove.

Derived property values are those that are added by the rule-based programming, primarily for the purpose of generating and supporting suggestions. The following discussion will give examples of these and explain their purpose.

The remainder of this section will give insight into the rule-based programming by describing the nine production rule cycles that are distributed over three typical

rule-based procedures. In particular, the state changes caused by each cycle will be specified. Example rules will be presented with English language statements only. All rules have the syntax shown in Figure (6.4a). The convention "(feature/object, property-name)," used in these example rules should be noted. Recall that the production rule system inference engine singles out one "feature" and uses it with each object in turn to perform inferencing. This is done by extracting property values from the feature and the object and using them as arguments for some subprogram. The convention above specifies whether the feature or object is being considered, and what property value is extracted. In the discussion that follows, the term "feature" will be used to refer to the argument passed to the production rule system and to refer to a user-generated alteration step. The context of each usage will make the meaning clear. Finally, although antecedent and consequent routines will not be specified *in* the sample rules, pertinent routines will be discussed to give some idea of their purpose.

Rule-Based Programming

The following discussion will describe the nine production rule cycles for a groove. The first seven cycles are found in procedure PRE-SUGGESTIVE-RULES-GROOVE, and cycles eight and nine are in SUGGESTIVE-RULES-GROOVE and POST-ALTERATIONS-RULES-GROOVE respectively.

The purpose of cycle one is to calculate preliminary values and to load local variables with various property values that will be useful for subsequent processing. Loading variables with the property values eliminates the need to reaccess the feature or object and speeds up processing. A typical production is as follows:

```
If    (object, "Main-feature") = "True",
Then  previous-digit-1 := (object, "Digit-1"),          (11)
```

and previous-digit-2 := (object, "Digit-2"),

and previous-digit-3 := (object, "Digit-3").

All objects in the design representation could potentially be the main feature. All, therefore, contain the derived property "Main-feature" and the three derived digit properties. "Digit-1" is the first Boothroyd-Dewhurst digit and pertains to the overall size of the original block. (Note that all objects redundantly have the same "Digit-1" property value.) "Digit-3" is the column number of the encoded Chart Six (as shown in Figure (8.2)) and specifies properties of the main feature. "Digit-2" is the row number of the encoded chart. This specifies some overall design property related to the main feature, usually the type of symmetry of the main feature. If a particular object is the main feature of an evolving design, the "Main-feature" property is linked to the value "True." This rule loads the Boothroyd-Dewhurst digit values of the current main-feature into local variables. Other values that are similarly loaded (with other productions of cycle one) are the dimensions and related direction vectors of the original block. The property "A-dim" is the longest dimension of the original block and the unit vector aligned with this dimension is linked to the "X-dir" property. The variables a-dimension and x-direction are loaded with these values. "B-dim" and "Y-dir," associated with the midlength dimension of the block, are loaded into b-dimension and y-direction. "C-dim" and "Z-dir," associated with the shortest block dimension, are loaded into c-dimension and z-direction. The consequent procedure VARIABLE-GETS-VALUE was written to load variables with feature and object property values, or with the values of other variables.

Cycle two is used to choose a third Boothroyd-Dewhurst digit for the input groove and assign it to a variable. Adding it as a derived property to the groove association list is not done until the next cycle. A typical production is shown below. Note that, like the rule shown in Figure (6.4), this rule is matched and fired with one

object only, since all the required properties come from the feature. For simplicity, the antecedent insuring that it matches with only one object is not shown.

If (feature, "Direction") is parallel to x-direction,
and (feature, "Depth") is greater than a minimum-depth,
and (feature, "Width") is greater than a minimum-width, (12)
Then feature-column-number := 4.

Properties of the groove are compared to property descriptions on the chart to choose the correct chart column. This is exactly the task completed by a person using the charts. Minimum-depth and minimum-width are local variables that are related to the dimensions of the original block. Two antecedent procedures are of interest here. IS-PARALLEL was written to test if two unit vector representations are parallel. LESS-EQUAL-GREATER is a multi-purpose antecedent that performs all equality and inequality comparisons of scalar quantities.

To this point, we have not added any properties to any association-list representations. The purpose of cycle three is to add the "Digit-1," "Digit-2," and "Digit-3" properties to the feature representation and to choose a column of the chart that will be used in the generation of suggestions. First, we explain how the three digits are specified. Since "Digit-1" relates to the dimensions of the block, it is readily available in the variable previous-digit-1. The value of "Digit-3" was determined in the previous cycle and is found in the variable feature-column-number. Only "Digit-2" is left unspecified. Recall from the discussion of cycle one that "Digit-2" usually relates to the symmetry of the particular object. Recall from Chapter Six that features are input by the user in a freehand style using a digitizing pen on the graphics screen. The user does not input exact dimensions, locations, and relationships to other features. It is impossible, then, to input a single groove that is symmetric about any of the three axes. (There are, however, explicitly symmetric features available to

the user. The system aids the user in the construction of a single symmetric feature or symmetric groups of features.) The feature will be either *grossly asymmetric* or *slightly asymmetric*, depending on how far "off center" the feature is. The details of this distinction do not merit discussion here: the idea of gross and slight asymmetry should be obvious. The important point is that there are only two possible "Digit-2" values for a groove feature, 4 if the groove is grossly asymmetric and 9 if the groove is slightly asymmetric.

The other task of cycle three is to determine which column of the chart will be used in the generation of suggestions, and to assign the column number to the variable motivating-column-number. The basic principle of generating suggestions is that a main feature column is chosen and the design is altered to place the design ranking in a more favorable row, while maintaining the main feature column. Different columns have different orderings of row desirability. To achieve the design modification, only the *tentative* feature geometry can be altered. None of the incorporated geometry can be changed. Considering the entire object list and the tentative feature together as an *interim design* representation (i.e. what would the design be if the feature was incorporated "as is"?), it is clear that the added feature is not necessarily the main feature. A design with a single grossly asymmetric groove object provides an example. Adding another grossly asymmetric groove feature that is smaller than the object groove will leave the object groove as the main feature. Suggestions will be generated from the input feature to improve the ranking row of the incorporated object. In most cases, however, an added groove will be the main feature of the interim representation. Adding a groove to a design with some symmetry, for example, will always yield an asymmetric interim design. The problem is simplified somewhat by noting that the motivating column is either the column of the feature, or the column of the main feature object (previously the most important object of the design). A sample production illustrates the case where a groove

is added to a previously symmetric design (again, matched and executed over one object):

If previous-digit-2 not equal to 4,
and previous-digit-2 not equal to 9,
and feature is grossly asymmetric,
Then add (feature, "Digit-1") with value of previous-digit-1,
and add (feature, "Digit-2") with value of 4, (13)
and add (feature, "Digit-3") with value feature-column-number,
and motivating-column-number := feature-column-number,
and add (feature, "Main-feature") with value of "Not-assigned."

To paraphrase, if the design was previously symmetric (previous-digit-2 not equal to 4 or 9), and the feature is grossly asymmetric, then add "Digit-1," "Digit-2," and "Digit-3" properties to the feature, and note that the motivating column is the feature column. The property "Main-feature" is also added to the feature with the value of "Not-assigned." The consequent routine ADDPROPERTY was written to add property-value pairs to the association list representations. Note now that the representation of the feature is as follows:

(. . . all primary properties . . .
(Digit-1, **SCALAR**), (14)
(Digit-2, **SCALAR**),
(Digit-3, **SCALAR**),
(Main-feature, **STRING**))

Cycle four is used to derive all the information needed to display and process the suggestions. The suggestions, if used instead of the feature, will cause the design to

be ranked in a different row of the chart. This cycle creates the geometry necessary to display the suggestions and also determines the relationship between each suggestion and the row number it would cause. Figure (8.4) shows the six other individual grooves that are derived from the groove feature by some translation and/or rotation of the points that are attached to the "Geometry" property (Groove-1 undergoes no transformation). These six point lists are attached to the feature association list with the property names "Groove-1" ... "Groove-6," and will be used later to actually render the geometry of suggestions on the CAD screen. Subsets of this set of six grooves make up individual suggestions. Suggestion-3, for example, uses Groove-1 and Groove-6 to form a suggestion that is symmetric about the direction axis of the groove. The symmetries of the suggestions are known with respect to the local groove coordinate system (shown in Figure (8.4)). It is necessary to determine the symmetries with respect to the global axes of the block in order to assign each suggestion a row number. Since the symmetry with respect to the local axes is known, to find the global symmetries the productions simply find the relationship between the local and global axes and use this information to assign a row number to each suggestion. The properties "Suggestion-1" ... "Suggestion-6" are added to the feature with the determined row numbers as values. A typical production is as follows (matched and executed over one object):

```
If    (feature, "Direction") is parallel to x-direction.
      and (feature, "Normal") is parallel to y-direction,
Then  translate and rotate (feature, "Geometry"),
      and add (feature, "Groove-1") with value of altered point list,
      . . .
      and add (feature, "Groove-6") with value of altered point list,      (15)
      and extents-1 := extents of (feature, "Groove-1"),
      . . .
```

and extents-6 := extents of (feature, "Groove-6").

and add (feature, "Suggestion-1") with value of row number.

. . .

and add (feature, "Suggestion-6") with value of row number.

Note that the six extents are not added as properties but stored as variables since, as will be seen, they are not needed outside the scope of PRE-SUGGESTIVE-RULES-GROOVE. The consequent routines CALCULATE-GROOVE-EXTENTS, TRANSLATE, and ROTATE perform the various computations on the groove point lists. TRANSLATE and ROTATE can be used on any list of points.

The feature representation now appears as follows:

(. . . all primary properties . . .
(Digit-1, **SCALAR**),
(Digit-2, **SCALAR**),
(Digit-3, **SCALAR**),
(Main-feature, **STRING**), (16)
(Groove-1, **LIST-OF-8 POINTS**),
. . .
(Groove-6, **LIST-OF-8 POINTS**),
(Suggestion-1, **SCALAR**),
. . .
(Suggestion-6, **SCALAR**))

The information needed to generate suggestions has now been added to the representation of the feature. Note that the number of possible suggestions (six in this case) gives some idea of the power of the suggestive mode implementation. It would

be easy to add many more suggestions. Note for example, that none of the suggestions change the size of the original feature groove. Note also that information for all possible suggestions has been added to the feature. This is not always the case. In cases where the design is previously one-axis symmetric, as in Figures (7.6) and (7.7), only those suggestions that are three-axis symmetric or symmetric about the same axis will add derived properties to the feature. It is also possible that point lists from incorporated objects may be used to generate the suggestion information. This is the case in Figure (7.8), of course using a hole, where the goal is to reduce the amount of asymmetry. The basic idea in programming the rules in this case is to treat some object as the feature by creating a copy of the object association list and adding the relevant suggestion properties to the copy. This is how the smaller symmetric hole is made. Interestingly, this is programmed most effectively using a high-level "meta-rule," where each consequent causes one invocation of the production rule system. The *after* mode is programmed similarly by extracting an object and generating derived suggestion properties from it.

Although information for generating the suggestions has been added to the feature, some of these suggestions may not be possible. If one of the suggestions contains a groove that volumetrically interferes with some incorporated object, it should not be presented to the user. Cycle five uses the extents of the suggestion grooves (held in variables extents-1 ... extents-6) and the "Extents" property of the objects to detect this condition and note it in a variable. A typical production is as follows:

```
If    extents-3 clashes with (object, "Extents"),
Then  suggestion-1-possible := "False,"                    (17)
      and suggestion-4-possible := "False."
```

If Groove-3 interferes with some other object, then both Suggestion-1 and Suggestion-4 will be impossible because they both use Groove-3. Since this rule is matched over

all the objects. this rule will check if Groove-3 clashes with any object of the design. There are six such rules, one for each suggestion groove. The antecedent procedure CLASH was written to compare the extents of two geometric objects and determine if they clash. Note that this type of production provides a very rudimentary, but useful, form of geometric reasoning.

The rules of cycle five note the impossibility of a suggestion with a variable rather than simply deleting the suggestion property because the suggestion property may not have been added to the feature representation. Cycle six deletes the properties if they have been added and are impossible. Recall from the discussion of cycle four, that in certain cases, not all six suggestion properties are added to the feature. An error will occur if the system tries to delete a property that is not there. A sample production is shown below (matched and executed over one object):

If (feature, "Suggestion-4") exists,
and suggestion-4-possible = "False," (18)
Then delete (feature, "Suggestion-4").

This removes the entire property-value pair from the association list. The antecedent procedure FEATURE-PROPERTY-EXISTS checks if the feature has the property and the consequent procedure DELETE-PROPERTY deletes the property value pair.

The final cycle of PRE-SUGGESTIVE-RULES-GROOVE uses the motivating column number (see the discussion of cycle three) and the overall shape of the block to order the suggestion row numbers. Given that the suggestions, if incorporated, will cause different row numbers, this cycle determines which row is best, second best, and so on and adds this ordering as a property to the feature. The following is a typical rule (matched and executed over one object):

If (feature, "Digit-1") = 8,

and motivating-column-number = 1. (19)

Then add (feature, "Suggestion-order") with value (0, 1, 2, 3).

