Design Reuse in a CAD Environment

A thesis submitted for the degree of Doctor of Philosophy

By

Peter T. J. Andrews

Department of Manufacturing and Engineering Systems,
Brunel University

January 1999

Abstract

For many companies, design related information mainly exists as rooms of paper-based archives, typically in the form of manufacturing drawings and technical specifications. This 'static' information cannot be easily reused.

The work presented in this thesis proposes a methodology to ease this problem. It defines and implements a computer-based design tool that will enable existing design families to be transformed into 'dynamic' CAD-based models for the Conceptual, Embodiment and Detailed stages of the design process.

Two novel concepts are proposed here, i) the use of a Function Means Tree to store Conceptual and Embodiment design and ii) a Variant Method to represent Detailed design. In this way a definite link between the more abstract conceptual and the concrete detailed design stages is realised by linking individual detailed designs to means in the Function Means Tree. The use of the Variant Method, incorporating 'state-of-the-art' developments in Solid Modelling, Feature-Based Design and Parametric Design, allows an entire family of designs to be represented by a single Master Model. Therefore, instances of this Master Model need only be stored as a set of design parameters. This enables current design families and new design cases to be more created more efficiently.

Industrial Case Studies, including a Lathe Chuck family, a Drive-End casting and a family of Filtration Systems are given to prove the methodology.

Contents

List of Figures

Acknowledgements

The author would like to thank Dr. Sangarapillai Sivaloganathan for his untiring guidance throughout this project. His breadth and depth of knowledge, as well as his endless Tamil sayings have helped make this project a success. Although I am sure he would only tell me that 'Hard work is only rewarded by more hard work' (Sivaloganathan 1999).

Many thanks are also due to Dr. Tamer M.M. Shahin, Mr. S. Srikumaran and Mrs. E. Sivakumar of the Engineering Design Group, Brunel University, UK, for their assistance, support and entertainment.

Last, but not least, I would like to thank my family and close friends, and especially Michelle.

Chapter 1

Introduction

1.1 Computer Aided Design

The advent of computers in engineering has made significant progress in the past few decades. It has opened up several new opportunities, which would not have even been thought of with traditional design practices. As Besant and Lui (1986) rightly point out, in Computer Aided Design, man and machine work as a team where one complements the other. They identify the strengths and weaknesses of each of them in the following way:

> " The computer has three main functions:
>
> 1) To serve as an extension to the memory of the designer.
> 2) To enhance the analytical and logical power of the designer.
> 3) To relieve the designer from routine, repetitious tasks.
>
> The designer is left to perform the following activities:
>
> 1) Control of the design process in information distribution.
> 2) Application of creativity, ingenuity and experience.
> 3) Organisation of design information. "
>
> Besant and Lui (1986)

In the early stages computers were mainly used for intensive, number crunching tasks. However, work by Sutherland (1963) at M.I.T. on the development of the 'SKETCHPAD' interactive computer graphics system prompted the rapid development of computer technology into other areas of engineering. Initially, computer graphics concentrated on the development of techniques and software to facilitate the development of Engineering Drawings. Drafting packages such as AutoCAD (Autodesk) are implementations of this kind. Thus the computer was used, primarily, as a drafting tool. Further developments soon extended to using the computer as a Modelling Tool and an entirely new branch of study called 'Geometric Modelling' was born. The past few decades have witnessed the development of various types of modellers to address specific industrial needs. The combined

development of abundant computing power, display facilities, storage media, and input devices, together with evolutionary advances in 'Geometric Modelling' has resulted in a situation where the Computer System, constituting a partnership between hardware and software, has developed into a powerful tool, for the engineering industry. These advances in technology have now reached a state of transition; from regarding the computer as a tool for 'detailed' modelling and analysis, into a tool to assist design as a whole. Applications for this 'State of the Art' area of research include, computer-based Conceptual Design and *Design Reuse.*

Increasingly innovative applications must be envisaged to exploit this powerful tool, (Shah et al, 1996). ***This research is aimed at developing such an application, where the 'Computer System' is used in a novel way, facilitating the traditional engineering companies to computerise their operations with much less effort. This will enable them to reuse their past designs more efficiently, and develop next generation products built on their strengths through the latest developments in science and technology.***

1.2 The Need and Associated Problems

The majority of Small and Medium Enterprises (SME's), deal with the design and manufacture of a specific range of products, from individual piece-parts to complex multi-part assemblies. These enterprises typically archive a large collection of manufacturing drawings, for both discontinued and current 'in-service' products, which must be maintained and made accessible, when needed. The problem for these companies is to successfully adopt computerisation of this library of drawings, so that they can enjoy the resultant benefits of computer technology.

Thus the need here, is to establish an easy way of computerising these designs in a manner that will enable specific information from various design cases to be accessed at the press of a button.

The problems associated with meeting such a need are as follows:

1. Establishing a structure for the information that will be required at various levels of design abstraction

2. Establishing a methodology to efficiently store the structured information.
3. A mechanism to retrieve and use this information.

1.3 The Project

In this project, the structures of design information at different levels of abstraction were identified as:

a) Solution concept described as a Function Tree
b) Embodiment Design described as a Parts Tree
c) Detailed Design represented as a geometric, solid model

The principles, comprising a methodology for storing this information are as follows:

a) The Function Means Tree to store the solution concept and embodiment designs, and
b) A Variant Model and associated parameter database to store the detailed design.

A retrieval mechanism for the detailed design was developed in the form of a skeletal 'Master Model'. The 'Master Model' reads the parameters of a specified instance from the database to build the corresponding geometric model (or instance). This novel method eliminates the creation of one geometric model for each design, from scratch, and creates all instances (or geometric models) of a family from the same master model. This instance can then be modified or utilised for the next generation of products. In this way, the activity of computerisation is made much simpler, and information relating to past designs is made available to the designer at different levels of abstraction.

1.4 Structure of the Thesis

This research follows the design model outlined by Jones (1980), an adaptation of which is shown in Figure 1.1. In the first stage, *Divergence*, all the data related to the project in terms of design representations and geometric and solid modelling is collated. This enabled the understanding of the state of the art, and was analysed

critically to select the important attributes, characteristics and methods for integration to the proposed method, and is described in Chapter 2. In the next stage, *Transformation*, the elements identified as being important are developed and combined to form novel methods. The transformation process specifically looked at two possible methods for storing the detailed designs, the Generative and Variant methods, and two possible methods for storing conceptual and embodiment designs, the Chart-Based and Function-Means Tree methods. A combination of The Function Means Tree and Variant Model were selected as the novel method for development into a software system (*Convergence*). The methodology developed is described in Chapter 3. The software developed is given in Chapter 4. Three case studies, the Guindy Machine Tools Ltd. 'Lathe Chuck Family', the Lucas Varity Drive-End-Shield Casting and the Hydroflow Rotary Drum Filter are presented in Chapter 5. Chapter 6 presents the conclusions drawn from this work, and discusses the merits and demerits of the method and finally highlights the areas for further work.

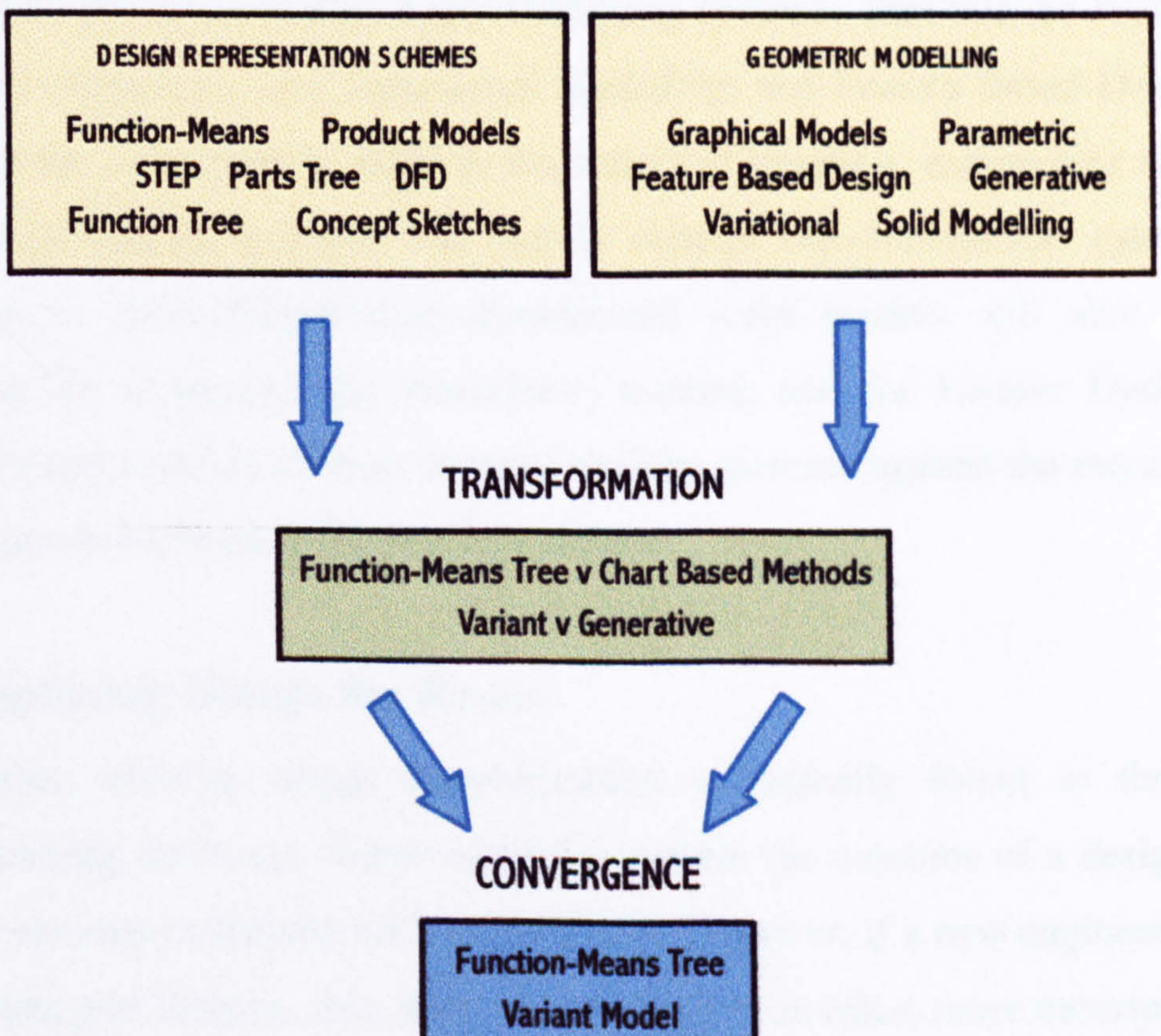


Figure 1.1 – The 3-Stage Design Model – Adapted from Jones (1980)

Chapter 2

Background & Theory

Overview

This chapter will discuss the theoretical background of Traditional and Computer Aided Design methods that are relevant to this research. It will begin with a general discussion of the *Design Process*, what elements of this process need to be *Captured* to enable effective *Design Reuse* and how this information can be structured and stored for efficient retrieval. Methods for structuring *Conceptual* and *Embodiment* design shall be discussed, including the *Function-Means Tree* and Design Function Deployment (DFD). The representation of *Detailed* design involves the study of Geometric, and in particular, Solid Modelling systems. This will be followed by a review of *Parametric and Variational Modelling*, and Feature Based Design - both of which are techniques to assist in the design of adaptive, engineering models. An analysis of existing methods that aim to convert two-dimensional manufacturing drawings to fully-fledged three-dimensional solid models will also be given, including the Generative (or Procedural) method, and the *Variant Method*. In all cases, the applicability of these theories shall be assessed against the requirements of this project as outlined in the previous chapter.

2.1 Capturing Design for Reuse

Traditional, existing design documentation is typically found in the form of manufacturing drawings. These structures contain the outcome of a design process, and are obvious candidates for Design Reuse. However, if a new engineer is to fully understand past designs, they will also need access to other, more descriptive, forms of design documentation, such as the initial design brief, ideas generated throughout the design, and lessons learnt by adopting a particular technique. This information requires the capture of information at various stages of the Design Process. Finger outlines the more specific needs to capture the design process.

" Explanation – to explain how and why a particular decision was made,
Verification – to determine if characteristics of the final design are consistent with the intended characteristics as represented by the top-level objectives,
Modification – to predict the effect of making changes to the design,
Reuse – to synthesise a design from a previous design with a similar specification and,
Instruction – to guide novice designers. " Finger (1998)

These needs require the identification of which stages of the design process are relevant to computerisation of past designs. These are discussed in the following section.

2.2 The Design Process

Much of design research has viewed the Design Process from a synthesis, or top-down approach. However, the emphasis in this research is from a bottom-up direction, as the goal of this project is to store a design for reuse, using the finished product (the manufacturing drawings) as a starting point. Shigley (1977) outlines the idealised, top-down design process as a chain of events (figure 2.1a) with iteration. For this research, the Recognition, Definition and Synthesis stages can be 'refined' into a more manageable series of events, as outlined in figure 2.1b, by Evbuomwan et al (1996).

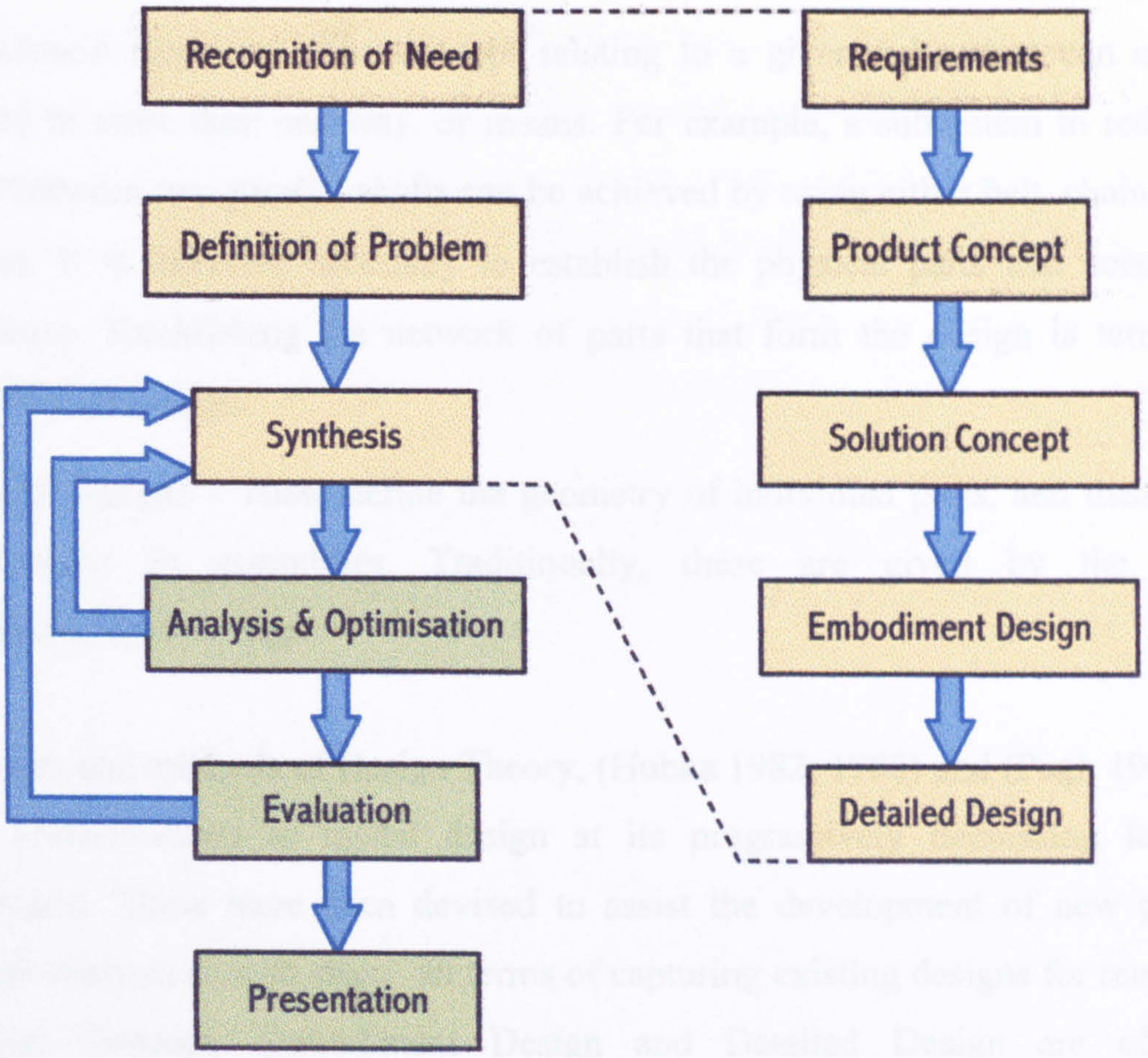


Figure 2.1 – The design process , (a) left: Shigley , (b) right Evbuomwan et al.

Requirements – The starting-point of the design and development of a product is its societal need. This need is represented by a set of prioritised requirements. Therefore, in this context, a Requirement can be defined as an element of a need.

Specifications - also termed *Product Concepts*, are a list of functions, that the design or artefact should perform to realise the mentioned requirements. These descriptions include the limitations imposed by factors such as geometry, space, working environment, legal and other considerations, which are collectively termed as the design *Constraints*. Specifications are generally, not solution specific, i.e. their content does not rely on a particular solution.

Solution Concepts – The list of functions to be performed as specified by the Product Concepts is broken into sub-groups, to which sub-solutions are proposed for their realisation. The combination of these sub-solutions, often termed Subsystems, form the design solution. Therefore, the Solution Concept may be defined as the combination of all conformable subsystems, which satisfy all listed functions and constraints in a holistic manner.

Embodiment Designs – The concept relating to a given subsystem can often be realised in more than one way, or means. For example, a subsystem to reduce the speed between two parallel shafts can be achieved by using either belt, chain or gear devices. It is therefore necessary to establish the physical parts that constitute a subsystem. Establishing the network of parts that form the design is termed the Embodiment Design.

Detailed Designs – These define the geometry of individual parts, and their spatial relationships in assemblies. Traditionally, these are given by the set of manufacturing drawings.

Strategies and methods of Design Theory, (Hubka 1982, 1988) and (Pugh 1991), use these classifications to model design at its progressively decreasing levels of abstraction. These have been devised to assist the development of new products through analysis at each stage. In terms of capturing existing designs for reuse, only Solution Concept, Embodiment Design and Detailed Design are of major significance. This is because the initial requirements specified at the beginning of a 'new' design process may differ somewhat to the functions the evolved design actually exhibits. Whether requirements are useful for design reuse or not, is somewhat trivialised by the fact that they are implicitly represented in the less abstract Solution Concepts, as functional requirements, (Malmqvist 1995). Similarly, the Product Concepts as outlined in the design process above, are of limited benefit to the less abstract representation of already formalised designs. Furthermore, the creativity and analysis activities of design are more heavily concentrated in the solution concept, embodiment and detailed stages of design, and therefore are more fruitful in terms of reuse.

From the preceding analysis it can be said that, for the task of capturing existing design cases for reuse, the following stages of the design process are of greatest significance:

a) Solution Concept,
b) Embodiment Design,
c) Detailed Design.

A substantial literature survey of design capture and reuse has shown that, to date, no commercial system to capture and reuse mechanical engineering designs, at all levels, has materialised. This subject is still the topic of much academic and (joint) industrial research, (Duffy 1998) and (Shah et al, 1996). This statement is especially true for the less well defined area of conceptual design, as the complete design process is not yet fully understood (Maher et al, 1995).

The following sections describe the prominent, existing techniques and theories developed to represent and capture information relating to the areas of Conceptual, Embodiment and Detailed design.

2.3 Conceptual Design

A Conceptual Design is the outcome from the process of developing solution concepts. It is the first stage of design where creativity and innovation are exercised, obeying engineering and scientific principles. A poor solution concept can never be improved by good embodiment and detailed designs. Therefore conceptual designs of existing products are a useful representation of successful designs, particularly for reuse. However, in real design situations, the conceptual design stage is rarely recorded. In this section, prominent methods for representing conceptual designs are reviewed. A concrete mixer design is used as an example in all cases.

Although this research does not focus on the principle of creating a new design from scratch, many of the theories relating to both conceptual and embodiment designs do. Moreover, in the majority of cases, information relating to the traditional conceptual phase of past designs would have been discarded, leaving only the detailed manufacturing drawings as a record of past designs. However, if a design is to be adequately reused, some functional description of what the product and its components do is necessary. Therefore these theories are reviewed in the following subsections, with an emphasis to structuring concepts for reuse.

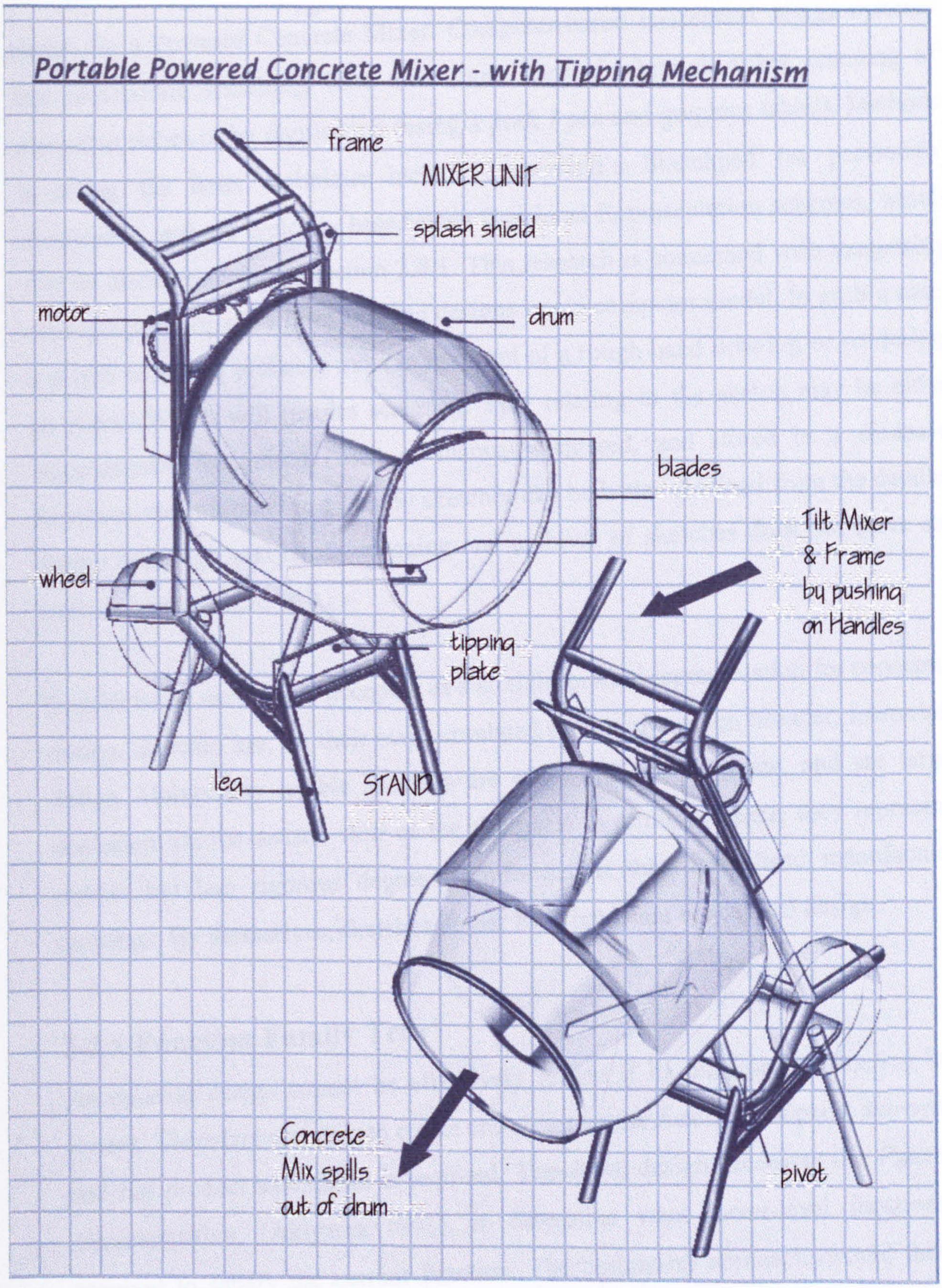


Figure 2.2 – A Conceptual Sketch for a Concrete Mixer

2.3.1 Sketching

The most obvious form of conceptual design is sketching (Cross 1991), which is both easily and universally understood. With suitable annotation, sketching is a leading candidate for recording design intent. Figure 2.2 shows an example of a

sketch for a Powered Concrete Mixer. Computer-based conceptual design systems that incorporate sketching, base their input methods through either scanning of manual sketches or by digitisation (using a puck / pen and graphics tablet). Methods involving the latter technique include Sutherland's Sketchpad (as previously mentioned), and are typically based upon Graphical Representation schemes, which will be discussed later, in section 2.8.1. This research is concerned with integrating conceptual information from existing designs into a computer model. In such a case, a sketch will most probably exist in the form of a rough hand drawing or rendering, on paper. Which will require scanning. Text relating to the sketch may be either automatically recognised (IEE) or manually entered, and stored in a database. However, the information given by sketches can be better obtained from the detailed design drawings, and hence scanning and archival of sketches does not serve any realistic purpose here.

Despite being universally accepted as a straightforward representation for conceptual design, sketches are, on their own, unrelated pieces of a much broader, interrelated design. Universally legible sketches are often difficult to create, and are largely dependent on the artistic skill of the designer. In terms of reuse, they represent a similar but less rigorous degree of information than formalised manufacturing drawings. By themselves, sketches do not fully represent conceptual design.

2.3.1 Function Family Tree

An existing design cannot be effectively reused if its purpose or Function is not known. Therefore a system to create and structure the functions of parts, sub-systems and full product assemblies is required. Top-down design processes use *Functional Decomposition*, (Akiyama 1991), to determine what lower-level functions are required to satisfy the current function. The bottom-up approach, would therefore *Compose* higher order functions from those prescribed by lower-level parts and sub-systems.

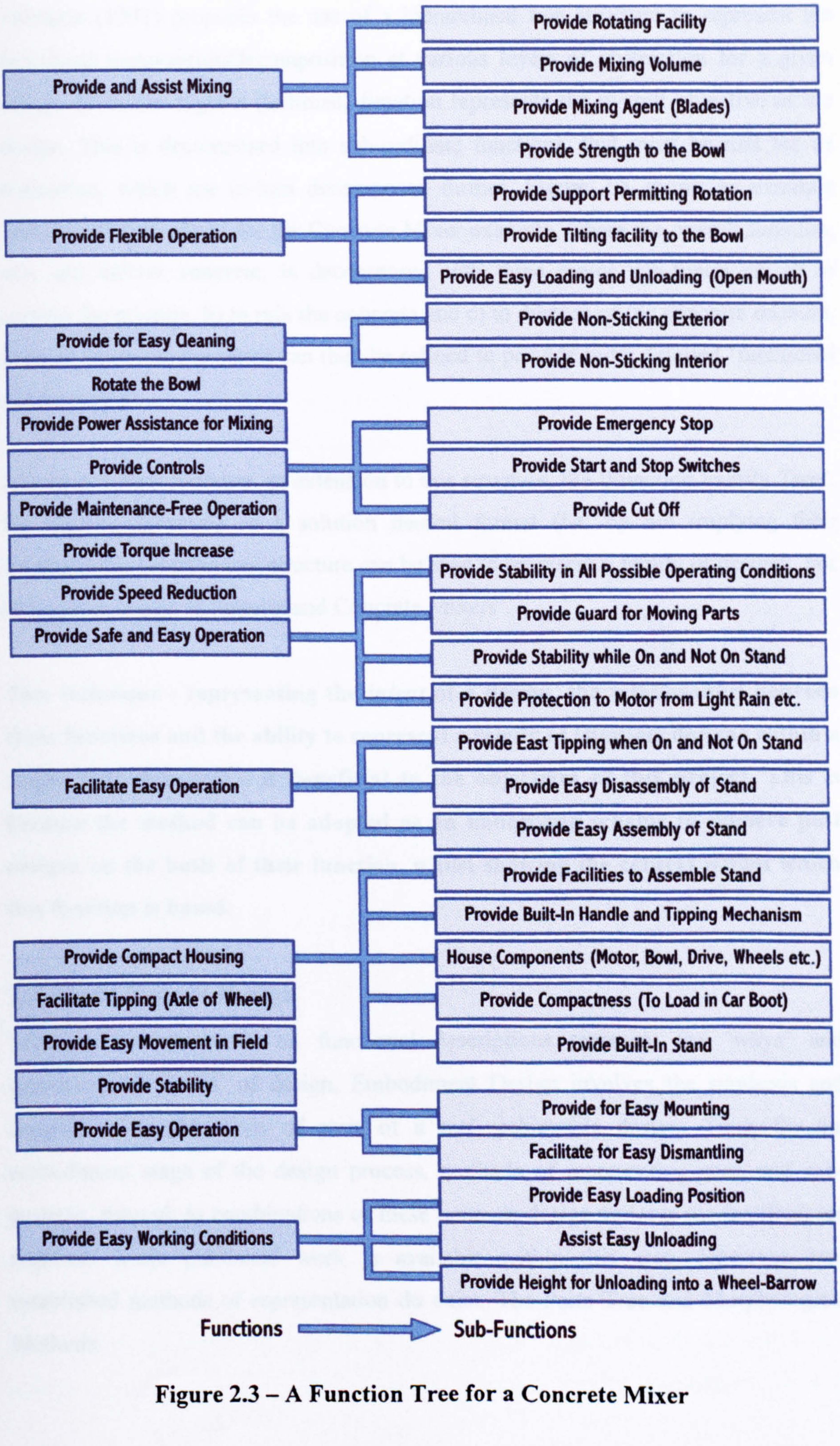


Figure 2.3 – A Function Tree for a Concrete Mixer

Akiyama (1991) proposes the use of a hierarchical tree structure to represent the functional composition/decomposition at various levels of abstraction for a given design. Here, the highest (leftmost) function represents the overall objective of the design. This is decomposed into sub-ordinate functions, that must be met for its realisation, which are in-turn decomposed further. Figure 2.3 shows an example 'Function Family Tree' for the Concrete Mixer example, where the overall function, mix and deliver concrete, is decomposed into three major sub-functions: a) to contain the mixture, b) to mix the concrete and c) to dispose of the concrete mixture. Each of these sub-functions can then be refined to provide more detailed 'functional requirements'.

Akiyama further proposes an extension to this structure, the 'Function Family Tree'. By keeping functions in a solution neutral format (i.e. by not implying their solution), the function tree structure can be seen to represent a family of designs. For example, a family of 'Mortar and Concrete Mixers'.

This technique - representing the *intent* of a design, the relationships between these functions and the ability to represent a family of (similar) designs within a single data structure - is beneficial to the objectives of this project. This is because the method can be adopted as an underlying scheme to retrieve past designs on the basis of their function, whilst showing the context within which this function is based.

2.4 Embodiment Design

Whereas systems based on functional descriptions represent the 'whys' and (partially) the 'hows' of design, Embodiment Design involves the synthesis and analysis of combinations of parts of a real, achievable design. Thus, for the embodiment stage of the design process, methods of representing parts and sub-systems, through to combinations of these parts, as design variants (or families) are required. Little published work is available within this area. However, two established methods of representation do exist: The Parts Tree and Morphological Methods.

2.4.1 Parts Tree

Pahl and Beitz (1988) identify the Parts Tree data structure as an ideal method of representing part and sub-system relationships, as a hierarchical tree. These relations are typically connectivity based, i.e. the hierarchical order in which parts and sub-systems are assembled. Such a scheme forms the natural representation of many commercial Assembly Modelling applications, and is also adept to kinematic analysis. Figure 2.4 shows a parts tree for a variant of the Concrete Mixer. By observation, it is evident that the highest node of the tree is the full product, and the leaves (the lowest nodes) relate to physical parts. Any node in-between these represents a sub-assembly, or sub-system. Therefore, the parts tree can be said to represent a design in terms of its manufacturing assembly layout.

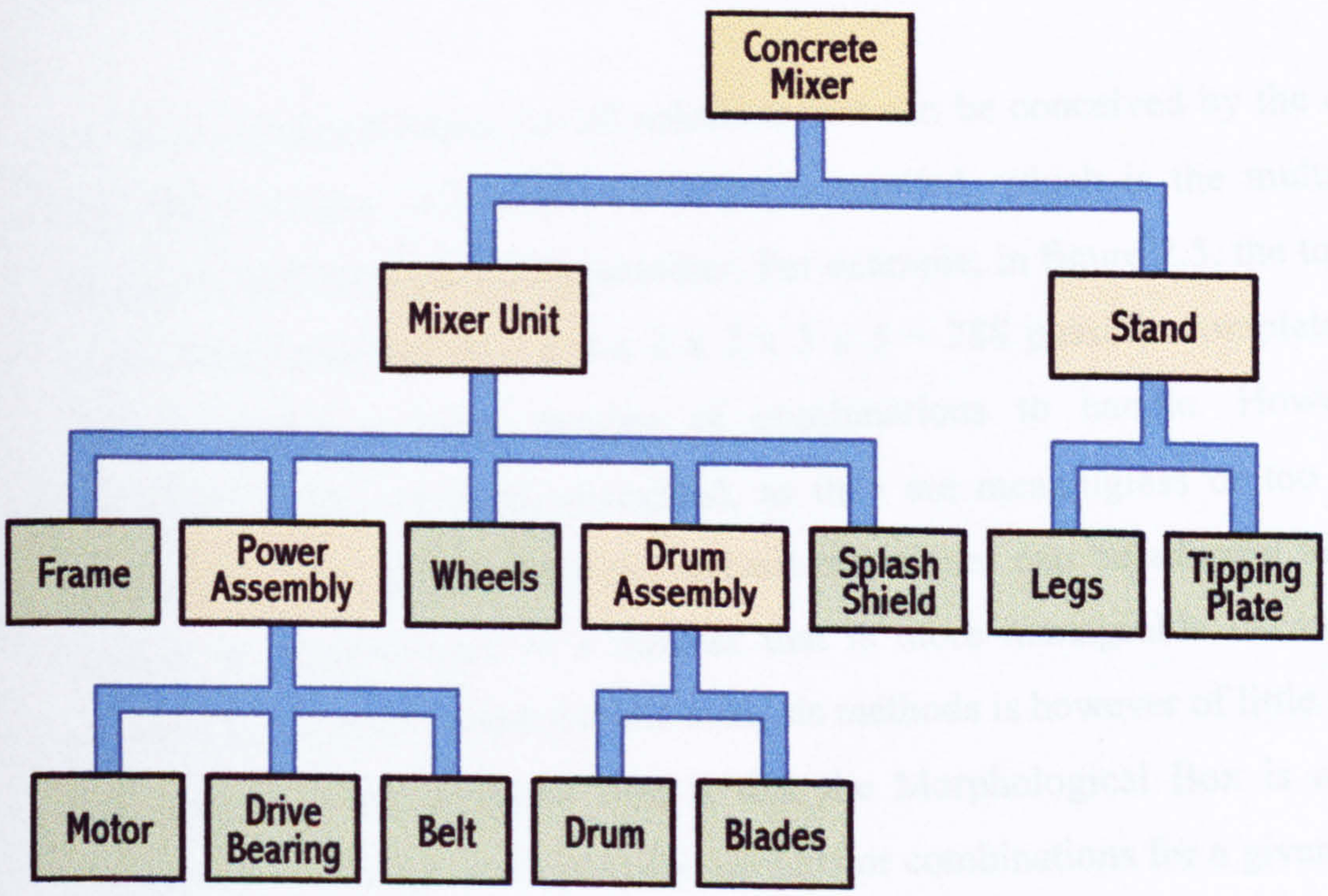


Figure 2.4 – A Parts Tree for a Concrete Mixer

2.4.2 Morphological Methods

Embodiment Design techniques involving morphological methods (as has been stated) are concerned with the synthesis and analysis of possible combinations of parts that can form a given design. As their name suggests, typical representations are pictorial and similar in appearance to conceptual sketches. Although this is not always the case, as some examples include purely textural representations, (Cross

1991). Two related morphological methods are prominent here, the Morphological Box (or Chart) and the Morphological Tree.

2.4.3 The Morphological Box

This method represents solutions for a given set of sub-functions as a two-dimensional array, (Grant 1977), and is also known as the Morphological Chart (Cross1991). In a Morphological Box, functions, known as the design 'parameters' each take-up a single row. The solutions (or variants) for each parameter sit in successive columns of their representative parameter. Thus the box is an unordered representation of all conceivable combinations of a design. Figure 2.5 shows an example morphological box for the Mortar and Concrete Mixer example, where a possible complete solution is given by the combination of the greyed-out boxes.

As this is a representation for all solutions that can be conceived by the designer, a very large number of possible solutions is implied, which is the multiple of the number of solutions for each parameter. For example, in figure 2.5, the total number of complete solutions is: 2 x 4 x 2 x 2 x 3 x 3 = 288 possible complete solutions. This is clearly a large number of combinations to handle. However, some combinations can be easily discarded, as they are meaningless or too difficult to implement. Also, Morphological Analysis techniques can be adopted to reduce the number of combinations to a number that is more manageable. A more detailed explanation of these morphological analysis methods is however of little relevance to this research. The emphasis here is that the Morphological Box is a useful and simple representation of all possible variants or combinations for a given design. For existing designs, the number of solutions per parameter will be much smaller. Therefore, this method allows the designer to 'pick and choose' elements from a database of existing components to synthesise a new design.

This method can be effectively used to design the next generation of an existing product.

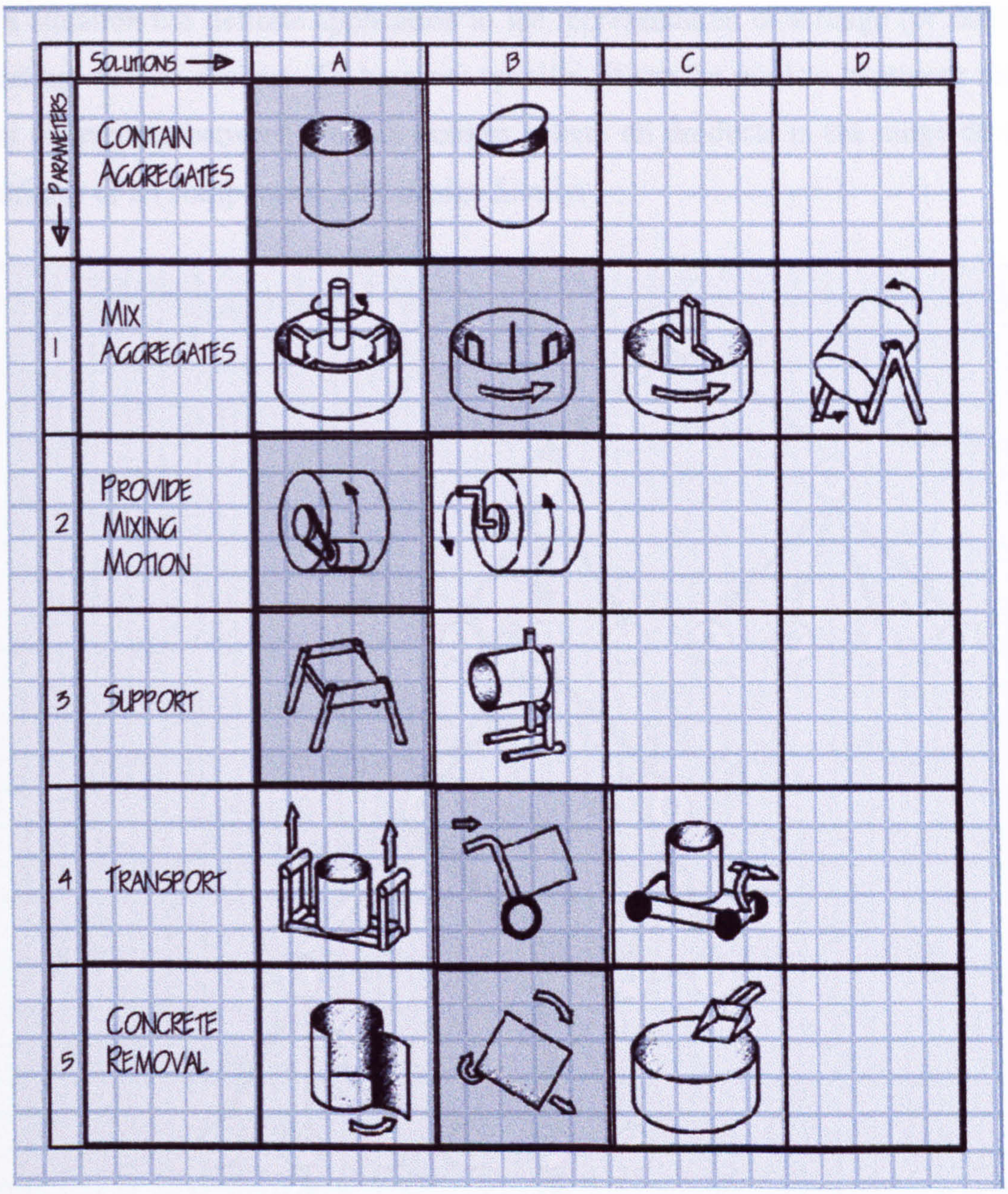


Figure 2.5 – A Morphological Box for the Mortar and Concrete Mixer

2.4.4 The Morphological Tree

In the previous section, the morphological box was shown to be an unordered representation of possible embodiments, or solutions. The contents of this box can also be represented as a tree structure, to directly show the possible combinations of solutions, and is termed the Morphological Tree, or the Decision / Alternatives Tree. (Grant 1977).

In this case, each level of the tree corresponds to a parameter, or row, in the morphological box. To begin with, each node for a given level represents the solutions for that level. However, for a given node, branches in the next level correspond only to compatible solutions, as demonstrated in figure 2.6.

Such a situation has definite application to the representation of a range (or family) of existing, similar designs. To be more specific, it can be used to represent more radical differences between product designs, where all products in the range do not use variants of all components, of a design family.

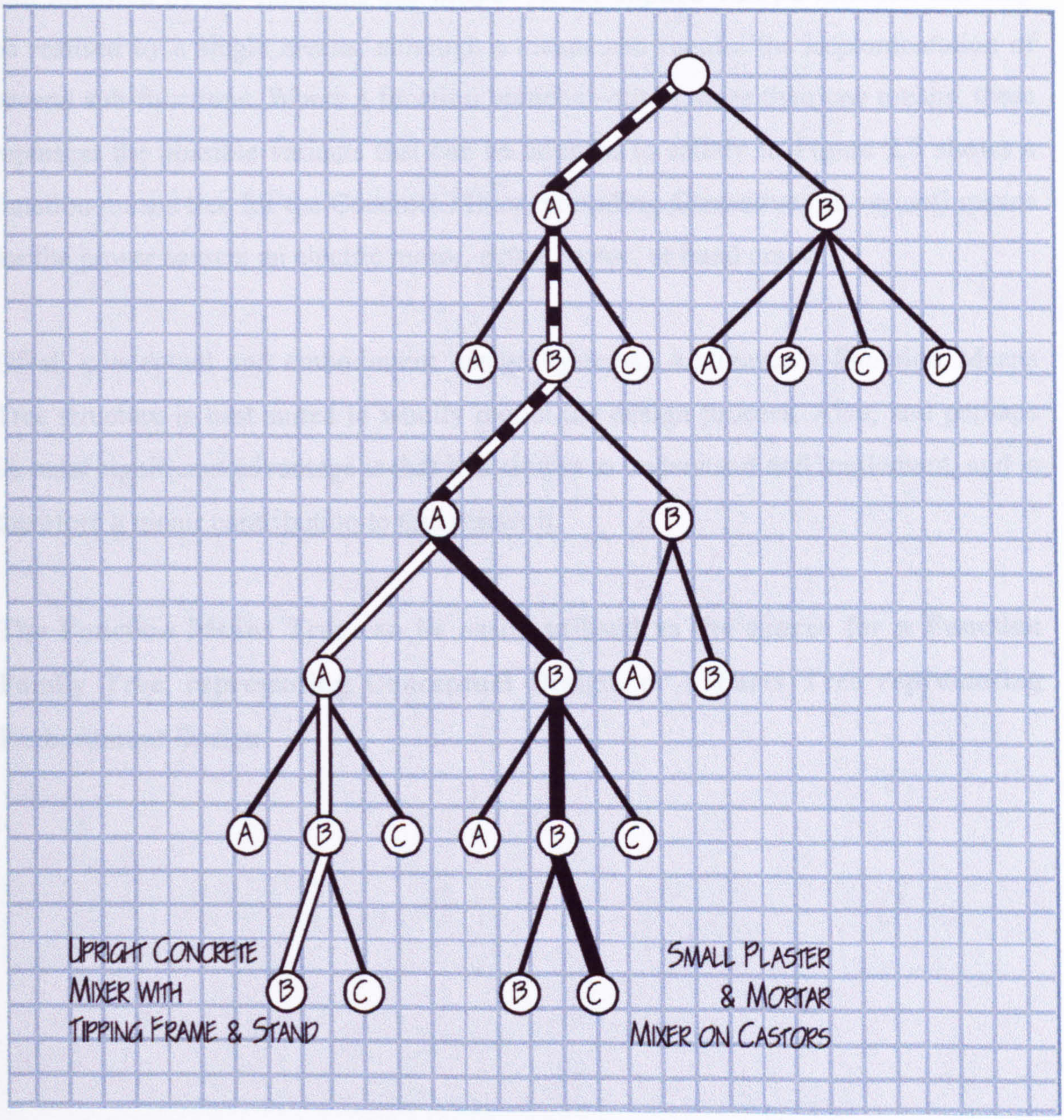


Figure 2.6 – A Morphological Tree for the Mortar and Concrete Mixer

2.5 Function Means Tree

The Morphological Box and the Morphological Tree are established tools for representing both conceptual and embodiment design under a single data structure. However, they are sometimes implied, and not explicitly defined within these structures. Conversely, the Function Means Tree (Andreasen 1980) is a definitive

relationship between the function (or concept) and its satisfying means (or embodiment). It is essentially a combination of both the function tree and the parts tree, although it is structurally representative of the former, being a tool to aid design synthesis. Here, an overall function is fulfilled by its realising means, which is in-turn followed by sub-functions and means. As an implied 'rule', a function can only be realised by a single means, although a means can require the implementation of several sub-functions. Where a function branches-out to more than one means, these represent the possible variants that can be adopted to satisfy it. Figure 2.7 shows a function means tree for the Concrete Mixer, providing alternative (or variant) means for the power source: an electric motor, petrol motor, or hand crank.

Of all conceptual and embodiment design systems, Andreasen's Function Means Tree structure is best suited to wholly model the design process. Also, and perhaps its most significant advantage is that it is simple to understand and implement, and is therefore a major contribution to this research.

The Function Means Tree can be easily utilised as the source for a Function Family Tree, representing Conceptual Design, or a Parts Tree representing Embodiment Design.

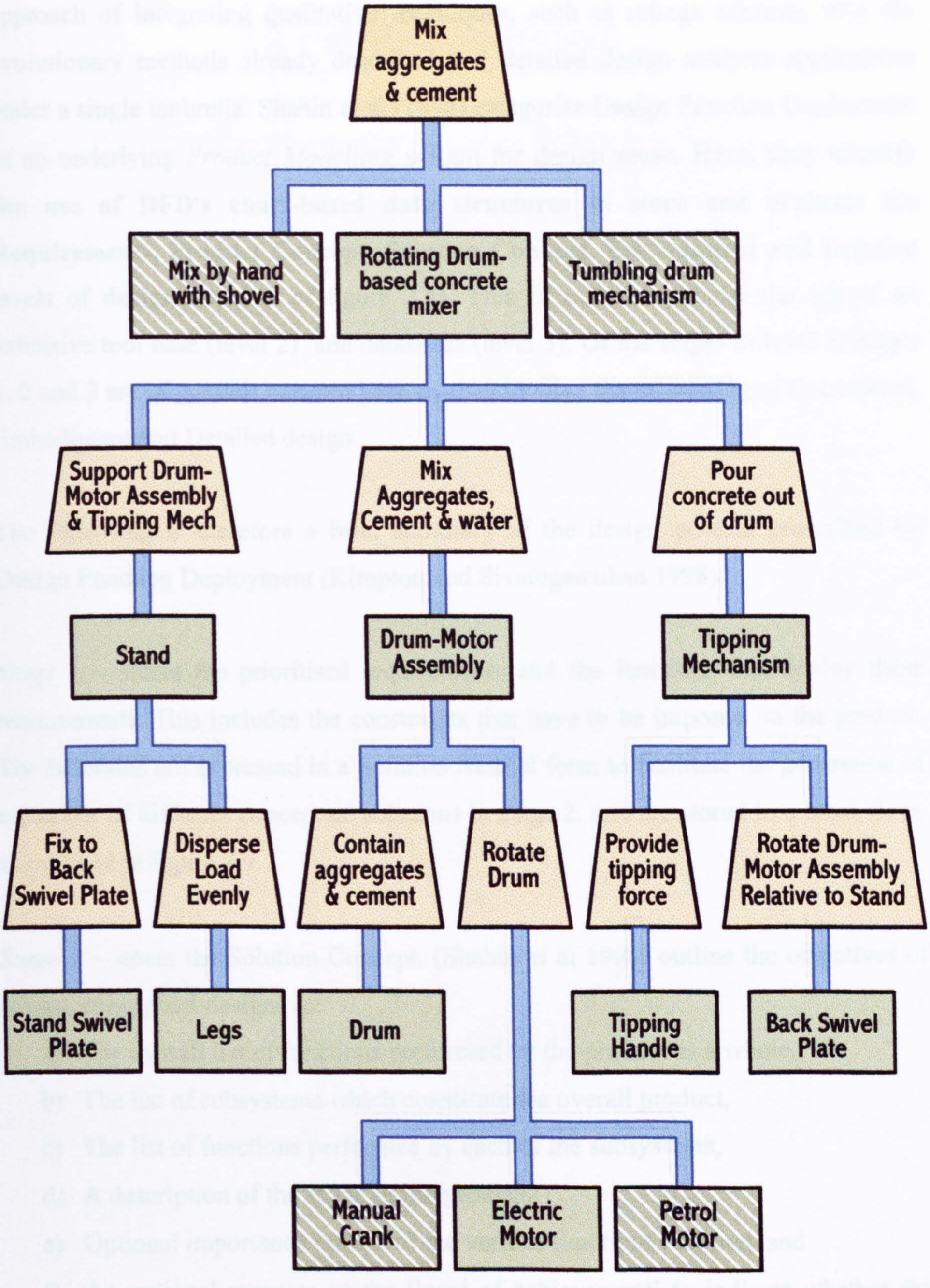


Figure 2.7 – A Function Means Tree for the Concrete Mixer

2.6 Design Function Deployment (DFD)

The design methods outlined so far, do little to provide the designer with a system containing the required tools to quantitatively analyse various stages of the design process. Design Function Deployment, (Sivaloganathan et al 1995), takes the

approach of integrating qualitative techniques, such as ratings schemes with the evolutionary methods already described and detailed design analysis applications under a single umbrella. Shahin et al (1998) categorise Design Function Deployment as an underlying *Product Modelling* system for design reuse. **Here, they identify the use of DFD's chart-based data structures to store and evaluate the Requirements, Product Concept, Solution Concept, Embodiment and Detailed levels of design** (level 1 of figure 2.8). This is achieved though the use of an extensive tool base (level 2) and databases (level 3). Of the stages in level 1, stages 1, 2 and 3 are of greatest concern here, as they involve the processing of Conceptual, Embodiment and Detailed design.

The following is therefore a brief summary of the design process prescribed by Design Function Deployment (Kimpton and Sivaloganathan 1998):

Stage 1 – stores the prioritised requirements and the functions that deploy these requirements. This includes the constraints that have to be imposed on the product. The functions are expressed in a Solution Neutral form to facilitate the generation of a number of different conceptual solutions in stage 2, and are stored in a chart form, as outlined in figure 2.9.

Stage 2 – stores the Solution Concept. (Shahin et al 1998) outline the objectives of storing conceptual designs as:

a) The overall list of functions performed by the product as a whole,
b) The list of subsystems which constitute the overall product,
c) The list of functions performed by each of the subsystems,
d) A description of the shape of the product,
e) Optional importance ratings of the various functions required, and
f) An optional measure of the 'level of achievement' to indicate whether the function is provided well by the concept or not.

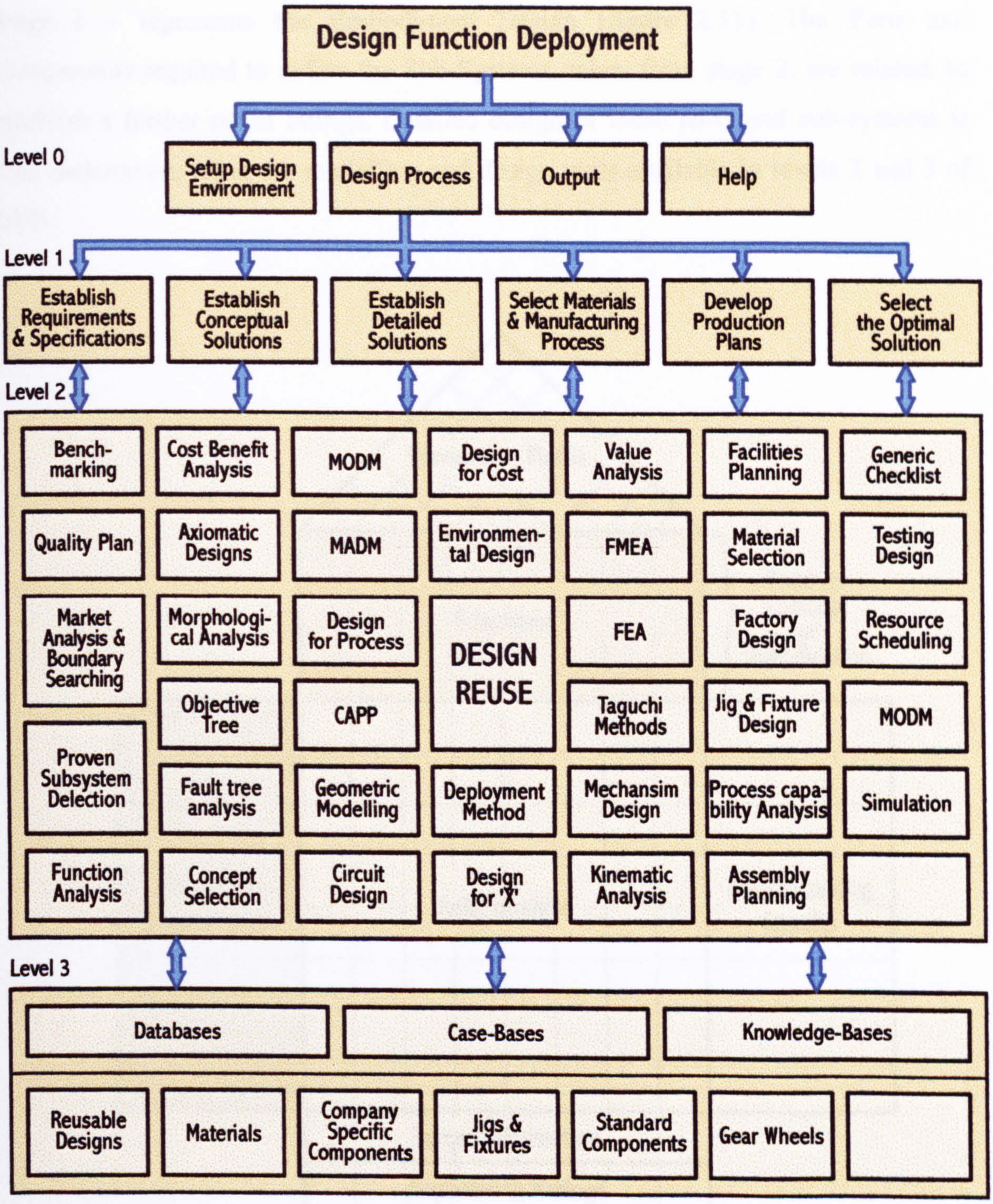


Figure 2.8 – Structural Overview of Design Function Deployment

A chart similar to that in figure 2.10 is used to store the conceptual solution, and each solution is stored in a separate chart. This chart relates the Functions of stage 1, along with their importance ratings, to sub-systems, that have been determined using the design methods outlined in level 2 of DFD, e.g. the morphological box. The result of this relation is another set of importance ratings per architecture (or conceptual design).

Stage 3 – represents the Embodiment Design (figure 2.11). The Parts and Components required to define the Sub-Systems, taken from stage 2, are related, to establish a further set of ratings. Detailed design of these parts and sub-systems is then undertaken, using the modelling and design tools available in levels 2 and 3 of DFD.

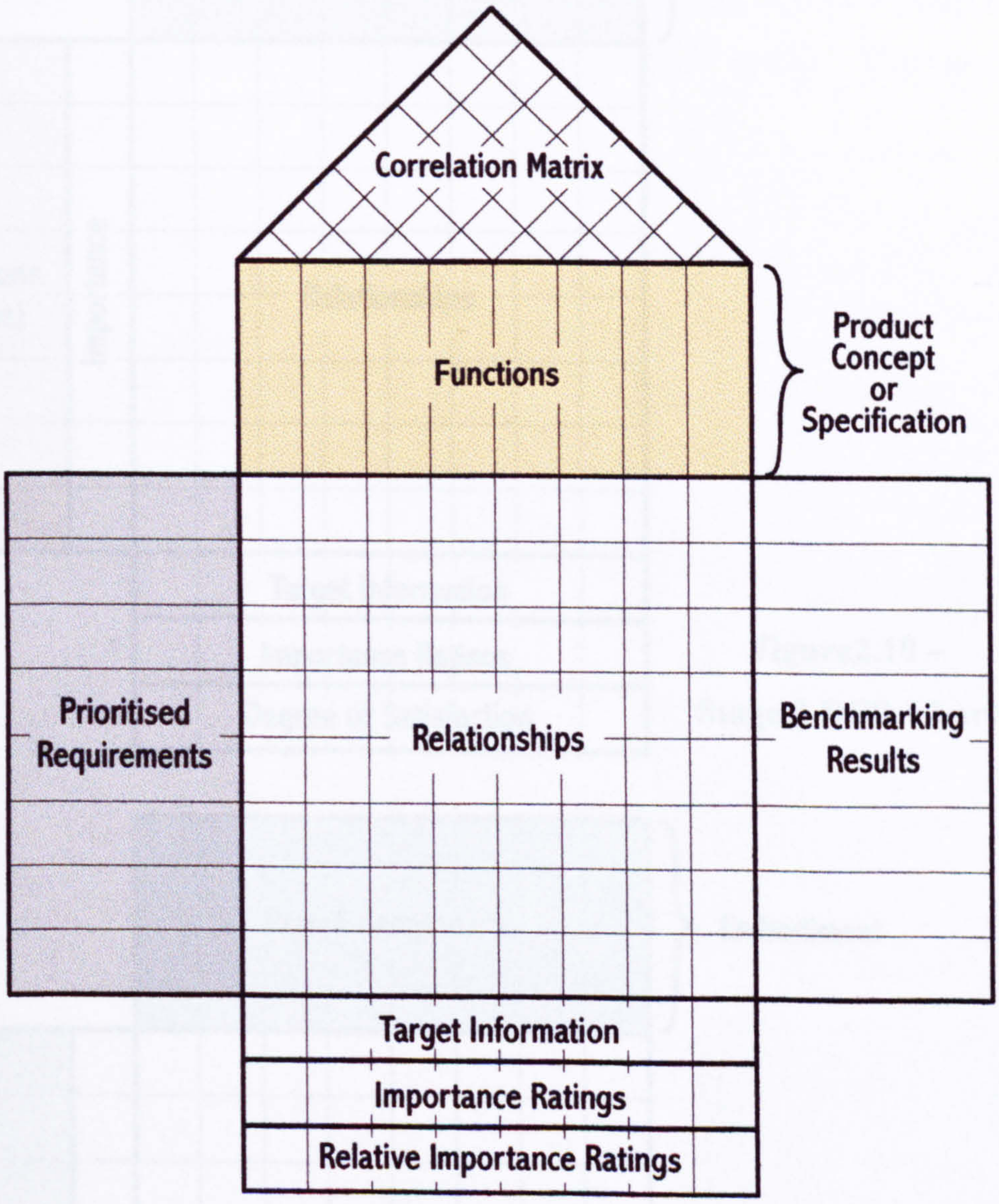


Figure2.9 – Stage 1 DFD Chart

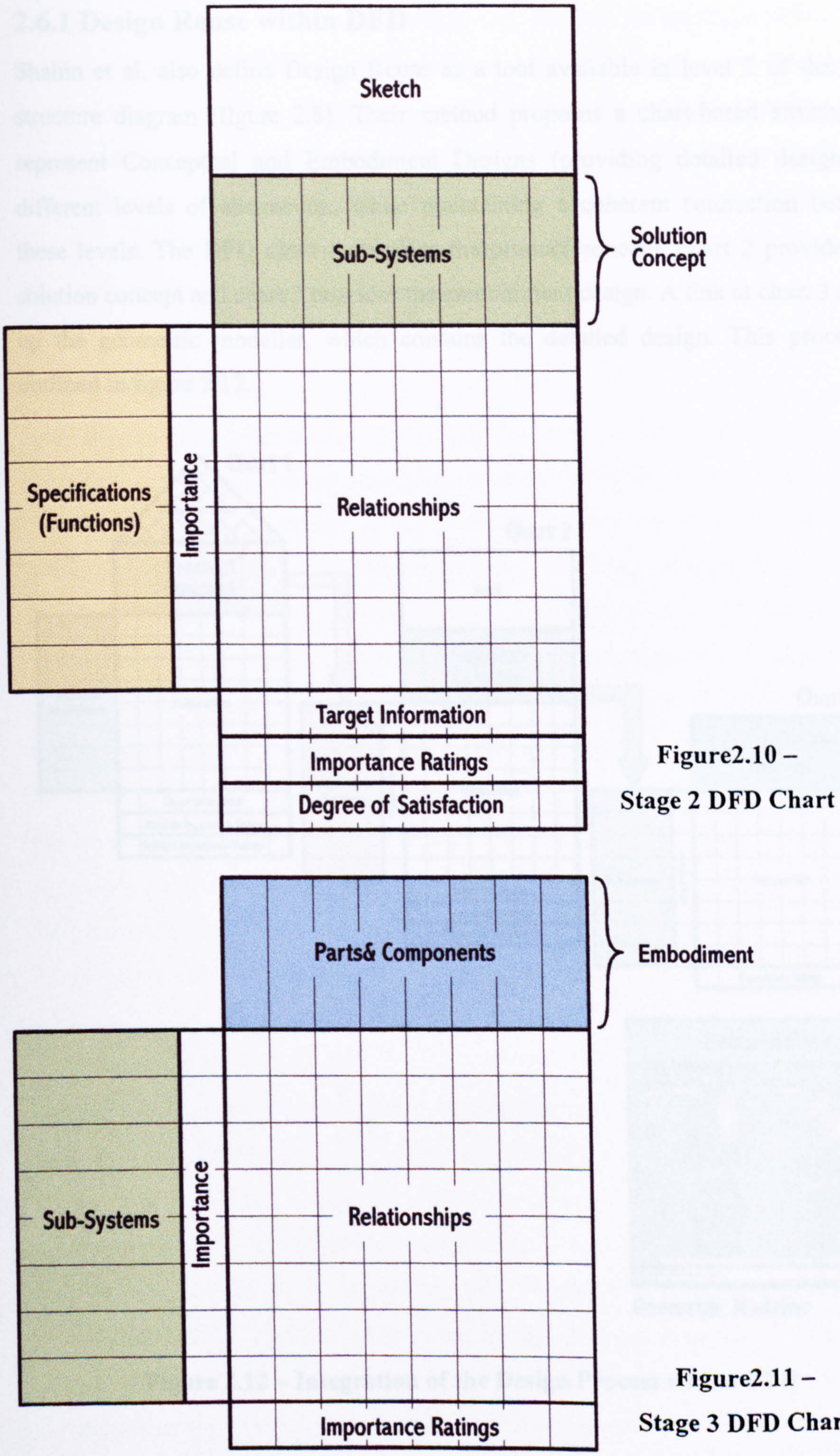


Figure2.10 – Stage 2 DFD Chart

Figure2.11 – Stage 3 DFD Chart

2.6.1 Design Reuse within DFD

Shahin et al. also define Design Reuse as a tool available in level 2 of the DFD structure diagram (figure 2.8). Their method proposes a chart-based structure to represent Conceptual and Embodiment Designs (providing detailed designs) at different levels of abstraction, while maintaining a coherent connection between these levels. The DFD chart 1 provides the product concept, chart 2 provides the solution concept and chart 3 provides the embodiment design. A link at chart 3 opens up the geometric modeller, which contains the detailed design. This process is outlined in figure 2.12.

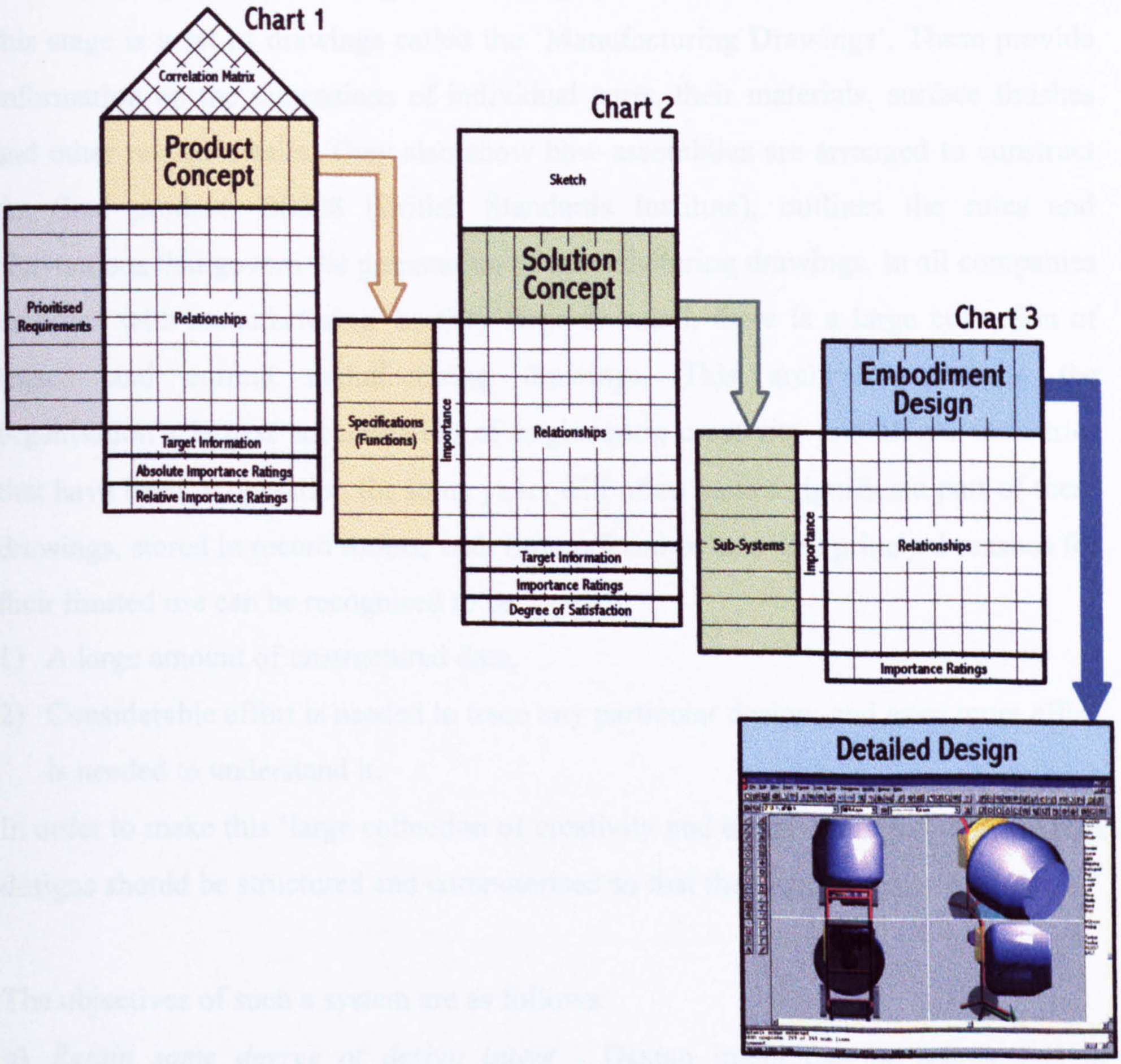


Figure 2.12 – Integration of the Design Process within DFD

The information contained in these charts is very detailed, for all stages of the design process. **The DFD method provides a system to cope with every eventuality, but is therefore, somewhat cumbersome and difficult to use. As a result, many organisations, especially small and medium sized companies, may view DFD as a complicated means of re-defining what is already known, which is true for the case of many well understood, existing designs.** Hence, DFD is not wholly suitable for the condition of simplifying the modelling of past designs.

2.7 Detailed Design

Detailed design is the final stage of the design process. Traditionally, the outcome of this stage is a set of drawings called the 'Manufacturing Drawings'. These provide information on the dimensions of individual parts, their materials, surface finishes and other related details. They also show how assemblies are arranged to construct the final product. BS308 (British Standards Institute), outlines the rules and conventions that govern the preparation of manufacturing drawings. In all companies involved with manufacturing, be they large or small, there is a large collection of legacy and current manufacturing drawings. This archive represents the organisation's largest accumulation of engineering creativity and effort. Industries that have been in operation for some years will often have a significant part of these drawings, stored in record rooms, with little referral or use. The principal reasons for their limited use can be recognised to be:

1) A large amount of unstructured data,
2) Considerable effort is needed to trace any particular design, and even more effort is needed to understand it.

In order to make this 'large collection of creativity and effort' more exploitable, past designs should be structured and computerised so that they can be easily reused.

The objectives of such a system are as follows:

a) *Retain some degree of design intent* - Design intent can be represented in manufacturing drawings either directly by textural (annotated) descriptions, attached by labels to various elements of a drawing, or indirectly through particular dimensions that are characteristic of the design. Figure 2.13 illustrates examples of this. The 'Through Hole' (left) and 'Square Thread' (right), of a

pipe-bending design, show how essential characteristics of a design are represented on manufacturing drawings. Initial techniques for modelling detailed design ignored these characteristics. **Hence the use of labelled text and the engineering significance of an 'entity' should be preserved.**

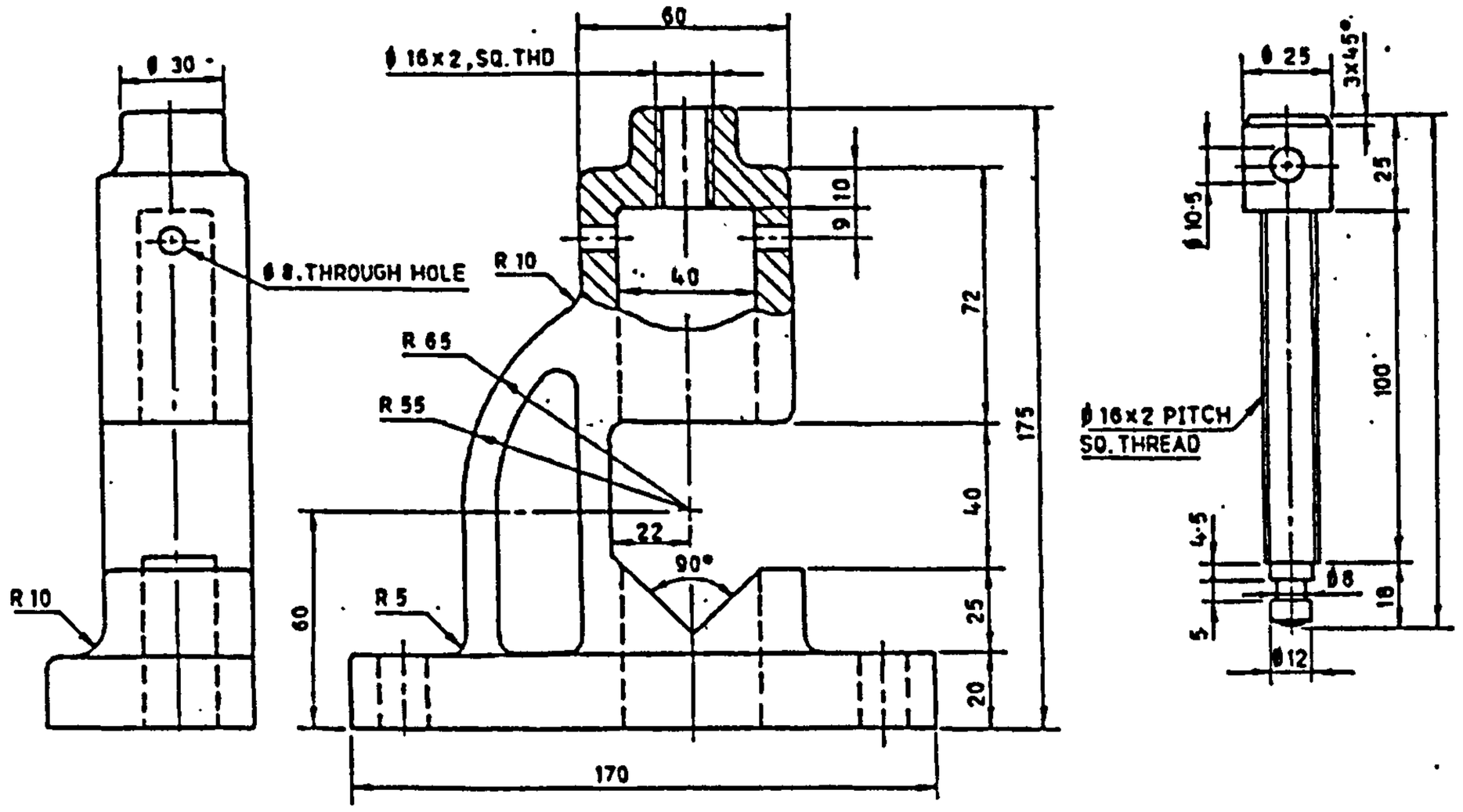


Figure2.13 – Representing Design Intent in a Manufacturing Drawing of a Pipe-Bender: (left) Main Body and (right) Screw Shaft

b) *Similar parts and products should be grouped into families* - Identifying and grouping similarities between designs has advantages in both design and manufacturing. As well as cataloguing benefits, duplication can be minimised, thereby reducing the effort required.

c) *Designs should be easily adaptable* - Paper-based drawings are static, that is, they cannot be easily modified when a (sometimes minor) change is required. The adoption of computers in design was an attempt to overcome this deficiency. However, the degree to which computer generated models can be adapted varies widely.

d) *The models should be useful for future developments* - The emphasis behind this research is to allow companies to computerise their designs, with minimum effort, so that they can use the latest computer technology to improve and generate new designs. Therefore, the design representation should reflect this desire, i.e. the design model should be in a format that can be easily used, or

transformed, for downstream applications, for example Finite Element Analysis or CNC manufacturing.

In order to achieve these objectives, the following two important constituents are necessary:

1) **Models to store the detailed design, and**
2) **An easy method of converting the paper-based drawings into these models.**

Section 2.8 represents a survey on Geometric Modelling systems, which is followed by the successive developments of Parametric and Variational Modelling (section 2.9) and Feature Based design (section 2.10). Section 2.11 surveys the methods for converting paper drawings into computer models.

2.8 Geometric Modelling

Geometric Modelling can be defined as a branch of study which 'brings together and applies analytic geometry, vector calculus, topology, set theory, and an arsenal of computation methods to model geometric entities'

Mortenson (1985)

It essentially deals with the modelling of the following four constituent, geometric entities of an object:

a) Vertices
b) Edges
c) Surfaces
d) Solids

Mathematical theories and techniques have been developed to represent each of these entities. The fundamental objective of their development is to have a representation scheme, which can be used to represent all members or varieties within a class (e.g. straight, circular and other *edges*), and their manipulations (e.g. extension, truncation etc.). Homogenous co-ordinates have been developed to store points, or vertices. Parametric representations of curves were developed to represent

both simple and composite space curves and surfaces, several representative techniques to model *solid* objects have also been developed.

The fundamental objective of Solid Modelling is to provide a complete representation of a solid object. Requicha defined solid modelling as:

'an emerging body of theory, techniques and systems focused on informally complete representations of solids – representations that permit (at least in principle) any well defined property of any represented object to be calculated automatically.'

Requicha (1980)

A solid modelling system can be defined as being the combination of a modelling engine and a set of algorithms, which can answer geometric questions by scanning the geometric model. This definition is schematically represented in figure 2.14.

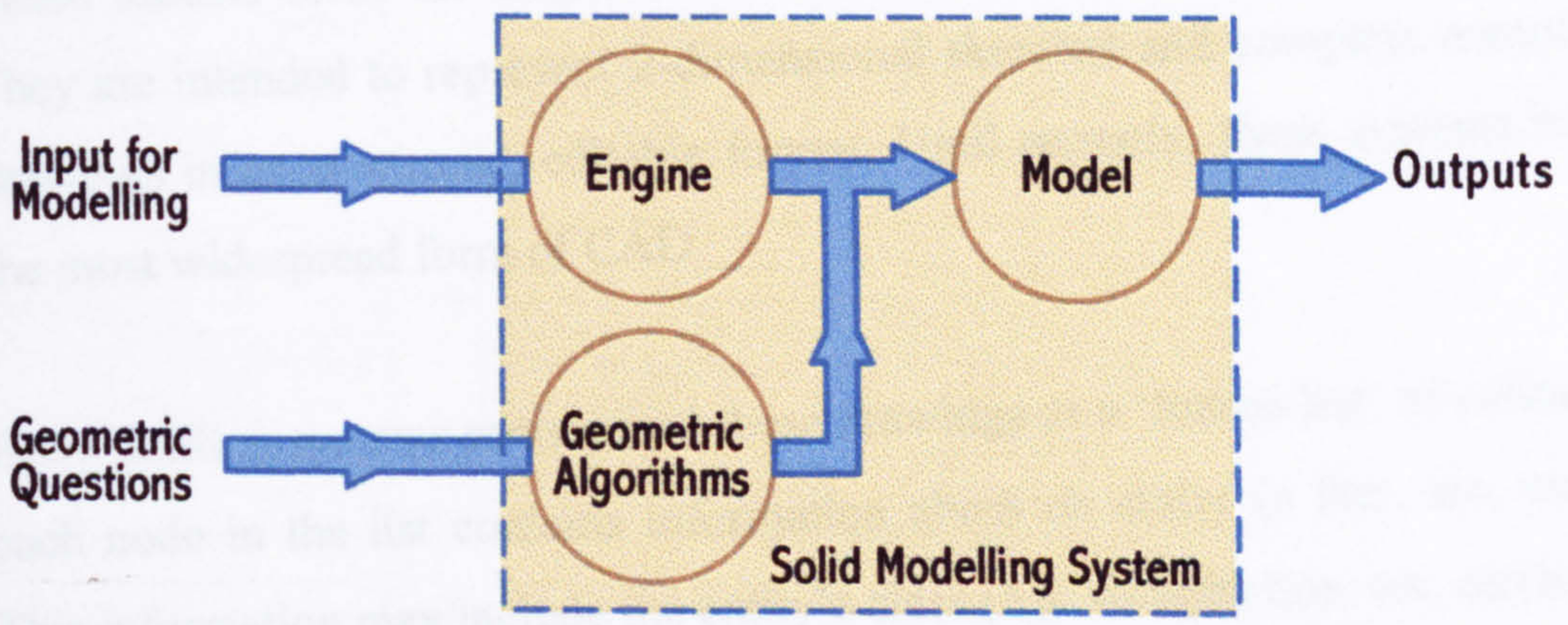


Figure 2.14 – A Solid Modelling System

The effectiveness of this model and modelling system depends upon the number of algorithms that are available within the system to answer geometric questions. This concept is a key issue in selecting the most suitable representations to store part designs.

The development of CAD systems has been incremental, and the motivation for this has stemmed from different industrial needs. The first application that saw the development of what are now termed Graphical Systems, used the computer as a drafting tool. This was followed by attempts to use the computer as a sophisticated modelling tool. This led to several such models that were developed to cater for

varying industrial requirements. In general, these models fall into the following categories:

1) Graphical Models - to aid the generation of manufacturing 2D drawings.
2) Shape Models - to represent raster (scanned) images for image processing.
3) Surface Models - to create complex curves and surfaces.
4) Solid Models - to capture complete representations of 3D geometry

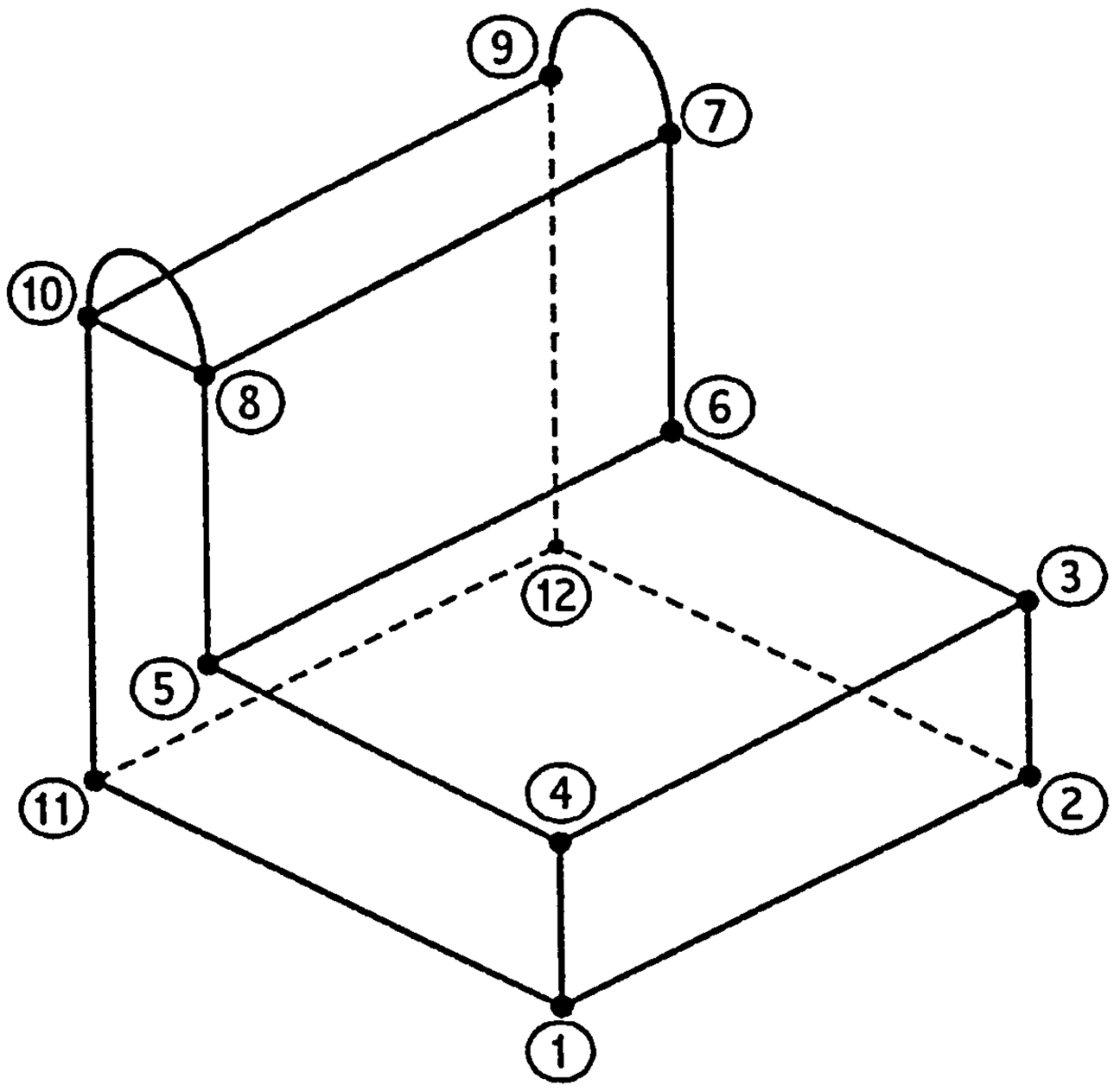
Of these categories, Graphical and Solid Modelling techniques are of particular interest to this research, since paper-based drawings are akin to graphical models, and solid models maintain a complete representation of the object.

2.8.1 Graphical Models

These models form the original definition of CAD, Computer Assisted Drafting. They are intended to represent 2-dimensional sketches and complete manufacturing drawings in an electronic, editable format. Until recently, these systems have been the most widespread form of CAD.

Early drafting systems represented these drawings as a 'linked-list' of entities, where each node in the list contains information about an entity (a line, arc, circle etc.). This information may include the entity's class (e.g. straight-line, arc, circle etc.), the line-type (continuous, dashed etc.), geometry (e.g. start-point, end-point co-ordinates) and connectivity etc,. A linked-list representation for a general geometric object is shown in figure 2.15.

As well as enabling the use of standard *primitive* types, e.g. lines, circles and arcs, a number of graphical systems have invoked the use of *associative graphical primitives*, enabling a Parametric form of drafting to be adopted. Parametric design (or in this case drafting) is a process where parameters (typically geometric dimensions) relating to elements of the design, can be modified. For example, the radius of a circle can be changed from 10mm to 5mm. This is not the same as deleting the 10mm radius circle and creating a new 5mm circle. Both Parametric and Variational Design techniques shall be discussed, in more depth, in section 2.9.



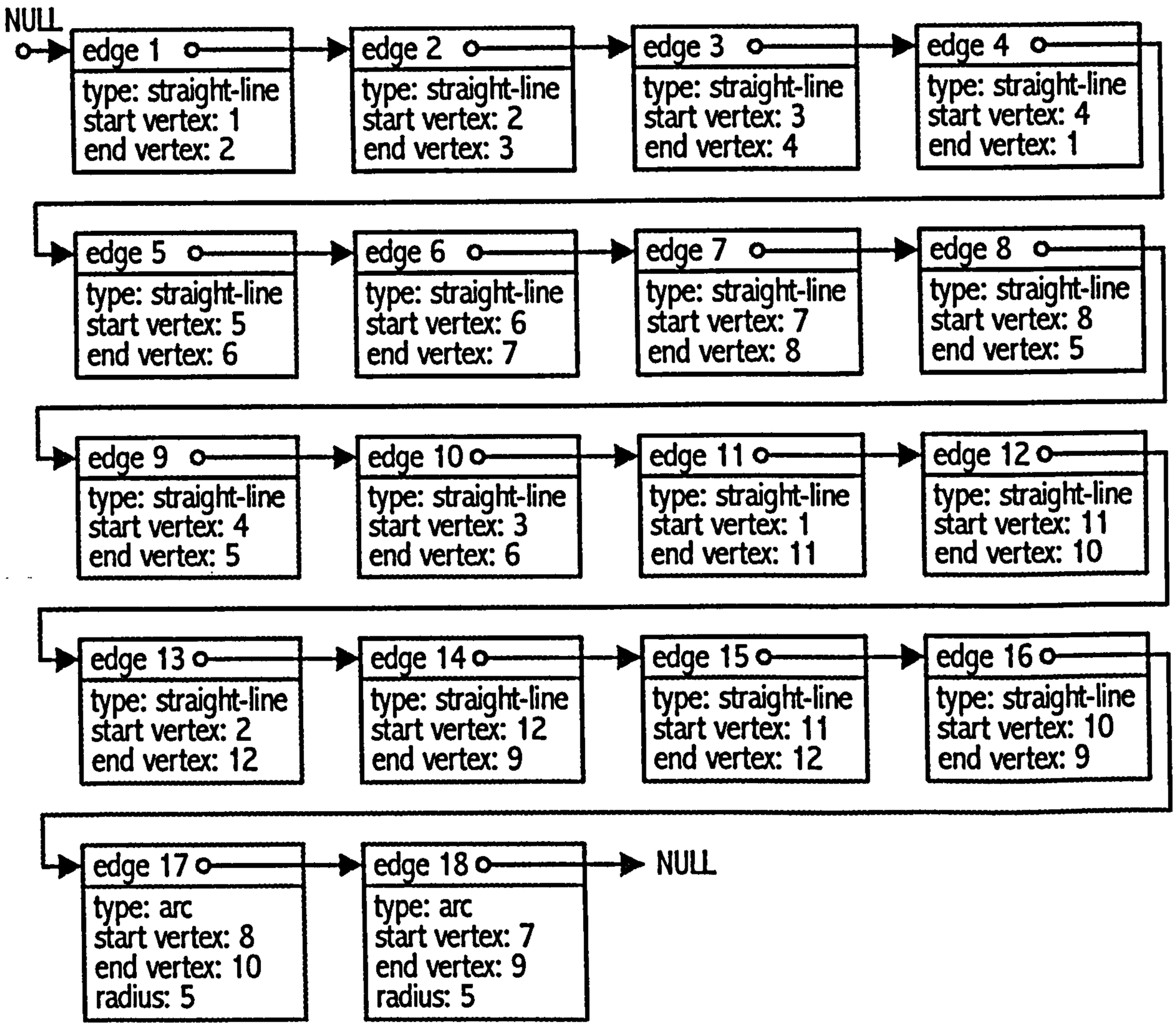


Figure 2.15 – Linked-List Representation of a Graphical Model

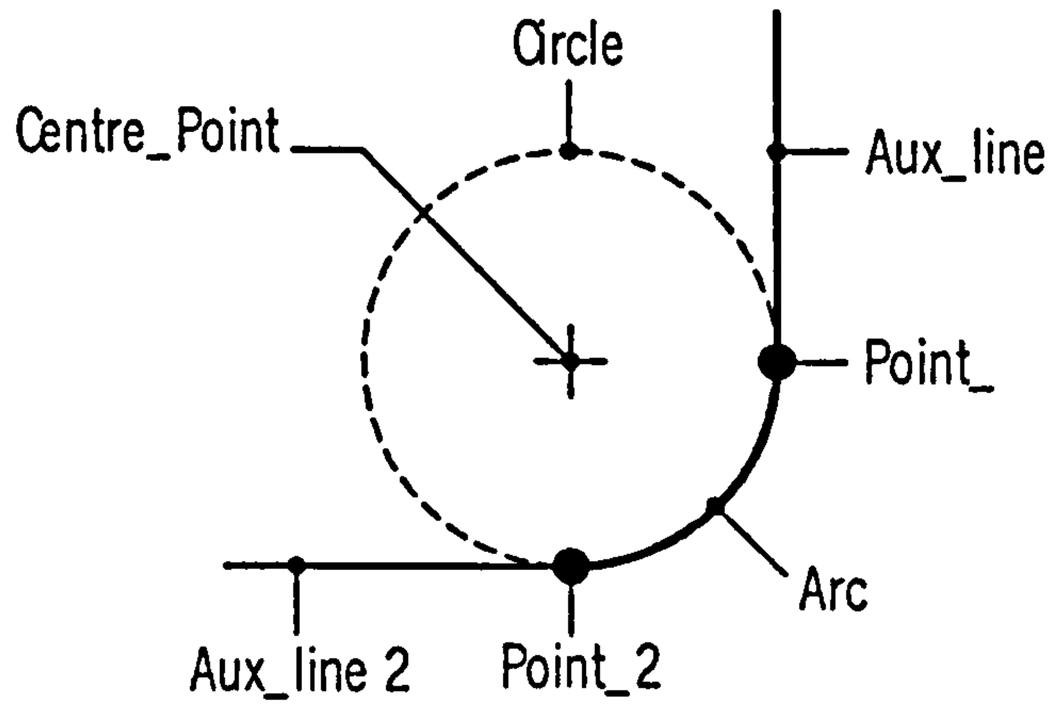


Figure 2.16 - A Fillet and its Associative Primitives (Shah 1995)

For the majority of these systems, primitives are represented internally using *Associative Representation*, where the construction process used to create the primitive is stored. For example, when constructing the fillet (a circular Arc) of figure 2.16, a further, associated primitive (a Circle) is required. Both of these primitives may be represented as:

For the Arc:

```
Construction technique: fillet_arc_between_straight_line_segments
      Point_1: intersection (Circle, Aux_line_1)
      Point_2: intersection (Circle, Aux_line_2)
```

For the Circle:

```
Construction technique: circle_touching_two_line_segments
      Radius: given_by_the_user
      Centre_point: (some computation involving the two lines)
```

If the user of this system decides to say, change the Radius of the fillet, they can simply modify the Radius parameter, and re-execute the construction history representation. Unlike non-parametric situations, where the fillet-arc would have been deleted and replaced with a primitive of a different radius, associative information (i.e. to lines 1 and 2, and the circle) is maintained.

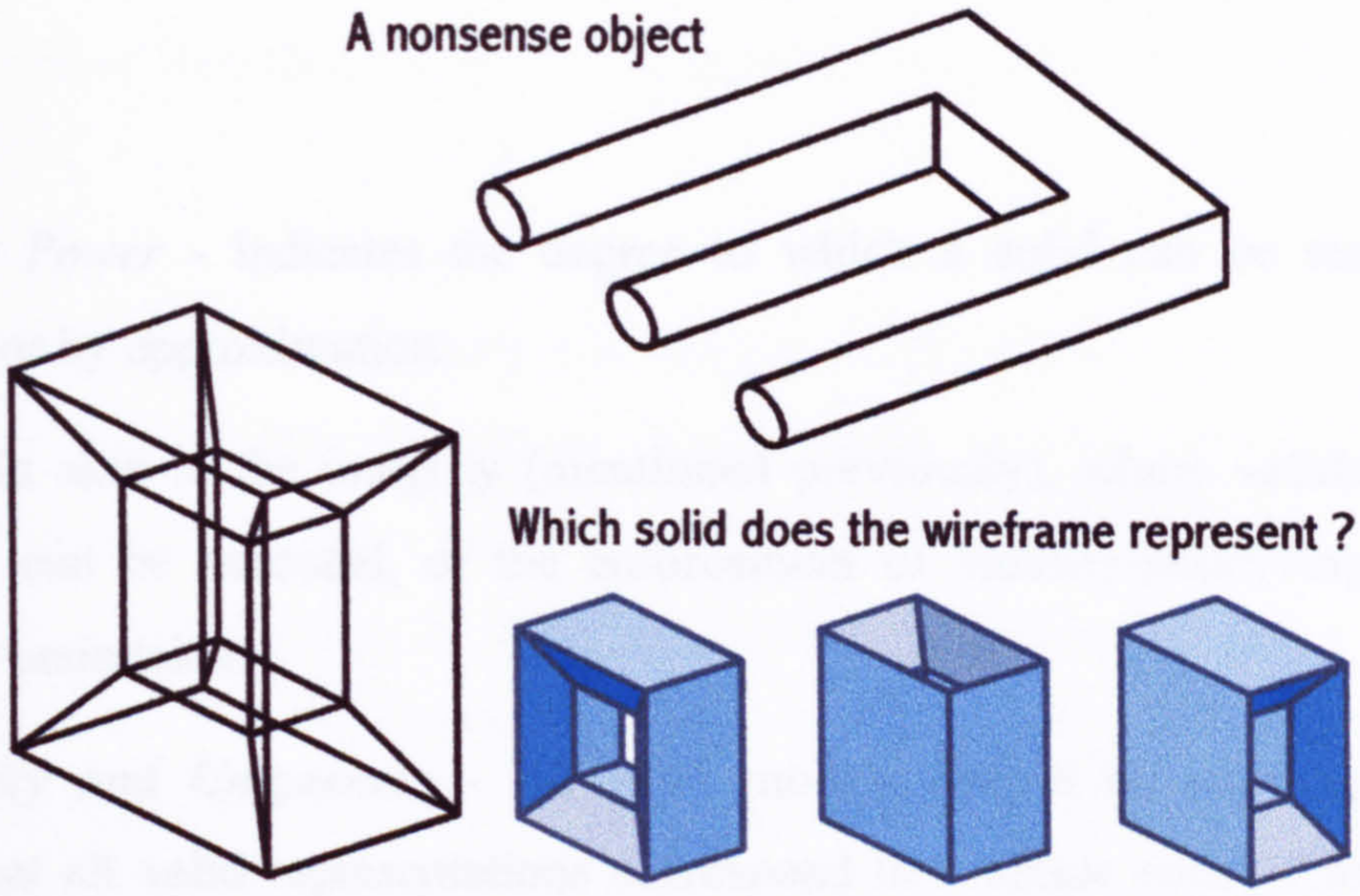


Figure 2.17 Ambiguous 3D Graphical wireframes

Three-dimensional graphical models are an extension of their two-dimensional parent, being represented essentially by the inclusion of an extra dimension (as x, y and z for a point). These are termed wireframe models, as they hold no direct volumetric interpretation. Hence, whilst being very fast to reproduce on a graphics terminal, they can be ambiguous. Examples of this include those shown in figure 2.17. Enhancements to graphical models, through the use of layers and colours etc., only represent entities of an object, and not its solid form. This makes it difficult to visualise complex, and even simple objects (again see figure 2.17), and due to this weakness, graphical models are not wholly suitable from a design reuse perspective.

2.8.2 Solid Models

The aim of Solid Modelling is to create a complete and robust representation of a 3-dimensional geometric design, and in comparison to 3-dimensional, graphical models, in an unambiguous manner. There are a number of factors that influence the capability of a solid modelling system. Of these, two are prominent. The ability to maintain the *integrity* of a model, through an integrity-checking algorithm, or by limiting model construction to only integrity-preserving operations. Also, it is useful to handle large models at differing levels of complexity (or abstraction), which calls for the use of part and assembly modelling. Further characteristics of solid modelling

techniques can be used to classify various approaches to the requirements of this research:

Expressive Power - indicates the degree to which a solid can be modelled, i.e. accurately or by approximation.

Validity - is akin to the integrity (mentioned previously), where validity-checking algorithms can be executed, or the enforcement of validity-preserving modelling techniques undertaken.

Unambiguity and Uniqueness - All solid models should be unambiguous. This requires that all valid representations correspond to a single solid. Furthermore, if only one representation of a solid exists, then that representation is said to be unique.

Description Languages - specify the 'input method' for a given representation.

Conciseness - characterises the amount of space required to store the representation. Clearly this should be kept to a minimum.

Computational Ease and Applicability - are measures of the algorithms that can/must be written to realise the representation scheme, from an applications viewpoint. And also implies the suitability of a particular scheme to a given application.

Almost two decades ago, Requicha (1980) defined six such schemes, suitable for the representation of unambiguous solid models. The following sections will discuss only the representation schemes related to this research, and their particular relevance to storing adaptive solid models, being a primary objective of this research.

2.8.3 Pure Primitive Instancing

This is a parameter-based scheme, where a *generic primitive* is created to represent a family of similar designs. The scheme is based around an implicit, or *procedural*, representation of the solid. Therefore individual family members can be instanced by specifying their parameters and re-executing the stored procedure. Pure Primitive Instancing has its roots in a concept known as Group Technology (Hyde 1981). This is a technique used in Computer Integrated Manufacturing (CIM) to assist process

planning, design retrieval and scheduling (for example), by grouping similar parts into standardised families, thereby encouraging the use of standard parts and components.

The underlying principle of grouping families of similar designs into a single generic model, is of considerable interest to this research. To this end, the author has developed a similar technique, Parametric Primitive Instancing (Andrews 1996). The goal of this application is to efficiently distribute solid models of standard (catalogue) parts. This involves the creation of generic, primitive models for standard component families, such as spur-gears and bearing-housings, which can be fed into an intelligent engine, to produce the required instances (figure 2.18).

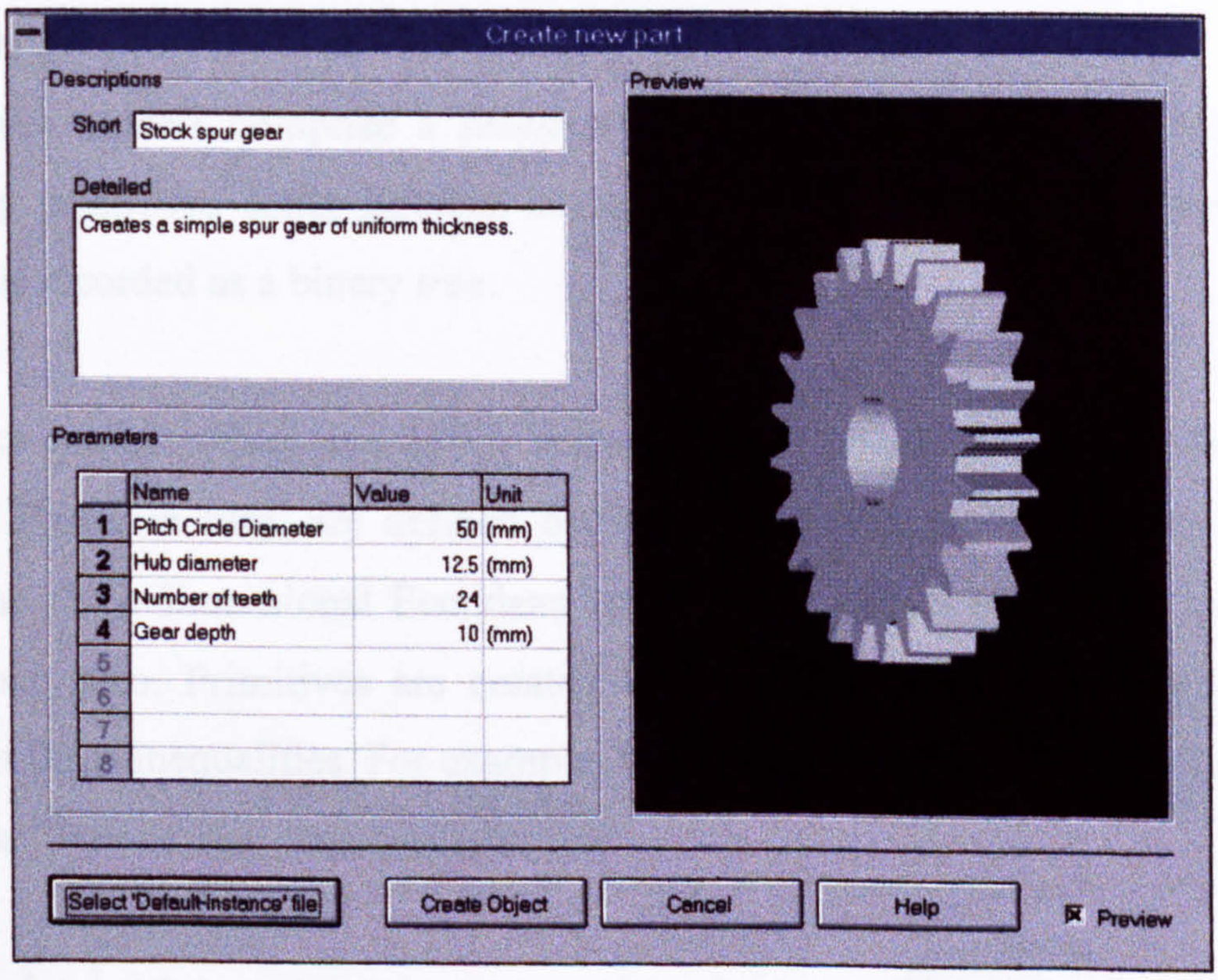


Figure 2.18 Example of the PPI application – Creating A Spur Gear Instance

This representation scheme holds many advantages over traditional geometric modelling systems. Firstly, its ease of modifying the *shape* of a solid. It is also very efficient in terms of storage, requiring only the generic primitive and the set of necessary parameters to store an entire family of designs. The scheme is also unambiguous and unique. To some degree, the original intentions of the designer are maintained, as these are hard-coded within the generic primitive. However, a major

drawback of this scheme is that, being procedural, only the geometry (or shape) of the generic primitive can be changed. Major changes in the *topology* of solids is difficult to achieve, as there is no scope for conditional parameter definitions. Also, the scheme can be slow and resource consuming, as it requires the solid to be built from scratch (generated) each time it is instanced.

Although this method has significant drawbacks, its foundations are relevant here. The ability to group a family of similar part designs into a single, generic model, is an efficient means of storing a family of past designs. Along with the ability to instance particular family members with a given set of parameters, Pure Primitive Instancing, in some form, can be used for this research.

2.8.4 Constructive Solid Geometry

Constructive models comprise a *set-theoretic* approach to representing solids by combining primitives using Boolean set operations. The history by which this is achieved is recorded as a binary *tree*.