The "Digit-1" property is used to specify the overall shape of the block. For the chart encoded here, there are thirteen different suggestion orders resulting from combinations of "Digit-1" and motivating-column-number values. After adding this property, the feature representation is as follows:

(. . . all primary properties . . .
(Digit-1, **SCALAR**),
(Digit-2, **SCALAR**),
(Digit-3, **SCALAR**),
(Main-feature, **STRING**), (20)
(Groove-1, **LIST-OF-8 POINTS**),
. . .
(Groove-6, **LIST-OF-8 POINTS**),
(Suggestion-1, **SCALAR**),
. . .
(Suggestion-6, **SCALAR**),
(Suggestion-order, **LIST-OF SCALAR**))

This representation is then passed into the procedure SUGGESTIVE-RULES-GROOVE which contains cycle eight. The purpose is to take the suggestion properties attached to the feature and use them to output the suggestions to the user in the proper order. This is done by arranging the match, resolve, and execute portions of the production rule system in the special purpose algorithm shown in Figure (8.5). To understand the function of this routine, consider one of the productions that is

used (matched and executed over one object):

```
If    (feature, "Suggestion-2") exists,
      and (feature, "Suggestion-2") equals row-number,
Then  Render and process Groove-2,
      and Render and process Groove-5,
      and Establish relationships between Groove-2 and Groove-5,        (21)
      and Add "Digit-1" prop. to Groove-2 and Groove-5 a-lists,
      and Add "Digit-2" prop. to Groove-2 and Groove-5 a-lists,
      and Add "Digit-3" prop. to Groove-2 and Groove-5 a-lists,
      and Add "Main-feature" prop. to Groove-2 and Groove-5 a-lists,
      and Allow user choice.
```

There is a production for each suggestion. For each cycle through the FOR loop a different row number is specified, according to the ordering of the row numbers found attached to the property "Suggestion-order." The algorithm tries to match all rules for every FOR loop cycle. Only those rules that will produce the row number under consideration will survive the match process. The survivors will be added to a conflict list. Executing all rules in this conflict list, in order, will output a prioritized list of suggestions that produce row numbers in the same order as that specified in "Suggestion-order." The user is allowed to selectively execute elements of the conflict list with a menu that is created. This menu shows the relative predicted qualities of the feature and suggestions in order of decreasing predicted quality. To view a suggestion, a particular element of the conflict list is executed. As the consequents above show, this will graphically render the grooves of the suggestion and create an individual association list for each one (i.e. "Render and process").[1] Once these association lists exist, relationships between them are established. In this case, for

---

[1] It is evident, therefore, that a suggestion can actually be a list of association

example, properties would be added to indicate that Groove-2 and Groove-5 are three-axis symmetric partners. Also, digit properties and a main feature property are added to all grooves of the suggestion. For the suggestions shown in Figure (8.4), each groove of a suggestion will have the same "Digit-1" and "Digit-3" values as the original feature groove. The "Digit-1" value is the same because the original block has not changed. The "Digit-3" value is the same because each suggestion groove has the same "Direction," "Depth," and "Width" properties as the feature groove (refer to the explanation of cycle two above). The "Digit-2" value for each suggestion groove is the row number the suggestion will cause. The "Main-feature" property is added with the value "Not-assigned." Finally, the user must choose to view other suggestions or stop at this point. A decision to stop will require a subsequent decision to keep this suggestion, keep the feature that motivated it, or discard both. The user can pick only one suggestion from the execute process and the routine USER-RESPONSE creates the menu interactivity that allows this choice. The consequent routine SUGGEST-GROOVE renders and processes a suggestion groove.

Finally, since most of the derived properties are used only for the generation of suggestions, they should be deleted from the feature representation prior to adding the feature to the design object list. This is accomplished with cycle nine contained in procedure POST-ALTERATION-RULES-GROOVE, which is invoked by procedure INCORPORATE-AND-PROCESS. A sample production is shown below (matched and executed over one object):

If (feature, "Suggestion-4") exists, (22)

Then delete (feature, "Suggestion-4").

---

lists. This is also true of a feature, e.g. a symmetric groove pair. For simplicity, the suggestive CAD system architecture described in Chapter Six presents features and suggestions as single association lists. The procedures described there actually use features and suggestions that are lists of association lists.

Rules similar to this delete the "Groove-1" ... "Groove-6," "Suggestion-1" ... "Suggestion-6," and "Suggestion-order" properties in order to minimize the size of features that will be added to the design representation. This helps to make the program run faster. Other productions in cycle nine establish relationships between suggestion and object representations if a *reduce asymmetry* suggestion is accepted. After the productions of cycle nine are executed, the feature representation should appear as:

(. . . all primary properties . . .
(Digit-1, **SCALAR**),
(Digit-2, **SCALAR**),
(Digit-3, **SCALAR**),
(Main-feature, **STRING**)) (23)

Discussion

First, it should be mentioned that the processing done by cycle nine can be quite wasteful. Some of the productions apply only to features and others apply only to suggestions. The POST-ALTERATION-RULES procedures, however, accept a general alteration step as an argument. The production rule system will try to match many rules that are certain to fail, causing some wasted processing. One way to remedy this problem is to set up different cycle nine cases: one if a feature is being incorporated and one if a suggestion is being incorporated. Different rule sets would be used for each case. There are several other situations where some amount of needless processing can be eliminated by adding more procedural control programming. Doing this reduces the size and increases the specificity of each rule set. Adding more procedural control knowledge causes a natural progression from a rule-based to a procedural system.

A second point is that there is a fundamental theme to the suggestion-generation process. Properties are derived, then added to the feature, then used to generate the suggestions, then deleted from the feature. This overall procedure was an ad-hoc creation, but seems to work satisfactorily.

Note also that after a suggestion is incorporated into the design, the main feature object is not known with certainty. More processing (done by procedure ENSURE-VALIDITY) is required to determine the most important object of the design representation. The fact that the tentative feature is the most important object of the *interim* design representation does not imply that suggestions generated from this feature will also be most important. Consider Figures (7.6a) and (7.6c). If the hole feature of Figure (7.6a) is incorporated, clearly it will be the main feature because it will cause the asymmetry. The symmetric hole suggestion of Figure (7.6c) will not be the main feature because the grooves with the same symmetry are more accessible to filters and orienting devices. On the other hand, in Figures (7.3) through (7.5), since there were no other objects in the initial block, each suggestion generated from the main feature of the *interim* design representation would become the main feature if incorporated. Through the determination of the motivating column, the importance of the feature is known with certainty. To determine the importance of each suggestion would require comparing it with the entire design, and this is too computationally costly to do *during* the suggestive cycle. The suggestions are output not knowing if they would become the main feature. The design representation is made correct after some suggestion has been accepted.

A final point concerns the size of the knowledge base. For a groove, there are 77 productions in cycles one through seven, 9 rules in cycle eight and 24 rules in cycle 9. For a system with five features, therefore, there are about 500 rules relating to generating suggestions from features. There are perhaps an additional 200 rules for various utility programs (e.g. ENSURE-VALIDITY, automation of solid modeling

tasks) to yield a total of approximately 700 rules.

Suitable Properties of Design Domains

The preceding discussion suggests that certain characteristics of a design domain make it appropriate for modeling with a suggestive system. Note, for example, how the design for assembly charts specify not only design analysis knowledge, but also the necessary features. A suitable design domain must have well-defined knowledge *and* well-defined features. The predefined alteration steps provide building blocks for the designer and become the units of change for suggestive heuristics that can alter the design in order to improve it.

The domain must also provide some means to analyze the quality of the design. There must be some way to determine if designs are good or bad. This can be thought of as a classification problem. Designs are put into different classes, with each class having a different design quality. Suggestions can be generated by altering the design in some way, and then reclassifying it to determine if the alteration is an improvement.

It would be better if suggestions could be generated in a more informed way. Suggestions should be generated without having to reanalyze the design with each suggestion. Given a design and design analysis, the domain should provide some means to predict what alterations will improve the design. In other words, there should be some knowledge about the relationships between design alterations and the consequent reclassifications. Howe et al. [How.1] make similar observations in their description of the Dominic system. It is best if these predictions can be made with certainty, but uncertain predictions are also useful. In the design for assembly rule bases described in this chapter, this predictive capability is found in the list of row numbers attached to the "Suggestion-order" property. An interim design representation is first analyzed, yielding a suggestion-motivating column of the chart

(i.e. motivating-column-number). The system generates suggestions that would cause the design to be ranked in a different row. The relationships among the motivating column and the various rows of the chart allow the system to rank the suggestions in terms of predicted design quality. This predicted quality is clearly uncertain, since the design representation must be checked (with a program like ENSURE-VALIDITY) if a suggestion is incorporated.

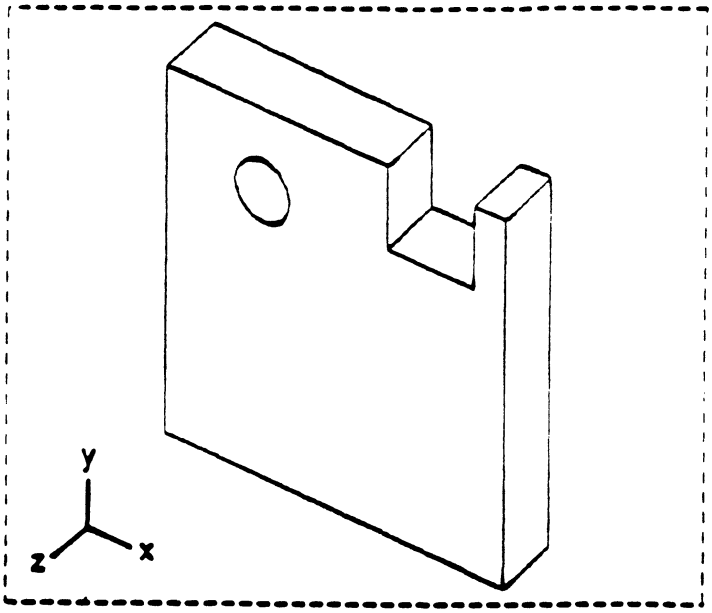


Figure 8.1: An example part for Boothroyd-Dewhurst analysis.

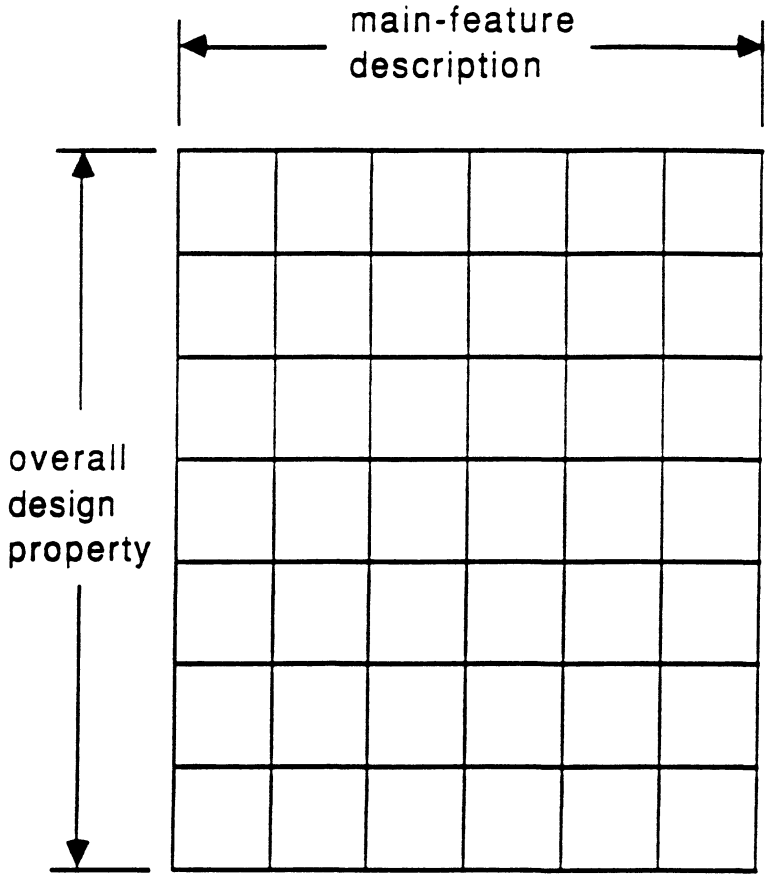


Figure 8.2: Simplification of the encoded Boothroyd-Dewhurst Chart Six.

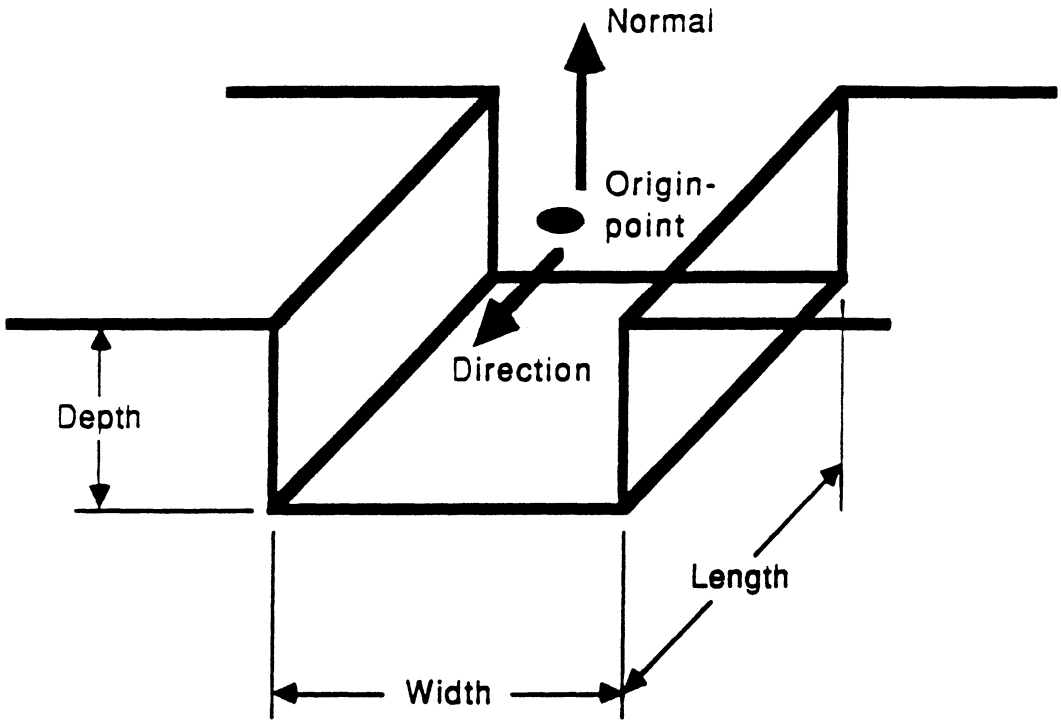


Figure 8.3: The primary properties of a groove.