Half-Space models (Requicha 1977) define a volume bound by a combination of surfaces. These, in turn, are defined by inequality relations, such as $z>0$, which defines the three-dimensional Euclidean space for all points with a 'z' co-ordinate greater than zero. Primitives are created by performing Boolean operations to a number of these inequalities. For example, the cylinder of figure 2.19 can be defined as follows:

$H_1 : x^2 + y^2 - r^2 < 0$

$H_2 : z > 0$

$H_3 : z - h > 0$

$\text{Cylinder} = H_1 \cap H_2 \cap H_3$

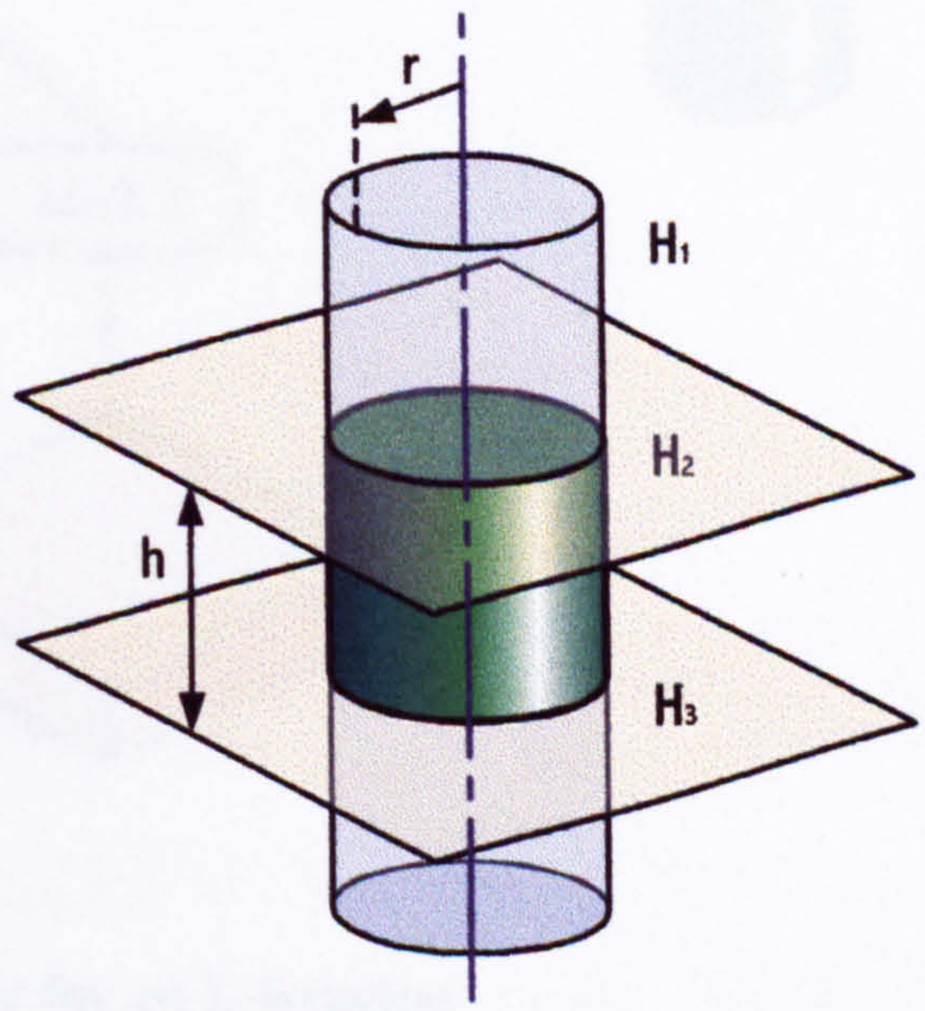


Figure 2.19 A Simple Half-Space Model (Mäntylä 1988)

By itself, the Half-Space model is of limited use, as it is often inconvenient to construct a model in terms of complex inequalities. Hence these models are usually used as the basis of representation of other schemes. Constructive Solid Geometry (CSG) models (Voelcker and Requicha 1977) make use of Half-Space models as bounded, pre-defined and parametric primitives, analogous to Pure Primitive Instancing. These can be instanced and combined by the use of *Union*, *Difference* and *Intersection* Boolean operations, and simple transformations to represent a complete solid model, and are structurally represented by the CSG-tree (figure 2.20 for example). The model of the 'L' bracket is formed by instancing two rectilinear blocks, using a union operation to create the L shape. A cylinder primitive is then instanced, and subtracted (by a difference operation) from the L.

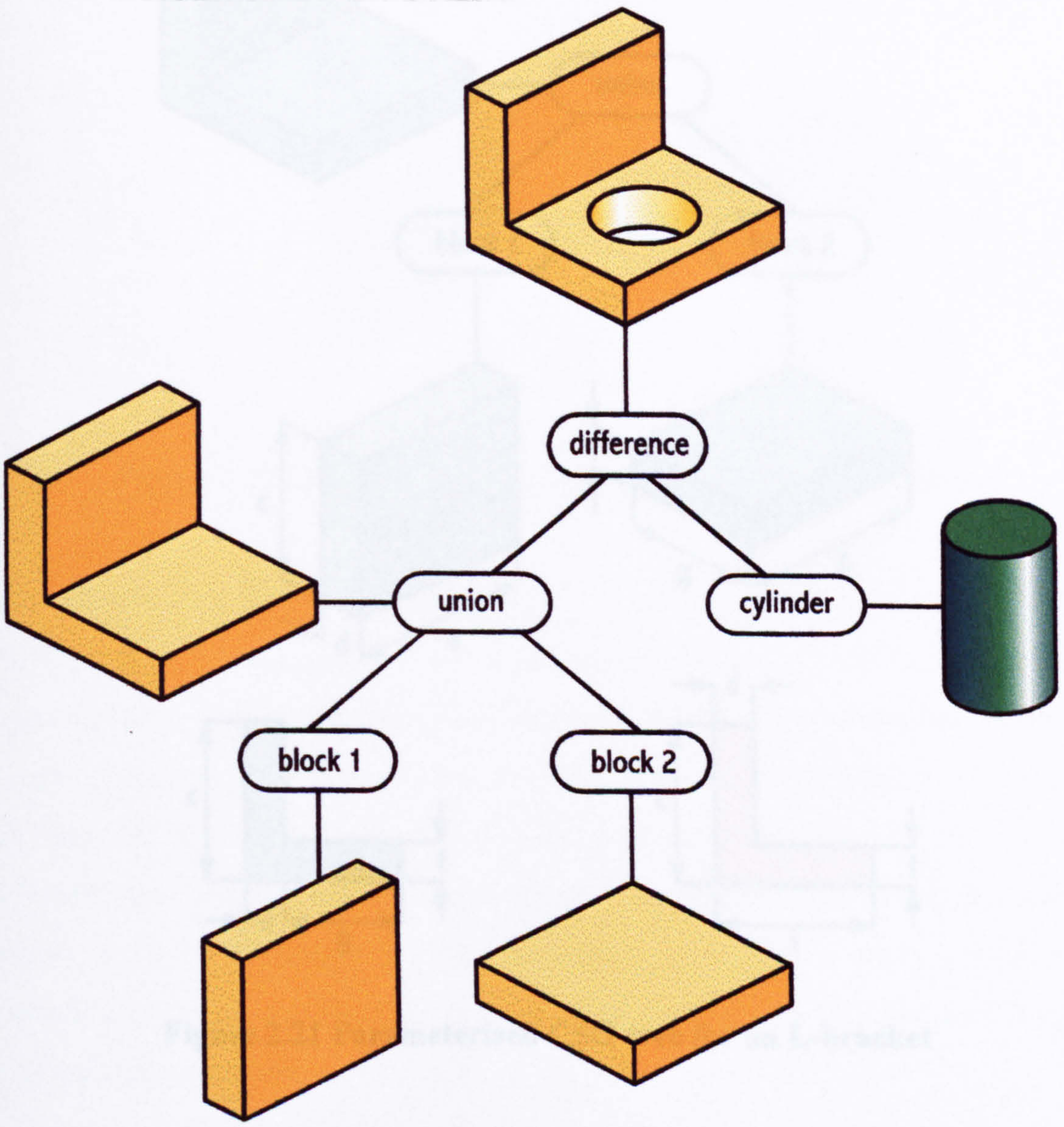


Figure 2.20 A CSG-tree for an L-bracket

The CSG representation scheme is very efficient in terms of storage requirements, being a high-level interpretation of the solids construction process. Its resultant solid models are unambiguous and valid, as they are based upon *regularised set operations*, which will always result in the interior closed volume of its *set-theoretic* operations. However, CSG is not unique. Also, being an implicit data structure, unforeseen future modifications to the CSG solid model are difficult to implement (Zuffante 1986). For example, figure 2.21 shows the 'parameterised' CSG-tree for the L-bracket (minus the hole).

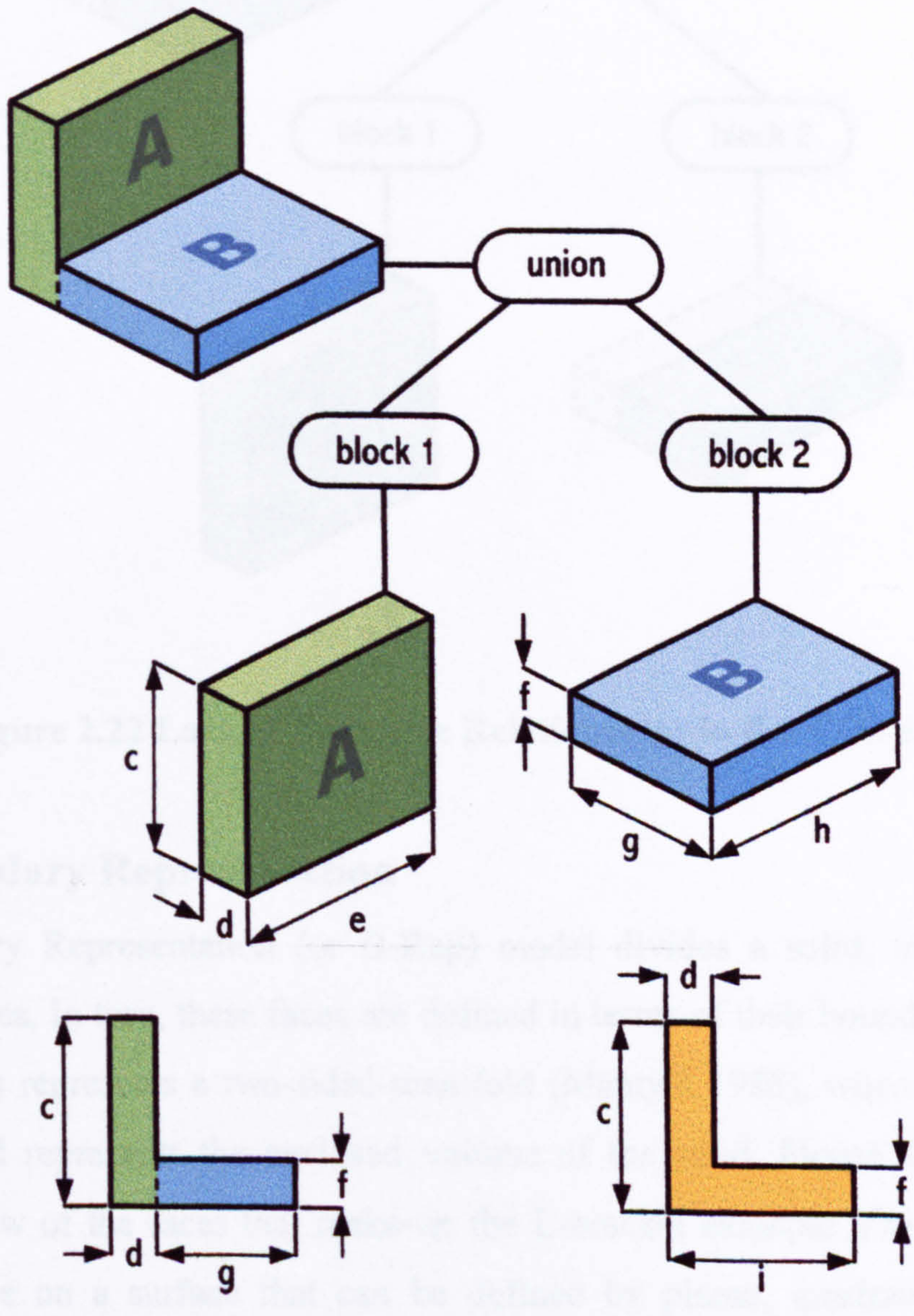


Figure 2.21 Parameterised CSG-tree for an L-bracket

A block 'A' of dimensions 'c x d x e', and a block 'B' of dimensions 'f x g x h' are united to form the L-shape. However, the user of the system may wish to represent the bracket dimensions in terms of overall height and width (e.g. 'c x i'). Such a

requirement cannot be fulfilled with the standard CSG representation scheme. Even if such parameterisation was possible, design intent can be lost, as CSG does not maintain information relating to mating of primitives (figure 2.22).

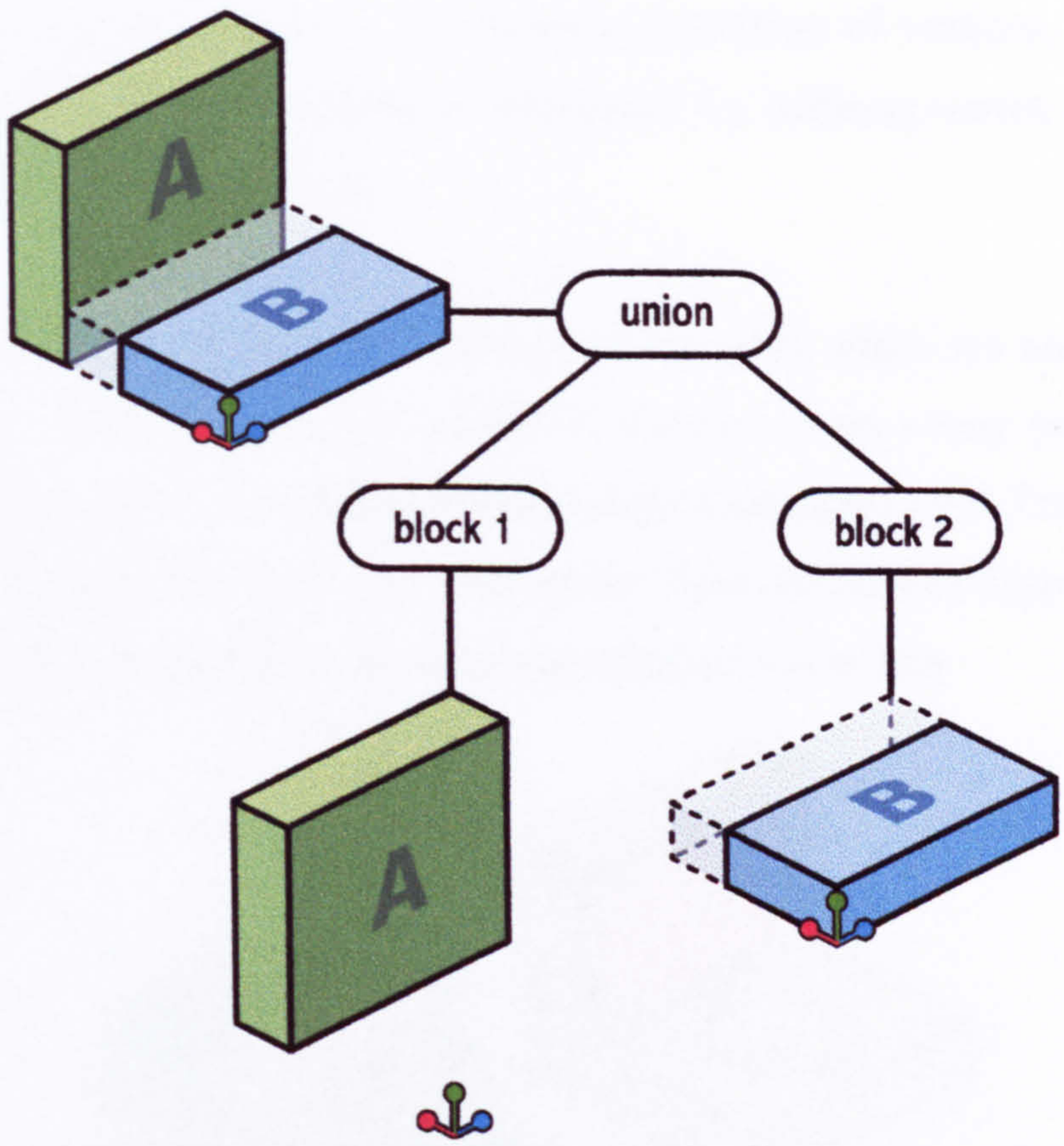


Figure 2.22 Lack of Primitive Relationships in the CSG-tree

2.8.5 Boundary Representation

The Boundary Representation (or B-Rep) model divides a solid, in terms of its bounding faces. In turn, these faces are defined in terms of their bounding edges and vertices. This represents a two-sided-manifold (Mäntylä 1988), where the inside of this manifold represents the enclosed volume of the solid. Figure 2.23 shows an exploded view of the faces that make-up the L-bracket example. Faces are usually derived to lie on a surface that can be defined by planar, quadratic, toroidal or parametric expressions, which are also included in the B-rep data structure. Typical B-rep structures include:

Polygon-based Boundary Models - where all edges are straight lines and, therefore, all faces are planar (polygons). This structure is used extensively in graphics based applications.

Vertex-based Boundary Models - the wasteful repetition of vertices, when defining faces in the polygon-based models, is eliminated by defining vertex entities, which can be referenced to define faces.

Edge-based Boundary Models - for models where some edges are not straight lines. Here, edges are defined as entities, which are closed to form a loop (see figure 2.24). Examples of this model include the Winged-edge (Baumgart 1974,75) and Half-edge (Mäntylä 1988) data structures, as well as the Face-Adjacency-Hypergraph (FAH) which is a useful representation for automatic feature extraction.

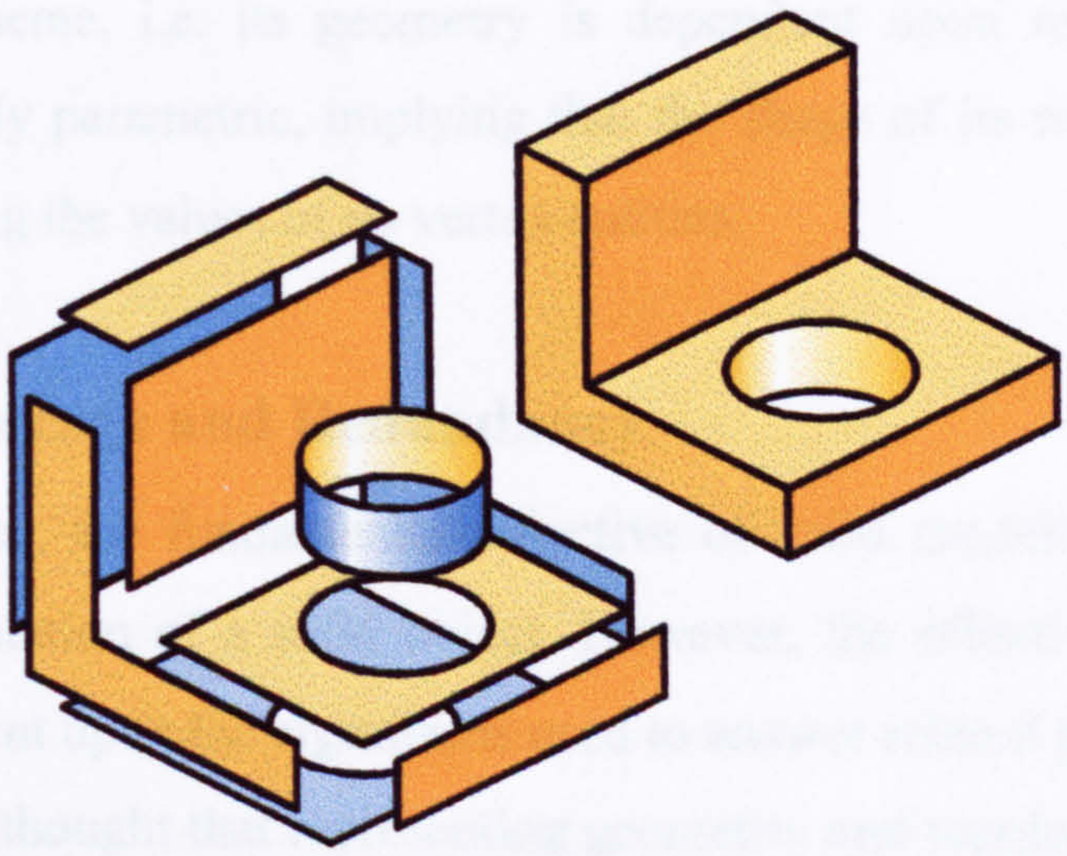


Figure 2.23 Faces Bounding the L-Bracket

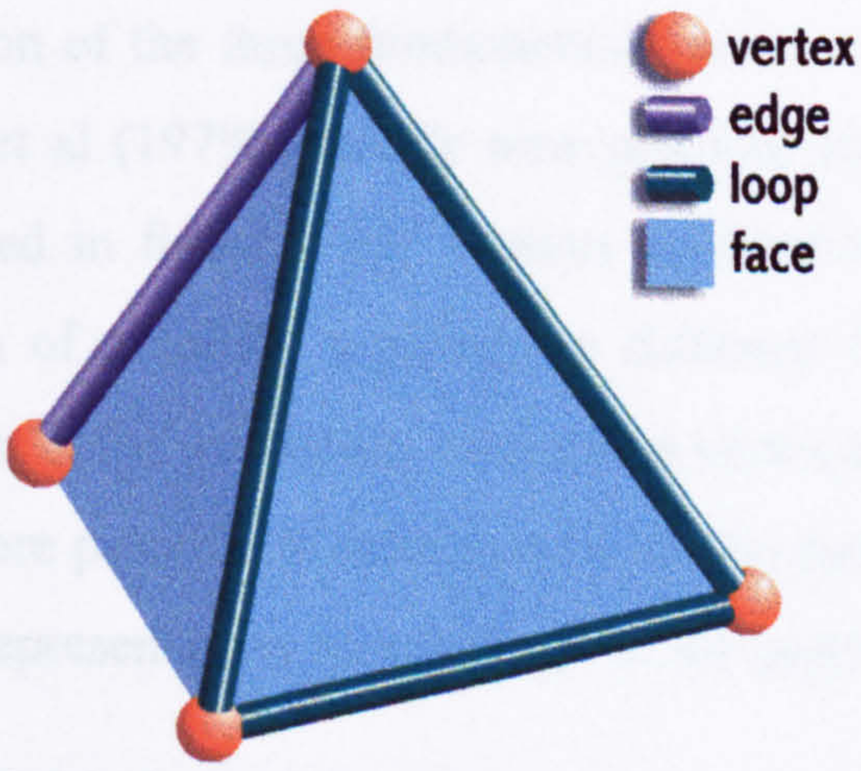


Figure 2.24 Various Entities of a B-rep model

Boundary Models can be created using a variety of techniques, of which the drafting interfaces of Graphical Representations are popular. Other techniques include Sweep Representations and CSG construction schemes. However, despite the expressive power of B-rep models, they are invariably difficult to validate. CSG conversion techniques can produce vulnerable results, and the use of incremental sweeping operations is considered unsafe (Braid 1979). Although the use of the Euler-Poincaré formula and its derived *Euler operators* (Mäntylä 1988) can be used to determine the integrity of Boundary Models. Further disadvantages of B-rep include the size of its models, and that its representations are not unique (Woo 1985).

The ease with which Boundary Models can be constructed (or rather input) has made the use of B-rep, in some form or another, a popular choice for current geometric modelling kernels. To this extent, they are of relevance here. B-rep is an explicit representation scheme, i.e. its geometry is dependent upon related entities. It is therefore inherently parametric, implying that the shape of its models can be easily altered by changing the values of its vertex entities.

2.8.5.1 Data Storage and Redundancy

As has been stated, the fundamental objective of solid modelling is to provide a *complete* representation of a solid object. However, the effectiveness of boundary models is dependent upon the algorithms used to answer related geometric questions. Originally, it was thought that representing geometric and topological data explicitly enhanced the capability of these algorithms.

Consider the representation of the three fundamental vertex, edge and face entities for a simple cube. Baer et al (1979) identify nine possible combinations for these representations, as outlined in figure 2.25. Various applications require (or rather prefer) the representation of a solid's topology in different forms, e.g. facets (or faces) are more useful for 'solid' rendering, whereas a vertex-only representation is more concise. It is therefore possible to state that no single data structure provides a completely satisfactory representation of topology in all practical cases, and some redundancy is inevitable.

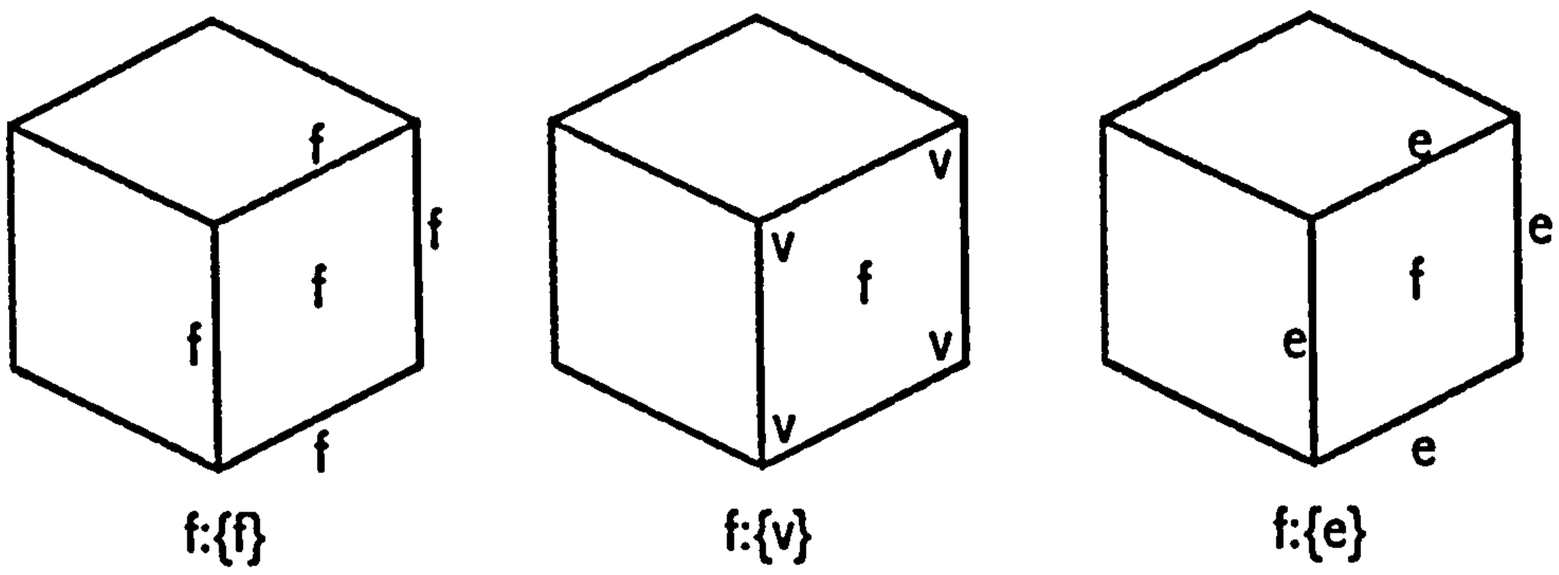


Figure 2.25 – Nine Topological Relationships [Baer at al.]

2.8.6 Relevance of Solid Modelling Systems

Of the representation schemes defined by Requicha (1980), the following three have been discussed to be of relevence to this research:

- Pure Primitive Instancing
- Constructive Solid Geometry
- Boundary Representation

Of these, CSG and B-rep hold major significance as they are successful and well established methods of representing solid models. In fact, current research and commercial solid modelling systems have combined the distinct advantages of these two schemes, to form *hybrid* modellers. Here, CSG is used primarily to validate representations, and B-rep is used to define loops and surfaces in a parametric

fashion. However, they do not readily facilitate the requirements of representing designs from a family range. On the other hand, Pure Primitive Instancing is based around this principle, be it typically only for piece parts. Although, it has the disadvantage of being limited to regularised shape changes.

2.8.7 Enhanced Solid Modelling Schemes

As they stand, CSG and B-rep schemes, and their hybrids, have evolved through four significant advances, as defined by Requicha and Voelcker (1983):

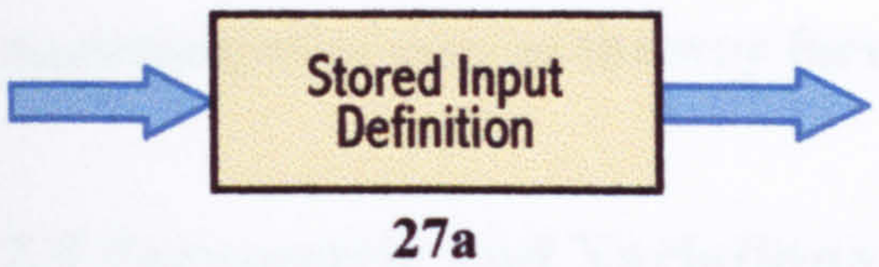
1) *Stored Input Definitions* - only the inputs (i.e. the construction history) is stored,

2) *Volatile Input Definitions* - is an initial attempt to store a useful representation of the solid, where the inputs are deemed unnecessary and discarded,

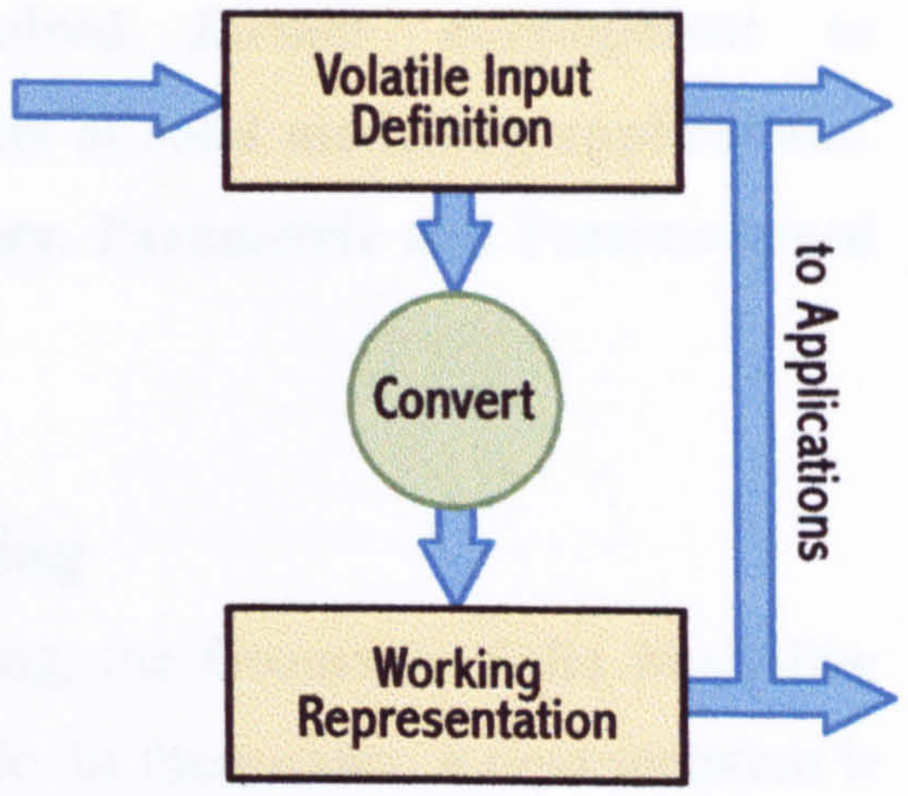
3) *Stored Input Definitions with Approximate Representations* - is an application of an approximated B-rep scheme,

4) *Stored or Volatile Input Definitions together with Auxiliary Representations* - here, auxiliary representations of the model are stored to assist validation and modification (for example), as well as the original input definition.

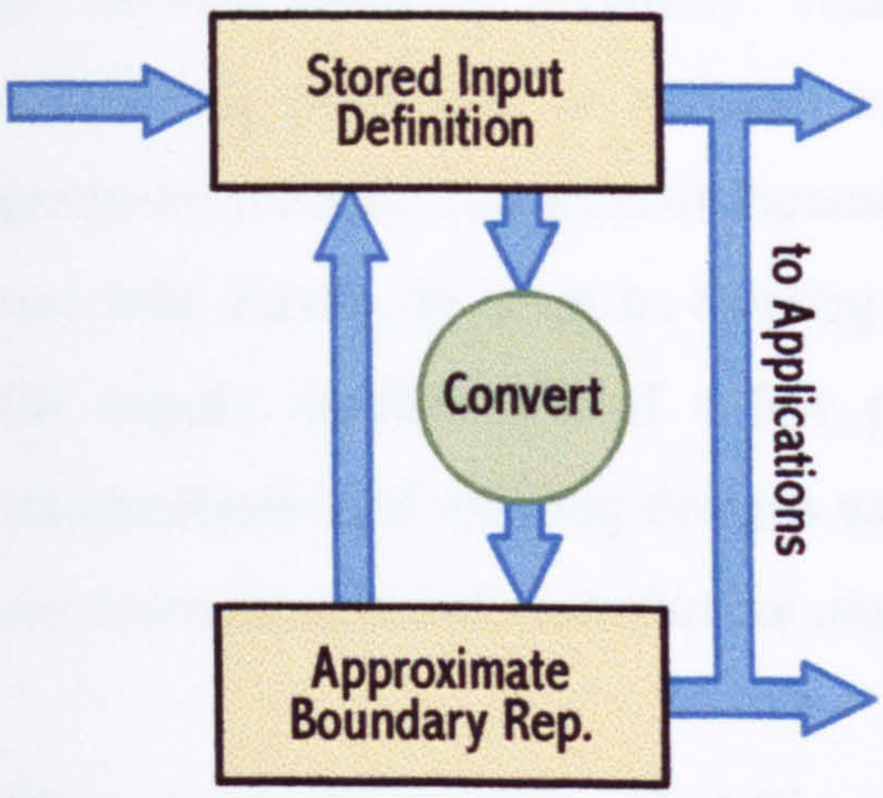
The significance of these definitions (figure 2.26), and particularly for this research that of figure 2.26d, is the use of auxiliary representations. Although these representations add to the size and complexity of a model definition, their use can overcome some of the more static properties of Geometric Modelling systems (Nielson 1987 and Voelcker 1988). Research over the past decade, has seen the growing use of Parametric, Variational and Feature-based (auxiliary) representations, which shall be discussed in the following sections.



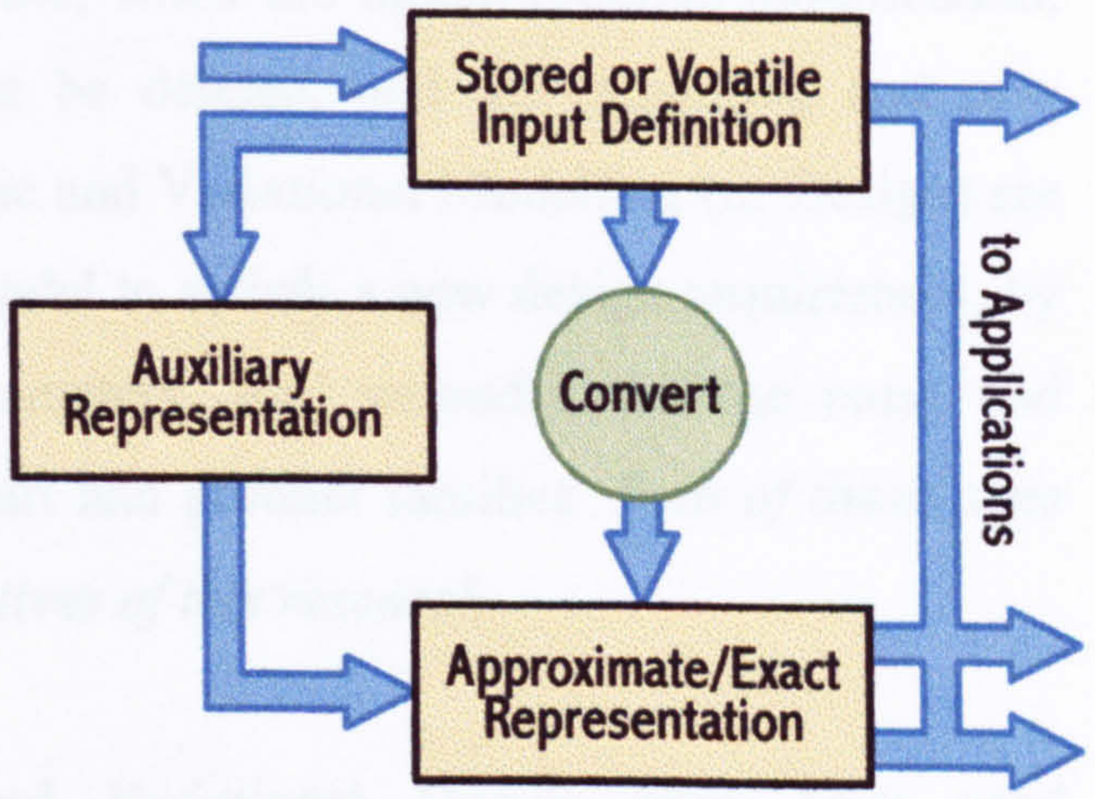
27a



27b



27c



27d

Figure 2.26 Solid Modelling Schemes Requicha and Voelcker (1983)

2.8.8 An Overview of Geometric Modelling

The preceding subsections, from 2.8.1 to 2.8.7, described the development of the 'Geometric Modelling' paradigm, and its applications.

It started with the objective of having a complete representation of the object modelled. Initial attempts were concerned with issues of ensuring Validity, Uniqueness etc. Primitive Instancing, Half-Space models, CSG and Boundary Representations were developed as promising modelling techniques. Application algorithms were also developed with these schemes. Redundant data storage is seen as a method to resolve application issues. Finally, Hybrid Modellers (having more than one representation scheme) were developed to contain the accumulated benefits of the schemes included. This paradigm, even

with its significant developments, required further development to accommodate unforeseen, future requirements of solid modelling applications. It was felt that a significant leap was necessary. Parametric and Feature-based modelling were seen as the way forward.

2.9 Parametric and Variational Modelling

With the exception of Pure Primitive Instancing, the Geometric Solid Modelling systems defined so far can be described as *static*. In these cases, a representation is created, where no definitive relationships between primitives (and parts) exist. These are defined solely by geometry. Therefore, when the model requires modification, obstructing primitives or surfaces must be deleted, and the remaining and new geometry created. The aims of Parametric and Variational Modelling (or Design) are two-fold. Firstly, to adapt an existing model to satisfy a new design requirement, by the simple modification of a few parameters. And secondly, for the reuse and standardisation of existing designs as part and product families. *Both of these aims are relevant and well suited to the objectives of this research.*

The terms Parametric Modelling and Variational Design have been used interchangeably in both academic and commercial domains (Kurland 1996). In fact, little or no distinction between the two may be apparent to the users of such systems, as their construction process is similar (figure 2.27):

1) Create a *nominal model* of the design using standard geometric modelling operations, but with no specific dimensions stated.

2) Define geometric *constraints* between entities. These are generally in the form of dimensional, or entity-to-entity constraints. E.g. set a line to be vertical, or set line A to be parallel to line B.

3) Evaluate, or *regenerate*, the models constraints, by use of a *general solution procedure*.

4) Create *variants* of the model, by changing parameter values and re-evaluating the general solution procedure.

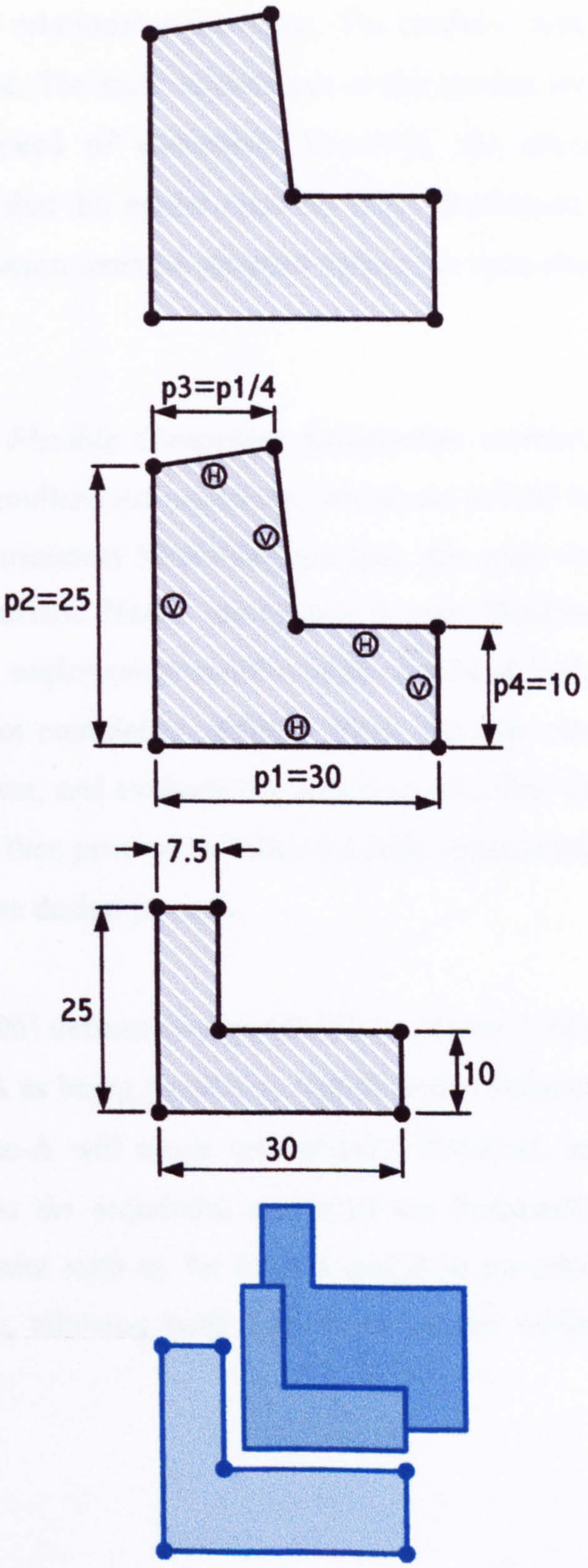


Figure 2.27 Constructing Parametric and Variant Models

The difference between the Parametric and Variational techniques lies with the method(s) used for the general solution:

Parametric Modelling techniques make use of a *Rigid Constraint Satisfaction* procedure. In this case, the construction history, parameter assignments and constraints are stored in a defined, sequential order. Parameter assignments can

include both numerical values and simple relational expressions. The model is then solved according to this recorded sequence. The main advantages of this system are its simplicity to implement, and its speed of execution. However, the main disadvantage of Parametric Modelling is that the model must be fully constrained. As each entity in this sequential representation must be satisfied before the next one can be solved.

Variational Modelling systems adopt a *Flexible Constraint Satisfaction* method. Constraints are represented by a set of simultaneous equations, which are solved to realise the design. The advantages of Variational Modelling are that, the order in which constraints are defined is not important. Hence the system is more flexible from the users perspective. Furthermore, *under-constrained* models can be solved, i.e. for models where the geometry is not completely defined. Here, the user can define which constraints are actually known, and evaluate the model to get-a-feel of how it will look and react to changes, and then proceed to achieve a fully constrained model. This also allows for a more intuitive design process.

To illustrate this difference, Kurland (1996) defines two parallel lines (figure 2.28). A Parametric Modeller may define line-A as being parallel to line-B, and a distance 'x' apart. So when line-B is moved, line-A will move respectively. However, an attempt to move line-A will fail, due to the sequential nature of the Parametric system. For a Variant Modeller, a constraint such as *'let lines A and B be parallel, and a distance 'x' apart'* may be given, allowing both lines to be moved whilst maintaining this constraint.

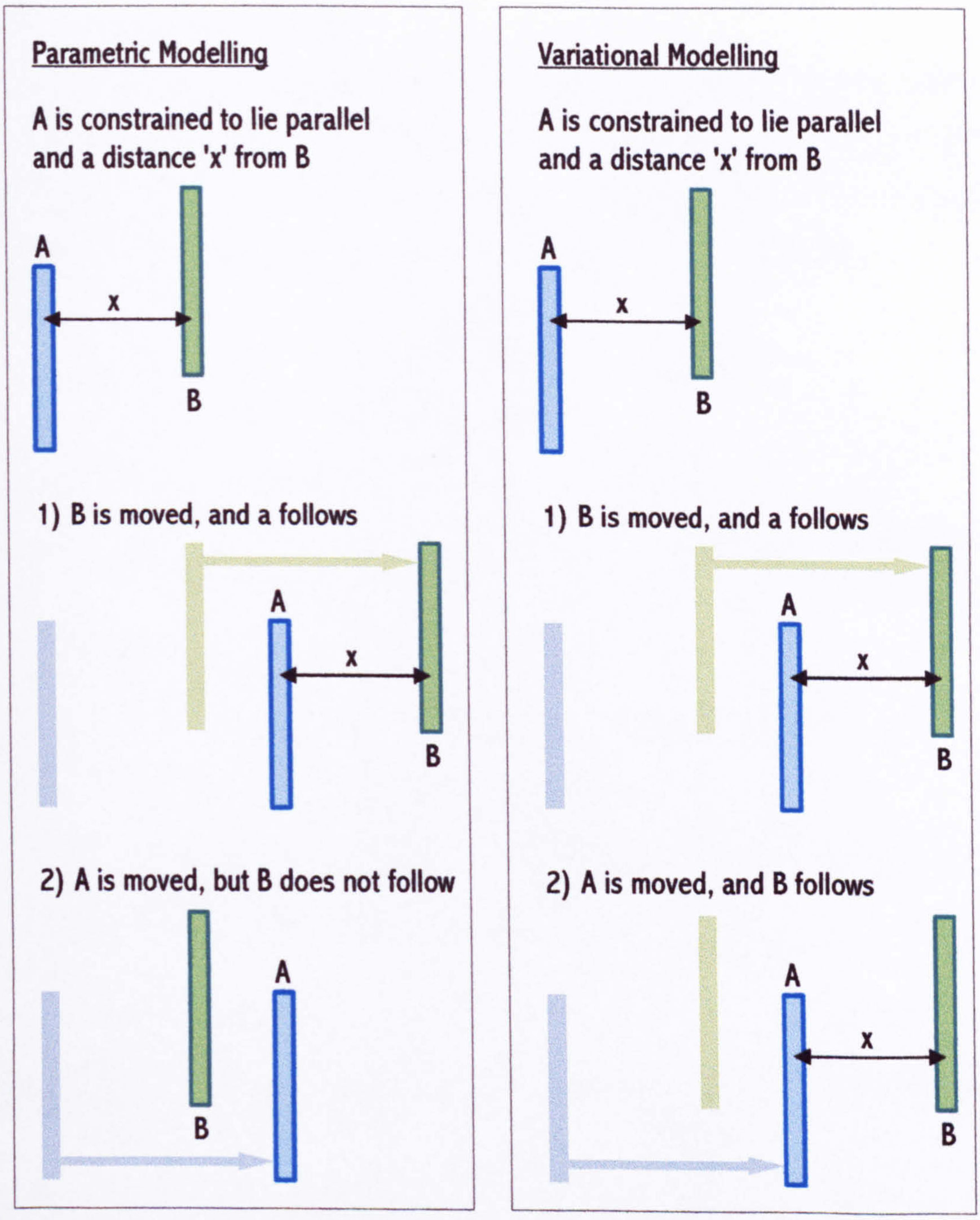


Figure 2.28 Difference between Parametric and Variational Systems

Many authors use differing terminology for these approaches. For example, Parametric Modelling can also be described as an *explicit* form of Variational Design (Shah and Mäntylä 1995), or more generally as being *Procedural*. Whereas Variant Design is termed as being *implicit*. Moreover, both the procedural or Parametric, and implicit Variational Modelling techniques are suited to storing a family of similar designs as a single adaptive model. However, both of these techniques still do not express the engineering significance of a model. Therefore, the following section shall discuss the use of features in solid modelling and design.

2.10 Feature Based Design

In their definition of features, Shah and Mäntylä (1995) state that a feature represents the engineering meaning or significance of the geometry of a part or assembly. Features can be thought of as building blocks for product definition, or for geometric reasoning. For example, consider the design represented in figure 2.29.

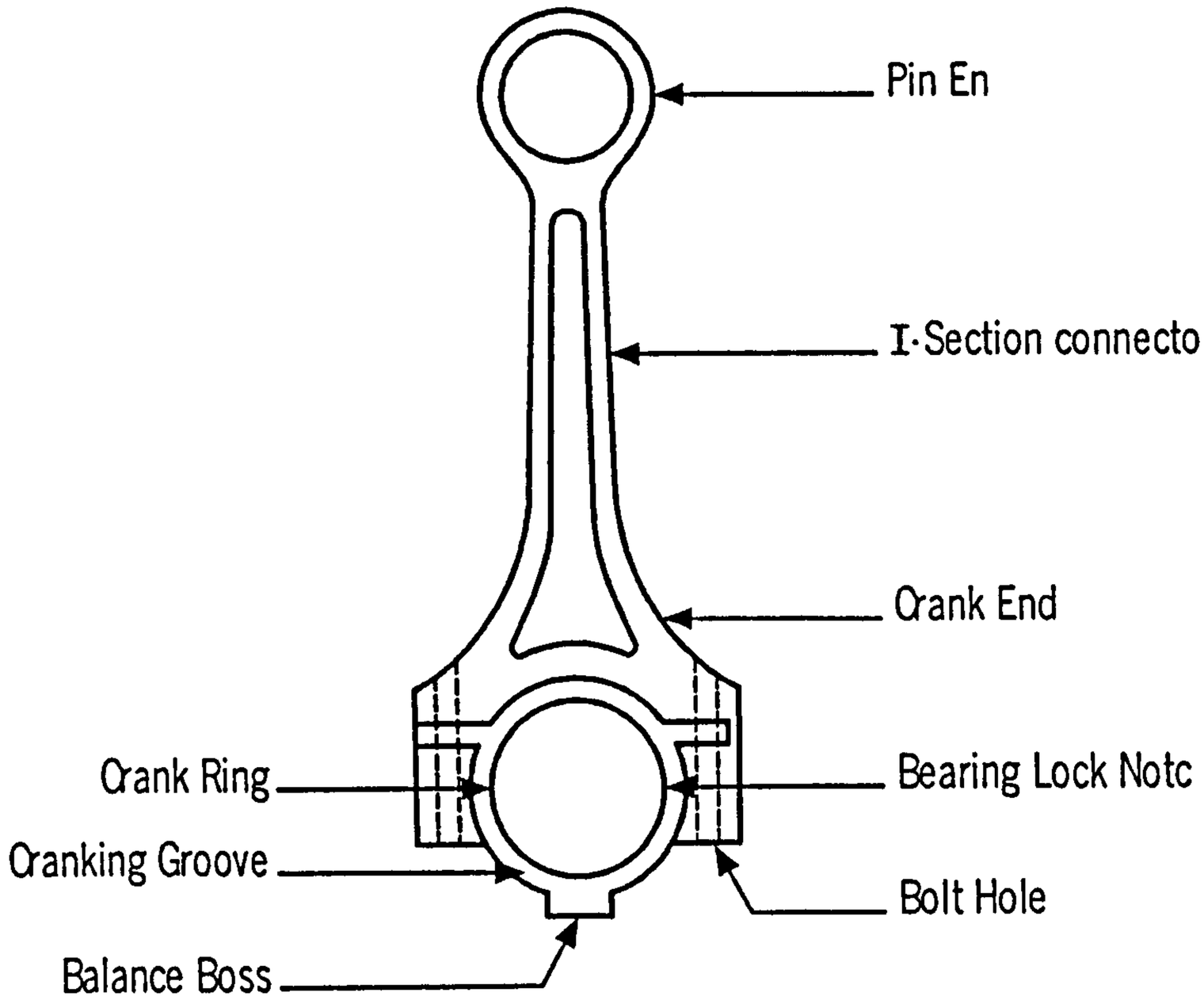


Figure 2.29 – Design Features of a Connecting Rod (Shah and Mäntylä 1995)

The figure shows the design features of a 'con-rod', and through the combination of these features a complete definition of the design is achieved. Therefore, the characteristics of a feature can be listed as follows:

a) a feature is a physical constituent of a part,
b) a feature is mappable to a generic part,
c) a feature has engineering significance, and
d) a feature has predictable properties.

A feature can be a single entity (or primitive), or a combination of related primitives, that perform a defined function. Features (should) also contain and maintain constraints to their surroundings. A simple example of a feature is a hole. In

geometric terms this can either be represented as a cylinder, subtracted from a given *base* model (for CSG), or as a cylindrical face, bound at both ends, but whose inner volume is void (for B-rep). However, an engineer will typically define a hole as being 'a cut-out of a given *diameter* and *depth*, or as being drilled straight *through* the base model', for example, as shown in figure 2.30a and 2.30b.

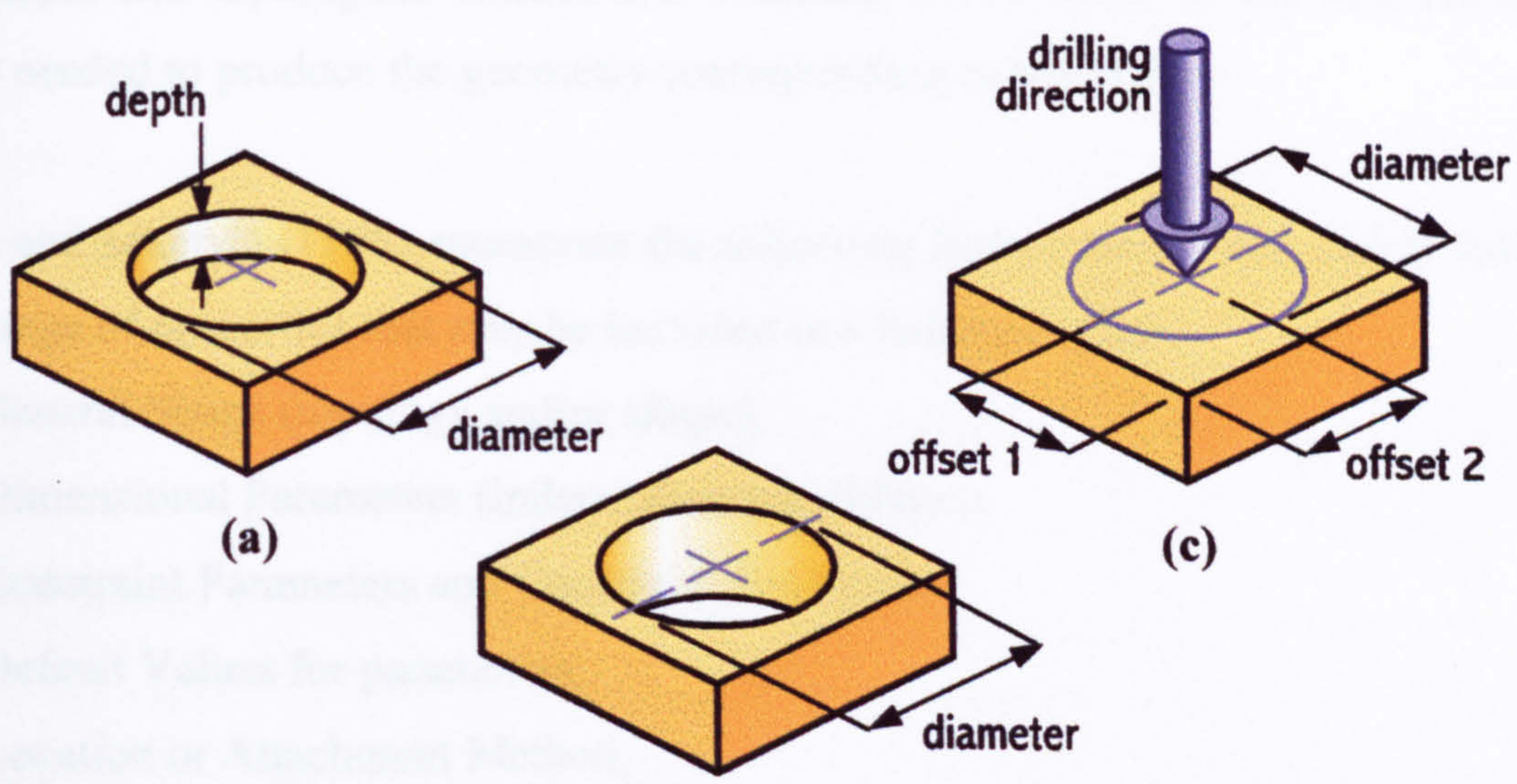


Figure 2.30 A Simple (Blind) Hole Feature

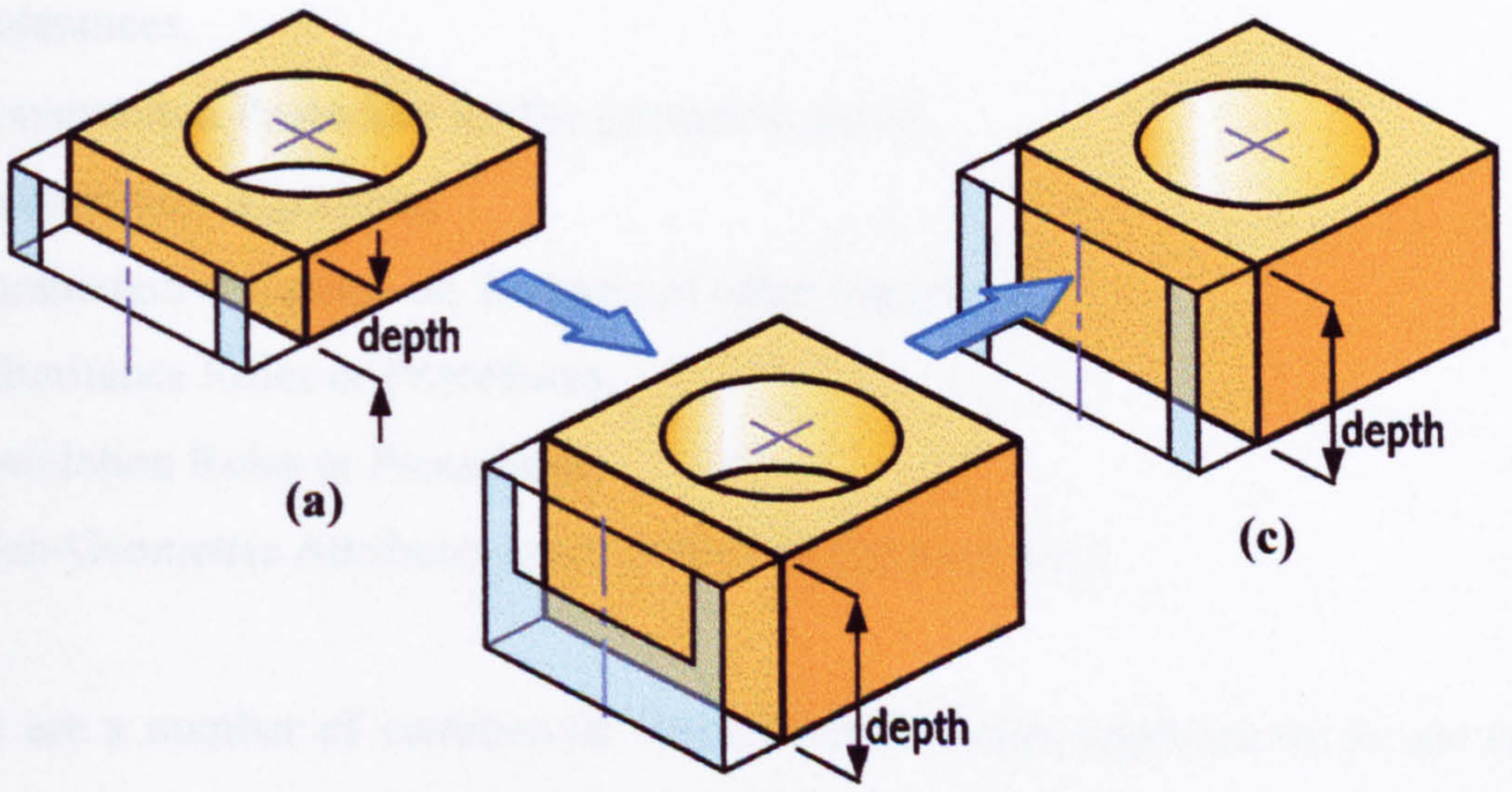


Figure 2.31 A Through-Hole Feature

This feature should also contain information as to its location and position on the base model (figure 2.30c), and if, say, defined as a *through-hole*, should be able to automatically adapt itself according to changes in its *parent* entities, i.e. the base part to which it is attached (figure 2.31).

A feature model is a data structure that represents a given part or assembly, primarily in terms of its constituent features. Each feature in the feature model is an identifiable entity that has some explicit representation. The shape of a feature, as shown earlier, may be expressed in terms of dimensional parameters, enumeration of geometric and topological entities and relations, or, in terms of the constructional steps needed to produce the geometry corresponding to the feature.

Shah and Mäntylä (1995) enumerate the following feature properties, which indicate the range of properties that may be included in a feature model:

a) General Shape (topology and/or shape),
b) Dimensional Parameters (independent parameters),
c) Constraint Parameters and Constraint Relations,
d) Default Values for parameters,
e) Location or Attachment Method,
f) Location Parameters,
g) Orientation Method,
h) Orientation Parameters,
i) Tolerances,
j) Construction Procedure for the geometric model,
k) Recognition Algorithm,
l) Parameters computed on the basis of other features,
m) Inheritance Rules or Procedures,
n) Validation Rules or Procedures,
o) Non-Geometric Attributes (part number or function etc.).

There are a number of commercial feature-based design applications in use today. Prominent examples include Pro/ENGINEER (Parametric Technology Corporation), Mechanical Desktop (Autodesk) and SolidWorks (SolidWorks Corp.). All of these systems provide a subset of the above characteristics of modelling with features and thus make the detailed design process more flexible and useful.

2.10.1 Feature Creation Methods

Features are clearly an integrated part of Computer Aided Design and Engineering. They possess reuse facilities for the design synthesis, manufacturing and adaptation stages. Therefore it is beneficial to represent the computer model, related to this research in terms of features. Shah (1991) and Feru et al (1992) define the two methods of feature creation as follows:

Form Feature Recognition - where features are recognised and extracted, by some means, from an existing, defined geometric model, and

Design by Features - the solid model is constructed as a combination of features.

2.10.2 Form Feature Recognition

With this method, a solid model, already created using the Geometric Modelling techniques described earlier in sections 2.8-2.9, is decomposed into form features. This process is governed by a *Feature Recognition System* and a *Feature Database*, which contains generic primitives of various features, to which elements of the solid model can be compared. This process can also be either interactive or fully automatic:

Interactive Feature Recognition – Here the created geometric model is displayed via a suitable user-interface. The user then picks elements of this model, which they wish to be recognised as a feature. The feature recognition system then compares this geometry to what is stored in the (feature) database, and extracts the relevant geometry from the solid model, whilst adding the feature to an evolving *feature model* (figure 2.32).

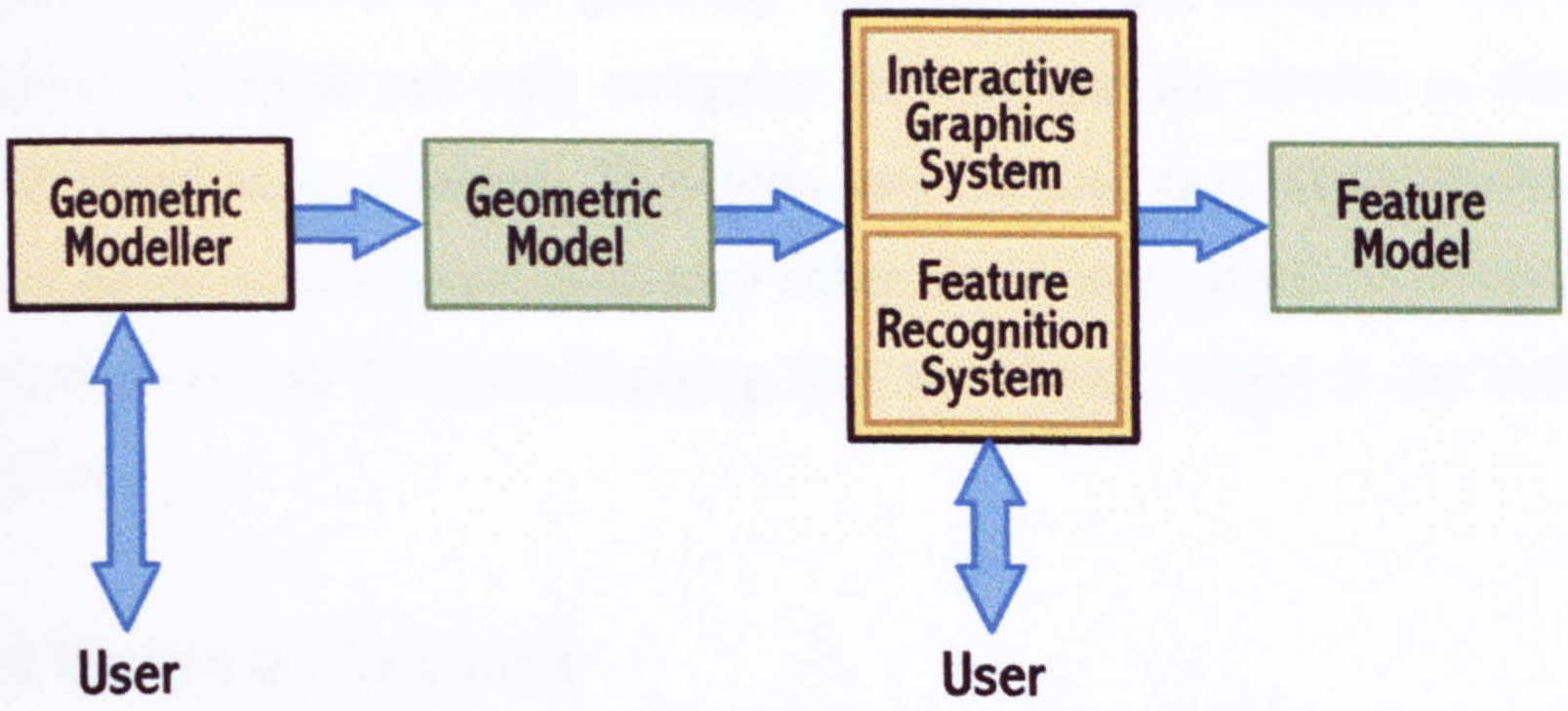


Figure 2.32 Interactive Feature Definition

Automatic Feature Recognition - This technique was originally developed as a method for *Machining Region Recognition* (a subset of CAPP). However, here interest lies in dealing with features bound by the interior volume of the solid model, and not from a machined volume. Therefore, we will discuss what is termed Pre-Defined Feature Recognition. This is a fully automated system (i.e. there is virtually no user-input to the recognition process). Again, the process starts with an existing solid model, which is processed through various recognition and extraction algorithms. These typically compare groupings of either B-rep or CSG-tree elements, to defined 'generic' features in the Feature Database, and perform the extraction to form a Feature Model (figure 2.33).

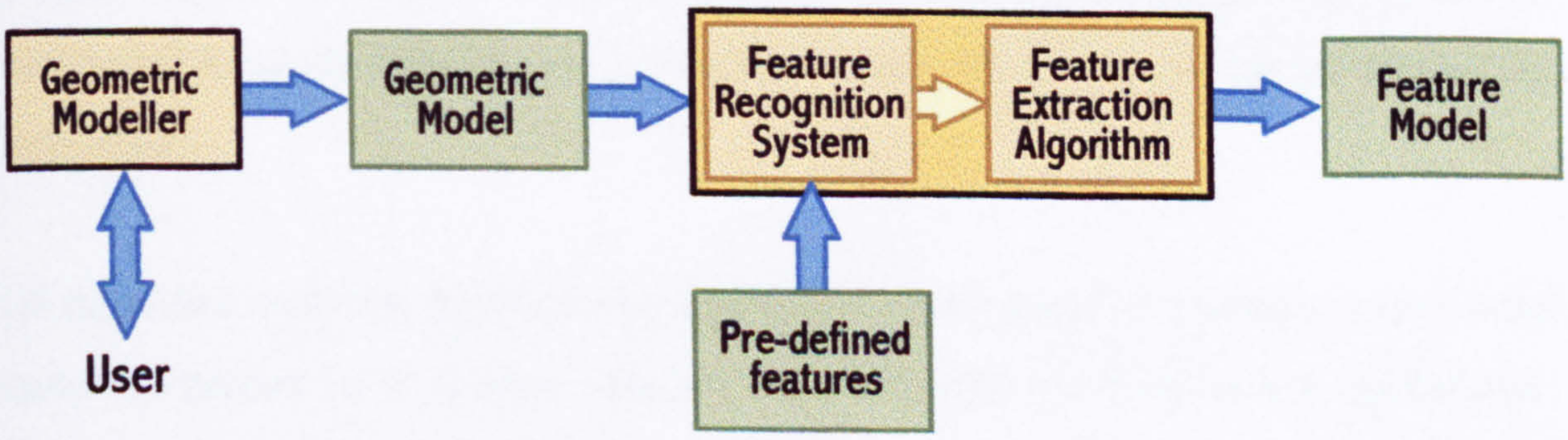


Figure 2.33 Automatic Feature Definition

Both interactive and automatic systems are clearly beneficial, to solid modelling, as they allow the designer to create a solid model solely in terms of its shape, without having to think about 'which feature to use where', as the process of feature

recognition and extraction is generally the task of the computer. However, a recognition algorithm can only recognise features that are similar to the *feature patterns* stored in its database. Therefore, new features (e.g. those created by the designer for an innovative product) may either not be recognised, or interpreted as a collection of known features. Implying that true design intent is not realistically maintained.

2.10.3 Design By Features

As the title suggests, this is a more manual process, consisting of an interface to a library of pre-defined, generic features, including primitives such as holes, rounds, bosses and keyways. The two authoritative forms of design by features shall be described here, Destructive Modelling with Features and Synthesis by Features:

Destructive Modelling with Features - (also termed Destructive or Deforming Solid Geometry) was originally proposed by Arbab (1982) and later by Cutkosky (1988) and Turner (1988). It is essentially a method of removing instanced features from a stock (or base) block. Such a process is akin to part machining operations, for which it was originally devised. Figure 2.34a shows an example of how the L-bracket can be created using this technique.

Synthesis by Features - begins the modelling process with a 'clean sheet', into which a base feature is inserted. Further features are synthesised and either added or subtracted from the base. Figure 2.34b shows how the L-bracket can be created by synthesis.

Of these two systems, Synthesis by Features is more popular amongst commercial systems vendors, as it is more intuitive to established solid modelling approaches. Destructive Modelling with Features is inherently a preferred for CAPP and NC part programming.

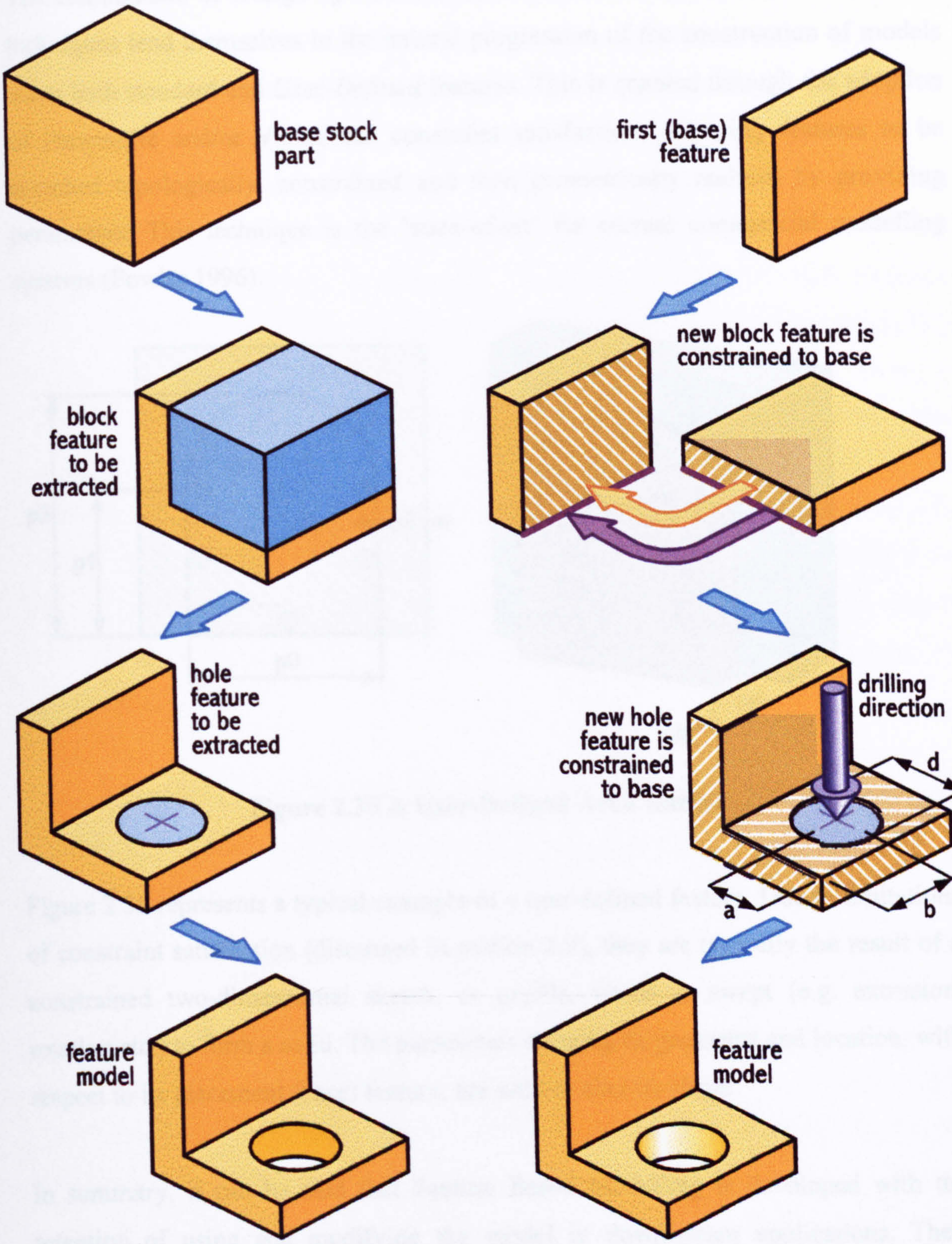


Figure 2.34a Destructive Modelling with features

Figure 2.34b Synthesis by Features

2.10.4 User-Defined Features

The combination of Design by Features and Parametric and Variational Modelling techniques lend themselves to the natural progression of the construction of models using both standard and *User-Defined* features. This is enabled through the adoption of Parametric and/or Variational constraint satisfaction. Allowing features to be sketched topologically, constrained and then geometrically realised by providing parameters. This technique is the 'state-of-art' for current commercial modelling systems (Fowler 1996).

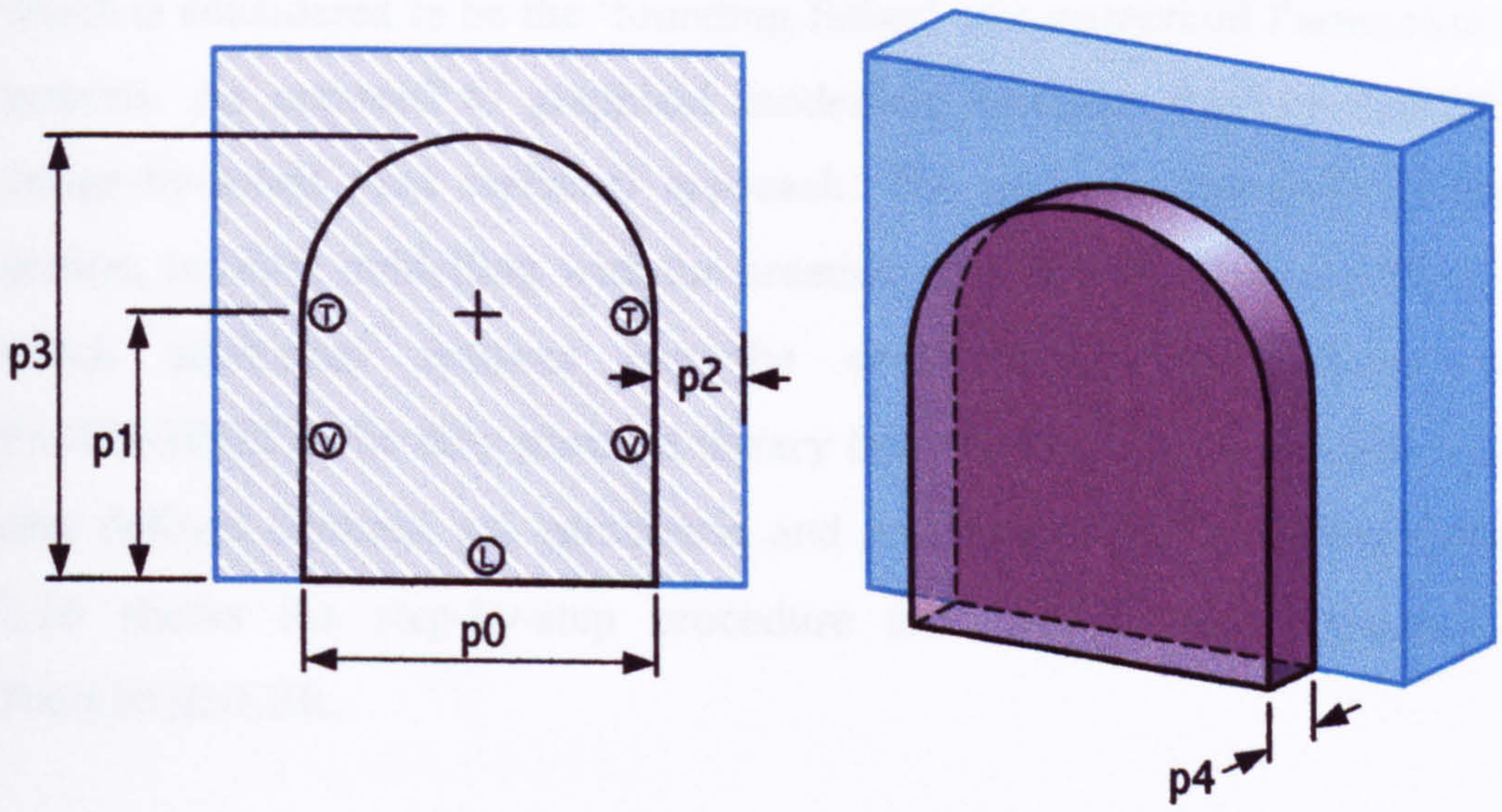


Figure 2.35 A User-Defined Arch feature

Figure 2.35 represents a typical example of a user-defined feature. Due to limitations of constraint satisfaction (discussed in section 2.9), they are typically the result of a constrained two-dimensional sketch, or profile, which is swept (e.g. extrusion, rotation etc.) to form a solid. The parameters defining its geometry and location, with respect to its placement (base) feature, are used to alter its shape.

In summary, it can be said that Feature Based Modelling is developed with the intention of using and modifying the model in downstream applications. Their requirements are introduced as parameters of the feature.

2.11 Commercial Feature Based Modelling Systems

The past decade has seen an increasing acceptance of Feature Based Design and Parametric and Variational modelling techniques into the commercial CAD sector. This section will outline the features of three such modelling systems, covering the top, middle and lower-ground of computer-based mechanical design.

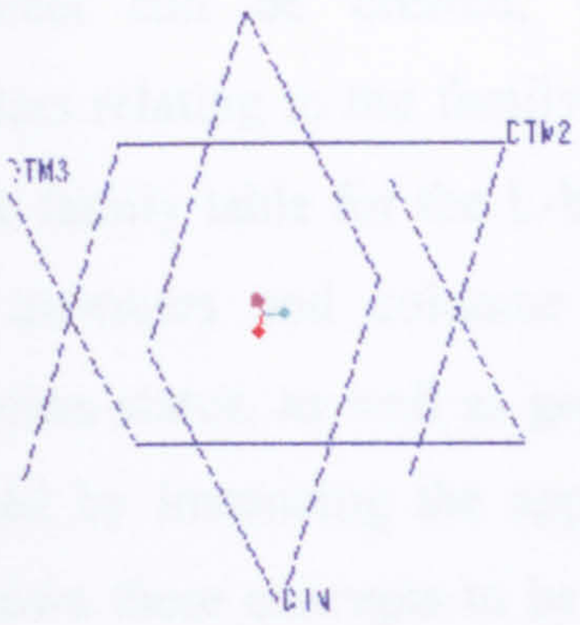
Pro/ENGINEER

At the top end of the market is Parametric Technology's Pro/ENGINEER package, which is considered to be the 'founding father' of commercial Parametric Modelling systems. As opposed to graphical modelling systems, Pro/ENGINEER adopts a design-by-solids (and surfaces) approach. The user, as discussed in the previous section, initiates modelling with the creation of a base feature (usually a datum), to which additional features can be constrained. All features created in Pro/ENGINEER, be they standard library features (such as rounds and chamfers), or user defined features, are parametric and are synthesised to a base feature. Figure 2.36 shows the step-by-step procedure for creating the L-bracket model in Pro/ENGINEER.

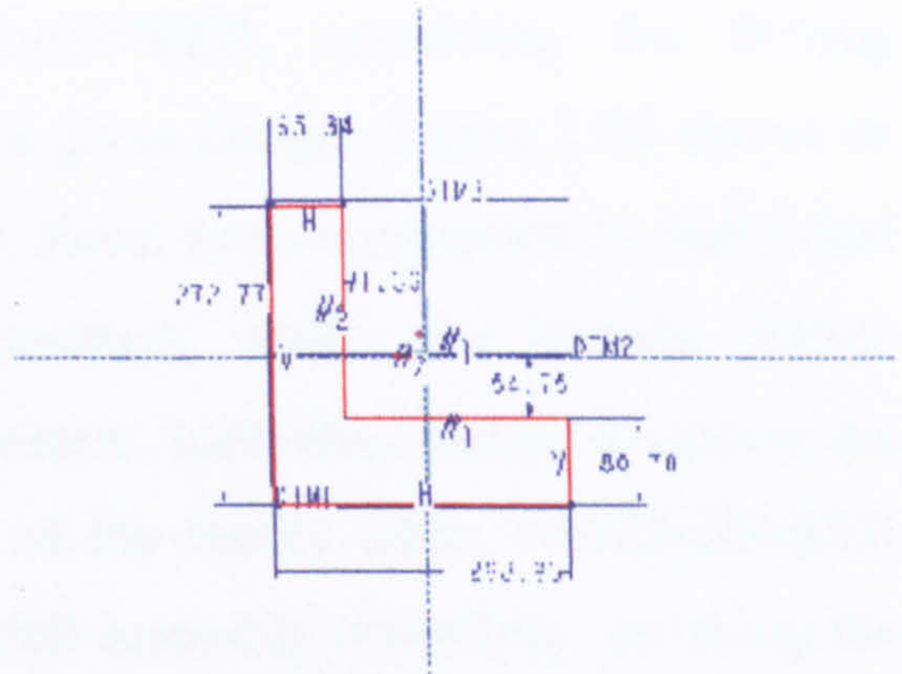
On the modelling side, Pro/ENGINEER has two useful features that are only partially available in other commercial CAD systems, these are:

a) Feature Suppression and
b) The Family Table.

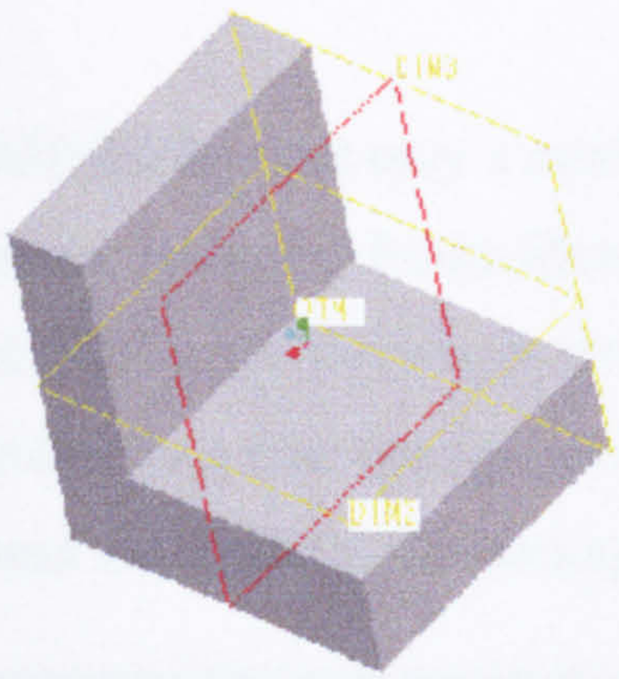
Feature Suppression allows chosen features of a given part to be turned on and off at will. This allows various design alternatives to be present within a single CAD model. An example of this can be to *regenerate* the L-bracket with, or without, its hole feature (figure 2.37).



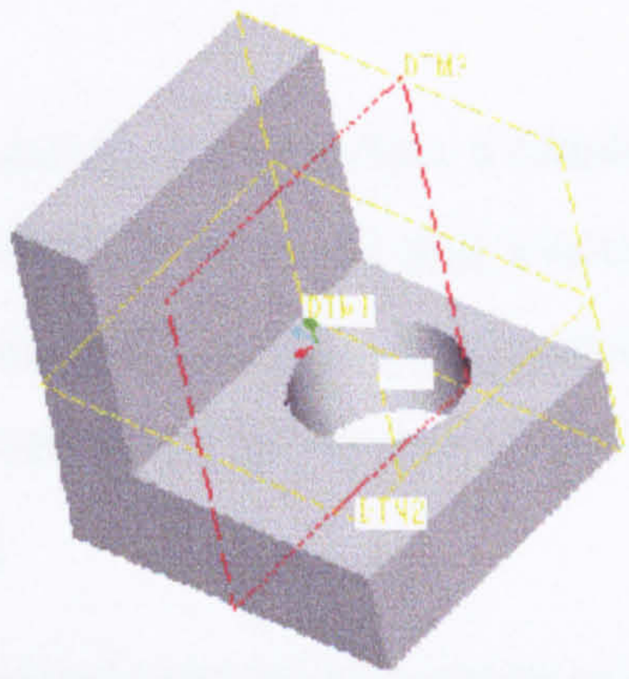
Start with a datum



Sketch the L-Profile

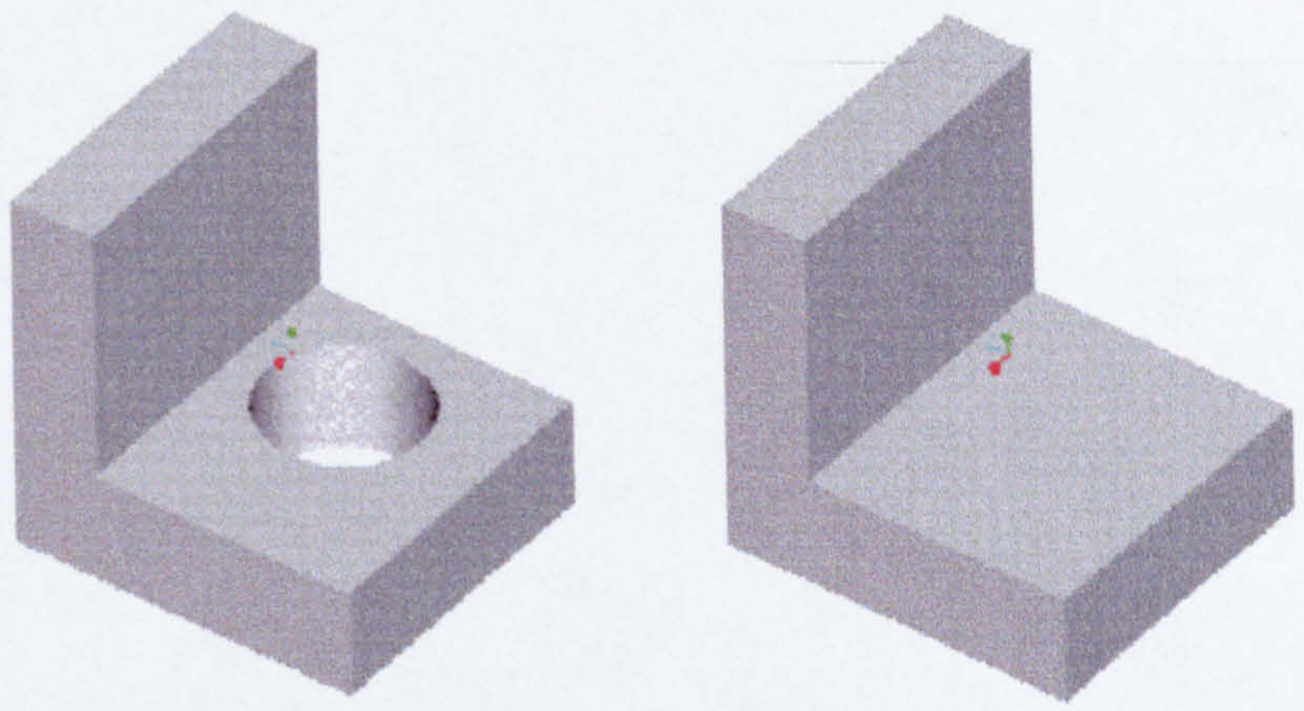


Extrude to form the solid



Create a hole feature

Figure 2.36 – Creating the L-bracket in Pro/ENGINEER



Hole feature Unsuppressed

Hole feature Suppressed

Figure 2.37 – Feature Suppression

The *Family Table* is where the essence of parametric design comes into play. Here, a spreadsheet can be created, within Pro/ENGINEER, containing the driving parameters relating to the family members of a given design. Figure 2.38 shows an example family table for the L-bracket family. Here, rows correspond to individual family members and columns refer to parameters, which can include feature suppression status, as well as geometric parameters. Individual family members are generated by instancing the appropriate row of the family table. Pro/ENGINEER also allows these concepts to be extended to full assembly modelling. Invoking the ability to concisely represent entire product ranges, which can be used for analysis purposes or automatically converted into 2-dimensional manufacturing drawings.

Pro/ENGINEER is not only a modelling-based application. It comprises a number of applications, including Finite Element Analysis (Pro/MECHANICA) and kinematics (Pro/MOTION). On the whole, Pro/ENGINEER is marketed as a complete design to manufacture tool (for detailed design), and has proven itself to be one of the most robust and successful CAD packages of recent years.

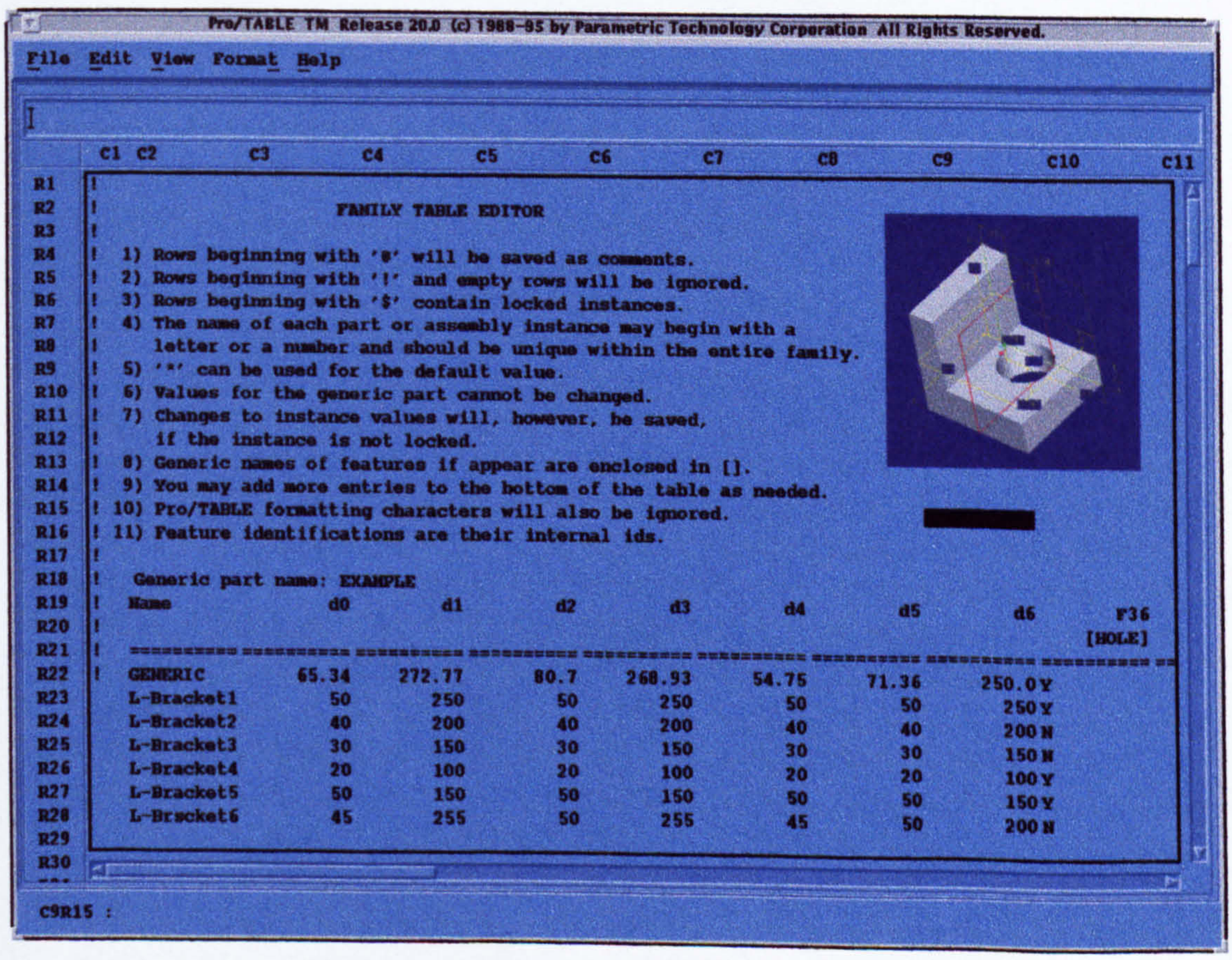


Figure 2.38 – A Family Table in Pro/ENGINEER

Mechanical Desktop

At the 'lower-end' of the market, Autodesk Mechanical Desktop (Autodesk) is a *bolt-on* product to the industry standard drafting package AutoCAD. Mechanical Desktop also adopts the modelling techniques of Feature Based Design, starting with a base feature, to which subsequent features can be added. Again these features are created from either a standard library, or as user defined (or sketched) features. One advantage of Mechanical Desktop is its easy to use user interface, where unlike Pro/ENGINEER, specific pictorial dialogs are used to assist the creation of library features, e.g. countersunk holes and extrusions etc.

However, Mechanical Desktop's roots are not based on a parametric modelling *kernel*, and as such, it is less robust than competing packages. Also, its inability to readily suppress features and parts, and the lack of a structure to represent families of designs, make it a less capable, but significantly less expensive application.

SolidWorks

The Solidworks modeller lies somewhere in-between Pro/ENGINEER and Mechanical Desktop. Although in terms of modelling alone, it is functionally as capable as Pro/ENGINEER. Solidworks also uses Parameter-based variant modelling techniques. Figure 2.39, demonstrates the power of Solidworks with a fully parametric spring example.

One of this package's major strengths is that it is a 'Native' Microsoft Windows application, i.e. unlike applications such as Pro/ENGINEER, it was not 'ported' from the workstation domains of Unix and Silicon Graphics based architectures. It is therefore well suited to the middle-ground of mechanical engineering industry. A further enhancement to Solidworks is its embodiment of an accessible API (Application Programming Interface) . The API can be used directly to automate the Solidworks application from an external source, for example, a database application or a custom coded application.

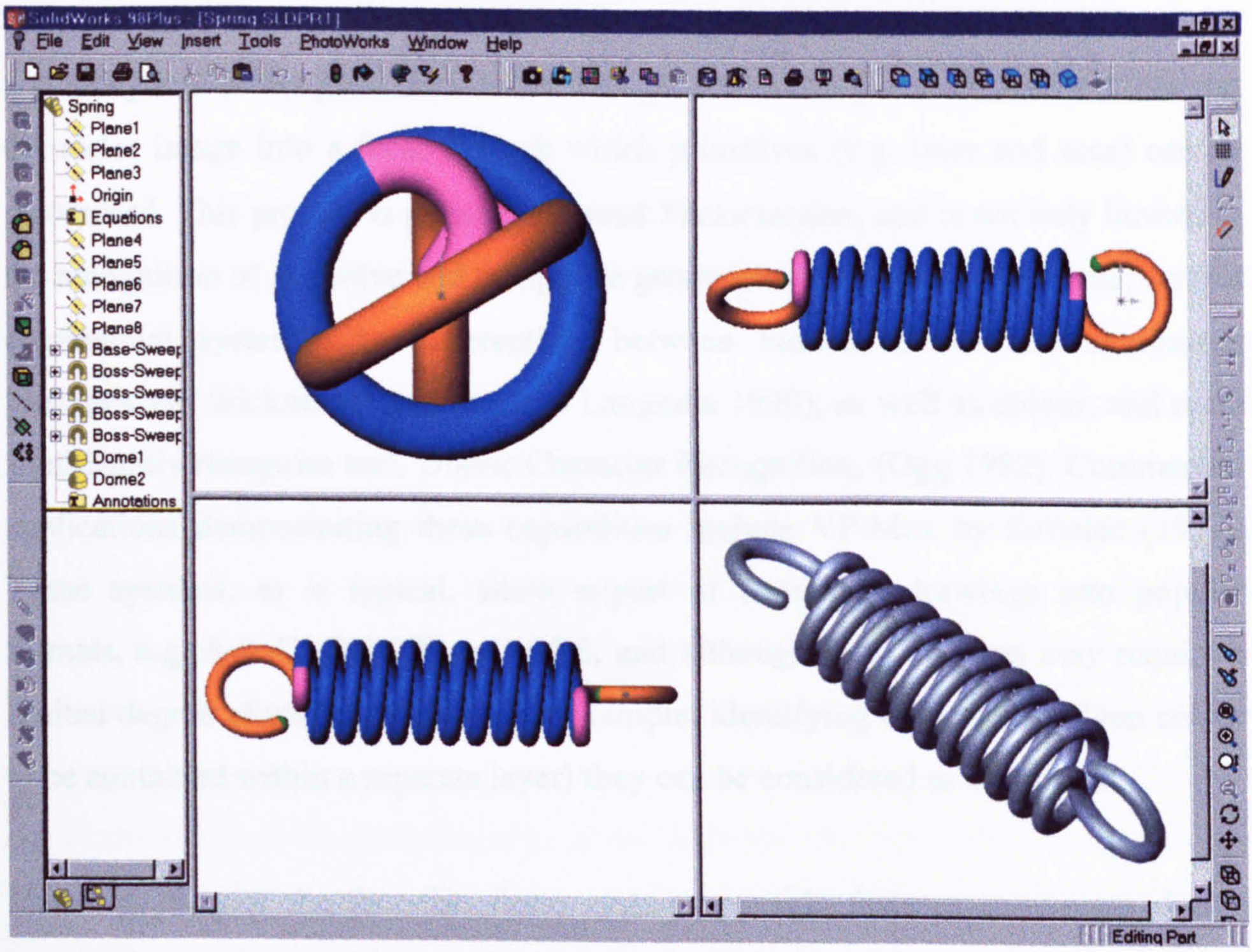


Figure 2.39 – A Parametric Spring in Solidworks

2.12 Working techniques for the Capture of Solid Geometry

In an ideal world, old, manufacturing drawings could be scanned into the computer and automatically transformed into complete, parametric, feature-based representations. If this was the case, then the purpose of this research would be (almost) meaningless. In reality, a significant quantity of research, and hardly any commercial applications for the automatic construction of three-dimensional solid models form their two-dimensional representations exist. Partially automated methods, known as *Interactive Systems*, do exist, though these have been represented commercially for only a few years. The following sections will outline the state-of-art in this field, and determine whether any use of available conversion systems and techniques can be used for this research.

2.12.1 Automatic Capture of Paper Based Manufacturing Drawings

Techniques and algorithms to convert two-dimensional, scanned *line drawings* (termed raster or bitmap images), into coherent CAD-based drawings have been

available for some time. These methods typically invoke a combination of heuristic and analytical techniques, such as the Hough Transform (Leavers 1992) to convert the raster image into a form through which primitives (e.g. lines and arcs) can be recognised. This process is generally termed *Vectorisation*, and is not only limited to the recognition of primitive and composite geometric elements. For example, current commercial systems can differentiate between hidden, centre and continuous linetypes and thickness (Lanasami and Langrana 1990), as well as colour, and more significantly recognise text, Object Character Recognition, (Ogg 1992). Commercial applications demonstrating these capabilities include VP-Max by Softelec (1997). These systems, as is typical, allow export of converted drawings into popular formats, e.g. AutoCAD DXF and IGES, and although these systems may require a limited degree of user-interaction (for example, identifying objects of a given colour to be contained within a separate layer) they can be considered as automatic.

However, despite the fact that these systems provide features to express design intent, they only partially realise our goal of being able to reuse a design's (geometric) model with the latest advances in CAD-based technology. For this, the representative three-dimensional solid model is generally required. The following sections discuss how this can be achieved.

2.12.2 Automatic Conversion to Solid Geometry

Research into the reconstruction / recognition of a three-dimensional object from its two-dimensional projections has been 'in-progress' for over thirty-years, from the stages when 2D sketching and drafting were also in their infancy, and can be defined as follows:

'*Reconstruction* - involves determining the geometric and topological relationship of an object's basic parts, whereas,

Recognition - deals with identifying an object by some form of template matching.'

(Wang 1992)

Both of these fields bear relevance to the meaningful conversion of 2D paper-based drawings to solid CAD models. Reconstruction methods are best suited to forming the solid model (in typically Brep or CSG form), and Recognition methods are more

applicable to the identification of features. The following will discuss the former method (reconstruction), as feature-based recognition methods have already been tackled.

The reconstruction problem can itself be categorised into several areas. Firstly, whether multiple (usually orthogonal) or a single, e.g. plan or perspective, view is given. Multiple views make the process significantly more manageable. Although many researchers have attempted the reconstruction of solids from single views, with some success. Examples of multi-view and single-view projections are given in figure 2.40. The second problem arises when choosing a representation scheme. Of the two established formats, Boundary Representation is perhaps the more naturally suited to this domain, as, like the projection from which it is formed, it is also structured from vertices, edges, curves and faces. On the other hand CSG-based approaches require the reconstruction of solid primitives. The third problem involves determining which of the, possibly many, interpretations is the true representation of the solid.

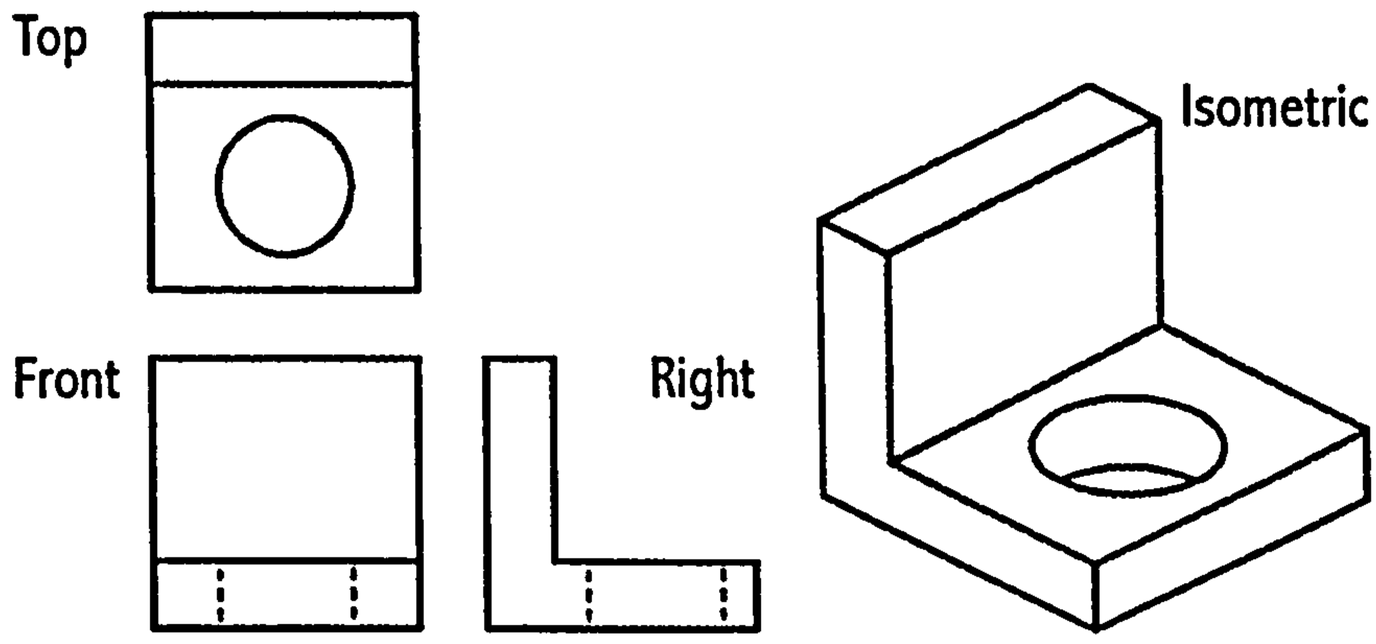


Figure 2.40 Orthogonal Projections (left) and Isometric (auxiliary) View (right)

For the majority of mechanical engineering cases, manufacturing drawings are usually created with more than one view, and are virtually always orthographic. Hence, the discussion of the reconstruction of single view drawings is somewhat irrelevant. For the cases where only a single projection is given, this is treated as a 2½D problem, and is relatively simple to solve, as a lofting exercise. The reconstruction of isometric, or even perspective, views of mechanical designs is unrealistic, as these are typically viewed as being auxiliary, may be inaccurate, and do not portray the 'blind-side' of the object. The following shall therefore discuss

only the reconstruction of solids using multiple view projections. It is also convenient to discuss the techniques developed in terms of their representation schemes, e.g. B-Rep or CSG.

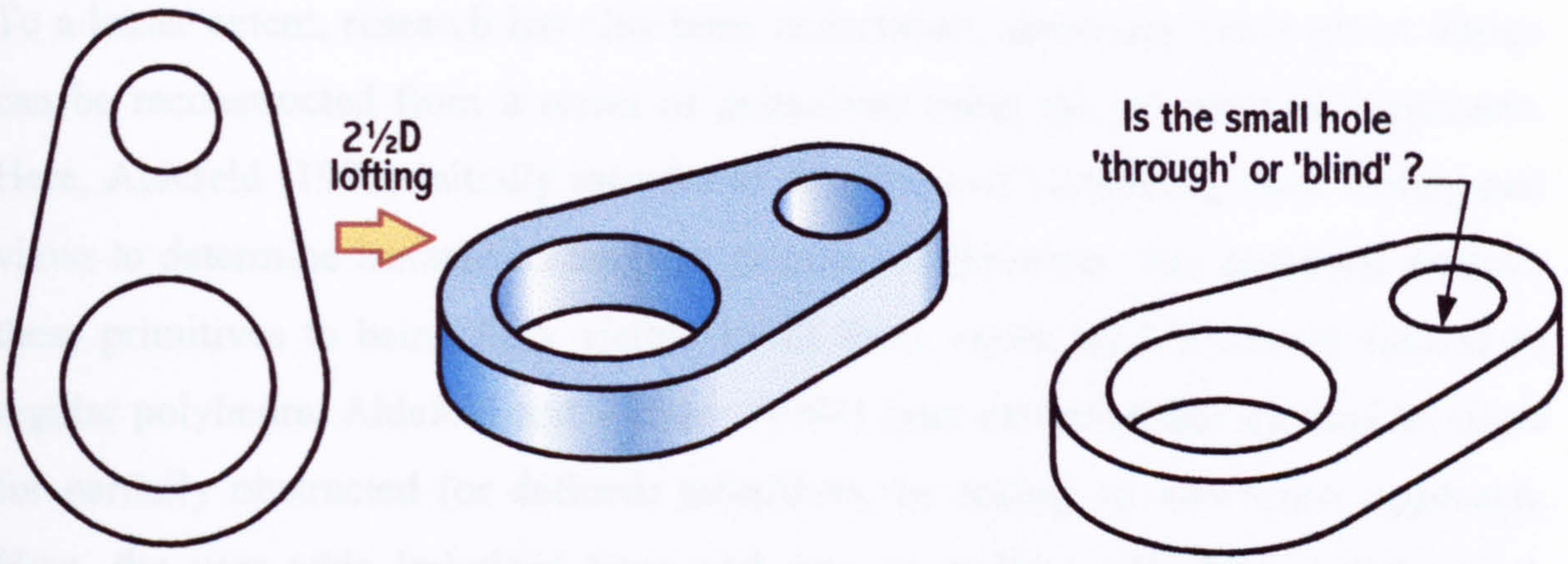


Figure 2.41 A simple 2½D loft (left) and ambiguous isometric view (right)

2.12.3 B-Rep Approaches

These approaches, in general, follow a similar pattern:

1) Transform the 2D vertices from their respective projections into 3D vertices.
2) Join these 3D vertices to generate 3D line segments.
3) Construct planar faces from these line segments.
4) Build 3D solids from the faces.