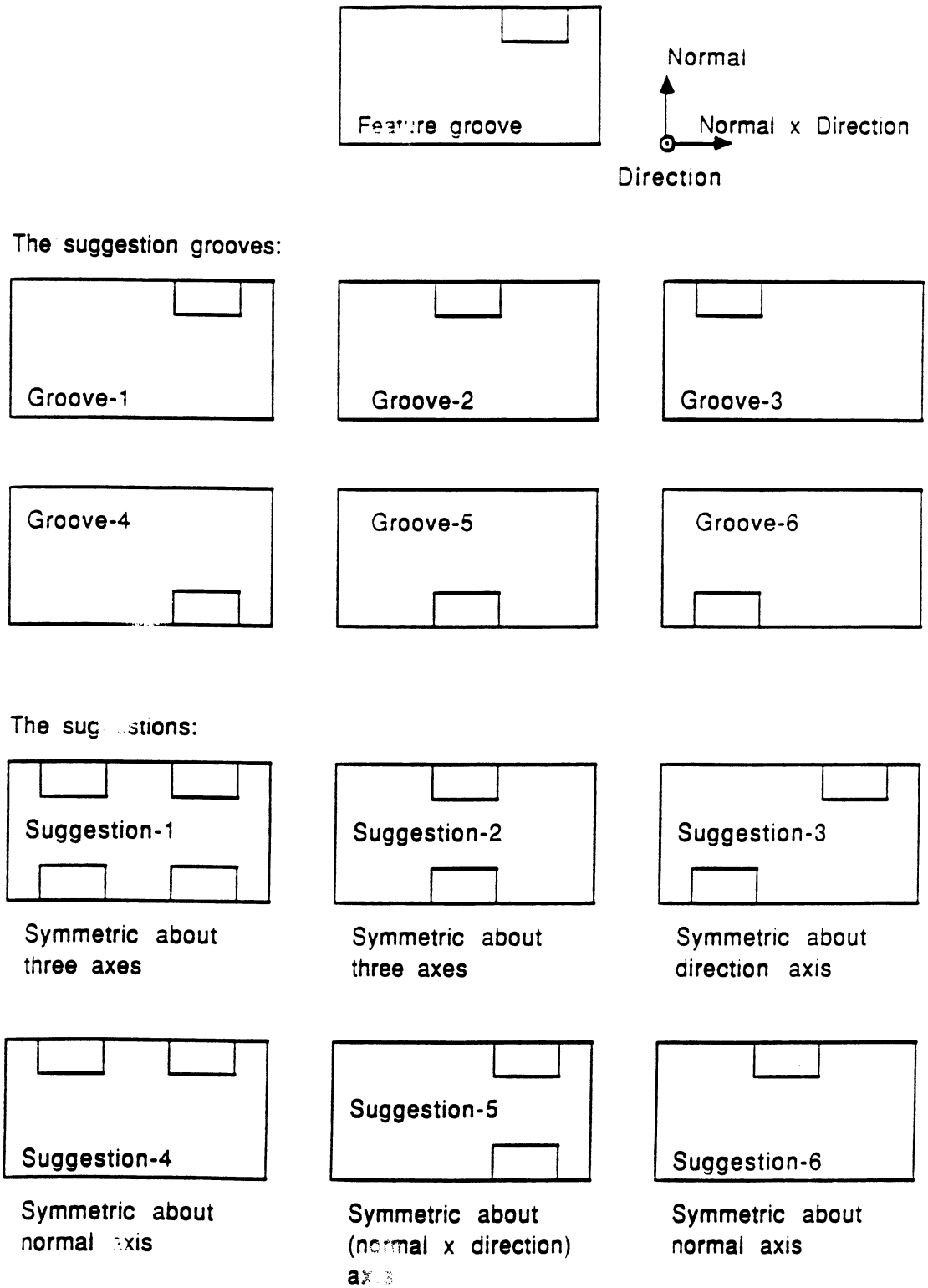


Figure 8.4: The grooves and suggestions derived from a groove feature.

```
suggestion-order := ( feature, "Suggestion-order" )

FOR each row-number in suggestion-order

    MATCH

    AUGMENT the conflict list

CONTINUE

EXECUTE the conflict list
```

Figure 8.5: The suggestion-production algorithm of procedure SUGGESTIVE-RULES-GROOVE.

CHAPTER IX

HUMAN-COMPUTER INTERACTION STUDIES

Introduction

This chapter describes the user testing that was done. Results are reported. Some conclusions are drawn, with respect to both the effectiveness of the interactivities and the general idea of suggestive design aids. A brief description of the experiment is provided, followed by an explanation of the motivation of the specific design assignment used in the experiment. With this background, the intellectual responsibilities of the test subjects are outlined. Finally, the results are presented and analyzed.

Description of Experiment

The experiment consisted of testing three interactivity modes with three groups of users to determine which mode helps designers create the best design solutions. The three modes were a control mode that provided no suggestions, and the two modes decided upon in Chapter Seven. Users of each mode were given the same design assignment and were instructed to pay attention to design for vibratory bowl

Unfortunately, it is more often the case that a version of a prototype is designed. Most designs seem to be redesigns, and design specifications are often in terms of form rather than function. This was clearly unacceptable for this experiment. To overcome this problem, we created an artificial design domain that is similar to the domain of digital circuit design. In the same way that each digital circuit element performs a single function (e.g. NAND gate, NOR gate, etc.), each feature in the feature-based modeler should perform a single function. This idea resulted in the "design of gauge tools" domain.

The fundamental idea is that features match each other in pairs. Consider, for example, a groove and a block. The groove gauges an upper tolerance of a single dimension of the block since if the block fits into the groove, the block is small enough. Conversely, a block gauges a lower tolerance of a groove since if the groove fits over the block, the groove is large enough. Pegs and holes are similar gauge pairs. Steps and chamfers are "gauges" in the sense that they provide a shape template to match with other steps and chamfers. A typical design problem is to configure a number of gauging features on a single, multipurpose gauge tool, given a set of gauging requirements. The design domain is more fully explained in the design assignment shown in Appendix B.

Note how there is a certainty of function with this domain. Grooves gauge blocks, holes gauge pegs, and so on. There is, on the other hand, a broad ambiguity of form. For a groove to gauge a block, for example, it is only necessary to specify the groove width. Values for other attributes, such as depth, length, location, and orientation are not needed. In this way, many designs are possible for a given set of functional requirements. Figure (7.11) shows a design that meets the assignment's Case A specifications. The original midlength side of the block is intended to gauge the three unit groove. The two grooves gauge the two boxes and the hole gauges the peg. Assuming that the two faces of a step are approximately equal in size, the

Unfortunately, it is more often the case that a version of a prototype is designed. Most designs seem to be redesigns, and design specifications are often in terms of form rather than function. This was clearly unacceptable for this experiment. To overcome this problem, we created an artificial design domain that is similar to the domain of digital circuit design. In the same way that each digital circuit element performs a single function (e.g. NAND gate, NOR gate, etc.), each feature in the feature-based modeler should perform a single function. This idea resulted in the "design of gauge tools" domain.

The fundamental idea is that features match each other in pairs. Consider, for example, a groove and a block. The groove gauges an upper tolerance of a single dimension of the block since if the block fits into the groove, the block is small enough. Conversely, a block gauges a lower tolerance of a groove since if the groove fits over the block, the groove is large enough. Pegs and holes are similar gauge pairs. Steps and chamfers are "gauges" in the sense that they provide a shape template to match with other steps and chamfers. A typical design problem is to configure a number of gauging features on a single, multipurpose gauge tool, given a set of gauging requirements. The design domain is more fully explained in the design assignment shown in Appendix B.

Note how there is a certainty of function with this domain. Grooves gauge blocks, holes gauge pegs, and so on. There is, on the other hand, a broad ambiguity of form. For a groove to gauge a block, for example, it is only necessary to specify the groove width. Values for other attributes, such as depth, length, location, and orientation are not needed. In this way, many designs are possible for a given set of functional requirements. Figure (7.11) shows a design that meets the assignment's Case A specifications. The original midlength side of the block is intended to gauge the three unit groove. The two grooves gauge the two boxes and the hole gauges the peg. Assuming that the two faces of a step are approximately equal in size, the

"characteristic dimension" of a step is the approximate depth of the faces. For a chamfer, this depth is a right-angle projection of the slanted face.

User Intellectual Responsibilities

The subjects in the experiment were required to fulfill various responsibilities and were allowed to make various decisions. This section describes these requirements and freedoms for the users in general, and for each user group.

The general setting of the experiment was a design problem. The users had to read and understand a design assignment and design an object to meet the given specifications. This required a mapping from function to form using the features available with the CAD system. The experiment had two sets of design specifications, known as "Case A" and "Case B." Case A provided only functional specifications while Case B provided different functional specifications (to discourage any "carry-over" designs from Case A to Case B) *and* other considerations. For both cases, the users were concerned with designing the object to be suitable for use with automated assembly machinery. The users were told to complete Case A within a two hour time limit, and if any time remained, to attempt Case B. While they were designing, the CAD graphics screen was videotaped and a log file of the user's menu choices was kept by the system.

There were common responsibilities for all three suggestive mode groups. All users had to read and understand the design assignment. They were given the design assignment immediately before the CAD session and were not allowed to begin designing until they had read the assignment and asked any questions.[1] In addition, they were allowed to ask questions at any time during the session. It was left to the users to ensure that they understood the assignment: they were not given

---

[1] The author was on hand at all times to answer questions.

any test that would demonstrate their understanding or lack thereof. After reading the assignment, they could work on their design. This required inputting features, in any order, to meet the design specifications. When they had finished the design, they had to generate a plot and annotate it to explain how their design met all the functional specifications. They were also allowed to use hand-drawn sketches to alter and add to the plot. This was often necessary because features were improperly created and there was not sufficient time to redo the design. The emphasis was on using the system to get design ideas rather than creating a flawless CAD model. The users also had to complete the questionnaire shown in Appendix B.

There was also a set of common decisions that all users were free to make. All were allowed to organize their overall design effort in any way they saw fit. They could make any number of design efforts and leave some of them incomplete. They decided when they were finished: they were not required to fill the entire time limit. Of course their most important decision was how to respond to the assistance received from the program. They decided whether they would consider the assistance, or ignore it and continue uninfluenced. Although the three modes provided assistance in different ways, all users received a quantitative indication of the design quality at the end of each design cycle. Also, all were allowed to refer to the standard Boothroyd-Dewhurst Charts.

The no-suggestion subjects used the interactivity diagrammed in Figure (7.1). The only assistance available to them was the standard charts and whatever design guidance they could deduce from the design quality indication at the end of each cycle. This output of a quality figure was sophisticated enough to provide some help to a concerned user. The system would highlight the object it interpreted as the main feature and provide the related quality figure. This could help the user decide on a subsequent design alteration.

The suggestions-during subjects, using the interactivity shown in Figure (7.2),

had specific responsibilities and required choices. In addition to the Boothroyd-Dewhurst Charts and the quality indication, these users also received specific design suggestions during each design cycle. The suggestions were generated from every feature input . . . were available for incorporation. There was usually more than one suggestion generated, so a menu was output that allowed the user to repeatedly view the suggestions. On the menu, the predicted design quality figures that would result from incorporation of each suggestion and from incorporation of the feature were shown. This allowed the user to compare the relative merit of each possibility. The program required that the user view at least one suggestion before proceeding, and to pick one suggestion from the list to consider incorporating. At that point, the user would choose to incorporate the feature, incorporate the suggestion, or reject both and begin a new design cycle.

These subjects were free to take a variety of actions in response to the suggestions. They could view as many of the suggestions as they wanted, as many times as they wanted, and they could pick any of the suggestions for possible incorporation. They were not required to incorporate either the feature or suggestion: they could discard both and investigate new design ideas that perhaps were a result of the program's assistance. Another important decision was the order of feature input.

The suggestions-after subjects, using the interactivity of Figure (7.10), also had unique responsibilities and choices. Again, the Boothroyd-Dewhurst Charts and system quality indication were available. The suggestions were output as a design postprocess. The users would complete their designs with no suggestive assistance, and then the system would output suggestions with respect to every feature, in the order it was input, as if it was the only feature on the block. Of course, suggestions about the block were made first. Like the suggestions-during users, a list of suggestions was normally output from which the user had to view at least one suggestion. Again the users could view all the suggestions repeatedly. These suggestions,

however, could only be viewed: they could not be incorporated.

The suggestions-after users were expected to configure a design, receive suggestions, and create a redesign based on the suggestions that were viewed. Within this basic sequence of events, there were many possibilities. The users could, for example, receive suggestions on a partially completed design. They were also free to receive suggestions on the redesign effort. The strength of the suggestive heuristics was such that two user redesign efforts could usually produce an object that the system could not improve. Any number of initial design efforts, time permitting, could be investigated. On the other hand, the users were not required to create a redesign. After viewing the suggestions, they could submit their initial unaided effort as the final solution.

Results

The overall results of the experiment are somewhat disappointing because there is not enough distinction in design qualities between the the three groups. In particular, there is not a marked difference between the suggestions-during and suggestions-after design qualities. Although it is possible that these two modes are equally helpful, it was hoped that there would be some distinction between them. It is possible, however, to derive some conclusions and formulate future research directions from a close look at the data. The following discussion will present the results of the experiment and will describe several of the individual design efforts that demonstrate important findings.