Initial work by Idesawa (1973) involves a mathematical approach to the problem. He determined that, despite the correspondence between views being known, the reconstruction process would possibly produce what are known as 'ghost figures', e.g. stray points, lines and faces; to which, various elimination criteria are introduced. However, Idesawa's method is only suitable for polyhedral designs. A similar approach was also taken by Wesley and Markowsky (1980), (1981) in their 'Fleshing-Out Wireframes' and 'Fleshing-Out Projections' papers. Although their work is also limited to polyhedra, the elimination of misrepresentative solutions is improved. A major advance from these methods is provided by Sakurai (1983), who introduced rotationally-symmetrical objects into the process. These include spheres, cylinders and cones etc. Further work by Gu et al. (1985) reduced many of the

restrictions, such as the requirement of orthogonal alignment of cylinders, imposed by Sakurai.

2.12.4 CSG-Based Approaches

To a lesser extent, research has also been undertaken, assuming that a given design can be reconstructed from a series of primitives using the set theoretic approach. Here, Aldefeld (1983) initially introduced a method of comparing three orthogonal views to determine isolated rectangular primitives. However, this approach restricts these primitives to being fully visible in all three views, and is clearly limited to regular polyhedra. Aldefeld and Richter (1984) later extended this method to allow for partially obstructed (or defined) primitives, by taking an interactive approach. Here, the user adds 'missing' lines and arcs to realise individual primitives. A commercial implementation of an interactive method is the 'Make-IT 3D' package, EMT (1998). Ho (1986) further extends this work by providing a more intuitive CAD-based approach, where the user identifies primitives from a set of orthogonal views and identifies their sense, i.e. by addition or difference. This method significantly reduces the time required to extract partially visible primitives.

2.12.5 Summary of Multi-View Reconstruction Approaches

Both B-Rep and CSG approaches are only partial attempts for the successful conversion of 2D projections to a solid model. Their current limitations are therefore listed below:

a) **lack of recognising 'real-world' designs - many designs contain complex curved surfaces and obscured views, only identifiable by cross-sections and hidden line auxiliary views. The reviewed systems cannot cope with these designs.**

b) **Inability to capture design intent – manufacturing drawings also contain constructive information relating to such areas as dimensions, tolerances and even the inclusion of features that cannot be seen, e.g. small fillets. These are ignored by these approaches.**

It can therefore be concluded that automated reconstruction techniques, in their current state of development, are not suitable for the modelling of real, complex engineering products.

2.13 State of the Art - Feature-Based Semi-Automated Methods

The semi-automated methods for capturing detailed designs are an attempt to incorporate the advantages of retaining a high level on design intent, whilst using techniques, such as Parametric and Variational Design and Feature Based Design, to automate the generation (or instancing) of similar designs, i.e. its variants. The two principal 'State of the Art' techniques for the semi-automated capture of past (and the creation of new) designs, are the Generative (sometimes called Procedural) and Variant Design Methods.

2.13.1 Generative Method

This method adopts a procedural technique to create a parametric model for a given design. The Generative Model is essentially a sequential list of events, or instructions, that represent the design's construction process. Real numbers, representing geometry, are replaced with variables, by editing this data structure. Other parameters, not necessarily relating to geometry can also be added. Individual instances can then be generated by declaring values for these variables and parameters, and then re-executing the procedural data structure.

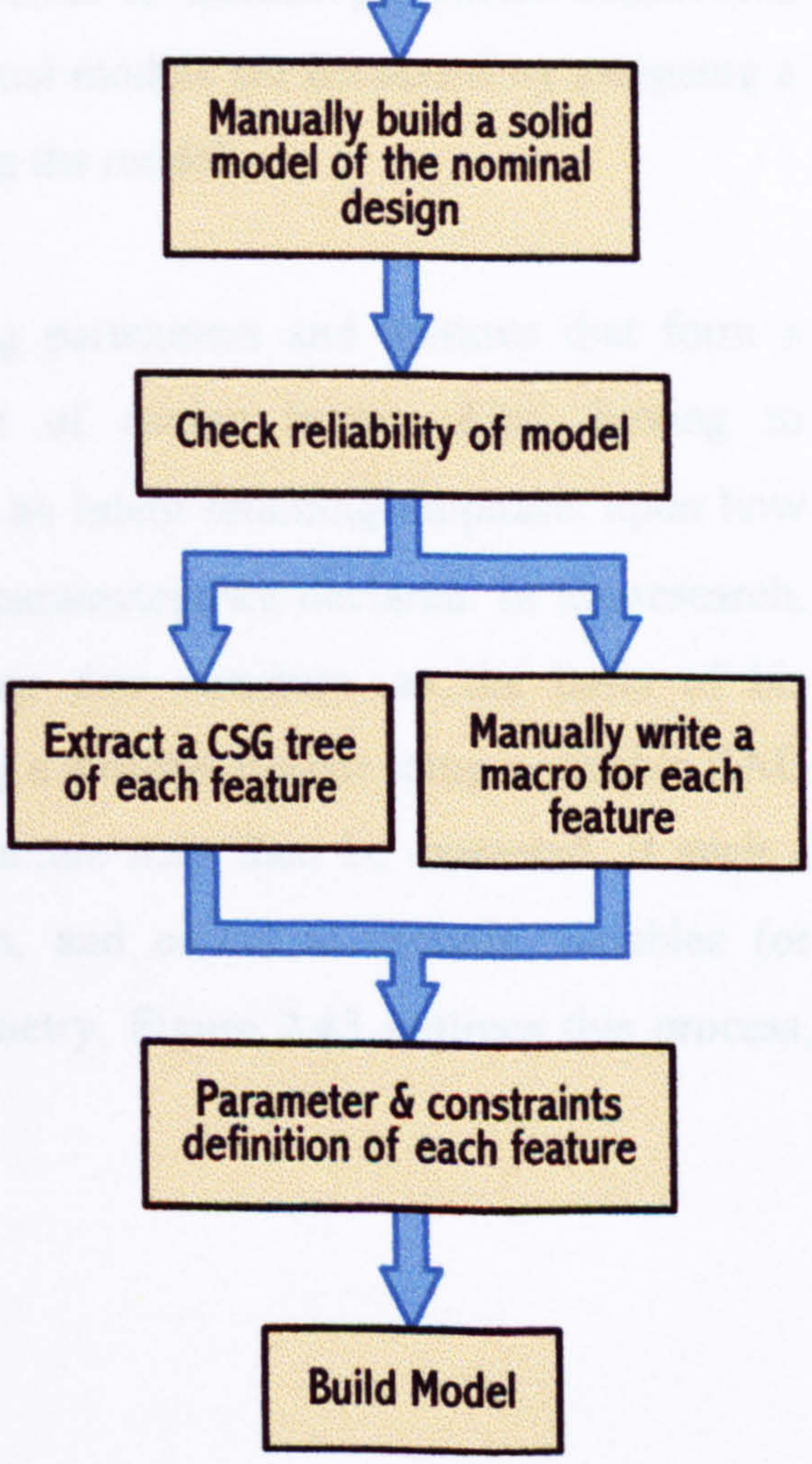


Figure 2.42 The Generative Methodology (Shahin)

Shahin (1996) encompasses the generative method in his PhD thesis, outlining a methodology to create a series of similar solid models from a single Generative Model, with a goal towards design

optimisation. Figure 2.42 outlines the relevant sections of this methodology, which is categorised by the following three objectives:

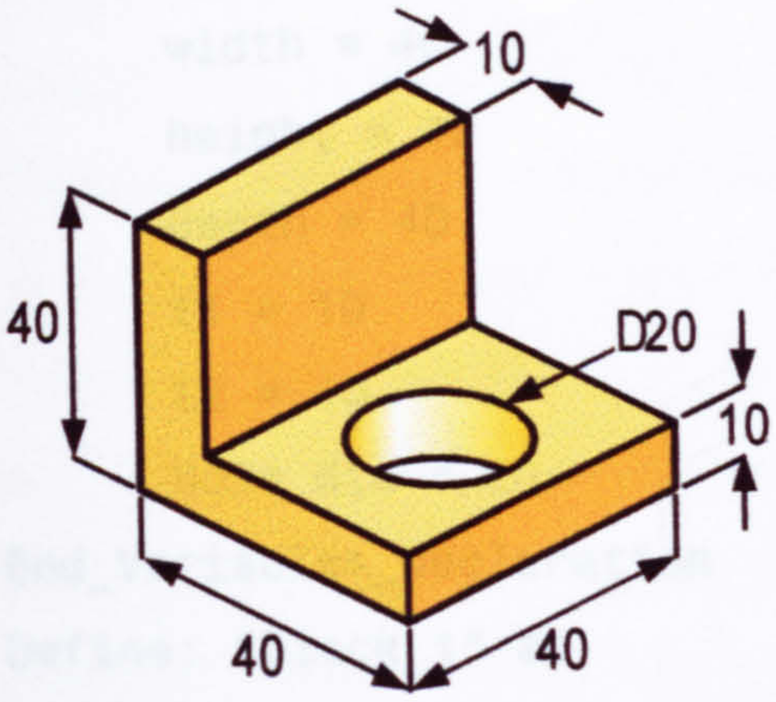
Objective 1 - defines various elements of the design that are related to *design intent*, e.g. parameters, features and constraints. Note that the user of this system is required to manually define the geometry, constraints and relations for geometric elements and features.

Objective 2 - is concerned with the creation of a *reliable model*. The nominal solid model should be the best possible representation of all instances that are to be generated.

Objective 3 - describes a scheme to explicitly model each feature of the nominal model by, either writing an application-specific macro, or by extracting its representative data structure. This is then edited to include parameter definitions, constraints and relationships. Finally individual models are instanced by assigning a new set of parameter values and re-generating the model.

Clearly the process of manually identifying parameters and features that form a given design is a distinct representation of design intent. Also, having to mathematically define these features places an intent retaining emphasis upon how their related elements will react when new parameters are declared. In his research, Shahin makes use of a hybrid CSG/B-Rep data structure, as the basis of his Generative model. This is formed by creating a nominal design using a suitable CAD modelling application. The hybrid data-structure may then be extracted, if such a feature is available within the application, and edited to include variables (or parameters) in the place of numerical geometry. Figure 2.43 outlines this process, using the L-Bracket example.

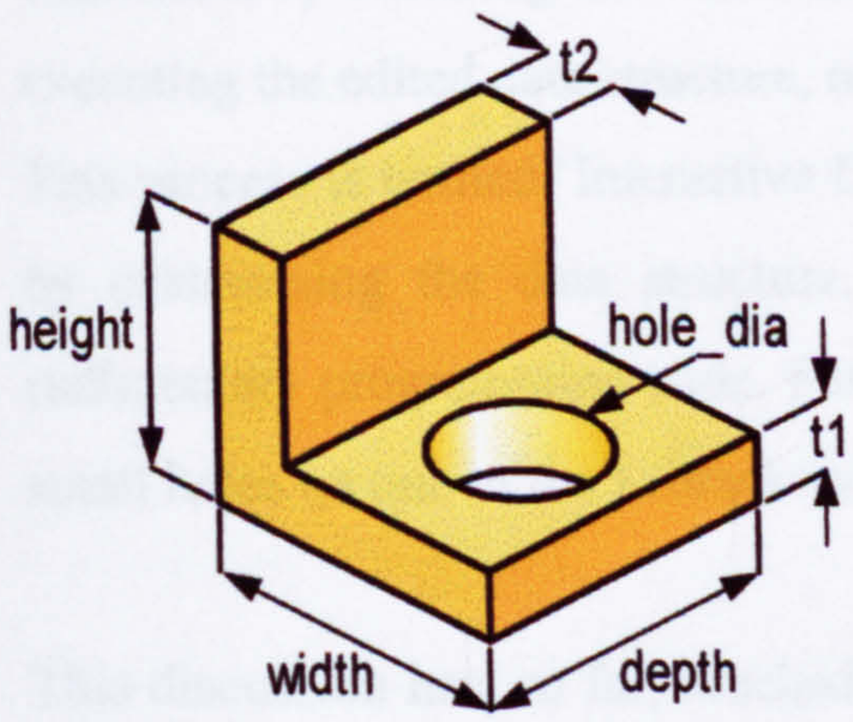
Nominal Model



Extracted CSG Tree

```
Define: “entity_1” as
      Create_Solid_Block: (0,0,0) (40,10,40)
End_definition_of “entity_1”
Define: “entity_2” as
      Create_Solid_Block: (0,0,0) (10,40,40)
End_definition_of “entity_2”
Define: “entity_3” as
      Union: “entity_1” “entity_2”
End_definition_of “entity_3”
Redefine “entity_3” as
      Subtract: “entity_3” from
      Create_Through_Hole: (25,10,20) (0,-1,0) 20
End_definition_of “entity_3”
```

Parameter Definition



Design Intent

Place Hole centrally on the resultant face of the ‘L-Bracket’

Editing of Data Structure

```
Declare_Variables:
        width = 40
        height = 40
        depth = 40
        t1 = 10
        t2 = 10
        Hole_dia = 20
End_Variables_declaration
Define: "block_1" as
        Create_Solid_Block: (0,0,0) (width,t1,depth)
End_definition_of "block_1"
Define: "block_2" as
        Create_Solid_Block: (0,0,0) (t2,height,depth)
End_definition_of "block_2"
Define: "L_bracket" as
        Union: "block_1" "block_2"
End_definition_of "L_bracket"
Redefine "L_bracket" as
        Subtract: "L_bracket" from
        Create_Through_Hole:  (width-(width-t2)/2,t1,depth/2)  (0,-1,0)
        hole_dia
End_definition_of "L_bracket"
```

Figure 2.43 - Example of the Generative Method for the L-Bracket

Therefore, by declaring the variables: width, height etc. with different values and re-executing the edited data structure, other instances of the L-bracket can be generated. This process is termed 'Interactive Design by Features' and can be further enhanced by customising the data structure, which is essentially a program listing, with rudimentary programming code. For example, by adding a loop to create a series of small holes on one of the L-brackets blocks.

This discussion has, so far, touched only on the advantages of this method. It does, however, impose a number of restrictions with regard to its implementation. Firstly, it requires a degree of mathematical, geometric knowledge and programming skills. Both of these qualities may not be available within a typical SME, that is only just

beginning to adopt CAD. A further disadvantage of the generative method is that the construction of its models is time consuming, especially for models with complex curved parts, as these will require an exact mathematical definition to be provided by the user. Furthermore, this method is procedural, implying that the model must be re-generated from scratch every time a single parameter is modified. For large, complex, multi-part models this can also be time consuming. **In conclusion, the Generative Method is well suited to geometrically modelling past designs, including that of design families under a single, adaptive model. It does, however, impose a heavy resource burden on the designer.**

2.13.2 Variant Method

Although similar in operation, the Variant approach to storing solid geometric models differs primarily in the construction of its models. Whereas the generative approach involves the often tedious operation of editing a complex data structure to enable parameterisation, the *Variant Method* makes use of Parametric and Variant Modelling techniques (see section 2.9) and Feature Based Design, in particular User Designed Features (section 2.10.5), to interactively *draft* a geometric model. It requires virtually no complex mathematical and programming operations, and is typically implemented via an efficient and familiar user-interface (Kurland 1996). Despite the difference in terminology, perhaps the most well known commercial example of this technique is the 'Parametric Modeller', Pro/ENGINEER, which was pioneered in 1990. More recently other applications vendors have adopted this technique, including Autodesk, with 'Mechanical Desktop' as an extension to AutoCAD, and 'SolidWorks'.

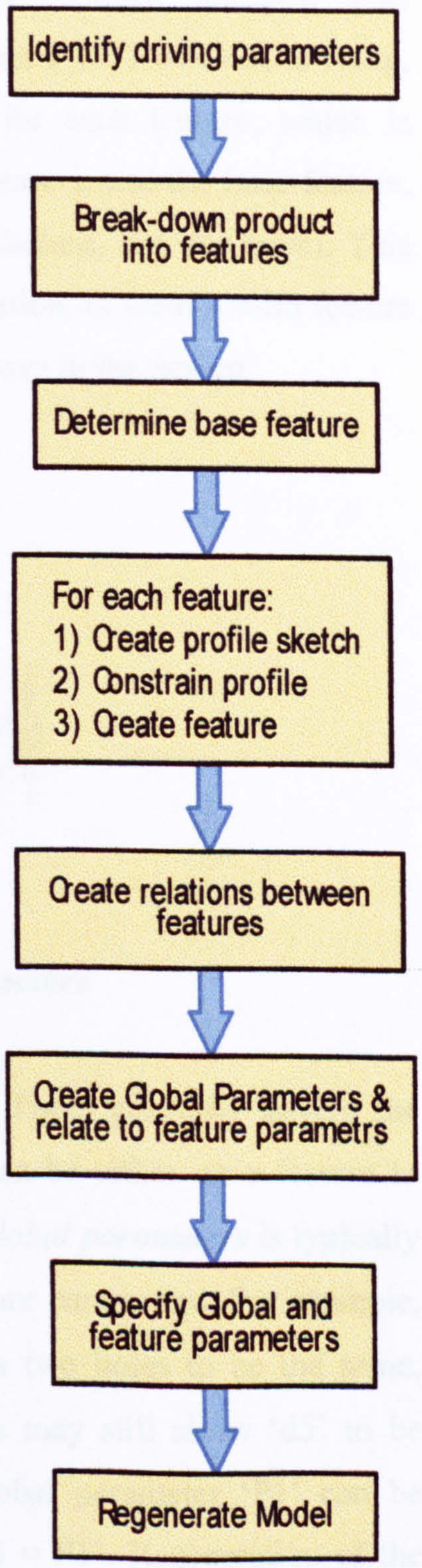


Figure 2.44 The Variant Method

The process of creating a variant model is initially similar to that of the generative model. Where, to begin with, the driving design parameters and features, are identified.

The majority of current modelling systems work with a 'Synthesis by Features' approach, where features are constructed in a hierarchical fashion, thereby requiring the creation of a base feature. Here, features are created by either using predefined, library features (primitives), or by generating User Defined features. This construction process has already been described in section 2.10. However, to recap, it involves the creation of a 2D sketch (or profile) for each feature, which is parametrically dimensioned and constrained. (If this feature is not the base feature, then its profile must also be constrained to its parent feature, e.g. the base). This profile is then transformed, typically by parametric extrusion, to form a solid feature model. And the process is repeated for all identified features in the design.

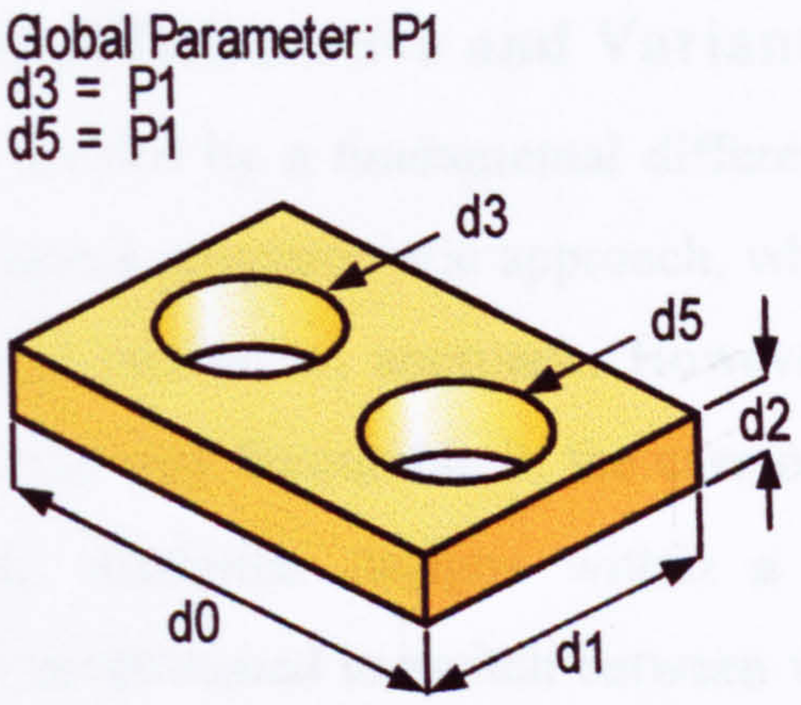


Figure 2.45 Use of Global Parameters

Finally, relations between features can be established. These generally govern the control of a given feature's driving parameters, and can be either on a feature to feature basis, or defined globally. In this case a set of *global parameters* is typically created to oversee the declaration of (subordinate) feature parameters. For example, the block, of figure 2.45, requires the diameters of its two holes to be the same. Setting the relation 'd3 = d5' will not suffice, as this may still allow 'd5' to be modified independently. Therefore the use of the global parameter 'P1' can be defined through relations as: 'let: d3 = P1' and 'let d5 = P1'. Regeneration of the models relations and constraints will always result in d3 and d5 being equal to P1's

declared value. Variants of this model can now be instanced by modifying features and global parameters, and re-solving the models constraint set (regeneration).

A further feature, that is typical in many variant design systems, is the ability to momentarily hide, or *Suppress*, various child features, and *Resume* these features when desired.

Variant based modelling systems are, on the whole much simpler to use than their generative counterparts. They also require less human resources to create a 'parametric', or adaptive, model for a given design. Furthermore, such systems based (even partially) on *Flexible Constraint Satisfaction* techniques (see 2.9), allow for faster model regeneration, as here only the modified and directly related features and entities are updated. However, innovative application methods have to be developed to exploit this power.

2.13.3 A Comparison of Generative and Variant Design Methods

These two methods are divided by a fundamental difference in their creation. The Generative Method employs a programmatic approach, whereas the Variant Method provides a more naturally, concurrent approach. However, generative models are highly customisable. This is very favourable in the case of attempting to combine a number of topologically dissimilar designs within a single model. Here the generative model can be programmed to switch between various features depending upon which individual design is required. Trying to attempt this problem with the variant method is difficult, as the variant method inherently 'varies' a given model, and cannot invoke and respond to yes/no decisions, by itself.

2.14 Essential Findings from the Literature Survey

2.14.1 Function Means Tree

Unlike the Parts Tree and Function Family Tree, the Function Means Tree relates both function and means (parts) under a single data structure. In particular, it directly relates a given function to it realising means, which is ideal for rapid component retrieval. Furthermore, a core consideration of this research is to simplify the process of storing past design cases. The Function Means Tree is a simple, clear and straight-forward structure to create for each past design case, as its elements are easy to identify (a means is a part or subsystem name, to which its function can be easily derived) and input. In comparison to the chart based methods, such as Design Function Deployment, it is less cumbersome, and does not overburden the designer too heavily.

2.14.2 Annotated Sketches

If available, sketches are highly regarded as a medium to express design intent, and demonstrate 'how things work'. Combined with suitable annotation (text), a given sketch can be stored along side its related function-means pair in the Function Means Tree.

2.14.3 Variant CAD Model

The traditional, static forms of geometric modelling do not allow existing CAD-based models to be easily adapted and modified. However, dynamic systems, such as the combined Parametric and Feature-Based modellers that are commercially available facilitate this requirement to some degree. Of the existing methods that can be adopted to transform an organisations legacy manufacturing drawings into solid CAD models, the Semi-Automated methods (see section 2.12) are most relevant. Automatic and Interactive Recognition techniques for 2D Projections (section 2.11) are still in their infancy, and have been argued to be deficient in representing real, complex engineering designs. These methods also do not wholly express the degree of design intent present in a typical manufacturing drawing, as they only represent elements of the drawing(s) that can be recognised from a pre-defined database.

Of the reviewed Semi-Automated design representation methods, both the Generative and Variant methods incorporate feature-based and parametric techniques, which allow for rapid design modifications. It is apparent that the Generative method is best suited to the evolution, or synthesis, of complex, innovative designs. In contrast the Variant method is better applied to a more well defined design scenario. In terms of actual modelling, the Variant methods is much simpler. Therefore, it follows that the Variant method is more readily applicable to the reconstruction of existing, pre-defined engineering designs, and is chosen here to represent the Detailed Design.

These findings were used in the development of the methodology for storing a family of detailed designs.

Chapter 3

Generic Methodology

Overview

In chapter 1 a number of objectives for this research were defined. These are:

1) To determine suitable data-structures to store the Solution Concept, Embodiment and Detailed stages of the design process,
2) To create a Methodology to store existing design families for efficient reuse, and,
3) Implement the methodology as a Software Application.

The previous chapter discussed the relevant data-structures to represent these stages of the design process. This chapter will discuss the proposal of two novel concepts, followed by a Generic Methodology, to realise these objectives. This will be subsequently illustrated using a simplified propeller-shaft example.

3.1 Data Structures

Before proposing a suitable data structure to effectively store past designs, it is useful to refresh, or identify, the underlying requirements of this research. Firstly, the chosen method(s) should represent a given design concisely, but with enough descriptive meaning, so that whole design, or parts of it can be retrieved by either name or descriptive function. This information should also be detailed enough to satisfactorily express the designer's original intent, so that new designers can understand and learn from past design cases. Secondly, designs should be stored in a manner that facilitates easy modification, which will allow existing designs to be readily modified to suit a new scenario of requirements.

To this end, the following design methods have been chosen to represent solution concept, embodiment and detailed designs:

a) Function Means Tree
b) Variant CAD Model

3.2 The Two Novel Concepts

The Generic Methodology proposed here, builds on two novel concepts, these are:

1) the Hybrid Function Means / Parts Tree and
2) the Variant Master Model.

The Hybrid Function Means / Parts Tree accommodates the conceptual and embodiment stages of the design process, while the Variant Master Model accounts for detailed design. These concepts are further discussed below.

3.2.1 The Hybrid Function Means / Parts Tree

This section proposes a combination of both the function oriented Function Means Tree, and the assembly oriented Parts Tree. The union of these two structures allows the designer to build and view a structure to represent the conceptual and embodiment stages of design according to their individual context and preference. For example, when synthesising a new (or viewing an old) design, it is preferable to design by an evolution of functions (see the Function Family Tree, section 2.3.1). Whereas, when converting an existing design into a CAD based model, it is easier to structure this model in terms of its parts and order of assembly.

Figures 3.1a and b respectively, show a generalised Function Family Tree and Parts Tree for a simple product. The relationships between functions and means (parts) can be represented as two lists of information, for both functions and parts, as in figure 3.2. This data structure is comprised of two linked-lists representing the structure of both trees, by storing the parent node indices (ID's) for each child node. Relationships between the lists (indicated by straight lines in figure 3.2) are also stored as the node ID's of the corresponding list. For example, the 'Tertiary Function A' (node/ID 2 in the Function Family Tree) is realised by 'Part B' (node/ID 3 in the Parts Tree).

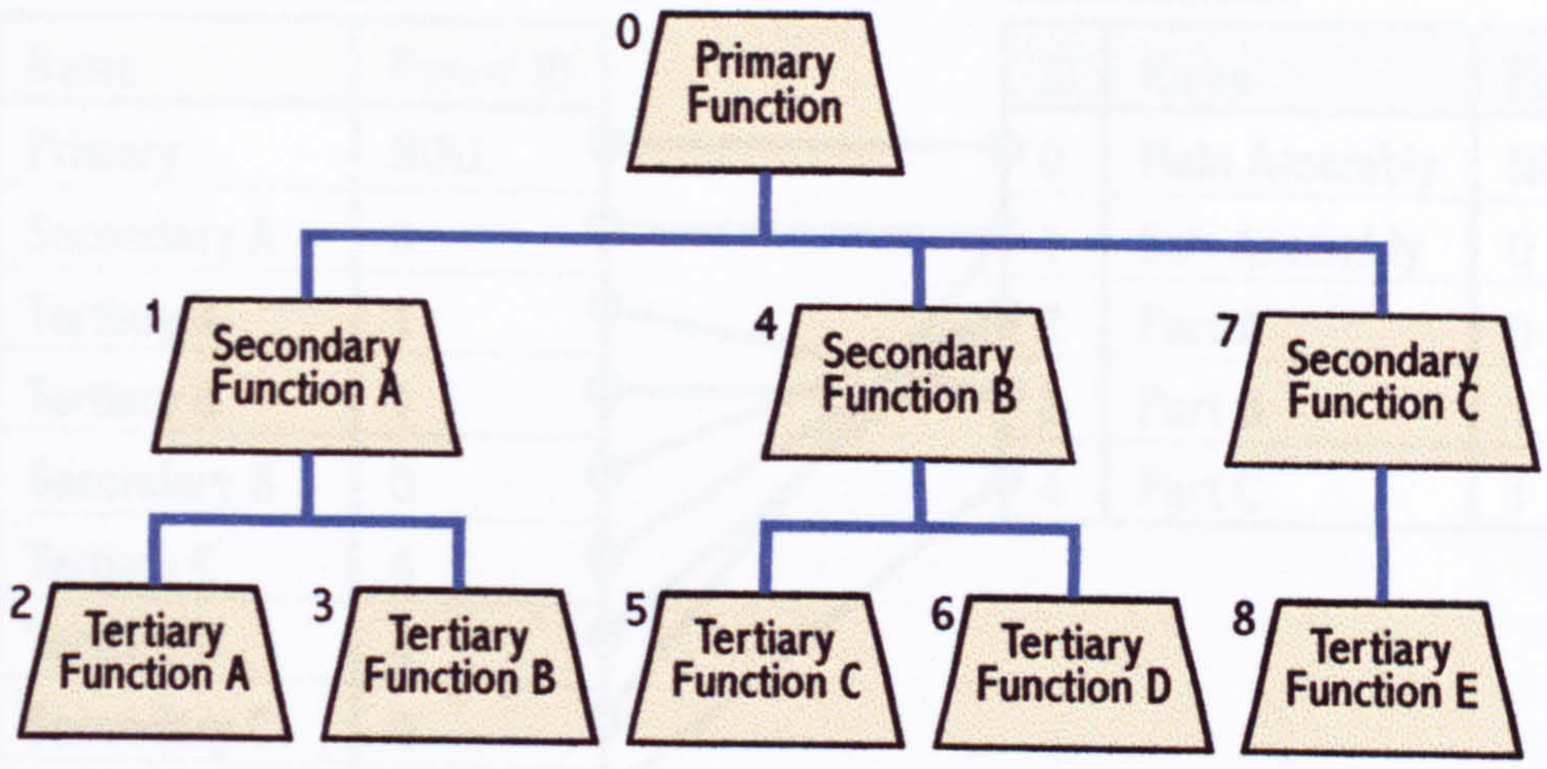


Figure 3.1a A Generalised Function Family Tree

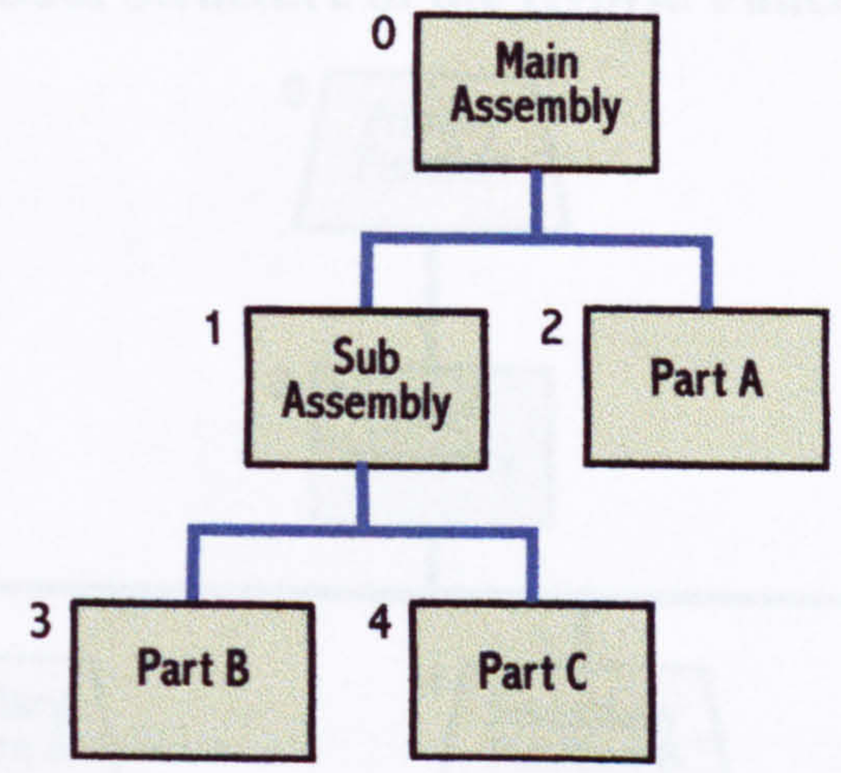


Figure 3.1b A Generalised Parts Tree

Figures 3.3 and 3.4 show the resulting Function oriented and Parts (assembly) oriented representations of the Hybrid Function Means/Parts Tree data structure, respectively.

Hybrid Function Means / Parts Tree

Function Family Tree List

ID	Name	Parent ID
0	Primary	NULL
1	Secondary A	0
2	Tertiary A	1
3	Tertiary B	1
4	Secondary B	0
5	Tertiary C	4
6	Tertiary D	4
7	Secondary C	0
8	Tertiary E	7

Parts Tree List

ID	Name	Parent ID
0	Main Assembly	NULL
1	Sub-Assembly	0
2	Part A	0
3	Part B	1
4	Part C	1

Figure 3.2 Schematic Data Structure of the Hybrid Function Means / Parts Tree

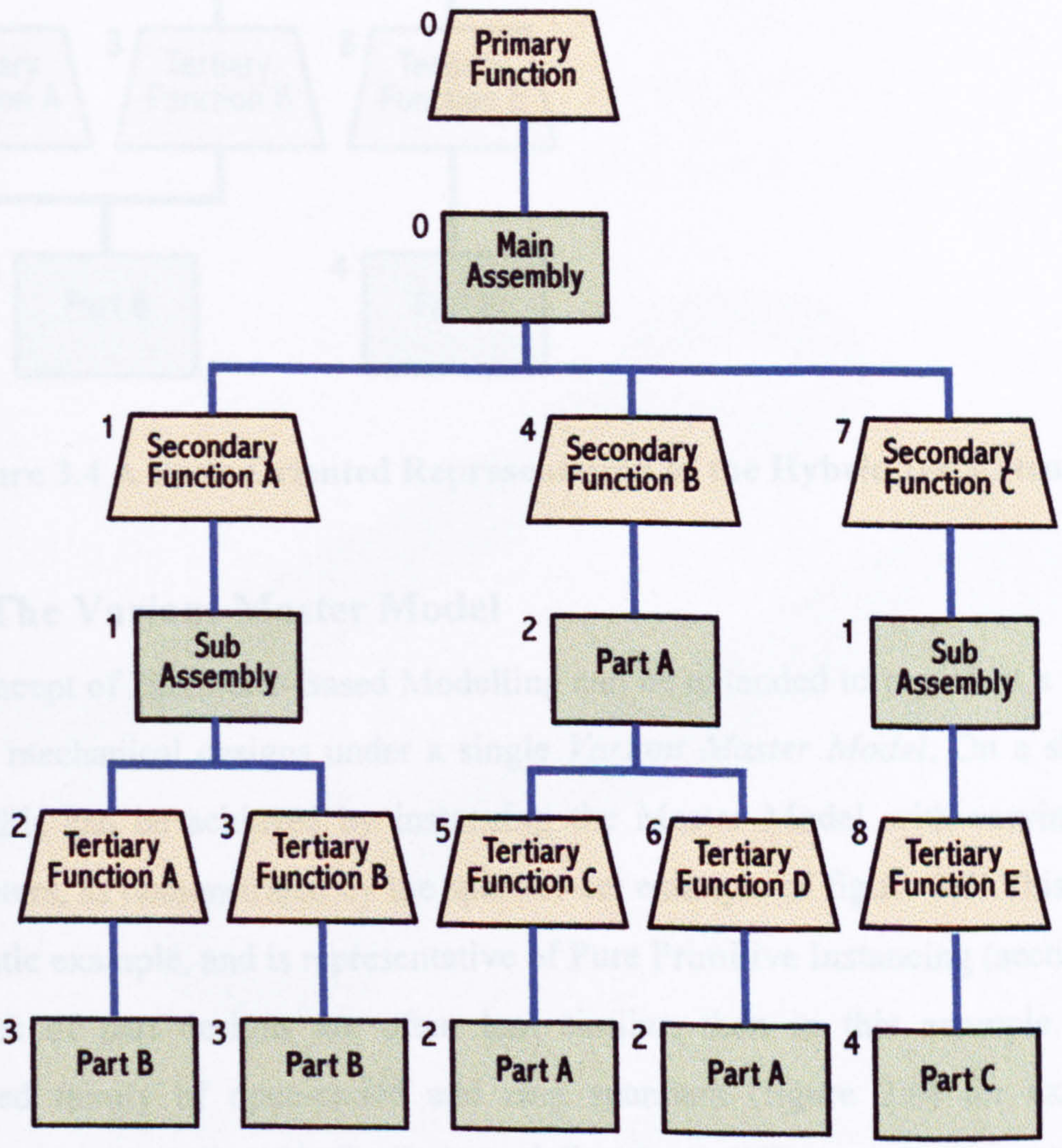


Figure 3.3 A Function Oriented Representation of the Hybrid Data Structure

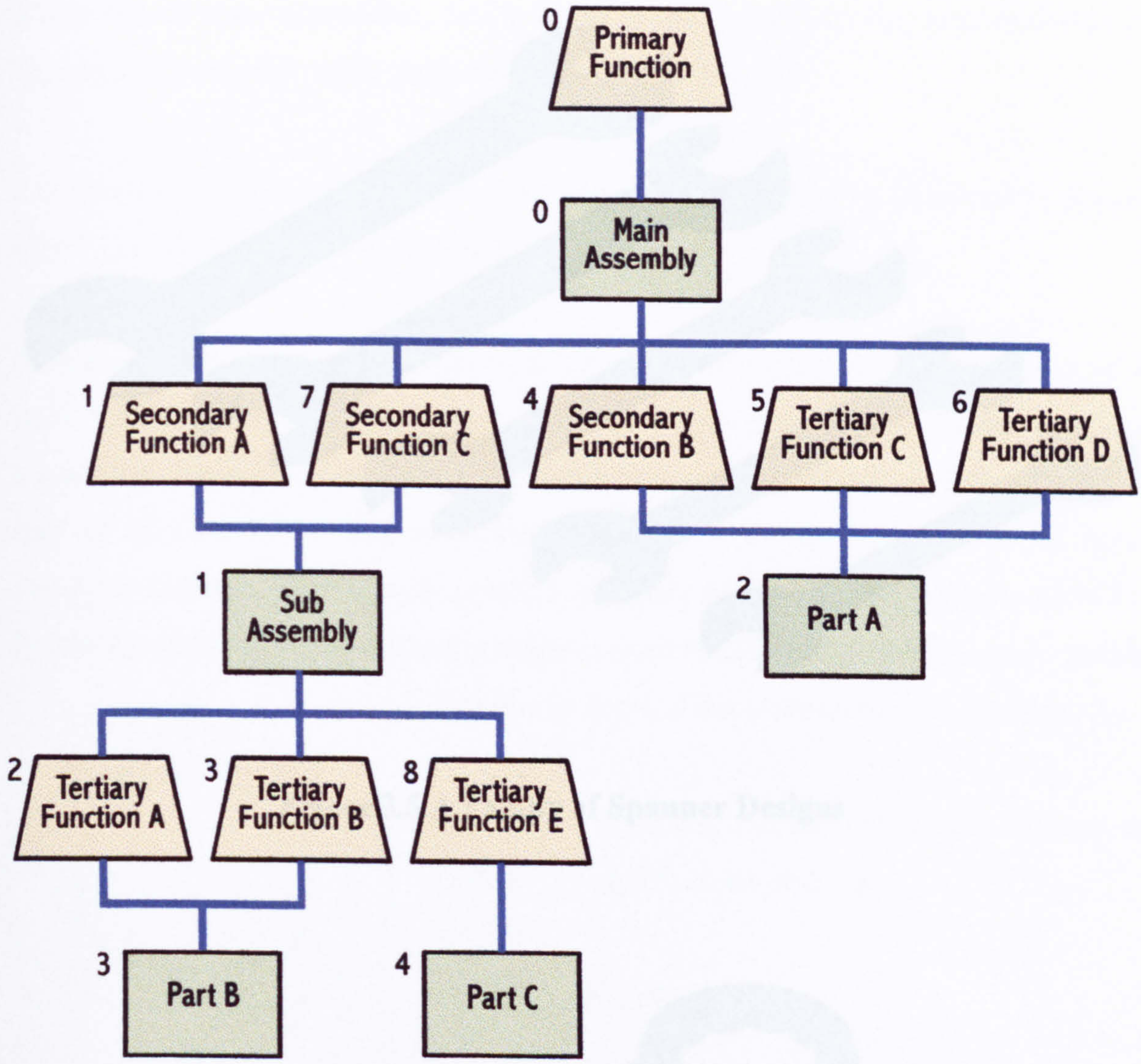


Figure 3.4 A Parts Oriented Representation of the Hybrid Data Structure

3.2.2 The Variant Master Model

The concept of Parameter-Based Modelling can be extended to represent a family of similar mechanical designs under a single *Variant Master Model*. On a single-part basis, this can be achieved by instancing the Master Model with varying sets of parameters, as demonstrated by the spanner-set example of figure 3.5. This is a very simplistic example, and is representative of Pure Primitive Instancing (section 2.8.3). Families of part designs are often less similar, than in this example. Take an extended family of open-ended and ring spanners (figure 3.6) for example. A solution to representing this family is to define these differences as separate features, all contained within the single Master Model, and, depending upon which design case is required, by turning selected features on and off.

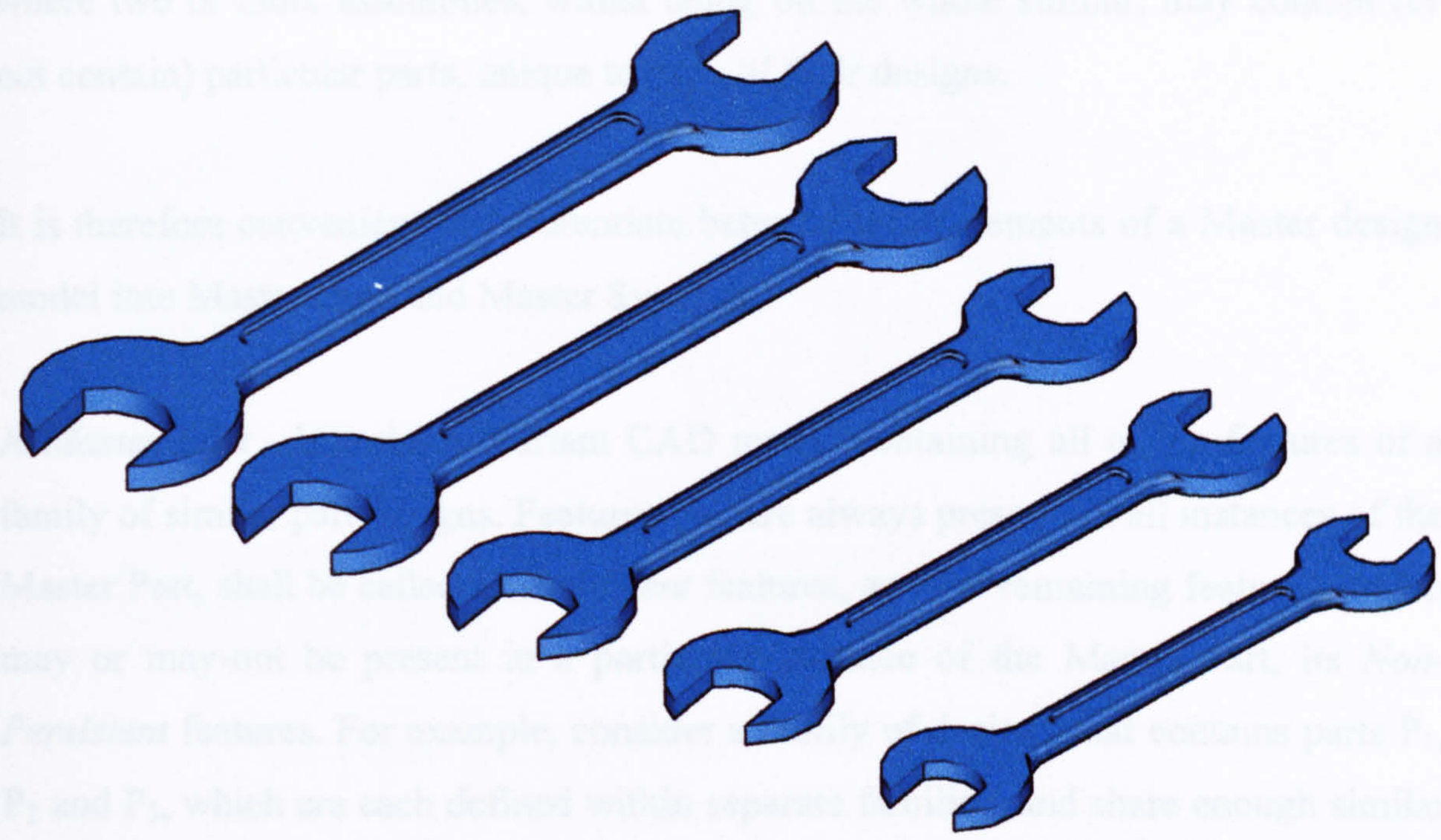


Figure 3.5 A Family of Spanner Designs

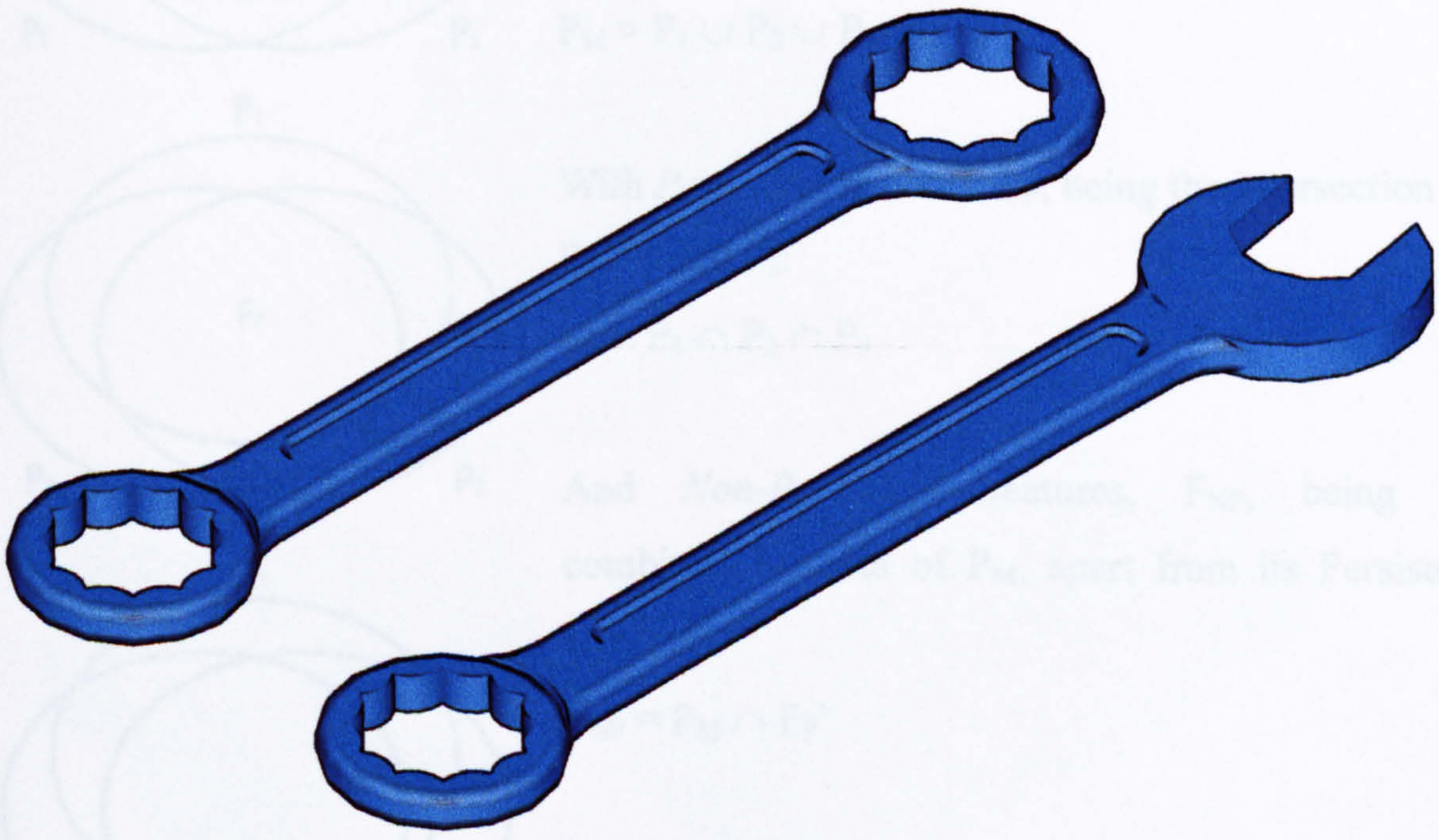
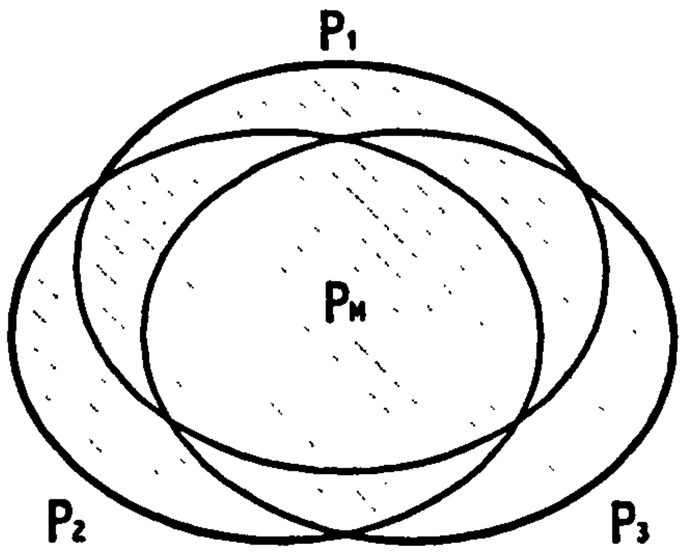


Figure 3.6 Members of a more Complex Spanner Family

Assemblies of parts share similar characteristics to those of individual part designs, where two or more assemblies, whilst being on the whole similar, may contain (or not contain) particular parts, unique to each of their designs.

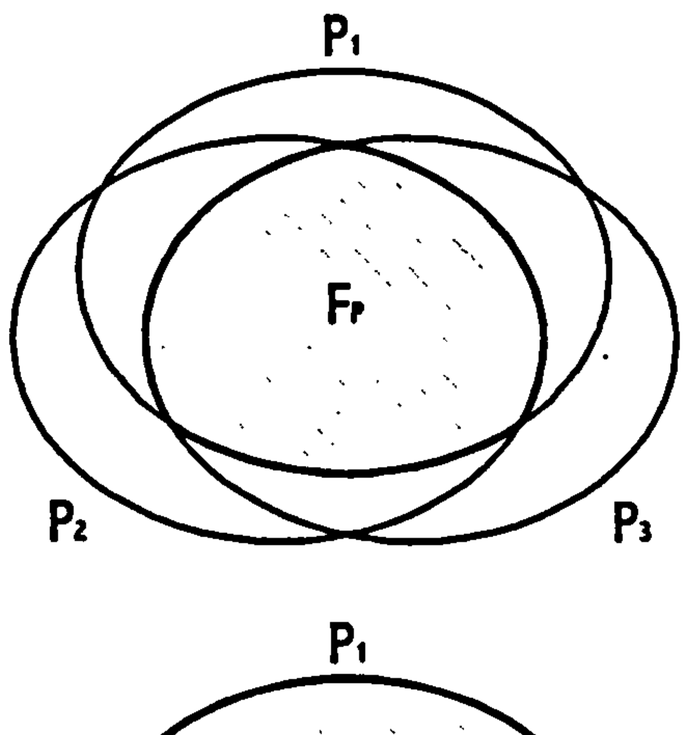
It is therefore convenient to differentiate between these elements of a Master design model into Master Parts and Master Systems.

A *Master Part* - is a single Variant CAD model containing all of the features of a family of similar part designs. Features that are always present, in all instances of the Master Part, shall be called its *Persistent* features, and the remaining features, which may or may-not be present in a particular instance of the Master Part, its *Non-Persistent* features. For example, consider a family of designs that contains parts P_1, P_2 and P_3, which are each defined within separate families, and share enough similar features to warrant the creation of a Master Part, P_M.



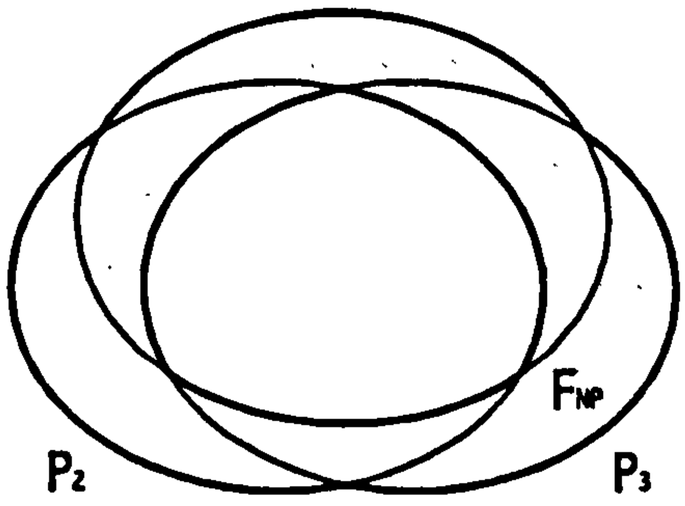
The Master Part, P_M, contains all of the features of parts P_1, P_2 and P_3 combined, i.e.

$$P_M = P_1 \cup P_2 \cup P_3$$



With *Persistent* features, F_P, being the intersection of P_1, P_2 and P_3:

$$F_P = P_1 \cap P_2 \cap P_3$$



And *Non-Persistent* features, F_{NP}, being all combined features of P_M, apart from its Persistent features:

$$F_{NP} = P_M \cap F_P'$$

Figure 3.8 gives an example Master Part from a connector.

Figure 3.7 A Venn Diagram Representation of a Master Part

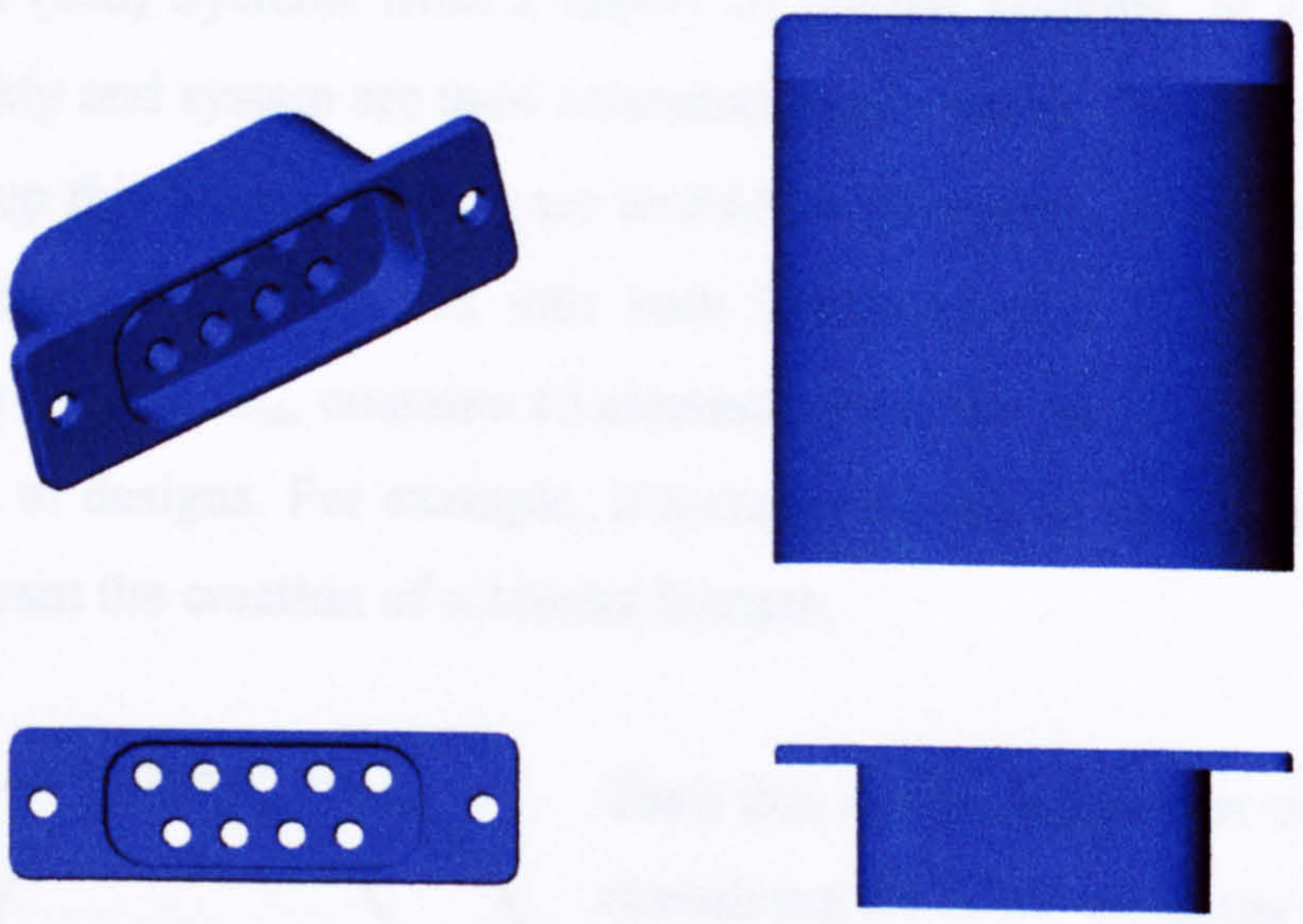


Figure 3.8 Master Part from a Connector

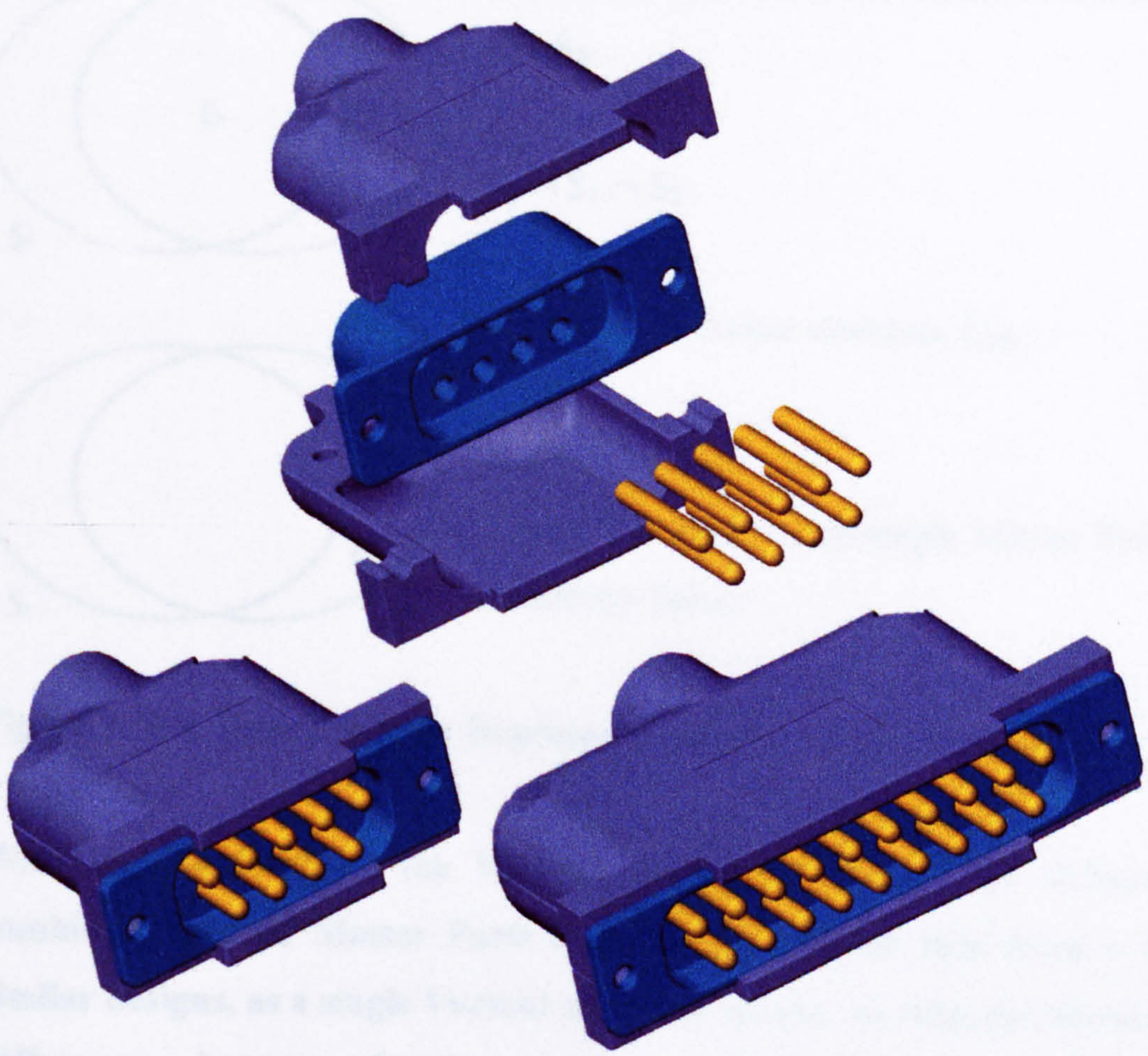
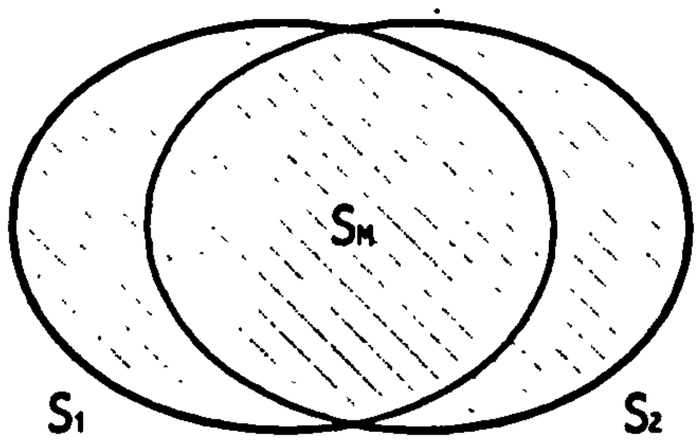


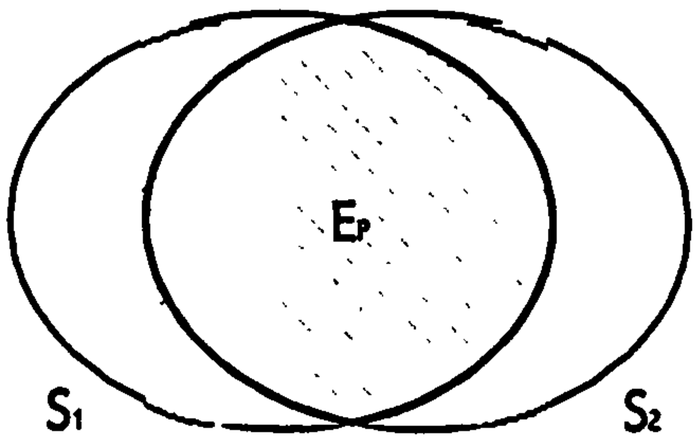
Figure 3.9 A Master System for a Connector and two Instances

A *Master System* - is a single Variant CAD model, made up of all Master Parts and Master (sub) Systems from a family of similar systems, or assemblies. (The terms assembly and system are used interchangeably here). The parts and sub-systems that make up this Master System are termed its *Elements*, and (again with similarities to the Master Part) they fall into both Persistent and Non-Persistent categories. A Master System, S_M, contains all elements from the set of similar systems that form a family of designs. For example, if systems S_1 and S_2 share enough similar elements to warrant the creation of a Master System.



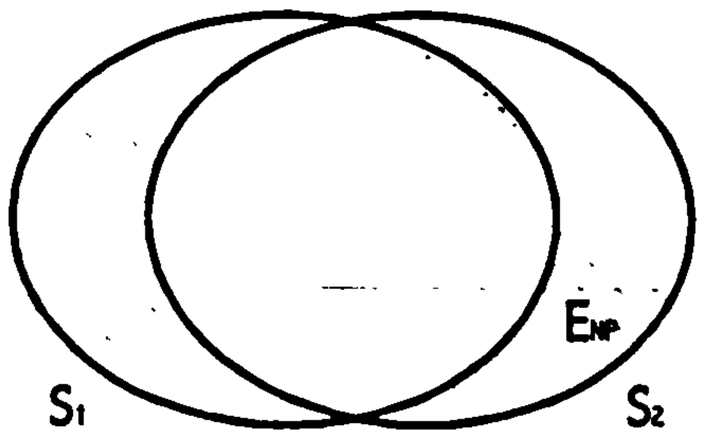
Then this can be defined as the Master System, S_M, containing all of the elements of S_1 and S_2:

$$S_M = S_1 \cup S_2$$



With *Persistent* elements, E_P, the intersection of S_1, and S_2

$$E_P = S_1 \cap S_2$$



And *Non-Persistent* elements, E_{NP} :

$$E_{NP} = S_M \cap E_P'$$

Figure 3.9 shows an example Master System for a connector family

Figure 3.10 A Venn Diagram Representation of a Master System

With these definitions, the Variant Master Model can be defined as the combination of all Master Parts and Master Systems that form a family of similar designs, as a single Variant assembly model. As with the Master System, differences between family members of the Master Model can be accommodated by modifying part parameters values, and/or by turning Parts and Systems on and off (suppression).

3.3 The Generic Methodology

This section presents a Generic Methodology to store Conceptual and Embodiment Designs as a combined Function Means / Parts Tree, and Detailed Designs under a Variant Master Model. The methodology (figure 3.11) is a step-by-step prescription, encompassed by the following objectives:

1) Collate and organise the family of existing manufacturing drawings into Master Parts and Master Systems,
2) Build the Variant Master Part, System and Family CAD models, and determine and create their driving parameters,
3) Create a database representation of the Master Model in the form of a Parts Tree and link the part, system and family CAD models to respective elements of this tree,
4) Determine the functions of each element of the design, and structure this representation in the database, along side the Parts Tree, to form a hybrid Function Means / Parts Tree.
5) Create individual database records for each member of the design family.

The following sections give a more detailed description of the stages of this method.

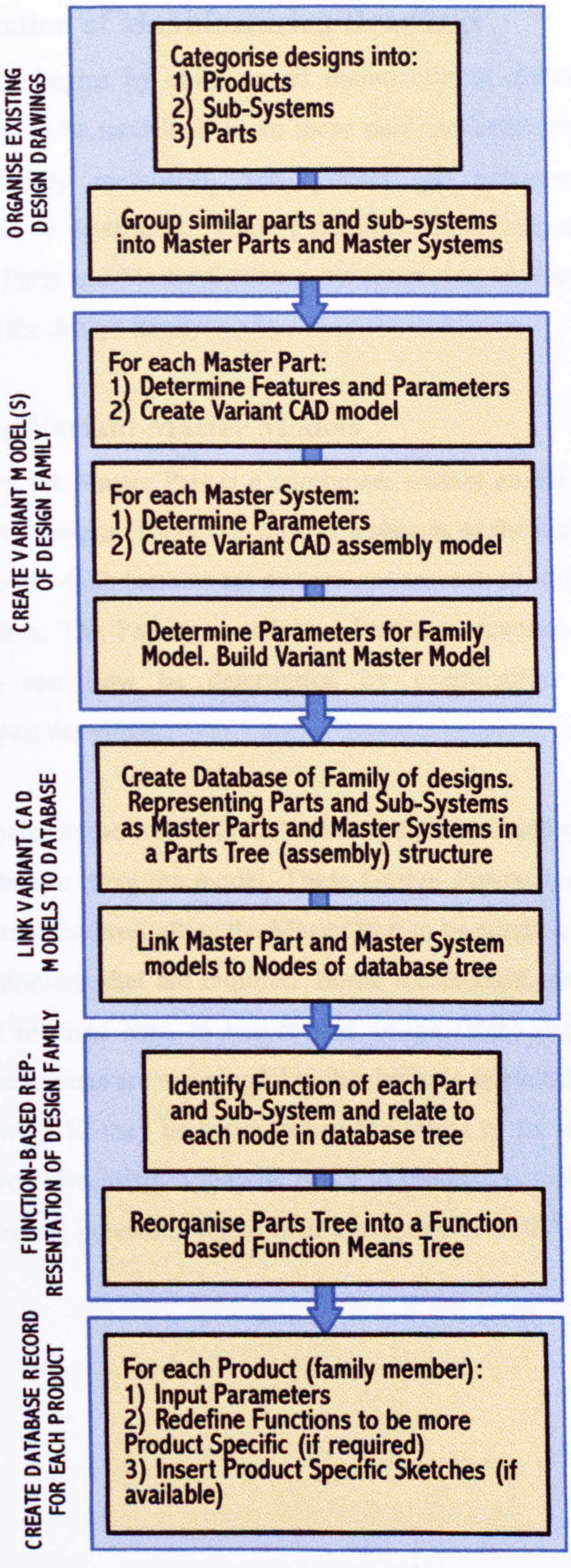


Figure 3.11 The Generic Methodology

3.3.1 Organisation of Manufacturing Drawings

The methodology begins by collating all manufacturing drawings for the entire family of designs to be modelled. From these part and assembly drawings, sets of Product (i.e. family members), Sub-Systems (or sub-assemblies) and Part (components) can be determined. These groupings can consequently be categorised into the Master Parts and Master Systems, by comparing similar part and assembly drawings across the design family.

3.3.2 Creating Variant Master Models

As stated earlier, the Master Part is a combined, variant model of all similar parts across a family of designs. Therefore various instances of the Master Part may differ by the definition of which features are and are-not present, as well as the values of its driving parameters. The Persistent and Non-Persistent features of the Master Part (section 3.2.2) can now be determined by examination of the concerned manufacturing part drawings.

An important factor in the creation of a Variant (or parametric) model is to define the correct parameters to drive the model. These *Global Parameters* and the Persistent and Non-Persistent features, allow the Master Part to be adapted to suit the variety of specific configurations that are required. Some forethought, and knowledge of the design field, is required here, to ensure that design intent is maintained, and that these driving parameters are meaningful to the design's application. For example, the channel of figure 3.12 may be better defined in terms of its width (w) rather than offsets from two sides. Also, it may be better to produce a model of a gear with its pitch (P) as a driving parameter rather than its number of teeth (N) as in figure 3.13.

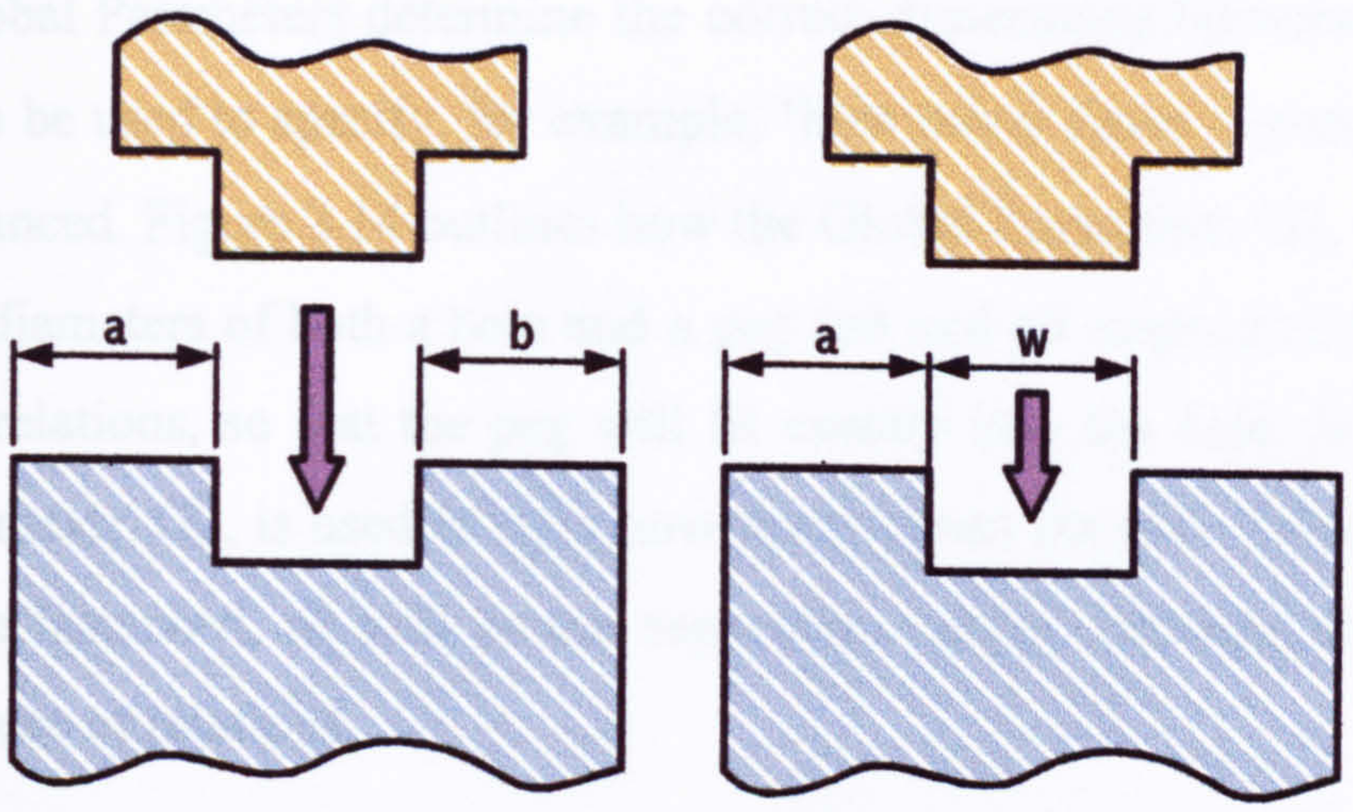
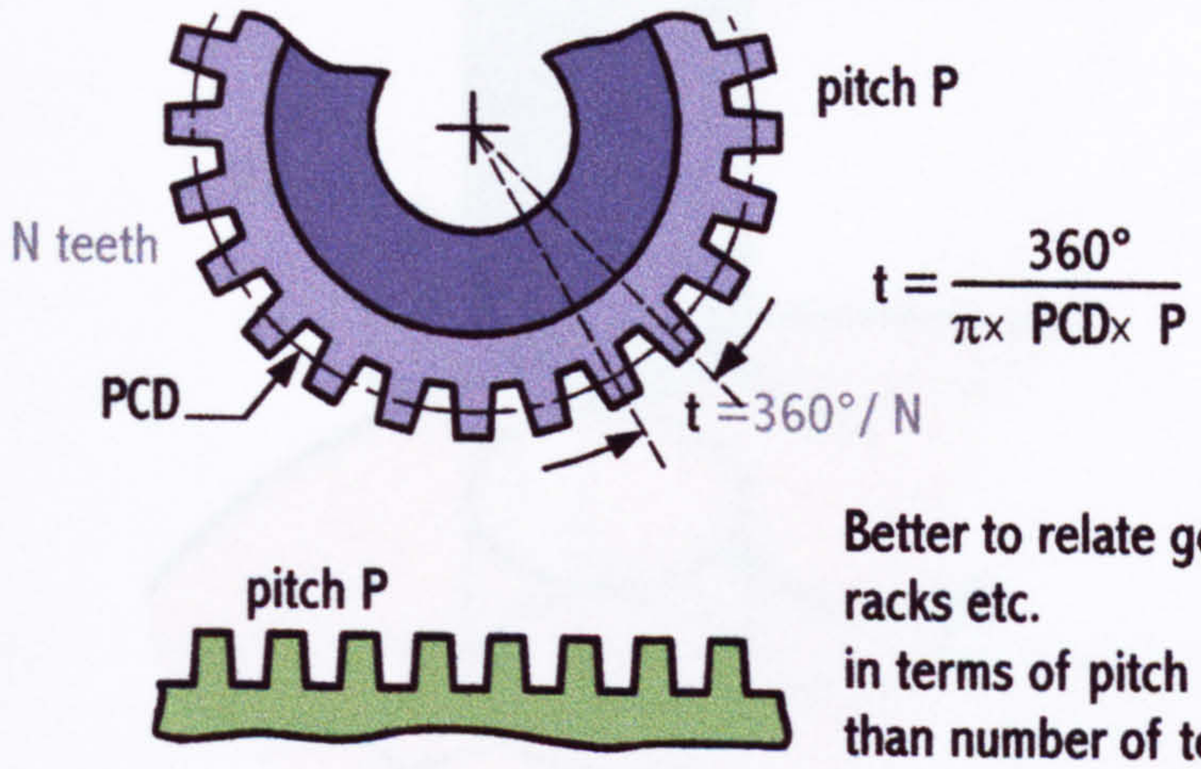


Figure 3.12 Possible Parameter definitions of a Channel



$$t = \frac{360°}{\pi \times PCD \times P}$$

Figure 3.13 Parameter definitions for a Gear design

Now that all modifiable elements have been defined, a Variant model of the Master Part can be created, using a suitable Variant CAD modelling system. This can be achieved by initially creating a *base-part* from all of the combined Persistent features. The Non-Persistent features can then be added and suppressed (hidden) as required. Global (or driving) Parameters can then be defined and related to the Master Parts own parameters. This procedure can be undertaken for all Master Parts of the design family.

A Master System is the combination of all similar systems within the design family, and as such contains Persistent and Non-Persistent elements (again see section 3.2.2). As with the Master Part, these elements of the Master System can be

determined by inspection of the manufacturing, assembly drawings. For the Master System, Global Parameters determine the correct dimensions between mating parts, and can also be used to specify, for example, 'how many times' a particular element is to be instanced. Figure 3.14 outlines how the Global Parameter, G1, can be used to control the diameters of both a hole and a peg (p3 and p9 respectively), through the use of two relations, so that the peg will fit exactly into the hole. In figure 3.15 a Global Parameter, G2, is used to state how many times the hole feature and the peg part, will be instanced, as well as the separation (angle) between instances, again through the use of relations.

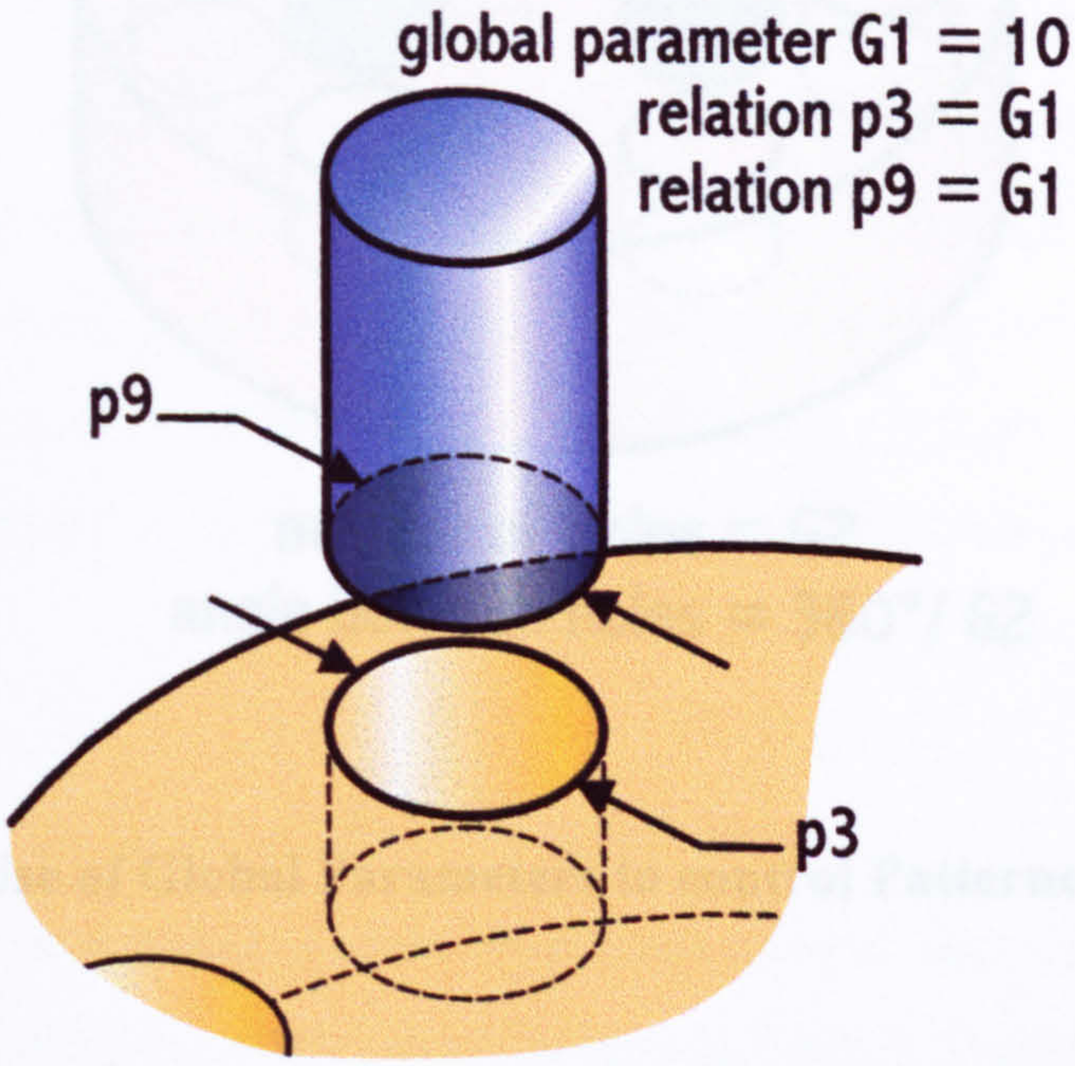


Figure 3.14 Global Parameters to retain Design Intent in an Assembly

From this, a Variant *assembly* CAD Model can be created to form the Master System. All Persistent elements are combined to create a *base system*, and all Non-Persistent elements are added and suppressed. The Master Systems set of driving, Global Parameters are then related to those of its constituent parts and sub-systems. Again, this procedure is repeated for all Master Systems.

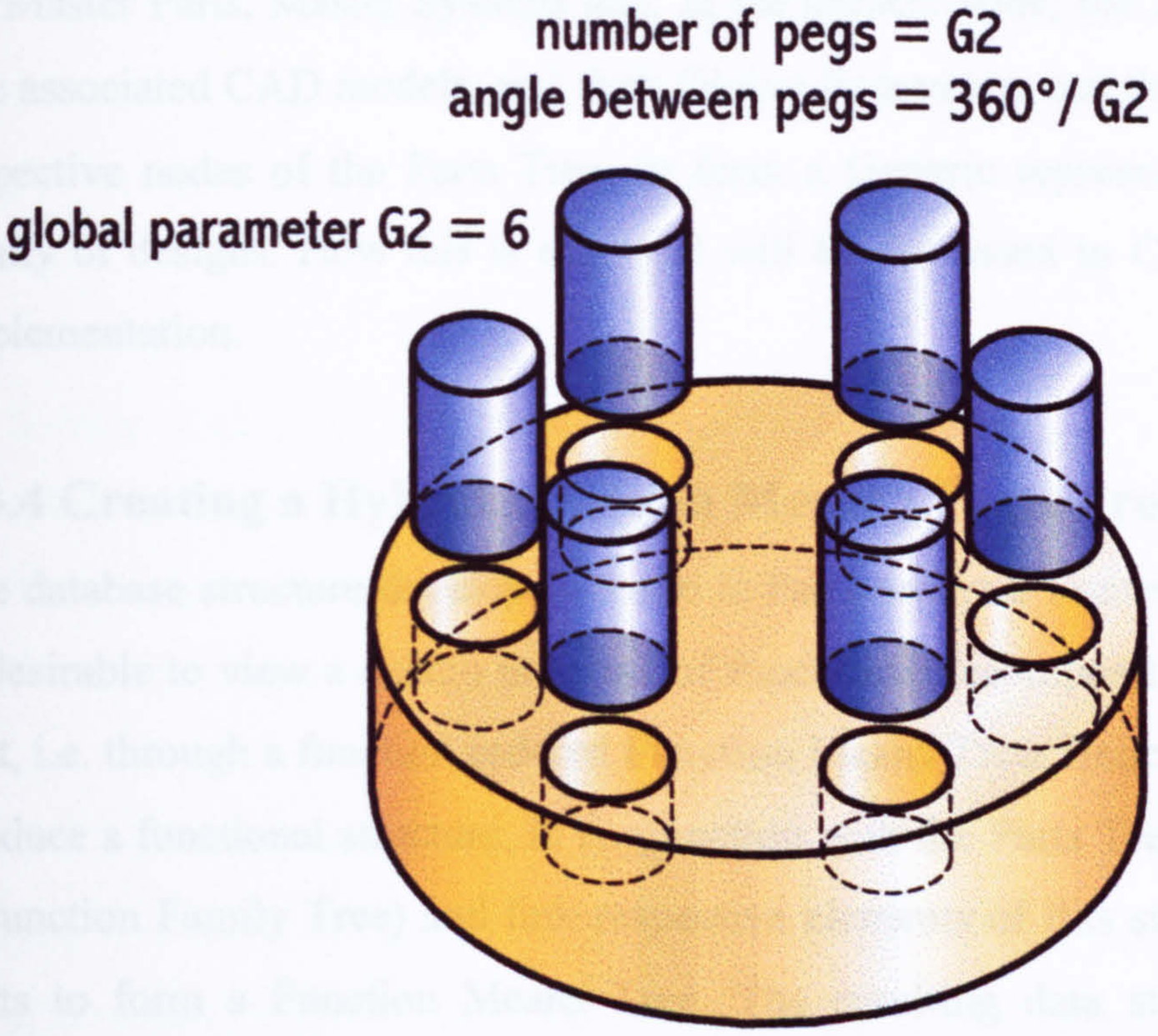


Figure 3.15 Use of Global Parameters to control Patterned Instancing

The Master Family can be regarded as a 'top-level' analogy of the Master System, as it is also an assembly model of (Master) parts and sub-systems. Hence, the procedures to create the Master Family are consistent with those of the Master System, where family-level Persistent and Non-Persistent elements, Global Parameters and a Variant CAD Model (the Master Model) can be created. The major difference here lies with the importance of the Master Family Model, and particularly with its driving parameters, as these by definition, have the highest level of control over the design.

3.3.3 Creating a Parts Related Database of the Master Model

The next stage of the methodology is to structure the elements of the Variant Master Model, within a database environment. This is initially performed in terms of a Parts Tree, as this hierarchy is already present in the assembly structure of the Master

Model. Here, a tree structure (or a 'linked-list') is created, whose elements relate to the Master Parts, Master Systems and, at the highest node, the Master CAD Model. The associated CAD models, and their Global Parameters can then be 'linked' to the respective nodes of the Parts Tree, to form a Generic representation of the entire family of designs. How this is achieved will be discussed in Chapter 4 – Software Implementation.

3.3.4 Creating a Hybrid Function Means / Parts Tree

The database structure developed above is Parts-ordered. For evaluation purposes it is desirable to view a design in terms of functional decomposition and the realising part, i.e. through a function ordered Function Means Tree. Hence the next stage is to produce a functional structure, in conjunction with the Parts Tree representation (i.e. a Function Family Tree) and link respective elements of this structure to individual parts to form a Function Means Tree. The resulting data structure can now be automatically re-ordered to display either a function or part ordered representation of the design family.

3.3.5 Recording Individual Family Members

At this stage, all Master CAD models have been created, and have been structured with their representative functional descriptions to form a hybrid Function Means / Parts oriented Tree. This is a Generic structure, representing the entire design family. Therefore parameters can now be entered for specific, individual products, as can product specific functions, and stored as records in a database. Along with this, information regarding which Non-Persistent Master Part features, Master-System, and Master Model elements are required, is also stored. Hence, each member of the family of designs is portrayed by a concise representation of its driving parameters, components and product specific functions.

3.4 An Illustrative Example

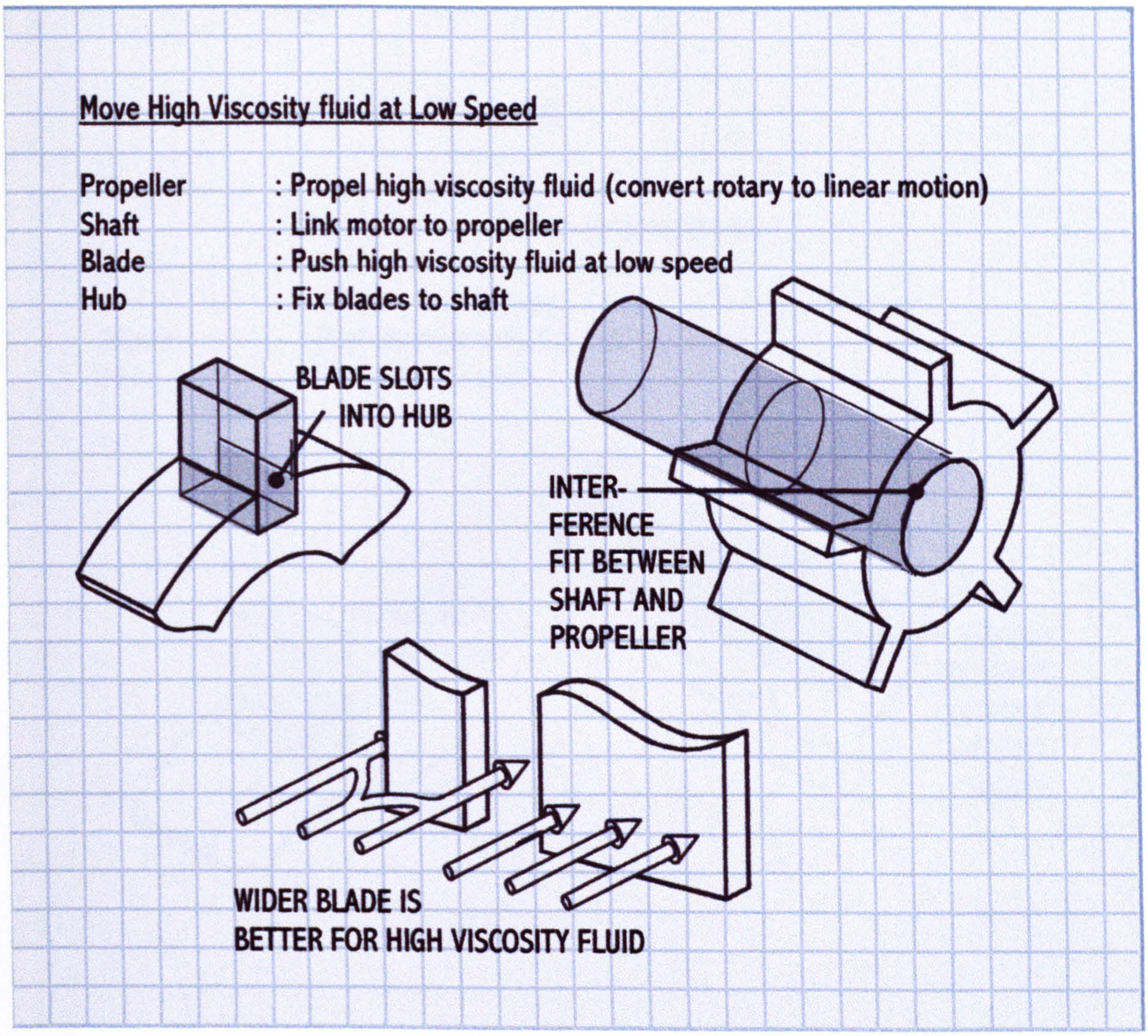
The Generic Methodology presented describes how an existing, entire family of designs can be transformed into a single variant model. This section will take a simple example design to illustrate this principle with regard to the following criteria:

- Storing families of designs in a structured manner (i.e. creating (Master) parts, systems and family).
- Recording design intent, concepts and functional information.
- Creating a reusable CAD model of a design family.

3.4.1 The Propeller – Shaft Assembly

Figures 3.16 and 3.17 illustrate two design examples from a (hypothetical) simplified propeller-shaft family. Each design contains a functional description of its constituent parts and systems, annotated conceptual sketches and dimensioned manufacturing drawings for the piece parts. Both designs share a common, primary (or family) function, which is to 'propel a fluid using a rotary motor' as a power source.

Design 'A' was created to move a high viscosity fluid (e.g. crude oil) at low speeds, hence the use of a greater number of deep blades. Conversely, the intention for Design 'B' was to propel a less viscous fluid (such as petrol) at a relatively higher speed, requiring fewer and narrower blades, to reduce inertia. The interference fit between the shaft and the hub of Design 'A' was found to be ineffective for Design 'B' (due to a combination of high kinematic forces / torque produced in the contact area and the low viscosity of the fluid). So, it was decided to use a keyway between the shaft and hub.



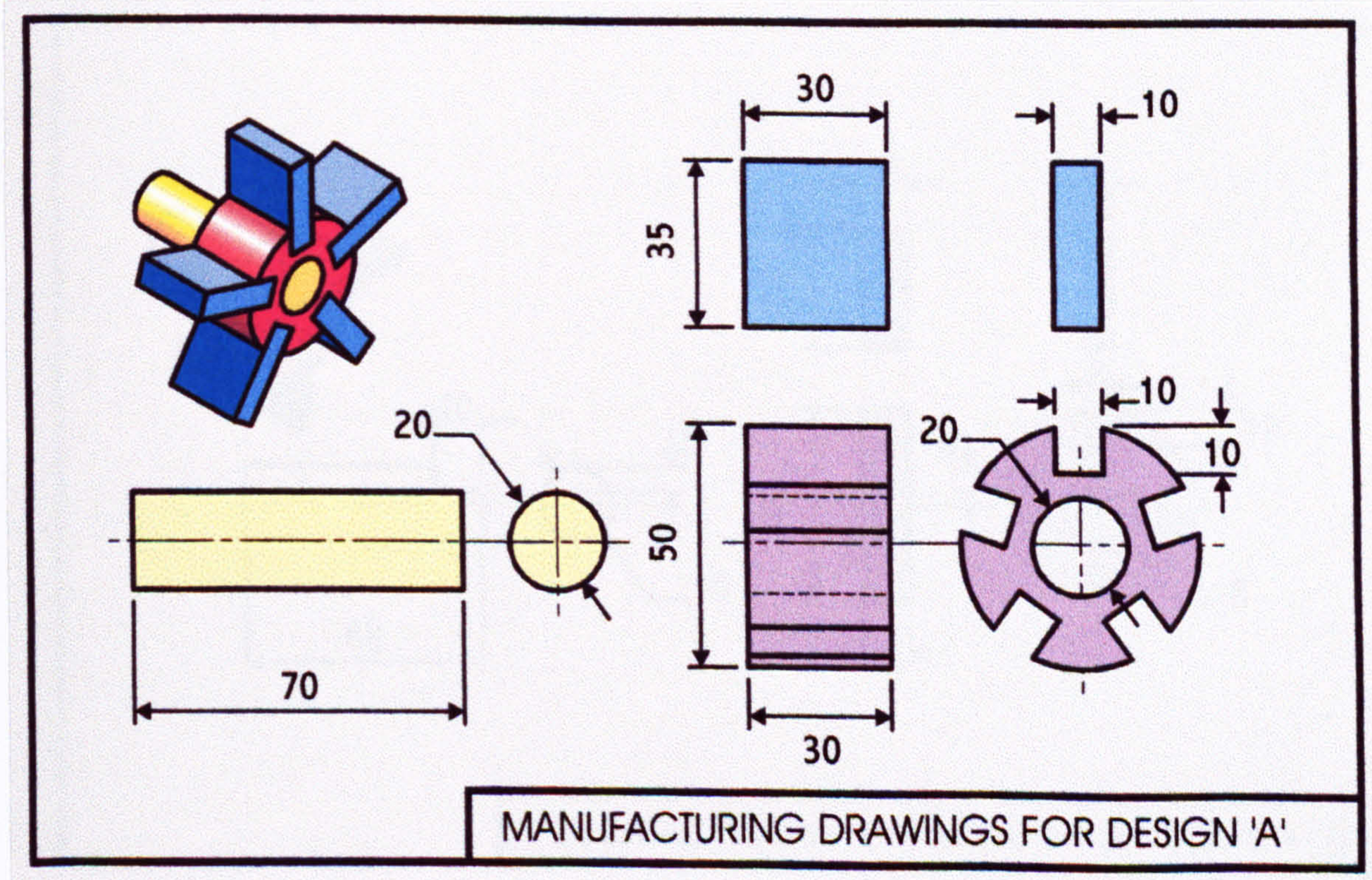
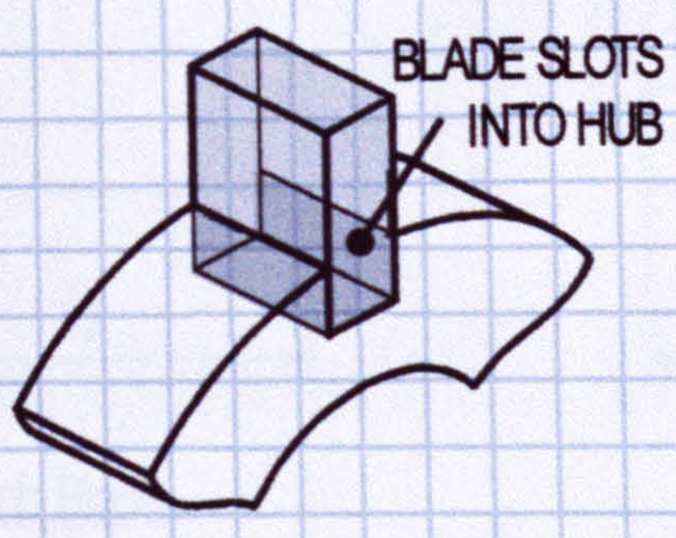
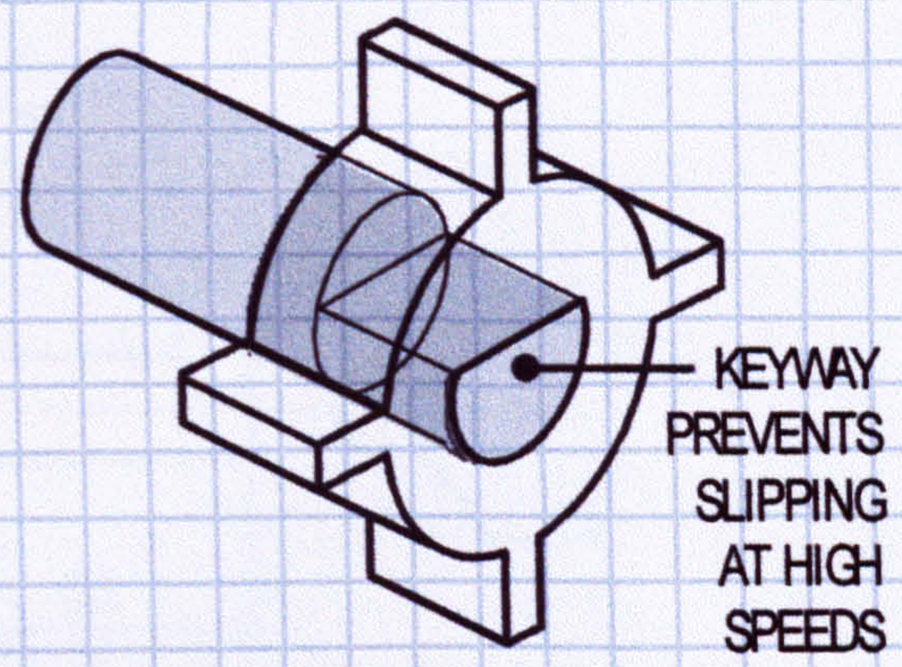


Figure 3.16 Example Design 'A' for a Prop-Shaft assembly

Move Low Viscosity fluid at High Speed

Propeller : Propel low viscosity fluid
(convert rotary motion to linear motion)
Shaft : Link motor to propeller and prevent
slipping with keyway
Blade : Push low viscosity fluid at high speed
Hub : Fix blades to shaft





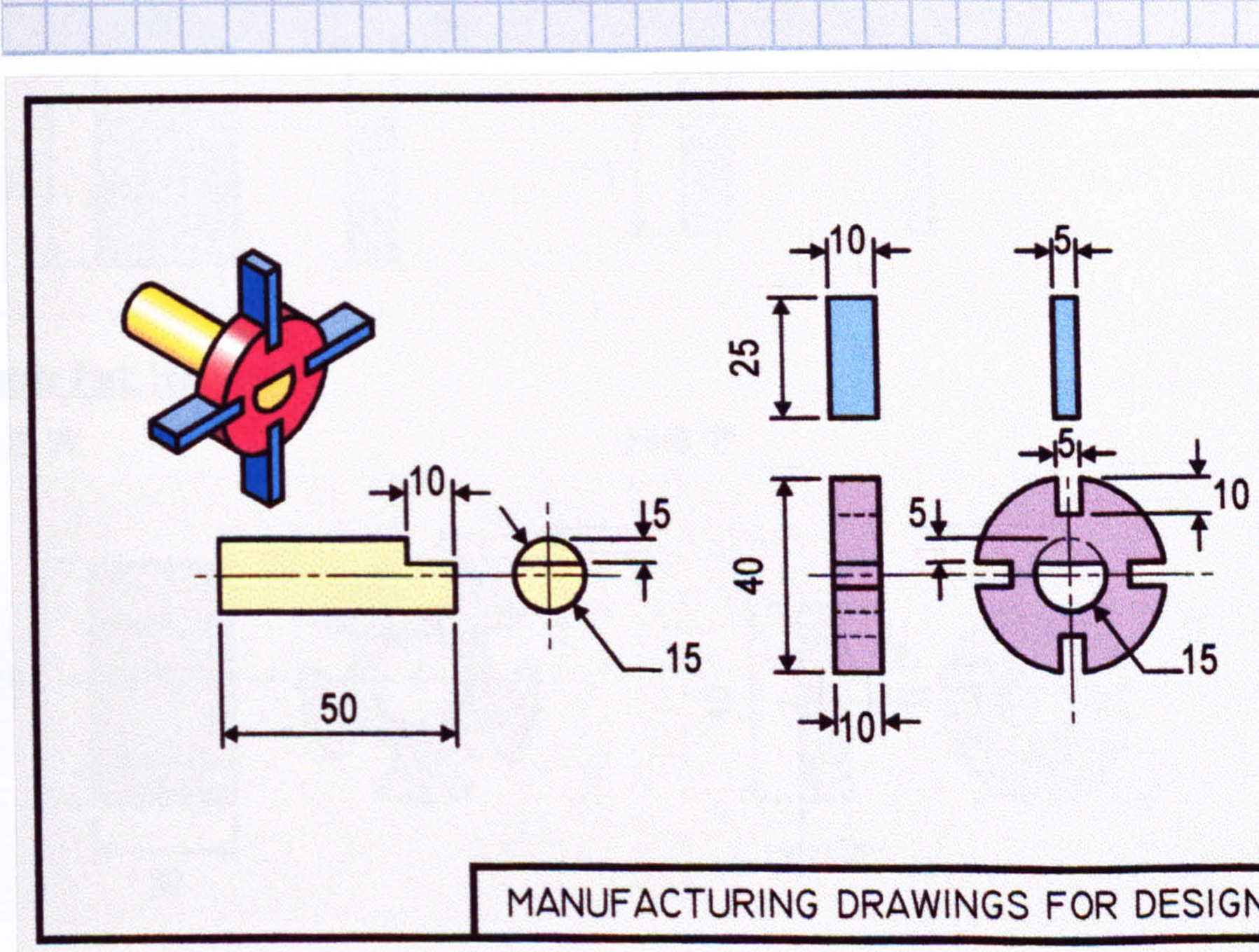


Figure 3.17 Example Design 'B' for a Prop-Shaft assembly

3.4.2 Organisation of Manufacturing Drawings

Both drawings show each product to consist of three Master Parts, the *Shaft*, *Hub* and *Blade*. (figure 3.18). Where each product contains only one shaft and hub and a variable number of blades. There is only one Master System in this example, the *Hub-Blades* assembly. This, combined with the shaft (Master Part) are the two elements that make-up the Master (family) Model (figure 3.19).