Basic Results

Figure (9.1) shows the basic result of the experiment, a plot of design quality versus suggestive mode. Note that this data is for Case A of the design assignment

only, as many of the users did not have time to attempt Case B. Also, the identification numbers assigned to the fifteen users are placed next to the data points. Recall from Chapter Eight that the design quality is defined as the ratio FC/OE. This value theoretically varies from zero for the best designs to infinity for the worst designs. All designs created by the users yielded qualitities between zero and sixteen.

The data seems to show that suggestions of some type do help. The two suggestive groups have a higher fraction of users with better designs (say, quality figure less than or equal to four). This is not a strong trend, however, and more data should be collected to amplify whatever distinction there is. The lack of distinction between the Mode 1 and Mode 2 users is disturbing. Without more data it is very difficult to draw any conclusions.

It might be useful to examine relationships between the user design qualities and other data that are available from the experiment. One comparison can be made using the *number of CAD model changes of each user.* When creating a design with the suggestive system, the users are also building a "model" with the underlying CAD system. The log of menu choices records the point at which work stops on one model and begins on another. This may be due to starting a new model from the beginning or returning to a model that was worked on previously. In many cases the user changes models because he has a new design idea, but it is possible that the old model was terminated by the CAD system because the user made an error in using the feature-based CAD program.

The hypothesis is that users with more CAD model changes should produce better designs. More CAD model changes should imply more design ideas, increasing the chances of a better final design solution. We expect, then, that the best designs were created by users who made the most CAD model changes. Figure (9.2), which shows this number for each user, does not support this hypothesis. Within any single suggestive mode, there is no evident correlation between the number of models and

the design quality. Indeed, the best designers for each mode made relatively few model changes. We might also hypothesize that the Mode 1 users would have fewer model changes than the Mode 2 users because the suggestions-during mode should optimize a design during the design effort, mimimizing the need for redesigns. The suggestions-after mode, on the other hand, requires the creation of redesigns as part of its normal operation. This is also not supported by the data. It is possible that the suggestions-during mode does cause some wholesale redesign ideas.

Also of interest is *the number of suggestions viewed by each designer.* Recall that the system often would generate a list of suggestions, and that the user was required to view at least one before proceeding. Since the suggestions would improve the design, it seems reasonable to predict that the users that viewed more suggestions would produce better designs. Viewing more suggestions should prompt more and better ideas. Again, the data as shown in Figure (9.3) do not support our expectation (repeated viewings of the same suggestion are counted). The Mode 3 users, of course, received no suggestions. The three best designers viewed relatively few suggestions and there does not seem to be a trend in any group. There is no noticeable distinction between the Mode 1 and Mode 2 designers. Users 06 and 04 are particularly notable as extremes. Users 07, 01, and 05 provide some positive evidence. Although their designs are not as good as those of users 13, 08, and 06, they did view a relatively large number of suggestions and their designs are of relatively high quality. Again, more data points would be very helpful.

With the existing data it is difficult to say which mode helps users create the best designs. Other comparisons of the systems, however, also merit discussion. The *productivity* allowed by each system is also important. In the context of true design creation, productivity is difficult to measure. The problem is very different from measuring productivity in a *design archival* task, where an existing design is being modeled. For design archival, measurements such as number of features input per

unit time, and number of models created per unit time are important because they indicate how quickly a design can be modeled. In a *design creation task* they are not as important because they give no indication that the design requirements are being met. The real issue is how quickly users can create designs that they are satisfied with. Their satisfaction depends upon the help they receive, both in generating design ideas and in modeling them. In this regard, then, it is useful to examine how many of the users finished Case A and attempted Case B in the allotted time. The suggestions-during mode is the clear winner. Four of the Mode 1 subjects made some serious attempt at Case B, compared to one of the Mode 2 subjects, and two of the Mode 3 subjects. More precise measurements of work output per unit time were not considered here because the system ran too slowly. The time required to generate a list of suggestions from a feature input could be as long as two minutes. Users spent a large portion of their time waiting for the system to respond rather than doing creative work, and this would make time measurements difficult to interpret. Brady proposes that such delays interrupt the attention of the user, causing overall creative productivity to decrease [Bra.1].

Another way to view the question of system productivity is to determine if the system actually eliminates the need for the standard Boothroyd-Dewhurst Charts. Do either of the suggestive modes integrate the chart information into the preliminary design process? Compiling the answers to question seven on the questionnaire (see Appendix B) shows that two Mode 1 users looked at the charts during the design process, three Mode 2 users looked (one Mode 2 user did not submit a questionnaire), and all five Mode 3 users looked at the charts. This type of result was expected since the Mode 3 users received no suggestive help and were forced to use the charts for detailed design for assembly information. Of the five Mode 1 and Mode 2 users that looked at the charts, four reported that they used the charts *only* to create an initial block shape, and not afterwards. (One Mode 2 user looked at the charts after

creating the initial block, but reported that they provided little help.) Four of the five Mode 3 users commented that they referred to the charts throughout the design session. Since the suggestions-during and suggestions-after designs are certainly no worse than the no-suggestion designs, we can conclude that the use of the charts for preliminary design has been automated to a large degree.

Specific Design Efforts

Since the overall data falls short of being conclusive, we will examine the efforts of particular experimental subjects in order to uncover other interesting issues related to the interactivities. The discussion below is derived from plots of the various designs, and the menu-choice log files.

Prior to running the experiment, one fear was that the output of the design quality index at the end of each design cycle would allow the no-suggestion subjects to deduce useful design guidelines that were presented to the other subjects as graphical suggestions. In other words, by observing the change in design quality with the input of various features, the no-suggestion users could determine how to favorably alter their designs without the benefit of graphical suggestions. It was observed that one Mode 3 subject, user 15, made this a primary design strategy by paying close attention to the quality index and terminating a design effort if the design quality appeared to drop too far. This resulted in eleven Case A design efforts (out of thirteen CAD model changes). A moderately good design was created with a great deal of effort.

A closer look at the designs of users 13, 07, and 01, the three most successful Mode 1 designers, uncovers some interesting findings. User 13 created a very good initial design that did not meet the functional specifications. Slight corrections were made in a second effort that was functionally correct and also very high quality. Users 07 and 01 both created flat, plate-like gauges that were symmetric about the Y axis (the

axis parallel to the midlength dimension of the original block). Both designs would have been much improved if they were symmetric about the X axis (the axis parallel to the longest dimension of the original block). Examining the menu choice logfile for user 01 shows that Y-axis symmetry resulted because the user began by adding Y-axis symmetric features to the block. The system does not generate suggestions that attempt to change the symmetry of already-symmetric feature input, although in this case it might have helped. The design proceeded to a conclusion with the system generating suggestions that maintained the Y-axis symmetry. The Y-axis symmetry of the design of user 07 resulted because the user chose to incorporate a nonoptimal suggestion. The log of menu choices shows that the user first established a flat, plate-like block and then added a groove feature. The system generated a list of suggestions, the best of which caused X-axis symmetry. (Recall that the possible suggestions are ranked and displayed with quality figures on a menu.) The user viewed this suggestion and two others that caused Y-axis symmetry, and chose to incorporate one of the Y-axis symmetric suggestions. The design was completed with the system making suggestions to maintain the existing symmetry.

The post-suggestive subjects provided perhaps the most interesting design sessions. User 08, who created the best final design, initially created a design that was a flat shape that was symmetric about the Y axis. After viewing the suggestions, the user made a similar higher quality design that was symmetric about all three axes. Some Mode 2 design sessions were notable for the radical redesigns that occurred. User 05 produced an initial effort that was very poor from an assembly standpoint, but clearly met all the functional requirements. It was a cube-like block with the required gauging features seemingly arbitrarily positioned. With a single redesign effort, a flat Y-symmetric design was created. As with users 07 and 01, this design would have been better if it was X-axis symmetric. User 02 had a similar design session. The first design was functionally correct but very low quality. A second,

very different effort was a long bar-like block that was symmetric about the Z axis (the axis parallel to the smallest dimension of the original block), a combination of dimensions and symmetries that yielded a fairly low quality design. Clearly, these two users payed very little attention to design for assembly ideas during their initial design. It seems as though their strategy was to anticipate the help provided by the suggestions. *After* viewing the suggestions, a genuine effort would be made to design the gauge tool for assembly

Analysis of Results

In general, the results of the experiment were somewhat disappointing. There is only weak evidence that designers who received suggestions created better designs and the data do not indicate which of two suggestive modes is more effective. It was found, however, that a suggestive system can integrate the concerns of downstream analyses into the initial design process. It should be emphasized that this was a preliminary experiment, intended to provide an initial useful experience in testing such systems as well as data. Although the results are not very conclusive, we can summarize some findings and make recommendations for subsequent efforts.

Obviously, there were deficiencies in the experiment. It would be useful to outline what was wrong and how it should be changed in future tests. First, it does seem as though the experiment made an interesting comparison. In subsequent experiments, it would still be useful to compare a suggestions-during mode with a suggestions-after mode. Along with the obvious interesting contrast of help during design versus help after design, the experiment revealed some unexpected user behavior that calls for further investigation. The *during* users created many design efforts, sometimes abandoning an idea to start fresh. Some *after* users advantageously adopted a "cynical" attitude toward using the system, clearly paying no attention to design for assembly

ideas during the initial design, and allowing the system to formulate design improvements that would motivate a radically changed redesign. The important finding is that they made no commitment to the original design: they expected it to change.

In subsequent efforts, a stronger, more robust system is necessary. The system must run faster in order to allow the user to explore more possibilities. There must also be an *undo* capability that might allow users to alter previously incorporated alteration steps, or more simply, go back to a previous state in the design process. The lack of such a capability was particularly troublesome for the *during* users. If these users obtained a new design idea, their only recourse was to start a new design. Some of them specifically recommended that an *undo* capability be implemented. Toriya et al. describe a solid modeler that represents the history of a solid design process as a tree of primitive modeling operations [Tor.1]. This allows an *undo* and *redo* capability. Such an approach could alleviate the problems here.

Using a different design domain might also be helpful. At the outset, the Boothroyd-Dewhurst system seemed like the ideal choice because it was feature-based and design quality was quantified. The encoded charts provided a fairly limited number of different design qualitites, however, and therefore did not allow an adequately fine measure of design quality. It would be better to use a more detailed ranking system. Additionally, a different design domain might eliminate some of the CAD system limitations that were experienced. Other domains (e.g. floor plan layout) could more readily allow the alteration of previously accepted design steps. It is uncertain, however, whether these other domains have any related concurrent engineering techniques analagous to the Boothroyd-Dewhurst system. The desire was to use realistic concurrent engineering knowledge in a mechanical design context. For this, the Boothroyd-Dewhurst system was an obvious choice.

Of course, the biggest problem was that there were not enough experimental subjects. Larger subject groups might clarify any distinctions that do exist.

Additionally, there is the question of how the users were prepared for the experiment. Training in basic design for assembly ideas seemed necessary to explain the motivation of the suggestions and to provide the control group with some minimal background they could use to complete their designs. Without training, the control group would not know how to use the standard charts available to them. It is possible, however, that the users were too well trained and could derive little benefit from the fundamental suggestive design rules encoded in the system. This might explain why the best designs were created by users who received relatively few suggestions (see Figure (9.3)). In a real-world setting, users who are ignorant of a design domain are the most likely users of a suggestive system. A test using untrained subjects might be worthwhile.

Moving from the overall results to an analysis of the specific design sessions reveals some potential problems with the two suggestive modes. From the questionnaires we find that the Mode 1 users who felt they were influenced by the system also produced the best designs. Why were some of their designs not as good as they could have been? Recall that users 01 and 07 produced designs that were fairly good, but could have been much better if they displayed a different symmetry. The subjects could have designed an object with the appropriate symmetry; why didn't they? The answer perhaps lies in the three basic suggestive strategies described in Chapter Seven. When three axis symmetry exists, the system will suggest *all symmetries*, ranking them according to their desirability. When some single-axis symmetry exists, the system will make suggestions to *maintain the same symmetry.* When the design is asymmetric, the system will make suggestions to *reduce asymmetry.* If some single axis symmetry exists, the system will not suggest a different single-axis symmetry. The general thinking behind this is that some symmetry is much better than none at all and that once the user has specified some symmetry, it is probably easier to get them to maintain it rather than change it. Is this a wise strategy?

In the pre-suggestive mode, if the user is made aware of the optimal symmetry, then yes, it is an adequate way to output the suggestions. User 07, for example viewed an optimal suggestion and then chose to incorporate one of lesser quality. A s estive system cannot force users to accept suggestions: the user has final authority. User 01, on the other hand, would have benefitted from a suggestion to change the existing symmetry. Recall that this user added a single-axis symmetric feature to a three-axis symmetric design. (Recall from Chapter Six that there are symmetric feature options, where the feature as built by the user is already symmetric.) Although it was not the optimal symmetry for the situation, the system made no suggestions to change it. This user was not aware that the design could be improved.

The strategy is definitely inadequate for the postprocessing mode. User 02 chose the worst possible single-axis symmetry for a long bar-like block. What is worse is that a second postprocess of the redesign would be guided by the *maintain same symmetry* strategy and would not suggest improved symmetries.

The real issue here is that the system should not make suggestions that disagree with the *established* intent of the designer. Although a fundamental premise of this research is that the system is ignorant of the designer's overall intentions, that is, the design goals, there are obviously situations where some small aspect of the user intent can be inferred and used to make the interaction more effective. In the suggestions-during mode, for example, if the user has considered and rejected an optimal suggestion in favor of another, the optimal suggestion should not be offered again. This would be obtrusive and irritating. With reference to the Mode 1 cases described above, if the user is made aware that X-axis symmetry is best and has opted for Y-axis symmetry, then X-axis symmetry should not be suggested again. If, however, the user has specified Y-axis symmetry without knowing that X-axis symmetry is better, then of course X-axis symmetry should be suggested.