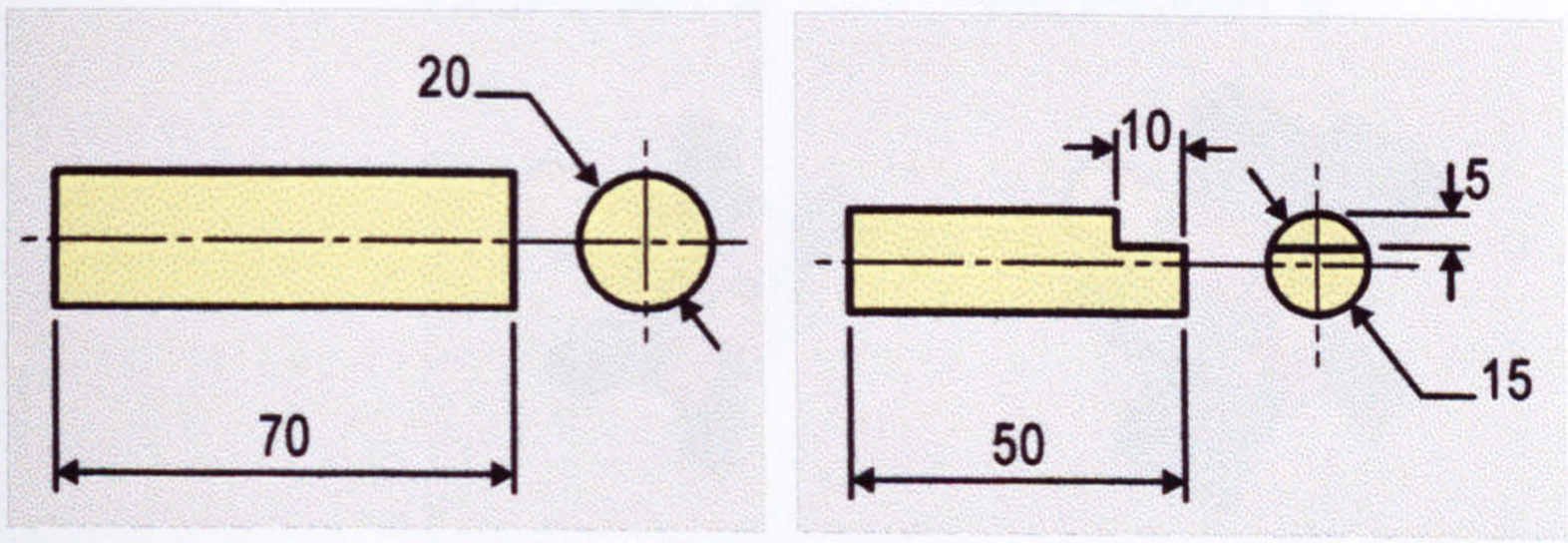


Figure 3.18 The Master Parts

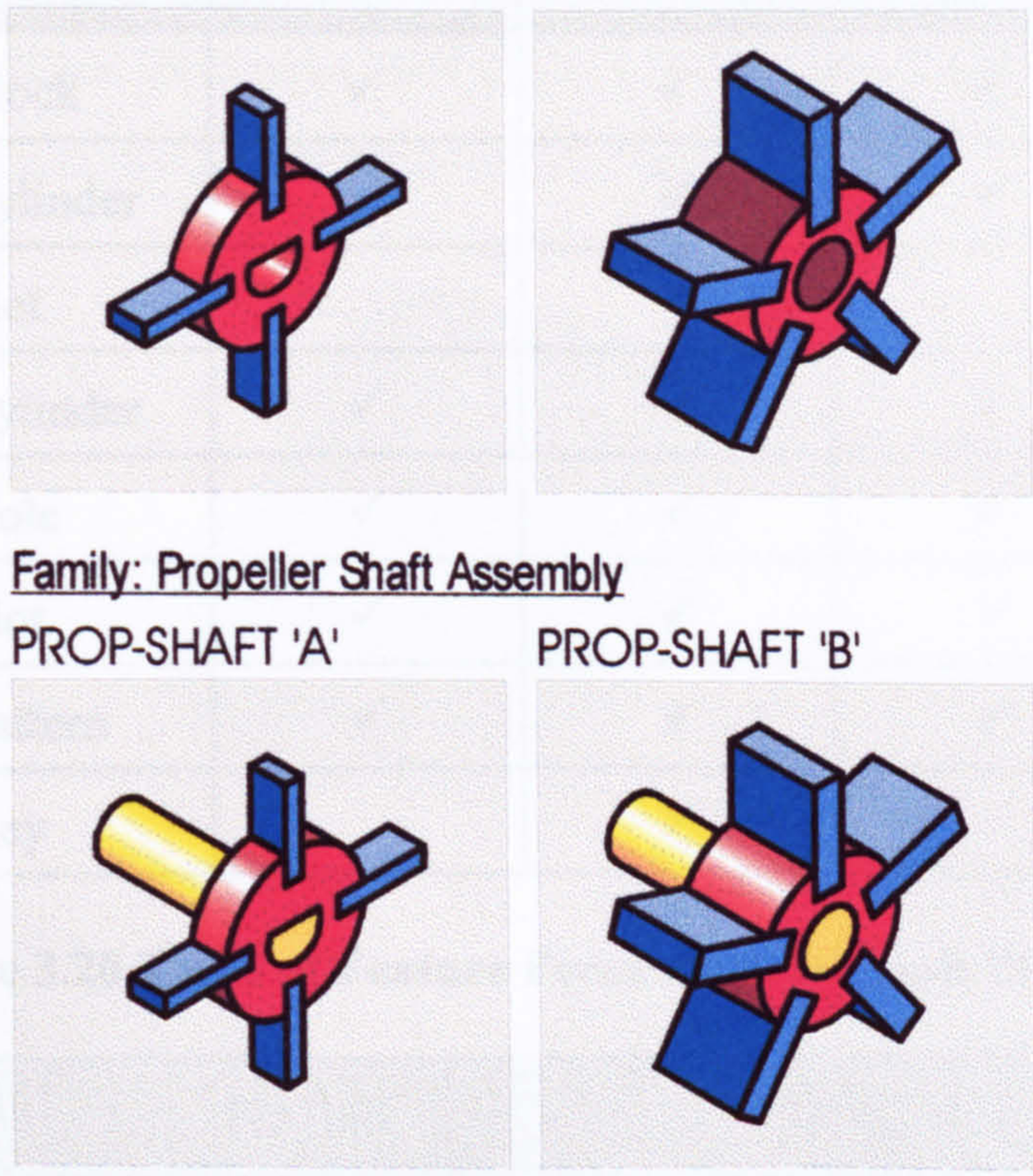


Figure 3.19 The Single Master System (above) and Master Model (below)

3.4.3 Creating the Variant CAD Models

Starting with the simplest Master Part, the blade has only one feature – a block, which must be persistent. The shaft contains one Persistent base feature, a cylinder, and a Non-Persistent slot feature. Similarly the hub contains a Persistent cylindrical base feature, a Persistent hole feature to fit the shaft and a Persistent slot feature to fit a blade. A further Persistent feature is the pattern (or array) of slots to hold multiple blades. Leaving only a Non-Persistent key feature. Figure 3.20 shows the Persistent and Non-Persistent feature sets for each product. Driving Parameters for each Master Part can now be determined, as per figure 3.21.

PART	FEATURE	DRAWING 1	DRAWING 2	PERSISTENT	NON-PERSISTENT
Blade	Block	✓	✓	✓	
Shaft	Cylinder	✓	✓	✓	
	Slot		✓		✓
Hub	Cylinder	✓	✓	✓	
	Hole	✓	✓	✓	
	Slot	✓	✓	✓	
	Pattern	✓	✓	✓	
	Key		✓		✓

Figure 3.20 Table of Feature Persistence for each Product

PART	FEATURE	PARAMETER
Blade	Block	Overall height
		Overall width
		Overall depth
Shaft	Cylinder	Overall diameter
		Overall length
	Slot	Height of key slot
		Depth of key slot
Hub	Cylinder	Overall diameter
		Overall depth
	Hole	Diameter
	Slot	Height of blade slot
		Width of blade slot
	Pattern	Total number of slots (blades)
	Key	Height of keyway

Figure 3.21 Table of Driving Parameters

The three Master Parts can now be created using a suitable variant modelling package. Here, Pro/ENGINEER is used, which is capable of feature suppression, for the two Non-Persistent Shaft-Slot and Hub-Key features. Figure 3.22 shows the Pro/ENGINEER CAD models for each Master Part

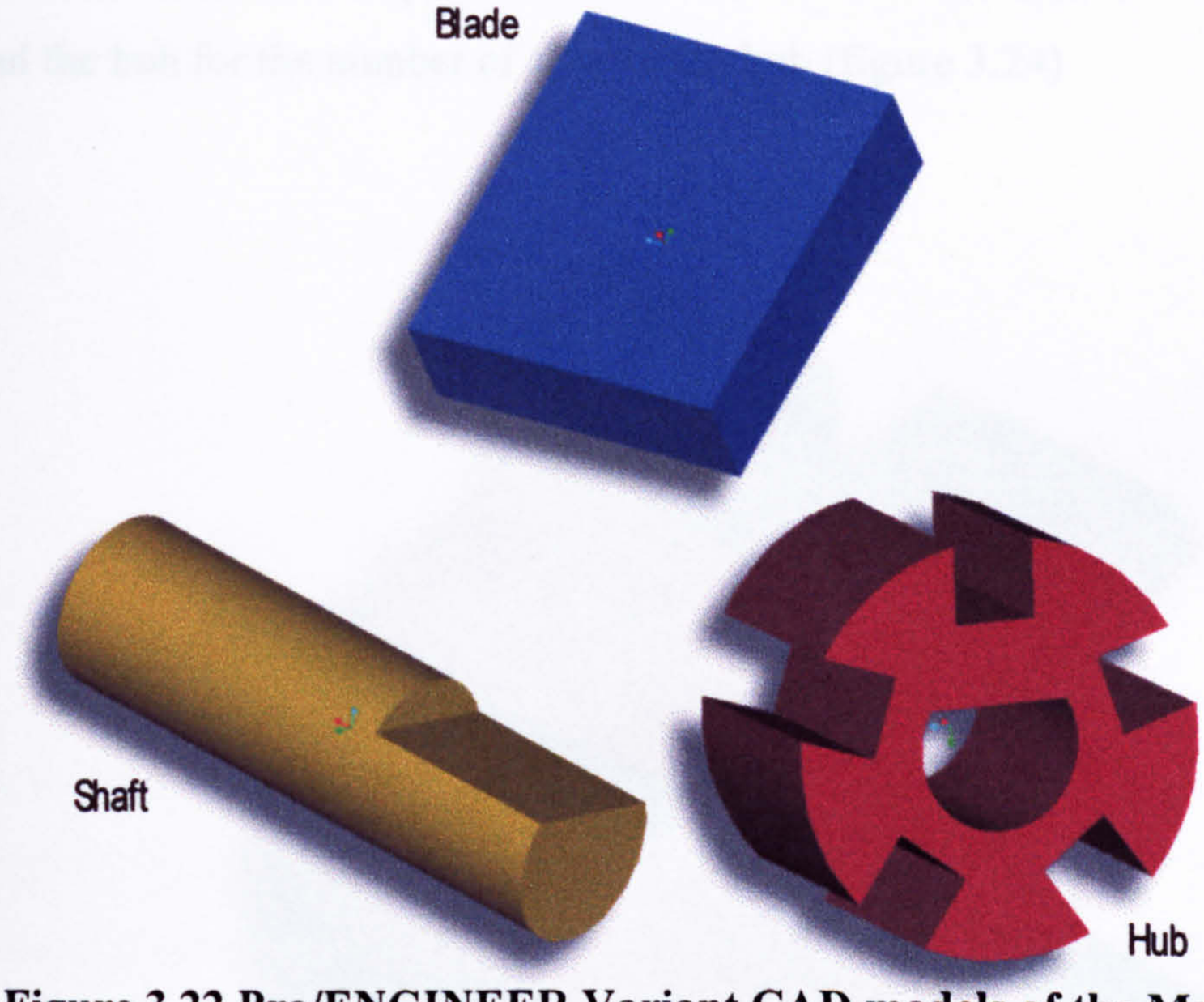


Figure 3.22 Pro/ENGINEER Variant CAD models of the Master Parts

Each Master Part is driven by a set of Global Parameters. These, along with a set of Non-Persistent features, are the parameters that drive the part model when linked to its Parts Tree node. Lists of the parameters are given below.

BLADE	SHAFT	HUB
B_height	S_dia	H_dia
B_width	S_length	H_depth
B_depth	S_key_slot_height	H_hole_dia
	S_key_slot_depth	H_blade_slot_dia
		H_blade_slot_width
		H_number_of_slots
		H_key_height

Figure 3.23 Table of Master Part Global Parameters

The single (Master) System in the prop-shaft family is the Propeller assembly model, in which all elements, the Hub part and the Blade part are Persistent, i.e. they exist in all products. So, there are no Non-Persistent elements in this Master System. A new assembly model can now be created, containing the Hub and Blade parts, and the blade constrained to a respective slot in the hub. It can then be 'insert-patterned' around the hub for the number of slots in the hub (figure 3.24)

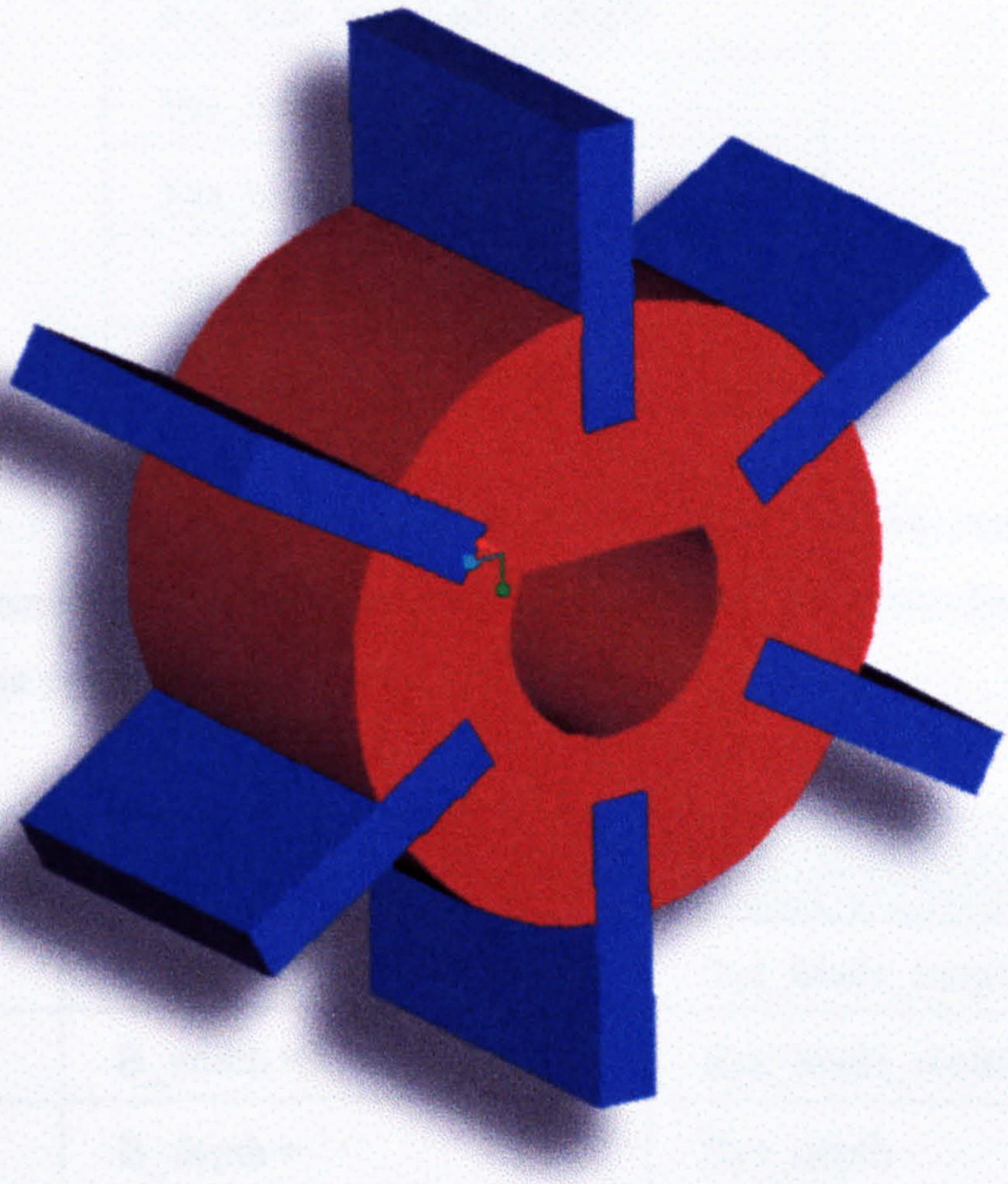


Figure 3.24 Pro/ENGINEER model of the Propeller Assembly

As for the Master Part model, a set of driving Global Parameters must be created to:

1) relate parameters between parts in the system, and
2) link the Master System CAD model to its Node in the Parts Tree.

For the Propeller system, the following Global Parameters were created :

GLOBAL PARAMETER
Sys_hub_dia
Sys_depth
Sys_hub_hole_dia
Sys_hub_blade_slot_height
Sys_hub_key_height
Sys_blade_height
Sys_blade_width
Sys_number_of_blades

Therefore, the following System to Part *relationships* can be created to ensure respective dimensions between connecting parts can be controlled by a single parameter at the system level:

PART	SYSTEM RELATIONSHIP	
for the Blade	B_height =	Sys_blade_height
	B_width =	Sys_blade_width
	B_depth =	Sys_depth
for the Hub	H_dia =	Sys_hub_dia
	H_depth =	Sys_depth
	H_hole_dia =	Sys_hub_hole_dia
	H_blade_slot_height =	Sys_hub_blade_slot_height
	H_blade_slot_width =	Sys_blade_width
	H_number_of_slots =	Sys_number_of_blades
	H_key_height =	Sys_hub_key_height

The Master Model can be viewed as a top-level Master System. Hence the procedure for developing a Master System is adopted. The two (sub-ordinate) elements that form the Master Family are the *Shaft Master Part* and the *Propeller Master System*, both of which are Persistent. Now, the final CAD model, the Family assembly, can be created by constraining the Propeller system assembly model to the Shaft part, as a new Pro/ENGINEER assembly model (figure 3.25).

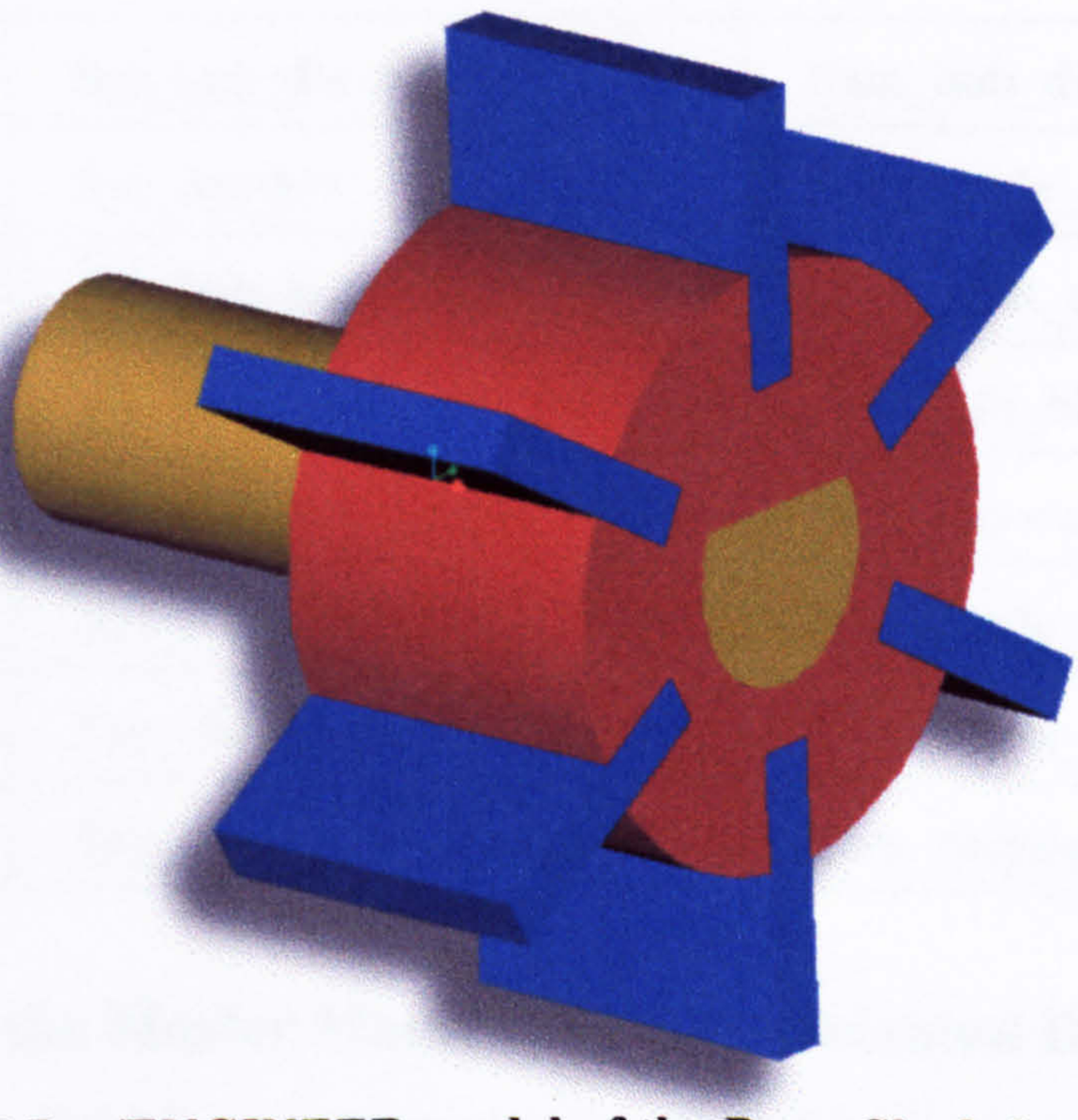


Figure 3.25 Pro/ENGINEER model of the Prop-Shaft family assembly

The set of Global Parameters created for this model will control all of the parameters the designer may wish to modify, for the entire design. Below are the nine Master Family Global Parameters, created for the Propeller-Shaft model, and their corresponding Shaft part and Hub-Blades system relations.

GLOBAL PARAMETER	GLOBAL PARAMETER
Fam_shaft_diameter	Fam_blade_depth
Fam_shaft_length	Fam_hub_diameter
Fam_keyway_height	Fam_hub_blade_slot_height
Fam_blade_width	Fam_number_of_blades
Fam_blade_height	

ELEMENT	FAMILY RELATIONSHIP	
for the Shaft	S_dia =	Fam_shaft_diameter
	S_length =	Fam_shaft_length
	S_key_height =	Fam_keyway_height
	S_key_length =	Fam_blade_depth
for the Hub - Blade assembly	Sys_hub_dia =	Fam_hub_diameter
	Sys_depth =	Fam_blade_depth
	Sys_hub_hole_dia =	Fam_shaft_diameter
	Sys_hub_blade_slot_height =	Fam_hub_blade_slot_height
	Sys_hub_key_height =	Fam_keyway_height
	Sys_blade_height =	Fam_blade_height
	Sys_blade_width =	Fam_blade_width
	Sys_number_of_blades =	Fam_number_of_blades

3.4.4 Linking the Master Model to a Parts Oriented Database

Now the completed CAD models can be represented in a parts tree, as a database. Chapter 4 will deal with the specific (software) implementation of how this is achieved, but for the purposes of illustration we shall consider the linked Parts Tree of figure 3.26. As well as linking the actual CAD model, the lists of Global Parameters for each Master Part, Master System and Master Model are also linked. This enables the CAD models parameters to be changed from within the database.

3.4.5 Creating a Function Means Tree

The next stage is to identify the functional structure of the design. Figure 3.27 illustrates a Function Family Tree for the Propeller Shaft Assembly. Therefore, the Means (or Parts) of figure 3.26 can be related to these functions to form a Function Means Tree (figure 3.28).

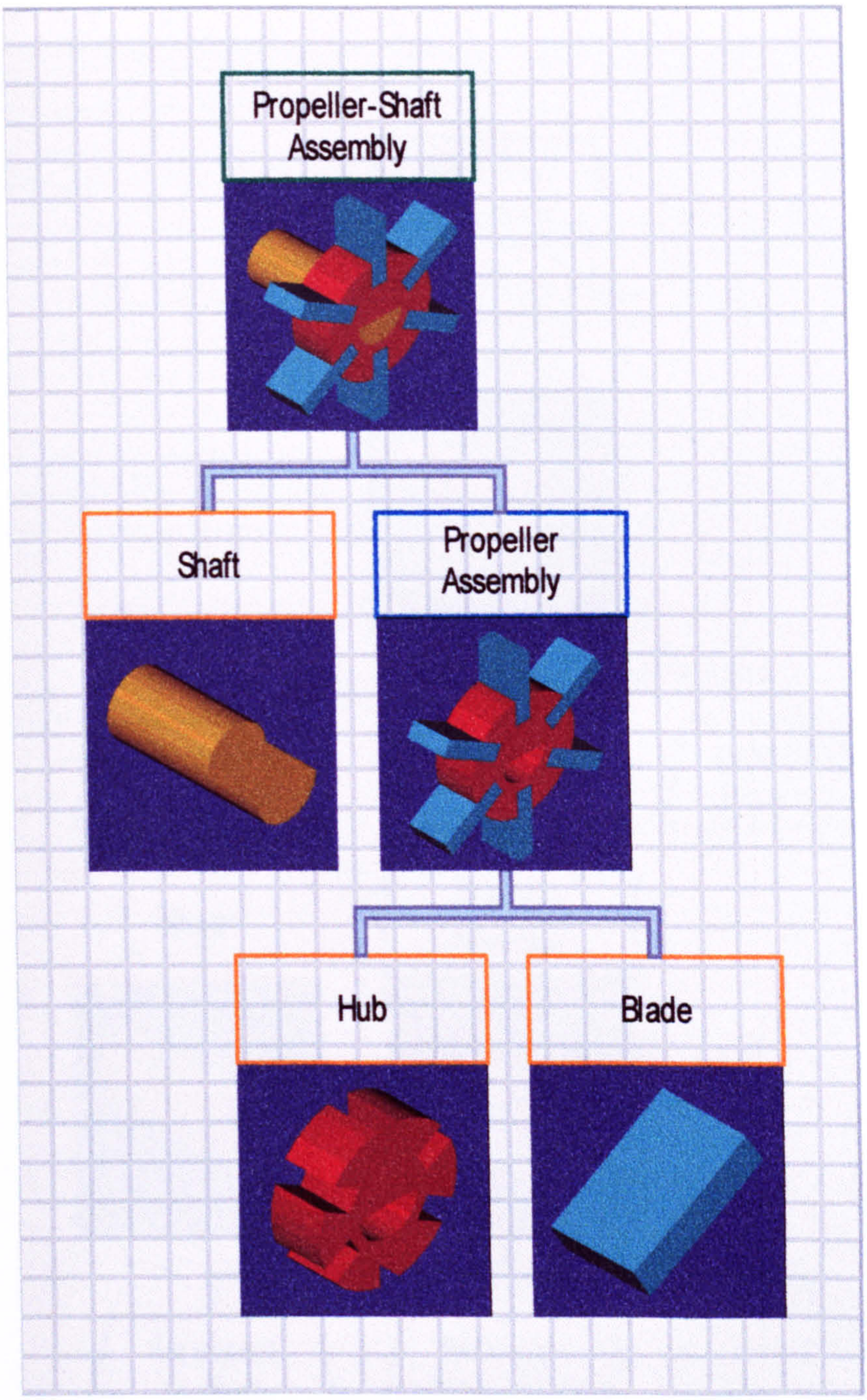
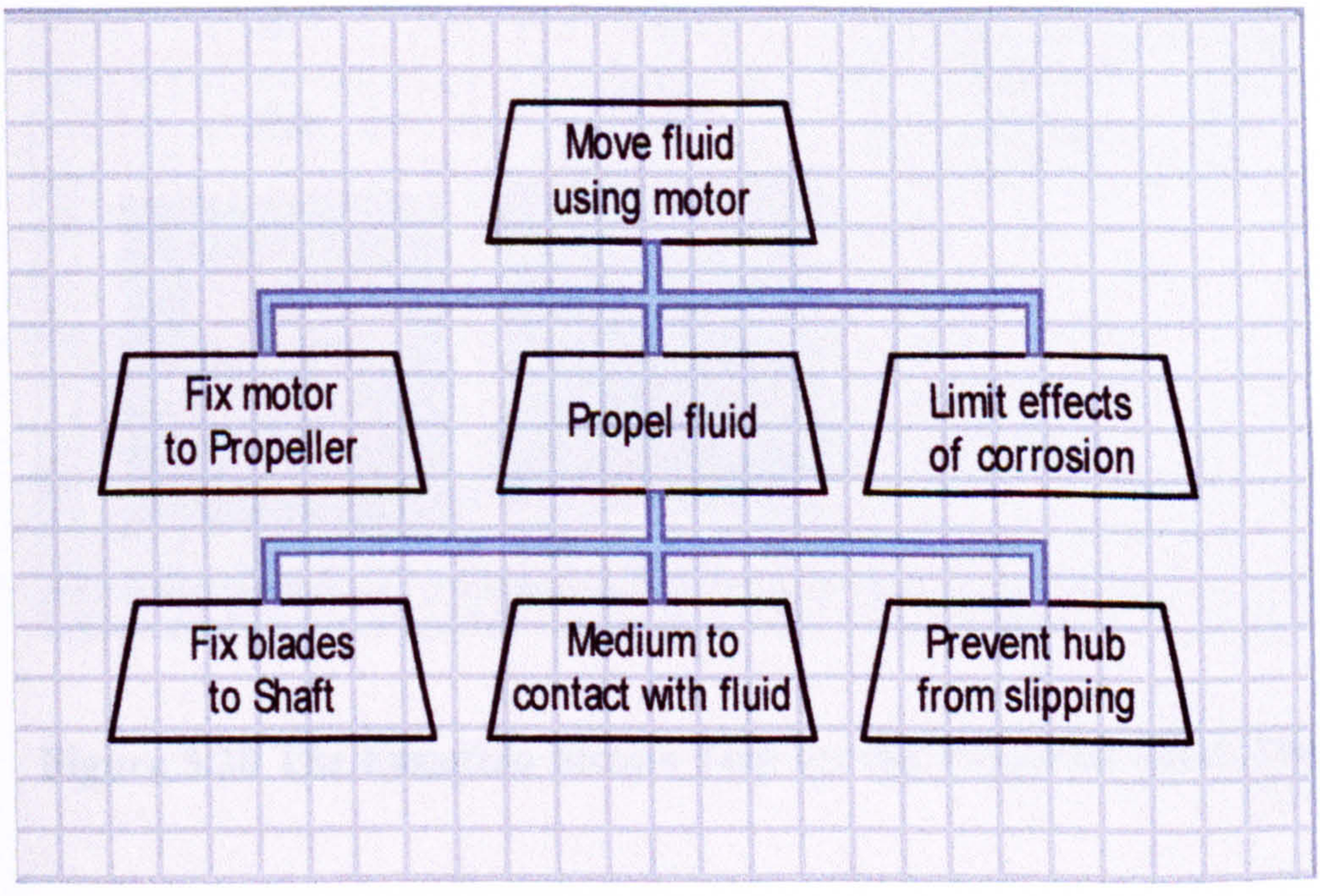


Figure 3.26 (left) A Parts Tree Representation of the Master Model.

Figure 3.27 (below) A Function Tree Representation



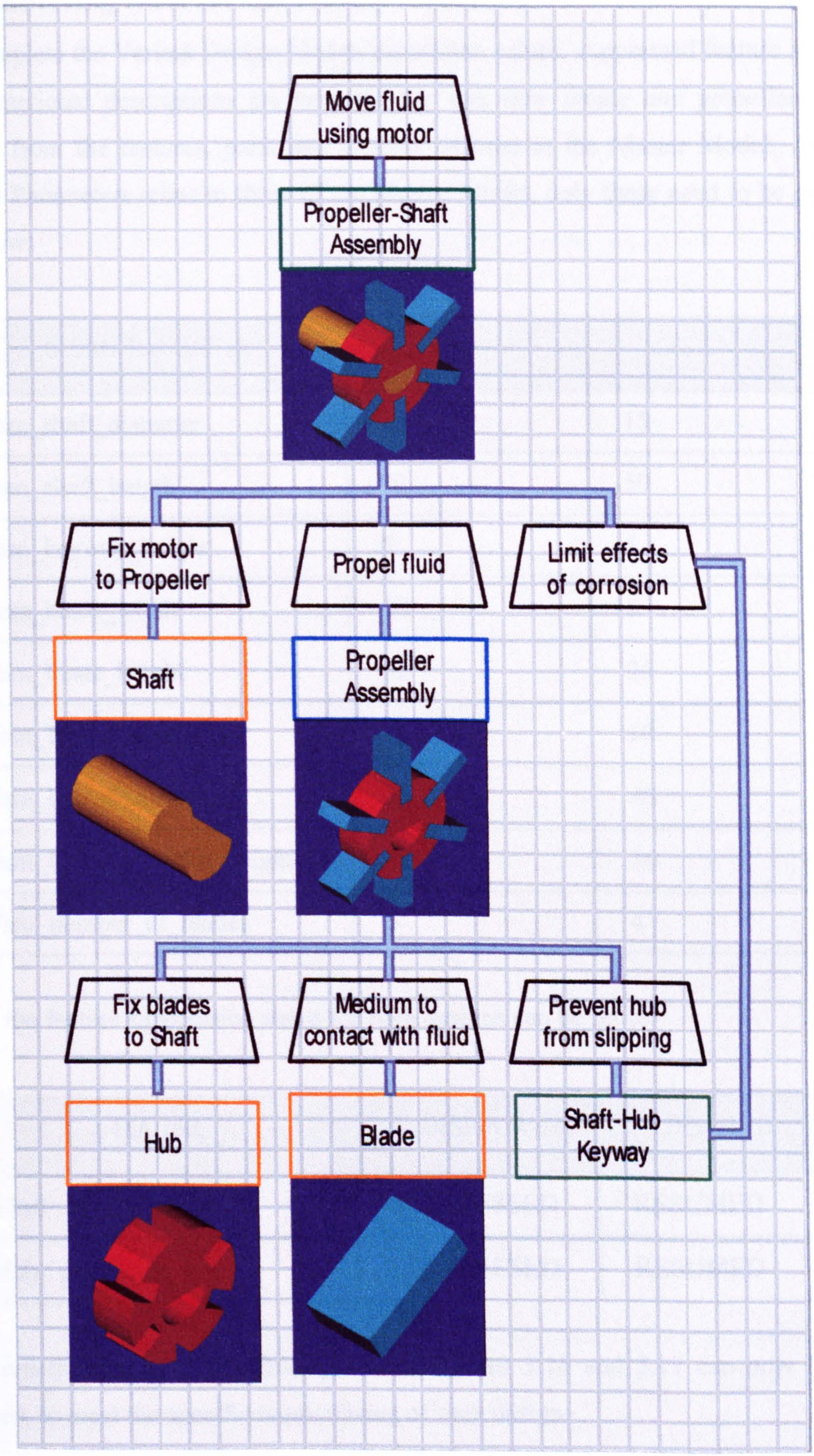


Figure 3.28 The Function Means Tree for the Propeller-Shaft Design

3.4.6 Entering Data for the Family Members

To complete the Variant Design Model, parameter values, suppressed feature status, and functional descriptions are entered. We can now create *any* propeller-shaft design from the features, parts and systems defined in the Master Model. As all Global Parameters relate to those of the Master Model, only these need to be given, as below:

GLOBAL PARAMETER	DESIGN 'A'	DESIGN 'B'
Fam_shaft_diameter	20	15
Fam_shaft_length	70	50
Fam_keyway_height	5	5
Fam_blade_width	10	5
Fam_blade_height	35	25
Fam_blade_depth	30	10
Fam_hub_diameter	50	40
Fam_hub_blade_slot_height	10	10
Fam_number_of_blades	5	4

Also, the feature suppression status for both designs can be set to:

FEATURE	DESIGN 'A'	DESIGN 'B'
Shaft : cut_keyway	SUPPRESSED	RESUMED
Hub : protrusion_key	SUPPRESSED	RESUMED

The functional and means descriptions of figures 3.16 and 3.17 can now be re-entered, to meet the specific requirements of each design.

Chapter 4

Software Implementation

Overview

This chapter outlines the data structures, algorithms and principles used for the software implementation of the Variant Methodology. The intention here is not to meticulously describe the line-by-line execution of each procedure, but to outline the methods used to achieve a computer-based implementation of the methodology proposed. A listing of the software code is, however, given in Appendix I.

4.1 Objectives of the Software

In essence, the software presented here covers the latter three stages of the Variant Methodology (figure 3.11, section 3.3), namely to:

a) link the, already created, variant CAD models to a parts oriented database (or structure)
b) create a function-based representation of this design family, and
c) create database records (or instances) for each member of a family of related products.

Consequential objectives of this software therefore also include the capability to:

a) use a hybrid Parts Tree and Function Family Tree structure to represent Conceptual and Embodiment design,
b) use the principles of the Variant Method to represent modifiable detailed designs,
c) make use of existing Variant and Parametric solid modelling systems to realise the detailed designs,
d) store the combined Conceptual, Embodiment and Detailed designs together, as a 'Generic Master Model', from which instances (the family members) can be created.

A key issue with regard to the applicability of this research is its industrial relevance and usability, particularly for small to medium enterprises, undergoing the transition of accepting computer-based design tools. This requires the software to act as an automated interface to the solid modelling packages, allowing the modification of existing detailed designs to be fully integrated with the reuse of conceptual and embodiment design.

4.2 Software Solution

The above objectives require the software to be designed from a users-viewpoint, i.e. from the information or 'Process Flow' prescribed by the Variant Methodology, which is represented in figure 4.1.

4.3 Achievable Solutions

This Process Flow can be broken down into realisable tasks (solutions to the objectives) that must be embodied in the software. They are:

1. Create a User-Interface to Create and Edit a Parts Tree structure,
2. Allow Parameter Names and Values and Feature Suppression to be displayed and edited for each part,
3. Allow a relevant CAD model to be 'linked' to each node in the Parts Tree,
4. Allow this CAD model to be modified according to changes in the Parameters and Feature Suppression Status, outlined in (2),
5. Create a User-Interface to Create and Edit a Function Family Tree structure,
6. Allow Part to Function Relationships to be created,
7. Allow the Parts (Means) Tree to be regenerated as a Parts Oriented Function/Means Tree,
8. Allow the Function Family Tree to be regenerated as a Function Oriented Function/Means Tree,
9. Allow Instances to be created from this 'Generic Instance' (the Master Model),
10. Create Auxiliary commands, such as:

    Output manufacturing drawings for a given part, Create a solid rendering, Output the Parts Tree, Function Family Tree and Part or Function-Oriented Function Means Tree.

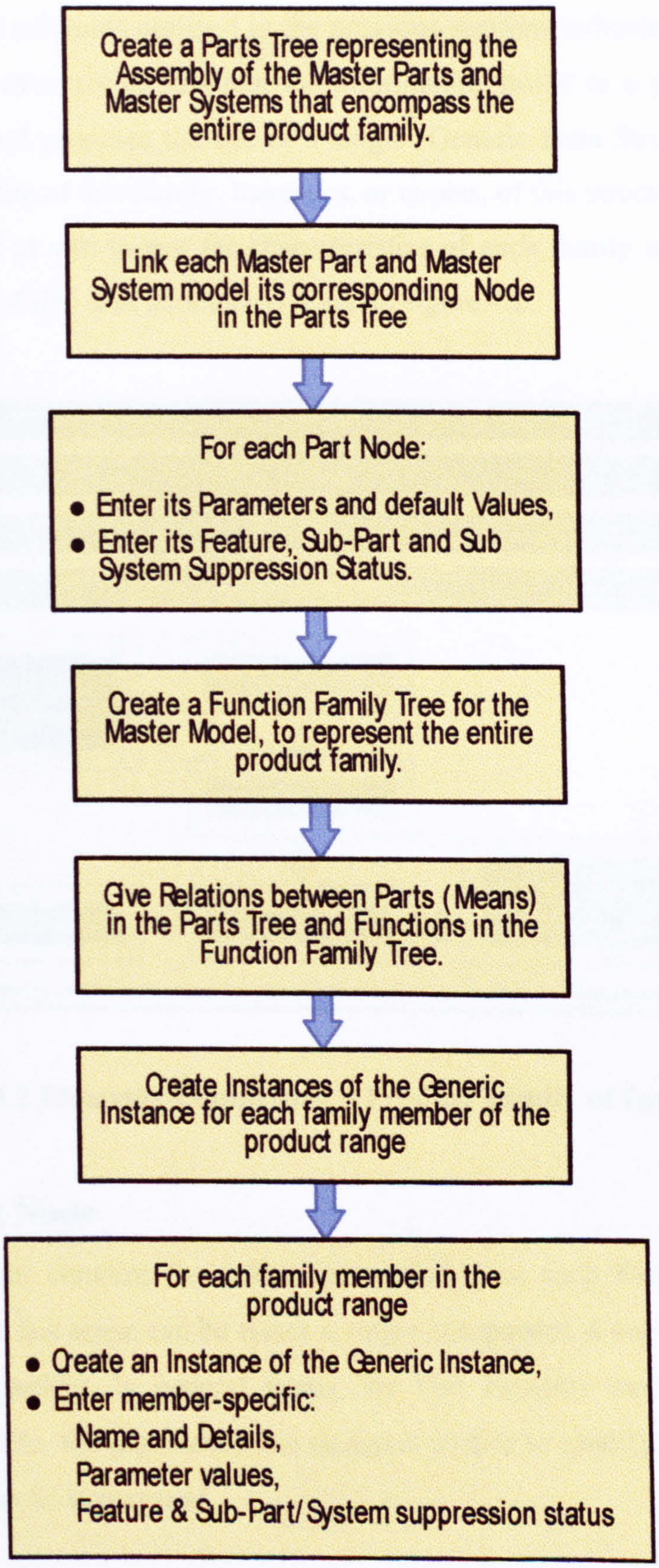


Figure 4.1 General Process Flowchart

4.3 Data Structures

The objectives and solutions outlined in the previous sections indicate the need for a well-defined data structure to maintain the information stored in a product family. The Variant Method proposes the use of a single 'Generic Data Structure' to store the skeleton structure of this family. Instances, or copies, of this structure can then be created and varied at will to suit the characteristics of each family member. In this respect, a schema of this data structure is given in figure 4.2.

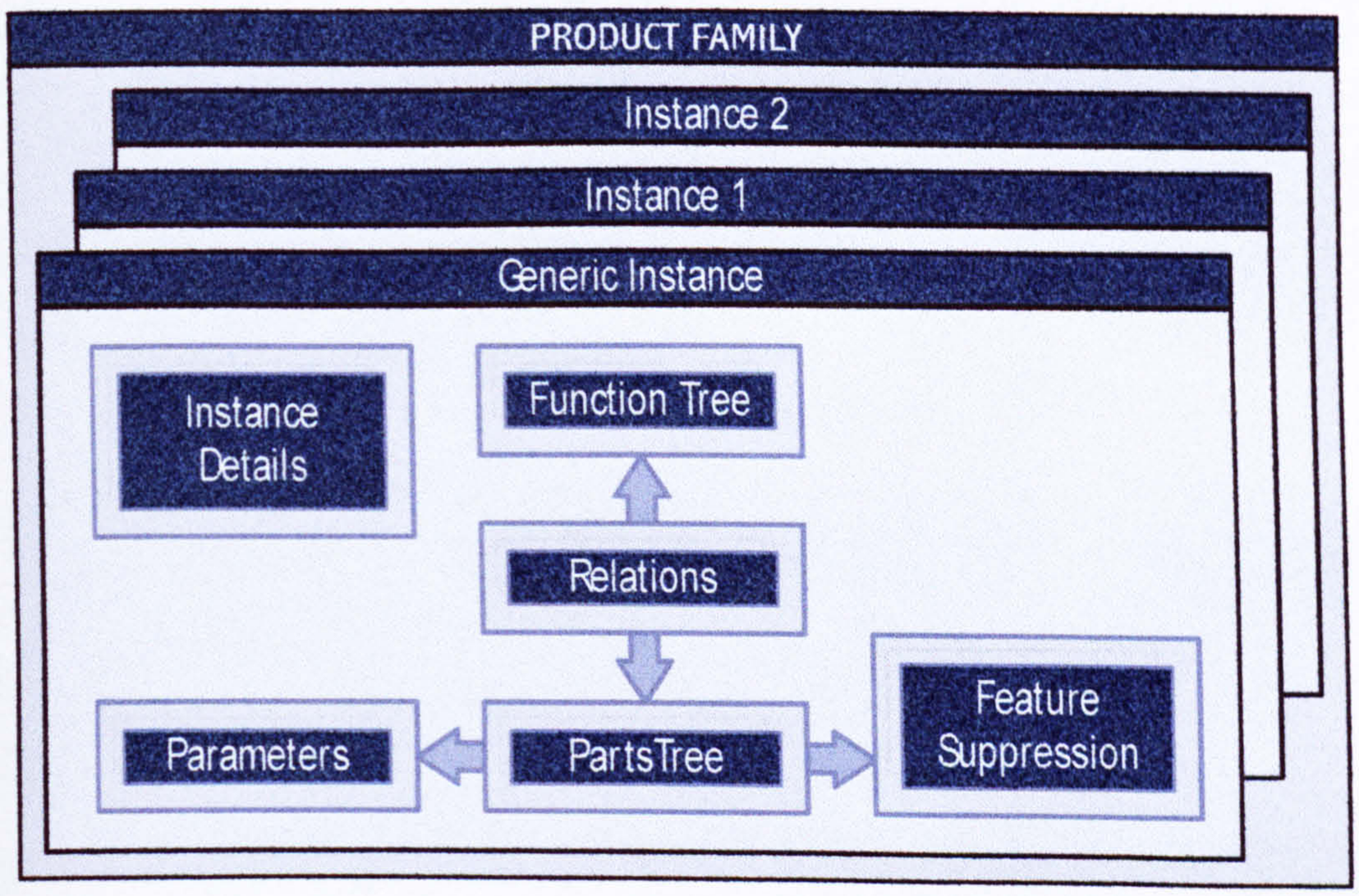


Figure 4.2 Elements Comprising a Product Family of Instances

4.3.1 The Part Node

This data structure contains the information relating to each Part of the Master Design. A Part in this sense can be either a single component, a sub-assembly or the full assembled product. In general terms, the Part structure stores a link to its respective CAD file, the parameters the designer wishes to modify in that part, and which features can be suppressed.

4.3.2 Parts Tree

This is essentially a linked list of Part Nodes, Linked by their index in a one-dimensional array. For example, a Part's Children can be expressed as a list of their array indices.

Part - Member Name	Type
Name	string
Number of Parents	integer
List of Parent's Part ID's	list of integers
Number of Children	integer
List of Children's Part ID's	list of integers
Number of Suppressed Entities	integer
List of Suppressed Entity Names	list of integers
Number of Function Relations	integer
List of Function Relations Function Tree ID's	list of integers
Level	integer
CAD filename	string
CAD filetype	string
Number of Parameters	integer
List of Parameter Names	list of strings
List of Parameter Values	list of doubles
List of Parameter Units	list of strings
Parts Suppression Status	boolean

Figure 4.3 Part Node Data Structure

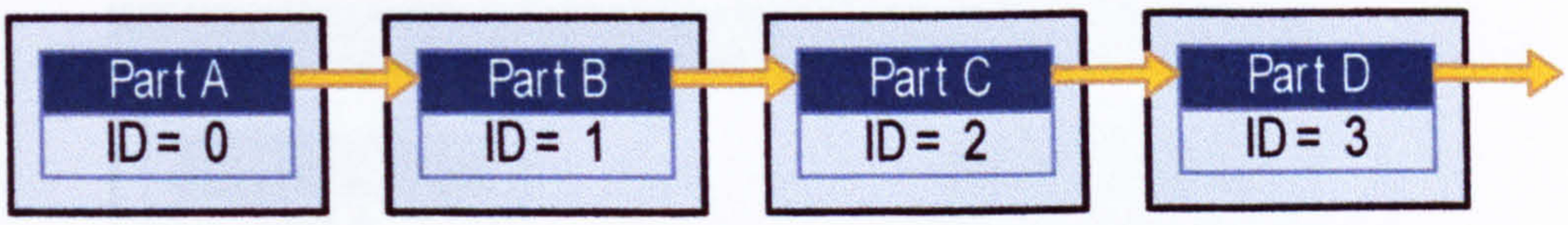


Figure 4.4 Parts Tree Data Structure

Function - Member Name	Type
Name	string
Number of Parents	integer
List of Parent's Part ID's	list of integers
Number of Children	integer
List of Children's Part ID's	list of integers
Number of Means Relations	integer
List of Means Relations Function Tree ID's	list of integers
Level	integer

Figure 4.5 Function Node Data Structure

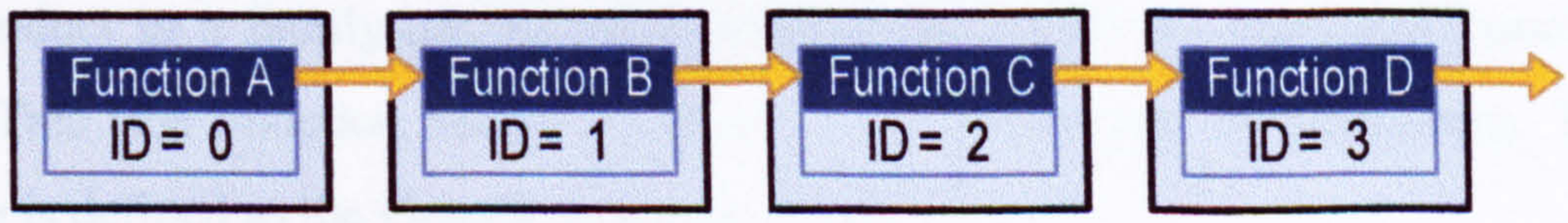


Figure 4.6 Function Family Tree Data Structure

4.3.3 Function Node

This is a limited version of the Part Node, comprising only Name, Parent, Child and Relations data.

4.3.4 Function Family Tree

Again, this is similar in construction to the Parts Tree, as a linked-list of Function Nodes.

4.3.5 Hybrid Function/Means(Parts) Tree

Defining relationships between the Parts Tree Nodes and Function Family Tree Nodes allows these structures to be regenerated to directly show the Function Means Tree, in either a Parts Oriented or Function Oriented fashion. Figure 4.7 outlines the data structure representing these relations.

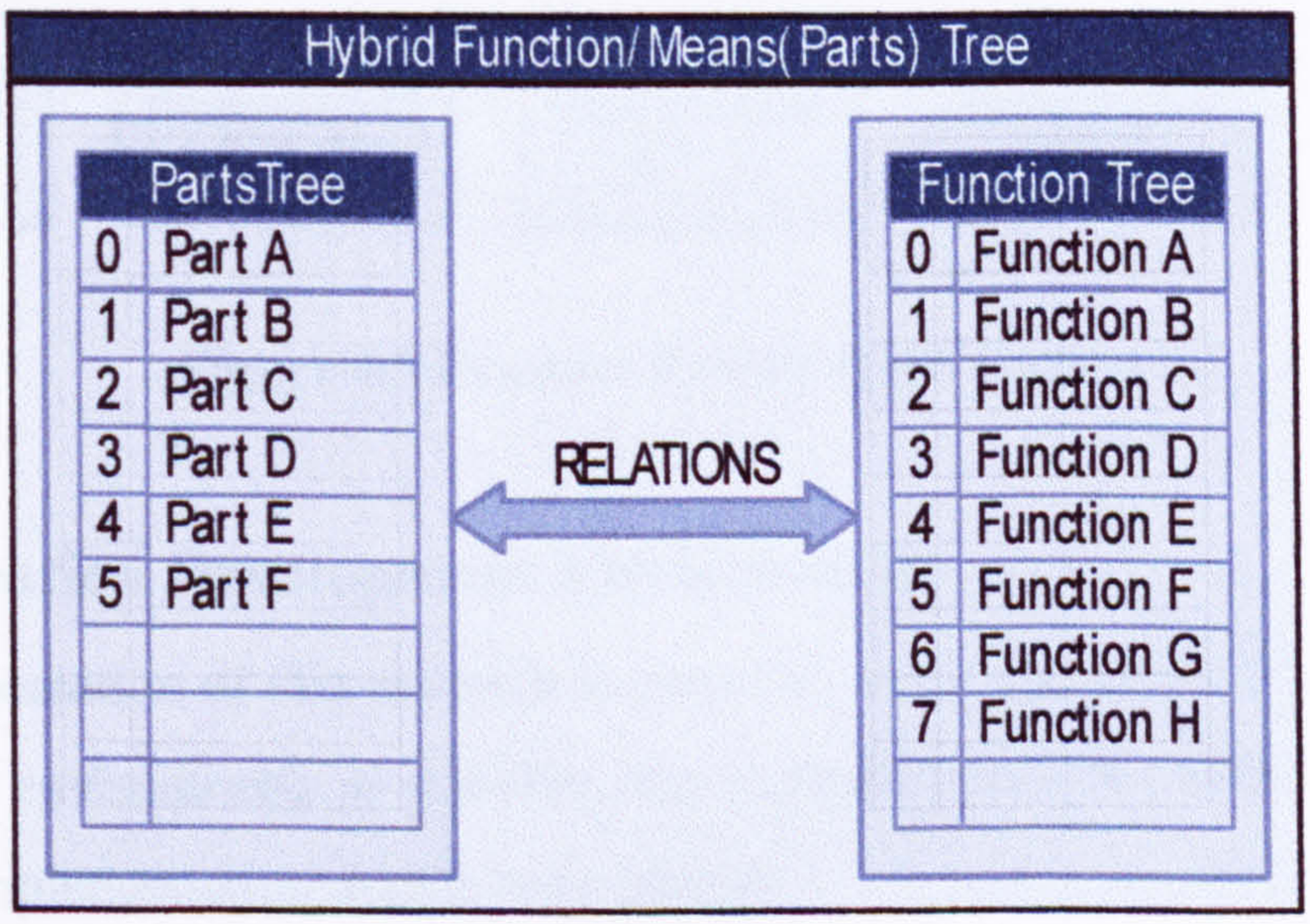


Figure 4.7 Hybrid Function/Means Data Structure

4.3.6 The Generic Instance

Each Product in a family can be represented by its combined Parts and Function Family Tree (the Function Means Tree), its CAD models and its parameters. This structure is defined as the Generic Instance, figure 4.8.

Generic Instance	Type
Name	string
Instance ID	string
Designer's Name	string
Date	string
Text Description	string
Parts Tree	list of Parts
Function Tree	list of Functions

Figure 4.8 Generic Instance Data Structure

4.3.7 The Product Family

Finally, an entire family of products (or instances of the Generic form) can be stored as a linked list of instance types, as shown below:

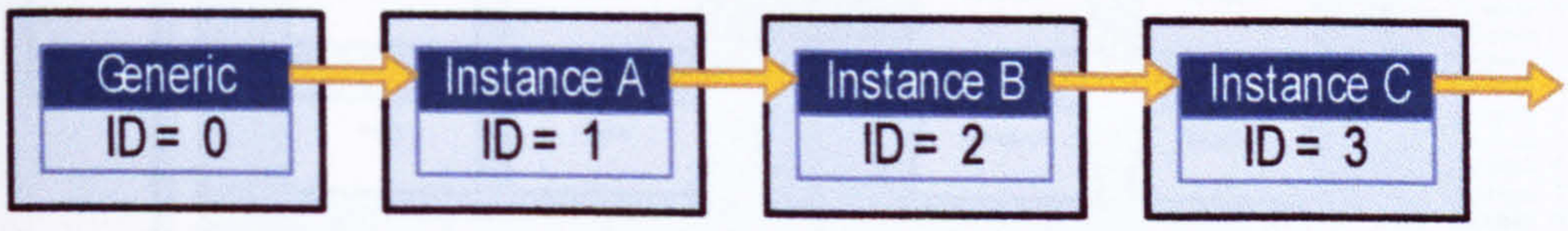


Figure 4.9 Product Family Data Structure

4.4 Application Development Environment

The implementation of this research is intended to be applicable to as wide a range of industrial environments as possible, this especially includes SME's. Therefore the following implementation details were chosen:

Operating System - Microsoft Windows 95/98/NT4

Development Language - Microsoft Visual Basic

CAD software - Pro/ENGINEER rel20

- SolidWorks 98Plus
- Autodesk Mechanical Desktop 1.2 and above.

4.5 User Interface

Using the appropriate buttons in the toolbar, the user can create and move part nodes to form a parts tree. When a given node is selected, its particular Parameters and Feature Suppression Status are displayed, and can be edited. The linked CAD file can also be updated to accommodate parameter modifications using the methods discussed in the following section.

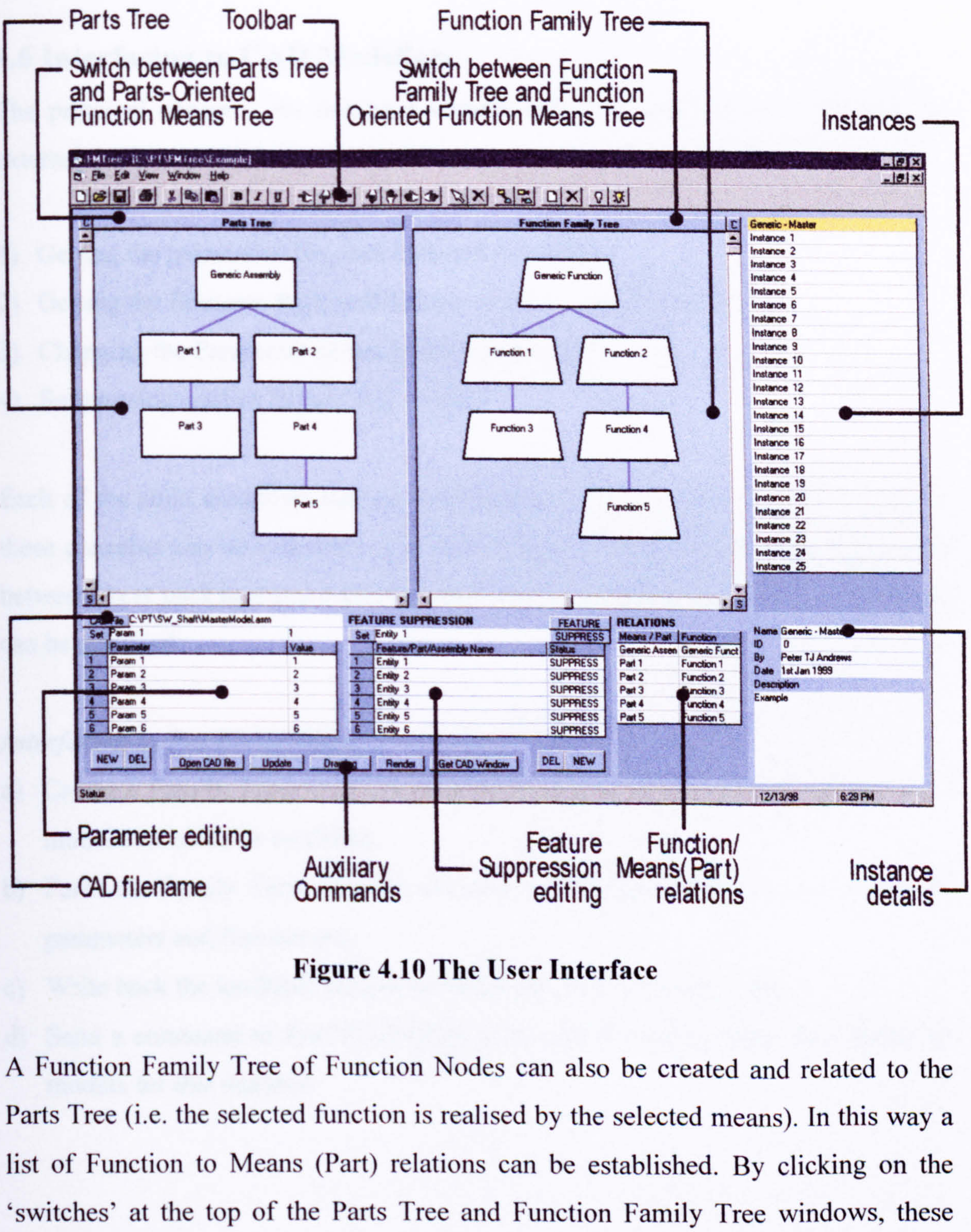


Figure 4.10 The User Interface

A Function Family Tree of Function Nodes can also be created and related to the Parts Tree (i.e. the selected function is realised by the selected means). In this way a list of Function to Means (Part) relations can be established. By clicking on the 'switches' at the top of the Parts Tree and Function Family Tree windows, these

structures can be combined to display a Parts Oriented Function Means Tree and a Function Oriented Function Means Tree, respectively.

This process constitutes the creation of the 'Generic Instance' from which all child instances can be modelled. New instances are created by selecting a 'Parent' instance in the 'Instances list' from which the child will be an exact copy of the selected parent. The Product Name and other details can also be entered for each instance.

4.6 Interfacing to CAD Modellers

The principal concerns for linking a commercial CAD package to a database or external application include:

1) Getting the parameters for each Part and Assembly,
2) Getting the Features, Parts and Sub-assemblies to be Suppressed,
3) Changing the Parameter values in the CAD model,
4) Suppressing entities in the CAD model.

Each of the solid modellers that are implemented here, vary with regard as to how these concerns can be overcome. The following is a brief description of how a link between these packages and a development language, such as Microsoft Visual Basic can be achieved.

Interfacing to Pro/ENGINEER

a) Create a Family Table in Pro/ENGINEER containing all elements of the CAD model that are to be modified,
b) Parse the Family Table into the software application (database) and extract the parameters and features etc.
c) Write back the modified parameter values etc, to the Family Table,
d) Send a command to Pro/ENGINEER to re-load the family table and update the models for this instance.

Interfacing to SolidWorks

a) Enter Parameter names, values and features etc. to be changed.
b) Use the SolidWorks API commands to directly modify the parameter values and suppression status of features, parts and assemblies.

Interfacing to Mechanical Desktop

a) Save a Parameter List in Mechanical Desktop (this is a built-in function of the CAD software)
b) Parse this file into the software and select the desired parameters to modify,
c) Write back the modified Parameters List
d) Send commands to Mechanical Desktop to re-load the Parameters List and Update.

4.7 An Illustrative example – the Propeller Shaft

This section illustrates the use of the software for the Propeller Shaft example, used originally to illustrate the Variant Methodology in Chapter 3.

Issues regarding the determination of what comprises the Master Parts and Systems has been covered in the previous chapter, and will not be discussed here. At this stage it is assumed that these parts have been established and created as variant CAD models.

The following five screenshots show the Generic Master Parts and Systems, linked to their respective nodes in the Parts Tree:

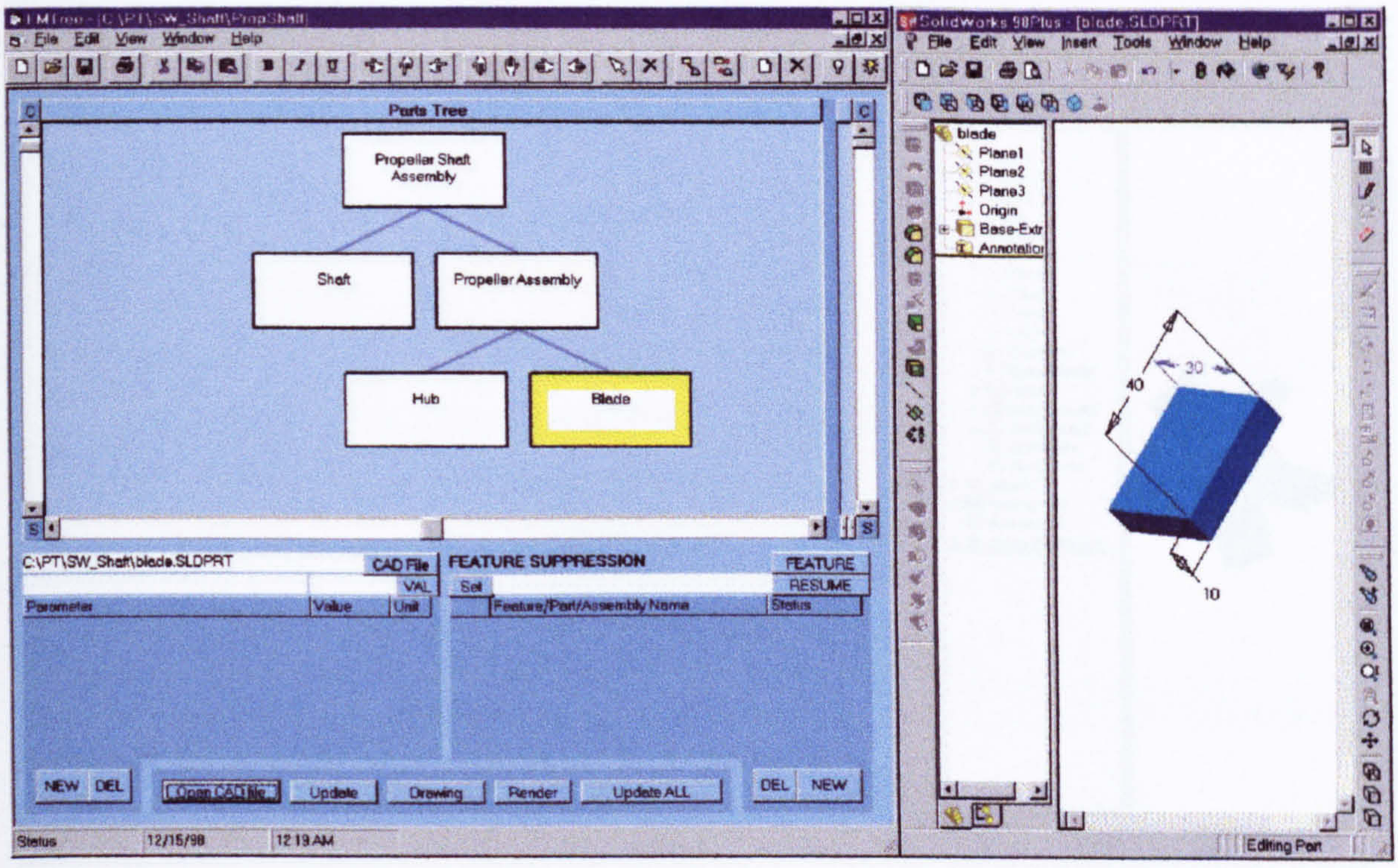


Figure 4.11a The Master Blade Part,

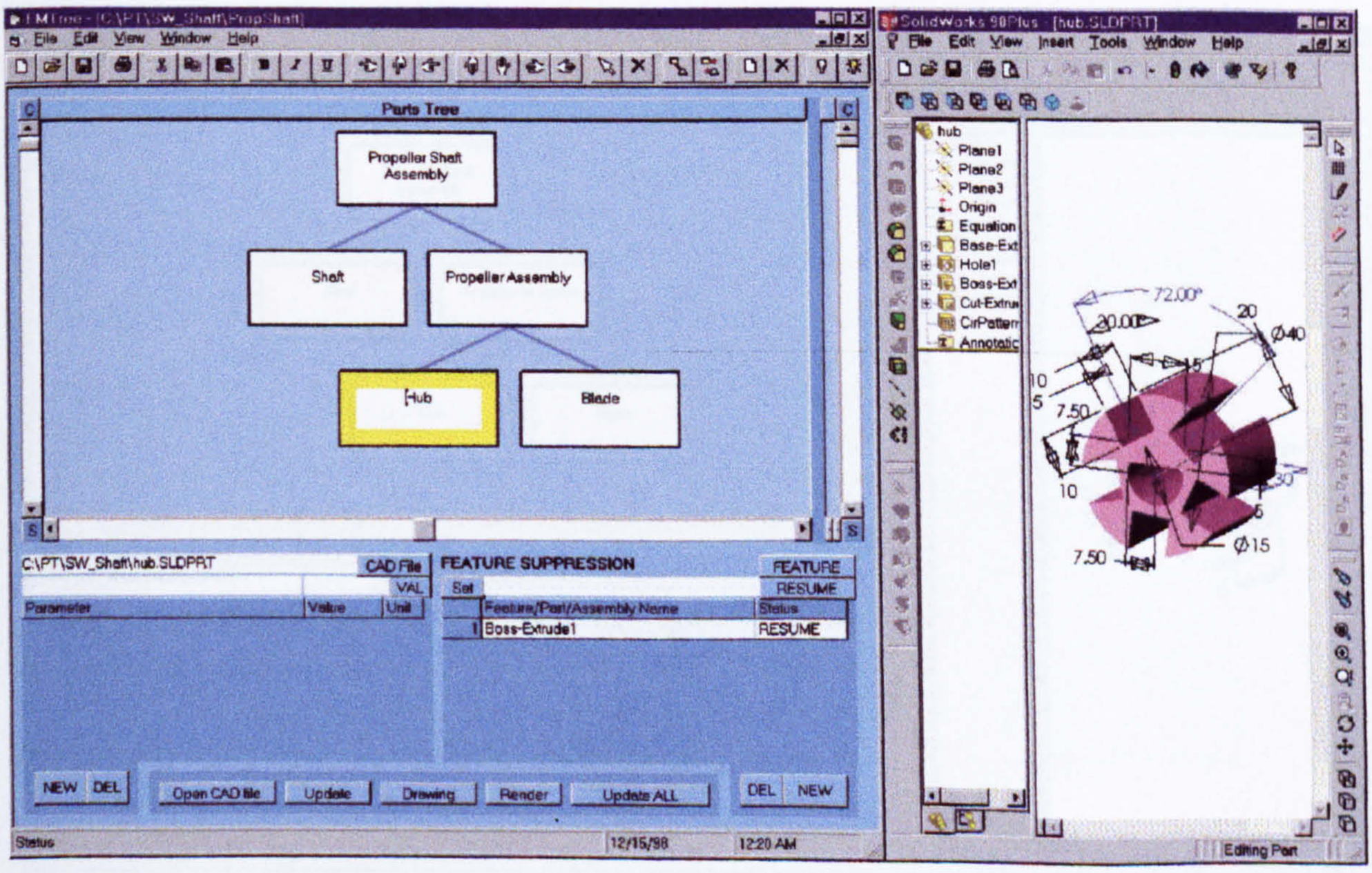


Figure 4.11b The Master Hub Part,

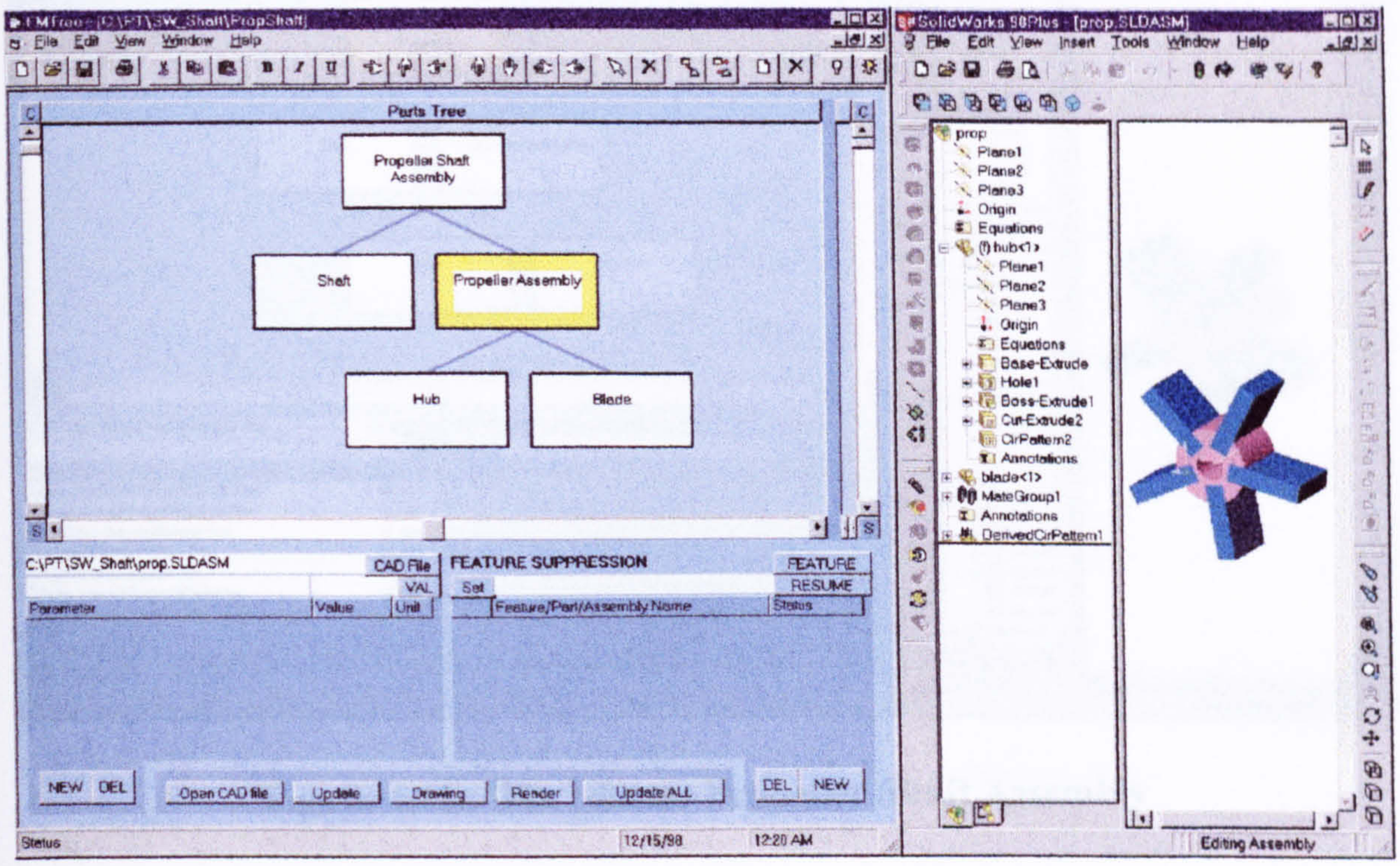


Figure 4.11c (top) The Master Propeller Assembly,

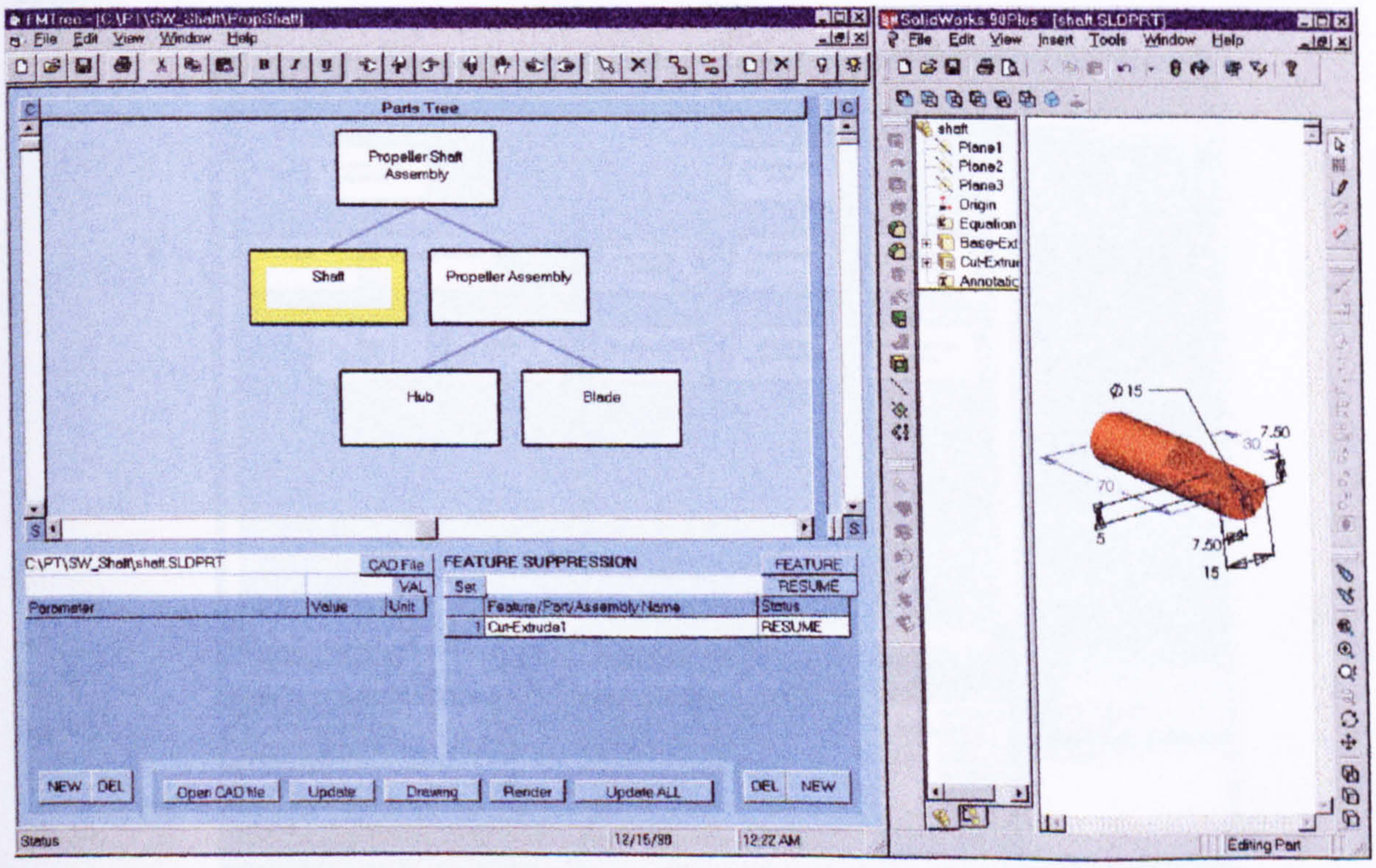


Figure 4.11d (above) The Master Shaft Part,

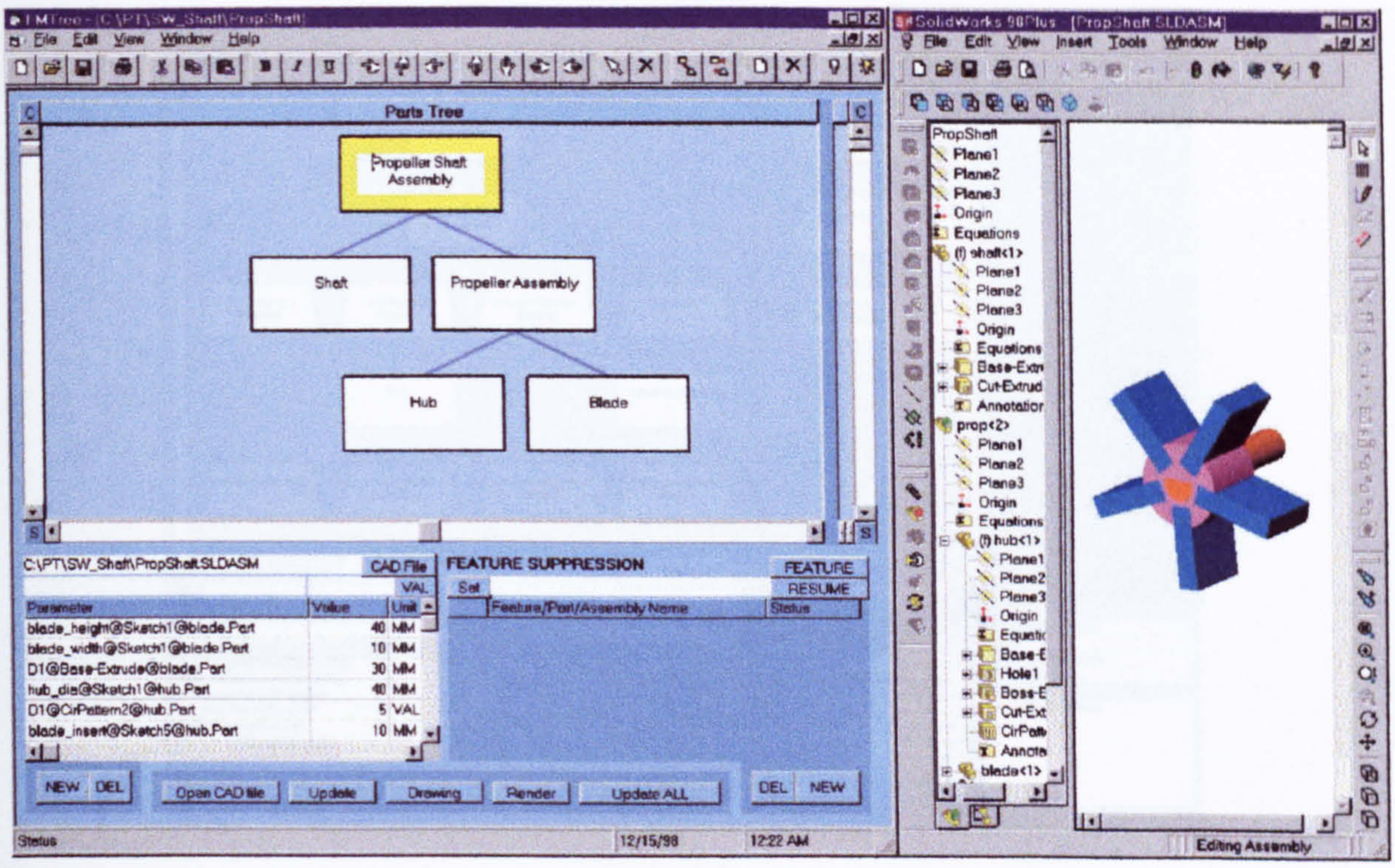


Figure 4.11e The Master Propeller Shaft Assembly

Figure 4.12 shows the Function Family Tree for the Propeller Shaft design, followed by figure 4.13, which gives the Parts Oriented representation of the Function Means Tree.

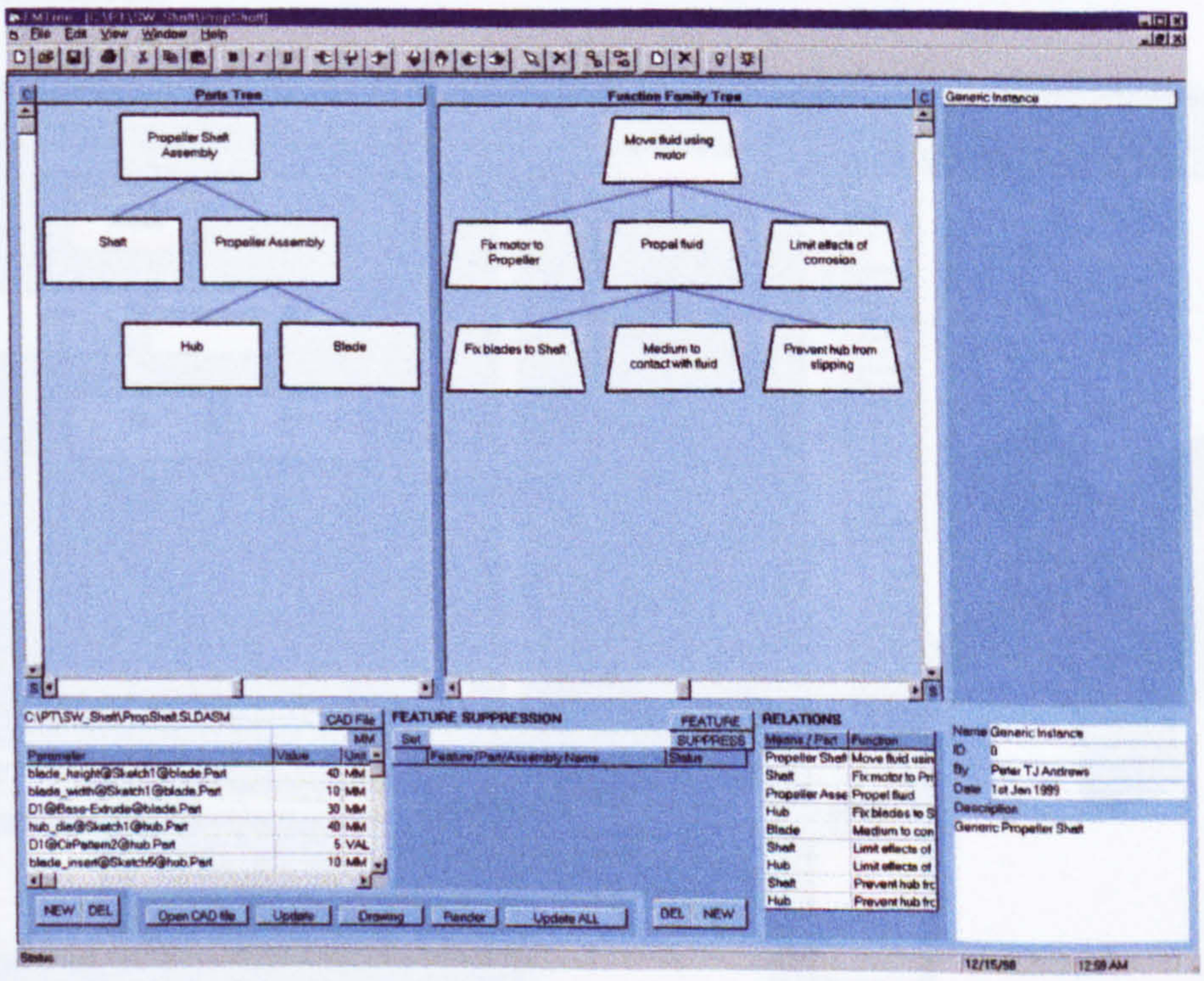


Figure 4.12 The Function Family Tree

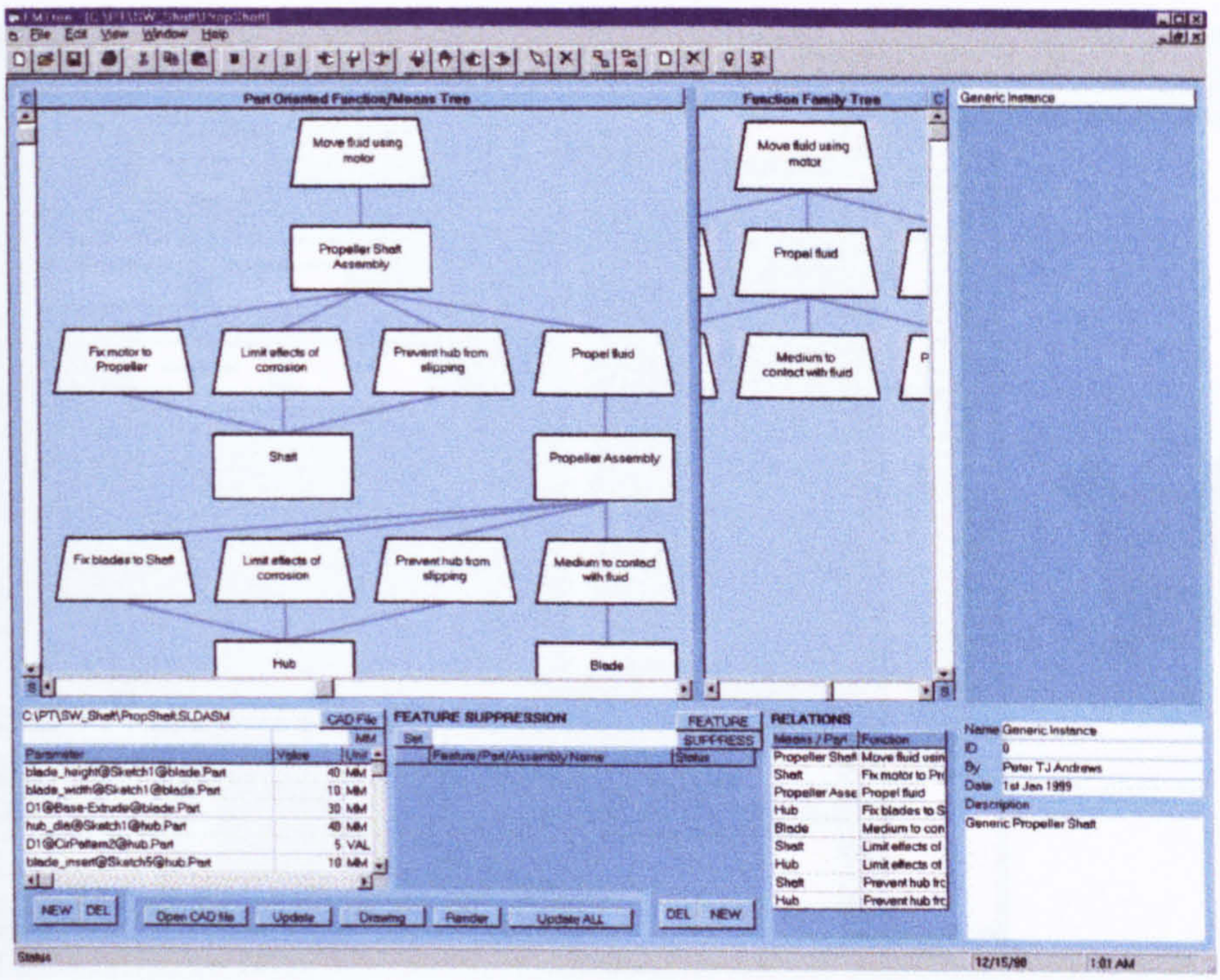


Figure 4.13 The Parts Oriented Function Means Tree

With the Generic Instance complete, two instances can be created to represent Designs A and B (see section 3.4.1). ***This is achieved by simply instancing the Generic Instance and changing a few parameter values***. Figures 4.14 and 4.15 show these instances.

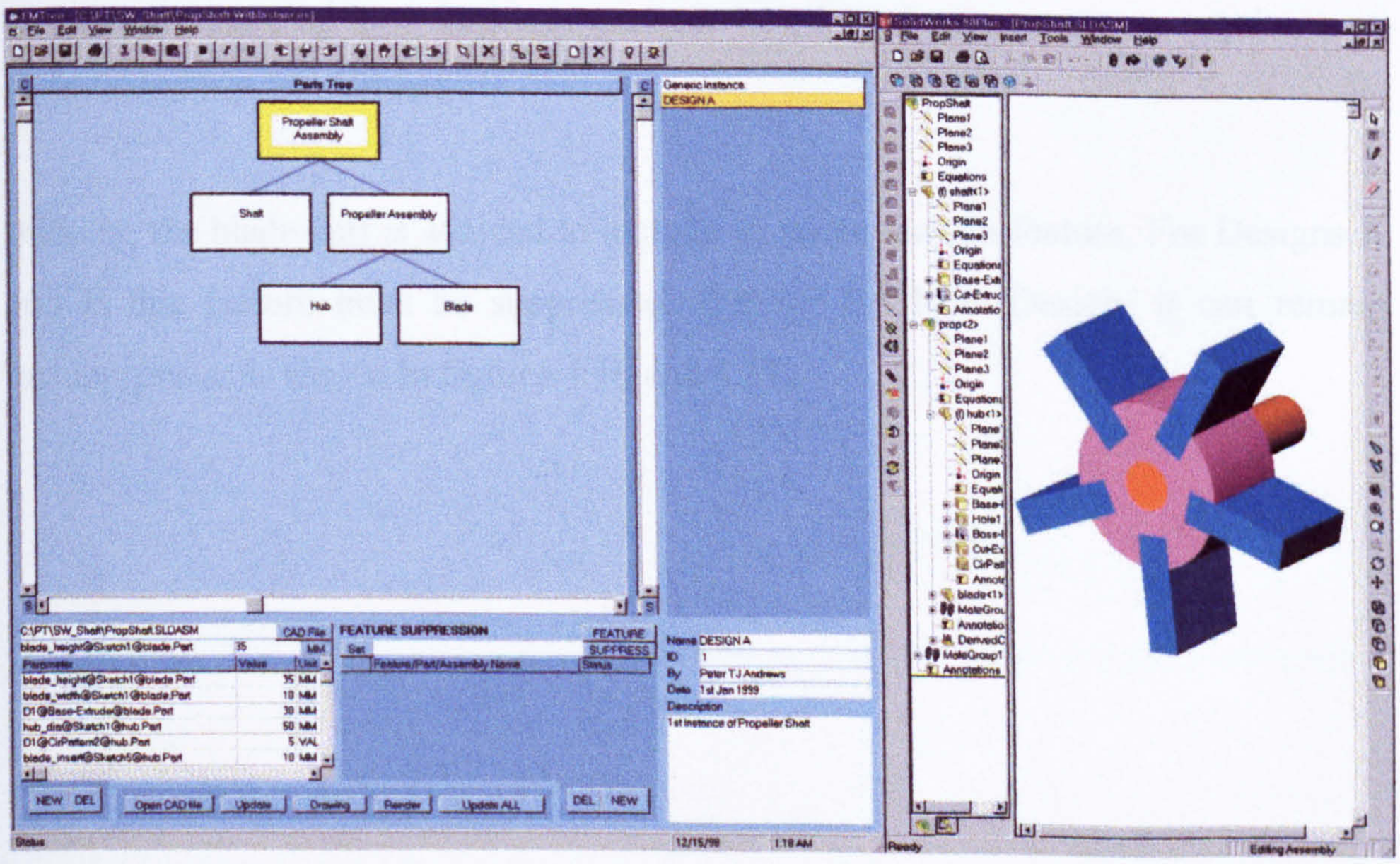


Figure 4.14 Instance for Design A

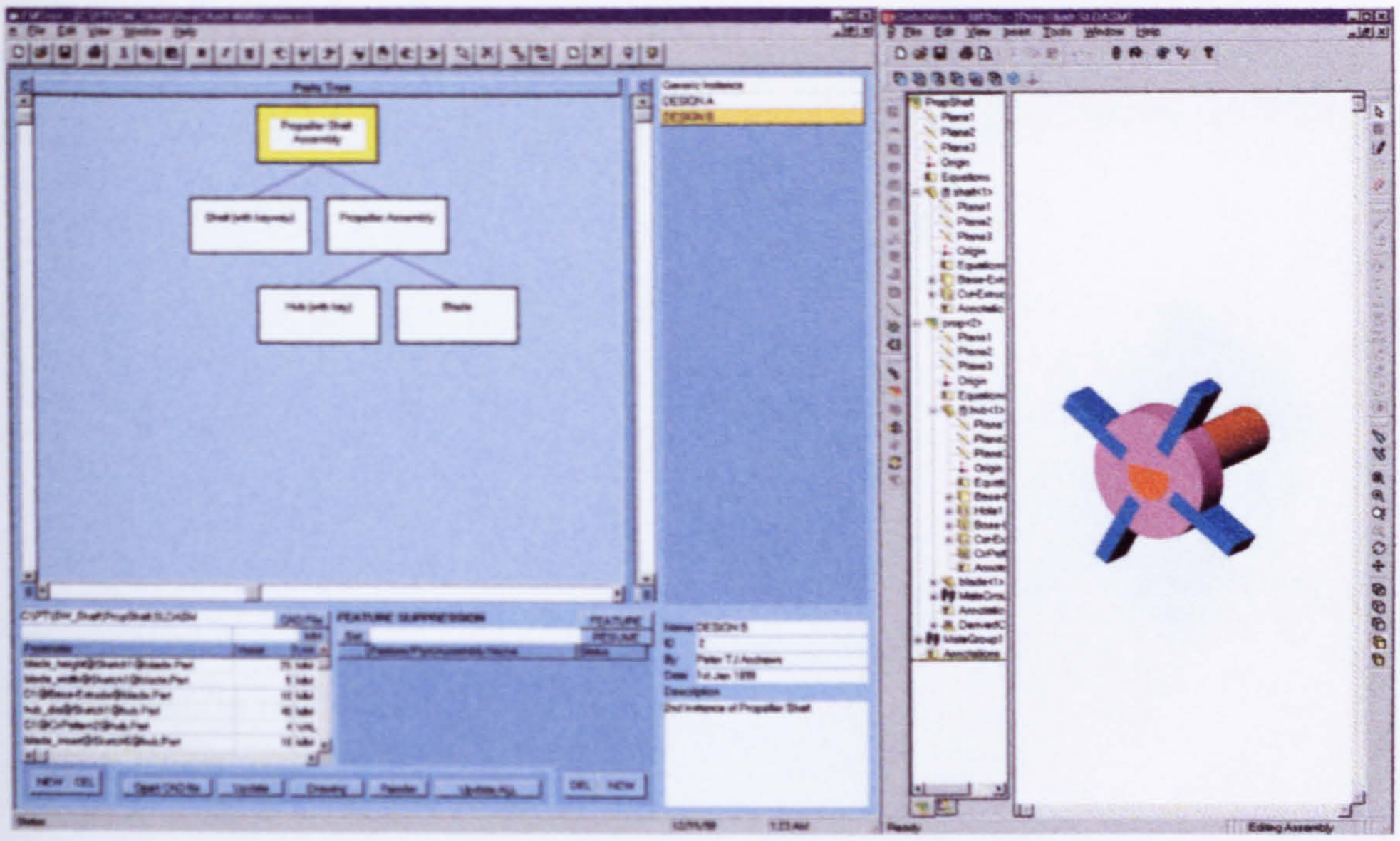


Figure 4.15 Instance for Design B

4.7.1 Reuse of the Propeller Shaft Model for a Modified Blade

The example, so far, shows how the software can be used to efficiently and more rapidly computerise a family of similar designs. However, these designs will probably have to be reused, and hence modified at some later stage. The following is an example showing how the blade part can be adapted to meet a new requirement, i.e. to contain a fin-element.

Initially, the blade part is adapted to include an additional fin feature. For Designs A and B this feature must be suppressed. But for the New Design, it can remain unsuppressed, as shown in figures 4.16 and 4.17.

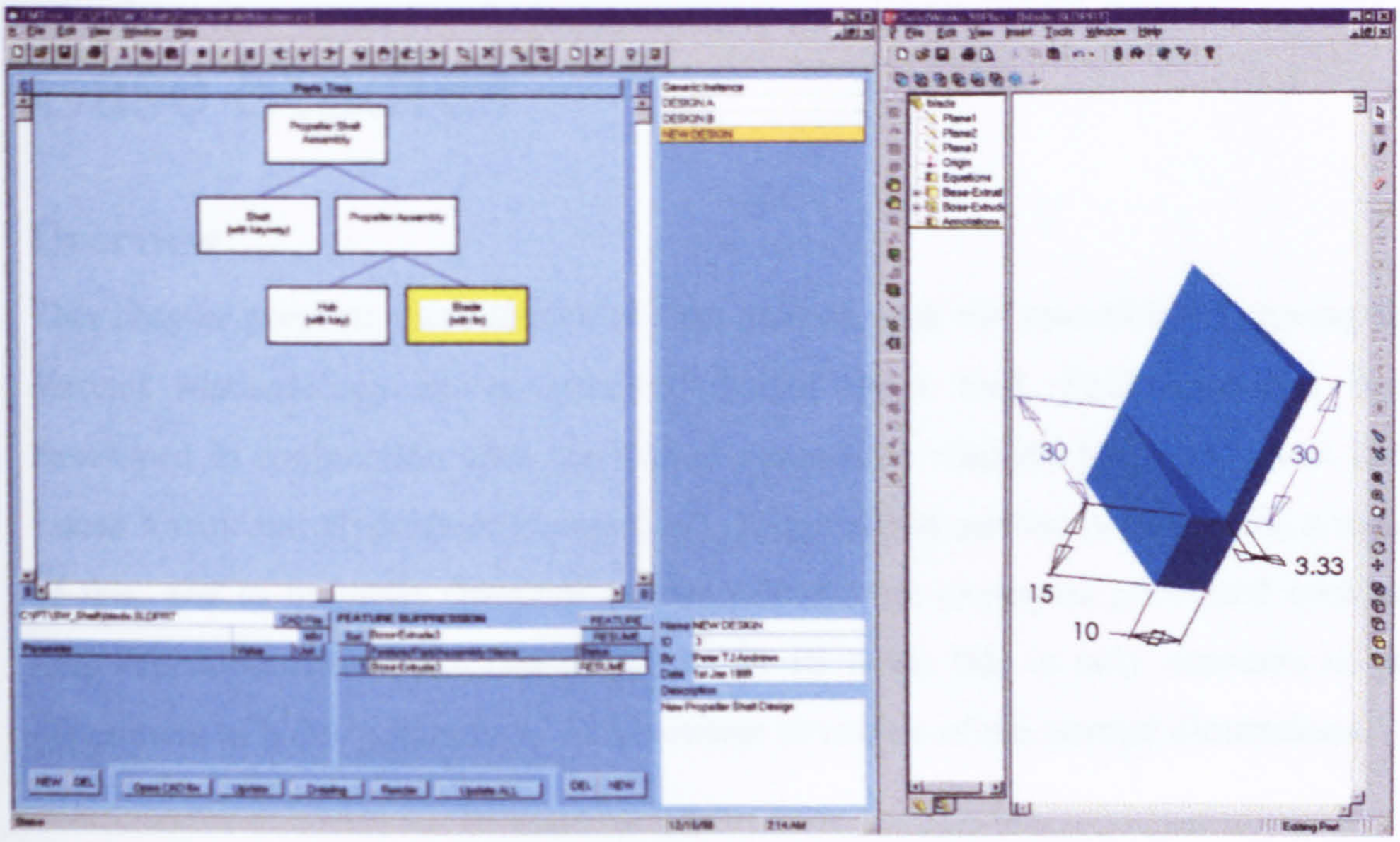


Figure 4.16 A Modified Blade Part (with fin feature)

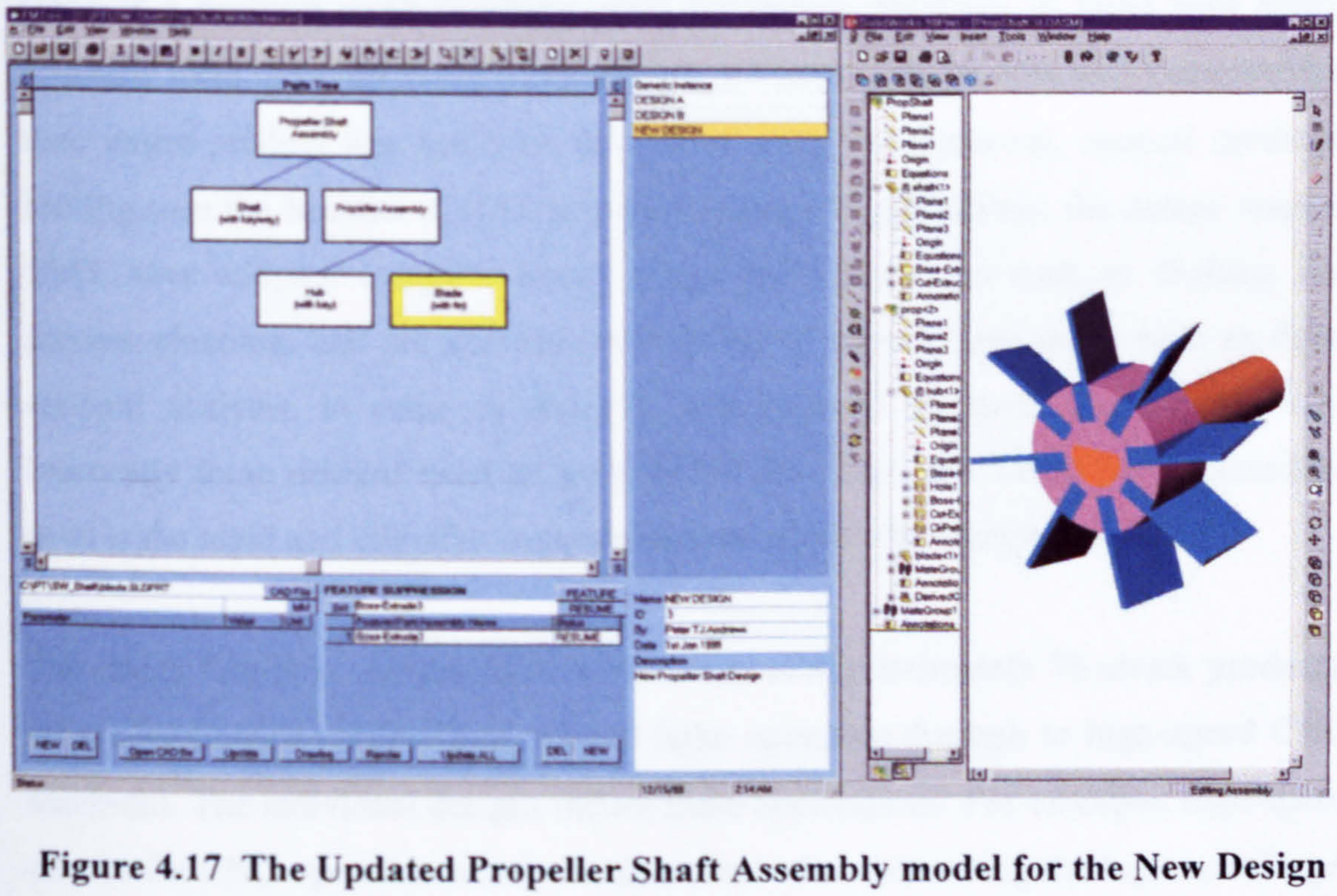


Figure 4.17 The Updated Propeller Shaft Assembly model for the New Design

Chapter 5

Case Studies

Overview

This chapter presents three industrial case studies with the intention of proving the Variant Methodology as an effective design reuse tool. Each case has been developed in conjunction with the related companies: Guindy Machine Tools Ltd., Lucas Varity and Hydroflow Europe Ltd., using real, in service products. As a result of this, and to maintain company confidentiality, the examples presented here are only *representative* of the true products. In all cases this is only amounts to the adjustment of a few parameters, with a minor deviation of the normal dimensions.

5.1 Guindy Machine Tools Ltd. Lathe Chuck Family

Lathe chucks are the main product of Guindy Machine Tools, of Madras, India. GMT is a medium sized company, with production facilities in three sites across southern India. As with many companies of a similar size, in Asia and Europe alike, their entire product line has been developed using conventional, manual methods. Having seen the benefits of CAD enjoyed by large organisations, the design team at GMT have adopted computer-based design tools for areas such as drafting and process planning, and are currently attempting to expand into areas such as finite element analysis, in order to quantify and improve on their existing products. Inherently these designs exist as paper-based drawings, and hence their immediate need is the rapid and effective computerisation of this vast design family.

The chuck family is comprised of a collection of approximately 70 chuck products, whose applications range from manual lathe operation through to high-speed CNC machines. The individual designs reflect these applications. For example, high-speed chucks for CNC operation require some means of counteracting the high centrifugal forces, whereas a manual lathe does not require this facility.

There are however, clear similarities throughout the GMT lathe chuck range, i.e. all chucks are connected to a 'Body' part and all hold the job using a number of 'Jaws'.

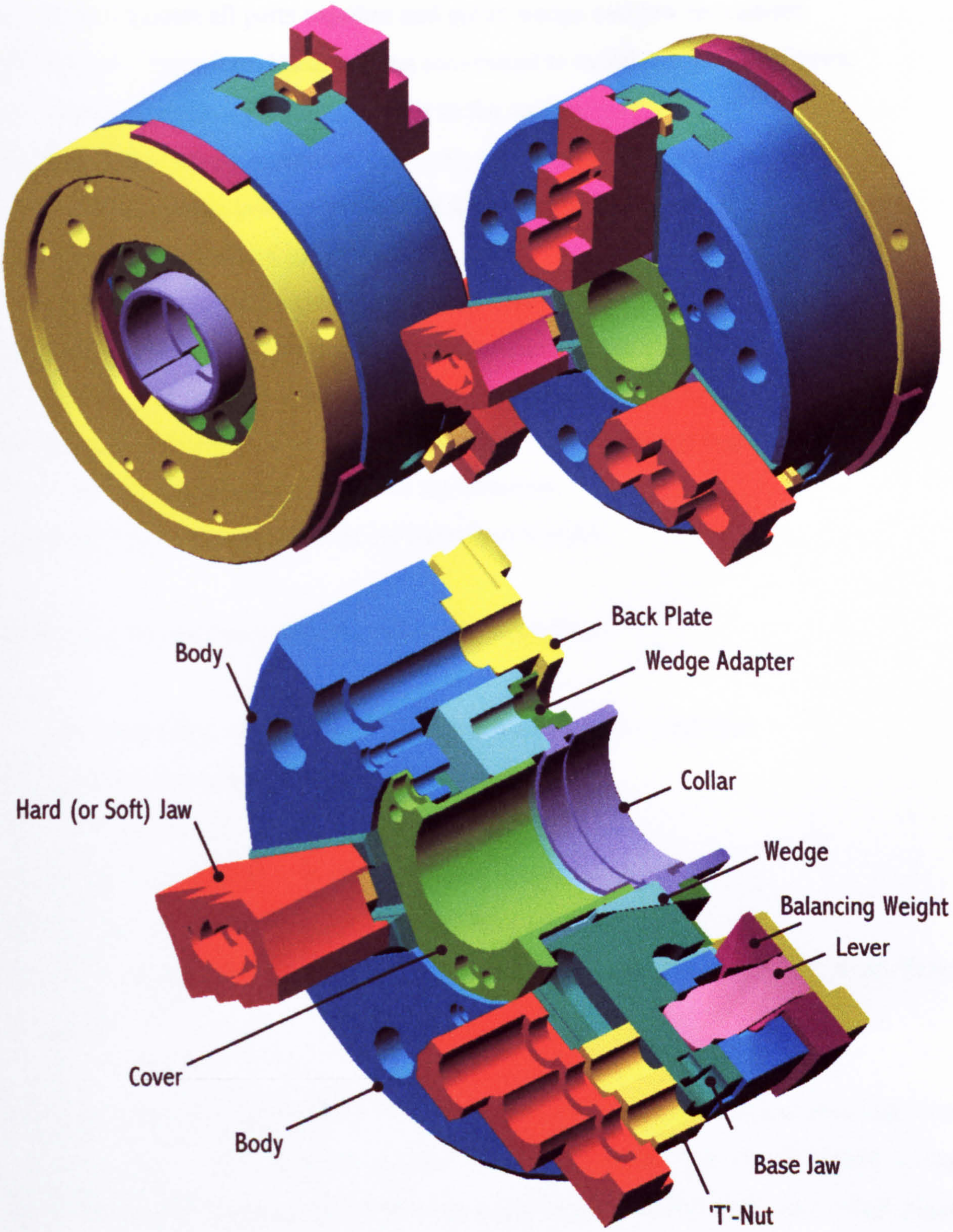


Figure 5.1.1 The Generic Section View of the GMT Lathe Chuck

Hence, the chuck family can be readily modelled using the Variant Method, whilst preserving the application of each chuck type (its solution concept and embodiment) through the Function Means Tree. Figure 5.1.1 shows an example of the 'Generic Chuck'. Details of each of the components from which it is comprised are listed below. Further details, including Global Parameters, are given in Appendix II.

1) Body – Locate all parts together and guide wedge and jaw movement.
2) Wedge – Transform linear pulling movement to radial movement of jaws.
3) Wedge Adapter – Fix pulling collar to the wedge.
4) Base Jaw – Medium between the body, wedge and 'T'-nut and jaws.
5) 'T'-Nut – enable jaws to be adjusted for irregular job sizes.
6) Hard Jaw – provide a rough grip onto a job.
7) Soft Jaw – provide a distortion-free grip into a job.
8) Collar – Medium between pulling mechanism and the chuck.
9) Cover – Prevents jaws from clashing and covers front the bore.
10) Balancing Weight – Counteract centrifugal force of jaws at high speeds.
11) Lever – Link balancing weight to the base-jaw.
12) Back Plate – Guide and hold the balancing weights.

These components also yield the following assemblies:

1) Jaw Assembly – Base Jaw, 'T'-Nut, Hard Jaw and the Soft Jaw
2) Wedge Assembly – Wedge and Jaw Assembly
3) Puller Assembly – Collar and Wedge Adapter
4) Gripping Assembly – Wedge Assembly and Puller Assembly
5) Balancing Assembly – Lever and the Balancing Weight
6) Chuck Assembly – Body, Gripping Assembly, Balancing Assembly, Back Plate and Cover.

Having studied the vast collection of chuck drawings, and visited and consulted the design team at GMT Madras, it was decided to model the chuck family using Pro/ENGINEER, being a well established and reliable application. Individual piece parts were modelled and defined various 'driving' (or global) parameters, as previously mentioned. The most important decisive parameters that can be changed in the resulting Variant Model is the Number of Jaws (Num_Jaws). Figures 5.1.2, 5.1.3 and 5.1.4 show examples of this for a 2-jaw, 3-jaw and 4-jaw chuck.

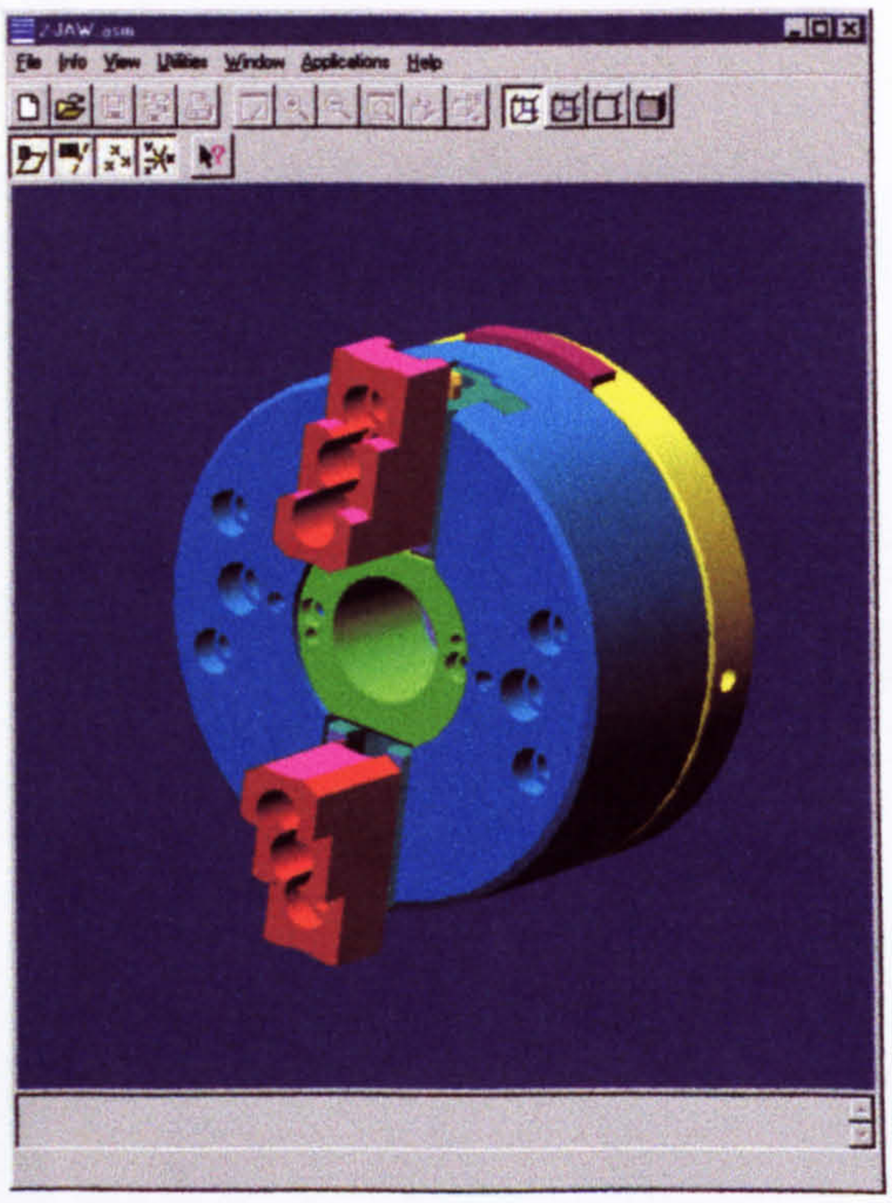


Figure 5.1.2 A 2-Jaw Chuck

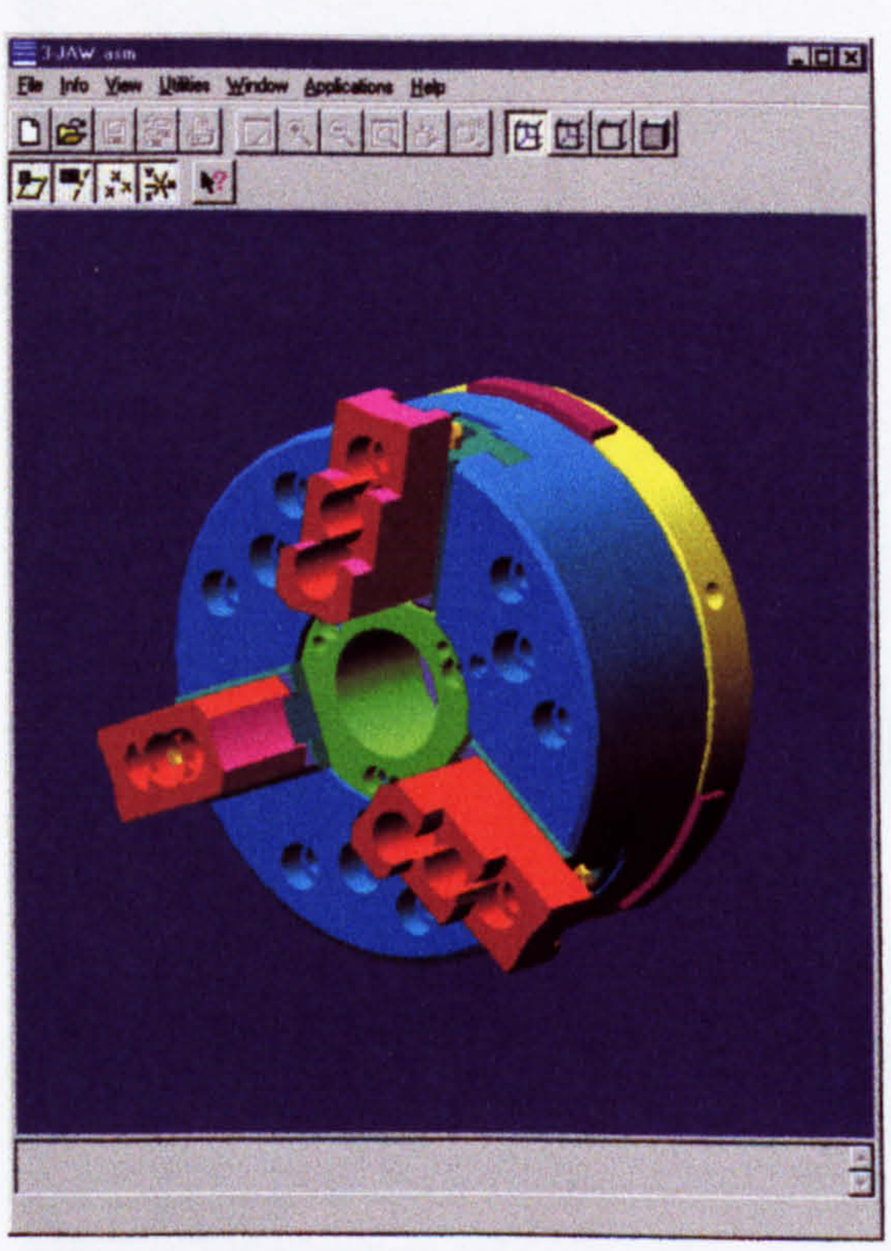


Figure 5.1.3 A 3-Jaw Chuck

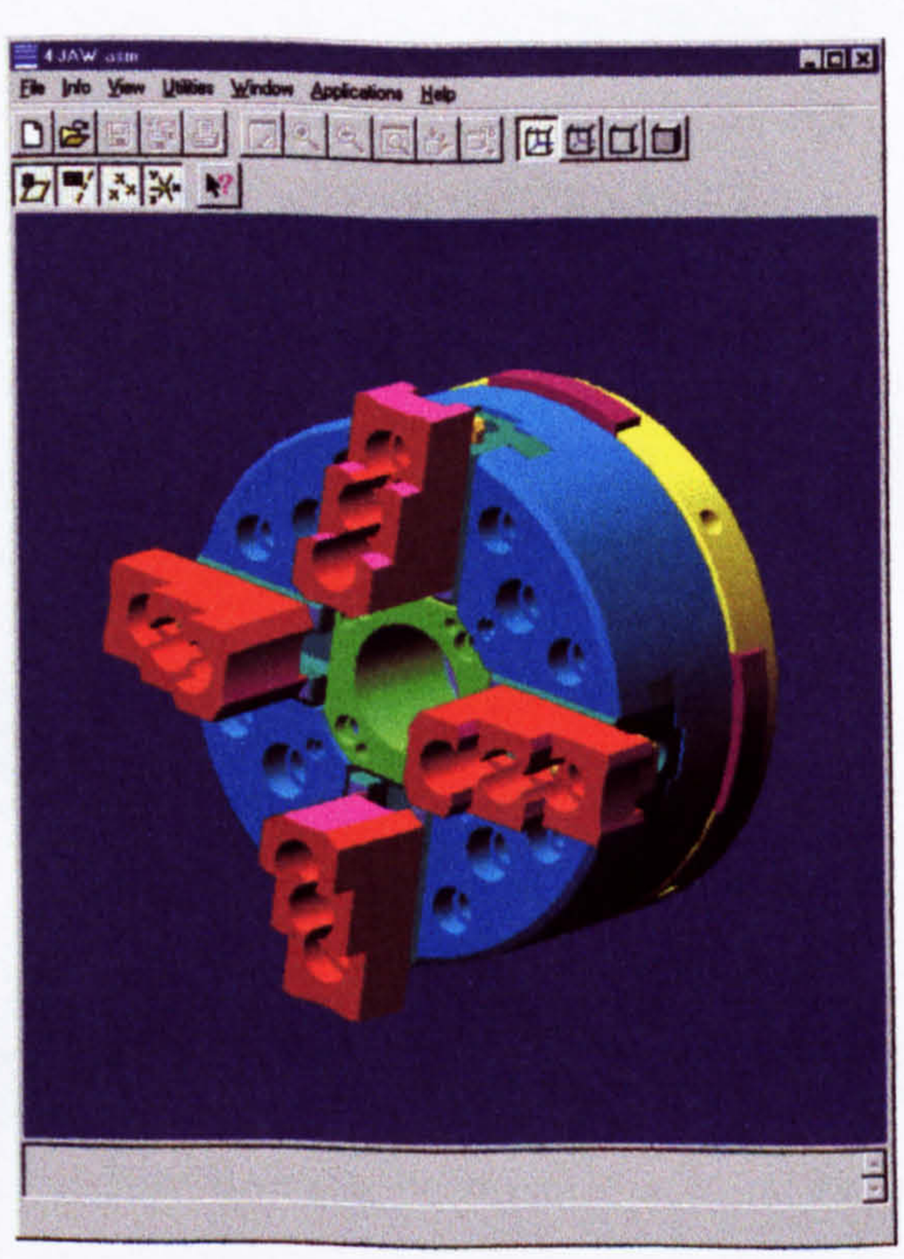


Figure 5.1.4 A 4-Jaw Chuck

Figures 5.1.5 and 5.1.6 show the generated Parts Tree and Function Means Tree structures from the FMT software.

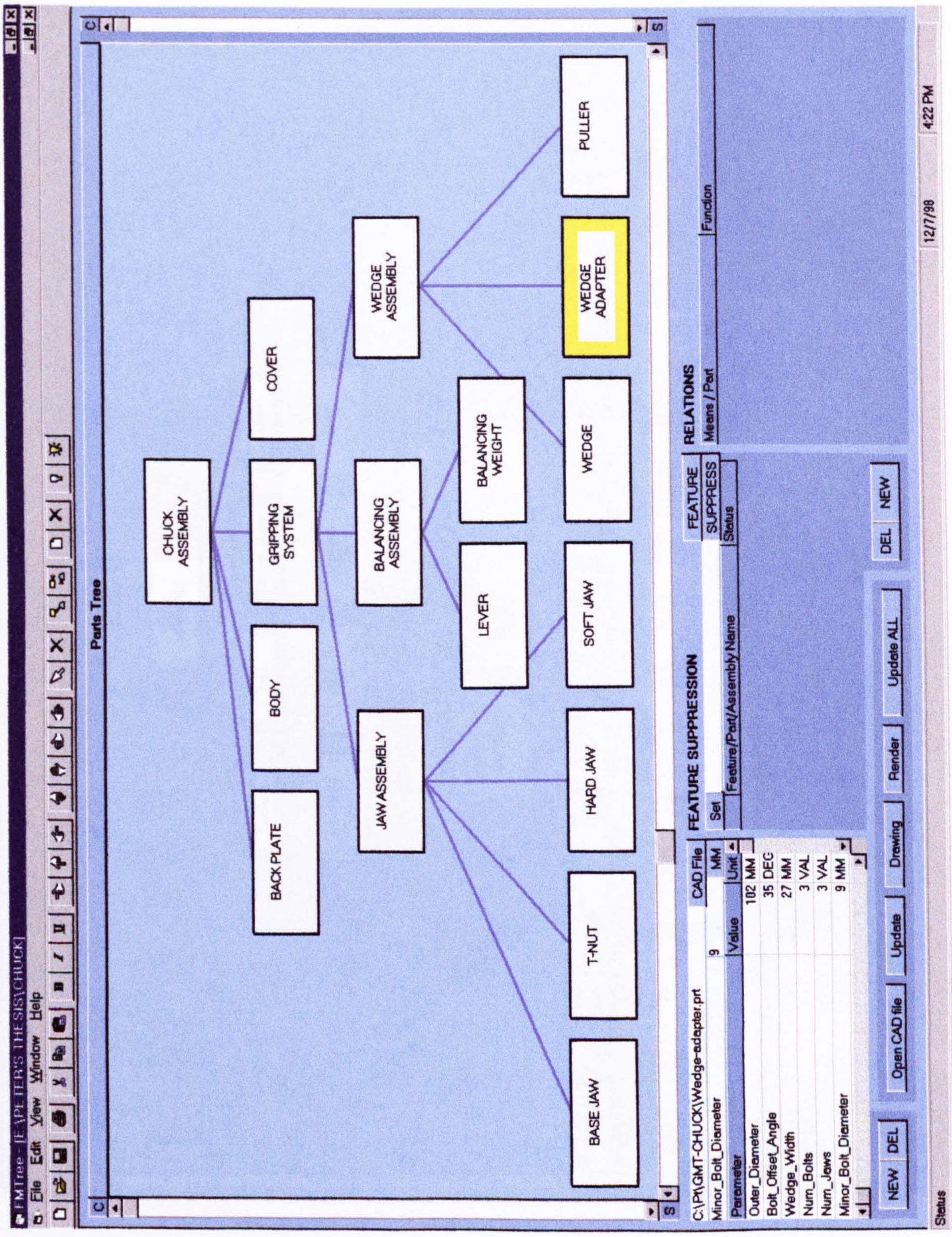


Figure 5.1.5 Parts Tree for the Generic Chuck

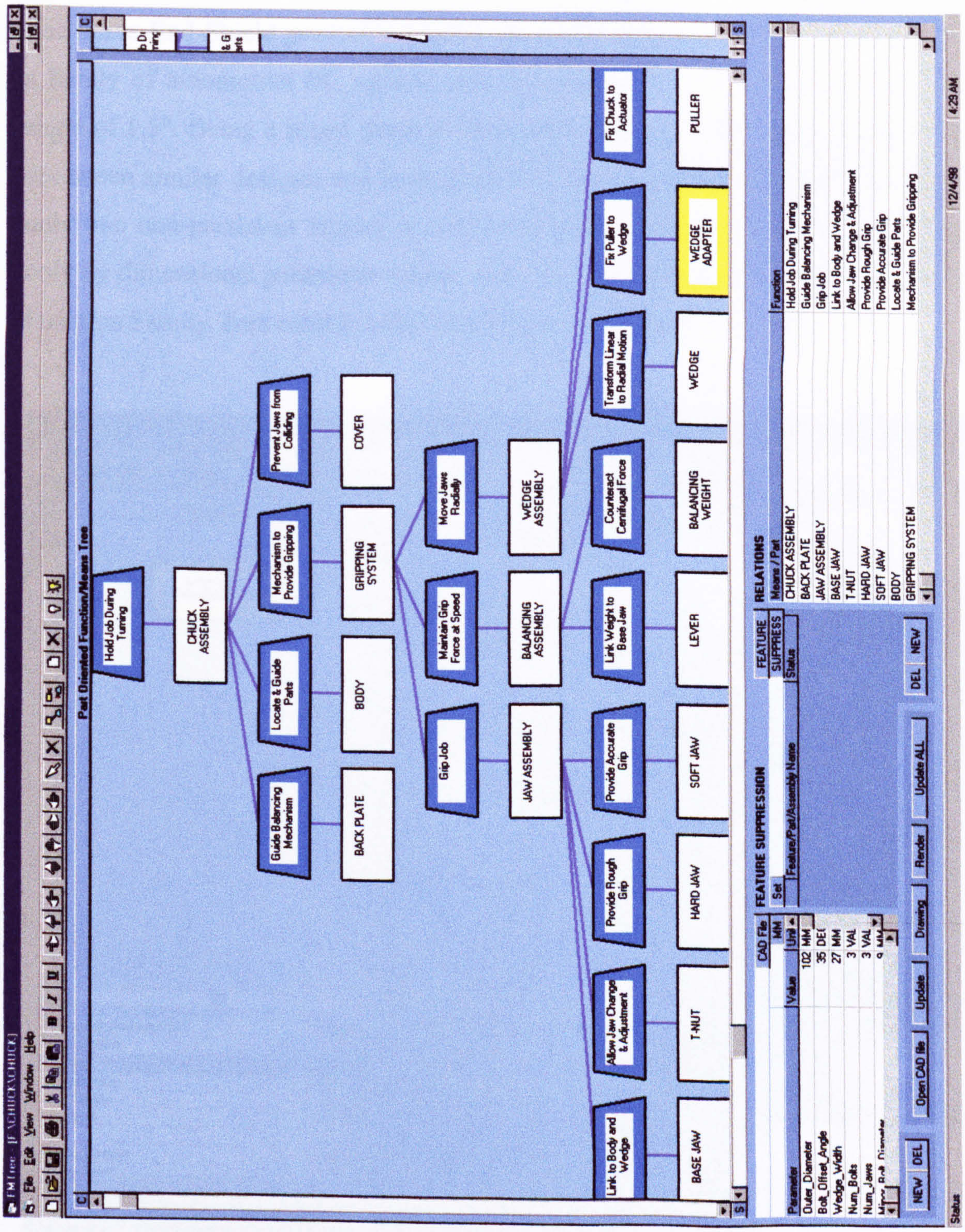


Figure 5.1.6 Function Means Tree for the Generic Chuck

5.2 Lucas Varity Drive End Shield Casting

This case study, a Lucas Varity 'Drive End Shield' casting, is used as a simple demonstration of the ease of generating new instances from a single master model. The Drive-End-Shield product is used to house the coil and support the end-shaft of a family of automotive DC motors, and is cast from S.G.Iron with a typical draft angle of 1.5°. Being a single piece product, the casting forms a combined family of seventeen similar designs, and is, in general, a simple product to model, containing only one non-persistent feature (a boss) and various persistent features that differ only by dimensional parameter values. Also, as a single part, both the Parts tree and Function Family Tree contain only one node, as is shown in figure 5.2.1.

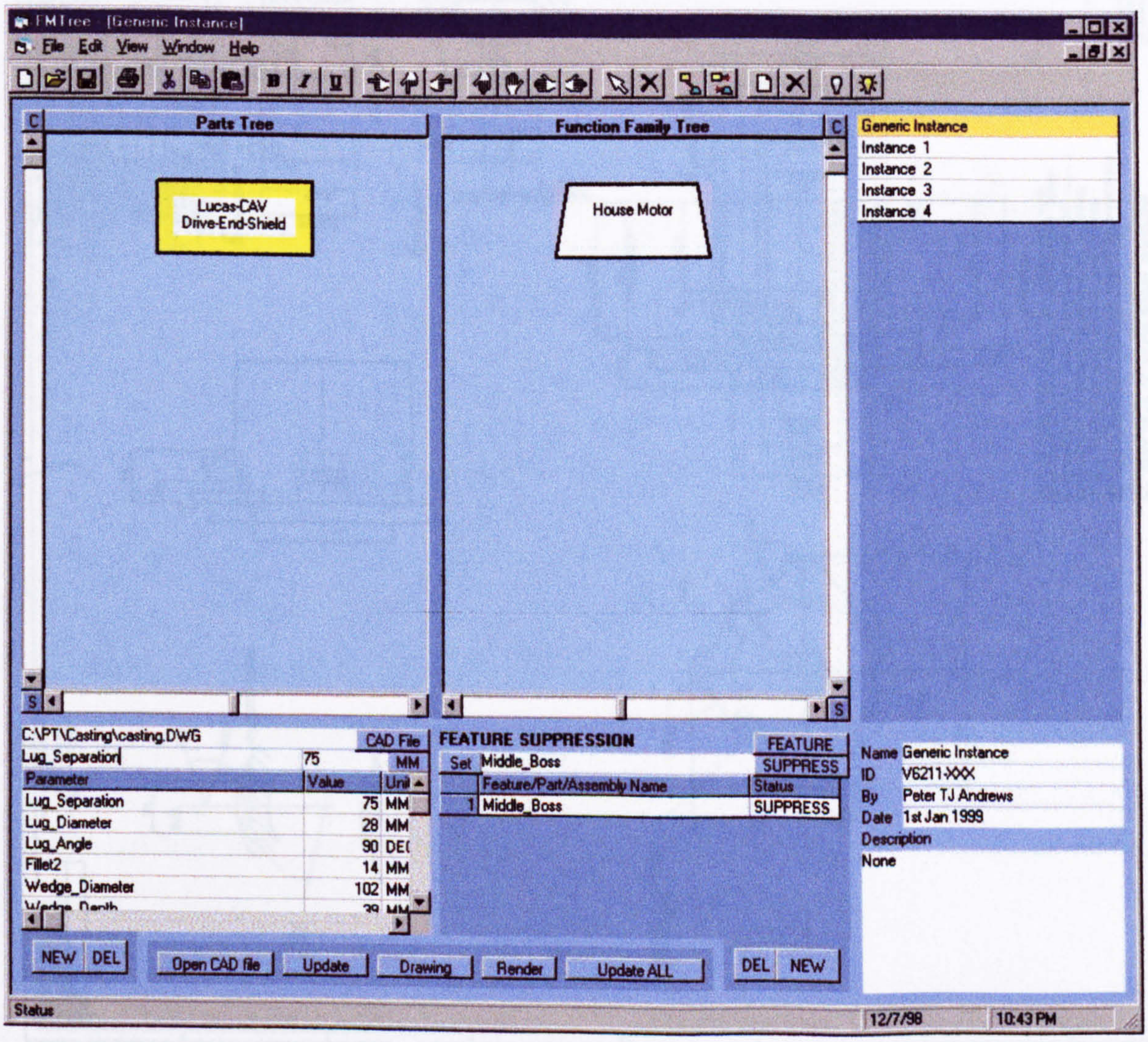


Figure 5.2.1 – FMT application for the Drive-End-Shield Casting

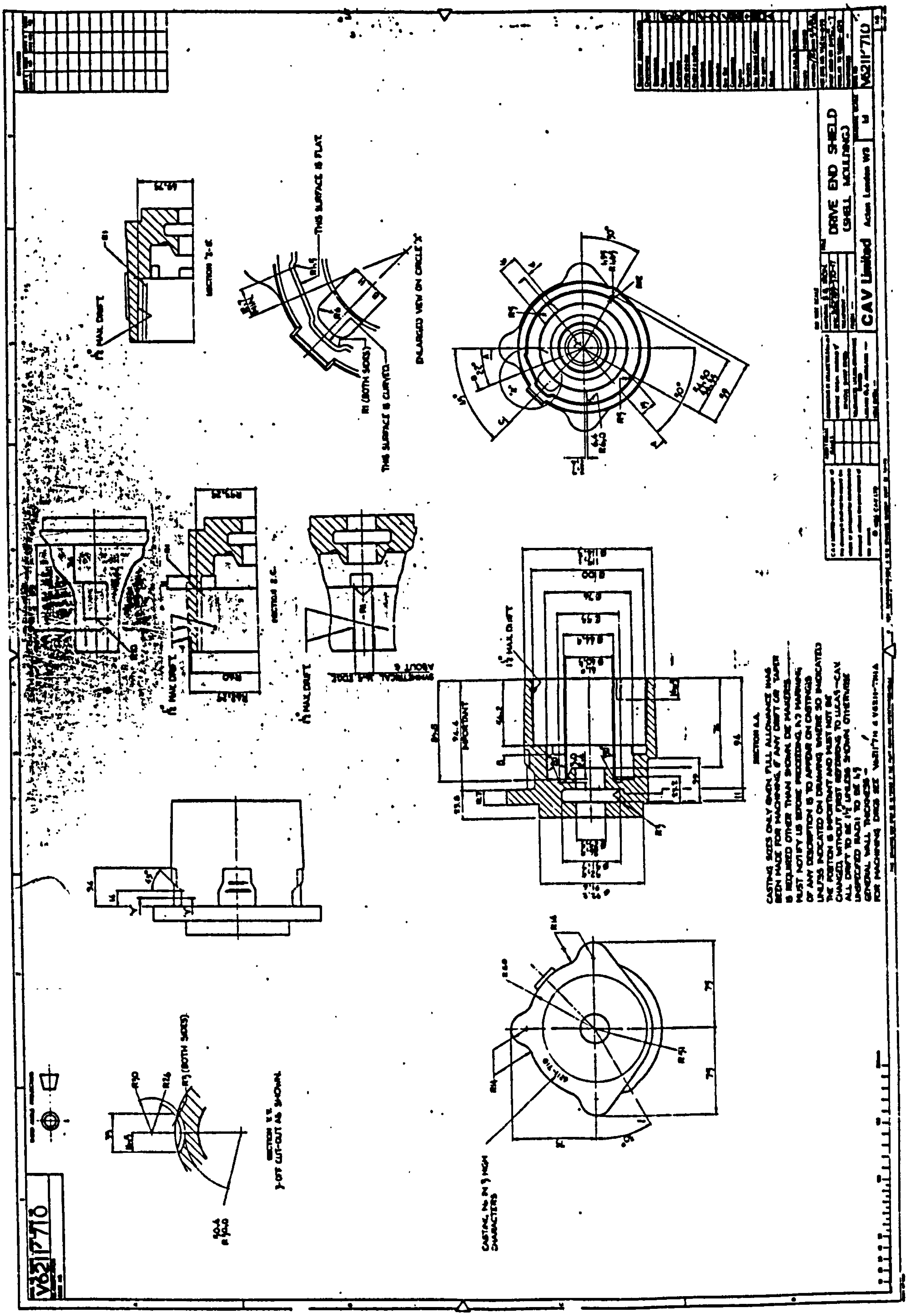


Figure 5.2.2 – Manufacturing Drawing for a Drive-End-Shield Casting Variant (Lucas-CAV)

Figure 5.2.2 shows a typical manufacturing drawing for the Drive-End-Shield moulding. These drawings are of the traditional (manually drafted) form. A comparison of the representative family of manufacturing drawings yields the persistent and non-persistent parameters and features. This is given in figure 5.2.3.

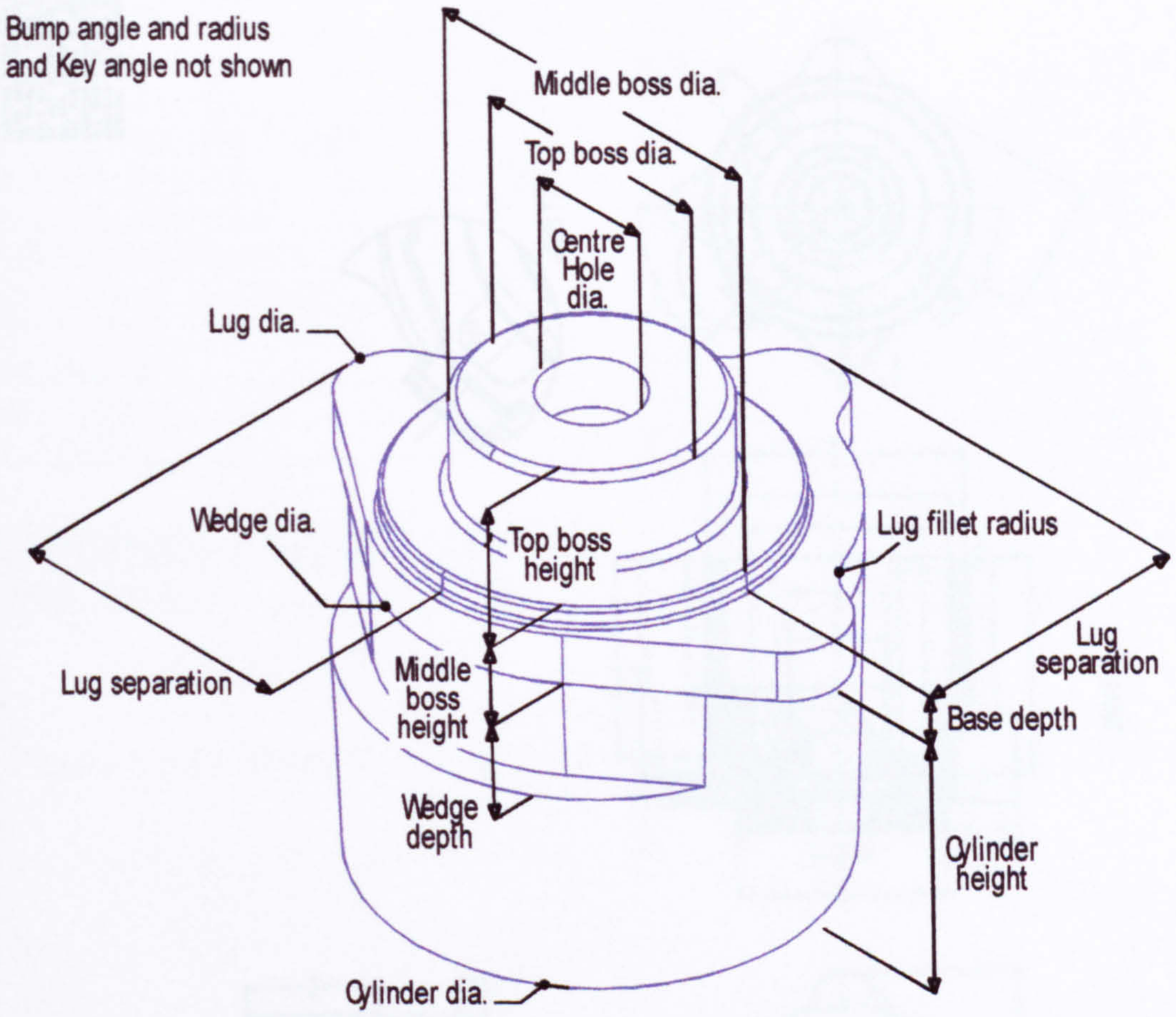


Figure 5.2.3 A Schematic Representation of the Generic Casting.

The casting was modelled using the Autodesk Mechanical Desktop package, which is suitable for the limited variance of this product. Figure 5.2.4 shows the Generic manufacturing drawing, generated automatically by Mechanical Desktop, via the FMT software. The Generic Instance of the Drive-End-Shield, containing all of the combined features of the casting family, is given in figure 5.2.5.

Further instances of the casting are given in Appendix II.

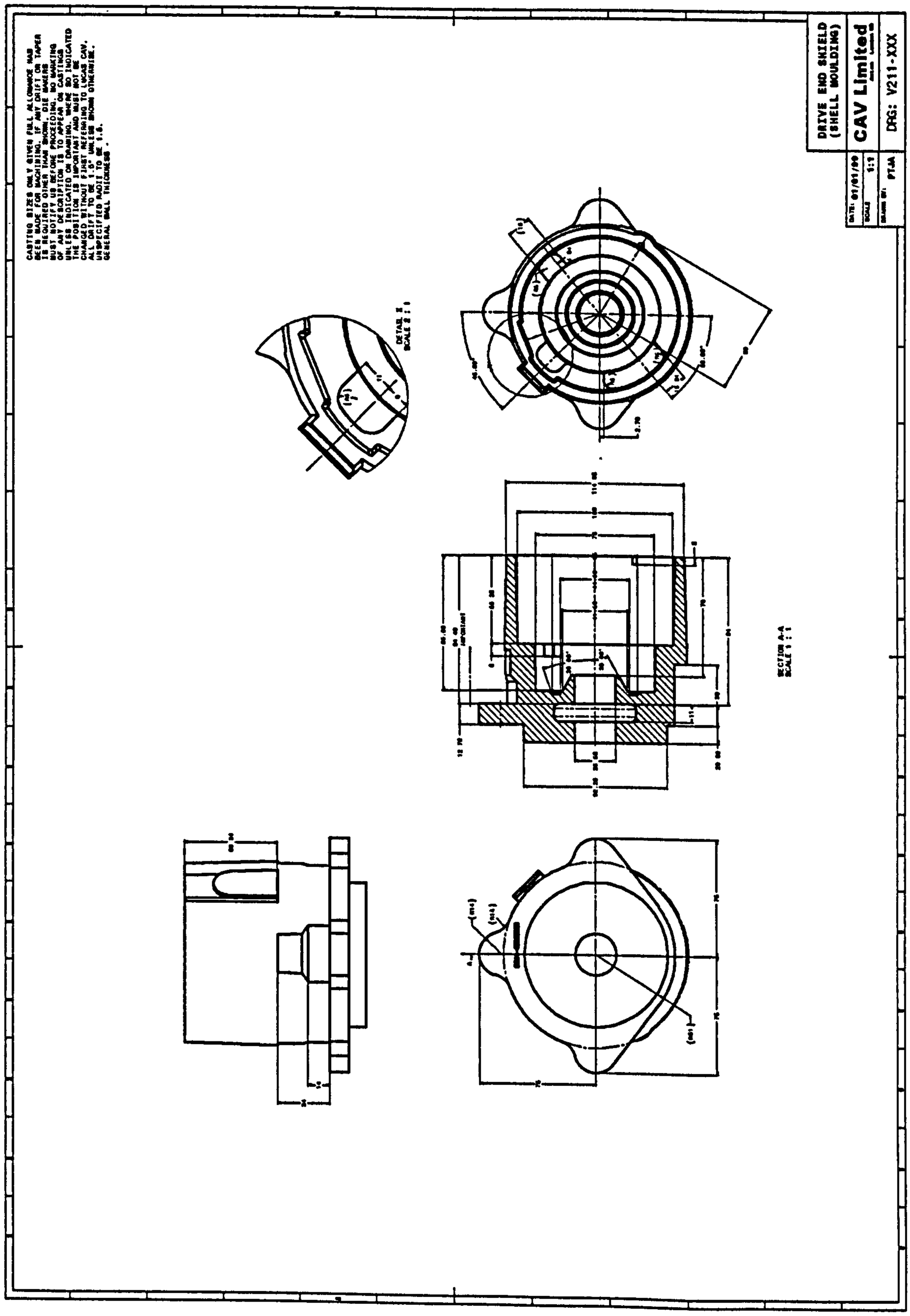


Figure 5.2.4 – Manufacturing Drawing for the Generic Drive-End-Shield

Parameter	Value
Lug Separation	75mm
Lug Diameter	28mm
Lug Angle	90°
Fillet 2	14mm
Wedge Diameter	102mm
Wedge Depth	39mm
Base Depth	12.7mm
Cylinder Height	94.4mm
Cylinder Bottom Diamet	114.3mm
Top Boss Height	23.8mm
Top Boss Diameter	92.2mm
Middle Boss Height	5mm
Middle Boss Diameter	96.05mm
Centre Hole Diameter	26.5mm
Bump Angle	30°
Bump Radius	12mm
Key Angle	45°

Non-Persistent Feature	Status
Middle Boss	Supp

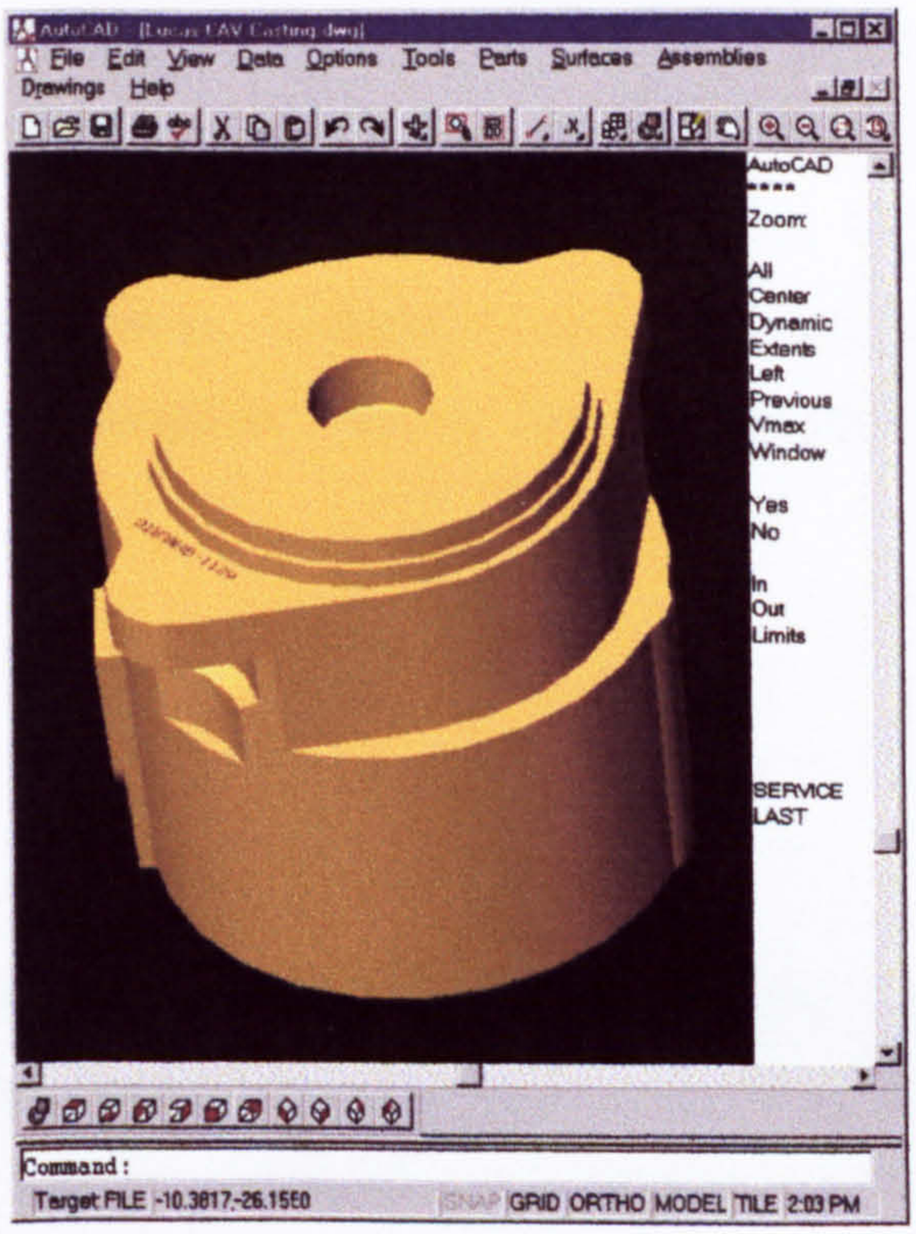


Figure 5.2.5 – Drive-End-Shield No. V6211-673 - CAD Model & Parameter

5.3 Hydroflow Rotary Drum Filter System

The Rotary Drum Filter, from Hydroflow Europe Ltd., is a modular sub-system, typically inserted into large conveyor filtration units, also manufactured by Hydroflow. The sole application for these systems is machining fluid filtration. This case study is itself a sub-set of ongoing research, undertaken by the author, to improve the design of Hydroflow's family of filtration systems. The key objectives of this study are to:

a) Reduce the cost of manufacture,
b) Reduce the size of the systems.

Establishing a high degree of modularisation within their design systems can substantially realise both of these objectives. Such a characteristic is inherent of the Variant Method, where existing designs can be combined into core (or Master) models, which can be varied to suit the particular design requirements.

Figure 5.3.1 shows an example assembly drawing for a typical filtration system. It consists of a number of 'Cleanliness Stages', e.g., Clean, Very Clean and Ultra Clean. Fluid is passed through each of these stages and finally extracted, for reuse, at the Ultra Clean stage. The main feature of this system is the Rotary Drum Filter, which, according to the size of the system can vary in drum length between 750mm and 1000mm. This case study will focus only on the Drum Filter module.

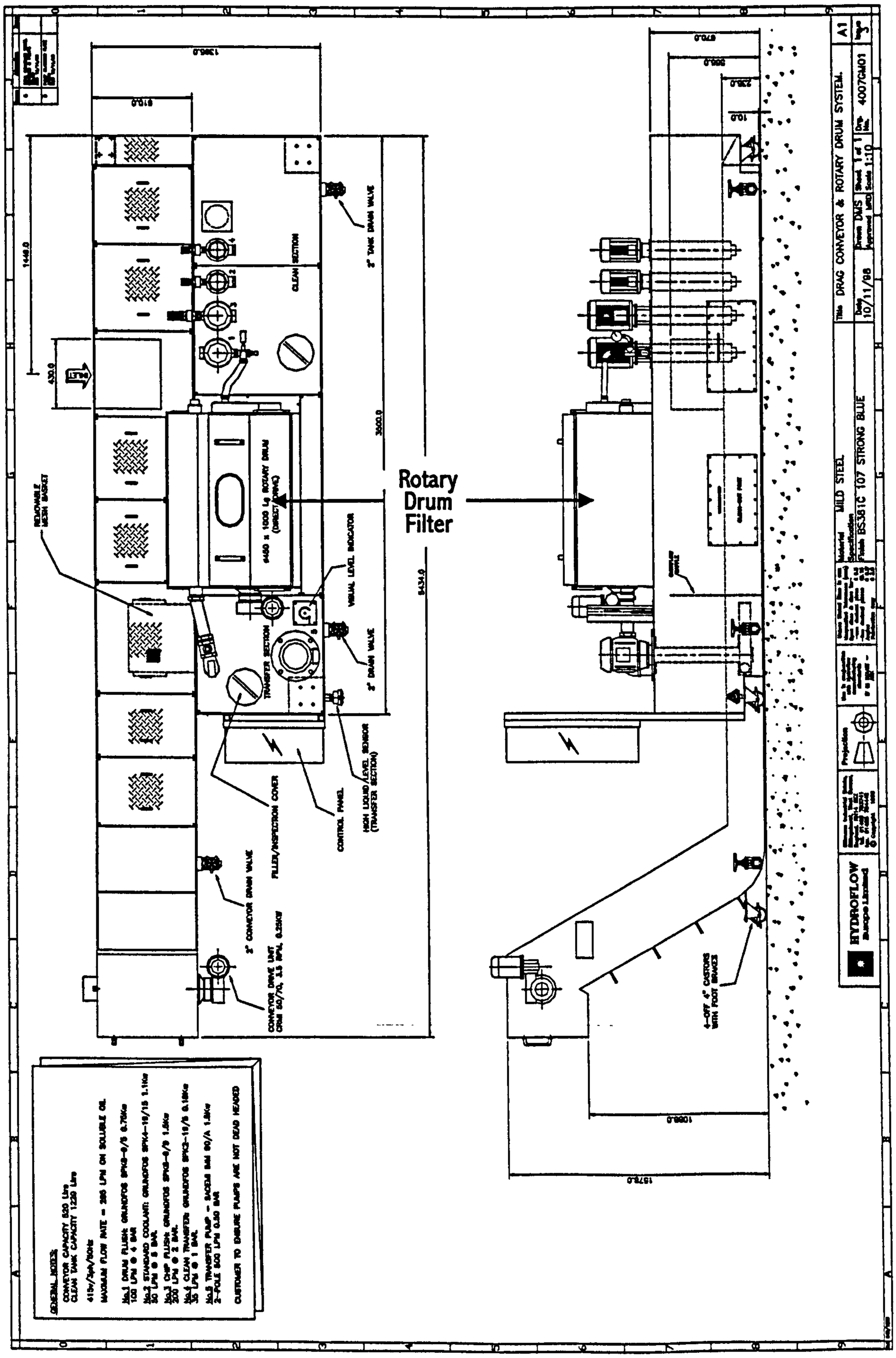


Figure 5.3.1 – Assembly Drawing of a Hydroflow Rotary Drum Filtration System

The Rotary Drum Module is essentially a cylindrical mesh filter and constituted of the following parts:

1) Drum Body – Holds fluid and Drum Unit.
2) Mesh Clamp – Fixes Mesh Roll on the Drum.
3) Drum Flush Pipe – Flushes the Drum internally
4) Drum Flush Pipe End – Provides periodical tilting of the Flush Pipe
5) Drum Endplate – Supporting end of the Drum
6) Viewing Hood – Lift-up hood for inspection
7) Drum Viewing Window – Perspex window for viewing Drum
8) Mesh – fine mesh roll for filtering
9) End Plate – Rolled, lipped ring to fix mesh to Drum Endplate
10) Rolled Ring – simple clamping ring to fix Mesh to Endplate
11) Drum End Guard – Enclosure for the open end of the Drum Body

Although Hydroflow use drafting systems, such as AutoCAD (in which the manufacturing drawings were supplied), solid modelling of this system is generally beyond the scope of Mechanical Desktop. Therefore parts and assemblies were modelled using SolidWorks, and are given in Appendix II.

Two variants of the Rotary Drum Module are given here, the 1000mm and 750mm length units. These are shown in figures 5.3.2 and 5.3.3 respectively. (Note that for both of these models the Drum Body and Mesh are shown as transparent, so that the inner detail can be seen). In reality (through the FMT software), only the major length variable need be adjusted to perform this variance, as the remaining, linked parameters have been defined as functions of this 'driving parameter'. For example, the length of the actual Drum Mesh is always 80mm less than the stated drum length, namely 920mm for the 1000mm unit and 670mm for the 750mm unit. Such relations allow more rapid design permutations to be considered.

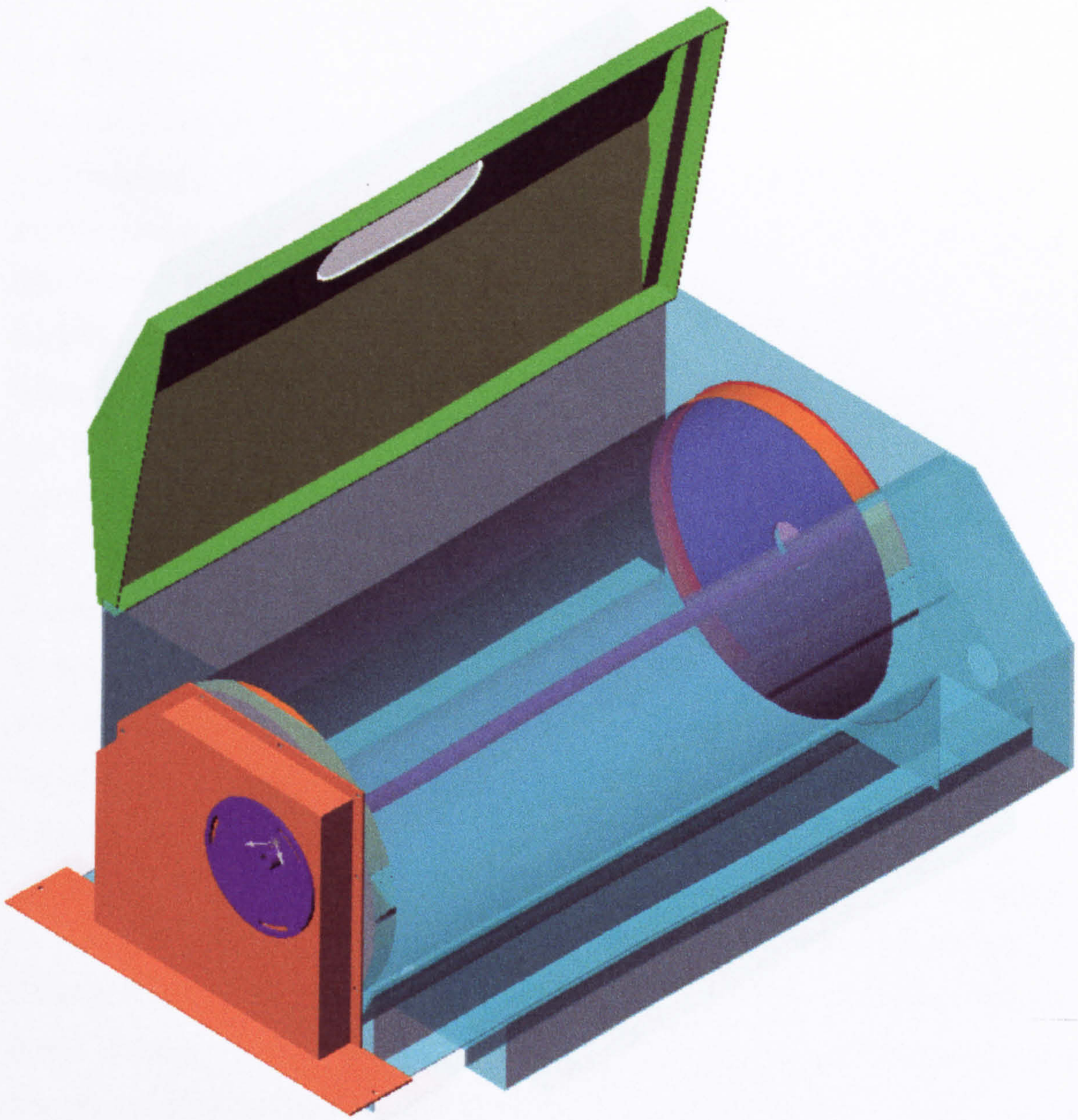


Figure 5.3.2 – Solidworks Model of a Hydroflow Rotary Drum Module – 450x1000mm unit

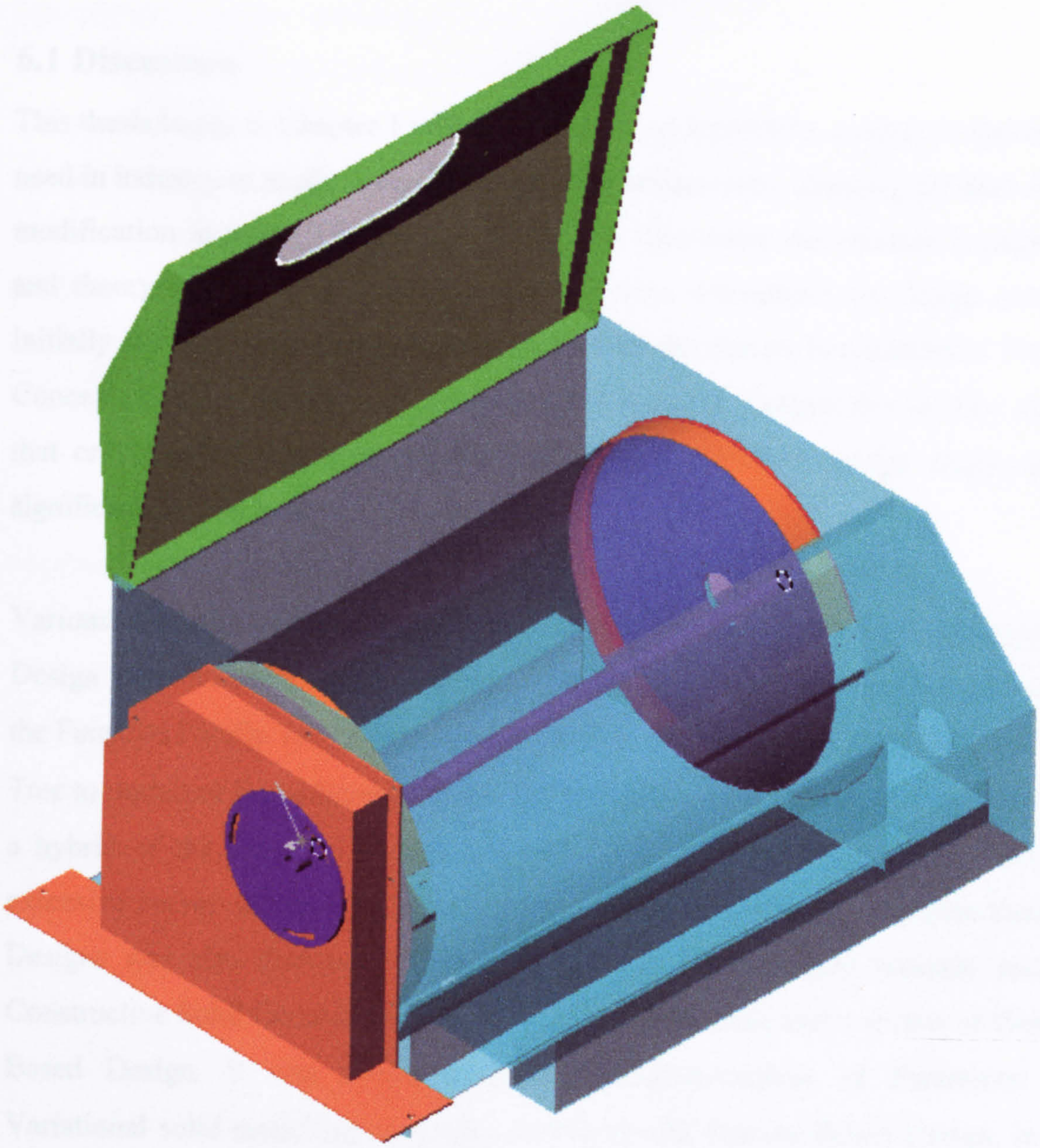


Figure 5.3.3 – Solidworks Model of a Hydroflow Rotary Drum Module – 450x750mm unit

Chapter 6

Discussion & Conclusions

6.1 Discussion

This thesis began in Chapter 1 with the statement of a problem, namely to facilitate a need in industry to easily computerise existing design cases, allowing for their future modification in reuse. Chapter 2 continued by discussing the relevant background and theory in the domains of engineering design throughout the design process. Initially the design process was categorised into the design Requirements, Product Concept, Solution Concept, Embodiment and Detailed Design. It was then argued that only the Solution Concept, Embodiment and Detailed Design stages are of significant interest here.

Various methods and techniques for representing Conceptual and Embodiment Design were reviewed, and it was discussed that, to satisfy the aims of this research, the Function Family Tree is best suited to represent Conceptual Design, and the Parts Tree to represent Embodiment Design. Both of these can be integrally represented by a hybrid of the Function Family and Parts Tree, The Function Means Tree. An extensive survey of representation methods to capture and store adaptable Detailed Designs was also discussed. This included solid representation schemes such as Constructive Solid Geometry and Boundary Representation, and a review of Feature-Based Design. It was concluded that an implementation of Parametric and Variational solid modelling methods, combined with Feature-Based Design, is best suited to represent families of designs in an efficient manner.

Two prominent methods, the Generative Method and the Variant Method were compared, and it was decided that the Variant Method will allow existing design cases to be efficiently modelled, with less effort and overhead that the Generative Method.

Chapter 3 proposed a Generic Variant Methodology to store families of existing designs. The Methodology allows the designer to encompass an entire family of designs, combining conceptual, embodiment and detailed design within a single Variant model. Chapter 4 implements the methodology as a software application, incorporating three established Parametric solid Modelling CAD systems; Pro/ENGINEER, SolidWorks and Mechanical Desktop. The software allows the designer to create a Parts Tree based representation of a family of designs, which is linked to a representative set of CAD models. These models can also be adapted, via driving parameters, in this database. ***This enables the system to store a family of designs, using just a single model, as it only requires the parameters for each instance to be stored.*** The database (termed an intelligent engine) handles regeneration of the model to satisfy the instance specific parameters.

The methodology and software are substantiated with three industrial case-studies; a Machine Tool - Lathe Chuck family, Drive-End-Shield Motor Casting and a Filtration System.

6.2 Conclusions

The implementation of the Variant Methodology has proven the usefulness of this research for a number of companies. It has enabled these companies to transform 2-dimensional manufacturing drawings (which are of limited use) into 3-dimensional *solid* models. Thus enabling them to use the full benefits of Computer Aided Design.

6.3 Recommendations for Future Work

In spite of the fact that the software implementation is a very useful industrial tool, there are a number of issues that will enhance this research further. One particular area is the lack of coherence between Functions and Parts. The study, of even a minor product, such as the Propeller Shaft example, highlights that many functions do not directly map to Parts. It would be more beneficial to provide an intermediate means, such as 'Features of Parts'. This idea is further enhanced by the use of Feature-based modelling.

Another area of interest is expanding the Variant Principle to embodiment and even conceptual design. This would raise the Variant Method to a level more suitable for 'Innovative Design', which has to date been the domain of the Generative Method. In this respect the author will be employed in the industrial sector to implement such a system.

Other, more immediate, concerns include the handling of constraint satisfaction within the software. As it stands, constraint satisfaction between parts and assemblies, is the responsibility of the CAD package being used. This does pose problems where the designer would expect constraints to be solved concurrently (see section 2.9) rather than sequentially.

Bibliography

Akiyama K., (1991),
'Function Analysis – Systematic Improvement of Quality and Performance', Productivity Press.

Aldefeld B., (1983)
'On Automatic Recognotion of 3D Structures from 2D Representations', Computer Aided Design, Vol. 15, No. 2

Aldefeld B. and Richter H.,(1983)
'Semiautomatic Three-Dimensional Interpretation of Line Drawings', Computers and Graphics, Vol. 8, No. 4

Andreasen M.M., (1980)
'Machine Design Methods Based on a Systematic Approach – Contribution to a Design Theory', Dissertation, Department of Machine Design, Lund Institute of Technology, Sweden

Arbab A.L.,(1982)
'Requirements and Architecture of CAM Oriented Systems for Design and Manufacture of Mechanical Parts', Ph.D. dissertation, University of California, Los Angeles, USA

Autodesk
'AutoCAD Computer Aided Drafting Application', Autodesk Inc, www.autodesk.com

Baer A., Eastman C. and Hanrion M., (1979)
'Geometric Modelling: A Survay', Computer Aided Design

Baumgart B., (1974)
'Geometric Modelling for Computer Vision', Ph.D. dissertation, Department of Computer Science, Stanford University

Baumgart B., (1975)
'A Polyhedron Representation for Computer Vision', National Computer Conference, AFIPS Conference Proceedings 44

Besant C.B., and Lui C.W.K., (1986)
'Computer Aided Design and Manufacture', 3rd Edition, *Ellis Horwood*, West Sussex.

Braid I.C., (1973)
'Designing with Volumes', *PhD Thesis*, University of Cambridge, 1973.

Braid I.C. (1979)
'Notes on Geometric a Modeller', Computer Laboratory, University of Cambridge, CAD Group Document 101.

British Standards Institute
'Engineering Drawing', BS308. HMSO Publications

Brown K.N., and Williams J.H.S., (1992)
'Grammars of Features in Design', *Artificial Intelligence in Design '92*, Netherlands.

Capoyleas C., et al. (1995)
'Generic Naming in Generative Constraint-Based Design', *Technical Report* 94-011, Dep. Computer Science, Purdue University, USA.

Chamberlain M.A. et al., (1993)
'Protrusion-Features Handling in Design and Manufacturing Planning', *Computer-Aided Design*, Volume 25, Number 1, January 1993.
Chen X. and Hoffmann C.M., (1995)
'Towards Feature Attachment', *Computer-Aided Design*, Volume 27, Number 9, September 1995.
Chiyokura H and Kimura F., (1985)
'A Method of Representing the Solid Design Process', *IEEE Computer Graphics & Applications*, April 1985.
Cross N., (1991)
'Engineering Design Methods', John Wiley and Sons, London
Cutkosky M., Tenenbaum J.M., Muller D.,(1988)
'Features in Process Based Design', ASME Computers in Engineering Conference, ASME Press
De Fazio T.L. et al., (1993)
'A Protoype of Feature-Based Design for Assembly', *Journal of Mechanical Design*, Volume 115/723, December 1993.
Duan W. et al., (1993)
'FSMT: A Feature Solid-Modelling Tool for Feature-Based Design and Manufacture', *Computer-Aided Design*, Volume 25, Number 1, January 1993.
Duffy A.H.B, Smith J.S., Duffy S.M., (1998)
'Design Reuse Research: A Computational Prespective', Keynote Paper – Engineering Design Conference '98, Brunel University, UK
EMT (1998)
'MakeIT-3D', EMT Software, www.emtsoft.com
Ertas A and Jones J.C., (1993)
'The Engineering Design Process', *John Wiley & Sons*, ISBN: 0-471-51796-8
Feng C., and Kusiak A., (1995)
'Constraint-Based Design of Parts', *Computer Aided Design*, Volume 27, Number 5 (May).
Féru F. et al., (1992)
'Feature-Based Modelling: State of the art and evolution, Manufacturing in the era of Concurrent Engineering', *IFIP*, 1992.
Finger S. and Safier S.A., (1990)
'Representing and Recognising Features in Mechanical Design', *Design Theory and Methodology* - DTM '90, ASME.
Finger S., (1998)
'Design Reuse and Design Research', Keynote Paper, Engineering Design Conference '98, Brunel University, UK
Fitzhorn P.A., (1990)
'Language of Topologically Valid Bounding Manifolds', *Computer-Aided Design*, Volume 22, number 7, September 1990.
Flasinski M., (1995)
'Use of Graph Grammars for the Description of Mechanical Parts', *Computer-Aided Design*, Volume 27, number 6, June 1995.
Grant D.P., (1977)
'The How To Of Decision Making', Design Methods and Theories, Vol. 11, No. 3, California, USA

Gu K, Tang Z., Sun J., (1985)
'Reconstruction of 3D objects from Orthographic Projections', Proceedings of COMPINT'85
Hall M.A. et al., (1990)
'Feature Abstraction in Knowledge-Based Critique of Designs', *ASME Winter Annual Meeting 1990.*
Ho B., (1986)
'Inputting Constructive Solid Geometry Representations Directly from 2D Orthographics Engineering drawings', Computer Aided Design, Vol. 18, No. 3
Hoffmann C.M., (1993)
'On the Semantics of Generative Geometry Representations', *Advances in Design Automation* - Volume 2, DE-Vol, 65-2.
Hoffmann C.M. and Rossignac, (1996)
'A Road Map to Solid Modelling', *IEEE Transactions on Visualization and Computer Graphics*, Volume 2, Number 1, March 1996.
Hsu T. and Sinha D.K., (1992)
'Computer-Aided Design: An Integrated Approach', *West Publishing Company*, ISBN: 0314-80781-0.
Hubka V.,(1982)
'Principles of Engineering Design', Butterworth Scientific, UK
Hubka V. and Eder W.E., (1988)
'Theory of Technical Systems', Springer Verlag, Berlin
Idesawa M., (1973)
'A System to generate a Solid Figure from Three Views', JSME, Vol. 16
IEE Computing and Control Division ,(1989)
'Character Recognotion and Applications', IEE, London
Johnson A., and Thornton A.C., (1993)
'Modelling Functionality in CAD: Implications for Product Representation', *International Conference on Engineering Design* (ICED-93), August 17-19.
Jones J.C., (1980)
'Design Methods', 1980 Edition, Wiley-Interscience Publications, New York
Keirouz W. et al., (1990)
'Integrating Parametric Geometry, Features, and Variational Modeling for Conceptual Design', *Design Theory and Methodology*, DTM '90, ASME.
Kimpton M. and Sivaloganathan S., (1998)
'Development of a New Product using Systematic Design Methods. Case Study: An Elevating Platform for Interior Decorating', Engineering Design Conference '98, Brunel University, UK
Kondo K., (1990)
'PIGMOD: Parametric and Interactive Geometric Modeller for Mechanical Design', *Computer-Aided Design*, Volume 22, No.10, December 1990
Kurland R.H, (1996)
'Understanding Variable-Driven Modeling', Technical Document, Technicom Inc.
Laakko T., Mäntylä M., (1993)
'Feature modelling by incremental feature recognition' *Computer-Aided Design*, Volume 25, No.8, August 1993

Lamure H., and Michelucci D., (1996)
'Solving Geometric Constraints By Homotopy', *IEEE Transactions on Visualization and Computer Graphics*, Vol-2, No 1 March 96 Issue.
Lanasami V. and Langrana N., (1990)
'Engineering Drawing Processing and Vectorisation System', Computer Vision, Graphics Image Processing
Leavers V.F.,(1992)
'Shape Detection in Computer Cision Using the Hough Transform', Springer-Verlag, London
Lee K., (1983)
'Shape Optimization of Assemblies Using Geometric Properties', *PhD Thesis*, Massachusetts Institute of Technology, December 1983.
Lee Y.C., and Fu K.S., (1987)
'Machine Understanding of CSG: Extraction and Unification of Manufacturing Features', *IEEE Computer Graphics &* Applications, January 1987.
Maher M., Balachandran B., Zhang D.M., (1995)
'Case-Based Reasoning In Design', Lawrence Erlbaum Associated Publishers, Hove, UK.
Malmqvist J., (1995)
'A Computer Based Approach Towards Including Design History Information In product Models and Function-Means Trees', Proceedngs of the Design Theory and Methodology Conference, DTM'95, Boston MA. USA
Mäntylä M., (1988)
'An Introduction to Solid Modelling', *Principles of Computer Science Series; 13*, Computer Science Press, ISBN: 0-88174-108-1.
Medland A.J., (1989)
'Approaching CAD Through Constraint Modelling', *CAM-I Design Automation Workshop on Current Trends in Product Modelling and Design Automation*, 7th March 1989.
Medland A.J. and Mullineux G., (1988)
'Principles of CAD: A Coursebook', *Kogan Page Ltd.*, London, ISBN:1-85091-534-2.
Medland A.J. and Mullineux G., (1993)
'A Constraint Approach to Feature-Based Design', *Int. J. Computer Integrated Manufacturing*, Volume 5, Numbers 1 & 2.
Meeran S.A., (1991)
'Automated Feature Recognition from 2D CAD Models', *PhD Thesis*, Cranfield Institute of Technology, April 1991.
Mortenson M.E., (1985)
'Geometric Modelling', John Wiley & Sons, New York
Nielsen E.H., Dixon J.R. and Simmons M.K. (1987)
'How Shall We Represent the Geometry of Designed Objects?' *Technical Report 6-87*, Mechanical Design Automation Laboratory, University of Massachusetts, Amherst, MA
Ogg H.C. and Ogg M.H., (1992)
'Optical Character Recognotion: - A Librarian's Guide', Meckler, London

Pahl G. and Beitz W., (1988)
'Engineering Design – A Systematic Approach', The Design Council, Springer-Verlag, Berlin
Pugh S.,(1991)
'Total Design', Addison Wesley Publlishing Company, UK
Requicha A.A.G., (1977)
'Mathematical Models of Rigid Solids', Technical Memo No.28, Production Automation Project, University of Rochester
Requicha A.A.G., (1980)
'Representations for Rigid Solids: Theory, Methods and Systems', Computing Surveys, Vol.12, No. 4
Requicha A.A.G. and Voelcker H.B., (1982)
'Solid Modeling: A Historical Summary and Contemporary Assessment', *IEEE Computer Graphics and Applications*, October 1983.
Requicha A.A.G. and Voelcker H.B. (1983)
'Solid Modelling: Current Status and Research Directions', IEEE Computer Graphics and Applications
Requicha A.A.G. and Rossignac J.R., (1992)
'Solid Modeling and Beyond', *IEEE Computer Graphics and Applications,* September 1992.
Roller D., (1991)
'An Approach to Computer-Aided Parametric Design', *Computer-Aided Design*, Volume 23, Number 5, June 1991.
Sakurai H., (1983)
'Solid Model Input Through Orthographic Views', Computer Aided Design, Vol. 17, No. 3
Shahin S. and Sivaloganathan S., (1998)
'Representing Conceptual Designs', Engineering Design Conference '98, Brunel University, UK
Shahin S., Andrews P.T.J. and Sivaloganathan S., (1998)
'A Design Reuse System', Engineering Design Conference '98, Brunel University, UK
Shah J.J., (1991)
'Assessment of features technology', *Computer-Aided Design*, Vol. 23, June 1991
Shah J.J. and Wu J., (1993)
'Hybrid Method for Muti-Target Parametric Design', *Advances in Design Automation* - Volume 1, DE-Col 65-1, ASME.
Shah J.J. and Mäntylä M., (1995)
'Parametric and Feature-based CAD/CAM', Wiley Interscience, New York
Shah J.J. et al, (1996)
'Research Opportunities in Engineering Design', NSF Strategic Planning Workshop – Final Report, National Science Foundation, USA
Shahin T.M.M., (1996)
'Automation of Feature-Based Modelling and Finite Element Analysis for Optimal Design', Ph.D. Thesis, Department of Manufacturing and Engineering Systems, Brunel university, UK
Sheu L.C. and Lin J.T., (1993)
'Representation Scheme for Defining and Operating Form Features', *Computer-Aided Design*, Volume 25, Number 6, June 1993.

Shigley J.E., (1977)
'Mechanical Engineering Design', 3rd Edition, McGraw-Hill Book Company, New York
Sittas E., (1989)
'Appropriate Forms of Solid Modelling for Conceptual Design', *PhD Thesis*, Brunel University, Department of Manufacturing and Engineering Systems.
Sivaloganathan S. (1991)
'Sketching Input For Computer-Aided Engineering', PhD Thesis, City University, England, September 1991
Sivaloganathan S., Evbuoman N.F.O., Jebb A. and Wynn H.P., (1995)
'Design Function Deployment – A Design System for the Future', Design Studies Journal, Vol. 16, No. 4
Sivaloganathan S. (1996)
'Computer-Aided Design' *MN217 Lecture Notes*, Brunel University, Department of Manufacturing and Engineering Systems, England.
Sodhi R and Turner J.U., (1994)
'Towards Modelling of Assemblies for Product Design', *Computer-Aided Design*, Volume 26, Number 2, February 1994.
Softelec (1997),
'VP-Max Automated Raster to Vector Software', Softelec, Gmbh., www.softelec.com
Solano L., and Brunet P., (1994)
'Constructive Constraint-Based Model for Parametric CAD Systems', *Computer-Aided Design*, Volume 26, Number 8 (August)
Suh H. and Ahluwalia R.S., (1995)
'Feature Modification in Incremental Feature Generation', *Computer-Aided Design*, Volume 27, Number 8, August 1995.
Sutherland I.E., (1963)
'SKETCHPAD: A man machine graphical communication system', Vol. 12 Spring Joint Computer Conference
Thornton A.C., (1993)
'Constraint Specification and Satisfaction in Embodiment Design', *PhD Thesis*, University of Cambridge, July 1993.
Thornton A.C., (1996)
'The Use of Design Knowledge to Improve the Search for Feasible Designs', *MIT Technical Paper*, May 1996.
Thornton A.C. and Johnson A., (1993)
'Constraint Specification and Satisfaction in Embodiment Design', *International Conference on Engineering Design*, ICED 93.
Turner G. and Anderson D.C,(1988)
'An Object Oriented Approach to Interactive feature-Based Design, for Quick Turnaround Manufacturing', ASME Computers in Engineering Conference, ASME Press
Vaiyapuri V. and Okogbaa O.G., (1994)
'An Integrated Design Approach for Feature Based Manufacturing Using Concurrent Engineering', *2nd Industrial Engineering Research Conference Proceedings*, Institute of Industrial Engineers.
Voelcker H.B. and Requicha A.A.G., (1977)
'Geometric Modelling of Physical Parts and Processes',IEEE Computers and Graphics Applications.

Voelcker H.B. (1988)
'Modelling in the Design Process' *National Academy Press*, Washington DC.
Wang W., (1992)
'On the Automatic Reconstruction of a 3D Objects Constructive Solid Geometry Representation from its 2D Projection Line Drawing', Ph.D. Thesis, Department of Computer Science, University of Massachusetts, Lowell, USA
Wesley M.A. and Markowsky G., (1981)
'Fleshing Out Projections, IBM Journal of Research and Development, Vol. 25, No. 6
Wesley M.A. and Markowsky G., (1980)
'Fleshing Out Wireframes', IBM Journal of Research and Development, Vol. 24, No. 6
Wesley M.A. and Markowsky G., (1986)
'Generation of Solid Models from two-dimensional and three-dimensional Data', solid Modelling by Computer: from theory to application
Woo T.C, (1985)
'A Combinatorial Analysis of a Boundary Data Schemata', IEEE Computer Graphics and Applications
Zuffante R.P., (1984)
'A Feature Based Representation for Solid Modelling', Masters Thesis, Massachusetts Institute of Technology

Publications by the Author

Peter T.J. Andrews & S. Sivaloganathan, (1996),
'Parametric Primitive Instancing and its Implications in Engineering Design', 12th International Conference on CAD/CAM Robotics and Factories of the Future, 1996.
Peter T.J. Andrews & S. Sivaloganathan, (1996),
'Implementing Parametric Primitive Instancing In Design Reuse', M&ES Research Conference, Brunel University
Awarded 'Best Poster Prize'
Peter T.J. Andrews & S. Sivaloganathan, (1998)
'A Variant Model for Storing Families of Mechanical designs' ,Engineering Design Conference 1998, Brunel University, UK
Peter T.J. Andrews, T.M.M Shahin & S. Sivaloganathan, (1998),
'Design Reuse of Detailed Designs – Four Case Studies', Computers and Industrial Engineering
T.M.M Shahin, Peter T.J. Andrews & S. Sivaloganathan, (1998),
'A Design Reuse System', Engineering Design Conference 1998, Brunel University, UK,
ImechE PartB

Appendix I

Software Code Listing

Overview

This chapter outlines the data structures, algorithms and principles used for the software implementation of the Variant Methodology. The software was written using Microsoft Visual Basic version 5.0. This compiler (or programming environment) undertakes the task of 'automatically' producing a large quantity of code or areas such as the user-interface etc. Hence, these sections of code have been omitted here, leaving only the relevant subroutines relating to the methodology itself. The program makes use of two 'User Defined Controls' (termed Active-X Controls). These controls are visual representations of the Part Node of a Parts Tree, and the Function Node of a Function Family Tree. In essence both of these are simply a shape (a rectangle for the Part Node and a rhombus for the Function Node) into which text can be entered. The definition of how these controls were created is of no relevance here.

Furthermore, this implementation of the Variant Methodology is purely for research purposes, and can be made available on request, either as source code or as an executable. Contact the author at '*peter.andrews@brunel.ac.uk*' for further details.

AI.1 – Definition of Data Sructures and Global Variables

Structures for both Functions and Means are defined here, as well as the structures outlined in Chapter 4.

Module1.bas

Defines the core Data Structures, the Means and Function Nodes, and the Instance structure.

```
Attribute VB_Name = "Module1"
Public Const myCol = &HFFDDBB
Public fMainForm As frmMain
' Create user-defined type for MEANS
' This is the MEANS NODE
Type Means
    NodeID As Integer
    Name As String
    CADfiletype As String
    Xpos As Long
    Ypos As Long
    NumParents As Integer
    Parents() As Integer
    NumChildren As Integer
    Children() As Integer
    NumFunctions As Integer
    Functions() As Integer
    NumOfParams As Integer
    ParamName() As String
    ParamValue() As Double
    ParamUnit() As String
    NumOfSupps As Integer
    SuppEntity() As String
    SuppType() As String
    SuppStatus() As String
    MyPathAndFile As String
    MyFileName As String
    MyCADfileType As Integer
    MyDrawing As String
    PartSuppression As String
    Level As Integer
End Type
```

```
' Create user-defined type for FUNCTION
' This is the FUNCTION NODE
Type Funct
    NodeID As Integer
    Name As String
    Xpos As Long
    Ypos As Long
    NumParents As Integer
    Parents() As Integer
    NumChildren As Integer
    Children() As Integer
    NumMeans As Integer
    Means() As Integer
    Level As Integer
End Type

' Create user-defined type for an INSTANCE
Type DocInstance
    DocType As String
    Name As String
    DrgNo As String
    Date As String
    By As String
    Description As String
    'Path As String
    'FileName As String
    PartsTree() As Means
    FunctTree() As Funct
    RelM() As Integer
    RelF() As Integer
    NumParts As Integer
    NumFuncts As Integer
    NumRels As Integer
    L1x1() As Integer
    L1y1() As Integer
    L1x2() As Integer
    L1y2() As Integer
    L2x1() As Integer
    L2y1() As Integer
```

```
    L2x2() As Integer
    L2y2() As Integer
End Type

Sub Main()
    frmSplash.Show
    frmSplash.Refresh
    Set fMainForm = New frmMain
    Load fMainForm
    Unload frmSplash
    fMainForm.Show
End Sub
```

AI.2 – Form Document

The main section of code, containing various subroutines to create and contol the Parts and Function Trees.