This shortcoming is easily corrected for the suggestions-during mode implemented

here: simply encode more sophisticated rule bases that keep track of what the user has and has not considered. In the postprocessing mode it is much more difficult for the system to infer anything about the designer's intent because there is no suggestive interactivity in which the user responds to different suggestions. The implementation developed here instead outputs a large number of uninformed and unrelated suggestions (suggestions are made with respect to every object, as if it was the *only* object in the block) and leaves it to the user to combine them in a way that improves the design. The designs of users 05 and 02 demonstrate that the users can sometimes devise combinations that yield suboptimal and even poor designs.

One way to relieve the user of the error-prone combination task is to take the entire design into account when generating suggestions. To overcome the "gridlock" problem discussed in Chapter Seven, the system may be limited to nongraphical suggestions and general recommendations (e.g. "Flat blocks are best if X-symmetric."). A more appealing solution is to allow the system to do the combining by performing the entire redesign itself. This would require that the system contain knowledge about allowable alterations to the initial form, knowledge to this point considered *segregated* from the suggestive system and used only by the designer. In the context of gauge tools, this knowledge would consist of the possible alernate locations, orientations and symmetries of the individual features. Certain dimensions of the features could not be varied because of the gauging requirements. The user would input an initial design and information about allowable form variations and the system would output a completed optimized redesign. In a more realistically complex domain, formulation of interactivity methods to specify allowable form variations, or conversely required form constraints, might itself be a significant problem.

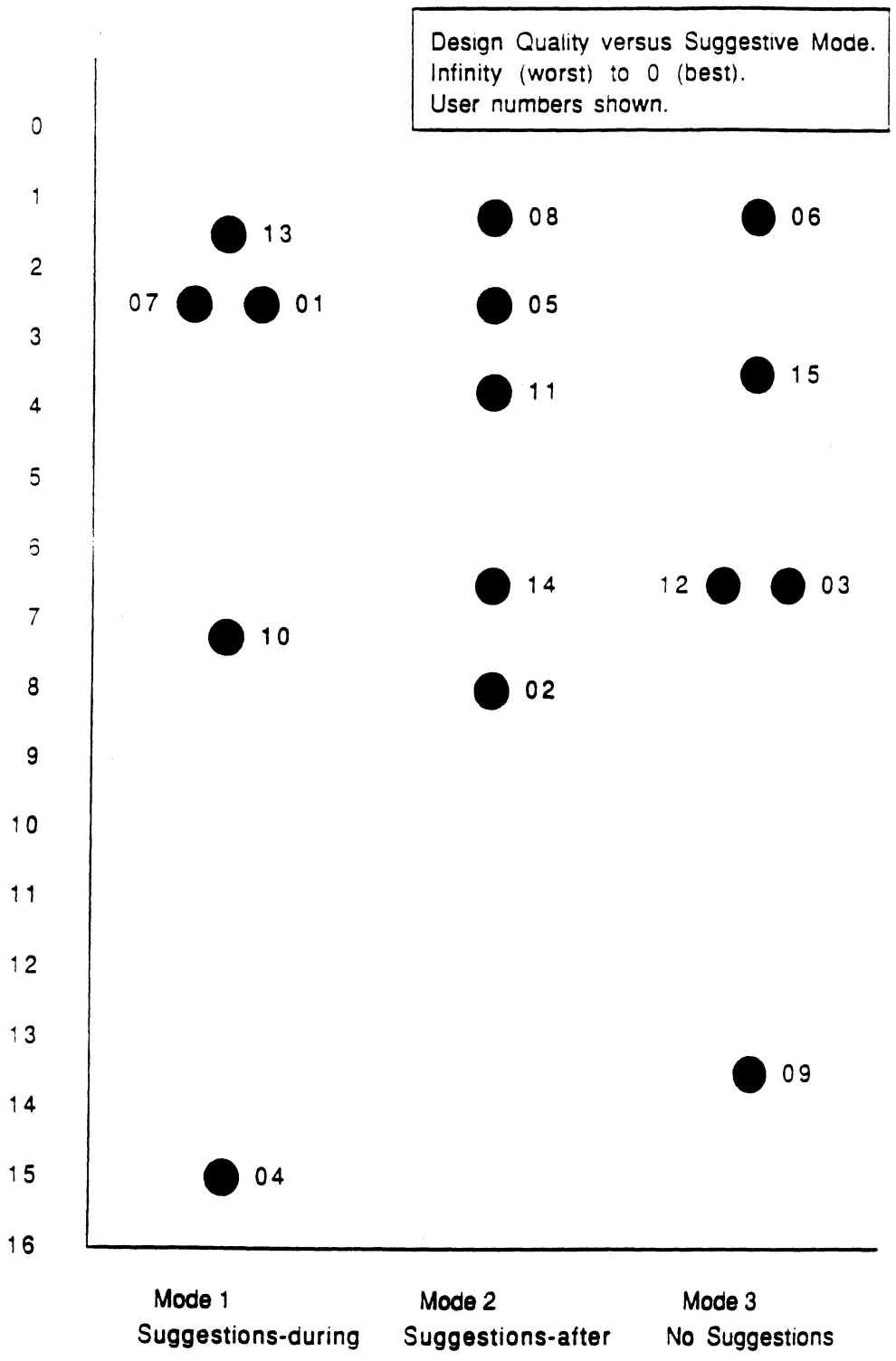


Figure 9.1: Design quality versus suggestive mode (Case A only).

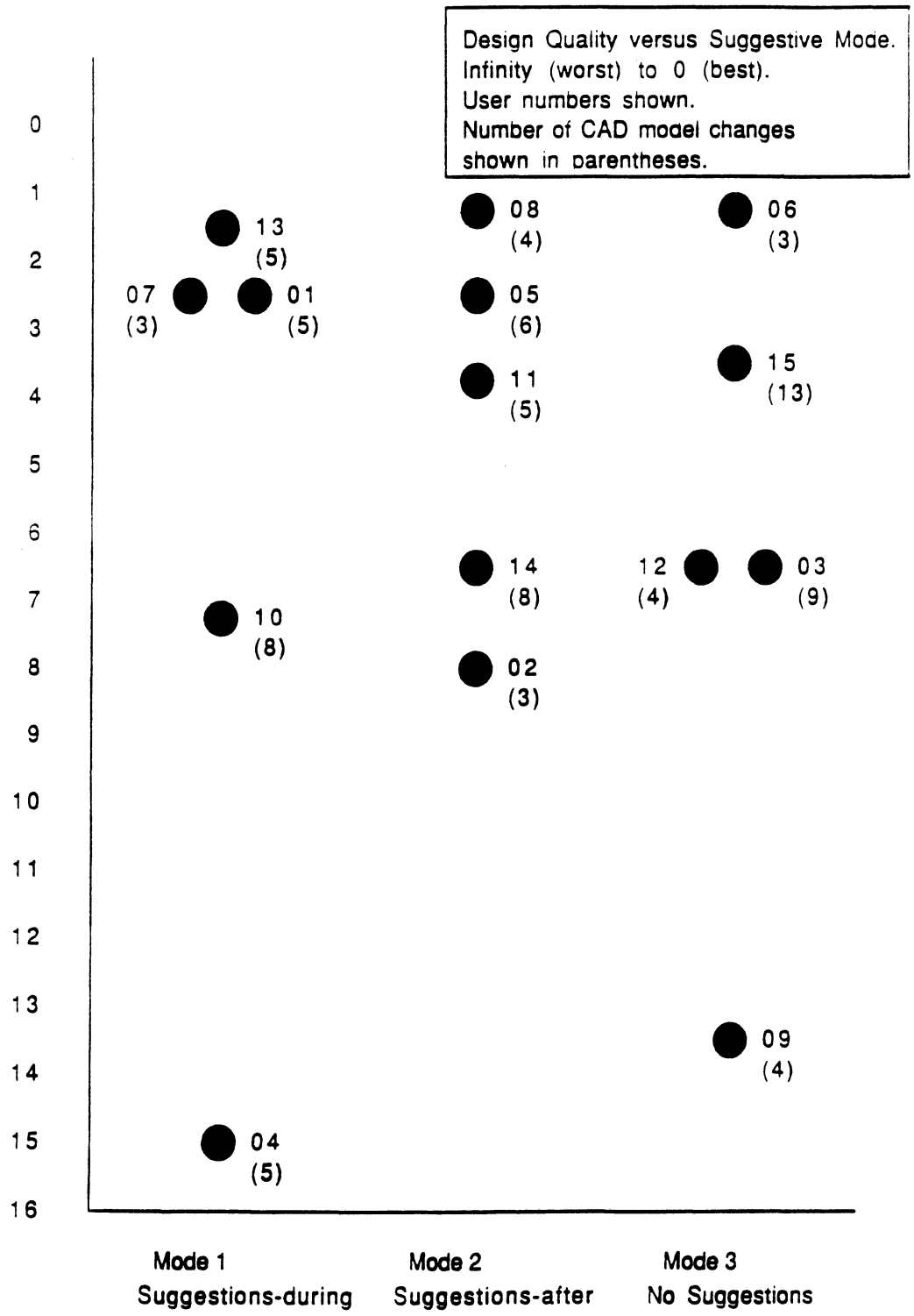


Figure 9.2: Design quality versus suggestive mode with number of CAD model changes shown (Case A only).

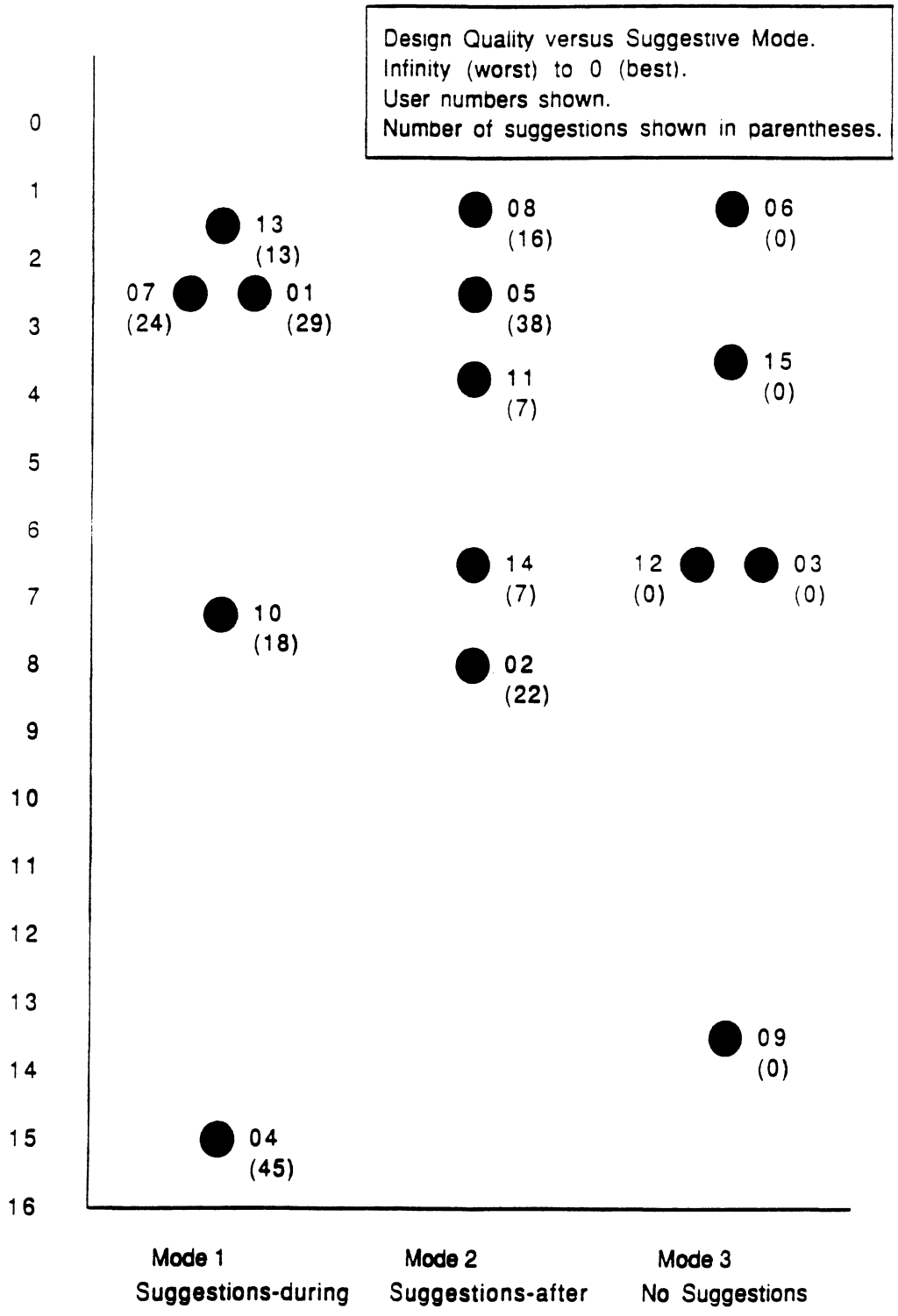


Figure 9.3: Design quality versus suggestive mode with number of suggestions viewed shown (Case A only).

CHAPTER X

CONCLUSION

Introduction

In this chapter we summarize our findings and clarify the overall conclusions derived from the research. First, a broader view of suggestive systems, emphasizing their key functions and characteristics, is outlined. Then, the two major issues stated at the outset are examined in the context of the research results. Major contributions of the work are discussed. Finally, possibilities for further research are offered.