FrmDocument.frm

```
VERSION 5.00
' Load AxtiveX Control
Object = "{8A05F940-8A46-11D2-927F-004A8C000000}#46.0#0"; "NodeMeans.ocx"
Object = "{C077CE62-8A41-11D2-927F-004A8C000000}#49.0#0"; "NodeFunction.ocx"

'Global Variables Declarations
Attribute VB_Name = "frmDocument"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Dim MoveSplit As Boolean
Dim Split As Double
Dim LeftDisplay As String
Dim RightDisplay As String
Dim SelPart As Integer
Dim SelFunct As Integer
Dim VGap, HGap, MGap As Integer
Dim DoWhat As String
Dim ParamsPart As Integer
Dim v1, v2, h1, h2 As Integer
Dim RelMeans, RelFunct As Integer
Dim NumFunc2 As Integer
Dim NumLine5 As Integer
Dim NumMeans2 As Integer
Dim NumLine6 As Integer
Dim DocType As String
Dim NumInstances As Integer 'Includes Generic Instance
' THE FAMILY OF PRODUCTS (INSTANCES)
Dim Instances() As DocInstance
Dim ThisInst As Integer
Dim WhichPart As Integer
Dim DispInstances As Boolean
```

```
'INITIALISATION DEFAULTS

Private Sub Form_Load()
Dim c As Integer
    DocType = "GENERIC"
    DispInstances = True
    ThisInst = 0
    NumInstances = 1
    ReDim Instances(NumInstances)
    'Instances(ThisInst).NumSuppParts = 0
    Instances(ThisInst).DocType = "GENERIC"
    Instances(ThisInst).By = "Unknown"
    Instances(ThisInst).Date = "Unknown"
    Instances(ThisInst).Description = "None"
    Instances(ThisInst).DrgNo = Str(ThisInst)
    Instances(ThisInst).Name = "Untitled"
    'Instances(ThisInst).FileName = "Untitled.fmt"
    'Instances(ThisInst).Path = "c:\PT\FMT-Documents\"
    Me.Caption = Instances(ThisInst).Name
    DoWhat = "NOTHING"
    WhichPart = -1
    ParamsPart = -1
    MGap = 122
    VGap = 488
    HGap = 244
    Instances(ThisInst).NumParts = 1
    Instances(ThisInst).NumFuncts = 1
    SelPart = -1
    SelFunct = -1
    MoveSplit = False
    Split = 0.5
    LeftDisplay = "Parts Tree"
    Command10.Caption = LeftDisplay
    RightDisplay = "Function Family Tree"
    Command11.Caption = RightDisplay
    Form_Resize
'Init first nodes
    ReDim Preserve Instances(ThisInst).PartsTree(Instances(ThisInst).NumParts - 1)
    Instances(ThisInst).PartsTree(0).PartSuppression = "False"
    Instances(ThisInst).PartsTree(0).myDrawing = "None"
```

```
Instances(ThisInst).PartsTree(0).NodeID = 0
Instances(ThisInst).PartsTree(0).Name = "Main Assembly"
Instances(ThisInst).PartsTree(0).Xpos = Picture1.Width / 2 - MNode1(0).Width / 2 + 32768 / 2
Instances(ThisInst).PartsTree(0).Ypos = 500
Instances(ThisInst).PartsTree(0).NumParents = 0
Instances(ThisInst).PartsTree(0).NumChildren = 0
Instances(ThisInst).PartsTree(0).NumFunctions = 0
Instances(ThisInst).PartsTree(0).Level = 0
UpdateMeansNode (0)
ReDim Preserve Instances(ThisInst).FunctTree(Instances(ThisInst).NumFuncts - 1)
Instances(ThisInst).FunctTree(0).NodeID = 0
Instances(ThisInst).FunctTree(0).Name = "Primary Function"
Instances(ThisInst).FunctTree(0).Xpos = Picture2.Width / 2 - FNode1(0).Width / 2 + 32768 / 2
Instances(ThisInst).FunctTree(0).Ypos = 500
Instances(ThisInst).FunctTree(0).NumParents = 0
Instances(ThisInst).FunctTree(0).NumChildren = 0
Instances(ThisInst).FunctTree(0).NumMeans = 0
Instances(ThisInst).FunctTree(0).Level = 0
'FNode1(0).Colour=
UpdateFunctNode (0)
' SCROLL BARS
VScroll1.Value = 0
   v1 = 0
   VScroll1.Min = 0
   VScroll1.Max = 32767
   VScroll1.LargeChange = 1024
   VScroll1.SmallChange = 128
VScroll2.Value = 0
   v2 = 0
   VScroll2.Min = 0
   VScroll2.Max = 32767
   VScroll2.LargeChange = 1024
   VScroll2.SmallChange = 128
HScroll1.Value = 32768 / 2
   h1 = 32768 / 2
   HScroll1.Min = 0
   HScroll1.Max = 32767
   HScroll1.LargeChange = 1024
   HScroll1.SmallChange = 128
```

```
HScroll2.Value = 32768 / 2
h2 = 32768 / 2
HScroll2.Min = 0
HScroll2.Max = 32767
HScroll2.LargeChange = 1024
HScroll2.SmallChange = 128
'Setup Parameters & Suppress window
ParamsGrid.ColWidth(0) = 3250
ParamsGrid.ColWidth(1) = ParamsGrid.Width - 450 - 3250 - 105
ParamsGrid.ColWidth(2) = 450
SuppGrid.ColWidth(0) = 450
SuppGrid.ColWidth(1) = 3100
SuppGrid.ColWidth(2) = SuppGrid.Width - 450 - 3100 - 105
ParamsGrid.Rows = 1
ParamsGrid.Row = 0
ParamsGrid.Col = 0
ParamsGrid.Text = "Parameter"
ParamsGrid.Col = 1
ParamsGrid.Text = "Value"
ParamsGrid.Col = 2
ParamsGrid.Text = "Unit"
ParamsGrid.Rows = 1
SuppGrid.Row = 0
'Relations
RelMeans = -1
RelFunct = -1
RelsGrid.Row = 0
RelsGrid.Col = 0
RelsGrid.Text = "Means / Part"
RelsGrid.Col = 1
RelsGrid.Text = "Function"
Instances(ThisInst).NumRels = 0
' Instancing
For c = 0 To NumInstances - 1
    UpdateInstance (c)
Next c
UpdateInstance (ThisInst)
End Sub
```

```
'Toggle between Parts Tree and Parts Oriented FMT
Private Sub Command10_Click()
   If (LeftDisplay = "Parts Tree") Then
      LeftDisplay = "Part Oriented Function/Means Tree"
   ElseIf (LeftDisplay = "Part Oriented Function/Means Tree") Then
      LeftDisplay = "Parts Tree"
   End If
   Command10.Caption = LeftDisplay
   Command9.SetFocus
   RedrawLeftDisplay
End Sub

'Toggle between Function Tree and Function Oriented FMT
Private Sub Command11_Click()
   If (RightDisplay = "Function Family Tree") Then
      RightDisplay = "Function Oriented Function/Means Tree"
   ElseIf (RightDisplay = "Function Oriented Function/Means Tree") Then
      RightDisplay = "Function Family Tree"
   End If
   Command11.Caption = RightDisplay
   Command9.SetFocus
   RedrawRightDisplay
End Sub

'Reset Parts Tree – Function Tree Split screen sizes
Private Sub Command1_Click()
   Split = 0.5
   Form_Resize
End Sub

'Print a CAD DRAWING of the current selected means
Private Sub Command13_Click()
Const swDocDRAWING = 3
Dim RetVal
Dim swApp As Object 'Def variable to hold app object
Dim myDrawing As Object 'Define variable to hold part object
If (SelPart >= 0) Then
      ' Set CancelError is True
      CommonDialog1.CancelError = True
      On Error GoTo ErrHandler
```

```
' Set flags
CommonDialog1.Flags = cdlOFNHideReadOnly
' Set filters
CommonDialog1.Filter = "All Files (*.*) | *.*" & _
" | Pro/ENGINEER Drawing (*.asm) | *.asm" & _
" | Mechanical Desktop (*.dwg) | *.dwg" & _
" | SolidWorks Drawing (*.SLDDRW) | *.SLDDRW"
' Specify default filter
CommonDialog1.FilterIndex = 4
    'Default filename
    If (Instances (ThisInst).PartsTree (SelPart).myDrawing <> "None") Then
       CommonDialog1.filename = Instances (ThisInst).PartsTree (SelPart).myDrawing
    End If
    ' Display the Open dialog box
    CommonDialog1.ShowOpen
    ' Display name of selected file
    Instances (ThisInst).PartsTree (SelPart).myDrawing =  CommonDialog1.filename
'This will attach to current SolidWorks session or start up new session in background.
    Set swApp = CreateObject ("SldWorks.Application")
    swApp.Visible (True)
' Load file from current directory.
Set myDrawing = swApp.OpenDoc (Instances (ThisInst).PartsTree (SelPart).myDrawing, swDocDRAWING)
    If myDrawing Is Nothing Then
       Exit Sub
    Else
            'Set myDrawing = swApp.ActivateDoc (CommonDialog1.FileTitle)
      myDrawing.EditRebuild
       'Print IT!!!
       myDrawing.PrintDirect
       swApp.UserControl (True)
     Beep
        End If
'Exit Sub
   Else
      MsgBox ("Select a Part Node First")
   End If
ErrHandler:
   'User pressed the Cancel button
   Exit Sub
End Sub
```

```
' UPDATE Parameters in ALL CAD files
Private Sub Command15_Click()
Dim oldPart As Integer
   Dim RetVal
   Const swDocPART = 1            ' These definitions are consistent with type names
   Const swDocASSEMBLY = 2        ' defined in \SldWorks\samples\appComm\swconst.h
   Const swDocDRAWING = 3
   Dim swApp As Object ' Define variable used to hold the app object
   Dim Part As Object  ' Define variable used to hold the part object
   Dim c, d, e As Integer
   Dim WhatType As Integer
   Dim myUnit As String
   Dim myVal As Double
   Dim nParts As Integer
   Dim maxLevel, thisLevel As Integer
   Dim tParts() As Integer

   nParts = Instances(ThisInst).NumParts
   oldPart = SelPart
   maxLevel = 0
  For e = 0 To nParts - 1
     thisLevel = Instances(ThisInst).PartsTree(e).Level
     If (thisLevel > maxLevel) Then
        maxLevel = thisLevel
     End If
  Next e
  ReDim tParts(nParts)
  For e = 0 To maxLevel
     For d = 0 To nParts - 1
      If ((Instances(ThisInst).PartsTree(d).Level = e) Or (Instances(ThisInst).PartsTree(d).NumChildren <= 0))
Then
           SelPart = d
  If (SelPart >= 0) Then
     'MakePartSelected (SelPart)
     'ActivateParams (SelPart)
    'Code example will be given for SOLIDWORKS only
     If ((Instances(ThisInst).PartsTree(SelPart).MyCADfileType = 5) Or
                                        (Instances(ThisInst).PartsTree(SelPart).MyCADfileType = 6))
Then
        If (Instances(ThisInst).PartsTree(SelPart).MyCADfileType = 5) Then
```

```
    WhatType = swDocPART
ElseIf (Instances(ThisInst).PartsTree(SelPart).MyCADfileType = 6) Then
    WhatType = swDocASSEMBLY
End If
' This will attach to current SolidWorks session or start up new session in background.
Set swApp = CreateObject("SldWorks.Application")
swApp.Visible (True)  ' Uncomment this if you wish to make the new SolidWorks session visible
' Load file from current directory.  This is currently hardcoded to c:\temp
Set Part = swApp.OpenDoc(Instances(ThisInst).PartsTree(SelPart).MyPathAndFile, WhatType)
                If Part Is Nothing Then
                    Exit Sub
                Else
                    Set Part = swApp.ActivateDoc(Instances(ThisInst).PartsTree(SelPart).MyFileName)
                End If
For c = 0 To Instances(ThisInst).PartsTree(SelPart).NumOfParams - 1
 'Sort Out UNITS
  myUnit = Instances(ThisInst).PartsTree(SelPart).ParamUnit(c)
                    Select Case myUnit
                        Case "MM"
                            myVal = Instances(ThisInst).PartsTree(SelPart).ParamValue(c) / 1000
                        Case "M"
                            myVal = Instances(ThisInst).PartsTree(SelPart).ParamValue(c)
                        Case "IN"
                            myVal = (Instances(ThisInst).PartsTree(SelPart).ParamValue(c) / 1000) * 25.4
                        Case "DEG"
                            myVal = Instances(ThisInst).PartsTree(SelPart).ParamValue(c) * (3.141592654 / 180)
                        Case "RAD"
                            myVal = Instances(ThisInst).PartsTree(SelPart).ParamValue(c)
                        Case "VAL"
                            myVal = Instances(ThisInst).PartsTree(SelPart).ParamValue(c)
                    End Select
                    Part.Parameter(Instances(ThisInst).PartsTree(SelPart).ParamName(c)).SystemValue = myVal
                Next c
                For c = 0 To Instances(ThisInst).PartsTree(SelPart).NumOfSupps - 1
                    'FeatureSuppression Instances(ThisInst).PartsTree(SelPart).SuppEntity(c),
                    Instances(ThisInst).PartsTree(SelPart).SuppStatus(c)
                Next c
                'Part.EditRebuild
                swApp.UserControl (False)
End If
```

```
            End If
        End If
    Next d
Next e
SelPart = oldPart
'MakePartSelected (SelPart)
'ActivateParams (SelPart)
nParts = Instances(ThisInst).NumParts
oldPart = SelPart
maxLevel = 0
For e = 0 To nParts - 1
    thisLevel = Instances(ThisInst).PartsTree(e).Level
    If (thisLevel > maxLevel) Then
        maxLevel = thisLevel
    End If
Next e
ReDim tParts(nParts)
For e = 0 To maxLevel
    For d = nParts - 1 To 0 Step -1
     If ((Instances(ThisInst).PartsTree(d).Level = e) Or (Instances(ThisInst).PartsTree(d).NumChildren <= 0))
Then
            SelPart = d
            If (SelPart >= 0) Then
               'MakePartSelected (SelPart)
               'ActivateParams (SelPart)
                    'ONLY CODE FOR SOLIDWORKS WILL BE GIVEN HERE
                    If ((Instances(ThisInst).PartsTree(SelPart).MyCADfileType = 5) Or
                    (Instances(ThisInst).PartsTree(SelPart).MyCADfileType = 6)) Then
                  If (Instances(ThisInst).PartsTree(SelPart).MyCADfileType = 5) Then
                     WhatType = swDocPART
                  ElseIf (Instances(ThisInst).PartsTree(SelPart).MyCADfileType = 6) Then
                     WhatType = swDocASSEMBLY
                  End If
                  ' This will attach to current SolidWorks session or start up new session in background.
                  Set swApp = CreateObject("SldWorks.Application")
                  swApp.Visible (True)        ' Uncomment this if you wish to make the new SolidWorks session
visible
                  ' Load file from current directory.  This is currently hardcoded to c:\temp
                  Set Part = swApp.OpenDoc(Instances(ThisInst).PartsTree(SelPart).MyPathAndFile, WhatType)
                  If Part Is Nothing Then
```

```
    Exit Sub
Else
    Set Part = swApp.ActivateDoc(Instances(ThisInst).PartsTree(SelPart).MyFileName)
End If
For c = 0 To Instances(ThisInst).PartsTree(SelPart).NumOfParams - 1
    'Sort Out UNITS
    myUnit = Instances(ThisInst).PartsTree(SelPart).ParamUnit(c)
    Select Case myUnit
        Case "MM"
    myVal = Instances(ThisInst).PartsTree(SelPart).ParamValue(c) / 1000
Case "M"
    myVal = Instances(ThisInst).PartsTree(SelPart).ParamValue(c)
Case "IN"
    myVal = (Instances(ThisInst).PartsTree(SelPart).ParamValue(c) / 1000) * 25.4
Case "DEG"
    myVal = Instances(ThisInst).PartsTree(SelPart).ParamValue(c) * (3.141592654 / 180)
Case "RAD"
    myVal = Instances(ThisInst).PartsTree(SelPart).ParamValue(c)
Case "VAL"
    myVal = Instances(ThisInst).PartsTree(SelPart).ParamValue(c)
        End Select
        Part.Parameter(Instances(ThisInst).PartsTree(SelPart).ParamName(c)).SystemValue = myVal
    Next c
    For c = 0 To Instances(ThisInst).PartsTree(SelPart).NumOfSupps - 1
        FeatureSuppression Instances(ThisInst).PartsTree(SelPart).SuppEntity(c),
        Instances(ThisInst).PartsTree(SelPart).SuppStatus(c)
    Next c
    Part.EditRebuild
    swApp.UserControl (False)
  End If
 End If
End If
    Next d
  Next e
  SelPart = oldPart
End Sub

' ADD NEW PARAMETER
Private Sub Command16_Click()
If (ParamsPart >= 0) Then
```

```
        MakePartSelected (ParamsPart)
        ActivateParams (ParamsPart)
        Instances(ThisInst).PartsTree(SelPart).NumOfParams =
Instances(ThisInst).PartsTree(SelPart).NumOfParams+1
            ReDim Preserve
            Instances(ThisInst).PartsTree(SelPart).ParamName(Instances(ThisInst).PartsTree(SelPart).NumOfParam
            s)
            ReDim Preserve
            Instances(ThisInst).PartsTree(SelPart).ParamValue(Instances(ThisInst).PartsTree(SelPart).NumOfParam
            s)
            ReDim Preserve
            Instances(ThisInst).PartsTree(SelPart).ParamUnit(Instances(ThisInst).PartsTree(SelPart).NumOfParams
            )
        ParamsGrid.Rows = ParamsGrid.Rows + 1
        ParamsGrid.Row = ParamsGrid.Rows - 1
        ParamsGrid.Col = 0
        Instances(ThisInst).PartsTree(SelPart).ParamName(Instances(ThisInst).PartsTree(SelPart).NumOfParams -
1) = "Param " + Str(ParamsGrid.Row)
        ParamsGrid.Text = "Param " + Str(ParamsGrid.Row)
        ParamsGrid.Col = 1
        Instances(ThisInst).PartsTree(SelPart).ParamValue(Instances(ThisInst).PartsTree(SelPart).NumOfParams -
1) = ParamsGrid.Row
        ParamsGrid.Text = Str(ParamsGrid.Row)
        ParamsGrid.Col = 2
        Instances(ThisInst).PartsTree(SelPart).ParamUnit(Instances(ThisInst).PartsTree(SelPart).NumOfParams - 1)
= Command3.Caption
        ParamsGrid.Text = Command3.Caption
    End If
End Sub

'TOGGLE SUPPRESSION TYPE BUTTON
Private Sub Command17_Click()
    If (Command17.Caption = "FEATURE") Then
        Command17.Caption = "PART"
    Elself (Command17.Caption = "PART") Then
        Command17.Caption = "ASSEMBLY"
    Elself (Command17.Caption = "ASSEMBLY") Then
        Command17.Caption = "FEATURE"
    End If
End Sub
```

```
' ADD NEW SUPPRESS

Private Sub Command18_Click()
If (ParamsPart >= 0) Then
      MakePartSelected (ParamsPart)
      ActivateParams (ParamsPart)
      Instances(ThisInst).PartsTree(SelPart).NumOfSupps = Instances(ThisInst).PartsTree(SelPart).NumOfSupps
+ 1
      ReDim Preserve
Instances(ThisInst).PartsTree(SelPart).SuppEntity(Instances(ThisInst).PartsTree(SelPart).NumOfSupps)
      ReDim Preserve
Instances(ThisInst).PartsTree(SelPart).SuppType(Instances(ThisInst).PartsTree(SelPart).NumOfSupps)
      ReDim Preserve
Instances(ThisInst).PartsTree(SelPart).SuppStatus(Instances(ThisInst).PartsTree(SelPart).NumOfSupps)
      SuppGrid.Rows = SuppGrid.Rows + 1
      SuppGrid.Row = SuppGrid.Rows - 1
      SuppGrid.Col = 0
      SuppGrid.Text = Str(SuppGrid.Row)
      SuppGrid.Col = 1
      Instances(ThisInst).PartsTree(SelPart).SuppEntity(Instances(ThisInst).PartsTree(SelPart).NumOfSupps - 1)
= "Entity " + Str(SuppGrid.Row)
      SuppGrid.Text = "Entity " + Str(SuppGrid.Row)
      SuppGrid.Col = 2
      Instances(ThisInst).PartsTree(SelPart).SuppStatus(Instances(ThisInst).PartsTree(SelPart).NumOfSupps - 1)
= "SUPPRESS"
      SuppGrid.Text = "SUPPRESS"
      Instances(ThisInst).PartsTree(SelPart).SuppType(Instances(ThisInst).PartsTree(SelPart).NumOfSupps - 1)
= "FEATURE"
      Command17.Caption = "FEATURE"
   End If

End Sub
```

```
' DELETE SELECTED ROW (SUPPS)

Private Sub Command20_Click()
Dim c, RowToDelete, OldNumOfRows As Integer
   Dim tempStr1, tempStr2 As String

   If (ParamsPart >= 0) Then
      MakePartSelected (ParamsPart)
ActivateParams (ParamsPart)
      RowToDelete = SuppGrid.Row
OldNumOfRows = SuppGrid.Rows
For c = RowToDelete To OldNumOfRows - 2

   SuppGrid.Row = c + 1
   SuppGrid.Col = 1
   tempStr1 = SuppGrid.Text
   SuppGrid.Col = 2
   tempStr2 = SuppGrid.Text

   SuppGrid.Row = c
      SuppGrid.Col = 1
      SuppGrid.Text = tempStr1
      SuppGrid.Col = 2
      SuppGrid.Text = tempStr2
    Instances(ThisInst).PartsTree(SelPart).SuppEntity(c - 1) =
Instances(ThisInst).PartsTree(SelPart).SuppEntity(c)
      Instances(ThisInst).PartsTree(SelPart).SuppStatus(c - 1) =
Instances(ThisInst).PartsTree(SelPart).SuppStatus(c)
         Next c
Instances(ThisInst).PartsTree(SelPart).NumOfSupps = Instances(ThisInst).PartsTree(SelPart).NumOfSupps - 1
    SuppGrid.Rows = SuppGrid.Rows - 1
    ReDim Preserve
Instances(ThisInst).PartsTree(SelPart).SuppEntity(Instances(ThisInst).PartsTree(SelPart).NumOfSupps)
     ReDim Preserve
Instances(ThisInst).PartsTree(SelPart).SuppStatus(Instances(ThisInst).PartsTree(SelPart).NumOfSupps)
   End If

End Sub
```

```
' DELETE SELECTED ROW (RELATIONS)

Private Sub Command22_Click()
Dim c, RowToDelete, OldNumOfRows As Integer
   Dim tempStr1, tempStr2 As String

   If (NumRels > 0) Then

      RowToDelete = RelsGrid.Row
      OldNumOfRows = RelsGrid.Rows
      For c = RowToDelete To OldNumOfRows - 2

         RelsGrid.Row = c + 1
         RelsGrid.Col = 0
         tempStr1 = RelsGrid.Text
         RelsGrid.Col = 1
         tempStr2 = RelsGrid.Text

         RelsGrid.Row = c
         RelsGrid.Col = 0
         RelsGrid.Text = tempStr1
            RelsGrid.Col = 1
            RelsGrid.Text = tempStr2
               Instances(ThisInst).RelM(c - 1) = Instances(ThisInst).RelM(c)
               Instances(ThisInst).RelF(c - 1) = Instances(ThisInst).RelF(c)
Next c
Instances(ThisInst).NumRels = Instances(ThisInst).NumRels - 1
      RelsGrid.Rows = RelsGrid.Rows - 1
      ReDim Preserve Instances(ThisInst).RelM(Instances(ThisInst).NumRels)
      ReDim Preserve Instances(ThisInst).RelF(Instances(ThisInst).NumRels)
   End If
End Sub
```

```
' SET PARAMETER NAME & VALUE

Private Sub Command3_Click()
If (Command3.Caption = "MM") Then
      Command3.Caption = "DEG"
   Elself (Command3.Caption = "DEG") Then
      Command3.Caption = "M"
   Elself (Command3.Caption = "M") Then
      Command3.Caption = "IN"
Elself (Command3.Caption = "IN") Then
   Command3.Caption = "VAL"
Elself (Command3.Caption = "VAL") Then
   Command3.Caption = "MM"
End If
If (ParamsPart >= 0) Then
   MakePartSelected (ParamsPart)
   ActivateParams (ParamsPart)
   ParamsGrid.Col = 0
   ParamsGrid.Text = Text2.Text
   Instances(ThisInst).PartsTree(SelPart).ParamName(ParamsGrid.Row - 1) = Text2.Text
   ParamsGrid.Col = 1
      ParamsGrid.Text = Text1.Text
      Instances(ThisInst).PartsTree(SelPart).ParamValue(ParamsGrid.Row - 1) = Val(Text1.Text)
      ParamsGrid.Col = 2
      ParamsGrid.Text = Command3.Caption
      Instances(ThisInst).PartsTree(SelPart).ParamUnit(ParamsGrid.Row - 1) = Command3.Caption
   End If
End Sub
```

```
' SET SUPPRESS ENTITY,TYPE and STATUS

Private Sub Command5_Click()
If (ParamsPart >= 0) Then
      MakePartSelected (ParamsPart)
      ActivateParams (ParamsPart)
      SuppGrid.Col = 1
      SuppGrid.Text = Text3.Text
      Instances(ThisInst).PartsTree(SelPart).SuppEntity(SuppGrid.Row - 1) = Text3.Text
      SuppGrid.Col = 2
      SuppGrid.Text = Command8.Caption
      Instances(ThisInst).PartsTree(SelPart).SuppStatus(SuppGrid.Row - 1) = Command8.Caption
      Instances(ThisInst).PartsTree(SelPart).SuppType(SuppGrid.Row - 1) = Command17.Caption
   End If
End Sub

' UPDATE Parameters in CAD file

Private Sub Command9_Click()
Dim RetVal
'AppActivate "SolidWorks 98Plus"
   Const swDocPART = 1          ' These definitions are consistent with type names
   Const swDocASSEMBLY = 2      ' defined in \SldWorks\samples\appComm\swconst.h
   Const swDocDRAWING = 3
   Dim swApp As Object          ' Define variable used to hold the application object
   Dim Part As Object           ' Define variable used to hold the part object
   Dim c As Integer
   Dim WhatType As Integer
   Dim myUnit As String
   Dim myVal As Double

   If (ParamsPart >= 0) Then
      MakePartSelected (ParamsPart)
      ActivateParams (ParamsPart)
'SOLIDWORKS
         If ((Instances(ThisInst).PartsTree(SelPart).MyCADfileType = 5) Or
          (Instances(ThisInst).PartsTree(SelPart).MyCADfileType = 6)) Then
        If (Instances(ThisInst).PartsTree(SelPart).MyCADfileType = 5) Then
          WhatType = swDocPART
        ElseIf (Instances(ThisInst).PartsTree(SelPart).MyCADfileType = 6) Then
```

```
    WhatType = swDocASSEMBLY
End If
' This will attach to current SolidWorks session or start up new session in background.
Set swApp = CreateObject("SldWorks.Application")
swApp.Visible (True)          ' Uncomment this if you wish to make the new SolidWorks session visible
' Load file from current directory.  This is currently hardcoded to c:\temp
Set Part = swApp.OpenDoc(Instances(ThisInst).PartsTree(SelPart).MyPathAndFile, WhatType)
If Part Is Nothing Then
    Exit Sub
Else
    Set Part = swApp.ActivateDoc(Instances(ThisInst).PartsTree(SelPart).MyFileName)
End If
For c = 0 To Instances(ThisInst).PartsTree(SelPart).NumOfParams - 1
    'Sort Out UNITS
    myUnit = Instances(ThisInst).PartsTree(SelPart).ParamUnit(c)
    Select Case myUnit
        Case "MM"
            myVal = Instances(ThisInst).PartsTree(SelPart).ParamValue(c) / 1000
        Case "M"
            myVal = Instances(ThisInst).PartsTree(SelPart).ParamValue(c)
        Case "IN"
            myVal = (Instances(ThisInst).PartsTree(SelPart).ParamValue(c) / 1000) * 25.4
        Case "DEG"
            myVal = Instances(ThisInst).PartsTree(SelPart).ParamValue(c) * (3.141592654 / 180)
        Case "RAD"
            myVal = Instances(ThisInst).PartsTree(SelPart).ParamValue(c)
        Case "VAL"
            myVal = Instances(ThisInst).PartsTree(SelPart).ParamValue(c)
    End Select
    Part.Parameter(Instances(ThisInst).PartsTree(SelPart).ParamName(c)).SystemValue = myVal
Next c
For c = 0 To Instances(ThisInst).PartsTree(SelPart).NumOfSupps - 1
            FeatureSuppression Instances(ThisInst).PartsTree(SelPart).SuppEntity(c),
Instances(ThisInst).PartsTree(SelPart).SuppStatus(c)
        Next c
        Part.EditRebuild
        swApp.UserControl (True)
    End If
  End If
End Sub
```

```
' IF A CELL IN THE INSTANCES PANEL IS CLICKED THENSET THE CURRENT INSTANCE TO THAT INSTANCE

Private Sub InstGrid_Click()
Dim c As Integer
Dim OldInst As Integer
Dim NewInst As Integer

    NewInst = InstGrid.Row
    OldInst = ThisInst

'Instances(OldInst).Name = Text4(0).Text
'Instances(OldInst).DrgNo = Text4(1).Text
'Instances(OldInst).By = Text4(2).Text
'Instances(OldInst).Date = Text4(3).Text
'Instances(OldInst).Description = Text4(4).Text

InstGrid.Col = 0

For c = 0 To InstGrid.Rows - 1
   InstGrid.Row = c
   If (InstGrid.CellBackColor = &H55DDFF) Then
      InstGrid.CellBackColor = vbWhite
   End If
Next c
InstGrid.Row = NewInst
InstGrid.CellBackColor = &H55DDFF
UpdateInstance (NewInst)
' Load All Data for NewInst into the form
ShowCurrentInstance OldInst, NewInst
ShowRelations NewInst
ShowParams NewInst
ThisInst = NewInst
   Caption = Instances(ThisInst).Name
   MakePartSelected SelPart
End Sub
```

```
' IF A CELL IN THE PARAMS GRID IS CLICK SET THAT PARAMETER FOR EDITING

Private Sub ParamsGrid_Click()
   ' DISPLAY SELECTED PARAM DETAILS
   If (ParamsPart >= 0) Then
      MakePartSelected (ParamsPart)
      ActivateParams (ParamsPart)
      If (ParamsGrid.Row >= 1) Then
         'Label1.Caption = "Parameter " + Str(ParamsGrid.Row)
         ParamsGrid.Col = 0
         Text2.Text = Instances(ThisInst).PartsTree(SelPart).ParamName(ParamsGrid.Row - 1)
         ParamsGrid.Col = 1
         Text1.Text = Val(Instances(ThisInst).PartsTree(SelPart).ParamValue(ParamsGrid.Row - 1))
         ParamsGrid.Col = 2
         Command3.Caption = Instances(ThisInst).PartsTree(SelPart).ParamUnit(ParamsGrid.Row - 1)
      End If
   End If
End Sub
```

```
' DELETE SELECTED ROW (PARAMETER)

Private Sub Command19_Click()
Dim c, RowToDelete, OldNumOfRows As Integer
   Dim tempStr1, tempStr2 As String

   If (ParamsPart >= 0) Then
      MakePartSelected (ParamsPart)
      ActivateParams (ParamsPart)
RowToDelete = ParamsGrid.Row
OldNumOfRows = ParamsGrid.Rows
For c = RowToDelete To OldNumOfRows - 2
   ParamsGrid.Row = c + 1
   ParamsGrid.Col = 1
   tempStr1 = ParamsGrid.Text
   ParamsGrid.Col = 2
   tempStr2 = ParamsGrid.Text

   ParamsGrid.Row = c
   ParamsGrid.Col = 1
      ParamsGrid.Text = tempStr1
      ParamsGrid.Col = 2
      ParamsGrid.Text = tempStr2
      Instances(ThisInst).PartsTree(SelPart).ParamName(c - 1) =
Instances(ThisInst).PartsTree(SelPart).ParamName(c)
         Instances(ThisInst).PartsTree(SelPart).ParamValue(c - 1) =
Instances(ThisInst).PartsTree(SelPart).ParamValue(c)
Next c
Instances(ThisInst).PartsTree(SelPart).NumOfParams = Instances(ThisInst).PartsTree(SelPart).NumOfParams - 1
ParamsGrid.Rows = ParamsGrid.Rows - 1
ReDim Preserve
Instances(ThisInst).PartsTree(SelPart).ParamName(Instances(ThisInst).PartsTree(SelPart).NumOfParams)
ReDim Preserve
Instances(ThisInst).PartsTree(SelPart).ParamValue(Instances(ThisInst).PartsTree(SelPart).NumOfParams)
   End If
End Sub

Private Sub Command4_Click()
   Dim dy, dx As Integer
   Dim c As Integer
```

```
VScroll2.Value = 0
HScroll2.Value = 32768 / 2
dy = v2 - VScroll2.Value
dx = h2 - HScroll2.Value
v2 = 0
h2 = 32768 / 2
For c = 0 To Instances(ThisInst).NumFuncts - 1
   FNode1(c).Move FNode1(c).Left + dx, FNode1(c).Top + dy, FNode1(c).Width, FNode1(c).Height
   Next c
End Sub
Private Sub Command12_Click()
   Dim dy, dx As Integer
   Dim c As Integer

   VScroll1.Value = 0
   HScroll1.Value = 32768 / 2
   dy = v1 - VScroll1.Value
   dx = h1 - HScroll1.Value
   v1 = 0
   h1 = 32768 / 2
   For c = 0 To Instances(ThisInst).NumParts - 1
      MNode1(c).Move MNode1(c).Left + dx, MNode1(c).Top + dy, MNode1(c).Width, MNode1(c).Height
   Next c
End Sub
```

```
'SET OPERATION TO BE DONE DEPENDING ON WHAT BUTTON HAS BEN CLICKED
Public Sub ToolbarClicks(buttonID As Integer)
   Select Case buttonID
      Case 1
         SelPart = -1
         DoWhat = "ADD_CO_LEFT"
      Case 2
         SelPart = -1
         DoWhat = "ADD_CHILD"
      Case 3
         SelPart = -1
         DoWhat = "ADD_CO_RIGHT"
      Case 5
         SelPart = -1
         DoWhat = "MOVE_DOWN"
      Case 6
         SelPart = -1
         DoWhat = "MOVE_UP"
      Case 7
         SelPart = -1
         DoWhat = "MOVE_LEFT"
      Case 8
         SelPart = -1
         DoWhat = "MOVE_RIGHT"
      Case 10
         DoWhat = "NOTHING"
      Case 11
         DoWhat = "DELETE"
      Case 13
         DoWhat = "ADD_RELATION"
      Case 14
         DoWhat = "REMOVE_RELATION"
      Case 16
         AddNewInstance
         DoWhat = "NOTHING"
      Case 17
         DeleteSelectedInstance
         DoWhat = "NOTHING"
      Case 19
         If (ThisInst = 0) Then
```

```
        MsgBox ("Cannot Suppress Parts/Systems for the Generic Instance")
        DoWhat = "NOTHING"
      Else
        DoWhat = "SUPPRESS"
      End If
    Case 20
      If (ThisInst = 0) Then
        MsgBox ("Cannot Resume Parts/Systems for the Generic Instance")
        DoWhat = "NOTHING"
      Else
        DoWhat = "RESUME"
      End If
    End Select
End Sub
```

```
'ADD A NEW CHILD MEANS
Private Sub AddChildMeans(Parent As Integer)
   Dim c, Child As Integer
         Instances(ThisInst).NumParts =    Instances(ThisInst).NumParts + 1
   Child = Instances(ThisInst).NumParts - 1
   ReDim Preserve Instances(ThisInst).PartsTree(Child)
   Load MNode1(Child)
   Instances(ThisInst).PartsTree(Child).Name = "New Part"
   Instances(ThisInst).PartsTree(Child).Xpos = Instances(ThisInst).PartsTree(Parent).Xpos
   Instances(ThisInst).PartsTree(Child).Ypos = Instances(ThisInst).PartsTree(Parent).Ypos +
MNode1(Instances(ThisInst).PartsTree(Parent).NodeID).Height + VGap
   Instances(ThisInst).PartsTree(Child).NodeID = Child
   Instances(ThisInst).PartsTree(Child).PartSuppression = Instances(ThisInst).PartsTree(Parent).PartSuppression
   UpdateMeansNode (Child)
   ' Setup Parent and Child relationships
   Instances(ThisInst).PartsTree(Parent).NumChildren = Instances(ThisInst).PartsTree(Parent).NumChildren + 1
   ReDim Preserve
Instances(ThisInst).PartsTree(Parent).Children(Instances(ThisInst).PartsTree(Parent).NumChildren - 1)
Instances(ThisInst).PartsTree(Parent).Children(Instances(ThisInst).PartsTree(Parent).NumChildren - 1) = Child
   Instances(ThisInst).PartsTree(Child).NumChildren = 0
   Instances(ThisInst).PartsTree(Child).NumParents = 1
   ReDim Preserve Instances(ThisInst).PartsTree(Child).Parents(0)
   Instances(ThisInst).PartsTree(Child).Parents(0) = Parent
   Instances(ThisInst).PartsTree(Child).NumOfParams = 0
   Instances(ThisInst).PartsTree(Child).NumOfSupps = 0
   Instances(ThisInst).PartsTree(Child).NumFunctions = 0
   Instances(ThisInst).PartsTree(Child).Level = Instances(ThisInst).PartsTree(Parent).Level + 1
   For c = 0 To Instances(ThisInst).NumParts - 1   MNode1(Instances(ThisInst).PartsTree(c).NodeID).BackColor
= vbWhite
   Next c
   SelPart = -1
   'LinkLine
   Load Line1(Child)
   Line1(Child).X1 = Instances(ThisInst).PartsTree(Parent).Xpos +
MNode1(Instances(ThisInst).PartsTree(Parent).NodeID).Width / 2
   Line1(Child).Y1 = Instances(ThisInst).PartsTree(Parent).Ypos +
MNode1(Instances(ThisInst).PartsTree(Parent).NodeID).Height
   Line1(Child).X2 = Instances(ThisInst).PartsTree(Child).Xpos +
MNode1(Instances(ThisInst).PartsTree(Child).NodeID).Width / 2
   Line1(Child).Y2 = Instances(ThisInst).PartsTree(Child).Ypos
```

```
    Line1 (Child).Visible = True
    UpdateLinesArrays ThisInst
End Sub

' ADD A MEANS NODE TO THE LEFT OR RIGHT
Private Sub AddCoMeans(Co As Integer)
    Dim c, NewCo, Parent As Integer

    Instances(ThisInst).NumParts = Instances(ThisInst).NumParts + 1
    NewCo = Instances(ThisInst).NumParts - 1
    ReDim Preserve Instances(ThisInst).PartsTree(NewCo)
    Load MNode1 (NewCo)
    Instances(ThisInst).PartsTree(NewCo).Name = "New Part"
    If (DoWhat = "ADD_CO_LEFT") Then
        Instances(ThisInst).PartsTree(NewCo).Xpos = Instances(ThisInst).PartsTree(Co).Xpos -
MNode1 (Instances(ThisInst).PartsTree(Co).NodeID).Width - HGap
    ElseIf (DoWhat = "ADD_CO_RIGHT") Then
        Instances(ThisInst).PartsTree(NewCo).Xpos = Instances(ThisInst).PartsTree(Co).Xpos +
MNode1 (Instances(ThisInst).PartsTree(Co).NodeID).Width + HGap
    End If
    Instances(ThisInst).PartsTree(NewCo).Ypos = Instances(ThisInst).PartsTree(Co).Ypos
    Instances(ThisInst).PartsTree(NewCo).NodeID = NewCo
    Instances(ThisInst).PartsTree(NewCo).PartSuppression = Instances(ThisInst).PartsTree(Co).PartSuppression
    UpdateMeansNode (NewCo)
    ' Setup Parent and Child relationships
    Parent = Instances(ThisInst).PartsTree(Co).Parents(0)
    Instances(ThisInst).PartsTree(Parent).NumChildren = Instances(ThisInst).PartsTree(Parent).NumChildren + 1
    ReDim Preserve
Instances(ThisInst).PartsTree(Parent).Children(Instances(ThisInst).PartsTree(Parent).NumChildren - 1)
Instances(ThisInst).PartsTree(Parent).Children(Instances(ThisInst).PartsTree(Parent).NumChildren - 1) = NewCo
    Instances(ThisInst).PartsTree(NewCo).NumChildren = 0
    Instances(ThisInst).PartsTree(NewCo).NumParents = 1
    ReDim Preserve Instances(ThisInst).PartsTree(NewCo).Parents(0)
    Instances(ThisInst).PartsTree(NewCo).Parents(0) = Parent
    Instances(ThisInst).PartsTree(NewCo).Level = Instances(ThisInst).PartsTree(Co).Level
    For c = 0 To Instances(ThisInst).NumParts - 1
MNode1 (Instances(ThisInst).PartsTree(c).NodeID).BackColor = vbWhite
    Next c
    SelPart = -1
    'LinkLine
```

```
    Load Line1(NewCo)
    Line1(NewCo).X1 = Instances(ThisInst).PartsTree(Parent).Xpos +
MNode1(Instances(ThisInst).PartsTree(Parent).NodeID).Width / 2
    Line1(NewCo).Y1 = Instances(ThisInst).PartsTree(Parent).Ypos +
MNode1(Instances(ThisInst).PartsTree(Parent).NodeID).Height
    Line1(NewCo).X2 = Instances(ThisInst).PartsTree(NewCo).Xpos +
MNode1(Instances(ThisInst).PartsTree(NewCo).NodeID).Width / 2
    Line1(NewCo).Y2 = Instances(ThisInst).PartsTree(NewCo).Ypos
    Line1(NewCo).Visible = True
    UpdateLinesArrays ThisInst
End Sub

' UPDATE MEANS NODE
Private Sub UpdateMeansNode(myTreeID As Integer)
    Dim myNodeID As Integer
    myNodeID = Instances(ThisInst).PartsTree(myTreeID).NodeID
    MNode1(myNodeID).Text = Instances(ThisInst).PartsTree(myTreeID).Name
    MNode1(myNodeID).Move Instances(ThisInst).PartsTree(myTreeID).Xpos,
Instances(ThisInst).PartsTree(myTreeID).Ypos, MNode1(myNodeID).Width, MNode1(myNodeID).Height
    MNode1(myNodeID).Visible = True
    MNode1(myNodeID).Refresh
End Sub

' ADD A NEW CHILD FUNCTION
Private Sub AddChildFunct(Parent As Integer)
    Dim c, Child As Integer
Instances(ThisInst).NumFuncts = Instances(ThisInst).NumFuncts + 1
    Child = Instances(ThisInst).NumFuncts - 1
    ReDim Preserve Instances(ThisInst).FunctTree(Child)
    Load FNode1(Child)
    Instances(ThisInst).FunctTree(Child).Name = "New Function"
    Instances(ThisInst).FunctTree(Child).Xpos = Instances(ThisInst).FunctTree(Parent).Xpos
    Instances(ThisInst).FunctTree(Child).Ypos = Instances(ThisInst).FunctTree(Parent).Ypos +
FNode1(Instances(ThisInst).FunctTree(Parent).NodeID).Height + VGap
    Instances(ThisInst).FunctTree(Child).NodeID = Child
    UpdateFunctNode (Child)
    ' Setup Parent and Child relationships
    Instances(ThisInst).FunctTree(Parent).NumChildren = Instances(ThisInst).FunctTree(Parent).NumChildren + 1
    ReDim Preserve
Instances(ThisInst).FunctTree(Parent).Children(Instances(ThisInst).FunctTree(Parent).NumChildren - 1)
```

```
    Instances(ThisInst).FunctTree(Parent).Children(Instances(ThisInst).FunctTree(Parent).NumChildren - 1) =
Child
    Instances(ThisInst).FunctTree(Child).NumChildren = 0
    Instances(ThisInst).FunctTree(Child).NumParents = 1
    ReDim Preserve Instances(ThisInst).FunctTree(Child).Parents(0)
    Instances(ThisInst).FunctTree(Child).Parents(0) = Parent
    Instances(ThisInst).FunctTree(Child).Level = Instances(ThisInst).FunctTree(Parent).Level + 1
    For c = 0 To Instances(ThisInst).NumFuncts - 1
        FNode1(Instances(ThisInst).FunctTree(c).NodeID).BackColor = myCol
    Next c
    SelFunct = -1
    'LinkLine
    Load Line2(Child)
    Line2(Child).X1 = Instances(ThisInst).FunctTree(Parent).Xpos +
FNode1(Instances(ThisInst).FunctTree(Parent).NodeID).Width / 2
    Line2(Child).Y1 = Instances(ThisInst).FunctTree(Parent).Ypos +
FNode1(Instances(ThisInst).FunctTree(Parent).NodeID).Height
    Line2(Child).X2 = Instances(ThisInst).FunctTree(Child).Xpos +
FNode1(Instances(ThisInst).FunctTree(Child).NodeID).Width / 2
    Line2(Child).Y2 = Instances(ThisInst).FunctTree(Child).Ypos
    Line2(Child).Visible = True
    UpdateLinesArrays ThisInst
End Sub

' ADD A FUNTION NODE TO THE LEFT OR RIGHT
Private Sub AddCoFunct(Co As Integer)
    Dim c, NewCo, Parent As Integer

    Instances(ThisInst).NumFuncts = Instances(ThisInst).NumFuncts + 1
    NewCo = Instances(ThisInst).NumFuncts - 1
    ReDim Preserve Instances(ThisInst).FunctTree(NewCo)
    Load FNode1(NewCo)
    Instances(ThisInst).FunctTree(NewCo).Name = "New Function"
    If (DoWhat = "ADD_CO_LEFT") Then
        Instances(ThisInst).FunctTree(NewCo).Xpos = Instances(ThisInst).FunctTree(Co).Xpos -
FNode1(Instances(ThisInst).FunctTree(Co).NodeID).Width - HGap
    ElseIf (DoWhat = "ADD_CO_RIGHT") Then
        Instances(ThisInst).FunctTree(NewCo).Xpos = Instances(ThisInst).FunctTree(Co).Xpos +
FNode1(Instances(ThisInst).FunctTree(Co).NodeID).Width + HGap
    End If
```

```
    Instances(ThisInst).FunctTree(NewCo).Ypos = Instances(ThisInst).FunctTree(Co).Ypos
    Instances(ThisInst).FunctTree(NewCo).NodeID = NewCo
    UpdateFunctNode (NewCo)
    ' Setup Parent and Child relationships
    Parent = Instances(ThisInst).FunctTree(Co).Parents(0)
    Instances(ThisInst).FunctTree(Parent).NumChildren = Instances(ThisInst).FunctTree(Parent).NumChildren + 1
    ReDim Preserve
Instances(ThisInst).FunctTree(Parent).Children(Instances(ThisInst).FunctTree(Parent).NumChildren - 1)
    Instances(ThisInst).FunctTree(Parent).Children(Instances(ThisInst).FunctTree(Parent).NumChildren - 1) =
NewCo
    Instances(ThisInst).FunctTree(NewCo).NumChildren = 0
    Instances(ThisInst).FunctTree(NewCo).NumParents = 1
    ReDim Preserve Instances(ThisInst).FunctTree(NewCo).Parents(0)
    Instances(ThisInst).FunctTree(NewCo).Parents(0) = Parent
    Instances(ThisInst).FunctTree(NewCo).Level = Instances(ThisInst).FunctTree(Co).Level
    For c = 0 To Instances(ThisInst).NumFuncts - 1
        FNode1(Instances(ThisInst).FunctTree(c).NodeID).BackColor = myCol
    Next c
    SelFunct = -1
    'LinkLine
    Load Line2(NewCo)
    Line2(NewCo).X1 = Instances(ThisInst).FunctTree(Parent).Xpos +
FNode1(Instances(ThisInst).FunctTree(Parent).NodeID).Width / 2
    Line2(NewCo).Y1 = Instances(ThisInst).FunctTree(Parent).Ypos +
FNode1(Instances(ThisInst).FunctTree(Parent).NodeID).Height
    Line2(NewCo).X2 = Instances(ThisInst).FunctTree(NewCo).Xpos +
FNode1(Instances(ThisInst).FunctTree(NewCo).NodeID).Width / 2
    Line2(NewCo).Y2 = Instances(ThisInst).FunctTree(NewCo).Ypos
    Line2(NewCo).Visible = True
    UpdateLinesArrays ThisInst
End Sub

' UPDATE FUNCTION NODE
Private Sub UpdateFunctNode(myTreeID As Integer)
    Dim myNodeID As Integer
    myNodeID = Instances(ThisInst).FunctTree(myTreeID).NodeID
    FNode1(myNodeID).Text = Instances(ThisInst).FunctTree(myTreeID).Name
    FNode1(myNodeID).Move Instances(ThisInst).FunctTree(myTreeID).Xpos,
Instances(ThisInst).FunctTree(myTreeID).Ypos, FNode1(myNodeID).Width, FNode1(myNodeID).Height
    FNode1(myNodeID).Visible = True
```

```
    FNode1(myNodeID).Refresh
End Sub
Private Sub MoveMeans(myID As Integer)
    Dim c, NodeID, dy, dx As Integer
    NodeID = Instances(ThisInst).PartsTree(myID).NodeID
    dy = 0
    dx = 0
    If (DoWhat = "MOVE_UP") Then
        dy = dy - (MNode1(0).Height / 2 + VGap / 2)
    ElseIf (DoWhat = "MOVE_DOWN") Then
        dy = dy + (MNode1(0).Height / 2 + VGap / 2)
    ElseIf (DoWhat = "MOVE_LEFT") Then
        dx = dx - (MNode1(0).Width / 2 + HGap / 2)
    ElseIf (DoWhat = "MOVE_RIGHT") Then
        dx = dx + (MNode1(0).Width / 2 + HGap / 2)
    End If
    Instances(ThisInst).PartsTree(myID).Xpos = Instances(ThisInst).PartsTree(myID).Xpos + dx
    Instances(ThisInst).PartsTree(myID).Ypos = Instances(ThisInst).PartsTree(myID).Ypos + dy
    MNode1(NodeID).Left = MNode1(NodeID).Left + dx
    MNode1(NodeID).Top = MNode1(NodeID).Top + dy
    Line1(NodeID).X2 = Line1(NodeID).X2 + dx
    Line1(NodeID).Y2 = Line1(NodeID).Y2 + dy
    If (Instances(ThisInst).PartsTree(myID).NumChildren > 0) Then
        For c = 0 To Instances(ThisInst).PartsTree(myID).NumChildren - 1
            Line1(Instances(ThisInst).PartsTree(Instances(ThisInst).PartsTree(myID).Children(c)).NodeID).X1 =
Line1(Instances(ThisInst).PartsTree(Instances(ThisInst).PartsTree(myID).Children(c)).NodeID).X1 + dx
            Line1(Instances(ThisInst).PartsTree(Instances(ThisInst).PartsTree(myID).Children(c)).NodeID).Y1 =
Line1(Instances(ThisInst).PartsTree(Instances(ThisInst).PartsTree(myID).Children(c)).NodeID).Y1 + dy
        Next c
    End If
End Sub
Private Sub MoveFunct(myID As Integer)
Dim c, myNodeID, dy, dx As Integer
myNodeID = Instances(ThisInst).FunctTree(myID).NodeID
dy = 0
dx = 0
If (DoWhat = "MOVE_UP") Then
    dy = dy - (FNode1(0).Height / 2 + VGap / 2)
ElseIf (DoWhat = "MOVE_DOWN") Then
    dy = dy + (FNode1(0).Height / 2 + VGap / 2)
```

```
ElseIf (DoWhat = "MOVE_LEFT") Then
   dx = dx - (FNode1(0).Width / 2 + HGap / 2)
ElseIf (DoWhat = "MOVE_RIGHT") Then
   dx = dx + (FNode1(0).Width / 2 + HGap / 2)
End If
Instances(ThisInst).FunctTree(myID).Xpos = Instances(ThisInst).FunctTree(myID).Xpos + dx
Instances(ThisInst).FunctTree(myID).Ypos = Instances(ThisInst).FunctTree(myID).Ypos + dy
FNode1(myNodeID).Left = FNode1(myNodeID).Left + dx
FNode1(myNodeID).Top = FNode1(myNodeID).Top + dy
   Line2(myNodeID).X2 = Line2(myNodeID).X2 + dx
   Line2(myNodeID).Y2 = Line2(myNodeID).Y2 + dy
   If (Instances(ThisInst).FunctTree(myID).NumChildren > 0) Then
      For c = 0 To Instances(ThisInst).FunctTree(myID).NumChildren - 1
         Line2(Instances(ThisInst).FunctTree(Instances(ThisInst).FunctTree(myID).Children(c)).NodeID).X1 =
Line2(Instances(ThisInst).FunctTree(Instances(ThisInst).FunctTree(myID).Children(c)).NodeID).X1 + dx
         Line2(Instances(ThisInst).FunctTree(Instances(ThisInst).FunctTree(myID).Children(c)).NodeID).Y1 =
Line2(Instances(ThisInst).FunctTree(Instances(ThisInst).FunctTree(myID).Children(c)).NodeID).Y1 + dy
      Next c
   End If
End Sub
```

```
' RESIZE AND UPDATE DISPLAY
Private Sub Form_Resize()
   Dim c, dx, dy As Integer
   Dim RealWidth As Integer
   On Error Resume Next

   If (DispInstances = True) Then
      RealWidth = ScaleWidth - Frame2.Width
   Else
      RealWidth = ScaleWidth - 120
   End If
   Frame1.Left = 0
   Frame1.Top = ScaleHeight - Frame1.Height
   Frame1.Width = RealWidth
   Picture3.Width = VScroll1.Width / 2
   Picture3.Left = Split * RealWidth
   Picture3.Height = Frame1.Top - Picture3.Top
   Picture1.Width = Picture3.Left - VScroll1.Left - VScroll1.Width
   Picture1.Height = Picture3.Height - HScroll1.Height - Command10.Height
```

```
VScroll1.Height = Picture1.Height
HScroll1.Top = Picture1.Top + Picture1.Height
HScroll1.Width = Picture1.Width
Command1.Top = HScroll1.Top
VScroll2.Left = RealWidth - VScroll2.Width
VScroll2.Height = VScroll1.Height
Picture2.Left = Picture3.Left + Picture3.Width
Picture2.Width = VScroll2.Left - Picture2.Left
Picture2.Height = Picture1.Height
HScroll2.Left = Picture2.Left + 10
HScroll2.Width = Picture2.Width - 10
HScroll2.Top = HScroll1.Top
Command2.Left = VScroll2.Left
Command2.Top = Command1.Top
Command4.Left = VScroll2.Left
Command10.Left = Picture1.Left
Command10.Width = Picture1.Width
Command11.Left = Picture2.Left + 10
Command11.Width = Picture2.Width - 10
RelsGrid.Width = VScroll2.Left + VScroll2.Width - RelsGrid.Left
RelsGrid.ColWidth(0) = RelsGrid.Width / 2 - 50
RelsGrid.ColWidth(1) = RelsGrid.Width / 2 - 50
If ((Label3.Width + Command22.Width) < RelsGrid.Width) Then
   Command22.Width = RelsGrid.Width - Label3.Width
   Command22.Left = RelsGrid.Left + RelsGrid.Width - Command22.Width
Else
   Command22.Width = 0
   Command22.Left = RelsGrid.Left + RelsGrid.Width
End If
If (DispInstances = True) Then
   Frame2.Left = ScaleWidth - Frame2.Width
   Frame2.Height = ScaleHeight
   Frame3.Top = ScaleHeight - Frame3.Height
   InstGrid.Height = Frame3.Top - 240
End If

End Sub
```

```
' PERFORM DO-WHAT WHEN A MEANS NODE IS CLICKED
Private Sub MNode1_Click(Index As Integer)
   Dim c As Integer

   If ((DoWhat = "NOTHING") And (MNode1(Index).BackColor = vbYellow)) Then
      MNode1(Index).BackColor = vbWhite
      SelPart = -1 'ie NO parts selected
   Else
      For c = 0 To (Instances(ThisInst).NumParts - 1)
   If (Instances(ThisInst).PartsTree(c).PartSuppression = "False") Then
      MNode1(Instances(ThisInst).PartsTree(c).NodeID).BackColor = vbWhite
   Else
      MNode1(Instances(ThisInst).PartsTree(c).NodeID).BackColor = myCol
   End If
   If (Instances(ThisInst).PartsTree(c).NodeID = Index) Then
      SelPart = c
   End If
Next c
If (Instances(ThisInst).PartsTree(Index).PartSuppression = "False") Then
   MNode1(Index).BackColor = vbYellow
   End If
   If (DoWhat = "NOTHING") Then
      ActivateParams (SelPart)
      DisplayPartsParams (ParamsPart)
   End If
End If
If (DoWhat = "ADD_CHILD") Then
   AddChildMeans (SelPart)
   DoWhat = "NOTHING"
End If
If ((DoWhat = "ADD_CO_LEFT") Or (DoWhat = "ADD_CO_RIGHT")) Then
   AddCoMeans (SelPart)
   DoWhat = "NOTHING"
End If
If ((DoWhat = "MOVE_UP") Or (DoWhat = "MOVE_DOWN") Or (DoWhat = "MOVE_LEFT") Or (DoWhat =
"MOVE_RIGHT")) Then
   MoveMeans (SelPart)
End If
If (DoWhat = "DELETE") Then
   DeletePart (SelPart)
```

```
    DoWhat = "NOTHING"
End If
If (DoWhat <> "ADD_RELATION") Then
    RelMeans = -1
    RelFunct = -1
End If
If (DoWhat = "ADD_RELATION") Then
    Beep
    RelMeans = SelPart
    AddNewRelation
End If
If (DoWhat = "SUPPRESS") Then
    Beep
    SuppressPart (SelPart)
    DoWhat = "NOTHING"
End If
If (DoWhat = "RESUME") Then
    ResumePart (SelPart)
    DoWhat = "NOTHING"
End If
End Sub
```

```
' PERFORM DO-WHAT WHEN FUNCTION NODE IS CLICKED
Private Sub FNode1_Click(Index As Integer)
    Dim c As Integer
    Beep
    If ((DoWhat = "NOTHING") And (FNode1(Index).Colour = "Yellow")) Then
        FNode1(Index).Colour = "White"
        FNode1(Index).Refresh
        SelFunct = -1 'ie NO functions selected
    Else
        For c = 0 To (Instances(ThisInst).NumFuncts - 1)
        FNode1(Instances(ThisInst).FunctTree(c).NodeID).Colour = "White"
        FNode1(Instances(ThisInst).FunctTree(c).NodeID).Refresh
        If (Instances(ThisInst).FunctTree(c).NodeID = Index) Then
            SelFunct = c
        End If
    Next c
    FNode1(Index).Colour = "Yellow"
    FNode1(Index).Refresh
```

```
End If
If (DoWhat = "ADD_CHILD") Then
   AddChildFunct (SelFunct)
   DoWhat = "NOTHING"
End If
If ((DoWhat = "ADD_CO_LEFT") Or (DoWhat = "ADD_CO_RIGHT")) Then
   AddCoFunct (SelFunct)
   DoWhat = "NOTHING"
End If
   If ((DoWhat = "MOVE_UP") Or (DoWhat = "MOVE_DOWN") Or (DoWhat = "MOVE_LEFT") Or (DoWhat =
"MOVE_RIGHT")) Then
      MoveFunct (SelFunct)
   End If
   If (DoWhat = "DELETE") Then
      DeleteFunct (SelFunct)
      DoWhat = "NOTHING"
   End If
   If (DoWhat <> "ADD_RELATION") Then
      RelMeans = -1
      RelFunct = -1
   End If
   If (DoWhat = "ADD_RELATION") Then
      RelFunct = SelFunct
      AddNewRelation
   End If
End Sub

' DESELECT NODE
Private Sub Picture1_Click()
   If (SelPart >= 0) Then
      If (Instances(ThisInst).PartsTree(SelPart).PartSuppression = "False") Then
         MNode1(Instances(ThisInst).PartsTree(SelPart).NodeID).BackColor = vbWhite
      Else
         MNode1(Instances(ThisInst).PartsTree(SelPart).NodeID).BackColor = myCol
      End If
      SelPart = -1 'ie NO parts selected
      Line3.Visible = False
      Line4.Visible = False
   End If
End Sub
```

```
' DESELECT NODE
Private Sub Picture2_Click()
   If (SelFunct >= 0) Then
      FNode1 (Instances(ThisInst).FunctTree(SelFunct).NodeID).Colour = "White"
      FNode1 (Instances(ThisInst).FunctTree(SelFunct).NodeID).Refresh
      SelFunct = -1 'ie NO functions selected
      Line3.Visible = False
      Line4.Visible = False
   End If
End Sub

' OPEN A SOLIDWORKS PART OR ASSEMBLY
Private Sub OpenSolidWorksFile()
   Dim RetVal
   'AppActivate "SolidWorks 98Plus"
   Const swDocPART = 1          ' These definitions are consistent with type names
   Const swDocASSEMBLY = 2      ' defined in \SldWorks\samples\appComm\swconst.h
   Const swDocDRAWING = 3
   Dim swApp As Object          ' Define variable used to hold the application object
   Dim Part As Object           ' Define variable used to hold the part object
   Dim c As Integer
   Dim WhatType As Integer

   If (Instances(ThisInst).PartsTree(SelPart).MyCADfileType = 5) Then
      WhatType = swDocPART
   ElseIf (Instances(ThisInst).PartsTree(SelPart).MyCADfileType = 6) Then
      WhatType = swDocASSEMBLY
   End If
' This will attach to current SolidWorks session or start up new session in background.
   Set swApp = CreateObject("SldWorks.Application")
swApp.Visible (True)         ' Uncomment this if you wish to make the new SolidWorks session visible
' Load file from current directory.  This is currently hardcoded to c:\temp
   Set Part = swApp.OpenDoc(Instances(ThisInst).PartsTree(SelPart).MyPathAndFile, WhatType)
   If Part Is Nothing Then
     Exit Sub
   Else
     Set Part = swApp.ActivateDoc(Instances(ThisInst).PartsTree(SelPart).MyFileName)
   End If
   swApp.UserControl (True)
End Sub
```

```
' Get CAD filename
Private Sub Command6_Click()
Dim RetVal
If (SelPart >= 0) Then
    ' Set CancelError is True
    CommonDialog1.CancelError = True
    On Error GoTo ErrHandler
    ' Set flags
    CommonDialog1.Flags = cdlOFNHideReadOnly
    ' Set filters
    CommonDialog1.Filter = "All Files (*.*) | *.*" & _
    " | Pro/ENGINEER Part (*.prt) | *.prt" & _
    " | Pro/ENGINEER Assembly (*.asm) | *.asm" & _
    " | Mechanical Desktop (*.dwg) | *.dwg" & _
    " | SolidWorks Part (*.SLDPRT) | *.SLDPRT" & _
    " | SolidWorks Assembly (*.SLDASM) | *.SLDASM"
    ' Specify default filter
    CommonDialog1.FilterIndex = 5
    ' Display the Open dialog box
    CommonDialog1.ShowOpen
      ' Display name of selected file
      Instances(ThisInst).PartsTree(SelPart).MyFileName = CommonDialog1.FileTitle
      Instances(ThisInst).PartsTree(SelPart).MyPathAndFile = CommonDialog1.filename
      'Exit Sub
      Text5.Text = CommonDialog1.filename
      Instances(ThisInst).PartsTree(SelPart).MyCADfileType = CommonDialog1.FilterIndex
  Else
     MsgBox ("Select a Part Node First")
  End If
ErrHandler:
   'User pressed the Cancel button
   Exit Sub
End Sub
```

```
' OPEN CAD FILE button
Private Sub Command7_Click()
If (ParamsPart >= 0) Then
      MakePartSelected (ParamsPart)
      If ((Instances(ThisInst).PartsTree(ParamsPart).MyCADfileType = 5) Or
(Instances(ThisInst).PartsTree(ParamsPart).MyCADfileType = 6)) Then
         OpenSolidWorksFile
      End If
   End If
End Sub

' GET FEATURE SUPPRESSION STATUS
Private Sub Command8_Click()
   If (Command8.Caption = "SUPPRESS") Then
      Command8.Caption = "RESUME"
   ElseIf (Command8.Caption = "RESUME") Then
      Command8.Caption = "SUPPRESS"
   End If
End Sub
Private Sub ActivateParams(myPart As Integer)
   ParamsPart = myPart
   SelPart = myPart
   Text5.Text = Instances(ThisInst).PartsTree(SelPart).MyPathAndFile
End Sub

' SHOW THE SELECTED PARTS PARAMETERS
Private Sub DisplayPartsParams(myPart As Integer)
   Dim c As Integer
   Text1.Text = ""
   Text2.Text = ""
   ParamsGrid.Rows = 1
   If (Instances(ThisInst).PartsTree(SelPart).NumOfParams > 0) Then
ParamsGrid.Rows = Instances(ThisInst).PartsTree(SelPart).NumOfParams + 1
For c = 0 To Instances(ThisInst).PartsTree(SelPart).NumOfParams - 1
   ParamsGrid.Row = c + 1
   ParamsGrid.Col = 0
   ParamsGrid.Text = Instances(ThisInst).PartsTree(SelPart).ParamName(c)
   ParamsGrid.Col = 1
   ParamsGrid.Text = Instances(ThisInst).PartsTree(SelPart).ParamValue(c)
   ParamsGrid.Col = 2
```

```
        ParamsGrid.Text = Instances(ThisInst).PartsTree(SelPart).ParamUnit(c)
    Next c
End If
Text3.Text = ""
SuppGrid.Rows = 1
If (Instances(ThisInst).PartsTree(SelPart).NumOfSupps > 0) Then
    SuppGrid.Rows = Instances(ThisInst).PartsTree(SelPart).NumOfSupps + 1
    For c = 0 To Instances(ThisInst).PartsTree(SelPart).NumOfSupps - 1
        SuppGrid.Row = c + 1
        SuppGrid.Col = 0
        SuppGrid.Text = c + 1
        SuppGrid.Col = 1
        SuppGrid.Text = Instances(ThisInst).PartsTree(SelPart).SuppEntity(c)
        SuppGrid.Col = 2
        SuppGrid.Text = Instances(ThisInst).PartsTree(SelPart).SuppStatus(c)
    Next c
  End If
End Sub

' SELECT A PART AND UPDATE DISPLAY
Private Sub MakePartSelected(myPart As Integer)
    Dim c As Integer
    For c = 0 To Instances(ThisInst).NumParts - 1
        If (MNode1(c).BackColor <> vbWhite) Then
            If (Instances(ThisInst).PartsTree(c).PartSuppression = "False") Then
                MNode1(c).BackColor = vbWhite
            End If
        End If
    Next c
    If (myPart >= 0) Then
        If (Instances(ThisInst).PartsTree(myPart).PartSuppression = "False") Then
            MNode1(myPart).BackColor = vbYellow
        End If
    End If
    SelPart = myPart
End Sub
```

```
' MOVE THE SPLIT SCREEN
Private Sub Picture3_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
    Dim dx As Integer
    Dim RealWidth As Integer

    If (DispInstances = True) Then
        RealWidth = ScaleWidth - Frame2.Width
    Else
        RealWidth = ScaleWidth
End If
Picture3.MousePointer = vbCustom
If (MoveSplit = True) Then
    'If (Image1.Left < (VScroll2.Left + VScroll2.Width)) Then
    '   Beep
    'ElseIf (Image1.Left > (VScroll1.Left - VScroll1.Width)) Then
    '   Beep
    'Else
        dx = X - Image1.Width / 2
        Picture3.Move Picture3.Left + X - Picture3.Width / 2, Picture3.Top, Picture3.Width, Picture3.Height
        Split = Picture3.Left / RealWidth
        Form_Resize
        Command10.Refresh
        Command11.Refresh
        HScroll1.Refresh
        HScroll2.Refresh
        Picture1.Refresh
        Picture2.Refresh

    'End If
  End If
End Sub

' DISPLAY SELECTED PARAM DETAILS
Private Sub SuppGrid_Click()
If (ParamsPart >= 0) Then
      MakePartSelected (ParamsPart)
      ActivateParams (ParamsPart)
      If (SuppGrid.Row >= 1) Then
         'Label1.Caption = "Parameter " + Str(ParamsGrid.Row)
```

```
            SuppGrid.Col = 1
            Text3.Text = Instances(ThisInst).PartsTree(SelPart).SuppEntity(SuppGrid.Row - 1)
            SuppGrid.Col = 2
            Command8.Caption = Instances(ThisInst).PartsTree(SelPart).SuppStatus(SuppGrid.Row - 1)
            Command17.Caption = Instances(ThisInst).PartsTree(SelPart).SuppType(SuppGrid.Row - 1)
        End If
    End If
End Sub
```

```
' UPDATE INSTANCE TEXT
Private Sub Text4_Change(Index As Integer)
    Dim myInst As Integer
    myInst = InstGrid.Row
    Select Case Index
        Case 0
            Instances(myInst).Name = Text4(Index).Text
            InstGrid.Text = Text4(Index).Text
        Case 1
            Instances(myInst).DrgNo = Text4(1).Text
        Case 2
            Instances(myInst).By = Text4(2).Text
        Case 3
            Instances(myInst).Date = Text4(3).Text
        Case 4
            Instances(myInst).Description = Text4(4).Text
    End Select
End Sub


Private Sub VScroll1_Change()
    Dim dy As Integer
    Dim c As Integer
    dy = v1 - VScroll1.Value
    v1 = VScroll1.Value
    If (LeftDisplay = "Parts Tree") Then
        For c = 0 To Instances(ThisInst).NumParts - 1
            MNode1(c).Move MNode1(c).Left, MNode1(c).Top + dy, MNode1(c).Width, MNode1(c).Height
            Instances(ThisInst).PartsTree(c).Ypos = Instances(ThisInst).PartsTree(c).Ypos + dy
            Line1(c).Y1 = Line1(c).Y1 + dy
            Line1(c).Y2 = Line1(c).Y2 + dy
```

```
    Next c
End If
If (LeftDisplay = "Part Oriented Function/Means Tree") Then
    For c = 0 To Instances(ThisInst).NumParts - 1
        MNode1(c).Move MNode1(c).Left, MNode1(c).Top + dy, MNode1(c).Width, MNode1(c).Height
    Next c
    For c = 1 To NumLine5
        Line5(c).Y1 = Line5(c).Y1 + dy
        Line5(c).Y2 = Line5(c).Y2 + dy
    Next c
    For c = 1 To NumFunc2
        FNode2(c).Move FNode2(c).Left, FNode2(c).Top + dy, FNode2(c).Width, FNode2(c).Height
    Next c
  End If
End Sub
Private Sub HScroll1_Change()
    Dim X As Long
    Dim dx As Integer
    Dim c As Integer
    dx = h1 - HScroll1.Value
    h1 = HScroll1.Value
If (LeftDisplay = "Parts Tree") Then
    For c = 0 To Instances(ThisInst).NumParts - 1
        MNode1(c).Move MNode1(c).Left + dx, MNode1(c).Top, MNode1(c).Width, MNode1(c).Height
        Instances(ThisInst).PartsTree(c).Xpos = Instances(ThisInst).PartsTree(c).Xpos + dx
        Line1(c).X1 = Line1(c).X1 + dx
        Line1(c).X2 = Line1(c).X2 + dx
    Next c
End If
If (LeftDisplay = "Part Oriented Function/Means Tree") Then
    For c = 0 To Instances(ThisInst).NumParts - 1
        MNode1(c).Move MNode1(c).Left + dx, MNode1(c).Top, MNode1(c).Width, MNode1(c).Height
Next c
For c = 1 To NumLine5
    Line5(c).X1 = Line5(c).X1 + dx
    Line5(c).X2 = Line5(c).X2 + dx
Next c
For c = 1 To NumFunc2
    FNode2(c).Move FNode2(c).Left + dx, FNode2(c).Top, FNode2(c).Width, FNode2(c).Height
Next c
```

```
    End If
End Sub

Private Sub VScroll2_Change()
    Dim dy As Integer
    Dim c As Integer
    dy = v2 - VScroll2.Value
    v2 = VScroll2.Value
    For c = 0 To Instances(ThisInst).NumFuncts - 1
        FNode1(c).Move FNode1(c).Left, FNode1(c).Top + dy, FNode1(c).Width, FNode1(c).Height
        Instances(ThisInst).FunctTree(c).Ypos = Instances(ThisInst).FunctTree(c).Ypos + dy
        Line2(c).Y1 = Line2(c).Y1 + dy
        Line2(c).Y2 = Line2(c).Y2 + dy
    Next c
End Sub
Private Sub HScroll2_Change()
    Dim dx As Integer
    Dim c As Integer
    dx = h2 - HScroll2.Value
    h2 = HScroll2.Value
    For c = 0 To Instances(ThisInst).NumFuncts - 1
        FNode1(c).Move FNode1(c).Left + dx, FNode1(c).Top, FNode1(c).Width, FNode1(c).Height
        Instances(ThisInst).FunctTree(c).Xpos = Instances(ThisInst).FunctTree(c).Xpos + dx
        Line2(c).X1 = Line2(c).X1 + dx
        Line2(c).X2 = Line2(c).X2 + dx
    Next c
End Sub

' INSERT NEW MEANS – FUNCTION RELATION
Private Sub AddNewRelation()
    Dim xA, xC As Integer
    Dim yA, yB, yC As Integer
    Dim w, h As Integer
    Dim M, f As Integer
    Dim R As Integer
    Dim c As Integer

    w = MNode1(0).Width / 2
    h = MNode1(0).Height / 2
```

```
If ((RelMeans >= 0) And (RelFunct >= 0)) Then
   MNode1(RelMeans).BackColor = vbBlue
   FNode1(RelFunct).Colour = "Blue"
   FNode1(RelFunct).Refresh
   xA = Instances(ThisInst).PartsTree(RelMeans).Xpos + w
   yA = Instances(ThisInst).PartsTree(RelMeans).Ypos + h
   xC = Instances(ThisInst).FunctTree(RelFunct).Xpos + w
   yC = Instances(ThisInst).FunctTree(RelFunct).Ypos + h
   yB = yA + (yC - yA) / 2
   Line3.X1 = xA
   Line3.Y1 = yA
   Line3.X2 = xA + 6000
   Line3.Y2 = yB
   Line4.X1 = xC
   Line4.Y1 = yC
   Line4.X2 = xC - 6000
   Line4.Y2 = yB
   Line3.Visible = True
   Line4.Visible = True
f = Instances(ThisInst).PartsTree(RelMeans).NumFunctions
M = Instances(ThisInst).FunctTree(RelFunct).NumMeans

'check to see if relation already exists
If (f > 0) Then
   For c = 0 To f - 1
      If (Instances(ThisInst).PartsTree(RelMeans).Functions(c) = RelFunct) Then
         MsgBox ("This Relationship Already Exists!")
         DoWhat = "NOTHING"
         RelMeans = -1
         RelFunct = -1
         Exit Sub
      End If
   Next c
End If

Instances(ThisInst).NumRels = Instances(ThisInst).NumRels + 1
Instances(ThisInst).PartsTree(RelMeans).NumFunctions = f + 1
Instances(ThisInst).FunctTree(RelFunct).NumMeans = M + 1
ReDim Preserve Instances(ThisInst).PartsTree(RelMeans).Functions(f)
```

```
ReDim Preserve Instances(ThisInst).FunctTree(RelFunct).Means(M)
Instances(ThisInst).PartsTree(RelMeans).Functions(f) = RelFunct
Instances(ThisInst).FunctTree(RelFunct).Means(M) = RelMeans
ReDim Preserve Instances(ThisInst).RelM(Instances(ThisInst).NumRels - 1)
ReDim Preserve Instances(ThisInst).RelF(Instances(ThisInst).NumRels - 1)
Instances(ThisInst).RelM(Instances(ThisInst).NumRels - 1) = RelMeans
Instances(ThisInst).RelF(Instances(ThisInst).NumRels - 1) = RelFunct

RelsGrid.Rows = RelsGrid.Rows + 1
    RelsGrid.Row = Instances(ThisInst).NumRels
    RelsGrid.Col = 0
    RelsGrid.Text = Instances(ThisInst).PartsTree(RelMeans).Name
    RelsGrid.Col = 1
    RelsGrid.Text = Instances(ThisInst).FunctTree(RelFunct).Name

    DoWhat = "NOTHING"
    RelMeans = -1
    RelFunct = -1
  End If

End Sub

' UPDATE DISPLAY
Private Sub RedrawLeftDisplay()
Dim c, d, e, f, g As Integer
Dim Vert As Integer
Dim maxLevel As Integer
Dim ThisWidth, MaxWidth As Long
Dim ThisNumFuncts As Integer
Dim WidestLevel As Integer
Dim OldTop, ThisTop, Middle, Left As Integer
Dim SortedMeans() As Integer
Dim SectWidth As Integer
Dim NumFunctAtLevel As Integer
Dim NumMeansAtLevel As Integer
Dim thisLevel As Integer
Dim myParent As Integer

If (LeftDisplay = "Part Oriented Function/Means Tree") Then
```

```
' Hide all Line1's
For c = 0 To Instances(ThisInst).NumParts - 1
   Line1(c).Visible = False
Next c
Line3.Visible = False
Line4.Visible = False
NumFunc2 = 0
NumLine5 = 0

'Get highest level no
maxLevel = 0
For c = 0 To Instances(ThisInst).NumParts - 1
   If (Instances(ThisInst).PartsTree(c).Level > maxLevel) Then
      maxLevel = Instances(ThisInst).PartsTree(c).Level
   End If
Next c

For thisLevel = 0 To maxLevel

   'Get NumFunctAtLevel and NumMeansAtLevel
   NumFunctAtLevel = 0
   NumMeansAtLevel = 0

   'Find width of level's functions
   MaxWidth = 0
   ThisNumFuncts = 0
   For c = 0 To Instances(ThisInst).NumParts - 1
      If (Instances(ThisInst).PartsTree(c).Level = thisLevel) Then
         ThisNumFuncts = ThisNumFuncts + Instances(ThisInst).PartsTree(c).NumFunctions
      End If
   Next c
   MaxWidth = ThisNumFuncts * FNode1(0).Width + (ThisNumFuncts - 1) * HGap

   OldTop = MNode1(0).Top
   Middle = MNode1(0).Left + MNode1(0).Width / 2

   For c = 0 To Instances(ThisInst).NumParts - 1
      If (Instances(ThisInst).PartsTree(c).Level = thisLevel) Then
         NumMeansAtLevel = NumMeansAtLevel + 1
         NumFunctAtLevel = NumFunctAtLevel + Instances(ThisInst).PartsTree(c).NumFunctions
```

```
    End If
Next c
'Make List of Means/Parts at ThisLevel
ReDim SortedMeans(NumMeansAtLevel)
d = 0
For c = 0 To Instances(ThisInst).NumParts - 1
    If (Instances(ThisInst).PartsTree(c).Level = thisLevel) Then
        SortedMeans(d) = c
        d = d + 1
    End If
Next c
'Sort List
For c = 0 To 1000
    For d = 0 To NumMeansAtLevel - 2
        e = Instances(ThisInst).PartsTree(SortedMeans(d)).Xpos
        f = Instances(ThisInst).PartsTree(SortedMeans(d + 1)).Xpos
        If (e > f) Then
            g = SortedMeans(d + 1)
            SortedMeans(d + 1) = SortedMeans(d)
            SortedMeans(d) = g
        End If
    Next d
Next c

d = NumMeansAtLevel
'display means for ThisLevel
ThisTop = FNode1(0).Height * (thisLevel + 1) + MNode1(0).Height * thisLevel + VGap * (thisLevel * 2 +
1) + FNode1(0).Height - VGap
Left = Middle - MaxWidth / 2
For c = 0 To d - 1
    If (Instances(ThisInst).PartsTree(SortedMeans(c)).NumFunctions > 0) Then
        SectWidth = Instances(ThisInst).PartsTree(SortedMeans(c)).NumFunctions * FNode1(0).Width +
(Instances(ThisInst).PartsTree(SortedMeans(c)).NumFunctions - 1) * HGap
        MNode1(SortedMeans(c)).Left = Left + SectWidth / 2 - MNode1(0).Width / 2
    Else
        SectWidth = 0
        MNode1(SortedMeans(c)).Left = Left
    End If
    MNode1(SortedMeans(c)).Top = ThisTop
    Left = Left + SectWidth + HGap
```

```
Next c
'display functions for ThisLevel
Left = Middle - MaxWidth / 2
ThisTop = ThisTop - VGap - FNode1(0).Height
For c = 0 To d - 1
   If (Instances(ThisInst).PartsTree(SortedMeans(c)).NumFunctions > 0) Then
      For e = 0 To Instances(ThisInst).PartsTree(SortedMeans(c)).NumFunctions - 1
         NumFunc2 = NumFunc2 + 1
         NumLine5 = NumLine5 + 1
            Load FNode2(NumFunc2)
            FNode2(NumFunc2).Visible = True
            FNode2(NumFunc2).Top = ThisTop
            FNode2(NumFunc2).Left = Left
            FNode2(NumFunc2).Text =
FNode1(Instances(ThisInst).PartsTree(SortedMeans(c)).Functions(e)).Text
            Left = Left + FNode1(0).Width + HGap
            Load Line5(NumLine5)
            Line5(NumLine5).Visible = True
            Line5(NumLine5).X1 = FNode2(NumFunc2).Left + FNode2(NumFunc2).Width / 2
            Line5(NumLine5).Y1 = FNode2(NumFunc2).Top + FNode2(NumFunc2).Height
            Line5(NumLine5).X2 = MNode1(Instances(ThisInst).PartsTree(SortedMeans(c)).NodeID).Left +
MNode1(Instances(ThisInst).PartsTree(SortedMeans(c)).NodeID).Width / 2
            Line5(NumLine5).Y2 = MNode1(Instances(ThisInst).PartsTree(SortedMeans(c)).NodeID).Top
            If (SortedMeans(c) > 0) Then
               NumLine5 = NumLine5 + 1
               Load Line5(NumLine5)
               Line5(NumLine5).Visible = True
               Line5(NumLine5).X1 = Line5(NumLine5 - 1).X1
               Line5(NumLine5).Y1 = Line5(NumLine5 - 1).Y1 - FNode1(0).Height
               Line5(NumLine5).X2 =
MNode1(Instances(ThisInst).PartsTree(Instances(ThisInst).PartsTree(SortedMeans(c)).Parents(0)).NodeID).Left
+ MNode1(0).Width / 2
               Line5(NumLine5).Y2 = Line5(NumLine5 - 1).Y1 - MNode1(0).Height - VGap
            End If

      Next e
   Else
      NumLine5 = NumLine5 + 1
      Load Line5(NumLine5)
      Line5(NumLine5).Visible = True
```

```
            Line5(NumLine5).X1 = MNode1(SortedMeans(c)).Left + MNode1(0).Width / 2
            Line5(NumLine5).Y1 = MNode1(SortedMeans(c)).Top
            Line5(NumLine5).X2 =
MNode1(Instances(ThisInst).PartsTree(Instances(ThisInst).PartsTree(SortedMeans(c)).Parents(0)).NodeID).Left
+ MNode1(0).Width / 2
            Line5(NumLine5).Y2 =
MNode1(Instances(ThisInst).PartsTree(Instances(ThisInst).PartsTree(SortedMeans(c)).Parents(0)).NodeID).Top
+ MNode1(0).Height
        End If
      Next c

   Next thisLevel
End If

If (LeftDisplay = "Parts Tree") Then
   'First cleanup from previous
   Beep
   For c = 1 To NumFunc2
      Unload FNode2(c)
   Next c
   NumFunc2 = 0
   For c = 1 To NumLine5
      Unload Line5(c)
   Next c
   NumLine5 = 0
   For c = 0 To Instances(ThisInst).NumParts - 1
      Line1(c).Visible = True
      UpdateMeansNode (c)
   Next c
End If

End Sub
End Sub
Public Sub PrintThisTree()
   VScroll1.Visible = False
   HScroll1.Visible = False
   PrintForm
   VScroll1.Visible = True
   HScroll1.Visible = True
End Sub
```

```
' OPEN A NEW FMT FILE
Public Sub OpenThisTree()
    Dim sFile As String
    Dim c, d As Integer
    Dim sPicFile, sCADFile As String
    Dim TrueFalse, Title As String
    Dim t As Integer
    Dim tmp As String

With CommonDialog1
   .Filter = "All Files (*.fmt) | *.fmt"
   .ShowOpen
   If Len(.filename) = 0 Then
      Exit Sub
   End If
   sFile = .filename
End With
Caption = sFile
Open sFile For Input As #1    ' Open file for Input.

'INPUT DOCUMENT TYPE - GENERIC / INSTANCE
Input #1, DocType
If (DocType = "GENERIC") Then
   Input #1, NumInstances
   ReDim Instances(NumInstances)
   For i = 0 To NumInstances - 1
      Input #1, Instances(i).By
      Input #1, Instances(i).Date
      Input #1, Instances(i).Description
      Input #1, Instances(i).DrgNo
      Input #1, Instances(i).Name
      'input #1, Instances(i).FileName
      'input #1, Instances(i).Path

Me.Caption = Instances(0).Name
'INPUT PARTS TREE
Input #1, tmp
Input #1, Instances(i).NumParts
ReDim Instances(i).PartsTree(Instances(i).NumParts)
For c = 0 To Instances(i).NumParts - 1
```

```
Input #1, tmp
Input #1, Instances(i).PartsTree(c).NodeID
Input #1, Instances(i).PartsTree(c).Name
Input #1, Instances(i).PartsTree(c).CADfiletype
Input #1, Instances(i).PartsTree(c).Xpos
Input #1, Instances(i).PartsTree(c).Ypos
Input #1, Instances(i).PartsTree(c).NumParents
ReDim Instances(i).PartsTree(c).Parents(Instances(i).PartsTree(c).NumParents)
For d = 0 To Instances(i).PartsTree(c).NumParents - 1
   Input #1, Instances(i).PartsTree(c).Parents(d)
Next d
Input #1, Instances(i).PartsTree(c).NumChildren
ReDim Instances(i).PartsTree(c).Children(Instances(i).PartsTree(c).NumChildren)
For d = 0 To Instances(i).PartsTree(c).NumChildren - 1
   Input #1, Instances(i).PartsTree(c).Children(d)
Next d
Input #1, Instances(i).PartsTree(c).NumFunctions
ReDim Instances(i).PartsTree(c).Functions(Instances(i).PartsTree(c).NumFunctions)
For d = 0 To Instances(i).PartsTree(c).NumFunctions - 1
   Input #1, Instances(i).PartsTree(c).Functions(d)
Next d
Input #1, Instances(i).PartsTree(c).NumOfParams
ReDim Instances(i).PartsTree(c).ParamName(Instances(i).PartsTree(c).NumOfParams)
ReDim Instances(i).PartsTree(c).ParamValue(Instances(i).PartsTree(c).NumOfParams)
ReDim Instances(i).PartsTree(c).ParamUnit(Instances(i).PartsTree(c).NumOfParams)
For d = 0 To Instances(i).PartsTree(c).NumOfParams - 1
   Input #1, Instances(i).PartsTree(c).ParamName(d)
   Input #1, Instances(i).PartsTree(c).ParamValue(d)
   Input #1, Instances(i).PartsTree(c).ParamUnit(d)
Next d
Input #1, Instances(i).PartsTree(c).NumOfSupps
ReDim Instances(i).PartsTree(c).SuppEntity(Instances(i).PartsTree(c).NumOfSupps)
ReDim Instances(i).PartsTree(c).SuppType(Instances(i).PartsTree(c).NumOfSupps)
ReDim Instances(i).PartsTree(c).SuppStatus(Instances(i).PartsTree(c).NumOfSupps)
For d = 0 To Instances(i).PartsTree(c).NumOfSupps - 1
   Input #1, Instances(i).PartsTree(c).SuppEntity(d)
   Input #1, Instances(i).PartsTree(c).SuppType(d)
   Input #1, Instances(i).PartsTree(c).SuppStatus(d)
Next d
Input #1, Instances(i).PartsTree(c).MyPathAndFile
```

```
Input #1, Instances(i).PartsTree(c).MyFileName
Input #1, Instances(i).PartsTree(c).MyCADfileType
Input #1, Instances(i).PartsTree(c).Level
Input #1, Instances(i).PartsTree(c).PartSuppression
Input #1, Instances(i).PartsTree(c).myDrawing
'Instances(i).PartsTree(c).Name = Instances(i).PartsTree(c).PartSuppression

'If (c > 0) Then
'   Load MNode1(c)
   'End If
   'UpdateMeansNode (c)
Next c

'INPUT FUNCTION FAMILY TREE
Input #1, tmp
Input #1, Instances(i).NumFuncts
ReDim Instances(i).FunctTree(Instances(i).NumFuncts)
For c = 0 To Instances(i).NumFuncts - 1
   Input #1, tmp
   Input #1, Instances(i).FunctTree(c).NodeID
   Input #1, Instances(i).FunctTree(c).Name
   Input #1, Instances(i).FunctTree(c).Xpos
   Input #1, Instances(i).FunctTree(c).Ypos
   Input #1, Instances(i).FunctTree(c).NumParents
   ReDim Instances(i).FunctTree(c).Parents(Instances(i).FunctTree(c).NumParents)
   For d = 0 To Instances(i).FunctTree(c).NumParents - 1
      Input #1, Instances(i).FunctTree(c).Parents(d)
   Next d
   Input #1, Instances(i).FunctTree(c).NumChildren
   ReDim Instances(i).FunctTree(c).Children(Instances(i).FunctTree(c).NumChildren)
   For d = 0 To Instances(i).FunctTree(c).NumChildren - 1
      Input #1, Instances(i).FunctTree(c).Children(d)
   Next d
   Input #1, Instances(i).FunctTree(c).NumMeans
   ReDim Instances(i).FunctTree(c).Means(Instances(i).FunctTree(c).NumMeans)
   For d = 0 To Instances(i).FunctTree(c).NumMeans - 1
      Input #1, Instances(i).FunctTree(c).Means(d)
   Next d
   Input #1, Instances(i).FunctTree(c).Level
   'If (c > 0) Then
```

```
    '   Load FNode1 (c)
    'End If
    'UpdateFunctNode  (c)
Next c

'INPUT RELATIONS
Input #1, tmp
Input #1, Instances (i).NumRels
ReDim Instances (i).RelM (Instances (i).NumRels)
ReDim Instances (i).RelF (Instances (i).NumRels)
RelsGrid.Rows = Instances (i).NumRels + 1
For c = 0 To Instances (i).NumRels - 1
    Input #1, Instances (i).RelM (c)
    Input #1, Instances (i).RelF (c)
    RelsGrid.Row = c + 1
    RelsGrid.Col = 0
    RelsGrid.Text = Instances (i).PartsTree (Instances (i).RelM (c)).Name
    RelsGrid.Col = 1
    RelsGrid.Text = Instances (i).FunctTree (Instances (i).RelF (c)).Name
Next c
'Parts Tree Link-Lines
Input #1, tmp
ReDim Instances (i).L1x1 (Instances (i).NumParts)
ReDim Instances (i).L1y1 (Instances (i).NumParts)
ReDim Instances (i).L1x2 (Instances (i).NumParts)
ReDim Instances (i).L1y2 (Instances (i).NumParts)
For c = 1 To Instances (i).NumParts - 1
    'Load Line1 (c)
    'Line1 (c).Visible = True
    Input #1, Instances (i).L1x1 (c)
        'Line1 (c).X1 = Instances (i).L1x1 (c)
    Input #1, Instances (i).L1y1 (c)
        'Line1 (c).Y1 = Instances (i).L1y1 (c)
    Input #1, Instances (i).L1x2 (c)
        'Line1 (c).X2 = Instances (i).L1x2 (c)
    Input #1, Instances (i).L1y2 (c)
        'Line1 (c).Y2 = Instances (i).L1y2 (c)
Next c
'Function Tree Link-Lines
Input #1, tmp
```

```
ReDim Instances(i).L2x1(Instances(i).NumFuncts)
ReDim Instances(i).L2y1(Instances(i).NumFuncts)
ReDim Instances(i).L2x2(Instances(i).NumFuncts)
ReDim Instances(i).L2y2(Instances(i).NumFuncts)
For c = 1 To Instances(i).NumFuncts - 1
   'Load Line2(c)
   'Line2(c).Visible = True
   Input #1, Instances(i).L2x1(c)
      'Line2(c).X1 = Instances(i).L2x1(c)
      Input #1, Instances(i).L2y1(c)
         'Line2(c).Y1 = Instances(i).L2y1(c)
      Input #1, Instances(i).L2x2(c)
         'Line2(c).X2 = Instances(i).L2x2(c)
      Input #1, Instances(i).L2y2(c)
         'Line2(c).Y2 = Instances(i).L2y2(c)
   Next c
'INPUT SUPPRESSED PARTS
   'Input #1, Instances(i).NumSuppParts
   'If (Instances(i).NumSuppParts > 0) Then
   '   ReDim Instances(i).SuppParts(Instances(i).NumSuppParts)
   '   For c = 0 To Instances(i).NumSuppParts - 1
   '      Input #1, Instances(i).SuppParts(c)
   '   Next c
   'End If
  Next i
'ElseIf (DocType = "INSTANCE") Then
'  input #1, Instances(0).By
'  input #1, Instances(0).Date
'  input #1, Instances(0).Description
'  input #1, Instances(0).DrgNo
'  input #1, Instances(0).Name
'  'input #1, Instances(0).FileName
'  'input #1, Instances(0).Path
  End If
Close #1   ' Close file.
  RegenerateInstances (0)
  UpdateInstance (0)
  ShowCurrentInstance -1, 0
  Form_Resize
End Sub
```

```
' SAVE CURRENT FMT FILE
Public Sub SaveThisTree()

    Dim sFile As String
    Dim c, d, i As Integer
    Dim sPicFile As String
    Dim mX, mY As Integer

    With CommonDialog1
      'To Do
      'set the flags and attributes of the
      'common dialog control
      .Filter = "All Files (*.fmt) | *.fmt"
      .ShowSave
      If Len(.filename) = 0 Then
         Exit Sub
      End If
      sFile = .filename
  End With
  Caption = sFile
  Open sFile For Output As #1    ' Open file for output.
  sPicFile = Mid(sFile, 1, Len(sFile) - 4)

  UpdateLinesArrays (ThisInst)


  'OUTPUT DOCUMENT TYPE - GENERIC / INSTANCE
  Print #1, DocType
  If (DocType = "GENERIC") Then
     Print #1, NumInstances
     For i = 0 To NumInstances - 1
        Print #1, Instances(i).By
        Print #1, Instances(i).Date
        Print #1, Instances(i).Description
        Print #1, Instances(i).DrgNo
        Print #1, Instances(i).Name
        'Print #1, Instances(i).FileName
        'Print #1, Instances(i).Path

  'OUTPUT PARTS TREE
```

```
Print #1, "PARTS_TREE"
Print #1, Instances(i).NumParts
For c = 0 To Instances(i).NumParts - 1
    Print #1, "NODE_" + Str(c)
    Print #1, Instances(i).PartsTree(c).NodeID
    Print #1, Instances(i).PartsTree(c).Name
    Print #1, Instances(i).PartsTree(c).CADfiletype
    Print #1, Instances(i).PartsTree(c).Xpos
    Print #1, Instances(i).PartsTree(c).Ypos
Print #1, Instances(i).PartsTree(c).NumParents
For d = 0 To Instances(i).PartsTree(c).NumParents - 1
    Print #1, Instances(i).PartsTree(c).Parents(d)
Next d
Print #1, Instances(i).PartsTree(c).NumChildren
For d = 0 To Instances(i).PartsTree(c).NumChildren - 1
    Print #1, Instances(i).PartsTree(c).Children(d)
Next d
Print #1, Instances(i).PartsTree(c).NumFunctions
For d = 0 To Instances(i).PartsTree(c).NumFunctions - 1
    Print #1, Instances(i).PartsTree(c).Functions(d)
Next d
Print #1, Instances(i).PartsTree(c).NumOfParams
For d = 0 To Instances(i).PartsTree(c).NumOfParams - 1
    Print #1, Instances(i).PartsTree(c).ParamName(d)
    Print #1, Instances(i).PartsTree(c).ParamValue(d)
    Print #1, Instances(i).PartsTree(c).ParamUnit(d)
Next d
Print #1, Instances(i).PartsTree(c).NumOfSupps
For d = 0 To Instances(i).PartsTree(c).NumOfSupps - 1
    Print #1, Instances(i).PartsTree(c).SuppEntity(d)
    Print #1, Instances(i).PartsTree(c).SuppType(d)
    Print #1, Instances(i).PartsTree(c).SuppStatus(d)
Next d
Print #1, Instances(i).PartsTree(c).MyPathAndFile
Print #1, Instances(i).PartsTree(c).MyFileName
Print #1, Instances(i).PartsTree(c).MyCADfileType
Print #1, Instances(i).PartsTree(c).Level
Print #1, Instances(i).PartsTree(c).PartSuppression
Print #1, Instances(i).PartsTree(c).myDrawing
Next c
```

```
'OUTPUT FUNCTION FAMILY TREE
Print #1, "FUNCTION_TREE"
Print #1, Instances(i).NumFuncts
For c = 0 To Instances(i).NumFuncts - 1
   Print #1, "NODE_" + Str(c)
   Print #1, Instances(i).FunctTree(c).NodeID
   Print #1, Instances(i).FunctTree(c).Name
   Print #1, Instances(i).FunctTree(c).Xpos
Print #1, Instances(i).FunctTree(c).Ypos
Print #1, Instances(i).FunctTree(c).NumParents
For d = 0 To Instances(i).FunctTree(c).NumParents - 1
   Print #1, Instances(i).FunctTree(c).Parents(d)
Next d
Print #1, Instances(i).FunctTree(c).NumChildren
For d = 0 To Instances(i).FunctTree(c).NumChildren - 1
   Print #1, Instances(i).FunctTree(c).Children(d)
Next d
Print #1, Instances(i).FunctTree(c).NumMeans
For d = 0 To Instances(i).FunctTree(c).NumMeans - 1
   Print #1, Instances(i).FunctTree(c).Means(d)
Next d
   Print #1, Instances(i).FunctTree(c).Level
Next c

'OUTPUT RELATIONS
Print #1, "RELATIONS"
Print #1, Instances(i).NumRels
For c = 0 To Instances(i).NumRels - 1
   Print #1, Instances(i).RelM(c)
   Print #1, Instances(i).RelF(c)
Next c

'Parts Tree Link-Lines
Print #1, "LINE1"
For c = 1 To Instances(i).NumParts - 1
   Print #1, Instances(i).L1x1(c)
   Print #1, Instances(i).L1y1(c)
   Print #1, Instances(i).L1x2(c)
   Print #1, Instances(i).L1y2(c)
```

```
Next c
'Function Tree Link-Lines
Print #1, "LINE2"
For c = 1 To Instances(i).NumFuncts - 1
   Print #1, Instances(i).L2x1(c)
   Print #1, Instances(i).L2y1(c)
   Print #1, Instances(i).L2x2(c)
   Print #1, Instances(i).L2y2(c)
Next c

    'OUTPUT SUPPRESSED PARTS
    'Print #1, Instances(i).NumSuppParts
    'If (Instances(i).NumSuppParts > 0) Then
    '   For c = 0 To Instances(i).NumSuppParts - 1
    '      Print #1, Instances(i).SuppParts(c)
    '   Next c
    'End If
  Next i
'ElseIf (DocType = "INSTANCE") Then
'   Print #1, Instances(0).By
   '   Print #1, Instances(0).Date
   '   Print #1, Instances(0).Description
   '   Print #1, Instances(0).DrgNo
   '   Print #1, Instances(0).Name
   '   'Print #1, Instances(0).FileName
   '   'Print #1, Instances(0).Path
  End If

  Close #1    ' Close file.

  Me.Caption = sPicFile
End Sub
```

```
' ADD A NEW INSTANCE
Public Sub AddNewInstance()
    Dim c, d As Integer
    Dim CopyOf As Integer

    CopyOf = InstGrid.Row
    NumInstances = NumInstances + 1
```

```
ReDim Preserve Instances(NumInstances - 1)
ThisInst = NumInstances - 1

' Copy contents of instance CopyOf to ThisInst

Instances(ThisInst).By = Instances(CopyOf).By
Instances(ThisInst).Date = Instances(CopyOf).Date
Instances(ThisInst).Description = Instances(CopyOf).Description
Instances(ThisInst).DocType = "INSTANCE"
Instances(ThisInst).DrgNo = Str(ThisInst)
Instances(ThisInst).Name = "COPY OF (" + Str(CopyOf) + ") " + Instances(CopyOf).Name
Me.Caption = Instances(0).Name
Instances(ThisInst).NumFuncts = Instances(CopyOf).NumFuncts
Instances(ThisInst).NumParts = Instances(CopyOf).NumParts
Instances(ThisInst).NumRels = Instances(CopyOf).NumRels
ReDim Instances(ThisInst).PartsTree(Instances(ThisInst).NumParts)
For c = 0 To Instances(ThisInst).NumParts - 1
   Instances(ThisInst).PartsTree(c).NodeID = Instances(CopyOf).PartsTree(c).NodeID
   Instances(ThisInst).PartsTree(c).Name = Instances(CopyOf).PartsTree(c).Name
   Instances(ThisInst).PartsTree(c).CADfiletype = Instances(CopyOf).PartsTree(c).CADfiletype
Instances(ThisInst).PartsTree(c).Xpos = Instances(CopyOf).PartsTree(c).Xpos
Instances(ThisInst).PartsTree(c).Ypos = Instances(CopyOf).PartsTree(c).Ypos
Instances(ThisInst).PartsTree(c).NumParents = Instances(CopyOf).PartsTree(c).NumParents
ReDim Instances(ThisInst).PartsTree(c).Parents(Instances(ThisInst).PartsTree(c).NumParents)
For d = 0 To Instances(ThisInst).PartsTree(c).NumParents - 1
   Instances(ThisInst).PartsTree(c).Parents(d) = Instances(CopyOf).PartsTree(c).Parents(d)
Next d
Instances(ThisInst).PartsTree(c).NumChildren = Instances(CopyOf).PartsTree(c).NumChildren
ReDim Instances(ThisInst).PartsTree(c).Children(Instances(ThisInst).PartsTree(c).NumChildren)
For d = 0 To Instances(ThisInst).PartsTree(c).NumChildren - 1
   Instances(ThisInst).PartsTree(c).Children(d) = Instances(CopyOf).PartsTree(c).Children(d)
Next d
Instances(ThisInst).PartsTree(c).NumFunctions = Instances(CopyOf).PartsTree(c).NumFunctions
ReDim Instances(ThisInst).PartsTree(c).Functions(Instances(ThisInst).PartsTree(c).NumFunctions)
For d = 0 To Instances(ThisInst).PartsTree(c).NumFunctions - 1
   Instances(ThisInst).PartsTree(c).Functions(d) = Instances(CopyOf).PartsTree(c).Functions(d)
Next d
Instances(ThisInst).PartsTree(c).NumOfParams = Instances(CopyOf).PartsTree(c).NumOfParams
ReDim Instances(ThisInst).PartsTree(c).ParamName(Instances(ThisInst).PartsTree(c).NumOfParams)
ReDim Instances(ThisInst).PartsTree(c).ParamValue(Instances(ThisInst).PartsTree(c).NumOfParams)
```

```
ReDim Instances(ThisInst).PartsTree(c).ParamUnit(Instances(ThisInst).PartsTree(c).NumOfParams)
For d = 0 To Instances(ThisInst).PartsTree(c).NumOfParams - 1
   Instances(ThisInst).PartsTree(c).ParamName(d) = Instances(CopyOf).PartsTree(c).ParamName(d)
   Instances(ThisInst).PartsTree(c).ParamValue(d) = Instances(CopyOf).PartsTree(c).ParamValue(d)
   Instances(ThisInst).PartsTree(c).ParamUnit(d) = Instances(CopyOf).PartsTree(c).ParamUnit(d)
Next d
Instances(ThisInst).PartsTree(c).NumOfSupps = Instances(CopyOf).PartsTree(c).NumOfSupps
ReDim Instances(ThisInst).PartsTree(c).SuppEntity(Instances(ThisInst).PartsTree(c).NumOfSupps)
ReDim Instances(ThisInst).PartsTree(c).SuppType(Instances(ThisInst).PartsTree(c).NumOfSupps)
ReDim Instances(ThisInst).PartsTree(c).SuppStatus(Instances(ThisInst).PartsTree(c).NumOfSupps)
For d = 0 To Instances(ThisInst).PartsTree(c).NumOfSupps - 1
   Instances(ThisInst).PartsTree(c).SuppEntity(d) = Instances(CopyOf).PartsTree(c).SuppEntity(d)
   Instances(ThisInst).PartsTree(c).SuppType(d) = Instances(CopyOf).PartsTree(c).SuppType(d)
   Instances(ThisInst).PartsTree(c).SuppStatus(d) = Instances(CopyOf).PartsTree(c).SuppStatus(d)
Next d
Instances(ThisInst).PartsTree(c).MyPathAndFile = Instances(CopyOf).PartsTree(c).MyPathAndFile
Instances(ThisInst).PartsTree(c).MyFileName = Instances(CopyOf).PartsTree(c).MyFileName
Instances(ThisInst).PartsTree(c).MyCADfileType = Instances(CopyOf).PartsTree(c).MyCADfileType
Instances(ThisInst).PartsTree(c).Level = Instances(CopyOf).PartsTree(c).Level
Instances(ThisInst).PartsTree(c).PartSuppression = Instances(CopyOf).PartsTree(c).PartSuppression
   Instances(ThisInst).PartsTree(c).myDrawing = Instances(CopyOf).PartsTree(c).myDrawing
Next c
ReDim Instances(ThisInst).FunctTree(Instances(ThisInst).NumFuncts)
For c = 0 To Instances(ThisInst).NumFuncts - 1
   Instances(ThisInst).FunctTree(c).NodeID = Instances(CopyOf).FunctTree(c).NodeID
   Instances(ThisInst).FunctTree(c).Name = Instances(CopyOf).FunctTree(c).Name
   Instances(ThisInst).FunctTree(c).Xpos = Instances(CopyOf).FunctTree(c).Xpos
   Instances(ThisInst).FunctTree(c).Ypos = Instances(CopyOf).FunctTree(c).Ypos
   Instances(ThisInst).FunctTree(c).NumParents = Instances(CopyOf).FunctTree(c).NumParents
   ReDim Instances(ThisInst).FunctTree(c).Parents(Instances(ThisInst).FunctTree(c).NumParents)
   For d = 0 To Instances(ThisInst).FunctTree(c).NumParents - 1
      Instances(ThisInst).FunctTree(c).Parents(d) = Instances(CopyOf).FunctTree(c).Parents(d)
   Next d
   Instances(ThisInst).FunctTree(c).NumChildren = Instances(CopyOf).FunctTree(c).NumChildren
   ReDim Instances(ThisInst).FunctTree(c).Children(Instances(ThisInst).FunctTree(c).NumChildren)
   For d = 0 To Instances(ThisInst).FunctTree(c).NumChildren - 1
      Instances(ThisInst).FunctTree(c).Children(d) = Instances(CopyOf).FunctTree(c).Children(d)
   Next d
   Instances(ThisInst).FunctTree(c).NumMeans = Instances(CopyOf).FunctTree(c).NumMeans
   ReDim Instances(ThisInst).FunctTree(c).Means(Instances(ThisInst).FunctTree(c).NumMeans)
```

```
    For d = 0 To Instances(ThisInst).FunctTree(c).NumMeans - 1
        Instances(ThisInst).FunctTree(c).Means(d) = Instances(CopyOf).FunctTree(c).Means(d)
    Next d
    Instances(ThisInst).FunctTree(c).Level = Instances(CopyOf).FunctTree(c).Level
Next c
ReDim Instances(ThisInst).RelM(Instances(ThisInst).NumRels)
ReDim Instances(ThisInst).RelF(Instances(ThisInst).NumRels)
For c = 0 To Instances(ThisInst).NumRels - 1
    Instances(ThisInst).RelM(c) = Instances(CopyOf).RelM(c)
    Instances(ThisInst).RelF(c) = Instances(CopyOf).RelF(c)
Next c
ReDim Instances(ThisInst).L1x1(Instances(ThisInst).NumParts)
ReDim Instances(ThisInst).L1y1(Instances(ThisInst).NumParts)
ReDim Instances(ThisInst).L1x2(Instances(ThisInst).NumParts)
ReDim Instances(ThisInst).L1y2(Instances(ThisInst).NumParts)
For c = 1 To Instances(ThisInst).NumParts - 1
    Instances(ThisInst).L1x1(c) = Line1(c).X1
    Instances(ThisInst).L1y1(c) = Line1(c).Y1
    Instances(ThisInst).L1x2(c) = Line1(c).X2
    Instances(ThisInst).L1y2(c) = Line1(c).Y2
Next c
ReDim Instances(ThisInst).L2x1(Instances(ThisInst).NumFuncts)
ReDim Instances(ThisInst).L2y1(Instances(ThisInst).NumFuncts)
ReDim Instances(ThisInst).L2x2(Instances(ThisInst).NumFuncts)
ReDim Instances(ThisInst).L2y2(Instances(ThisInst).NumFuncts)
For c = 1 To Instances(ThisInst).NumFuncts - 1
        Instances(ThisInst).L2x1(c) = Line2(c).X1
        Instances(ThisInst).L2y1(c) = Line2(c).Y1
        Instances(ThisInst).L2x2(c) = Line2(c).X2
        Instances(ThisInst).L2y2(c) = Line2(c).Y2
Next c
'Instances(ThisInst).NumSuppParts = Instances(CopyOf).NumSuppParts
  'If (Instances(ThisInst).NumSuppParts > 0) Then
  '   ReDim Instances(ThisInst).SuppParts(Instances(ThisInst).NumSuppParts)
  '   For c = 0 To Instances(ThisInst).NumSuppParts - 1
  '       Instances(ThisInst).SuppParts(c) = Instances(CopyOf).SuppParts(c)
  '   Next c
  'End If
'Instances(ThisInst) = Instances(CopyOf)
Instances(ThisInst).Name = "Instance " + Str(ThisInst)
```

```
    InstGrid.Rows = InstGrid.Rows + 1
    InstGrid.Row = InstGrid.Rows - 1
    InstGrid.Text = Instances(ThisInst).Name
    UpdateInstance (ThisInst)
    'ShowCurrentInstance ThisInst, ThisInst
End Sub
Private Sub DeleteSelectedInstance()

' UPDATE LINK LINES
End Sub
Private Sub UpdateLinesArrays(myInst As Integer)
    Dim c As Integer

    ReDim Instances(myInst).L1x1 (Instances(myInst).NumParts)
    ReDim Instances(myInst).L1y1 (Instances(myInst).NumParts)
    ReDim Instances(myInst).L1x2 (Instances(myInst).NumParts)
    ReDim Instances(myInst).L1y2 (Instances(myInst).NumParts)
    For c = 1 To Instances(myInst).NumParts - 1
        Instances(myInst).L1x1 (c) = Line1 (c).X1
        Instances(myInst).L1y1 (c) = Line1 (c).Y1
        Instances(myInst).L1x2 (c) = Line1 (c).X2
        Instances(myInst).L1y2 (c) = Line1 (c).Y2
Next c
ReDim Instances(myInst).L2x1 (Instances(myInst).NumFuncts)
ReDim Instances(myInst).L2y1 (Instances(myInst).NumFuncts)
ReDim Instances(myInst).L2x2 (Instances(myInst).NumFuncts)
ReDim Instances(myInst).L2y2 (Instances(myInst).NumFuncts)
For c = 1 To Instances(myInst).NumFuncts - 1
    Instances(myInst).L2x1 (c) = Line2 (c).X1
    Instances(myInst).L2y1 (c) = Line2 (c).Y1
    Instances(myInst).L2x2 (c) = Line2 (c).X2
    Instances(myInst).L2y2 (c) = Line2 (c).Y2
    Next c
End Sub
Public Sub ViewInstances(TrueOrFalse As Boolean)

    DispInstances = TrueOrFalse
    If (DispInstances = True) Then
        Frame2.Visible = True
    ElseIf (DispInstances = False) Then
```

```
        Frame2.Visible = False
    End If
    Form_Resize
End Sub
Private Sub UpdateInstance(myInst As Integer)
    'Instancing Layout
    Text4(0).Text = Instances(myInst).Name
    Text4(1).Text = Instances(myInst).DrgNo
    Text4(2).Text = Instances(myInst).By
    Text4(3).Text = Instances(myInst).Date
    Text4(4).Text = Instances(myInst).Description
    InstGrid.ColWidth(0) = InstGrid.Width
    'InstGrid.Col = 0
    'InstGrid.Row = myInst
    'InstGrid.Text = Instances(myInst).Name
End Sub
```

'DISPLAY THE SELECTED INSTANCE

```
Private Sub ShowCurrentInstance(OldInst As Integer, NewInst As Integer)
Dim c As Integer

If (OldInst = NewInst) Then
    Exit Sub
End If

'Unload Old Instance
If ((OldInst >= 0) And (OldInst <> NewInst)) Then
    UpdateLinesArrays (OldInst)
    For c = 1 To Instances(OldInst).NumParts - 1
        Unload MNode1(c)
        Unload Line1(c)
    Next c
For c = 1 To Instances(OldInst).NumFuncts - 1
    Unload FNode1(c)
    Unload Line2(c)
Next c

If (LeftDisplay = "Part Oriented Function/Means Tree") Then
    For c = 1 To NumFunc2
        Unload FNode2(c)
```

```
    Next c
    For c = 1 To NumLine5
        Unload Line5(c)
    Next c
End If
If (RightDisplay = "Function Oriented Function/Means Tree") Then
    For c = 1 To NumMeans2
        Unload MNode2(c)
    Next c
    For c = 1 To NumLine6
        Unload Line6(c)
    Next c
  End If

End If

'Load New Instance
ThisInst = NewInst
If (OldInst <> NewInst) Then
    For c = 1 To Instances(NewInst).NumParts - 1
    Load MNode1(c)
    Load Line1(c)
    Line1(c).Visible = True
    Line1(c).X1 = Instances(ThisInst).L1x1(c)
    Line1(c).Y1 = Instances(ThisInst).L1y1(c)
    Line1(c).X2 = Instances(ThisInst).L1x2(c)
    Line1(c).Y2 = Instances(ThisInst).L1y2(c)
Next c
For c = 1 To Instances(NewInst).NumFuncts - 1
    Load FNode1(c)
    Load Line2(c)
    Line2(c).Visible = True
    Line2(c).X1 = Instances(ThisInst).L2x1(c)
    Line2(c).Y1 = Instances(ThisInst).L2y1(c)
    Line2(c).X2 = Instances(ThisInst).L2x2(c)
    Line2(c).Y2 = Instances(ThisInst).L2y2(c)
Next c

For c = 0 To Instances(NewInst).NumParts - 1
    UpdateMeansNode (c)
```

```
    If (Instances(NewInst).PartsTree(c).PartSuppression = "True") Then
       MNode1(c).BackColor = myCol
    Else
       'MNode1(c).BackColor = vbWhite
    End If
Next c
For c = 0 To Instances(NewInst).NumFuncts - 1
    UpdateFunctNode (c)
Next c
    End If
    RedrawLeftDisplay
    RedrawRightDisplay
End Sub

' UPDATE INSTANCES
Private Sub RegenerateInstances(SelInst As Integer)
Dim c As Integer
    InstGrid.Rows = NumInstances
    InstGrid.Col = 0
    For c = 0 To NumInstances - 1
       InstGrid.Row = c
       InstGrid.Text = Instances(c).Name
    Next c
    InstGrid.Row = SelInst
End Sub
Private Sub SuppressPart(myPart As Integer)
Dim c As Integer
    'First ckeck to see if it is already suppressed
    If (Instances(ThisInst).PartsTree(myPart).PartSuppression = "True") Then
       MsgBox ("Part is ALREADY SUPPRESSED")

    'Part is NOT suppressed, so find all children to suppress too
    Else
       Instances(ThisInst).PartsTree(myPart).PartSuppression = "True"
       MNode1(Instances(ThisInst).PartsTree(myPart).NodeID).BackColor = myCol

    End If
End Sub
```

```
' UNSUPPRESS A PART
Private Sub ResumePart(myPart As Integer)
Dim c As Integer
Dim Suppressed As Boolean

   'First ckeck to see if it is already resumed
   If (Instances(ThisInst).PartsTree(myPart).PartSuppression = "False") Then
      MsgBox ("Part is NOT SUPPRESSED")
'Part is suppressed, so find all children to resume too
   Else
      Instances(ThisInst).PartsTree(myPart).PartSuppression = "False"
      MNode1 (Instances(ThisInst).PartsTree(myPart).NodeID).BackColor = vbWhite
   End If
End Sub
Private Sub ShowParams(myInst As Integer)
'Dim c, nParams As Integer
'   nParams = Instances(myInst).PartsTree(SelPart).NumOfParams
'   ParamsGrid.Rows = nParams + 1
'   If (nParams > 0) Then
'      For c = 0 To nParams - 1
'         ParamsGrid.Row = c + 1
'         ParamsGrid.Col = 0
   If (SelPart >= 0) Then
      DisplayPartsParams SelPart
   End If
End Sub
Private Sub ShowRelations(myInst As Integer)
Dim c, nRels As Integer
   nRels = Instances(myInst).NumRels
   RelsGrid.Rows = nRels + 1
   If (nRels > 0) Then
      For c = 0 To nRels - 1
         RelsGrid.Row = c + 1
         RelsGrid.Col = 0
         RelsGrid.Text = Instances(myInst).PartsTree(Instances(myInst).RelM(c)).Name
         RelsGrid.Col = 1
         RelsGrid.Text = Instances(myInst).FunctTree(Instances(myInst).RelF(c)).Name
      Next c
   End If
End Sub
```

```
' SUPPRESS FEATURES
Sub FeatureSuppression(SearchStr, Action)

Dim swApp As Object     ' Variable used to hold the SldWorks object
Dim Model As Object     ' Variable used to hold the ModelDoc object
Dim feat As Object      ' Variable used to hold the current Feature object
Dim featureName As String
Const swDocPART = 1                 ' These definitions are consistent with type names
Const swDocASSEMBLY = 2               ' defined in swconst.bas
Const swDocDRAWING = 3
Set swApp = CreateObject("SldWorks.Application")
Set Model = swApp.ActiveDoc             ' Attach to the active document
If Model Is Nothing Then        ' Exit if no model is active
   Exit Sub
End If
If (Model.GetType <> swDocPART) Then        ' Do not allow drawings or assemblies
   Msg = "Only Allowed on Parts"            ' Define message
   Style = vbOKOnly                       ' OK Button only
   Title = "Error"                        ' Define title
   Call MsgBox(Msg, Style, Title)           ' Display error message
   Exit Sub                             ' Exit this program
End If
Set feat = Model.FirstFeature               ' Get the 1st feature in part
Do While Not feat Is Nothing                ' While we have a valid feature
Let featureName = feat.Name                 ' Get the name of the feature
If InStr(1, featureName, SearchStr, 1) Then ' See if the feature name
res = Model.SelectByID(featureName, "BODYFEATURE", 0, 0, 0)
If (Action = "SUPPRESS") Then         ' User chose to suppress
        res = Model.EditSuppress()          ' Suppress the feature
      ElseIf (Action = "RESUME") Then    ' User chose to unsuppress
        res = Model.EditUnsuppress()        ' Unsuppress the feature
      End If
    End If
   Set feat = feat.GetNextFeature()         ' Get the next feature
  Loop                                  ' Continue until no more features exist
End Sub
```

Notes:

Appendix II

Case Studies – Further Examples

AII.1 Guindy Machine Tools Ltd. Lathe Chuck

This section contains the following material:

1) *Figures AII.1.1 – AII.1.13, sample manufacturing drawings for a Production GMT Lathe Chuck,*
2) *Figures AII.1.14 – AII.1. 25, the Generic Master Parts created in Pro/ENGINEER for the Chuck family,*
3) *Figures AII.1.26 – AII.1.38, manufacturing drawings created in Pro/ENGINEER for a sample Chuck,*
4) *Figures AII.1.39 – AII.1.42, variants of the Master Model representing actual Production Chucks,*