Suggestive Systems: A Broader View

Introduction

This section takes a broader view of suggestive systems by envisioning possible systems that are extensions of the one described in this thesis. These possible future systems and the system described in this thesis will share some general common attributes. These attributes are explained and potential variations of system characteristics related to each attribute are described. In this way, the potential range

of suggestive systems is outlined. The attributes and related characteristics are discussed by taking a structural, procedural, and knowledge base view of suggestive systems.

A Structural View

Figure (10.1) is a structural view. The major components of the system are shown without showing the procedures carried out by them. First, a designing agent can produce alteration steps, called features. Suggestive agents can similarly create alteration steps, called suggestions, usually in response to the actions of the designing agent. One or more design representations are changed by accepting some alteration step as part of the design. We will assume that the designing agent is authoritative, meaning that it controls the creation of the evolving design by choosing which alteration step to accept. The designing agent is also normally a human being. Also, note that it is required that the suggestive agents have the capability to produce alteration steps, and that these alteration steps are dependent upon the design representations. This requirement would exclude systems like those described in Chapter Four, and other influencing situations, such as requiring a designer to work under great stress or in unfamiliar surroundings.

These components exist within a certain context and are related to various design domains. A *domain* is the area of interest about which suggestions are made. Design for assembly is the domain for the system described in this thesis. A *context* is the more general type of design effort. The design for assembly domain is in the context of designing solid objects. The domain of "writing for readability" is in the context of text composition. Each suggestive agent is associated with a domain. In the system developed here, there is only one suggestive agent, associated with design for assembly. If there is more than one suggestive agent, it is reasonable to assume that

they are all in the same context. A combination design for assembly and writing for readability suggestive system, for example, seems highly unlikely.

There should be a separate representation, used by the designing agent, which facilitates the human-computer interactivity and the production of graphical renderings of the design. This representation is related to the other representations used by the suggestive agents. Ideally, each suggestive agent should have its own representation, tailored to its respective domain, but it is possible for two or more suggestive agents to share one multi-purpose representation. This might happen if the alteration steps for the domains are similar. Dixon et al. have also proposed the idea of many design representations related to various design concerns [Dix.1]. In their architecture, a *primary representation* of alteration steps is created from the designer input. *Secondary representations,* corresponding to other aspects of the design (manufacturability evaluation, graphics output, etc.) are derived from the primary representation. In the system described here, the primary representation would be the object list representation of the design, and secondary representations would be portions of each object that are used for various tasks. An alternate view is to consider the underlying CAD model to be the representation used by the designing agent and the object list representation to be the representation used by the suggestive agent. In this view, two representations are used.

Given the requirement of a single designing agent, the only attributes of the above structural description that can vary are the number of suggestive agents and the number of design representations.

A Procedural View

A procedural view of suggestive systems emphasizes what the systems do. The attributes made evident from a procedural view will pertain to the processes performed by the system.

The highest level process performed is the particular suggestive mode that is implemented. If this is considered an attribute, it can be varied by implementing various suggestive modes. Three different modes were implemented in this research effort.

Looking more specifically at the way the designing agent inputs alteration steps reveals two attributes. The first is the number of alteration steps that are available. Obviously, this attribute is varied by simply adding more. A second attribute is the number of alteration steps that can be built before allowing the suggestive agents to generate suggestions.[1] In the familiar feature-based CAD context, if this number is more than one, several *tentative* alteration steps could be built before invoking any suggestion-generation procedures. This attribute includes the possibility of compound alteration steps, where the individual alteration steps have some predefined relationship to one another. The system described in this thesis permits more than one alteration step only when there is a symmetry relation between them. As explained in Chapter Eight, suggestions are always generated from a single feature. Symmetric feature groups are treated like a single feature when generating suggestions. The problem of generating suggestions from one or more unrelated features was not considered here.

The suggestive agents also input alteration steps to the design, a process related to several attributes. First is the idea that the designing agent can omit accepted alteration steps from the design before the generation of suggestions. As discussed in Chapter Seven, this will cause the suggestive agents to generate different suggestions than would be generated if the entire design representation was used. A second, similar attribute is the capability of the designing agent to pick the alteration step that motivates the generation of suggestions. In the system described in this thesis,

---

[1] Although this attribute applies only to the *during* mode implemented here, it is considered important enough to warrant discussion.

either the feature or a system-specified object motivated the suggestions.

Another attribute related to the suggestive agents is the sophistication of the suggestions that are output. This will be explained in a feature-based CAD context, but the idea is very general. In short, what can the suggestive agents generate from a design alteration step? Consider a groove feature and the design for assembly suggestive agent described in Chapter Eight. All suggestions are the result of only duplicating and moving the groove. An added capability would be to alter various dimensions of the suggestion grooves. This would require more suggestive rules to output suggestions that use these parametrically altered grooves appropriately. An even more sophisticated suggestion would be to replace the groove with some other type of alteration step that may be appropriate and would improve the design. Again more productions, of a more complex nature, would be required. Related to the idea of suggestion sophistication is an attribute that would indicate the amount of useful information in each alteration step representation. More detailed representations might contain information about allowable form variations and properties that describe function. This information would certainly help in the generation of more sophisticated suggestions.

To summarize, the attributes related to a procedural view of suggestive systems are: the number of suggestive modes, the number of different features, the number of design agent alteration steps allowed before generating suggestions, the capability to omit accepted alteration steps from the design, the capability to choose the motivating alteration step, the sophistication of the suggestions, and the amount of information represented in each alteration step.

A Knowledge Base View

Two attributes are evident from considering a knowledge base view of suggestive systems. The first is the capability of the system to perform some type of automated

knowledge acquisition. How easily can a new suggestive domain be implemented without explicit programming? This attribute will vary with the amount of user intervention required. One possible method for automated knowledge acquisition is described in the "Further Work" section below. A second attribute is now sensibly the suggestions interact with the rest of the design. The rudimentary geometric reasoning capability described in Chapter Eight is an example of this sensibility. Note that this attribute is different than the suggestion sophistication attribute, which pertains to variety and complexity of the possible suggestions. The sensibility of the suggestions pertains to how informed the system is about the rest of the design when generating suggestions.

Figure (10.2) summarizes this section by listing all of these attributes and showing how they can vary. Certain attributes can vary by discrete values, and these rows of the chart are shown subdivided into individual boxes. If an attribute can vary in a continuous way, the row is shown undivided. The "X" marks indicate where the suggestive system described in this thesis would be placed on the chart.

Two Major Issues

Recall from Chapter One that this research had two goals: to help designers obtain better design ideas during the preliminary design process, and to create a computational tool for concurrent engineering. The hypothesis was that these two goals could be achieved with a system that made intelligent purposeful suggestions to designers during the preliminary design phase. These suggestions embody information that is different than the information normally used by the designer when creating the object. Most importantly, the suggestions are not based on design specifications. This hypothesis naturally implies a basic question: will these suggestions actually influence designers? The data collected in the user tests performed here in-

dicate (perhaps weakly) that suggestions can favorably influence designers and help them create better designs.

The suggestions seemed to give users better design ideas. Although the data did not show any gross effect, for the small number of subjects it did show that user groups that received suggestions had a higher fraction of users with good designs. Unfortunately, the results do not indicate the best method to provide suggestions. Further research is required to answer this question.

Regarding the effectiveness of an intelligent suggestive CAD system as a tool for concurrent engineering, it was seen that most of the users who received suggestions stopped using the original design for assembly charts very early in the design process, and relied solely on the system's assistance. They seemed to prefer the system over the charts. Their completed designs were as good or better than those of the unaided users. For the user groups tested, the computational suggestive system seemed to provide as much help as the charts.

Contributions to Other Disciplines

In the general area of *design*, we have developed a system that provides design improvement information to a designer during the preliminary design phase. Although refinement and additional testing are required, indications are that the system can help designers create better designs.

An architecture for an intelligent, feature-based *computer-aided design* system has been implemented. This architecture allows a system that can analyze an evolving design and actively suggest changes to the design to improve it with respect to the analysis domain. The architecture is based on a fe re representation of the design and a production rule representation of design knowledge.

The contributions to the area of *artificial intelligence*, while not significant, are worthy of note. The work here was primarily an application of existing artificial

intelligence programming techniques that are used to encode heuristics. Production rules for making design improvement suggestions were devised. Since these suggestions are intended to give designers new design ideas, the rules that cause them are both inherently heuristic and heuristic in the sense that the designer's subsequent actions are unknown. This additional uncertainty makes formulation of a suitable rule base very difficult.

Unfortunately, the results in the area of *human-computer interaction* were not as conclusive as originally hoped for. Most of the users that received suggestions created high quality designs. There are no conclusive results, however, that indicate which of two suggestive modes is best. It was evident that a feature library is an effective interface for a CAD system. Users can design quickly by using sensible entities that seem natural to them. The idea of incorporation of features, however, was a problem. It was seen that users often desire to make wholesale changes to a design in the middle of a session, and the architecture used here really didn't allow this (see Chapters Five and Nine). An "undo" capability is absolutely necessary.

The goal of the user testing was to answer a very general question: do suggestive systems help? Our interest was in determining if suggestive systems provide some advantage to designers, and if so, which of two very different systems is better. We were not interested in fundamental, low-level cognitive processes of the designers. It is unfortunate that the data collected does not show a more distinct gross effect with the small number of subjects. There is a trend indicating the potential benefit of suggestive modes and this effect might be clarified by an experiment with more subjects. Once a general effect is demonstrated with more conclusive data, it would be interesting to study the lower-level cognitive processes associated with designing while influenced by suggestions.

Further Work

Finally, we discuss possibilities for future work. Potential subsequent efforts seem to be of two basic types. One is to extend and improve suggestive systems like the one implemented here. The other is to learn more about how designers can be favorably influenced during the design process. The extensions and improvements will be discussed first.

Extensions and Improvements

Of course one obvious extension is to implement suggestive systems for other contexts. This might reveal issues not brought to light with the implementations of the system described in the previous chapters. Within any one context, better systems are those that have attributes that fall to the right in each row of Figure (10.2). If we limit our consideration to a feature-based CAD context, many of the extensions and improvements required for this more favorable rating would be fairly easy to achieve. More suggestive modes could be added by changing the high-level calling procedures. Additional features could be created and added to the system implementation. The capabilities to omit objects from the representation and to choose the motivating alteration steps are easy to achieve as long as there are correspondences between the representation used for the interface and the representations used by the suggestive agents.

Allowing the designer to pick an incorporated object to motivate the generation of suggestions raises the special problem of properly taking into account the presence of any tentative features. Note that feature representations are not part of the entire design representation, but still must influence the generation of suggestions. This problem was handled in the system implementations developed so far with the concept of an *interim design representation*, consisting of the feature and the object list. Increasing the number of tentative features that are built before generating

suggestions would make it more difficult to correctly interpret the important characteristics of the interim design. Another potentially difficult problem is to motivate suggestions with more than one unrelated feature alteration step (or incorporated object, or some combination of both). This would be equivalent to generating suggestions with respect to an entire gross portion of the design that is made up of many alteration steps. Information about allowable form variations would be useful here also. This capability was not considered in this research.

Other obviously useful extensions would be more difficult to implement. Increasing the number of suggestive domains and design representations might require some type of parallel computation to efficiently generate suggestions about each domain simultaneously. The system architecture must change to accomodate this. A question arises as to how the suggestions should be generated. Should suggestions be output for each domain separately, or should hybrid suggestions that are a compromise between suggestive domains be produced?

Implementing some type of automated knowledge acquisition would also be difficult. As seen in this thesis, suggestive systems first analyze a design to determine its quality and then suggest ways to modify the design to improve it. The analysis portion is basically a classification problem. Examination of the design properties allows the system to place the design in one of several classes, with the classes having different design qualities. The suggested alterations would cause placement in a more favorable class. A significant first effort would be to automate the acquisition of knowledge required for the classification process. A useful system would be one in which a number of designs with a range of corresponding design qualitites are used to automatically generate a classification system. The system would use the design representations and corresponding qualities to determine classification rules. The qualitites would pertain to a particular suggestive domain with knowledge that was not yet encoded. Once generated, the classification rules could be used to classify

other designs.

Boose [Boos.1] has devised a method of knowledge acquisition for classification problems based on the "personal construct psychology" introduced by Kelley [Kel.1]. This method could possibly be extended to create design classification systems. In the method, objects considered for the automated knowledge acquisition are taken three at a time to determine how one is unlike the other two. This "unlikeness" then becomes a property pertinent to the classification process. All objects are then given a numerical value for this property, indicating the degee to which they have the property. Several triplets are considered in this way in order to determine an adequate number of properties for the classification. Consider, as Boose does in [Boos.1], the problem of classifying potential vacation cities. Properties that might appear include cost and average temperature. Assigning values to these properties for each city and comparing the values could yield several heuristic rules. Cities that are warm, for example, are usually cheaper.

In a feature-based CAD context, the objects used in the comparison would be completed designs of known quality. The determined heuristics would provide a way to determine the quality of other designs. In the system described by Boose, a human user performs the comparison and the system automatically derives the rules. If the designs are represented appropriately, the comparison process could be automated to a certain degree, with the system making comparisons when it can and gracefully relying on human intervention when it cannot.

Another difficult future effort would be the postprocess redesign system proposed in Chapter Nine. Users would be able to create an initial design and specify allowable variations to the alteration steps. The system would use this information to create an optimized redesign. One problem is devising a way for the users to specify allowable variations. It will be difficult for the users to know how the form can be changed while still satisfying the function they intended. Another even more advanced possibility

is to specify a design in terms of its functions only. With this input, a system with design alteration steps associated with various functions might be able to perform simple initial designs with no human assistance. The major technical problem is that for all but a few design domains (e.g. for digital circuit design truth tables specify function), there exists no means to specify the function of a design.

Favorably Influencing Designers

Aside from developing better, more powerful suggestive systems, it would be interesting to investigate different ways to influence designers during the design process.

The suggestive system described here made suggestions about individual parts of the design. An alternate technique is to make suggestions about entire designs. Consider, for example, an experiment in which subjects are assigned the task of designing some complex object, say some type of metal removal machine tool. Along with the design specifications they are shown other metal removal tools that clearly do not meet the given functional specifications. What attributes of the other designs will appear in their design solution? What attributes will not? How is their design solution dependent upon their previous exposure to similar designs? Will subjects that are shown designs that are considered to be good also produce good designs? Answers to these questions might indicate a way to promote accepted "good design practices" by influencing designers with previous successful designs.

A related experiment might attempt to encourage *novelty.*[2] The goal here is to cause designers to produce designs that are not necessarily better, but certainly out of the norm. Consider now a group of experienced machine tool designers. Also assume that given a set of fairly standard machine tool specifications, because of accepted

---

[2] Recall from Chapter Two that Malhotra et al. have proposed a method to measure the similar abstract notion of *originality.* See [Mal.1].

design practices and conventions, all will produce very similar designs. What can be done to encourage the creation of novel solutions? Can the designers be stimulated in some way? Could other unrelated designs provide fresh ideas? If so, how are their design solutions influenced by the unrelated designs. Answers to these questions could provide a way to generate a larger number of innovative designs.

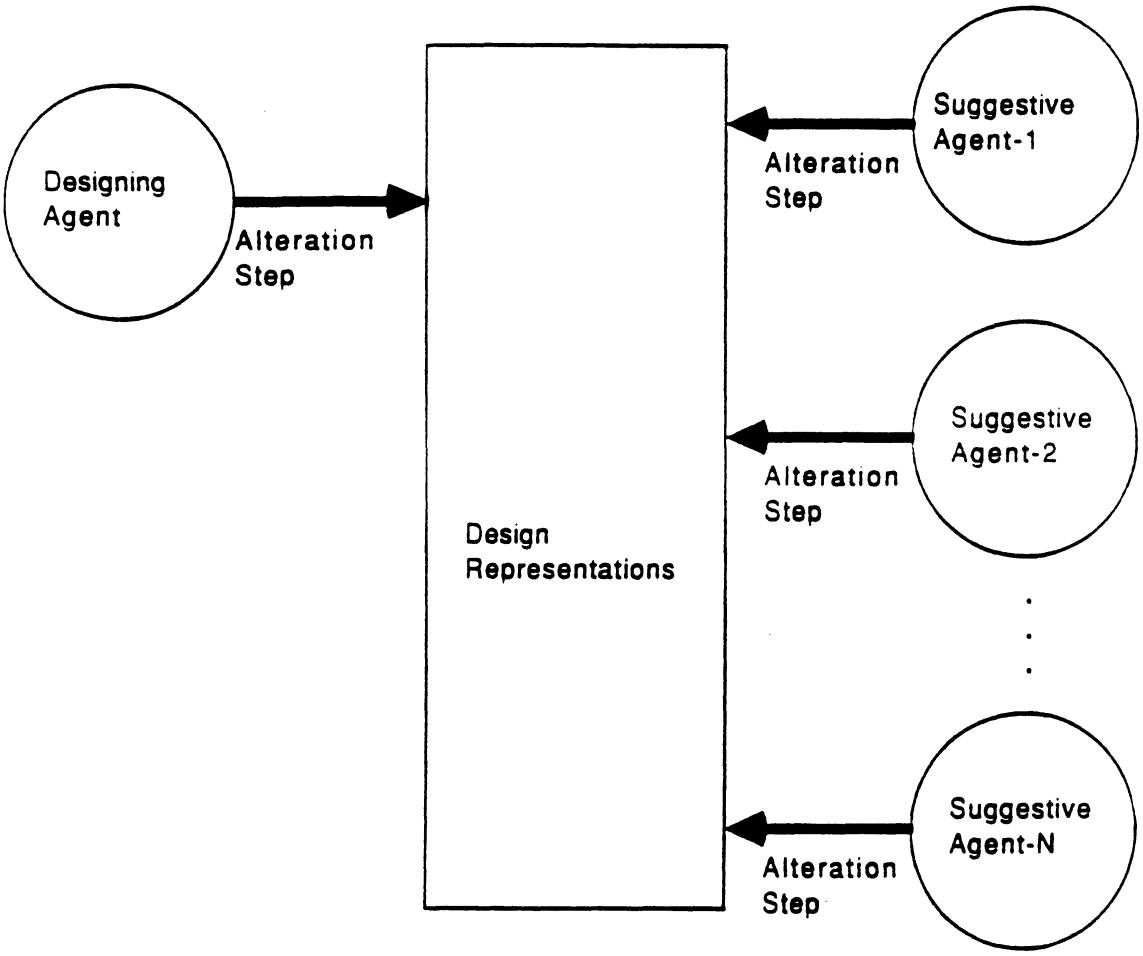


Figure 10.1: A structural view of suggestive systems.

Number of suggestive agents	1 X	2	3	4	5	. . .
Number of design representations	1	2 X	3	4	5	. . .
Number of suggestive modes	1	2	3 X	4	5	. . .
Number of features	1	2	3	4	5 X	. . .
Number of features before suggestion generation	1 X	2	3	4	5	. . .
Omit objects	No X					Yes
Choose motivating alteration step	No X					Yes
Suggestion sophistication	X					
Information content of each alteration step	X					
Knowledge acquisition	X					
Suggestion Sensibility	X					

Figure 10.2: Variations of the attributes of suggestive systems. X's refer to the system described in this thesis.

APPENDICES

APPENDIX A

INTRODUCTORY CAD ASSIGNMENT

FEATURE-BASED CAD ASSIGNMENT

You task is to produce a line printer plot as shown below. Please note the following points:

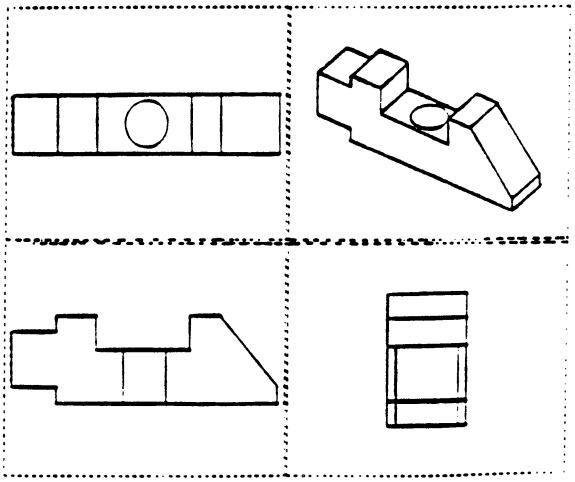
1. You must do a hidden line removal in the isometric viewport (i.e. upper right box).
2. The initial block is generally long. Refer to your Boothroyd/Dewhurst handouts for a definition of "long".
3. The steps are symmetric about the X-axis of the block. (Again, refer to your handout for a definition of "X-axis").
4. All other features are asymmetric. They are not made with the symmetric feature options.
5. Just make a model with the features in the same configuration. Dont worry about precise dimensions.

What you hand in (Due 29 March 1988):

1. The plot.
2. On the plot, write the name you gave to your model (I suggest the first six characters of your last name), and your full name. Write these on the border of the plot outside the viewports.
3. On a seperate attached sheet of paper, describe any difficulties you had, and the times you will be available for a 2 hour user testing session.

Remember the log-on procedure:

1. Turn on the machine and press carriage return.
2. USERNAME: DESLAB, PASSWORD: ME516 (followed by carriage returns.)
3. DDM (carriage return)
4. 1 RETRIEVE A FILED MODEL
5. MODEL NAME: STARTE
6. _!_ DESLABCAD
7. Remember to keep CapsLock and ask Craig Thams (in the back room) for any help.



APPENDIX B

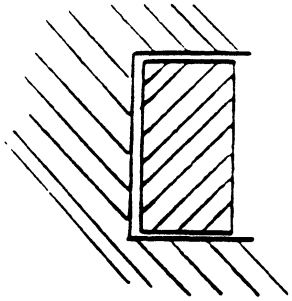
DESIGN ASSIGNMENT AND QUESTIONNAIRE

FEATURE-BASED DESIGN ASSIGNMENT

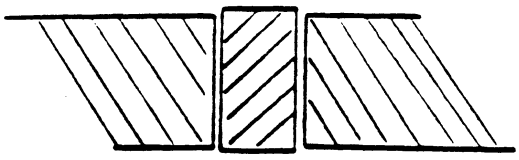
User Number:
Suggestive program:

Introduction

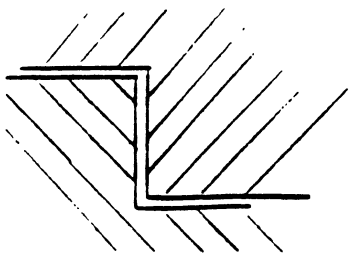
Consider the idea of *gauge pairs* for tolerance. A groove and a box, for example, are gauge pairs. The groove can gauge one dimension of the box since if the box fits into the groove, the box dimension satisfies an upper tolerance. On the other hand, if the box does not fit into the groove, the box dimension does not satisfy an upper tolerance. A box of known size may similarly gauge a groove.

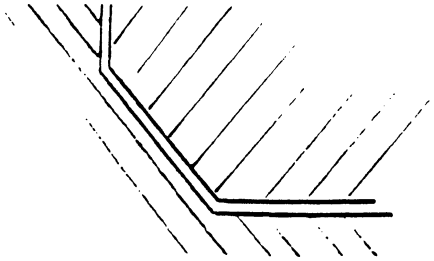


A hole and a peg are gauge pairs. If the peg fits into a known hole diameter, the peg diameter satisfies an upper tolerance. If the hole fits over a known peg diameter, the hole diameter satisfies a lower tolerance.



Steps and chamfers are gauge pairs with other steps and chamfers. They are not gauge pairs in the sense that they can determine if dimensions satisfy tolerances; rather they can be used to determine if the shapes of the pairs match. Fit the gauge against the gauged part, and if any light passes through, the gauged part is unacceptable.





Design Specifications

Design a multi-purpose gauge tool that will gauge required clearances of other parts in a machine. This gauge tool will be fed with a vibratory bowl feeder, to be placed and stored in a larger machine where all of the gauged parts are located. A service person will later find the gauge tool when performing routine gauging service. The dimensions given below are approximate to the extent that you can estimate them while designing free hand.

Design Assignment

Design the gauge tool for the following two cases:

Case A:

The gauge tool must perform the following gauging operations. It must gauge that a groove is greater than three units wide. It must gauge two boxes. One that has a dimension of about two units, and one that has a dimension of about four units. It must gauge that a peg is less than about 2 units in diameter. It must gauge either a step or a chamfer (you choose), with characteristic dimension of about 1.5 units.

Optimize the assemblability of the tool.

Models created for Case A (in order of creation):

Case B:

The gauge tool must perform the following gauging operations. It must gauge a chamfer with characteristic dimension of about 1 unit and a step with characteristic dimension of about three units. It must gauge that a peg is less than about 1.5 units in diameter. It must gauge a box with dimension about two units.

Optimize the assemblability, minimize the volume, and minimize the number of machining operations required to produce the gauge tool.

Models created for Case B (in order of creation):

Obtain line printer plots for both cases and annotate them to explain how

the design specifications are met.

SUGGESTION PROGRAM QUESTIONAIRRE

User Number:
Suggestive program:

1. Did the suggestions influence you? Please explain.

2. Did you view all of the suggestions for a feature before deciding which one to use? Please explain.

3. Did your ideas about the design change repeatedly as you viewed more suggestions? Please explain.

4. Did you do any total redesigns? How did your design evolve?

5. Were the graphical suggestions helpful? Was the text explanation helpful?

6. Did you enjoy using the program? What should be changed? Please comment.

7. Did you refer to the Boothroyd/Dewhurst charts while designing? Please explain the circumstances.

BIBLIOGRAPHY

BIBLIOGRAPHY

[And.1] Anderson, J. R., Boyle, C. F., Reiser, B. J., "Intelligent Tutoring Systems," *Science*, Vol. 228, April 1985, pps. 456-462.

[Andr.1] Andreasen, M. M., Kahler, S., Lund, T., *Design for Assembly*, IFS (Publications), and Springer-Verlag, Berlin, 1983.

[Ben.1] Bennet, J. S., Englemore, R. S., "Sacon: A Knowledge-Based Consultant for Structural Analysis," *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, August 1979, Tokyo, Japan.

[Boo.1] Boothroyd, G., Dewhurst, P., *Design for Assembly–A Designer's Handbook*, Department of Mechanical Engineering, University of Massachusetts at Amherst, 1983.

[Boo.2] Boothroyd, G., Ho, C.,"Natural Resting Aspects of Parts for Automatic Handling," American Society of Mechanical Engineers, Paper 76-WA/Prod-40, New York, 1976.

[Boo.3] Boothroyd, G., Poli, C., Murch, L., *Automatic Assembly*, Marcel Dekker Inc., New York and Basel, 1982.

[Boo.4] Boothroyd, G., "Making It Simple-Design for Assembly," *Mechanical Engineering*, Vol. 110, No. 2, February 1988, pps. 28-31.

[Boos.1] Boose, J. H., *Expertise Transfer for Expert System Design*, Elsevier, New York, 1986.

[Bra.1] Brady, J. T., "A Theory of Productivity in the Creative Process," *IEEE Computer Graphics and Applications*, Vol. 6, No. 5, May 1986, pps. 25-34.

[Bri.1] Brimson, J. A., Downey, P. J., "Feature Technology: A Key to Manufacturing Integration," *CIM Review*, Vol. 2, No. 3, Spring 1986, pps. 21-27.