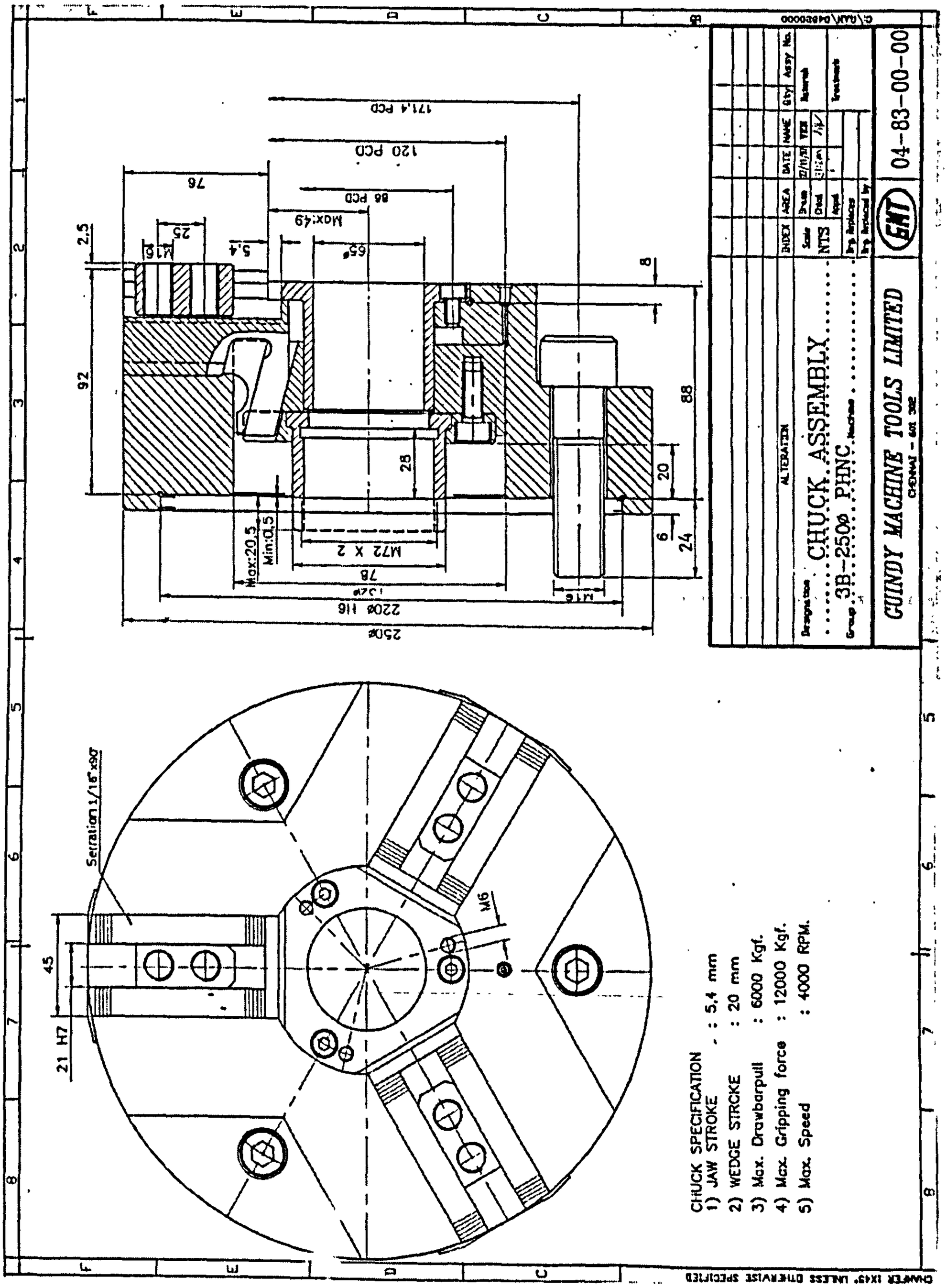


Figure AII.1.1 – GMT Chuck Assembly – Example Manufacturing Drawing

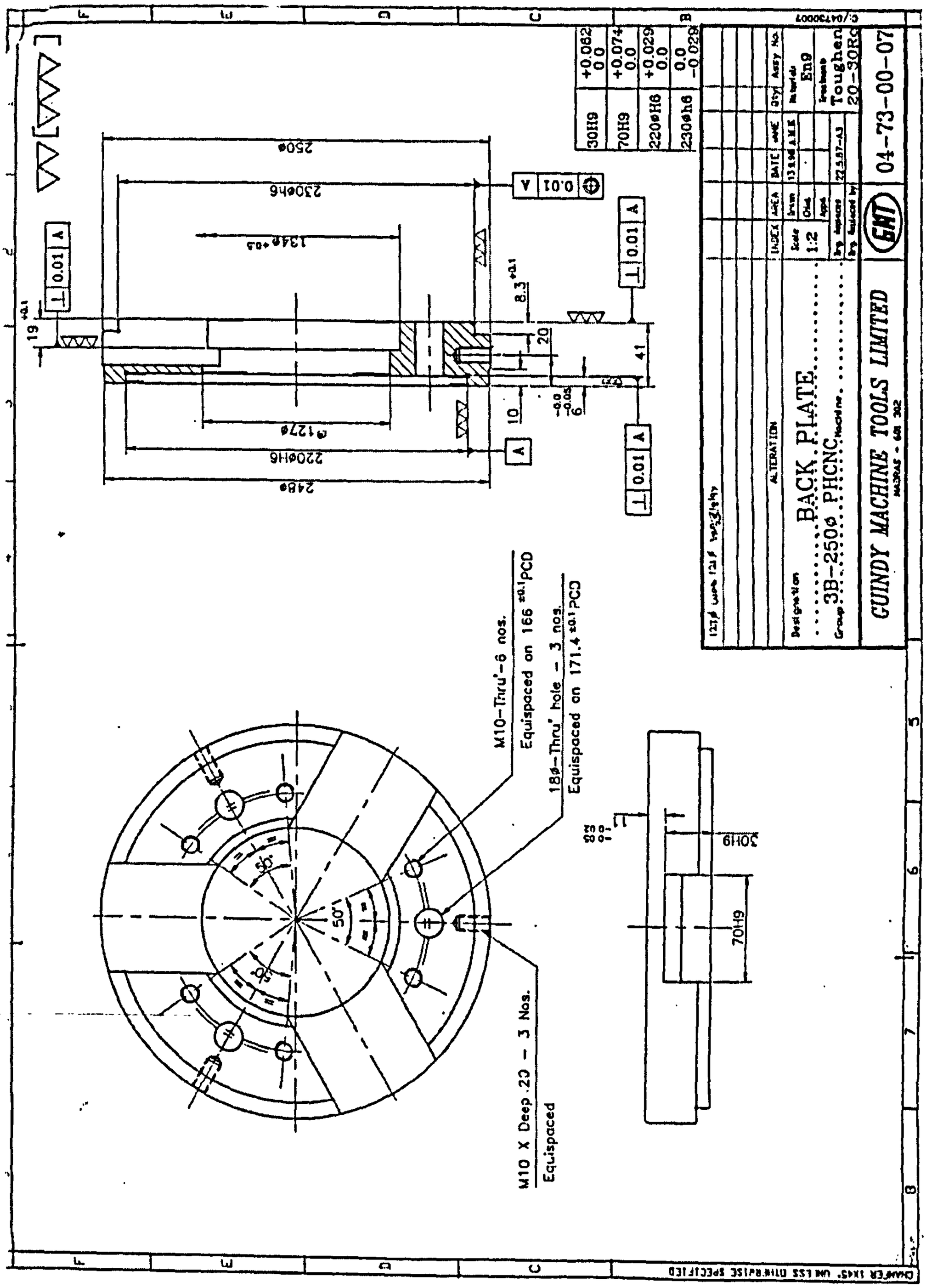


Figure AII.1.2 – GMT Chuck Back-Plate – Example Manufacturing Drawing

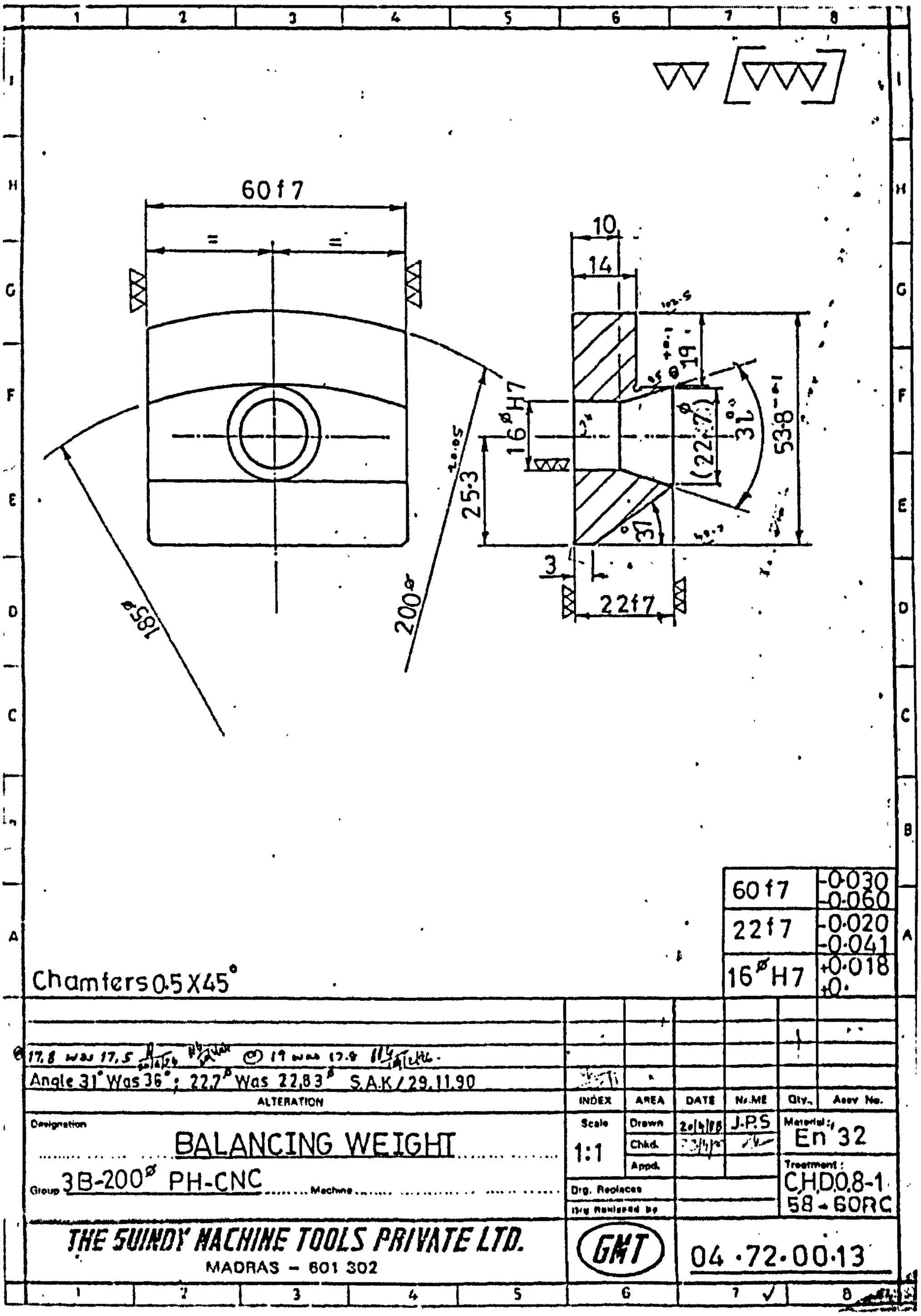


Figure AII.1.3 – GMT Chuck Balancing Weight – Example Manufacturing Drawing

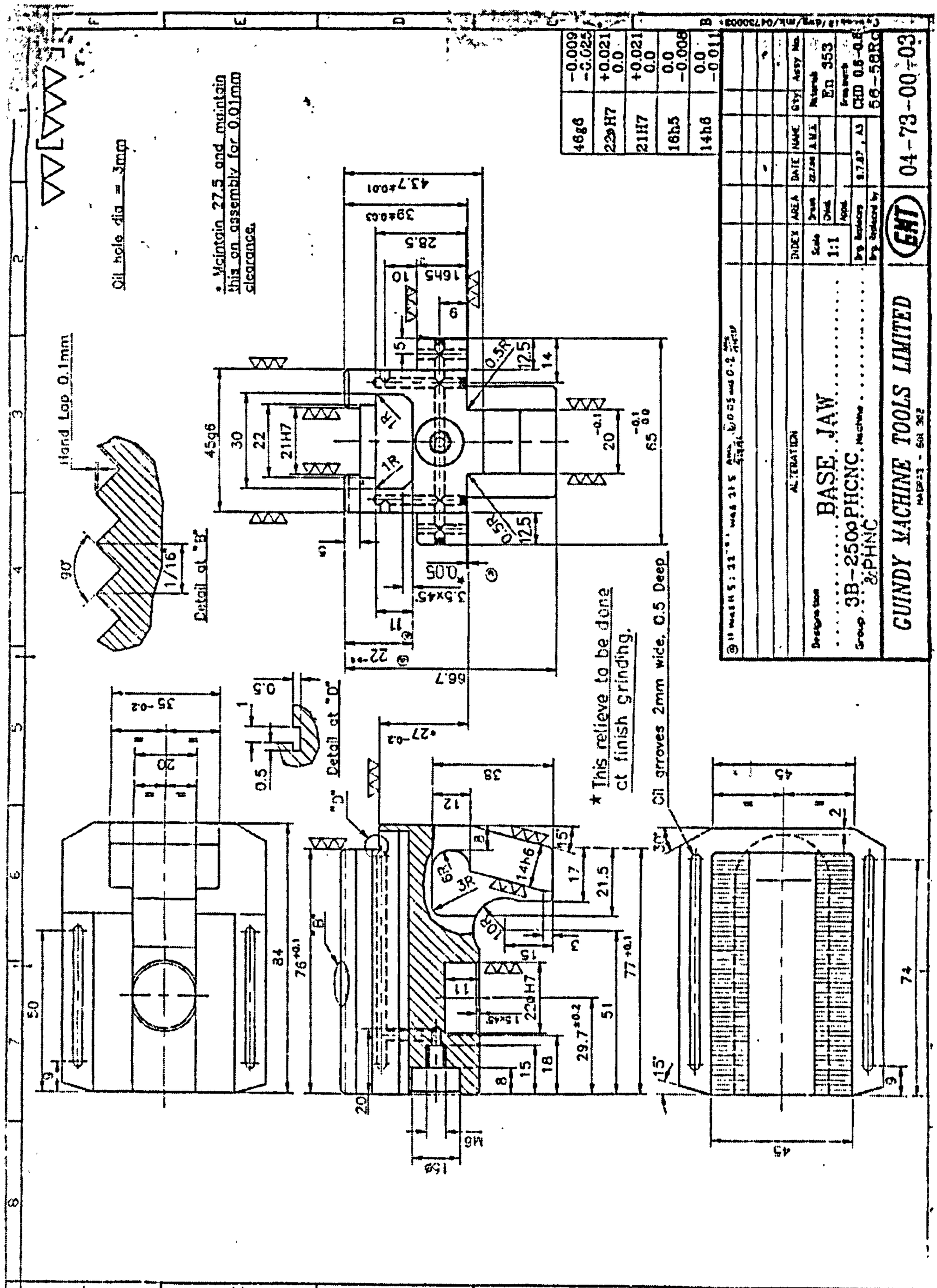


Figure AII.1.4 – GMT Chuck Base-Jaw – Example Manufacturing Drawing

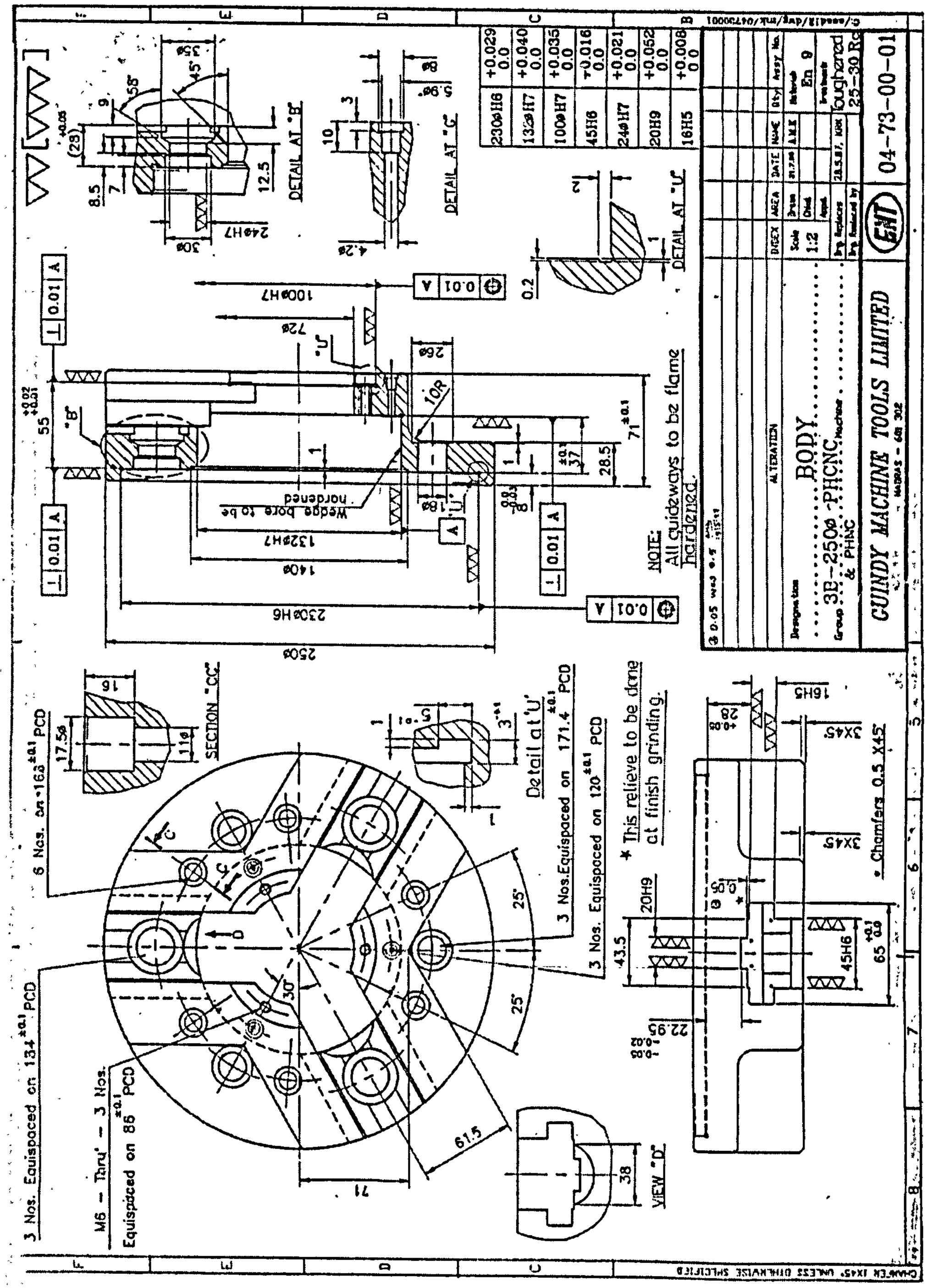


Figure AII.1.5 – GMT Chuck Body – Example Manufacturing Drawing

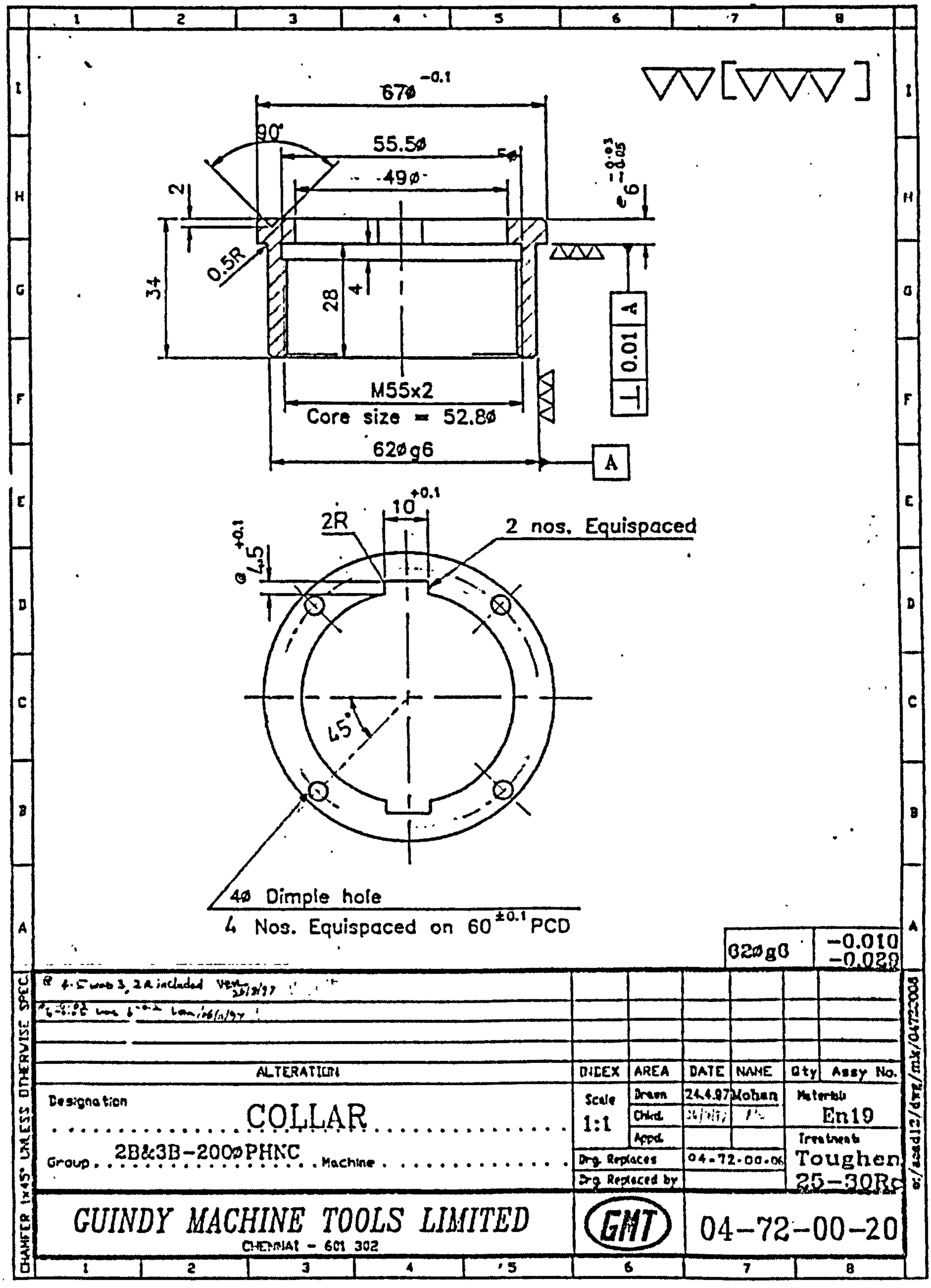


Figure AII.1.6 – GMT Chuck Collar – Example Manufacturing Drawing

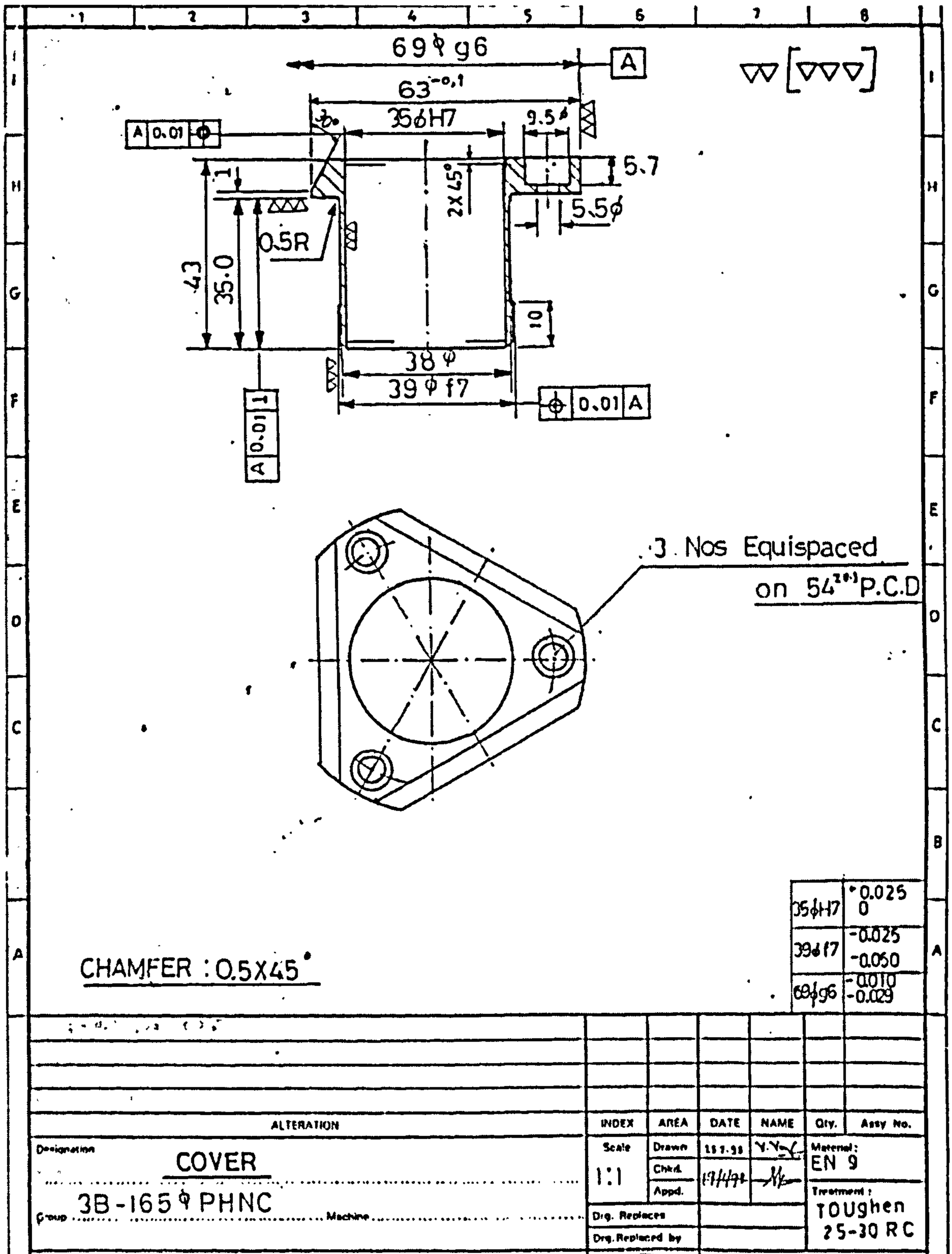


Figure AII.1.7 – GMT Chuck Cover – Example Manufacturing Drawing

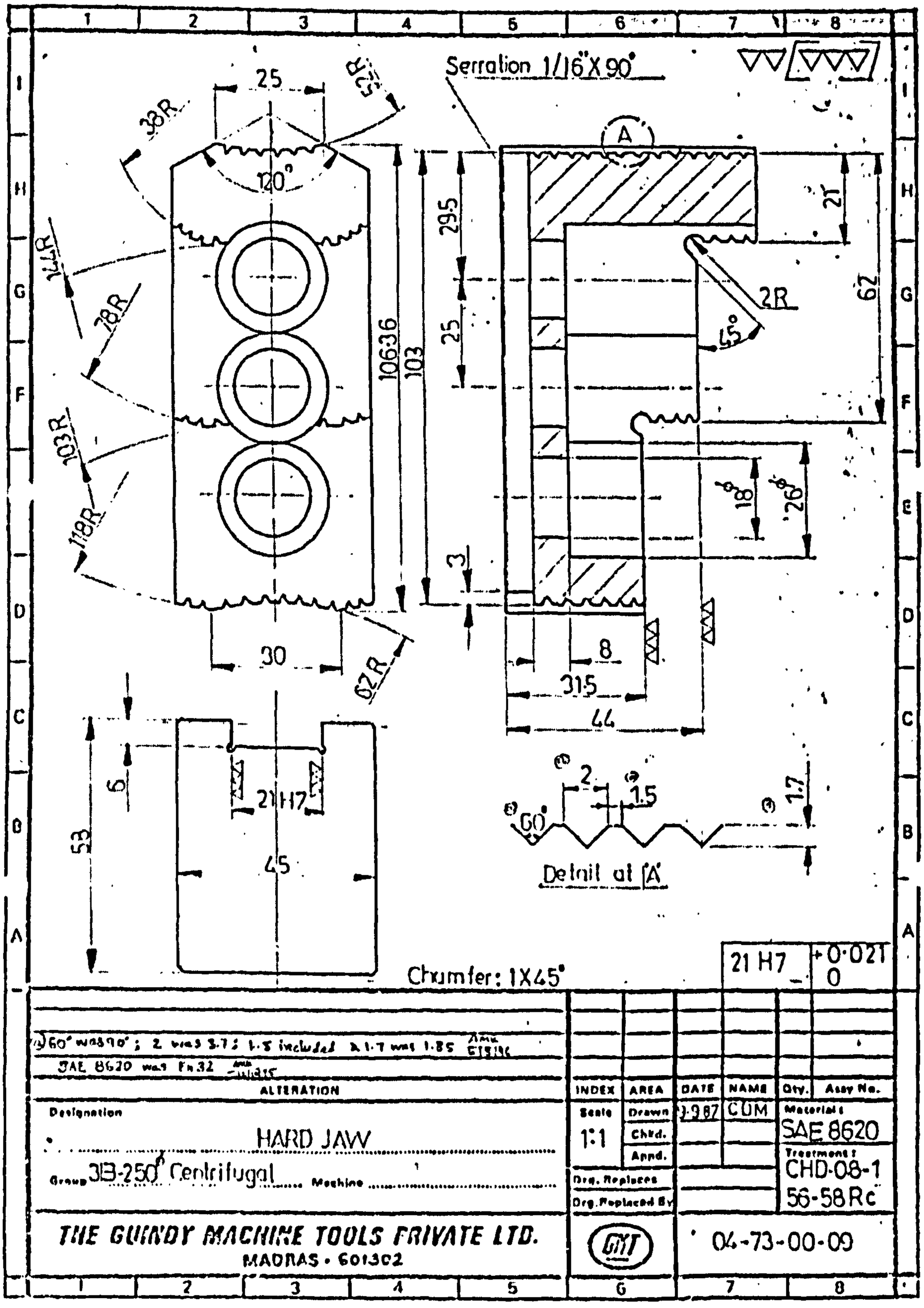
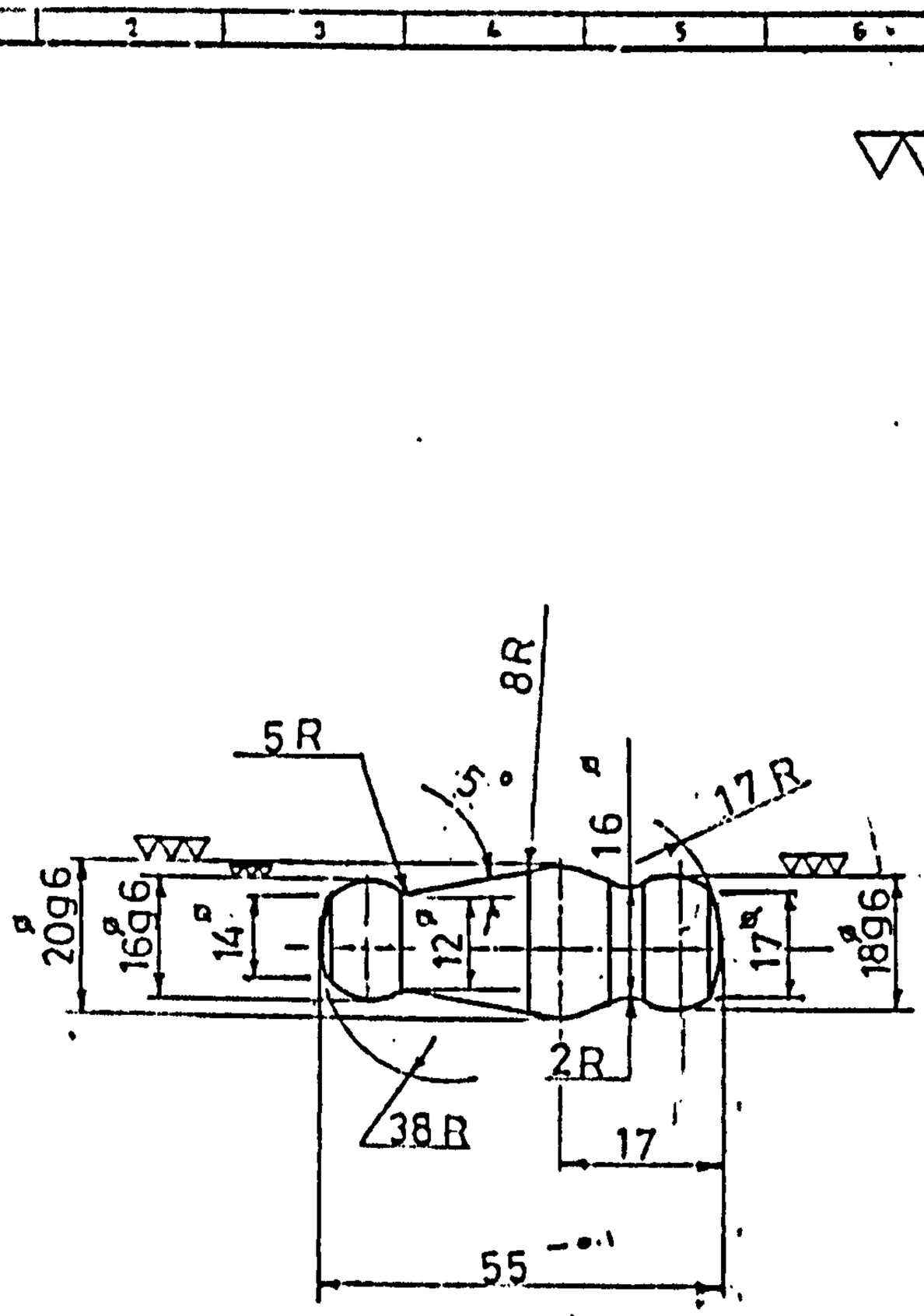


Figure AII.1.8 – GMT Chuck Hard-Jaw – Example Manufacturing Drawing



Remove all sharp corners.

20g6	-0.007 -0.020
18g6	-0.006 -0.017
16g6	-0.006 -0.017

ALTERATION	INDEX	AREA	DATE	NAME	Qty	Assy No.

Designation: LEVER

Group: 3B-200 PH-CNC Machine:

Scale: 1:1 | Drawn: 9/4/88 J.P.S | Chkd. | Appd.

Drg Replaced

Drg. Replaced by

Material: En-36

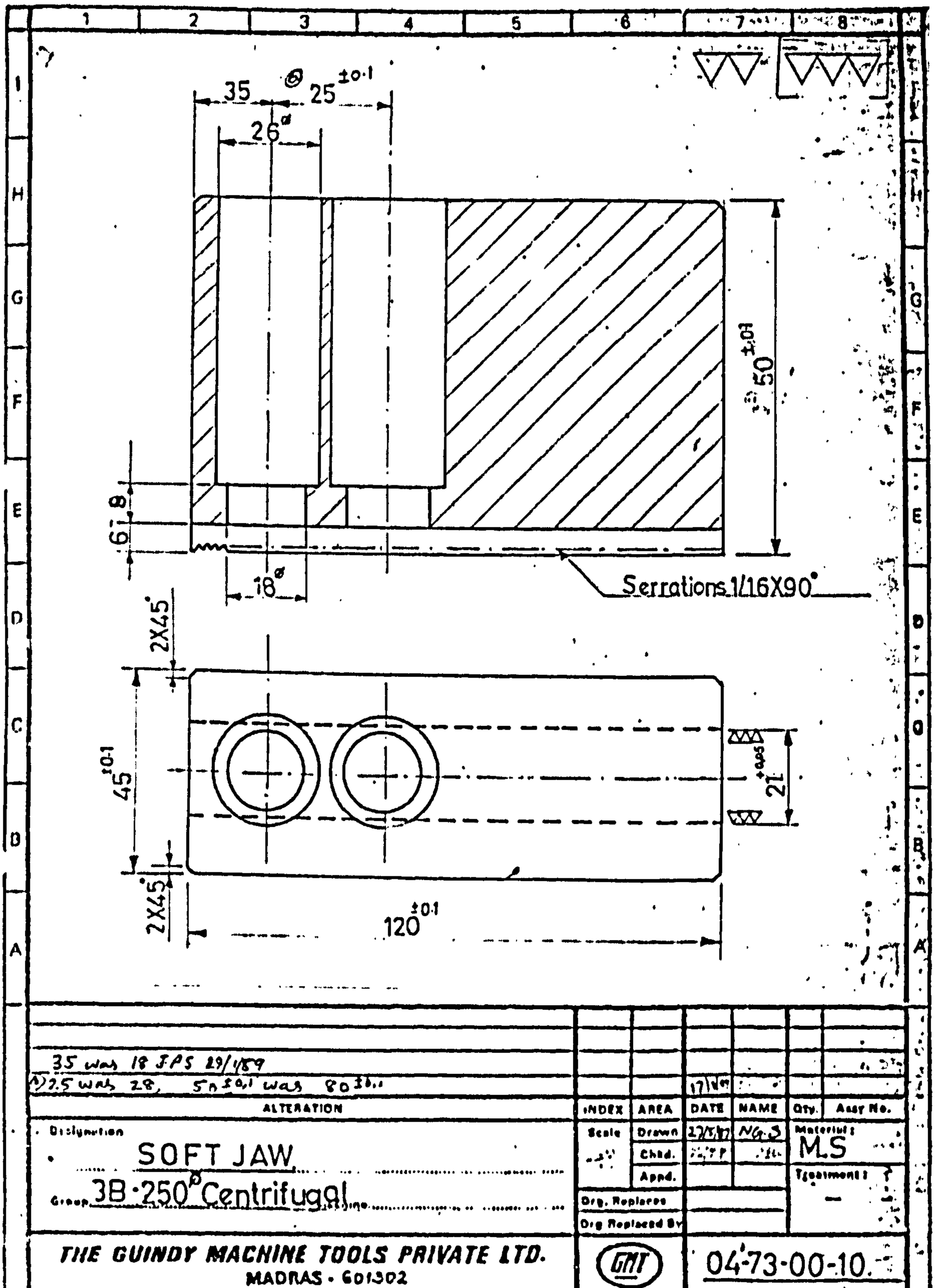
Treatment: CHD-6-8 55-60RC

THE GUINDY MACHINE TOOLS PRIVATE LTD.
MADRAS - 601 302

GMT

04·72·00·12

Figure AII.1.9 – GMT Chuck Lever – Example Manufacturing Drawing



35 was 18 JPS 29/1/89
A) 2.5 was 28, 5n±0.1 was 80±0.1

ALTERATION

Designation: SOFT JAW

Group: 3B·250ø Centrifugal

INDEX	AREA	DATE	NAME	Qty.	Assy No.
Scale	Drawn	27/5/87	NG.S	Material: MS	
	Chkd.			Treatment: —	
	Appd.				
Drg. Replaces					
Drg Replaced By					

THE GUINDY MACHINE TOOLS PRIVATE LTD.
MADRAS - 601302

GMT

04-73-00-10.

Figure AII.1.10 – GMT Chuck Soft-Jaw – Example Manufacturing Drawing

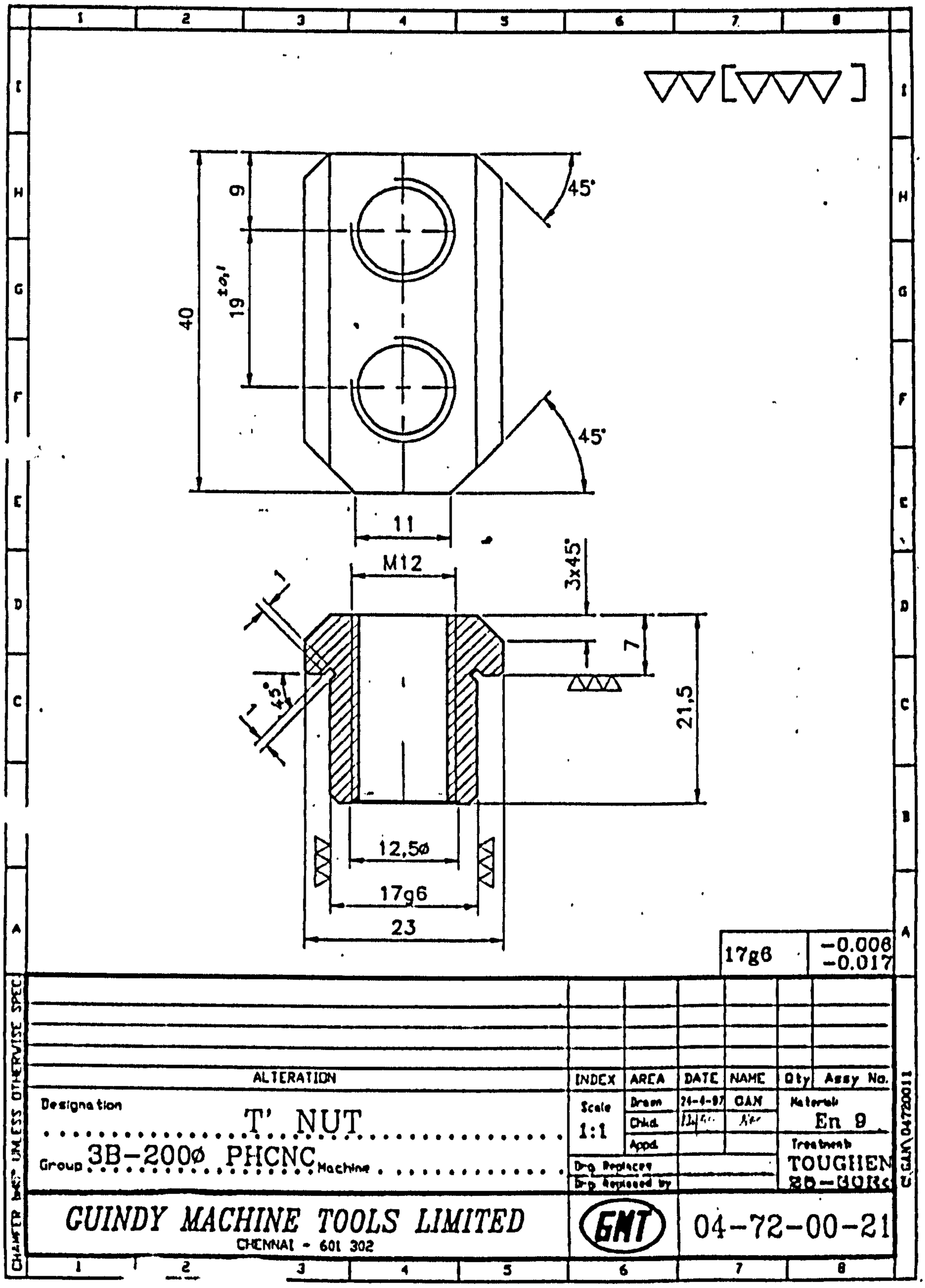


Figure AII.1.11 – GMT Chuck T-Nut – Example Manufacturing Drawing

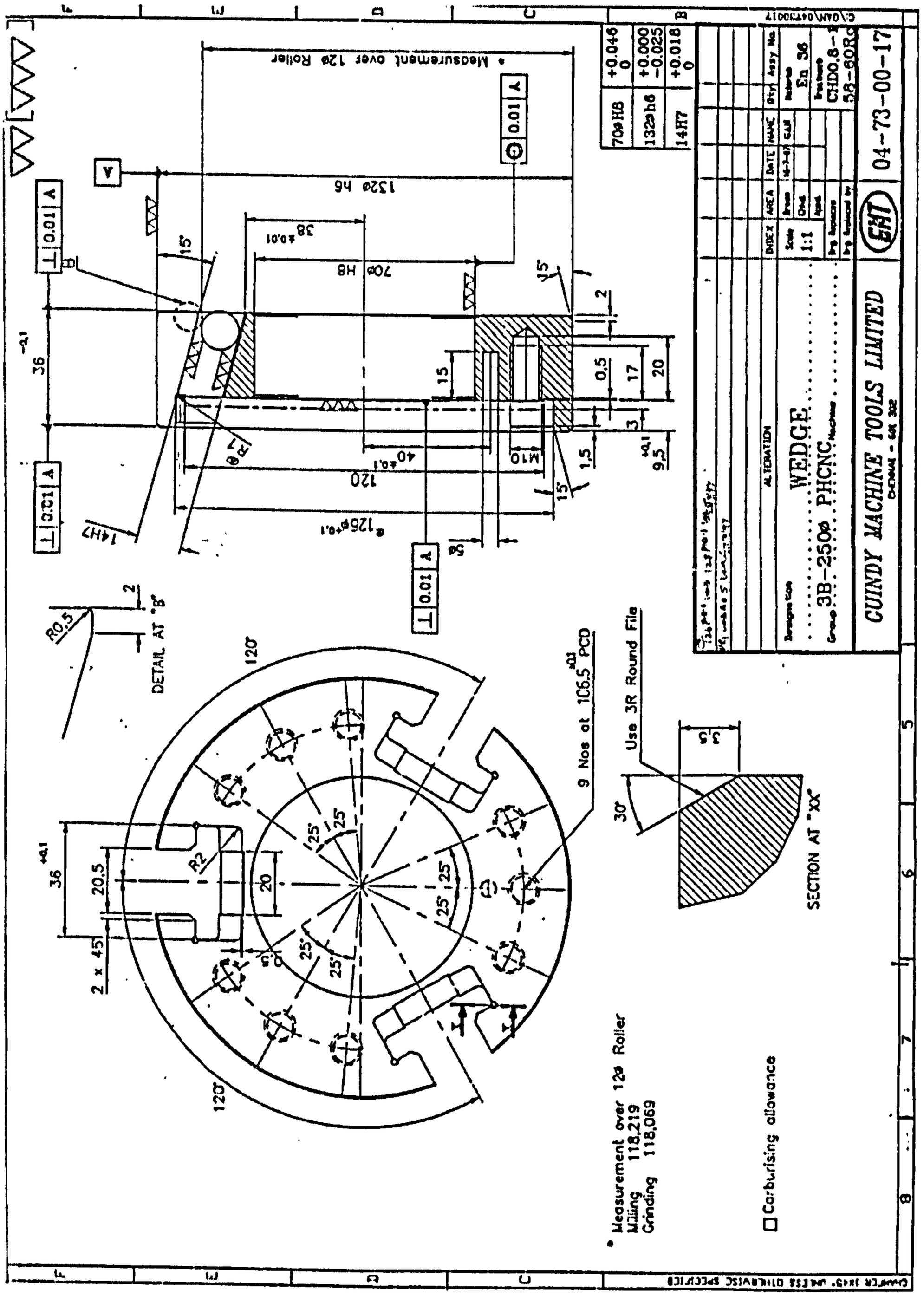


Figure AII.1.12 – GMT Chuck Wedge – Example Manufacturing Drawing

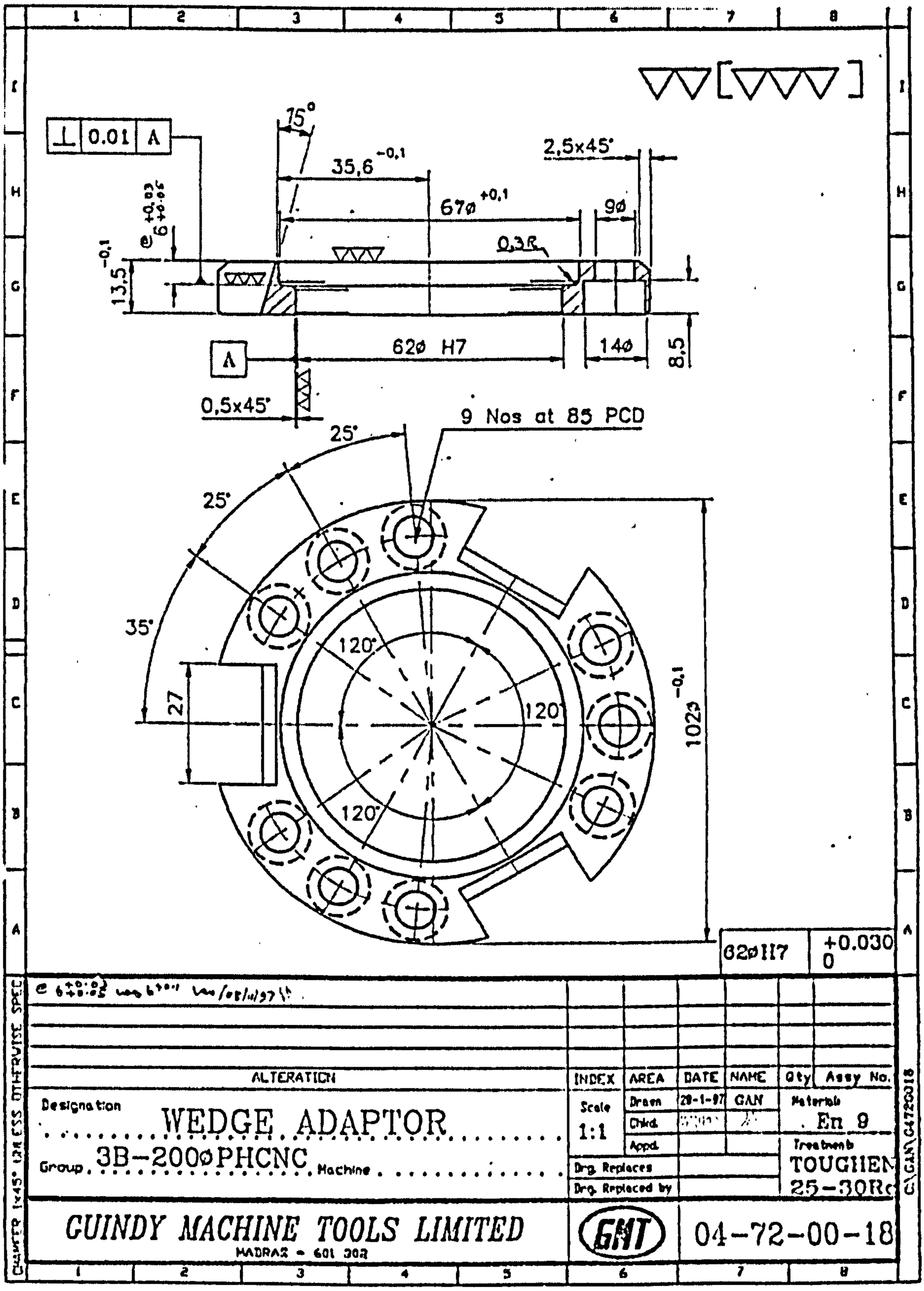


Figure AII.1.13 – GMT Chuck Wedge Adaptor – Example Manufacturing Drawing

Parameter	Value
Outer Diameter	200mm
Back Seat Diameter	170mm
Back Seat Depth	6mm
Back Bore Diameter	102mm
Front Bore Diameter	111mm
Front Bore Depth	19mm
Body Seat Diameter	19mm
Body Seat Depth	8.3mm
Lever Hole PCD	133.4mm
Lock Bolt Offset	15mm
Depth	33mm
Fix Bolt PCD	160mm
Fix Bolt Diameter	10mm
Fix Bolt Offset Angle	20°
Lock Bolt Diameter	10mm
Lock Bolt Depth	20mm
Weight Depth	22mm
Weight Height	60mm
Num Jaws	3

Non-Persistent Feature	Status

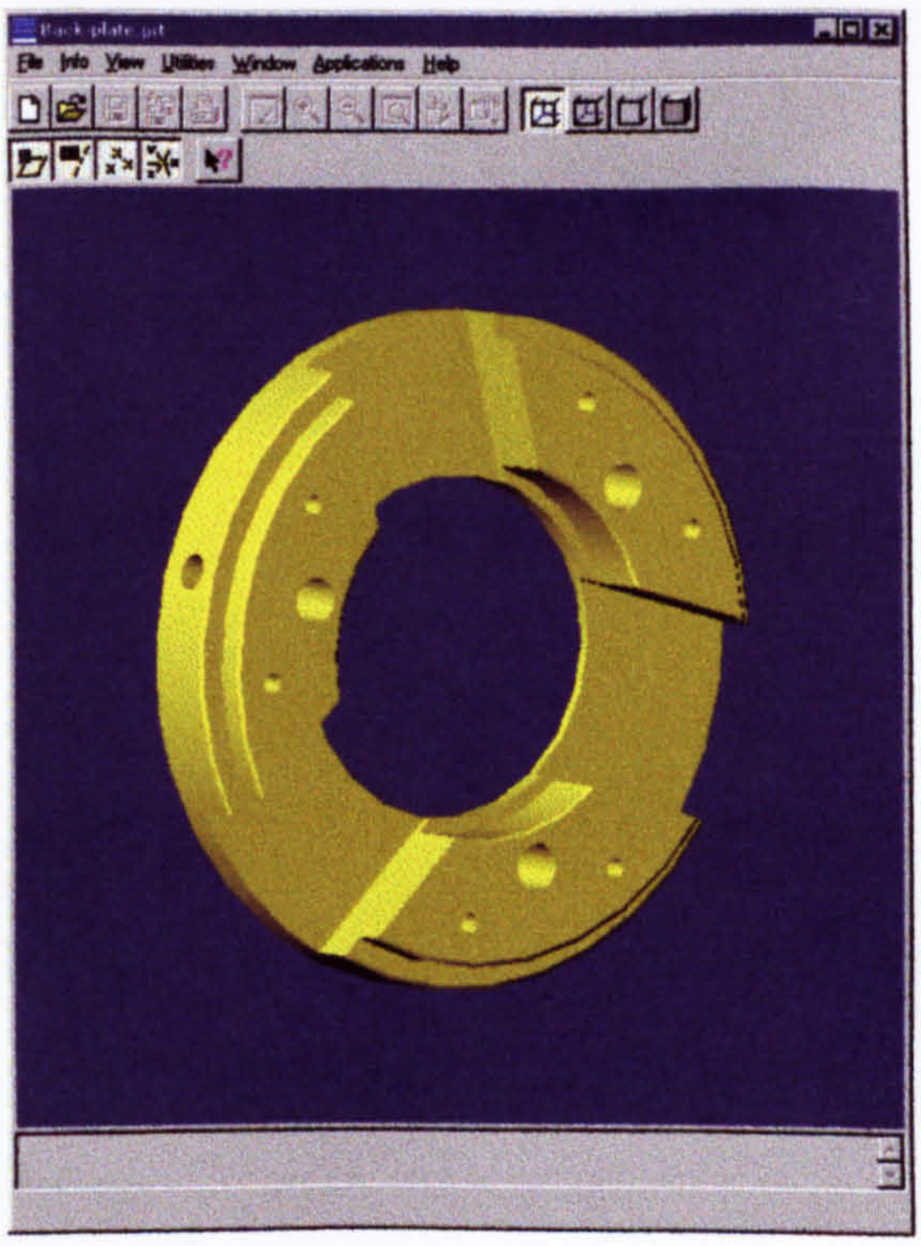


Figure 5.1.14 – GMT Generic Back Plate - CAD Model & Parameters

Parameter	Value
Width	60mm
Depth	22mm
Chuck Diameter	200mm
Seat Diameter	185mm
Bore Diameter	16mm
Straight Bore Depth	10mm
Seat Offset	14mm

Non-Persistent Feature	Status

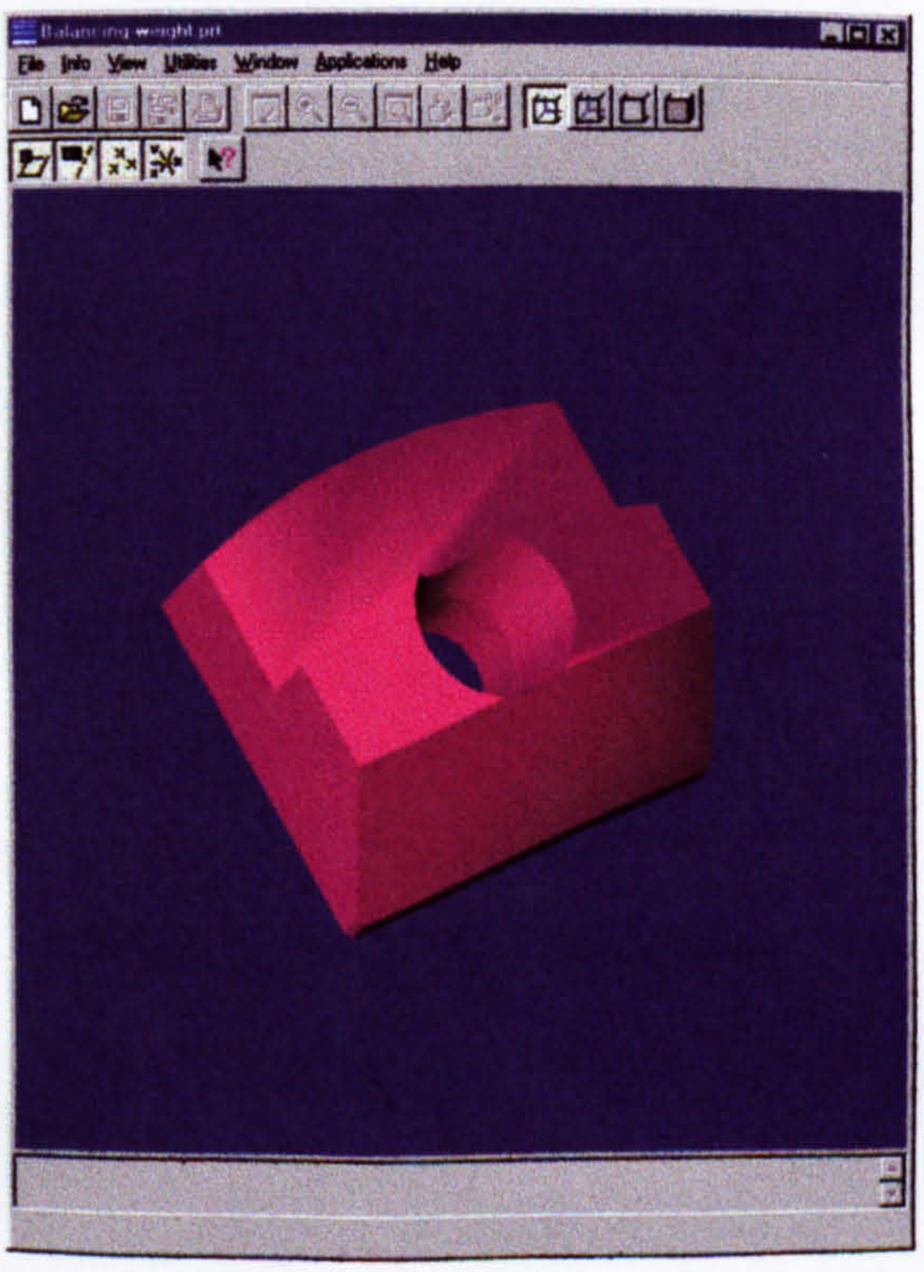


Figure 5.1.15 – GMT Generic Balancing Weight - CAD Model & Parameters

Parameter	Value
Length	67.5mm
Width	60mm
Height	65mm
Body Contact Width	35mm
T Nut Width	mm
Cut Out Width	14mm
Wedge Separation	19mm
Nose Major Width	
Nose Minor Width	
Wedge Angle	
Jaw Contact Length	
Bore Horiz Offset	
Bore Vertical Offset	
Bore Diameter	
Bore Depth	
Nose Depth	
Major Conatct Height	
Minor Contact Height	
Lever Offset	

Non-Persistent Feature	Status

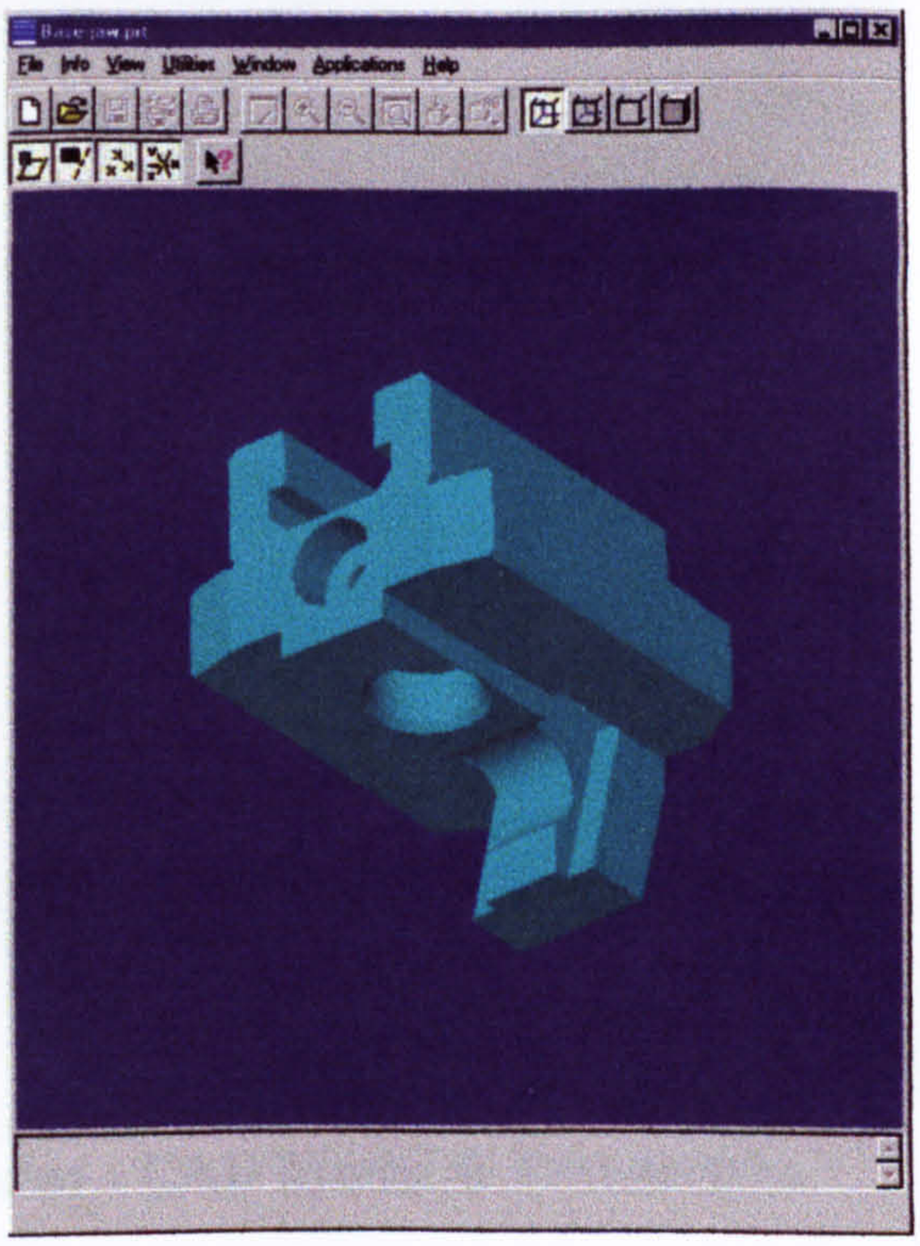


Figure 5.1.16 – GMT Generic Base Jaw - CAD Model & Parameters

Parameter	Value
Length	70mm
Width	35mm
Height	40mm
Fillet_2	14mm
Major_Bolt_Diameter	20mm
Minor_Bolt_Diameter	14mm
Bolt_Separation	19mm

Non-Persistent Feature	Status
Back Plate Bolt Hole	Resume
Lever Pivot Hole	Resume

Figure 5.1.17 – GMT Generic Body - CAD Model & Parameters

Parameter	Value
Lip Diameter	67mm
Seat Diameter	55.5mm
Top Bore Diameter	49mm
Height	34mm
Lip Depth	6mm
Core Diameter	55mm
Cylinder Diameter	62mm
Cut Width	10mm
Cut Height	4.5mm
PCD	60mm

Non-Persistent Feature	Status
Cut Out	Resume

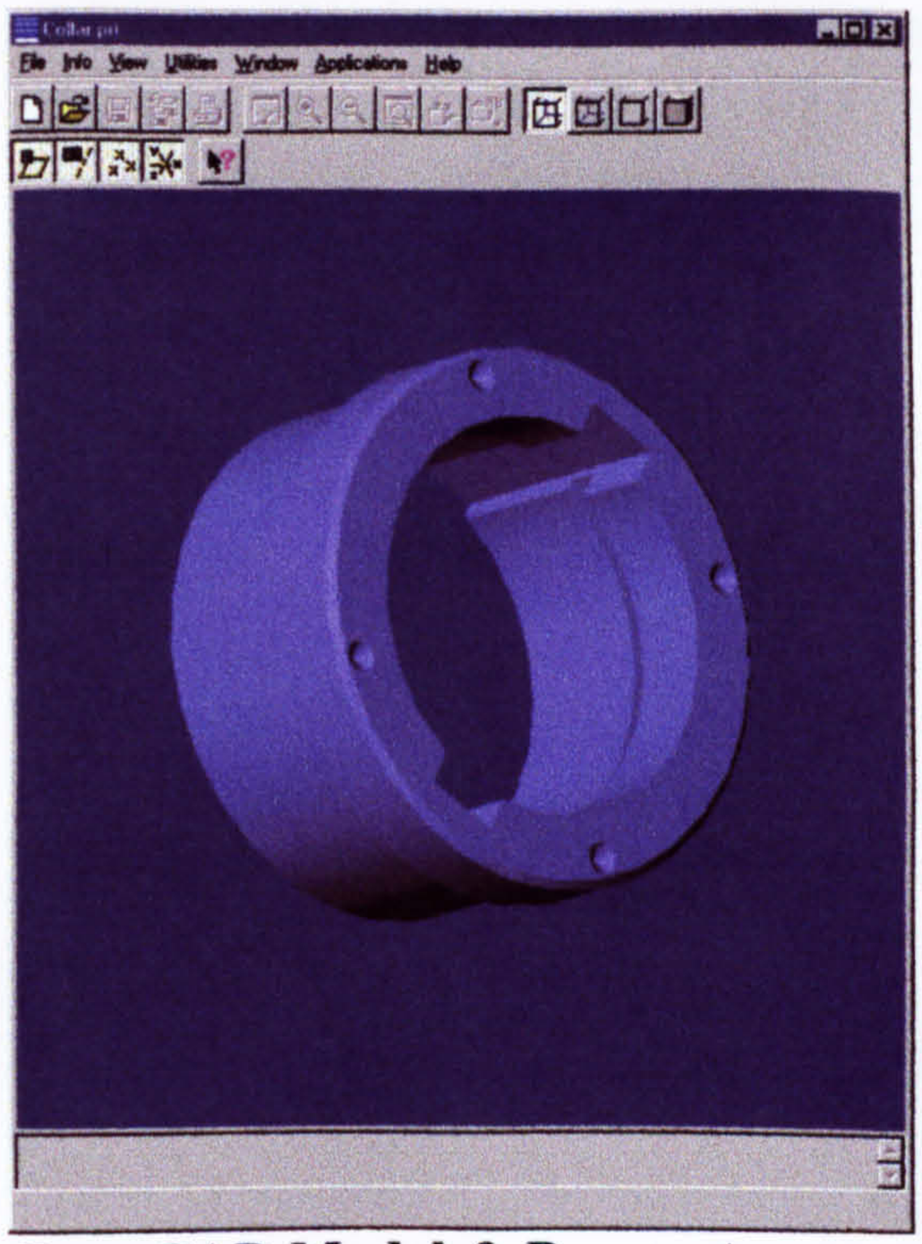


Figure 5.1.18 – GMT Generic Collar - CAD Model & Parameters

Parameter	Value
Outer Diameter	82mm
Chamfer End	76.7mm
Bore Diameter	48mm
Cylinder Diameter	53mm
Chamfer Offset	3mm
Height	51mm
Lip Offset	44mm
Bolt Major Diameter	11mm
Bolt Major Depth	5.7mm
Bolt Minor Diameter	6.6mm
Loc Hole Diameter	6mm
Loc Hole Offset Angle	16°
Bolt PCD	68mm
Num Jaws	3

Non-Persistent Feature	Status
Pin Holes	Resume

Figure 5.1.19 – GMT Generic Cover - CAD Model & Parameters

Parameter	Value
Length	75.07mm
Width	35mm
Height	50mm
Major Bolt Diameter	20mm
Minor Bolt Diameter	14mm
Bolt Separation	19mm

Non-Persistent Feature	Status

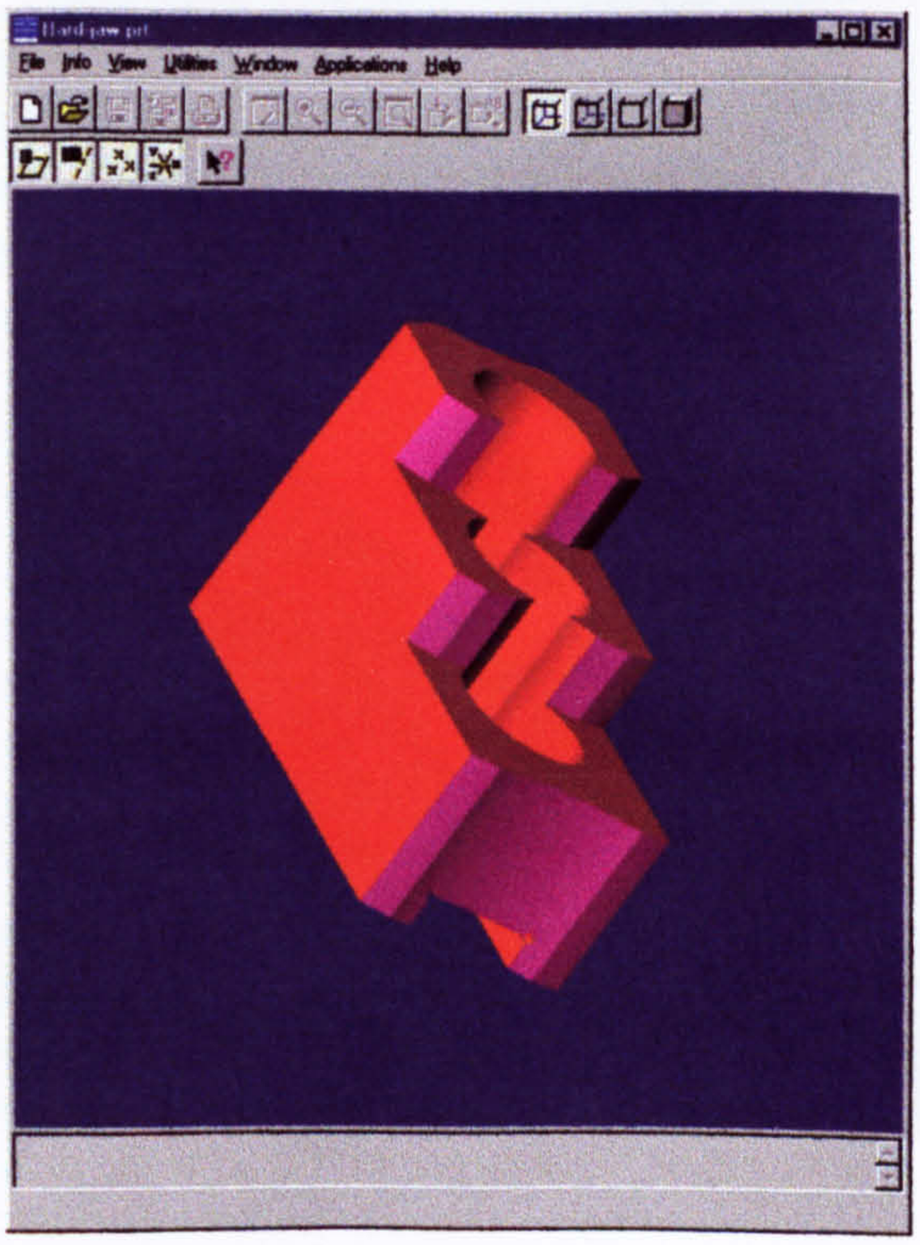


Figure 5.1.20 – GMT Generic Hard Jaw - CAD Model & Parameters

Parameter	Value
Length	55mm
Centre Ball Diameter	20mm
Rear Ball Diameter	16mm
Front Ball Diameter	18mm
Centre Offset	17mm

Non-Persistent Feature	Status

Figure 5.1.21 – GMT Generic Lever - CAD Model & Parameters

Parameter	Value
Length	70mm
Width	35mm
Height	40mm
Major Bolt Diameter	20mm
Minor Bolt Diameter	14mm
Bolt Separation	19mm

Non-Persistent Feature	Status

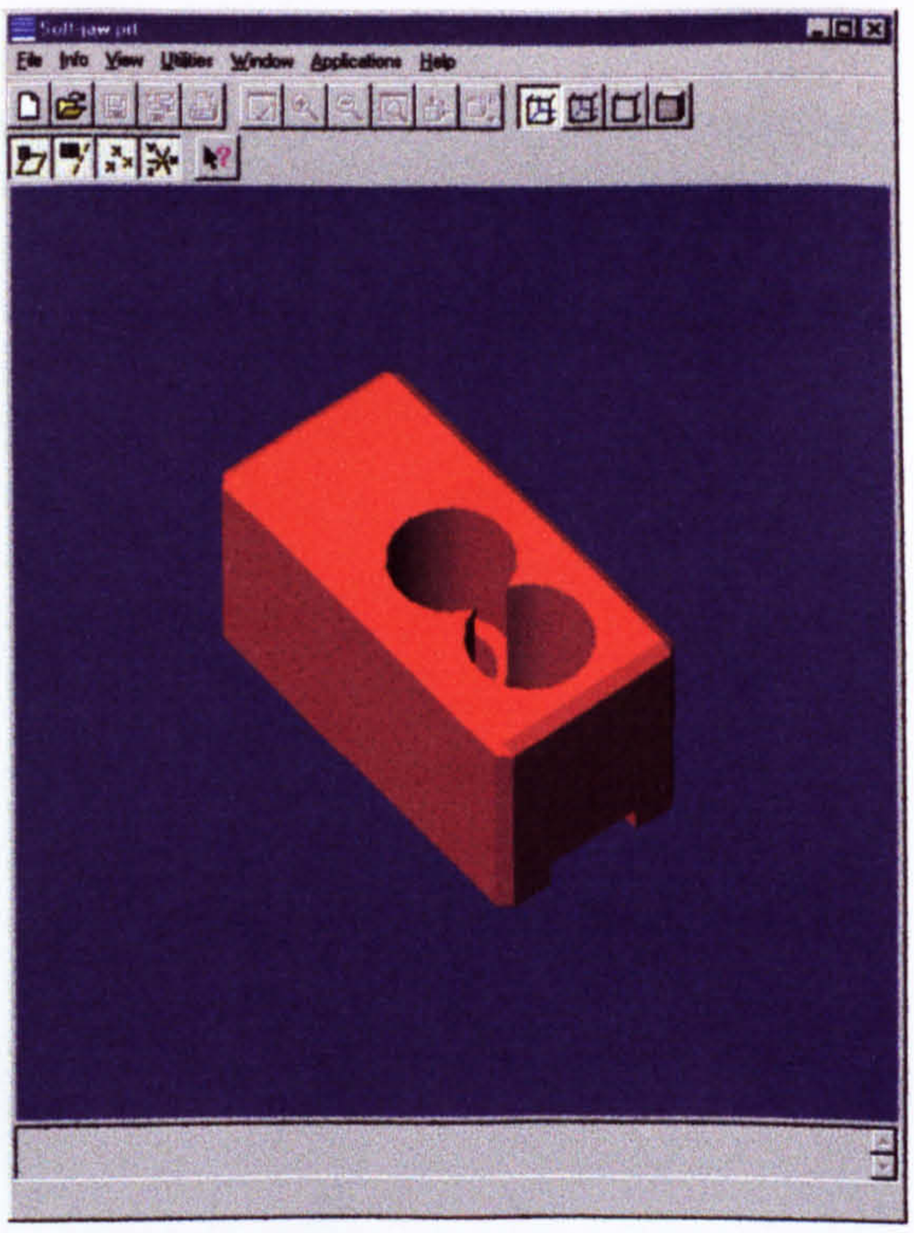


Figure 5.1.22 – GMT Generic Soft Jaw - CAD Model & Parameters

Parameter	Value
Length	40mm
Height	21.5mm
Base Width	17mm
Top Width	23mm
Nose Width	11mm
Bore Diameter	12mm
Hole Separation	19mm
Back Hole Offset	9mm

Non-Persistent Feature	Status

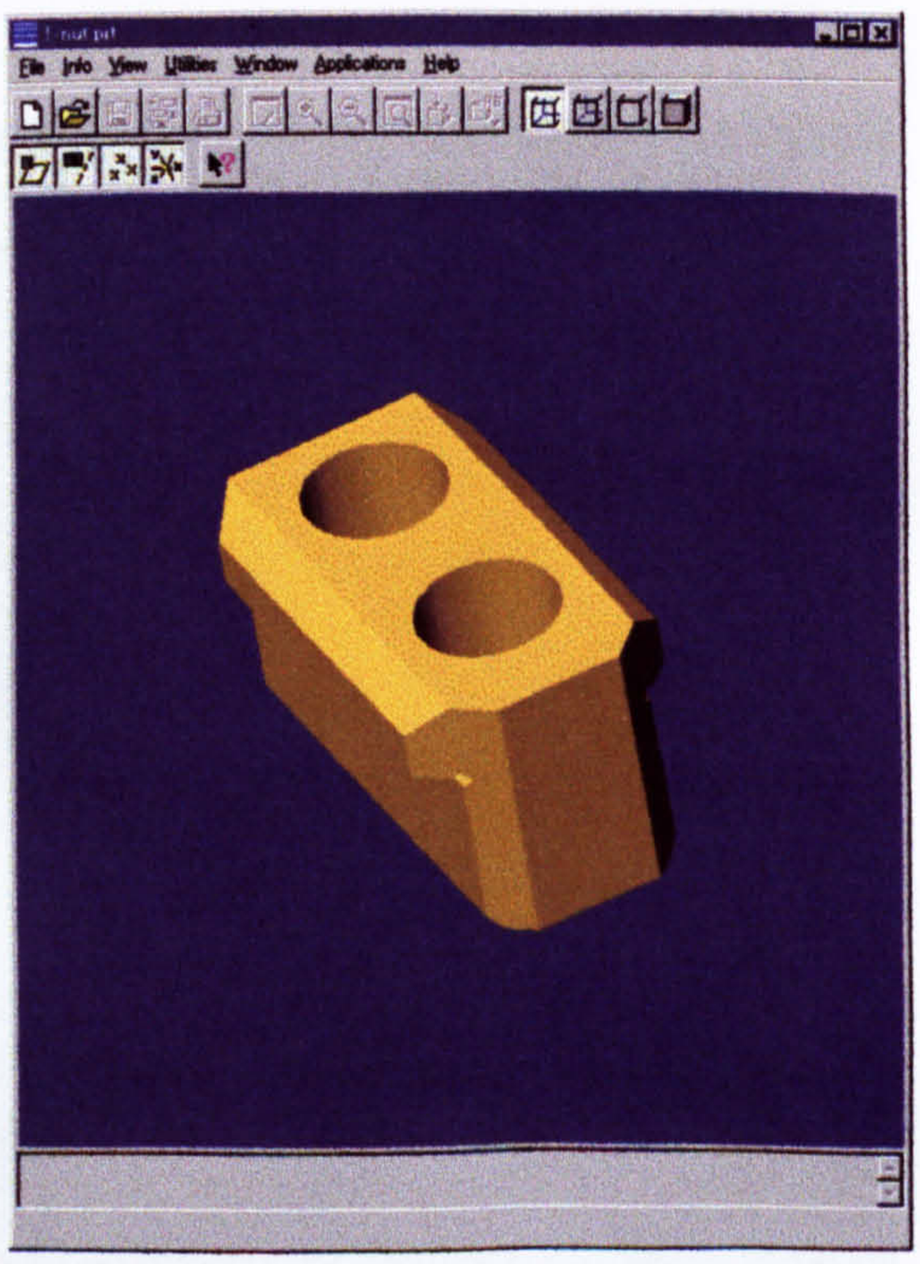


Figure 5.1.23 – GMT Generic 'T'-Nut - CAD Model & Parameters

Parameter	Value
Outer Diameter	102mm
Bolt Offset Angle	35°
Wedge Width	27mm
Num Bolts	3mm
Num Jaws	3mm
Minor Bolt Diameter	9mm
Major Bolt Diameter	14mm
Major Bolt Depth	8.5mm
Seat Diameter	67mm
Seat Depth	6mm
Bore Diameter	62mm
Height	13.5mm
Wedge Offset	35.6mm

Non-Persistent Feature	Status
Oil hole	Resume

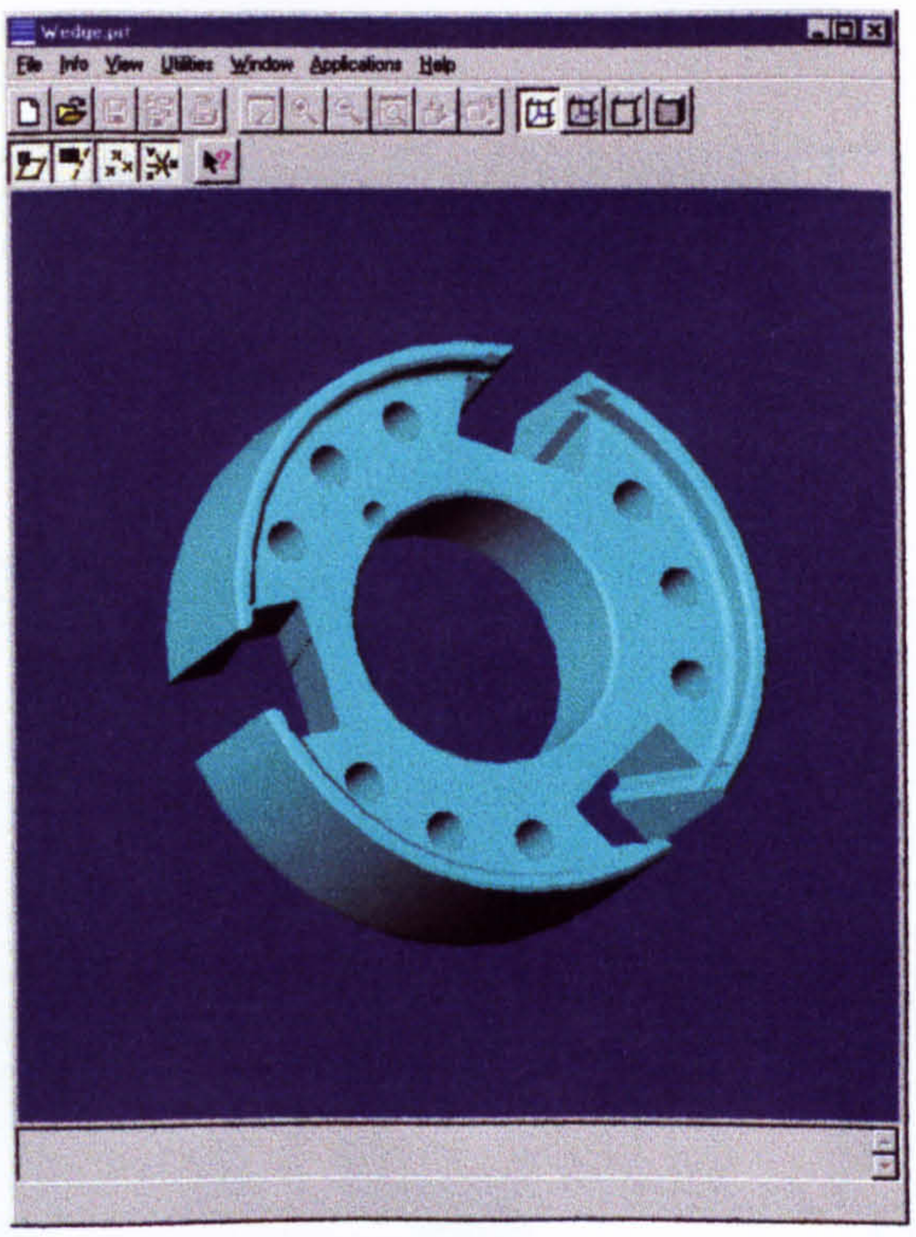


Figure 5.1.24 – GMT Generic Wedge - CAD Model & Paramete

Parameter	Value
Outer Diameter	102mm
Bolt Offset Angle	35°
Wedge Width	27mm
Num Bolts	3mm
Num Jaws	3mm
Minor Bolt Diameter	9mm
Major Bolt Diameter	14mm
Major Bolt Depth	8.5mm
Seat Diameter	67mm
Seat Depth	6mm
Bore Diameter	62mm
Height	13.5mm
Wedge Offset	35.6mm

Non-Persistent Feature	Status

Figure 5.1.25 – GMT Generic Wedge Adaptor - CAD Model & Parameters

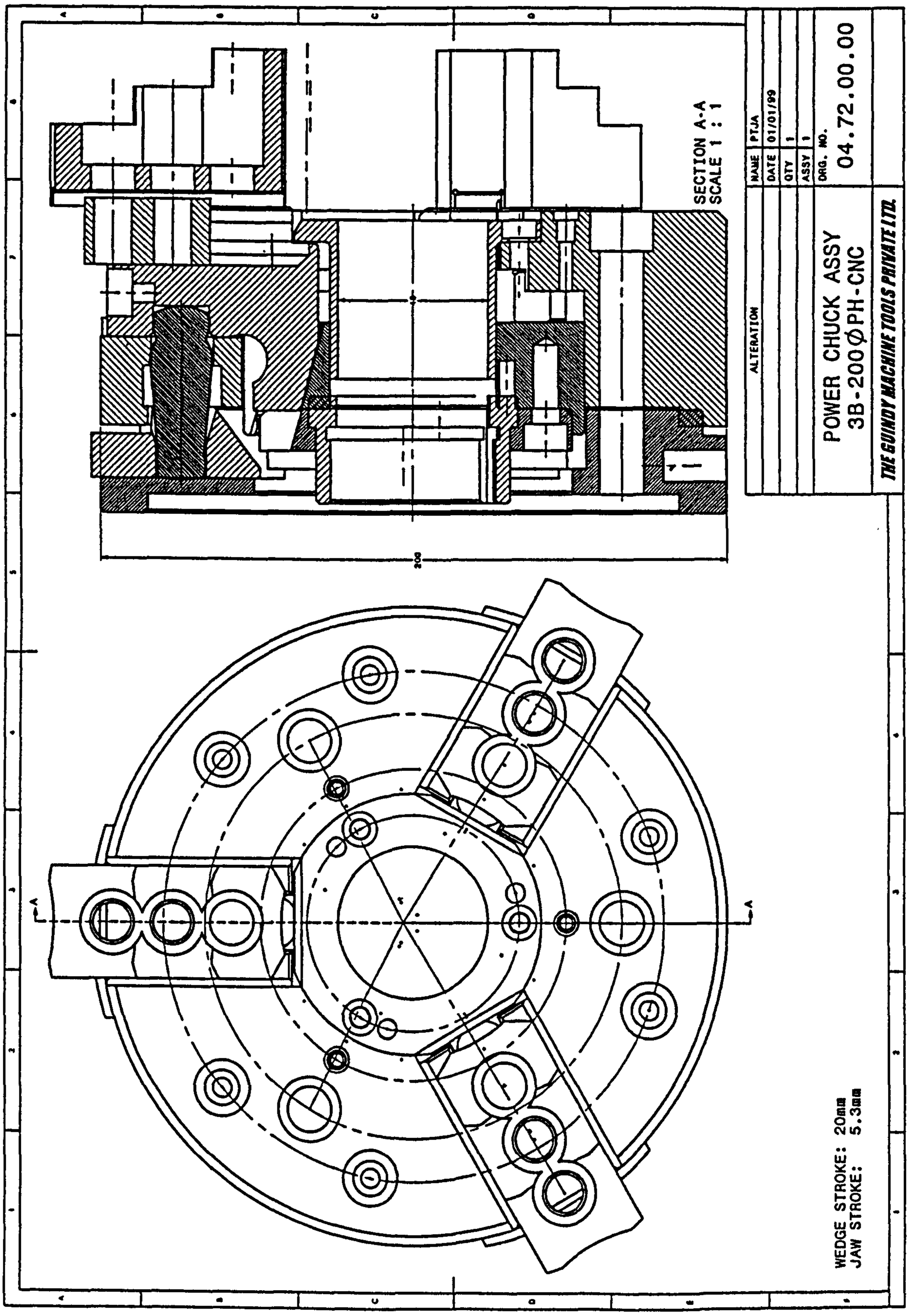


Figure AII.1.26 – GMT Chuck Assembly – Pro/ENGINEER Drawing

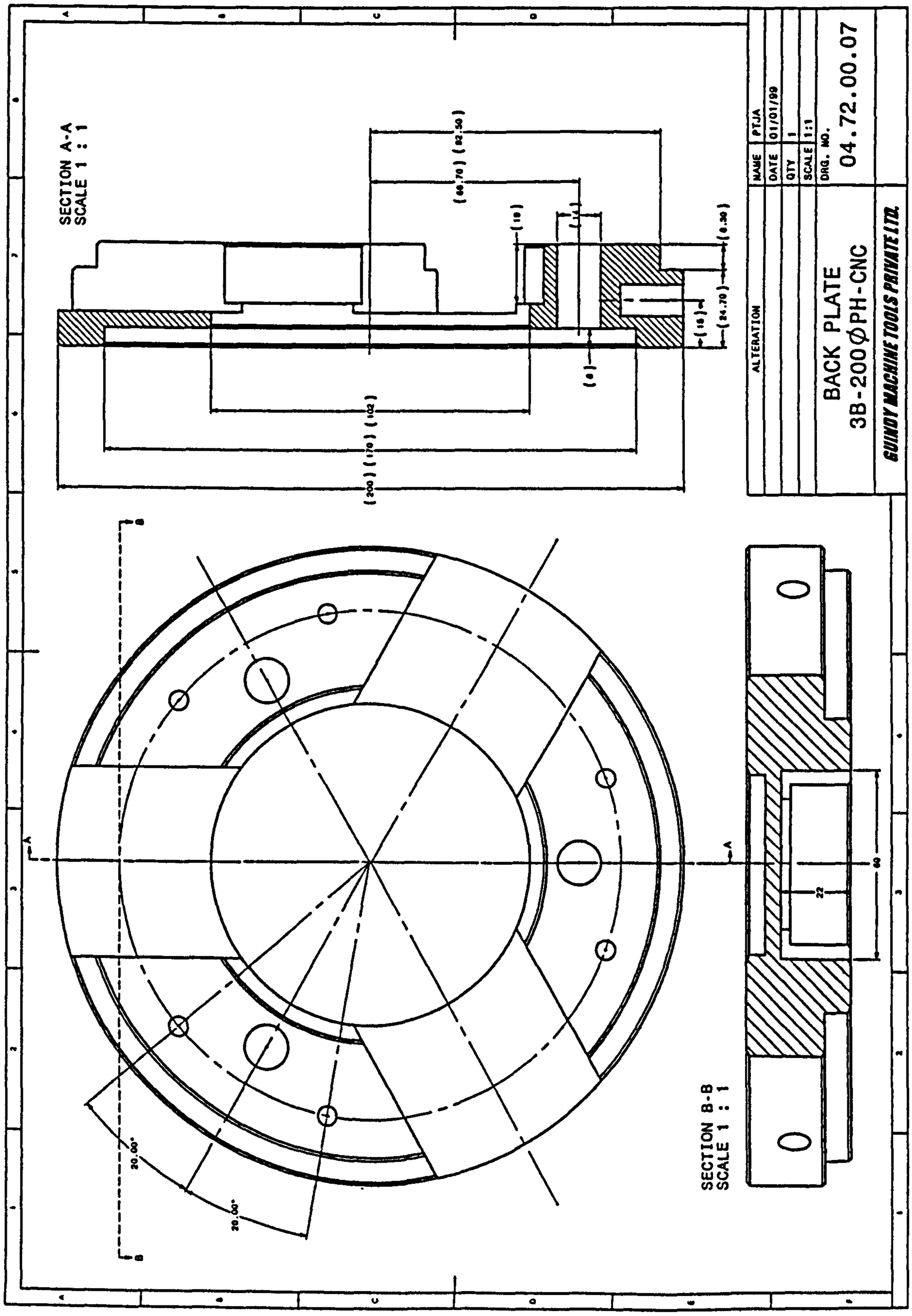


Figure AII.1.27 – GMT Back Plate – Pro/ENGINEER Drawing

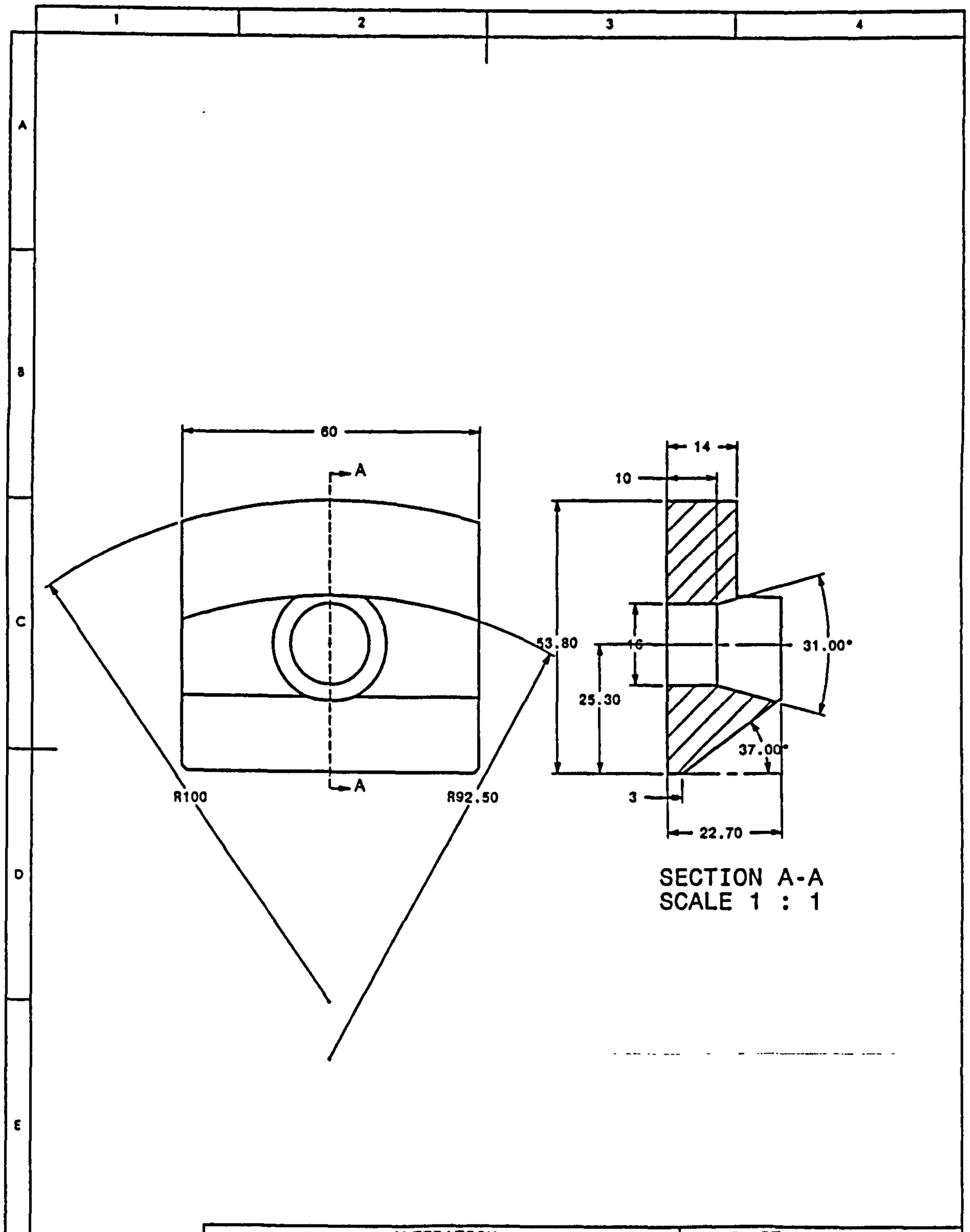


Figure AII.1.28 – GMT Balancing Weight – Pro/ENGINEER Drawing

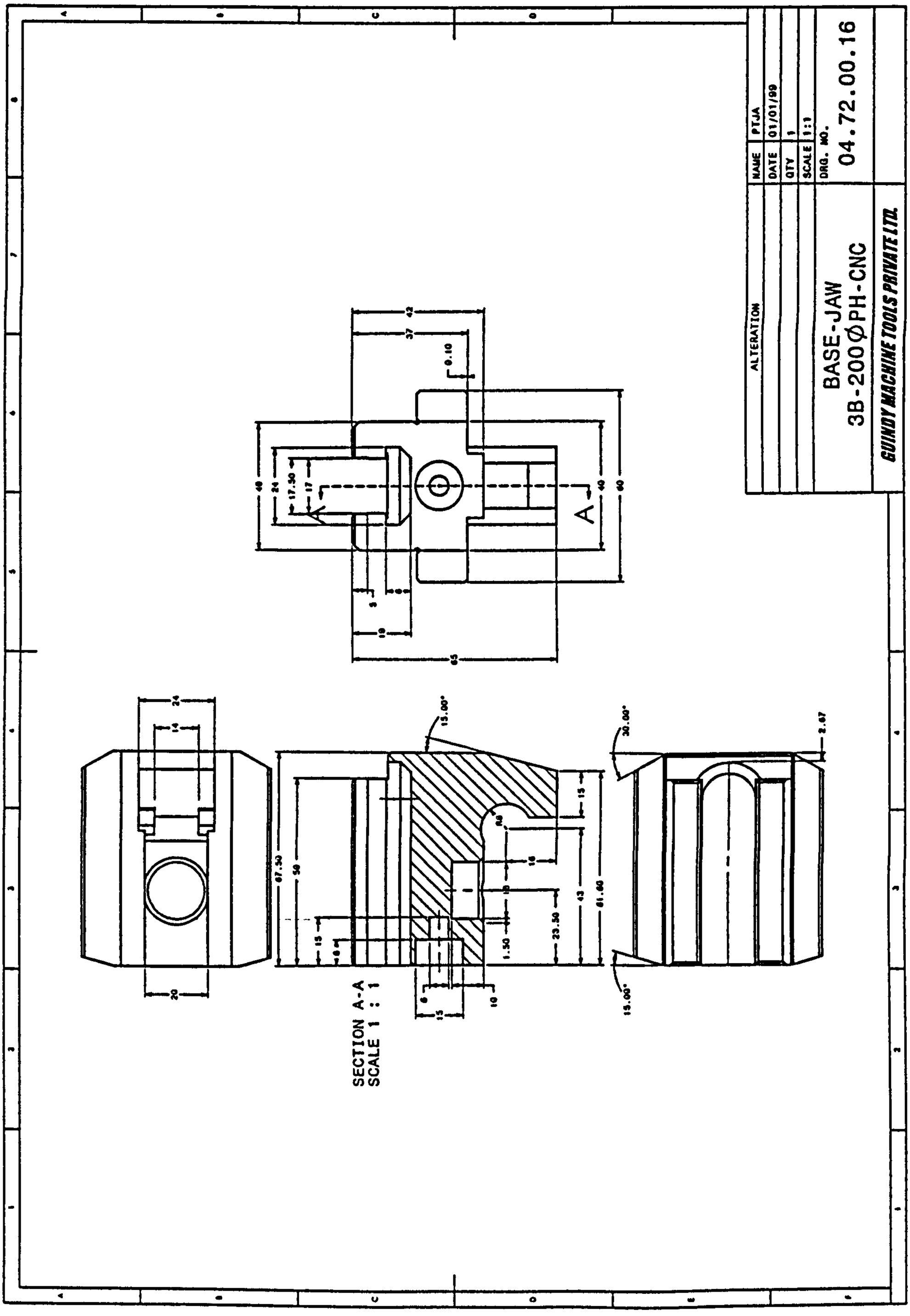


Figure AII.1.29 – GMT Base Jaw – Pro/ENGINEER Drawing

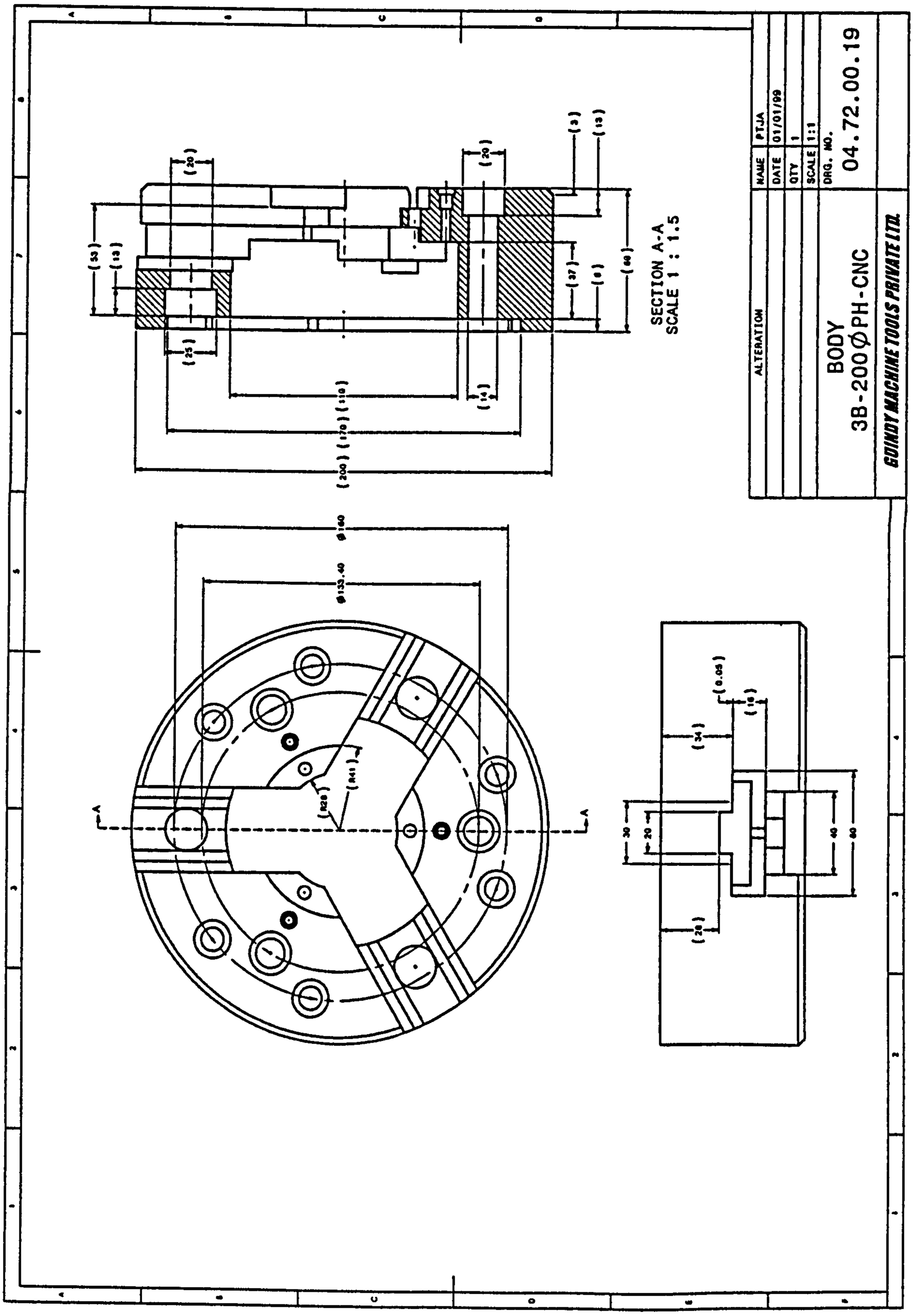
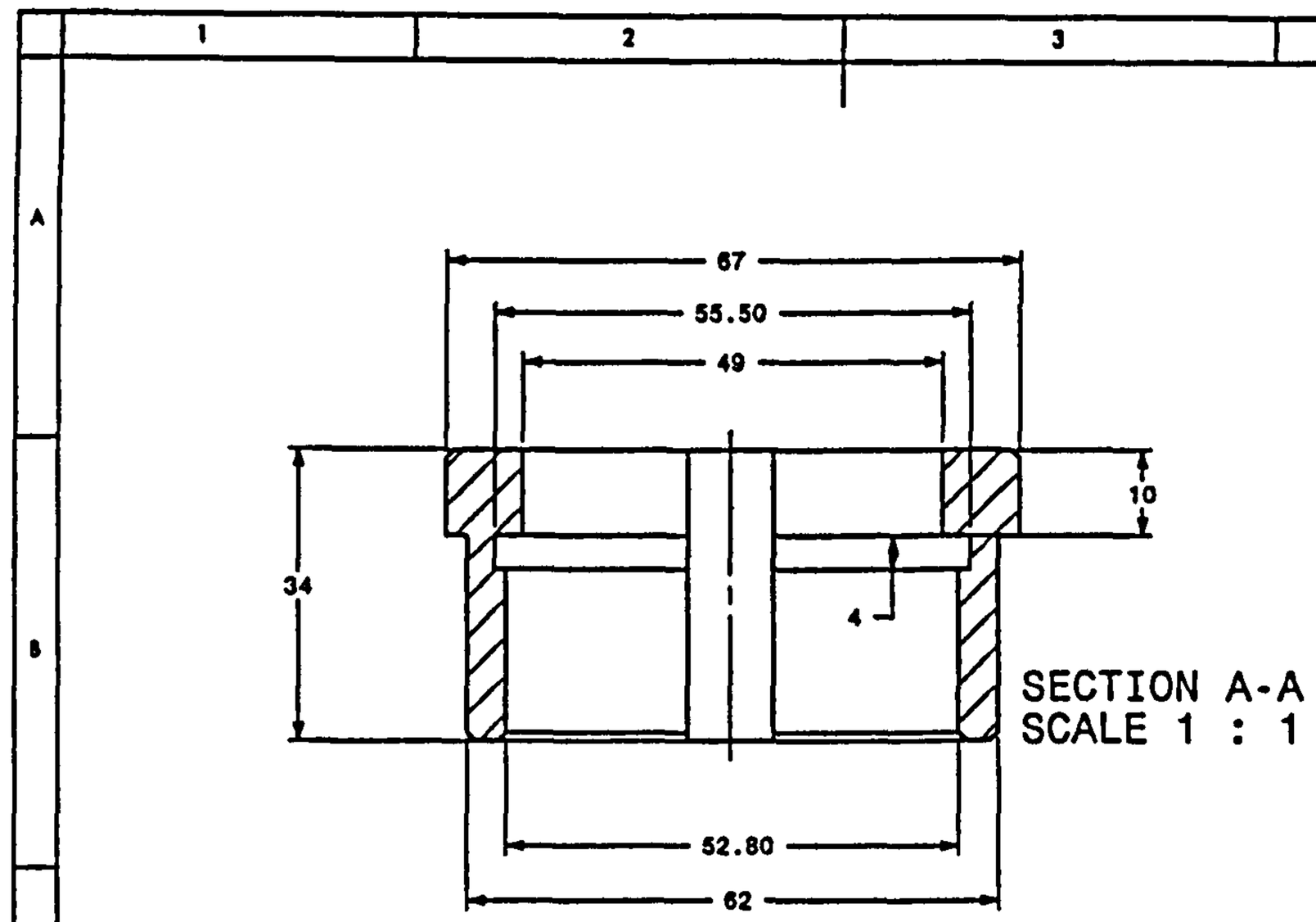
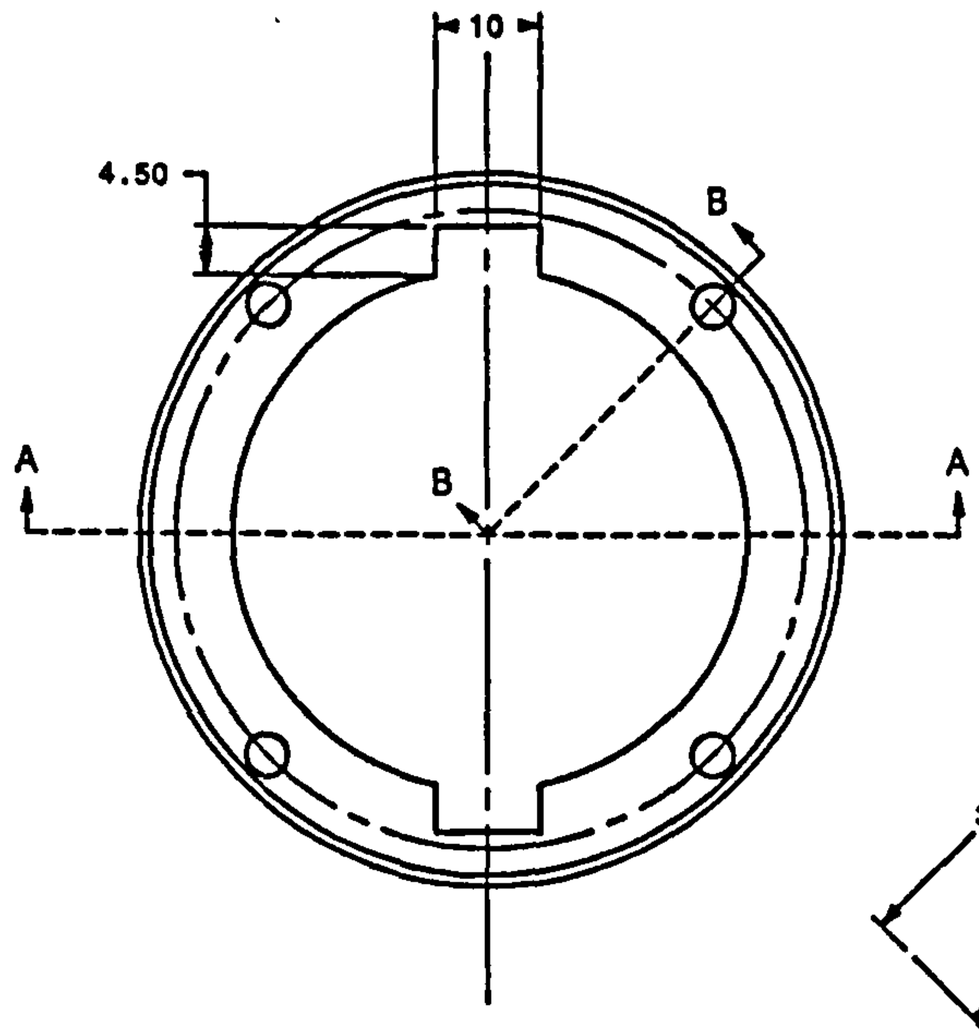
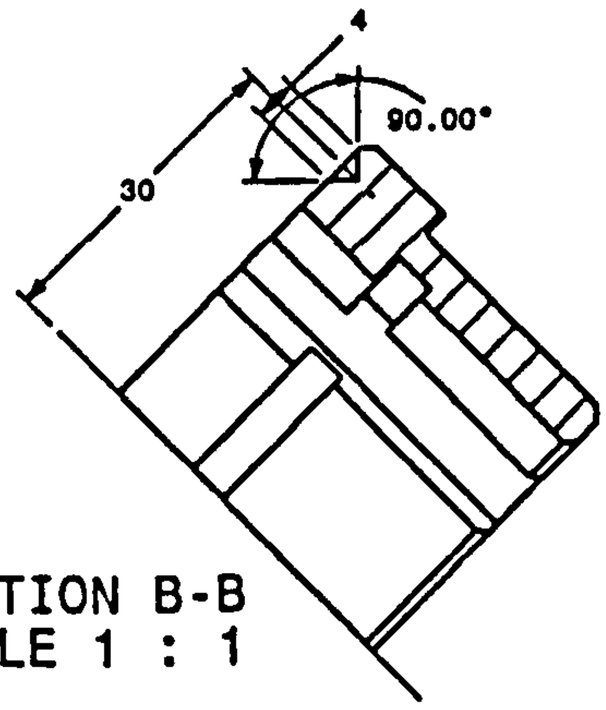


Figure AII.1.30 – GMT Body – Pro/ENGINEER Drawing



SECTION A-A
SCALE 1 : 1

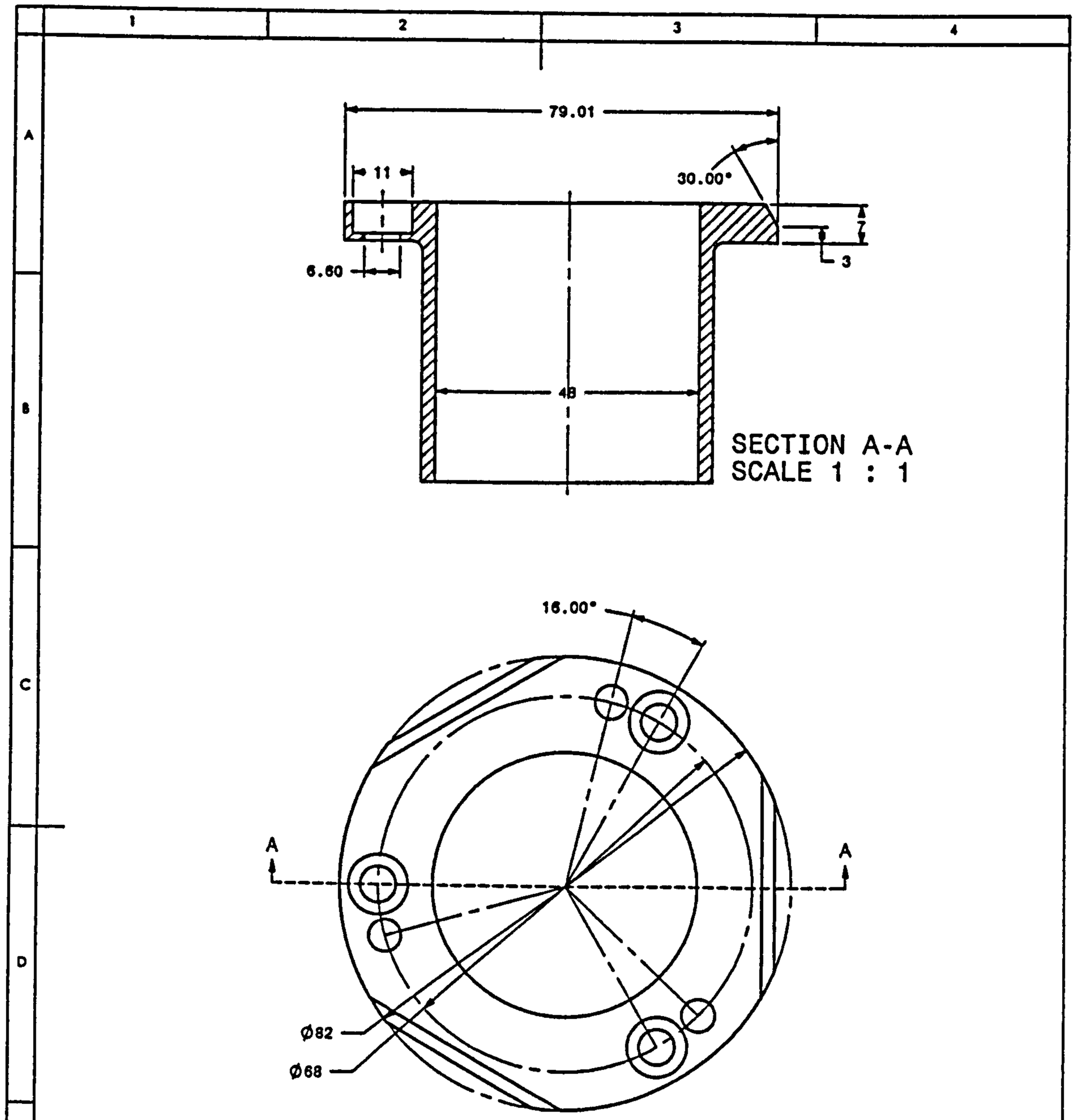




SECTION B-B
SCALE 1 : 1

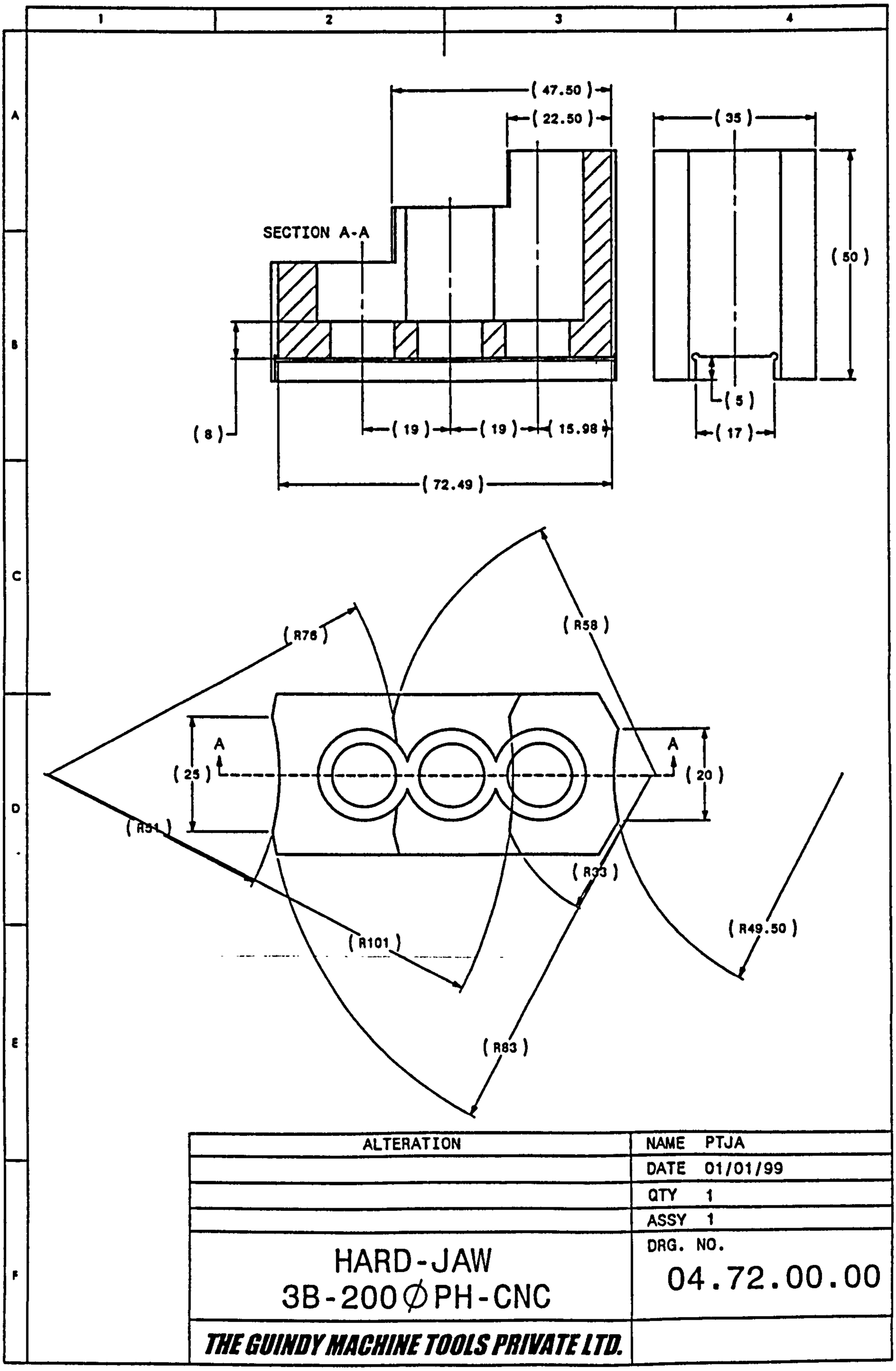
ALTERATION	NAME PTJA
	DATE 01/01/99
	QTY 1
	SCALE 1:1
COLLAR 3B-200∅PH-CNC	DRG. NO. 04.72.00.20
THE GUINDY MACHINE TOOLS PRIVATE LTD.	