[Bro.1] Brown, D. [illegible], Chandrasekaran, B., "Knowledge and Control for a Mechanical D[illegible]n Expert System," *IEEE Computer*, Vol. 19, No. 7, July 1986, pps. [illegible]00.

[Bur.1] Burton, R. R., Brown, J. S., "An Investigation of Computer Coaching for Informal Learning Activities," in *Intelligent Tutoring Systems*, (eds. D. Sleeman and J. S. Brown), Academic Press, London and New York, 1982, pps. 79-98.

[Car.1] Card, S. K., Moran, T. P., Newell, A., *The Psychology of Human-Computer Interaction*, Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1983.

[Cog.1] Cognition Inc., *Mechanical Advantage 1000 Product Description*, 900 Technology Park Drive, Billerica, MA 01821, 1985.

[Dav.1] Davies, R., Talbot, R. J., "Experiencing Ideas: Identity, Insight, and the Imago," *Design Studies*, Vol. 8, No. 1, January, 1987, pps. 17-25.

[deH.1] den Hamer, H. E., *Interordering-A New Method of Component Orientation*, Elsevier Scientific Publishing Co., Amsterdam, and New York, 1980.

[Dew.1] Dewhurst, P., Boothroyd, G., "Computer-Aided Design for Assembly," *Assembly Engineering*, February, 1983.

[Dix.1] Dixon, J. R., Cunningham, J. J., Simmons, M. K., "Research in Designing with Features," *Proceedings of the International Federation of Information Processing, Working Group 5.2, Workshop on Intelligent CAD*, October 6-8, 1987, Cambridge, MA, to be published by North-Holland, New York and Amsterdam.

[Dix.2] Dixon, J. R., Simmons, M. K., "Expert Systems for Engineering Design: Standard V-belt Design as an Example of the Design-Evaluate-Redesign Architecture," *Proceedings of the 1984 ASME Computers in Engineering Conference*, Las Vegas, NV, August 12-15, 1984, American Society of Mechanical Engineers, New York.

[Dwi.1] Dwivedi, S. N., Klein, B. K., "Design for Manufacturability Makes Dollars and Sense," *CIM Review*, Vol. 2, No. 3, Spring 1986, pps. 53-59.

[Dye.1] Dyer, M. G., Flowers, M., Hodges, J., "EDISON: An Engineering Design Invention System Operating Naively," *Artificial Intelligence in Engineering*, Vol. 1, No. 1, July 1986, pps. 36-44.

[Ebe.1] Eberts, R., Lang, G. T., Gabel, M., "Expert/Novice Differences in Designing with a CAD System," *Proceedings of the 1987 IEEE International Conference on Systems, Man, and Cybernetics*, 87CH2503-1, Vol. 3, Alexandria, VA, October 20-23, 1987, pps. 985-989.

[Eri.1] Ericsson, K. A., Simon, H. A., *Protocol Analysis: Verbal Reports as Data*, MIT Press, Cambridge, MA, 1984.

[Eva.1] Evans, B., "Simultaneous Engineering," *Mechanical Engineering*, Vol. 110, No. 2, February 1988, pps. 38-39.

[For.1] Forgy, C. L., *OPS5 User's Manual*, Carnegie-Mellon University, July, 1981.

[Gen.1] General Electric Calma Co., *System Reference Manual for DDM & DIMENSION III on the VAX and Apollo*, 9805 Scranton Rd., San Diego, Calif. 92121-1765, 1983.

[Gor.1] Gordon, W. J. J., *Synectics-The Development of Creative Capacity*, Collier-MacMillan Ltd. London, 1968.

[Gos.1] Gossard, D. C., Zuffante, R. P., Sakurai, H., "Representing Dimensions, Tolerances, and Features in MCAE Systems," *IEEE Computer Graphics and Applications*, Vol. 8, No. 2, March 1988, pps. 51-59.

[Hen.1] Henderson, M. R., *Extraction of Feature Information from Three Dimensional CAD Data*, Phd Thesis, Purdue University, Department of Mechanical Engineering, 1984.

[How.1] Howe, A. E., Cohen, P. R., Dixon, J. R., Simmons, M. K., "Dominic: A Domain-Independent Program for Mechanical Engineering Design," *Artificial Intelligence in Engineering*, Vol. 1, No. 1, July, 1986, pps. 23-28.

[ICA.1] ICAD Inc., *ICAD System Product Description*, 1000 Massachusetts Avenue, Cambridge, MA 02138, 1987.

[Jak.1] Jakiela, M., Papalambros, P., Ulsoy, A. G., "Programming Optimal Suggestions in the Design Concept Phase: Application to the Boothroyd Assembly Charts," *ASME Journal of Mechanisms, Transmissions, and Automation in Design*, Vol. 107, No. 2, June 1985, pps. 285-291.

[Jak.2] Jakiela, M., Papalambros, P., "A Design for Assembly Optimal Suggestion Expert System," *Proceedings of the Seventh International Conference on Assembly Automation*, IFS (Publications), and Springer-Verlag, Berlin, 1986, pps. 341-350.

[Jak.3] Jakiela, M. J., Papalambros, P. Y., "Design and Implementation of a Prototype "Intelligent" CAD System," American Society of Mechanical Engineers, Paper 87-DAC-51, New York, 1987, also *ASME Journal of Mechanisms, Transmissions, and Automation in Design* (to appear).

[Jon.1] Jones, J. C., *Design Methods: Seeds of Human Futures*, Wiley-Interscience, London and New York, 1970.

[Kel.1] Kelley, G. A., *The Psychology of Personal Constructs*, Norton, New York, 1955.

[Kro.1] Kroll, E., Lenz, E., Wolberg, J. R., "A Knowledge-Based Solution to the Design-For-Assembly Problem," *Proceedings of the Third International Conference on Computer-Aided Production Engineering*, June 1-3, 1988, Ann Arbor, Michigan, The Society of Manufacturing Engineers, pps. 261-268.

[Kul.1] Kulkarni, V. M., Dixon, J. R., Simmons, M. K., Sunderland, J. E., "Expert Systems for Design: The Design of Heat Fins as an Example of Conflicting Subgoals and the Use of Dependencies," *Proceedings of the 1985 ASME Computers in Engineering Conference*, Boston, MA, August 4-8, 1985, American Society of Mechanical Engineers, New York.

[Lan.1] Lansdown, J., "The Creative Aspects of CAD: A Possible Approach," *Design Studies*, Vol. 8, No. 2, April 1987, pps. 76-81.

[Lig.1] Light, R., Gossard, D., "Modification of Geometric Models through Variational Geometry," *Computer Aided Design*, Vol. 14, No. 4, July 1982, pps. 209-214.

[Lub.1] Luby, S. C., Dixon, J. R., Simmons, M. K., "Creating and Using a Features Data Base," *Computers in Mechanical Engineering*, November, 1986, pps. 25-33.

[Mal.1] Malhotra, A., Thomas, J. C., Carroll, J. M., Miller, L. A., "Cognitive Processes in Design," *International Journal of Man-Machine Studies* (1980) **12**, pps. 119-140.

[Mil.1] Miles L. D., *Techniques of Value Analysis and Engineering*, MacGraw-Hill Inc., New York, 1972.

[Mill.1] Miller, M. L., "A Structured Planning and Debugging Environment for Elementary Programming," in *Intelligent Tutoring Systems*, (eds. D. Sleeman and J. S. Brown), Academic Press, London and New York, 1982, pps. 119-135.

[Mit.1] Mittal, S., Dym, C. L., Morjaria, M., "Pride: An Expert System for the Design of Paper Handling Systems," *IEEE Computer*, Vol. 19, No. 7, July 1986, pps. 102-114.

[Mor.1] Morjaria, M., Mittal, S., Dym, C. L., "Interactive Graphics in Expert Systems for Engineering Applications," *Proceedings of the 1985 International Computers in Engineering Conference*, Boston, MA, August, 1985.

[Mur.1] Murthy, S. S., Addanki, S., "PROMPT: An Innovative Design Tool," *Proceedings of the Sixth National Conference on Artificial Intelligence*, July 13-17, 1987, Seattle, WA, American Association for Artificial Intelligence, pps. 637-642.

[Nor.1] Norris, K. W., "The Morphological Approach to Engineering Design," *Conference on Design Methods*, The MacMillan Co., New York, 1963.

[Osb.1] Osborn, A. F., *Applied Imagination*, Scribners, New York, 1963.

[Oxm.1] Oxman, R., Gero, J. S., "Using an Expert System for Design Diagnosis and Design Synthesis," *Expert Systems*, Vol. 4, No. 1, February 1987, pps. 4-15.

[Pra.1] Pratt, M. J., "Solid Modeling and the Interface Between Design and Manufacture," *IEEE Computer Graphics and Applications*, Vol. 4, No. 7, July 1984, pps. 52-59.

[Pre.1] Premack, T., "A Rule Based Computer Aided Design System," *National Aeronautics and Space Administration*, Technical Memorandum 86241, Goddard Space Flight Center, Greenbelt, Maryland, 1986.

[Pres.1] Pressman, R. S., *Software Engineering: A Practitioner's Approach*, McGraw-Hill, New York, 1982.

[Rin.1] Rinderle, J. R., Suh, N. P.,"Measures of Functional Coupling in Design," *ASME Journal of Engineering for Industry*, Vol. 104, November 1982, pps. 383-388.

[Run.1] Runciman, C., Swift, K., "Expert System Guides CAD for Automatic Assembly," *Assembly Automation*, Vol. 5, No. 3, August 1985, pps. 147-150.

[Ryc.1] Rychener, M. D., "Expert Systems for Engineering Design," *Expert Systems*, Vol. 2, No. 1, January 1985, pps. 30-44.

[Shr.1] Shrager, J., Finin, T., "An Expert System that Volunteers Advice," *Proceedings of the National Conference on Artificial Intelligence*, August 18-20, 1982, Pittsburgh, PA, American Association for Artificial Intelligence, pps. 339-340.

[Sim.1] Simon, H. A., *The Sciences of the Artificial*, (2d. ed.), MIT Press, Cambridge, MA, 1981.

[Sle.1] Sleeman, D., Brown, J. S., (eds.), *Intelligent Tutoring Systems*, Academic Press, London and New York. 1982.

[Ste.1] Stefik, M., Foster, G., Bobrow, D. G., Kahn, K., Lanning, S., Suchman, L., "Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings," *Communications of the ACM*, Vol. 30, No. 1, January, 1987, pps. 32-47.

[Sto.1] Stoll, H. W., "Design for Manufacture," *Manufacturing Engineering*, January, 1988, pps. 67-73.

[Stu.1] Stults, R., "Using Video in Design Environments," in *Results from the NSF Workshop on the Design Process*, (ed. M. B. Waldron) Ohio State University, Columbus, Ohio, 1987, pps. 103-105.

[Suh.1] Suh, N. P., Bell, A. C., and Gossard, D. C., "On an Axiomatic Approach to Manufacturing and Manufacturing Systems," *ASME Journal of Engineering for Industry*, Vol. 100, No. 2, May 1978, pps. 127-130.

[Swi.1] Swift, K. G., Firth, P. A., "Knowledge Based Expert Systems in Design for Automatic Handling," *Proceedings of the Fifth International Conference on Assembly Automation*, May 22-24, 1984, Paris, France, IFS (Publications), and North-Holland, Amsterdam, pps. 117-126.

[Tek.1] Teknowledge Inc., *M.1 System Documentation*, 525 University Ave., Palo Alto, CA 94301-1982.

[Tho.1] Thom, R., Zeeman, E. C., "Catastrophe Theory: Its Present State and Future Perspectives," *Springer Lecture Notes in Mathematics*, No. 488, (1975).

[Tor.1] Toriya, H., Satoh, T., Ueda, K., Chiyokura, H., "UNDO and REDO operations for Solid Modeling," *IEEE Computer Graphics and Applications*, Vol. 6, No. 4, April 1986, pps. 35-42.

[Ull.1] Ullman, D. G., Stauffer, L. A., Dietterich, T. G., "Preliminary Results of an Experimental Study of the Mechanical Design Process," in *Results from the NSF Workshop on the Design Process*, (ed. M. B. Waldron) Ohio State University, Columbus, Ohio, 1987, pps. 145-186.

[Ulr.1] Ulrich, K., Seering, W., "Conceptual Design as Novel Combination of Existing Device Features," *Proceedings of the ASME Design Automation Conference, Advances in Design Automation*, Vol. 1, September 27-30, 1987, Boston, MA., American Society of Mechanical Engineers, New York, pps. 295-300.

[Ulr.2] Ulrich, K., Seering, W., "Achieving Multiple Goals in Conceptual Design," *Proceedings of the International Federation of Information Processing, Working Group 5.2, Workshop on Intelligent CAD*, October 6-8, 1987, Cambridge, MA, to be published by North-Holland, New York and Amsterdam.

[Wal.1] Waldron, M. B., Waldron, K. J., "A Time Sequence Study of a Complex Mechanical System Design," *Design Studies*, Vol. 9, No. 2, April 1988, pps. 95-106.

[Wil.1] Wilde, D. J., *Globally Optimal Design*, Wiley Interscience, New York, 1978.

[Win.1] Winston, P. H., Horn, B. K. P., *Lisp*, (2d. ed.), Addison-Wesley, Reading, MA. 1984.

[Woo.1] Woo, T. C., "Interfacing Solid Modeling to CAD and CAM: Data Structures and Algorithms for Decomposing a Solid," *Computer Integrated Manufacturing*, November 1983, pps. 39-45.

[Wro.1] Wrona, S. K., Olszynski B., "Urbigraph: A Computer Aid for Design Participation," *Design Studies*, Vol. 6, No. 3, July 1985, pps. 134-140.

[Zwi.1] Zwicky, F., "The Morphological Method of Analysis and Construction." *Courant*, Anniversary Volume, 1948.