Figure AII.1.31 – GMT Collar – Pro/ENGINEER Drawing



ALTERATION	NAME PTJA
	DATE 01/01/99
	QTY 1
	SCALE 1:1
COVER 3B-200∅PH-CNC	DRG. NO. 04.72.00.04
THE GUINDY MACHINE TOOLS PRIVATE LTD.	

Figure AII.1.32 – GMT Cover – Pro/ENGINEER Drawing



ALTERATION	NAME PTJA
	DATE 01/01/99
	QTY 1
	ASSY 1
HARD-JAW 3B-200 Ø PH-CNC	DRG. NO. 04.72.00.00
THE GUINDY MACHINE TOOLS PRIVATE LTD.	

Figure AII.1.33 – GMT Hard Jaw – Pro/ENGINEER Drawing

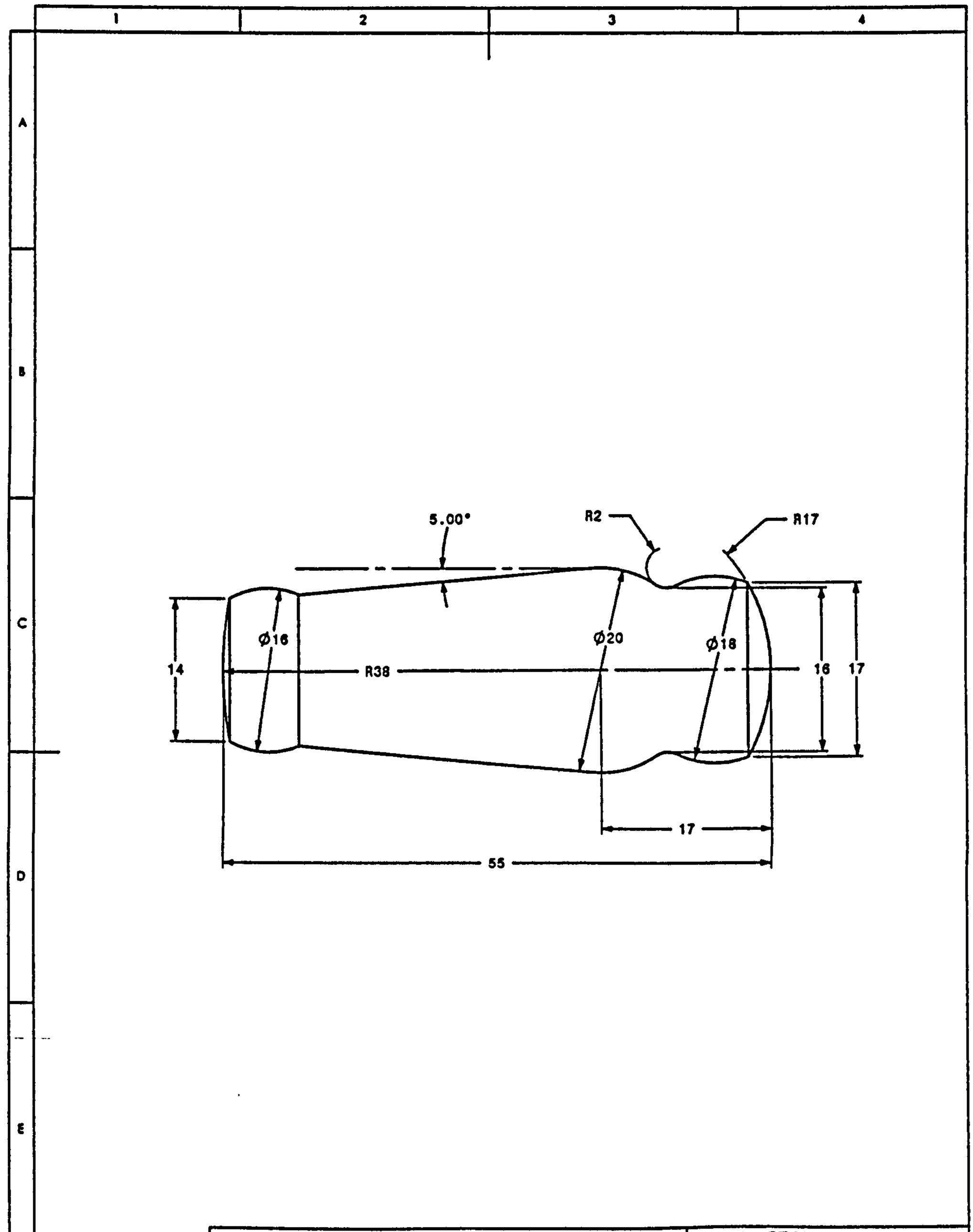


Figure AII.1.34 – GMT Lever – Pro/ENGINEER Drawing

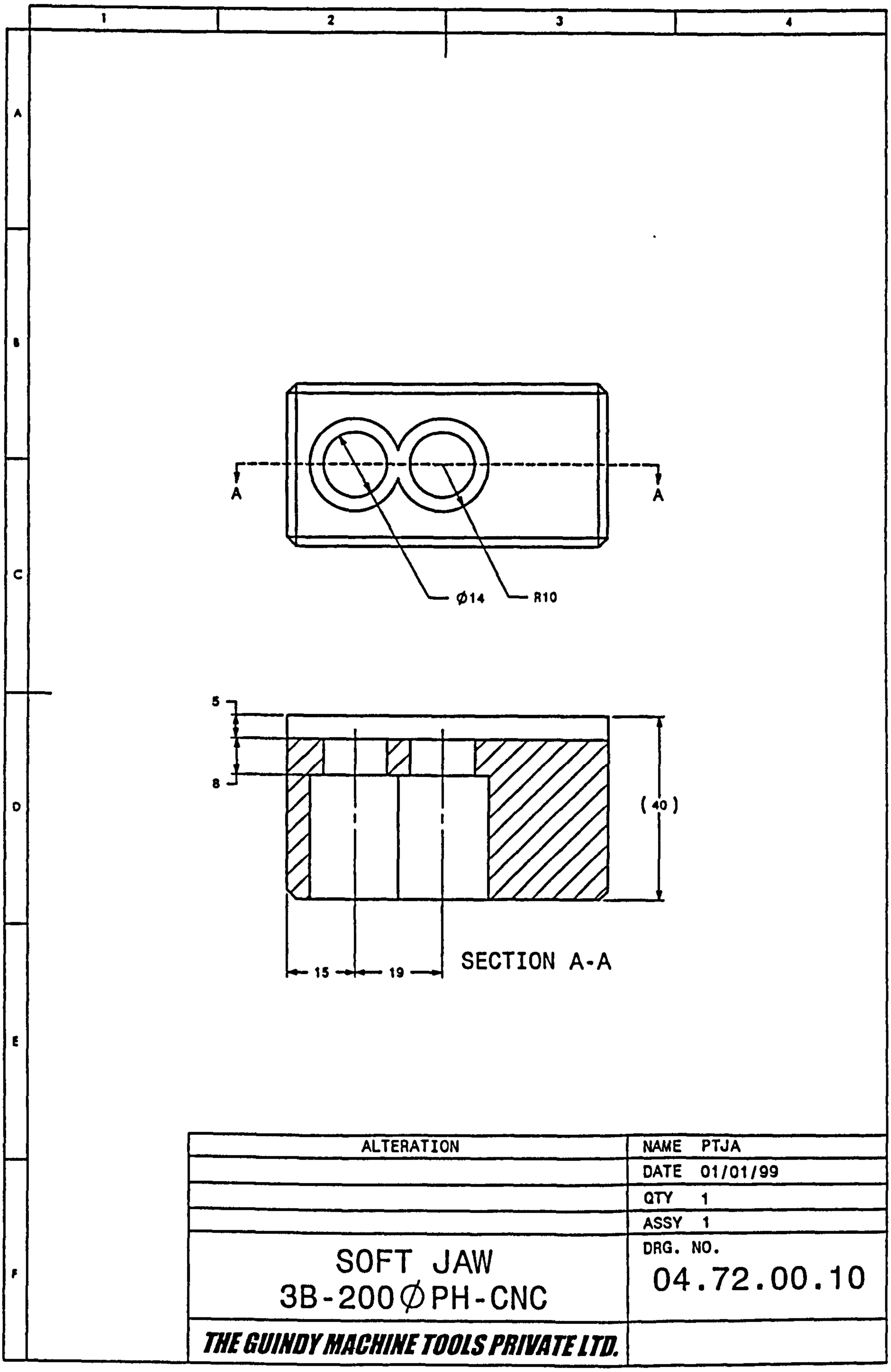
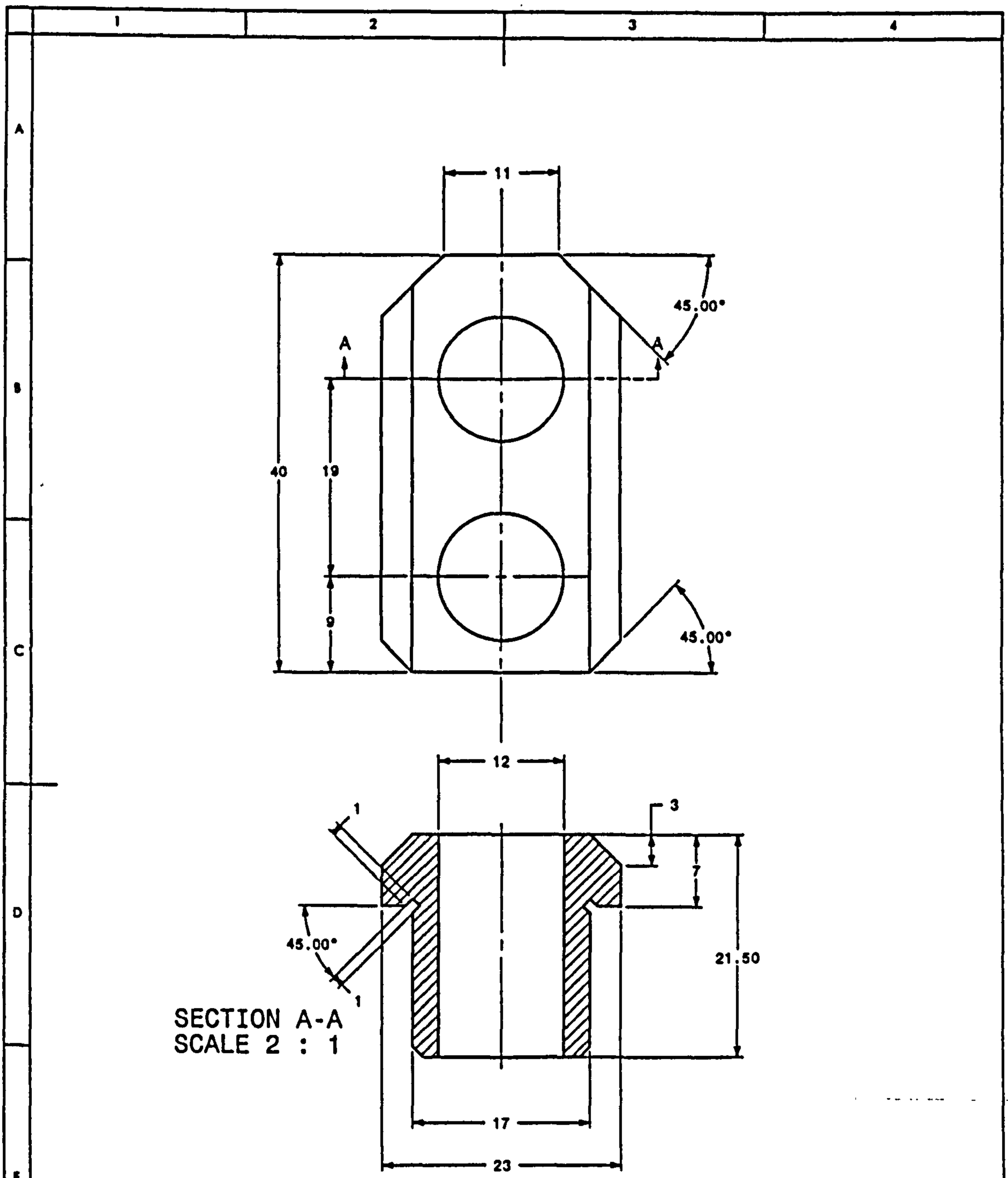


Figure AII.1.35 – GMT Soft Jaw – Pro/ENGINEER Drawing



ALTERATION	NAME PTJA
	DATE 01/01/99
	QTY 1
	SCALE 1:1
'T' NUT 3B-200∅PH-CNC	DRG. NO. 04.72.00.21
THE GUINDY MACHINE TOOLS PRIVATE LTD.	

Figure AII.1.36 – GMT 'T' Nut – Pro/ENGINEER Drawing

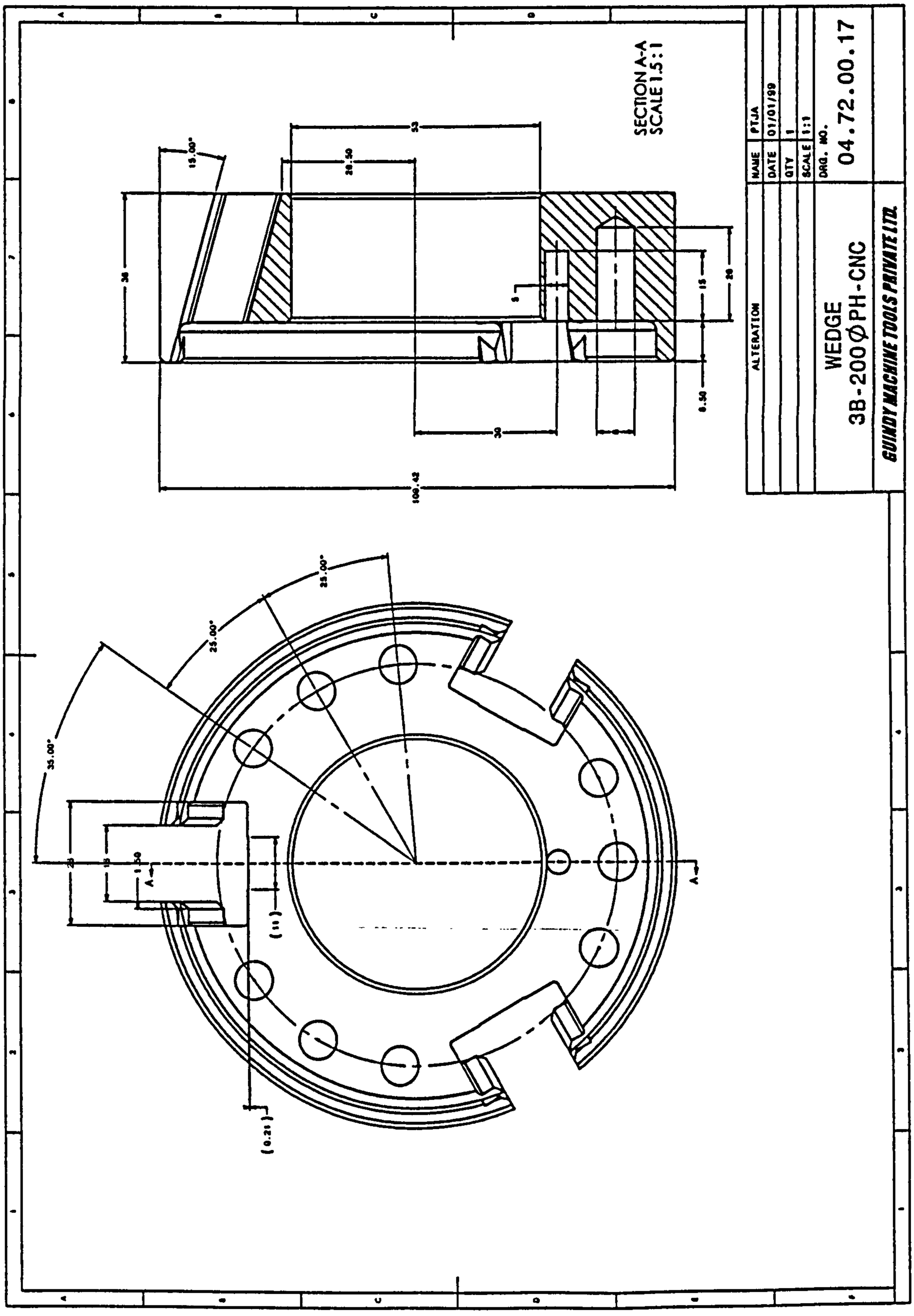
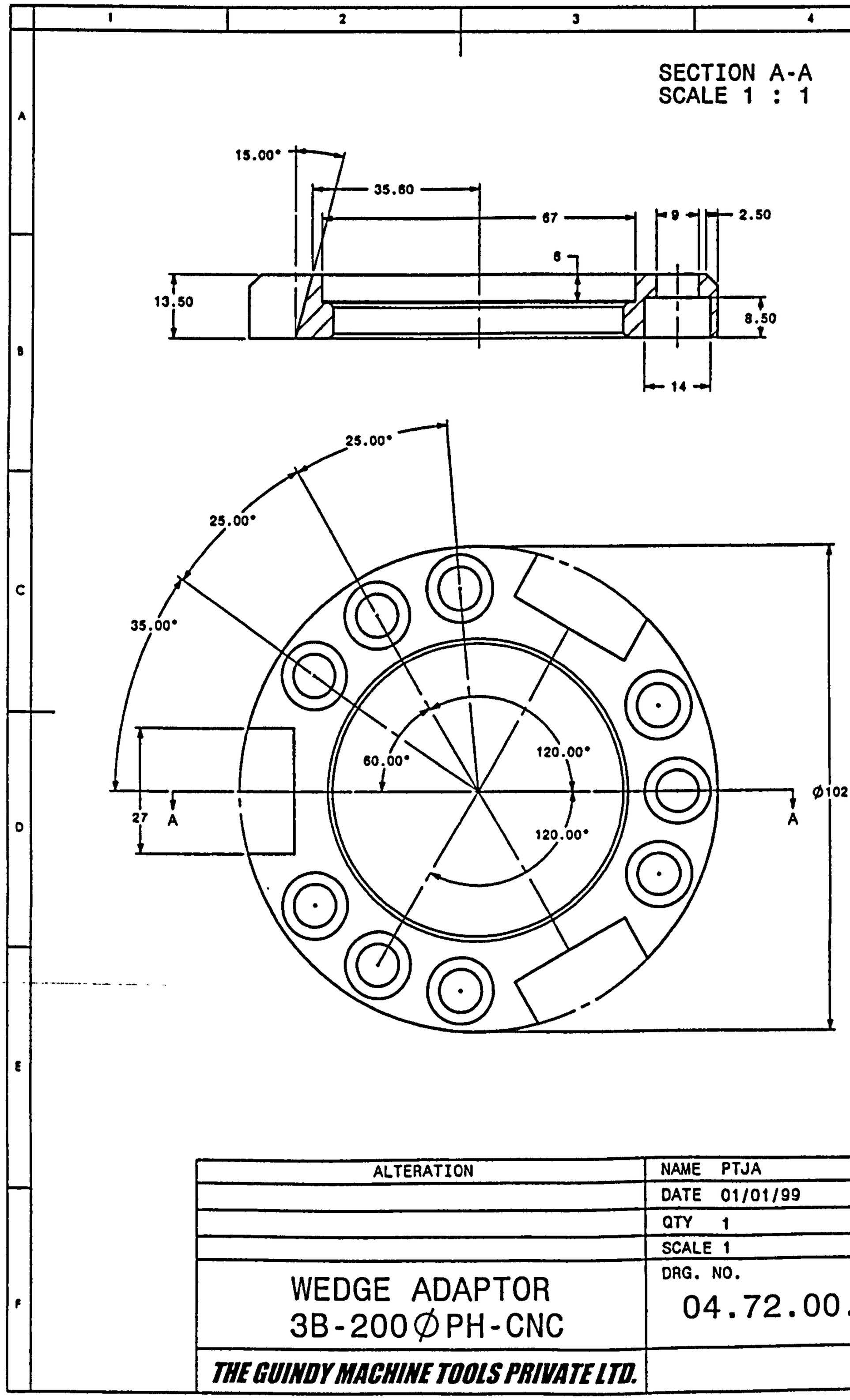
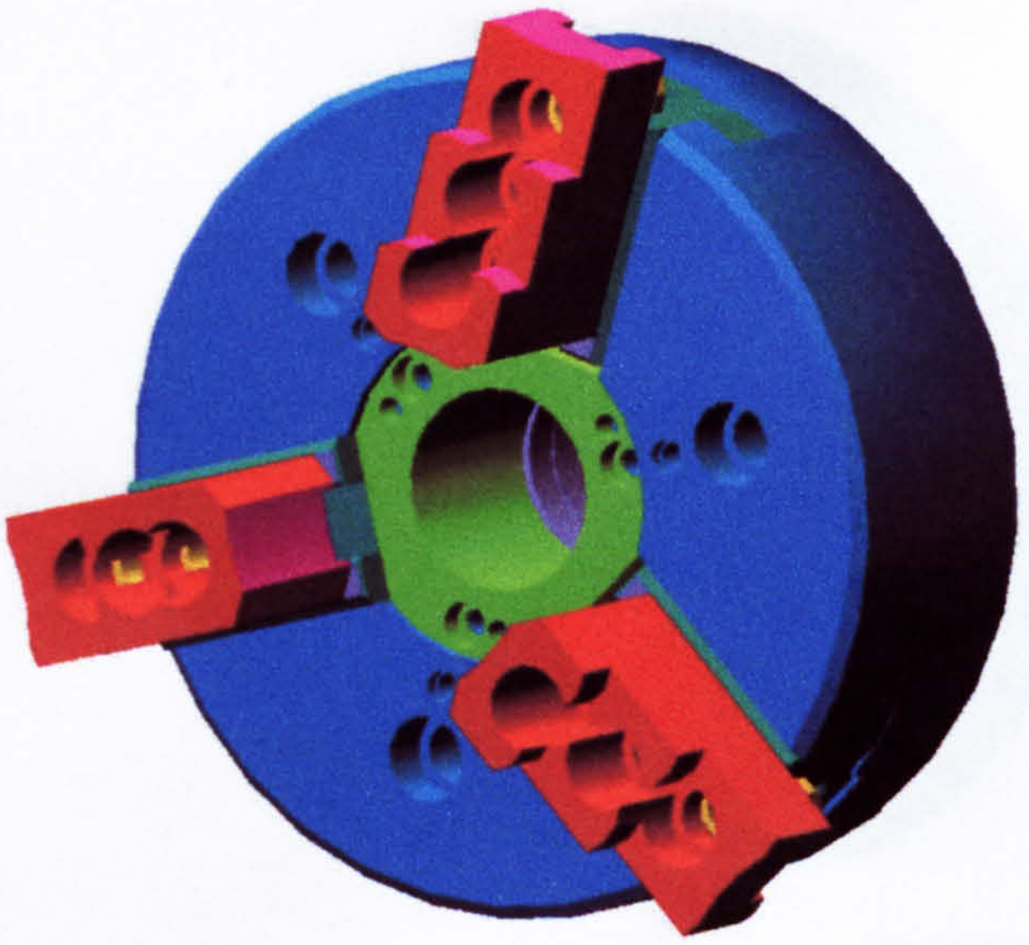


Figure AII.1.37 – GMT Wedge – Pro/ENGINEER Drawing



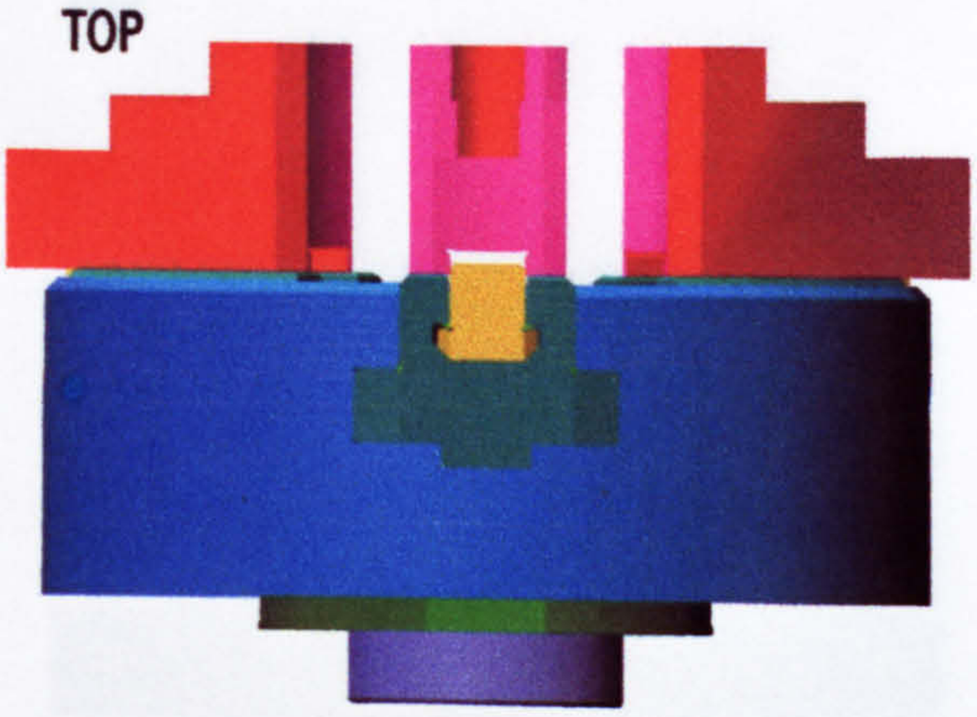
ALTERATION	NAME PTJA
	DATE 01/01/99
	QTY 1
	SCALE 1
WEDGE ADAPTOR 3B-200Ø PH-CNC	DRG. NO. 04.72.00.18
THE GUINDY MACHINE TOOLS PRIVATE LTD.	

Figure AII.1.38 – GMT Wedge Adaptor – Pro/ENGINEER Drawing

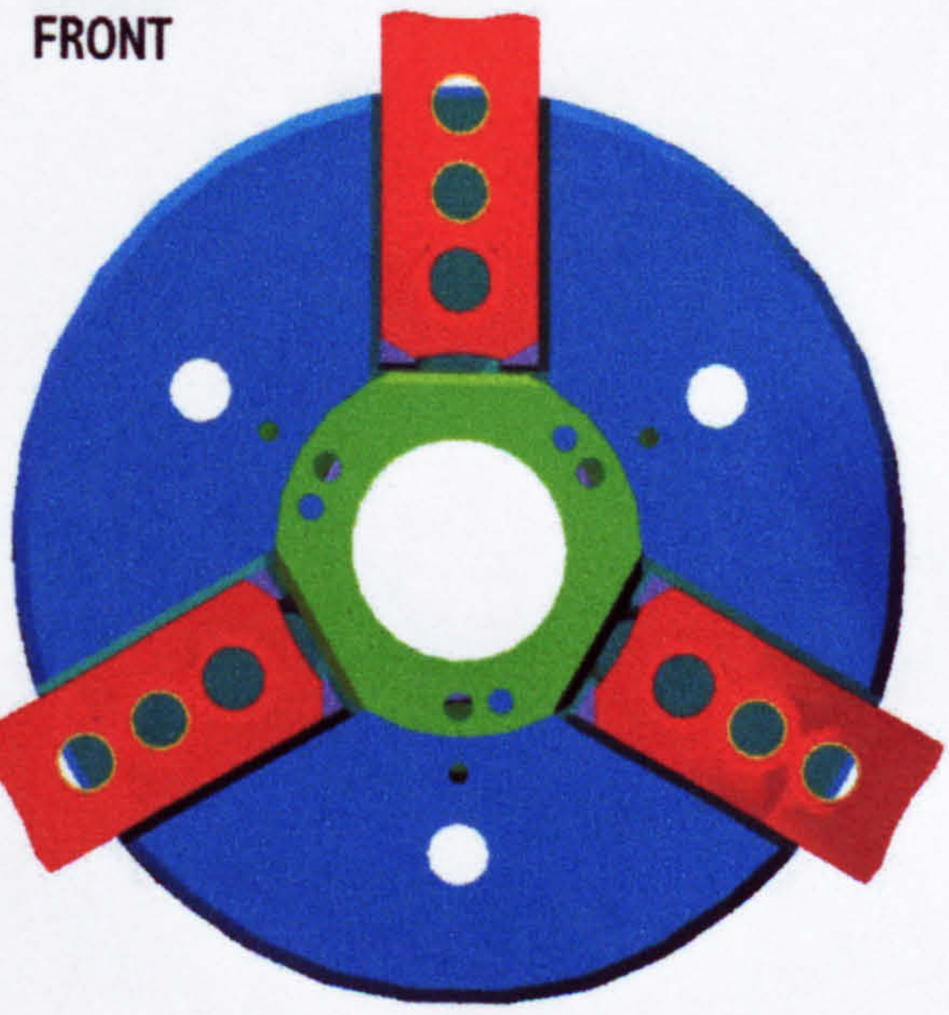


3D-VIEW

TOP



FRONT



SIDE

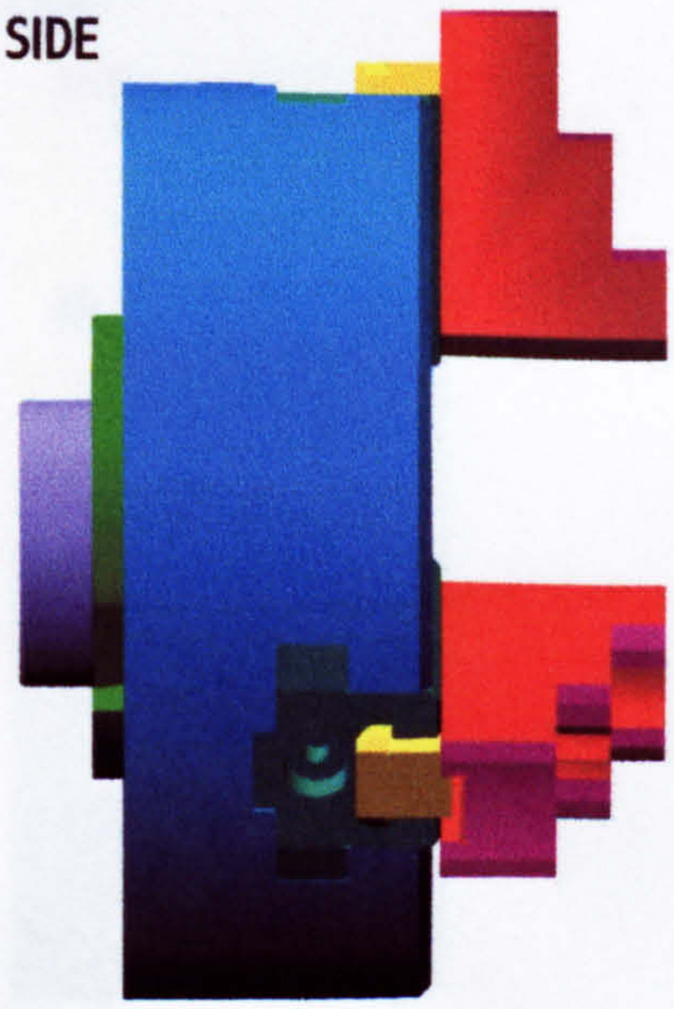
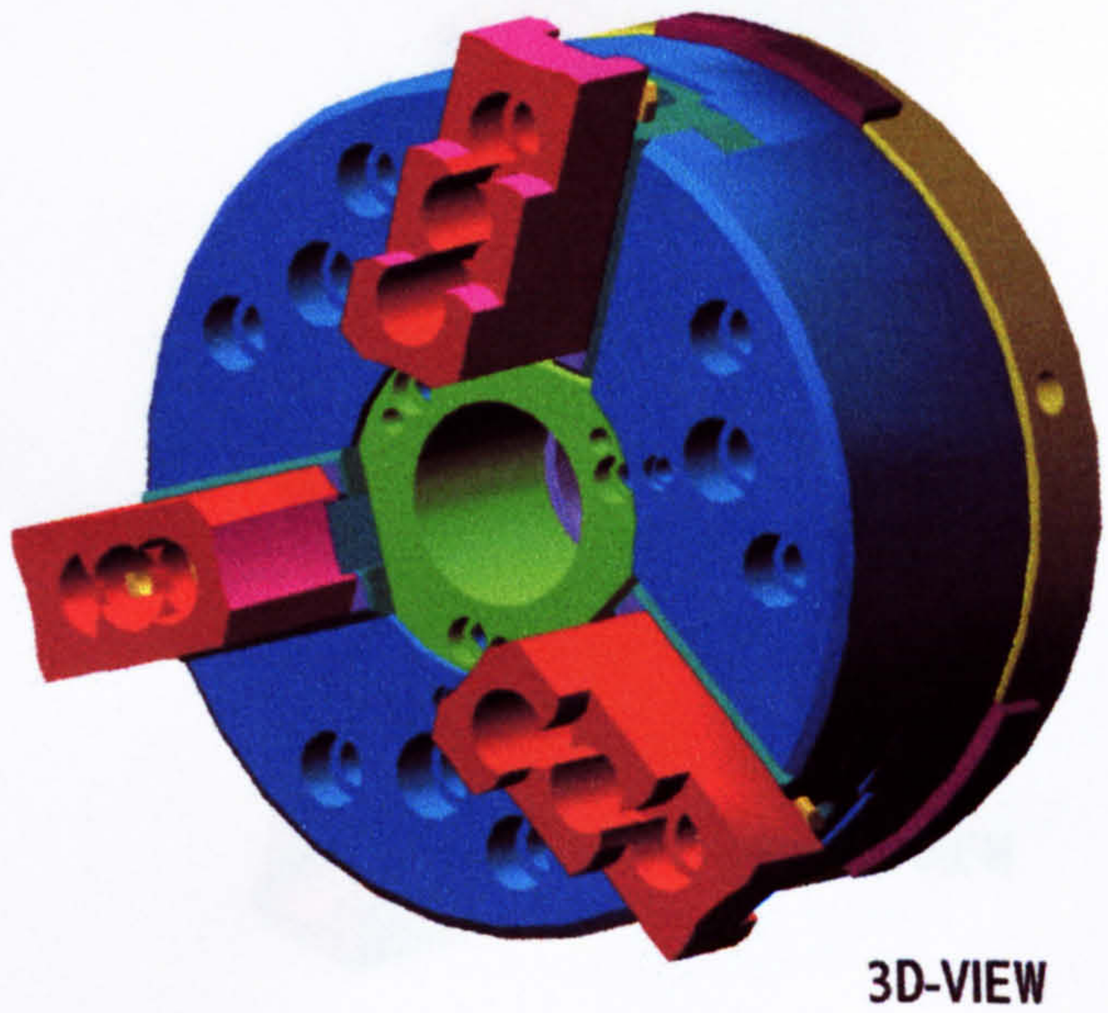


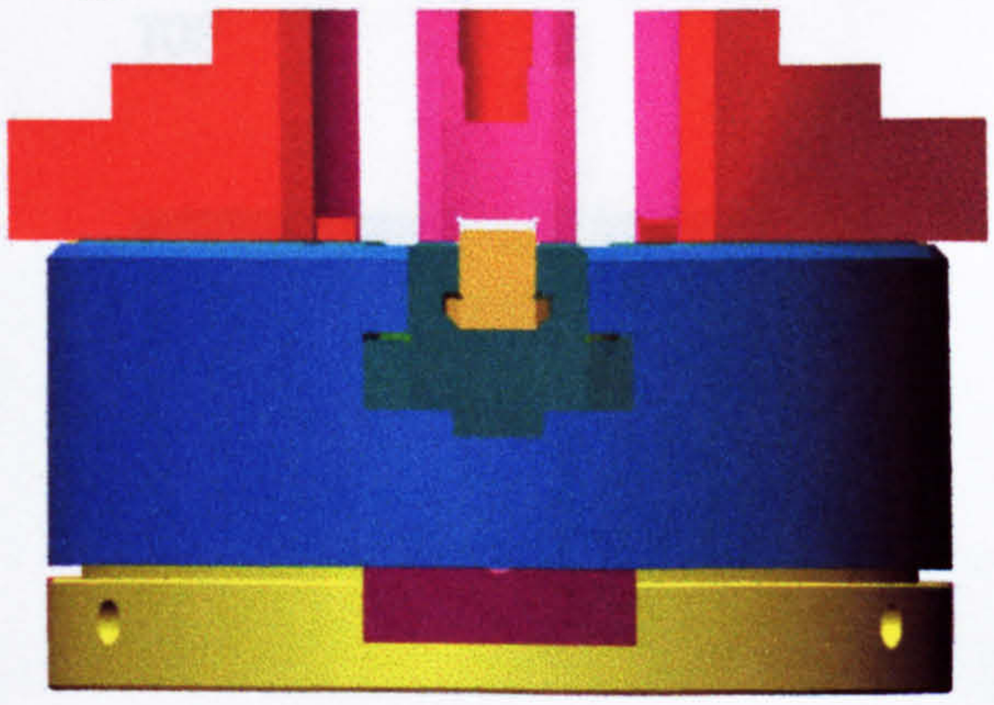
Figure AII.1.39 – Assembly Views for the 3B200-PHCNC Chuck

CROSS SECTION

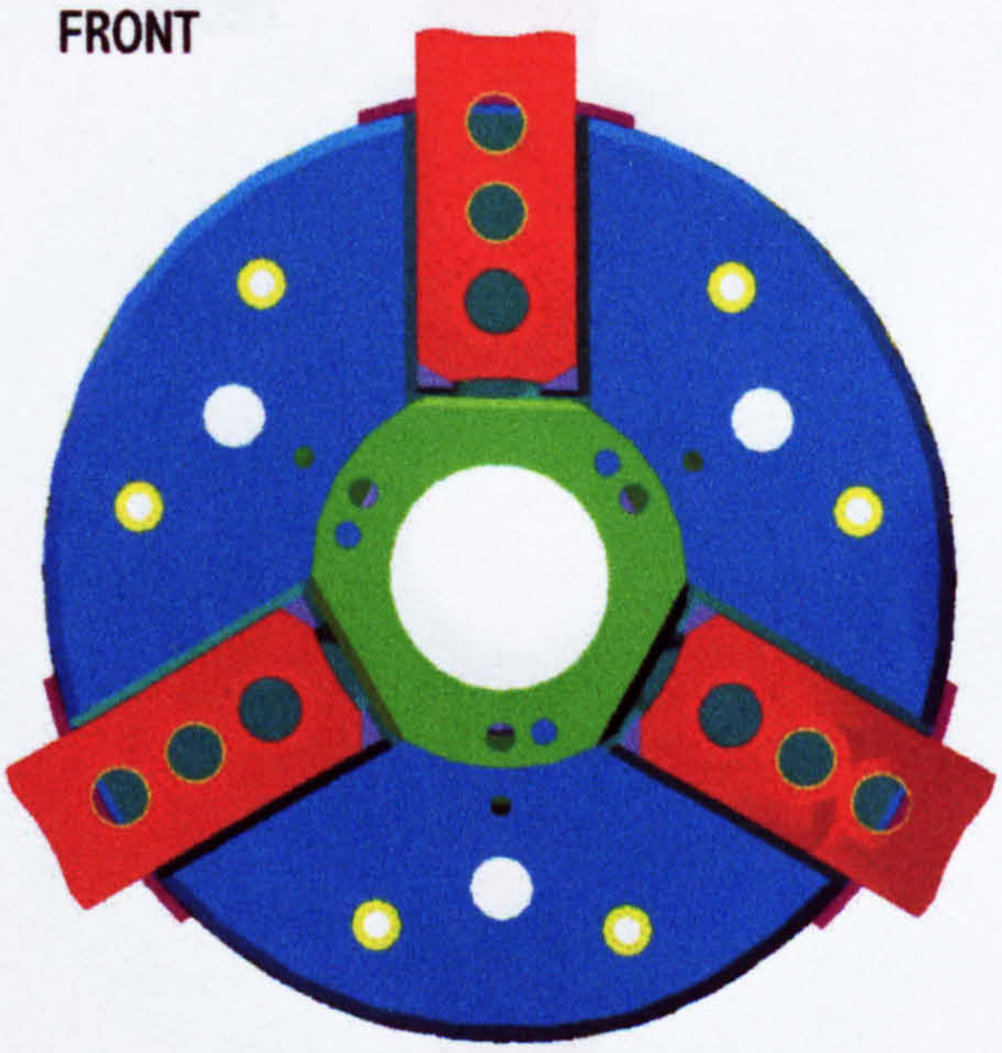


3D-VIEW

TOP



FRONT



SIDE

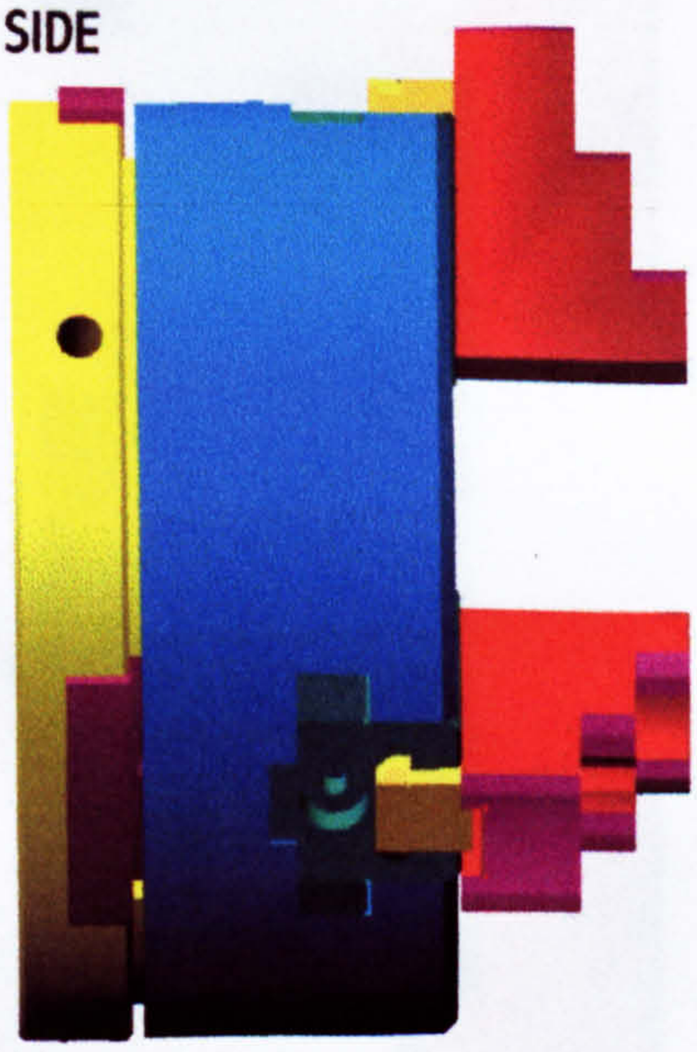
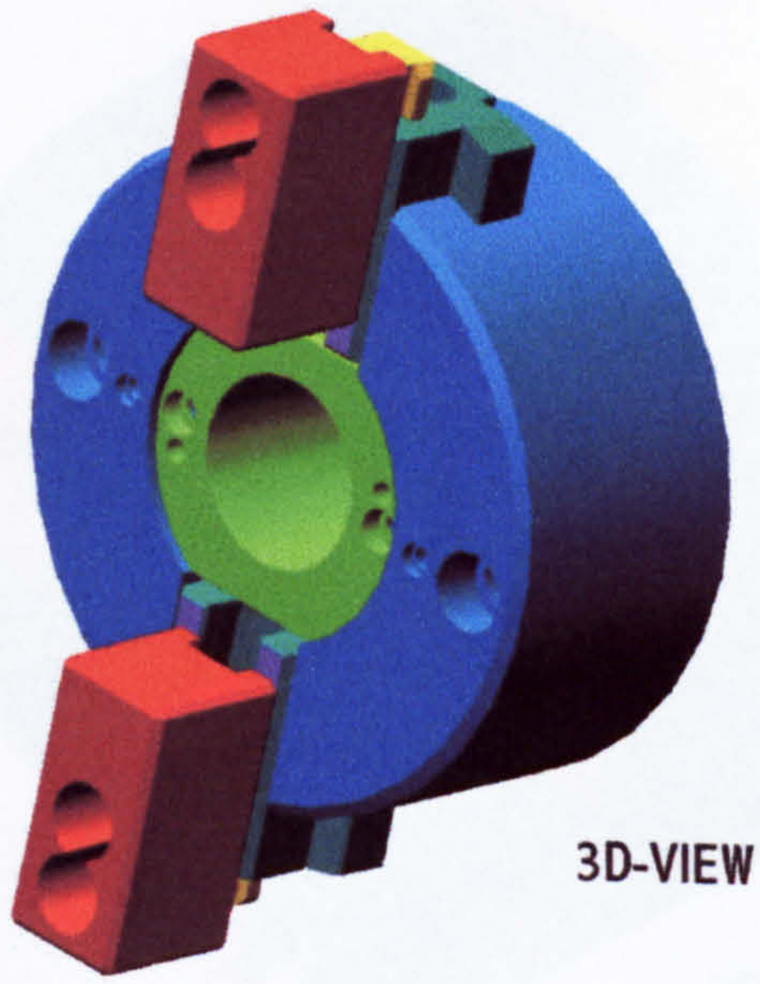
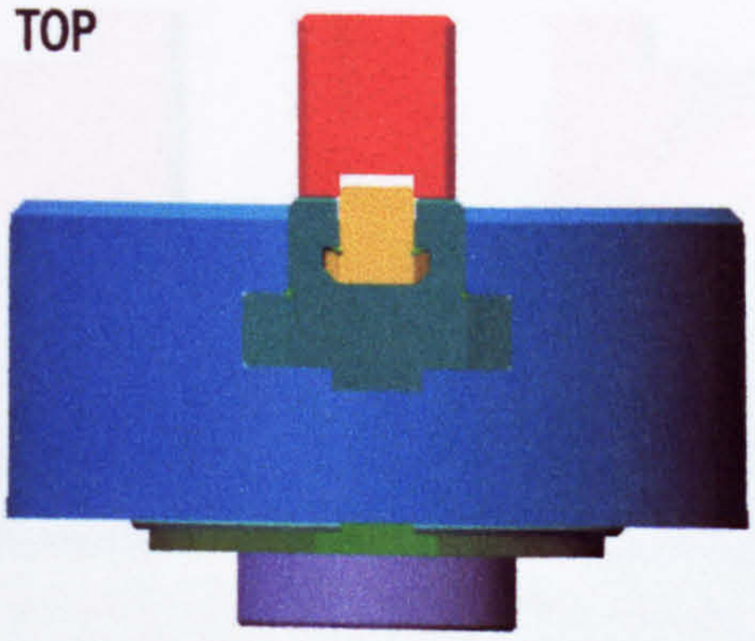


Figure AII.1.40 – Assembly Views for the 3B200-PHNC Chuck

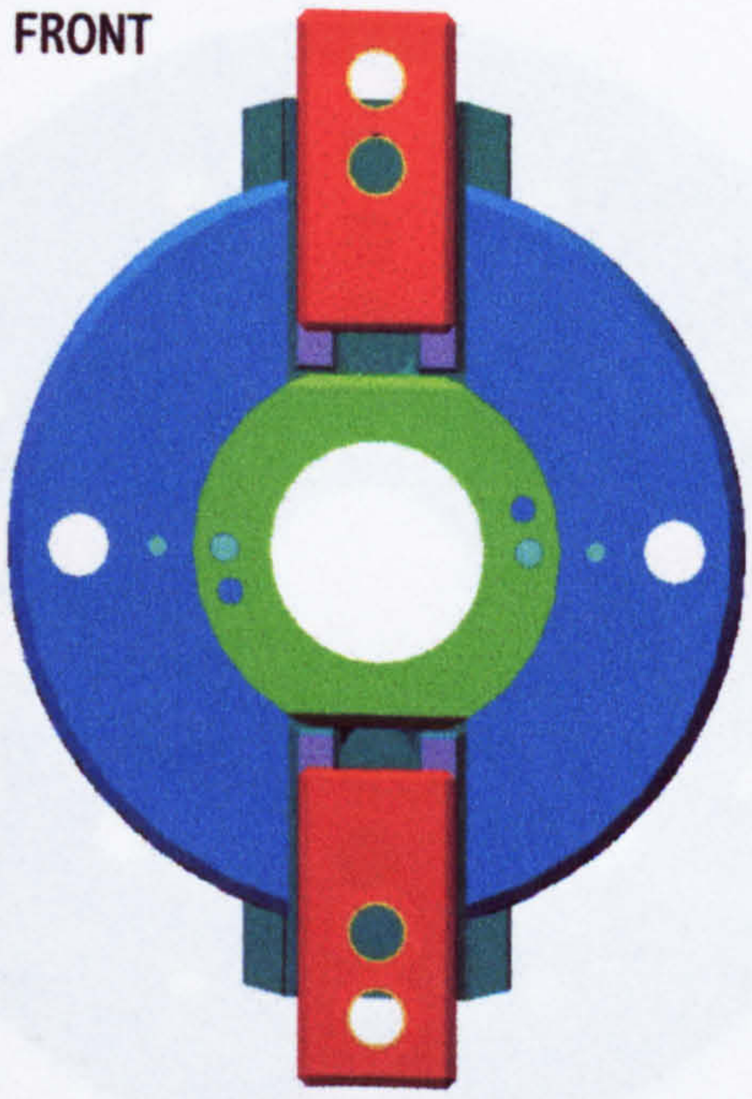


3D-VIEW

TOP



FRONT



SIDE

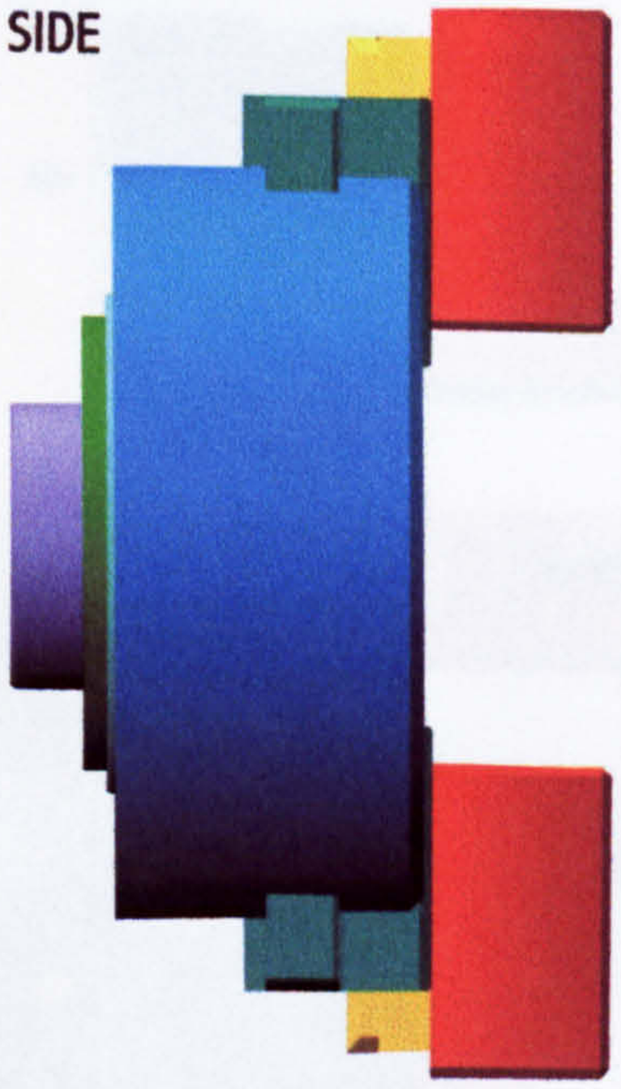


Figure AII.1.41 – Assembly Views for the 2B165-PHCNC Chuck

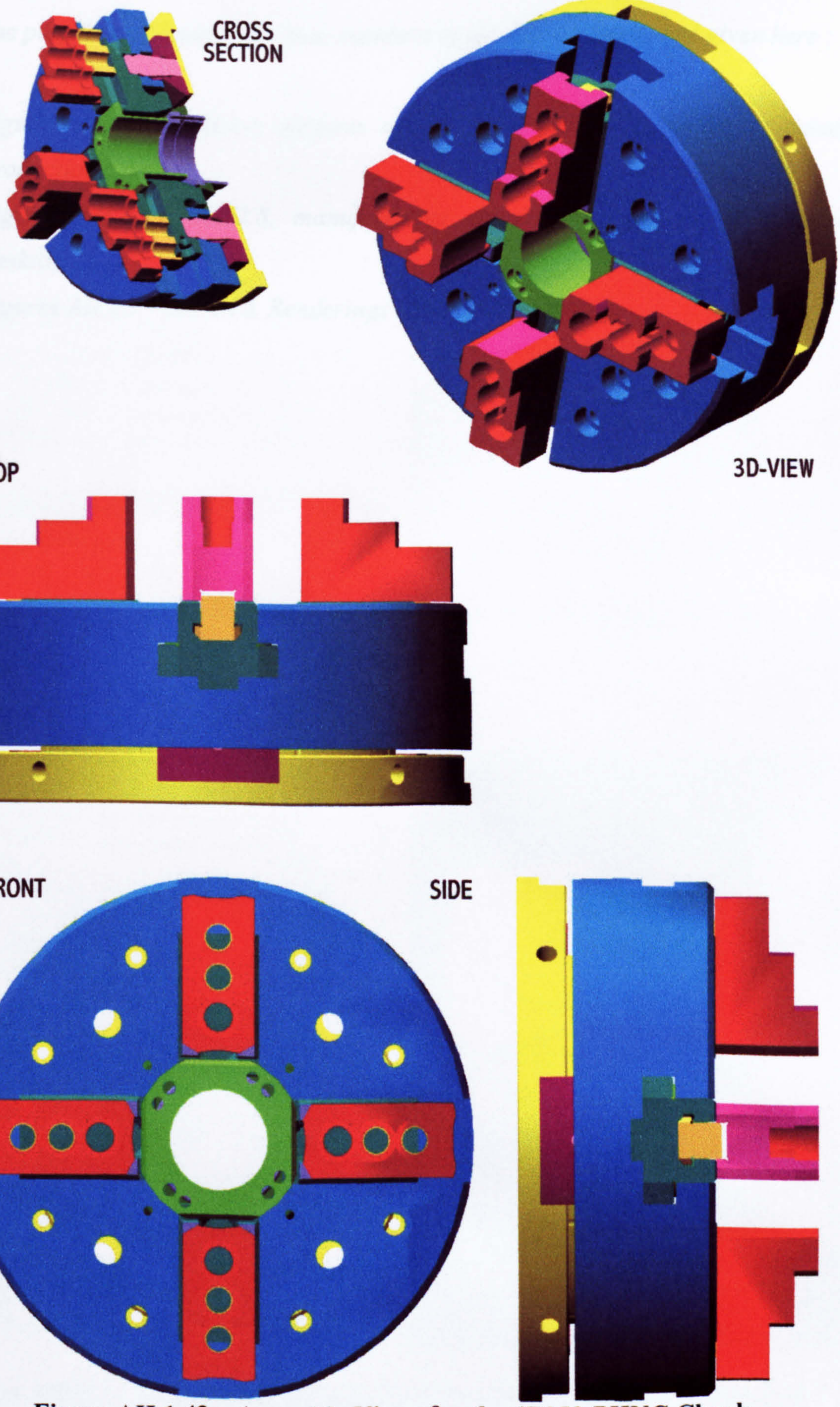


Figure AII.1.42 – Assembly Views for the 4B250-PHNC Chuck

AII.2 Lucas Varity Drive End Shield Casting

For the purposes of illustration, four members of the casting family are given here.:

1) *Figures AII.2.1 - AII.2.4, variants of the Master Model representing actual Production Castings,*
2) *Figures AII.2.5 - AII.2.8, manufacturing drawings created in Mechanical Desktop,*
3) *Figures AII.2.9 – AII.2.10, Renderings of the Lucas Casting.*

Parameter	Value
Lug Separation	75mm
Lug Diameter	28mm
Lug Angle	90°
Fillet 2	14mm
Wedge Diameter	102mm
Wedge Depth	24mm
Base Depth	12.7mm
Cylinder Height	76.2
Cylinder Bottom Diamet	114.3mm
Top Boss Height	42mm
Top Boss Diameter	64mm
Middle Boss Height	5mm
Middle Boss Diameter	96.05mm
Centre Hole Diameter	26.5mm
Bump Angle	30°
Bump Radius	12mm
Key Angle	45°

Non-Persistent Feature	Status
Middle Boss	Resumed

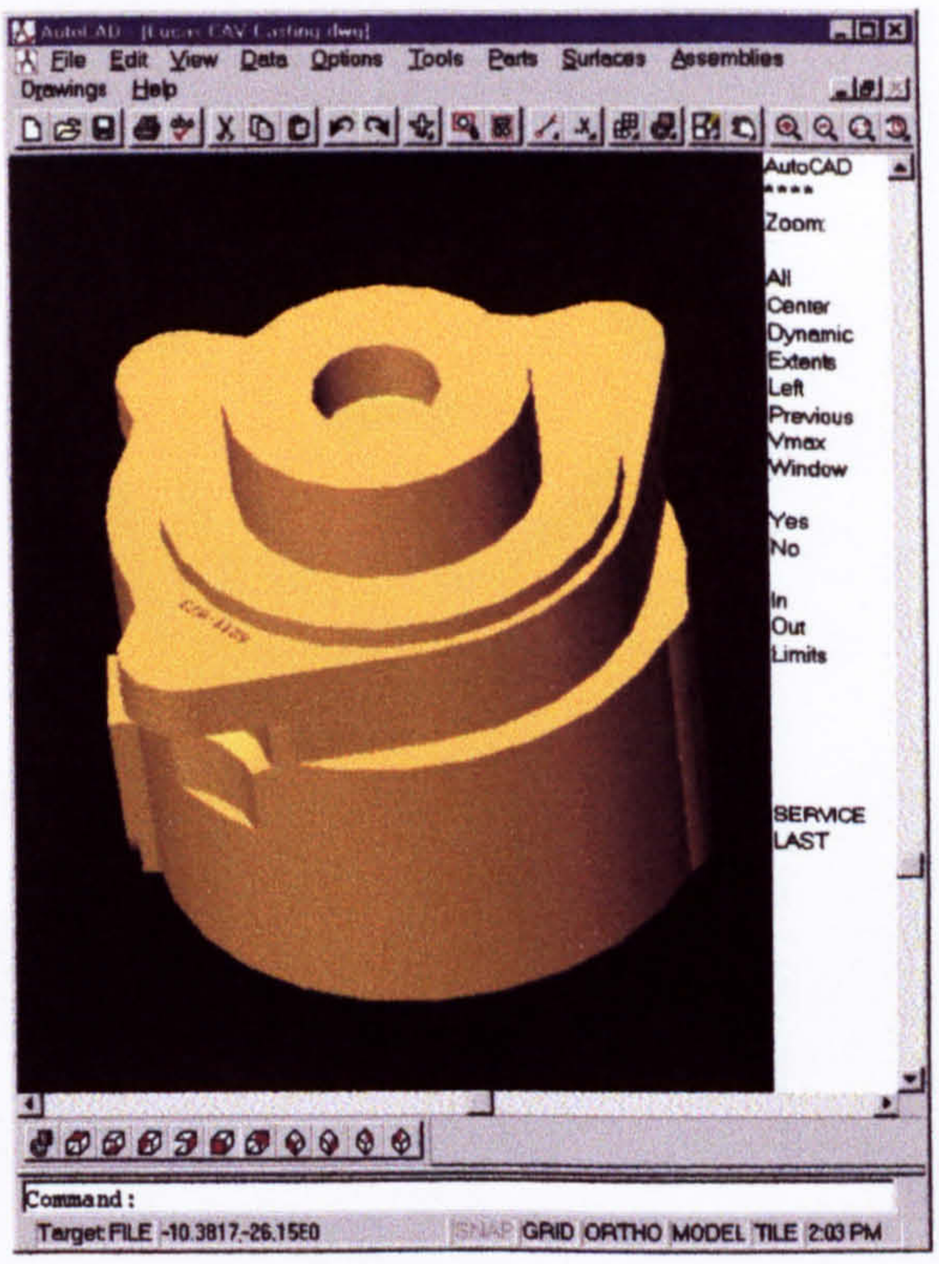


Figure AII.2.1 – Drive-End-Shield No. V6211-673 - CAD Model & Parameters

Parameter	Value
Lug Separation	75mm
Lug Diameter	30mm
Lug Angle	90°
Fillet 2	15mm
Wedge Diameter	102mm
Wedge Depth	44.7mm
Base Depth	12.7mm
Cylinder Height	103.2mm
Cylinder Bottom Diamet	114.3mm
Top Boss Height	17.5mm
Top Boss Diameter	82mm
Middle Boss Height	-
Middle Boss Diameter	-
Centre Hole Diameter	26.5mm
Bump Angle	75°
Bump Radius	12mm
Key Angle	10°

Non-Persistent Feature	Status
Middle Boss	Supp

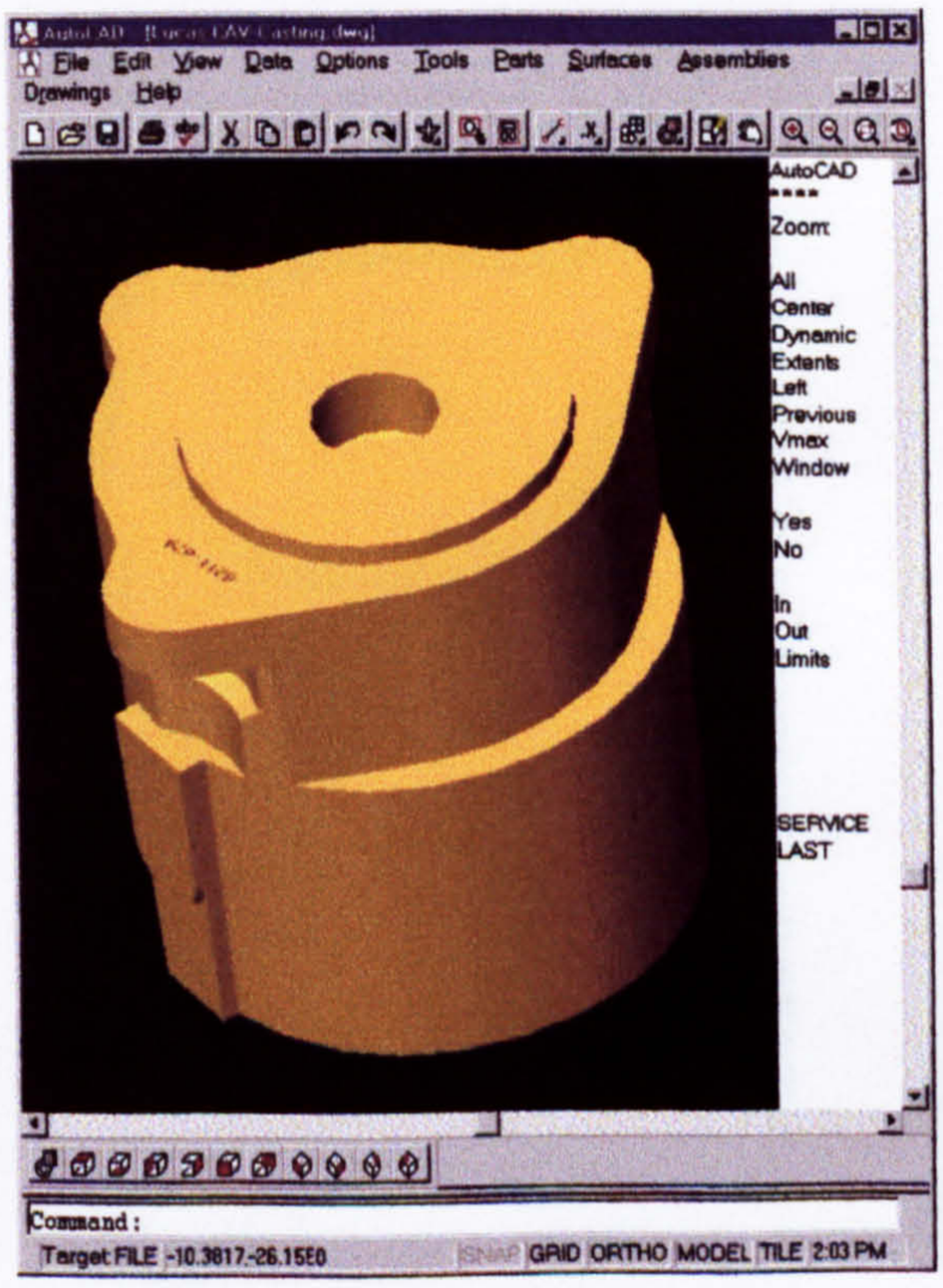


Figure AII.2.2 – Drive-End-Shield No. V6211-679 - CAD Model & Parameters

Parameter	Value
Lug Separation	75mm
Lug Diameter	28mm
Lug Angle	20°
Fillet 2	14mm
Wedge Diameter	102mm
Wedge Depth	42mm
Base Depth	12.7mm
Cylinder Height	94.4mm
Cylinder Bottom Diamet	114.3mm
Top Boss Height	23.8mm
Top Boss Diameter	92.2mm
Middle Boss Height	-
Middle Boss Diameter	-
Centre Hole Diameter	26.5mm
Bump Angle	30°
Bump Radius	12mm
Key Angle	45°

Non-Persistent Feature	Status
Middle Boss	Supp

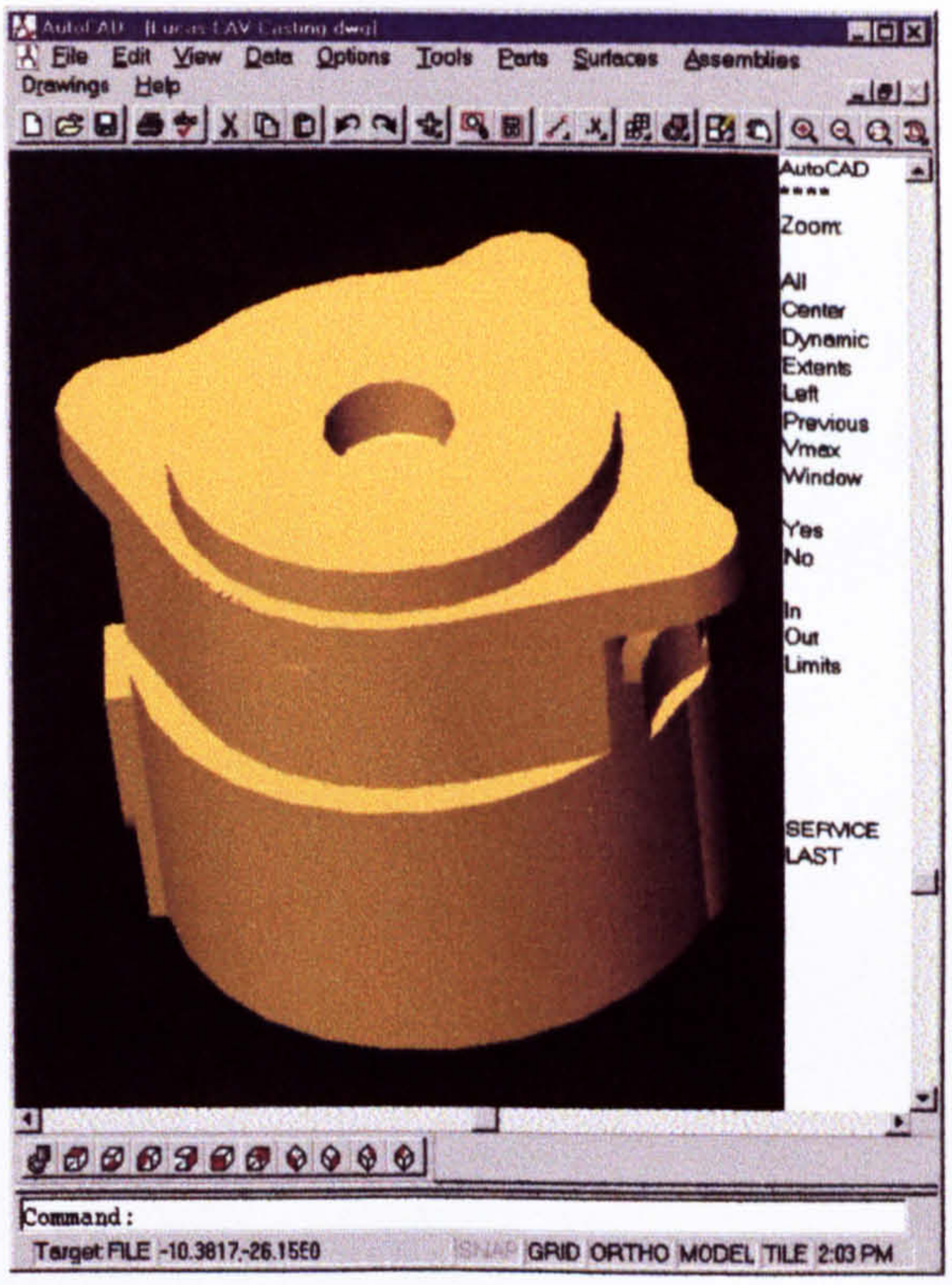


Figure AII.2.3 – Drive-End-Shield No. V6211-695 - CAD Model & Parameters

Parameter	Value
Lug Separation	75mm
Lug Diameter	28mm
Lug Angle	90°
Fillet 2	14mm
Wedge Diameter	102mm
Wedge Depth	39mm
Base Depth	12.7mm
Cylinder Height	94.4mm
Cylinder Bottom Diamet	114.3mm
Top Boss Height	23.8mm
Top Boss Diameter	92.2mm
Middle Boss Height	-
Middle Boss Diameter	-
Centre Hole Diameter	26.5mm
Bump Angle	30°
Bump Radius	12mm
Key Angle	45°

Non-Persistent Feature	Status
Middle Boss	Supp

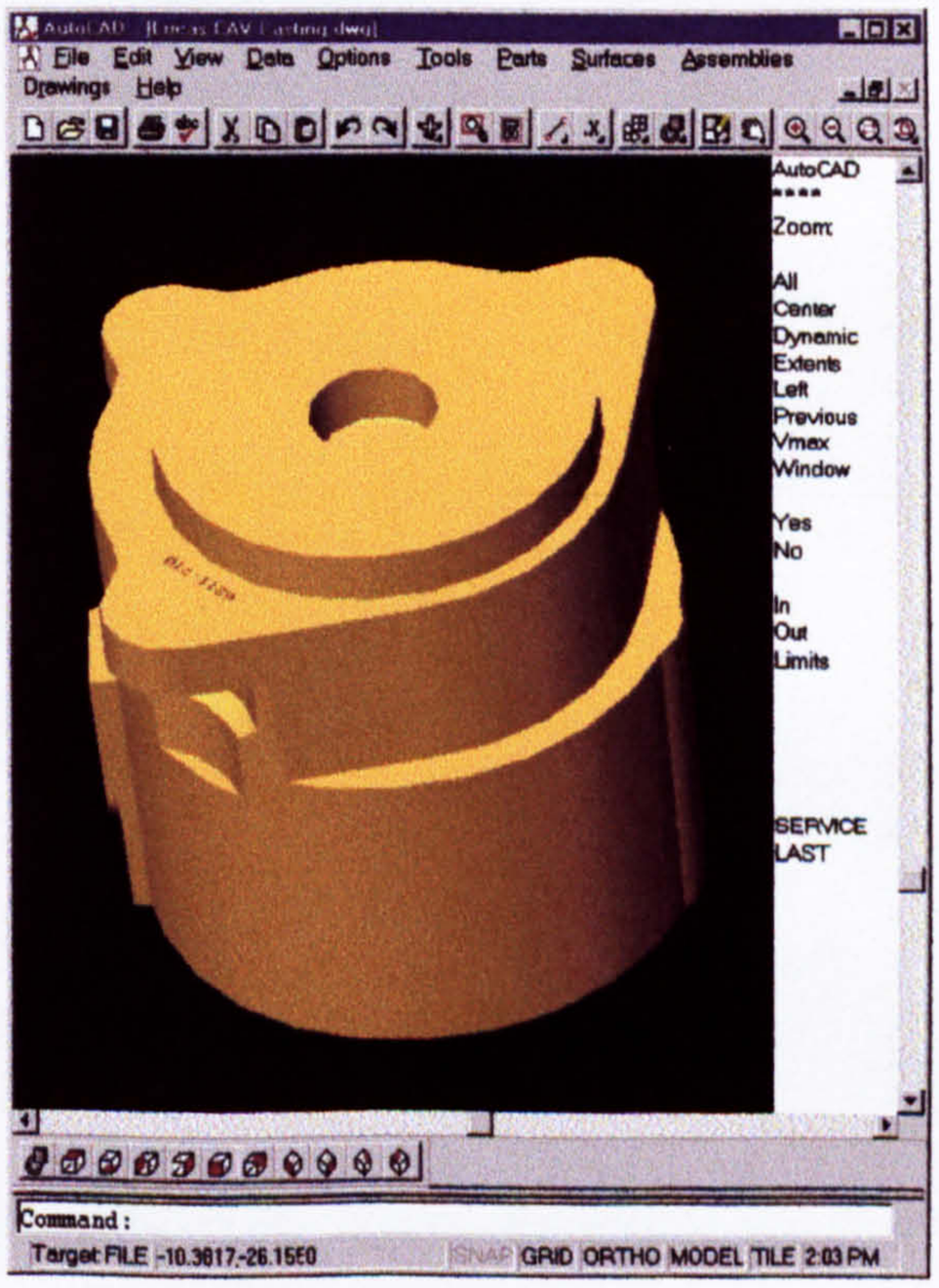


Figure AII.2.4 – Drive-End-Shield No. V6211-710 - CAD Model & Parameters

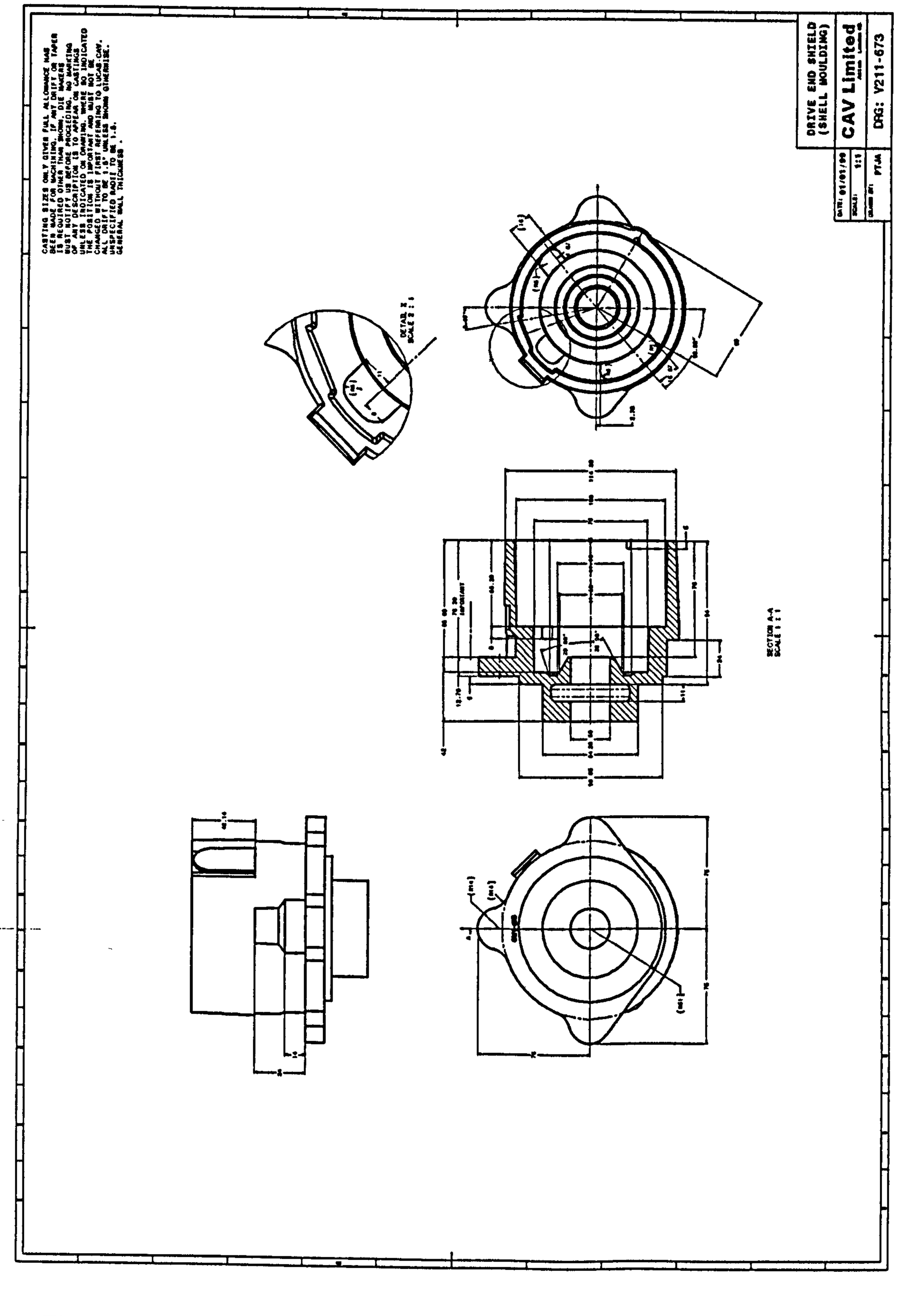


Figure AII.2.5 – Drive-End-Shield No. V6211-673 – Manufacturing *Drawing*

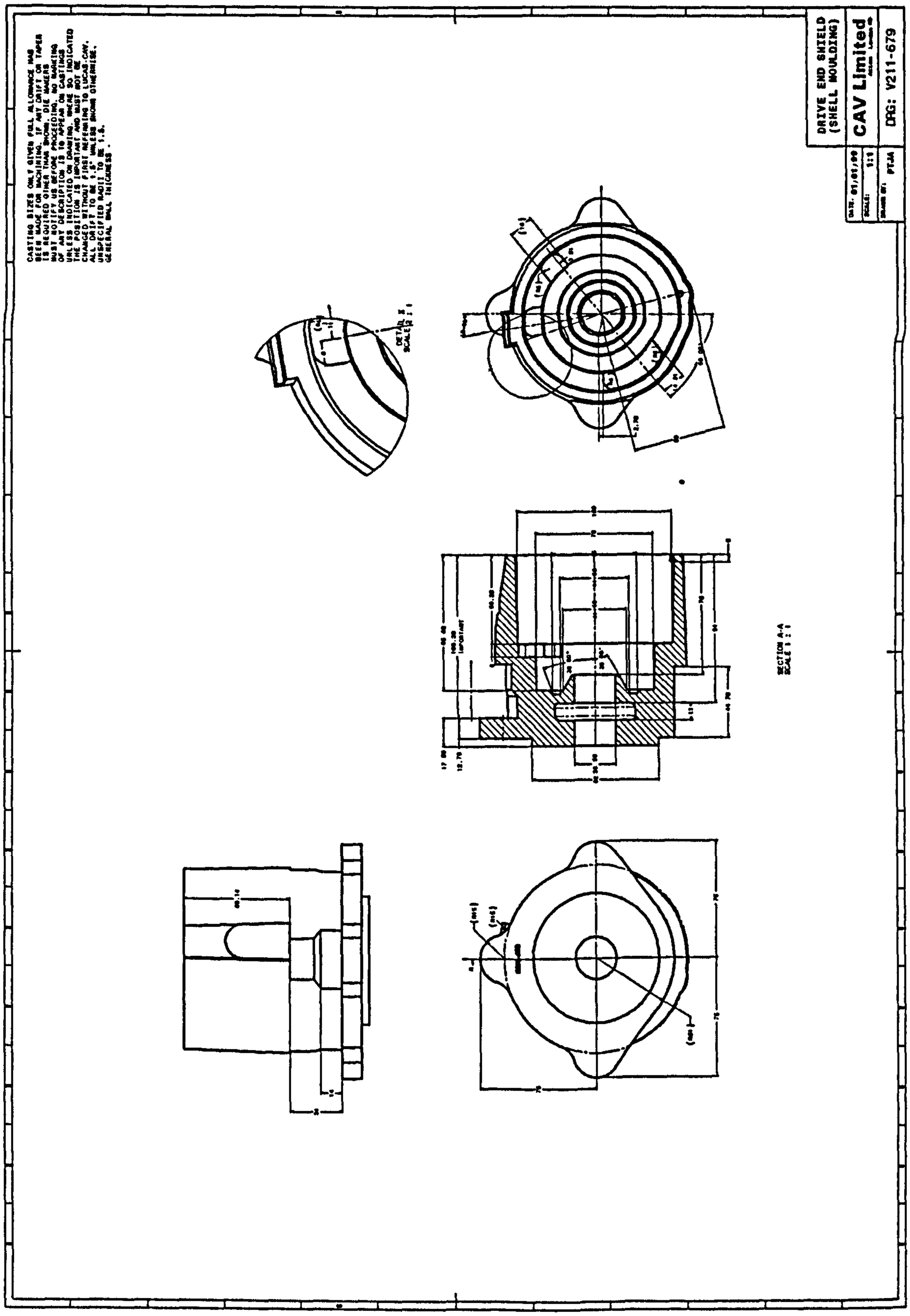


Figure AII.2.6 – Drive-End-Shield No. V6211-679 - Manufacturing Drawing

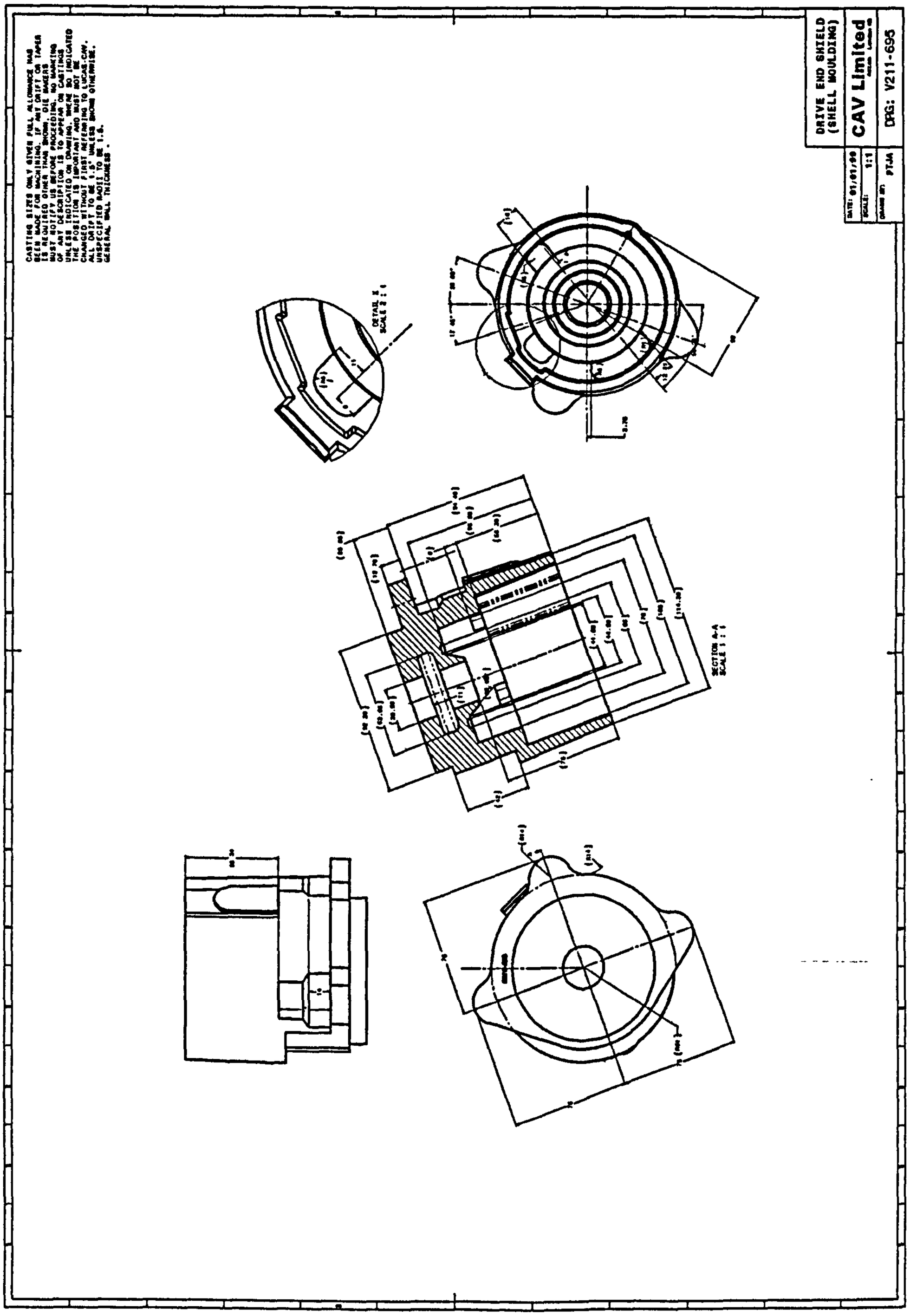


Figure AII.2.7 – Drive-End-Shield No. V6211-695 - Manufacturing Drawing

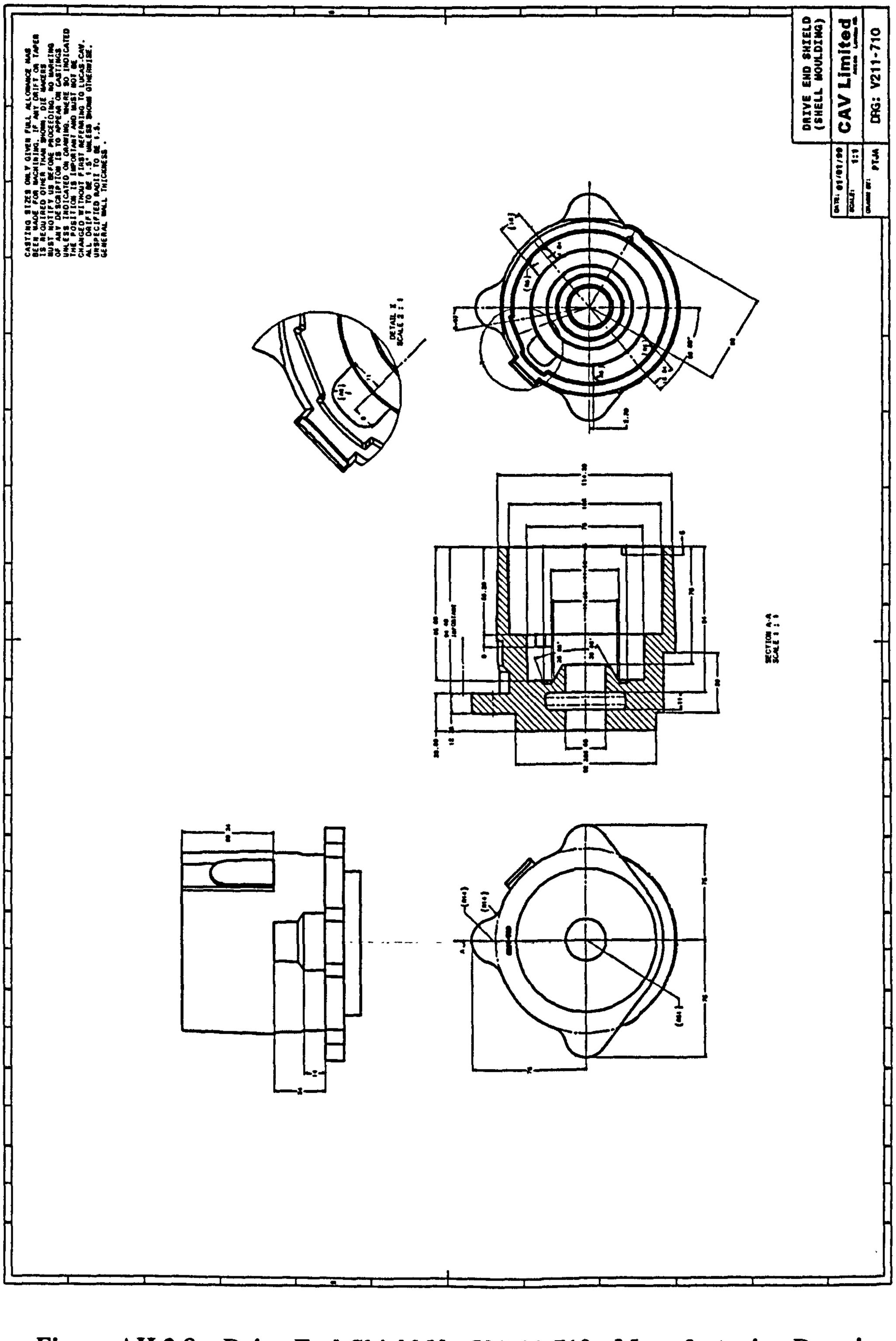


Figure AII.2.8 – Drive-End-Shield No. V6211-710 - Manufacturing Drawing

Figure AII.2.9 – Drive-End-Shield – Rendering

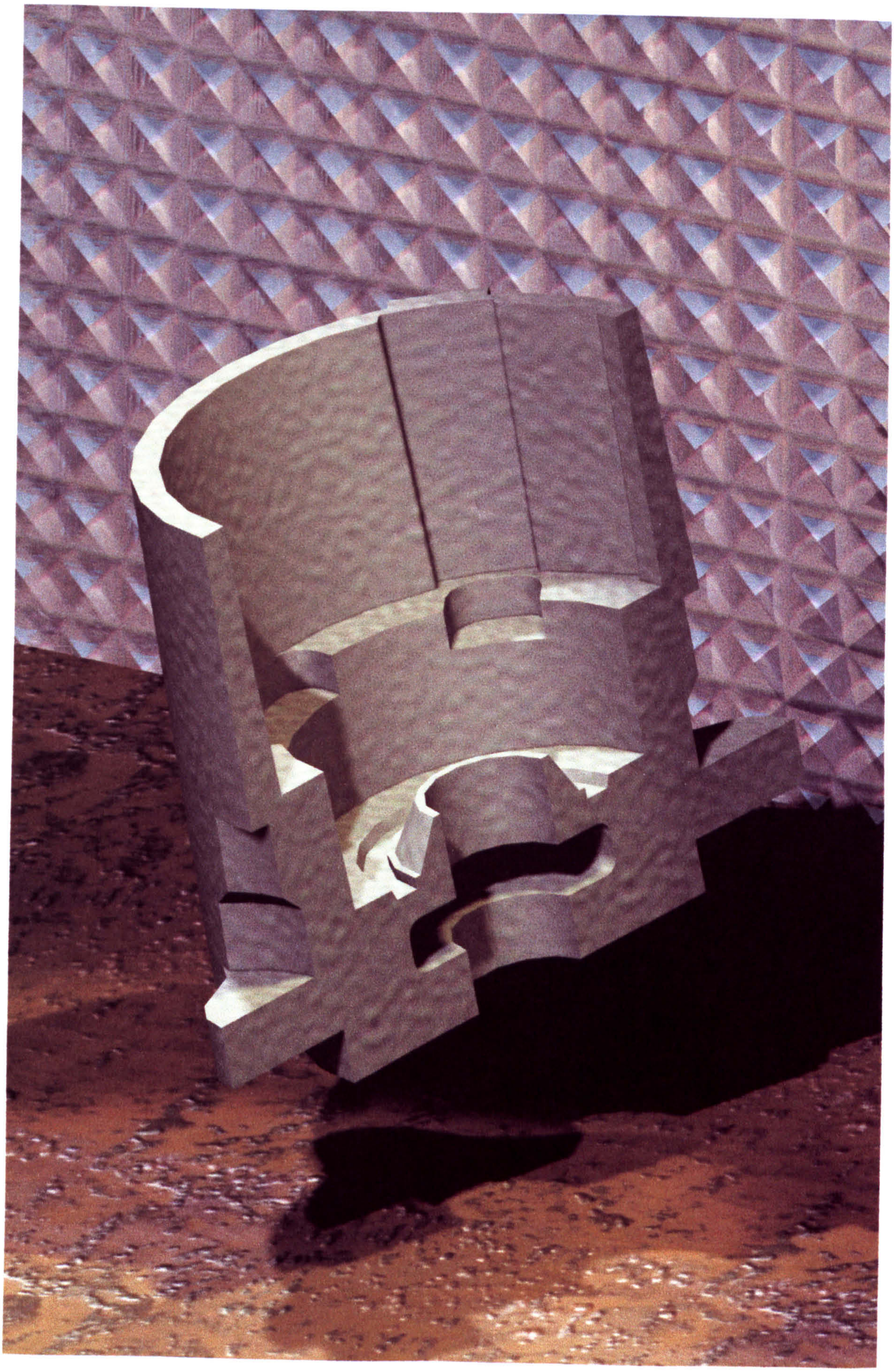


Figure AII.2.10 – Drive-End-Shield – Rendering (Section View)

AII.3 HydroFlow Rotary Drum Filter System

This section contains the following material:

Figures AII.3.1 – AII.3.6, sample manufacturing drawings the Rotary Drum Filter Unit

Figures AII.1.7 – AII.1. 14, the Generic Master Parts created in Solidworks for the Rotary Drum Filter Unit

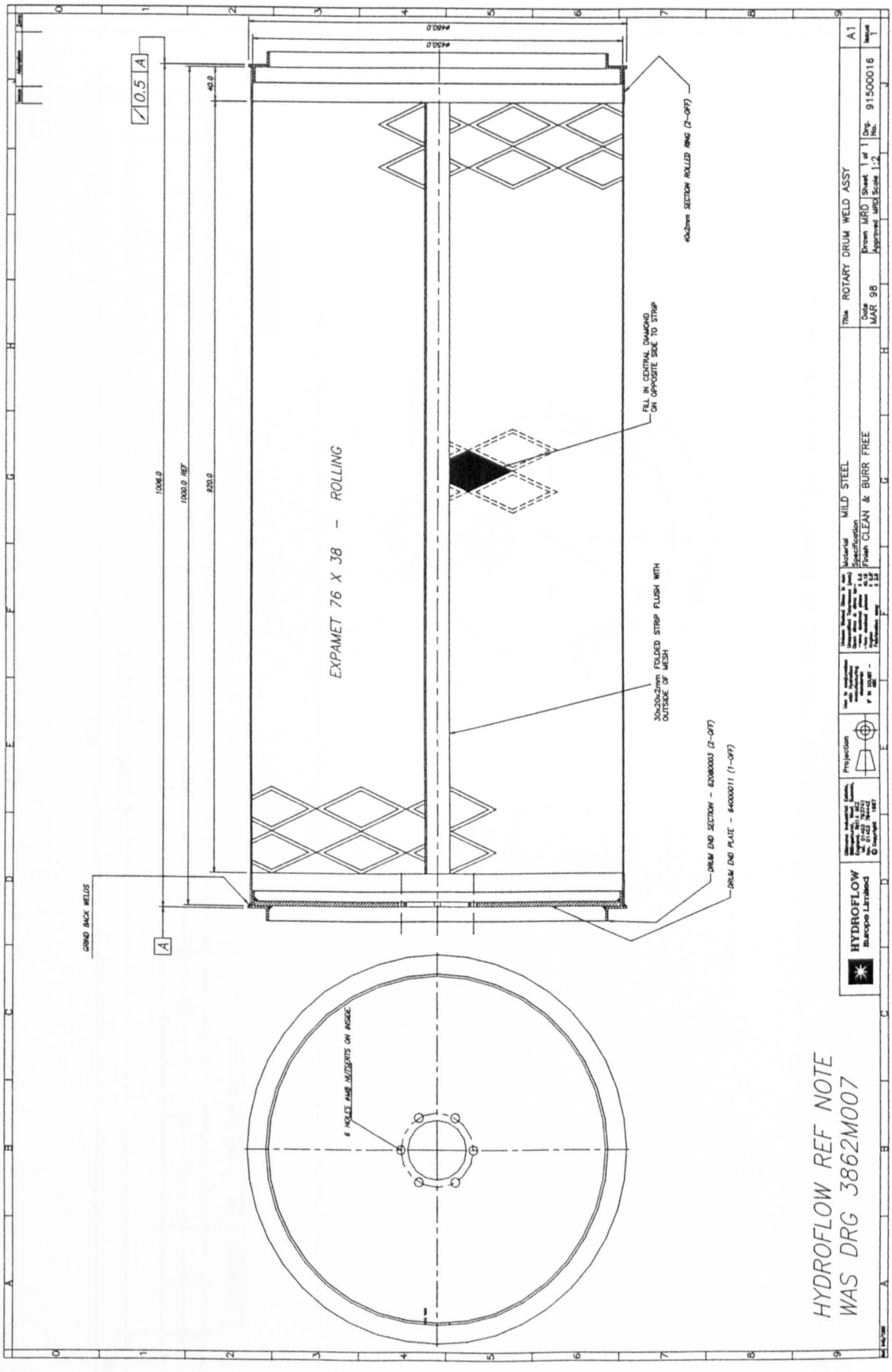


Figure AII.3.1 – Hydroflow Rotary Drum Weld Assembly– Example Manufacturing Drawing

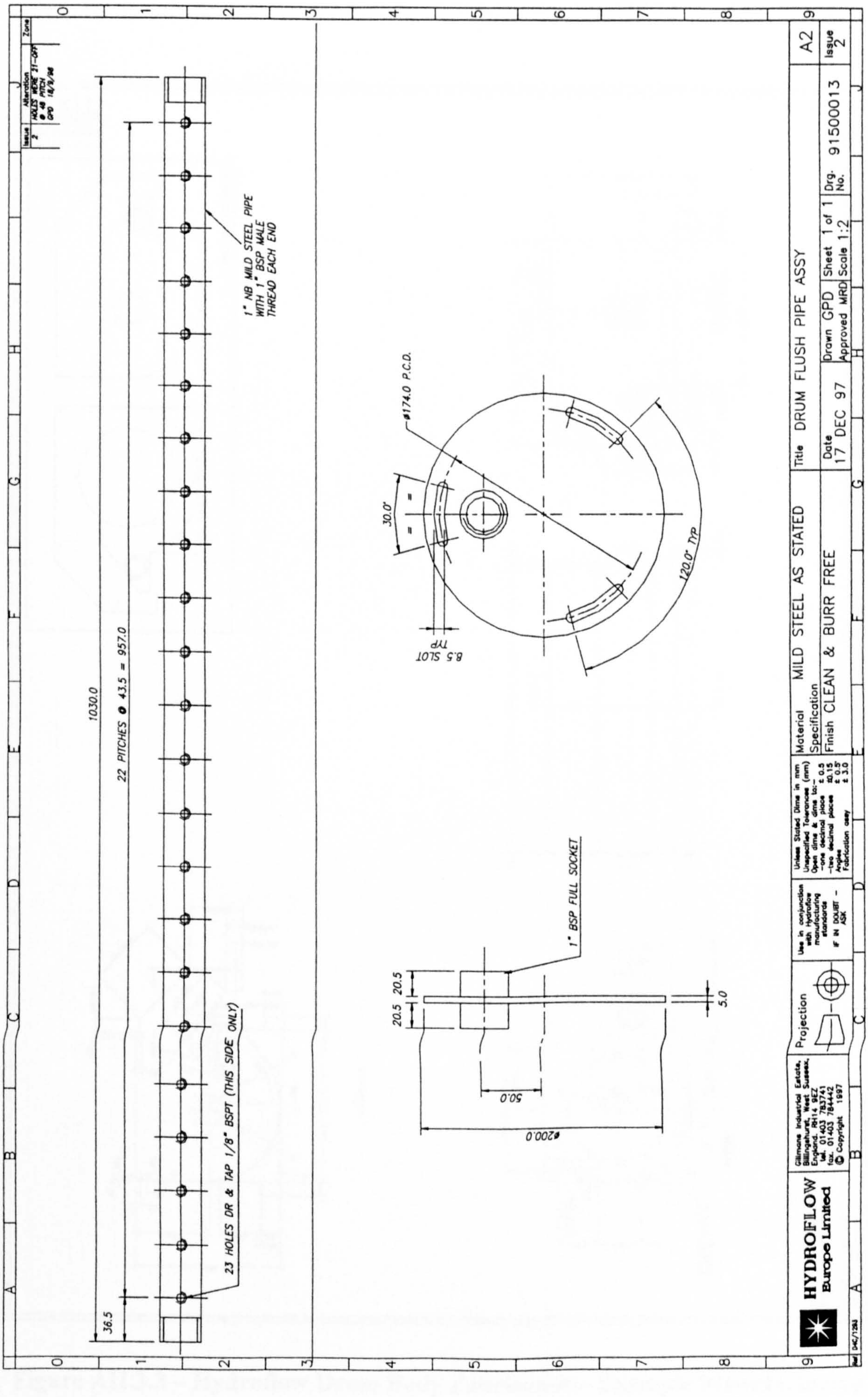


Figure AII.3.2 – Hydroflow Drum Flush Pipe Assembly– Example Manufacturing Drawing

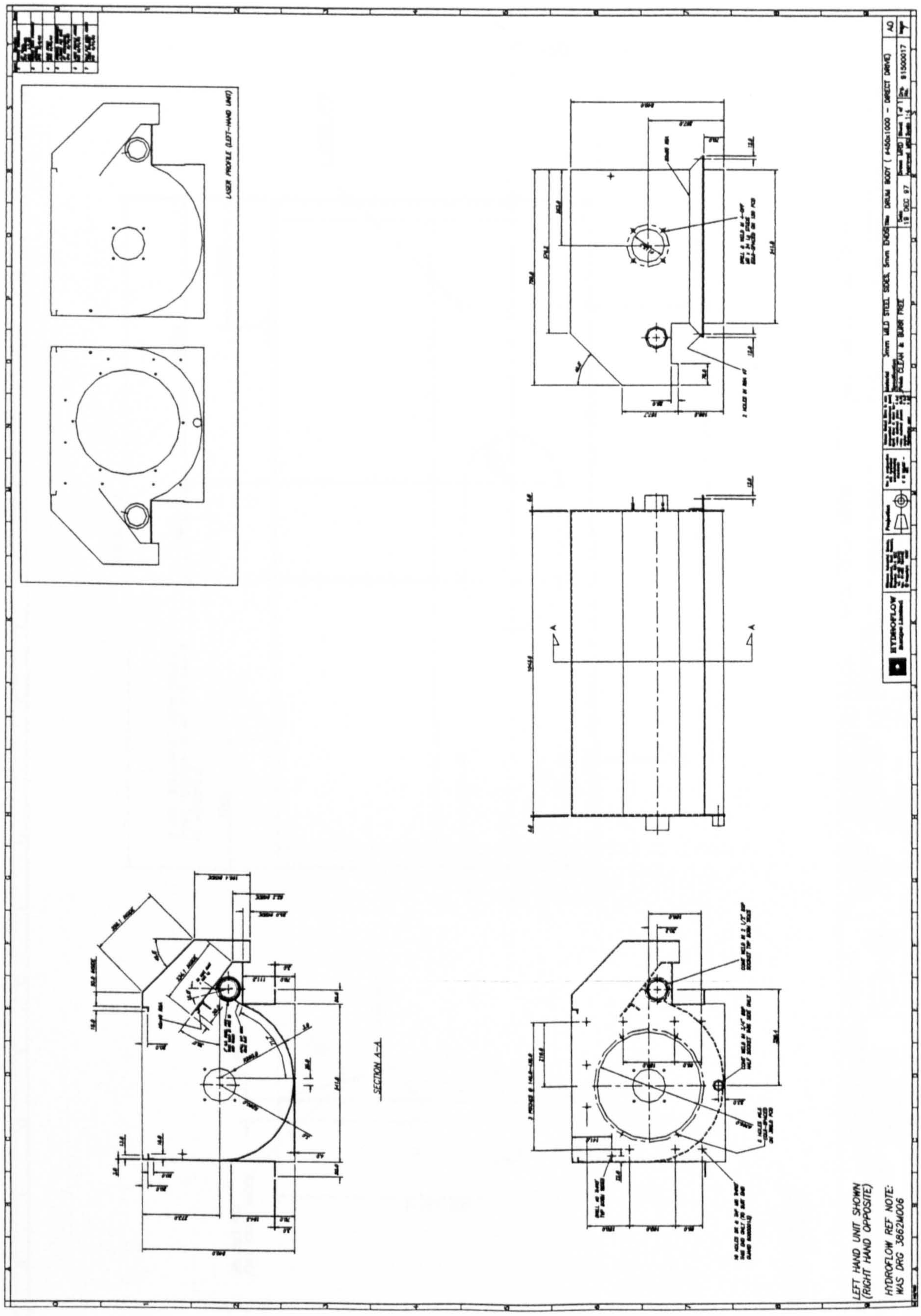


Figure AII.3.3 – Hydroflow Drum Body Fabrication– Example Manufacturing Drawing

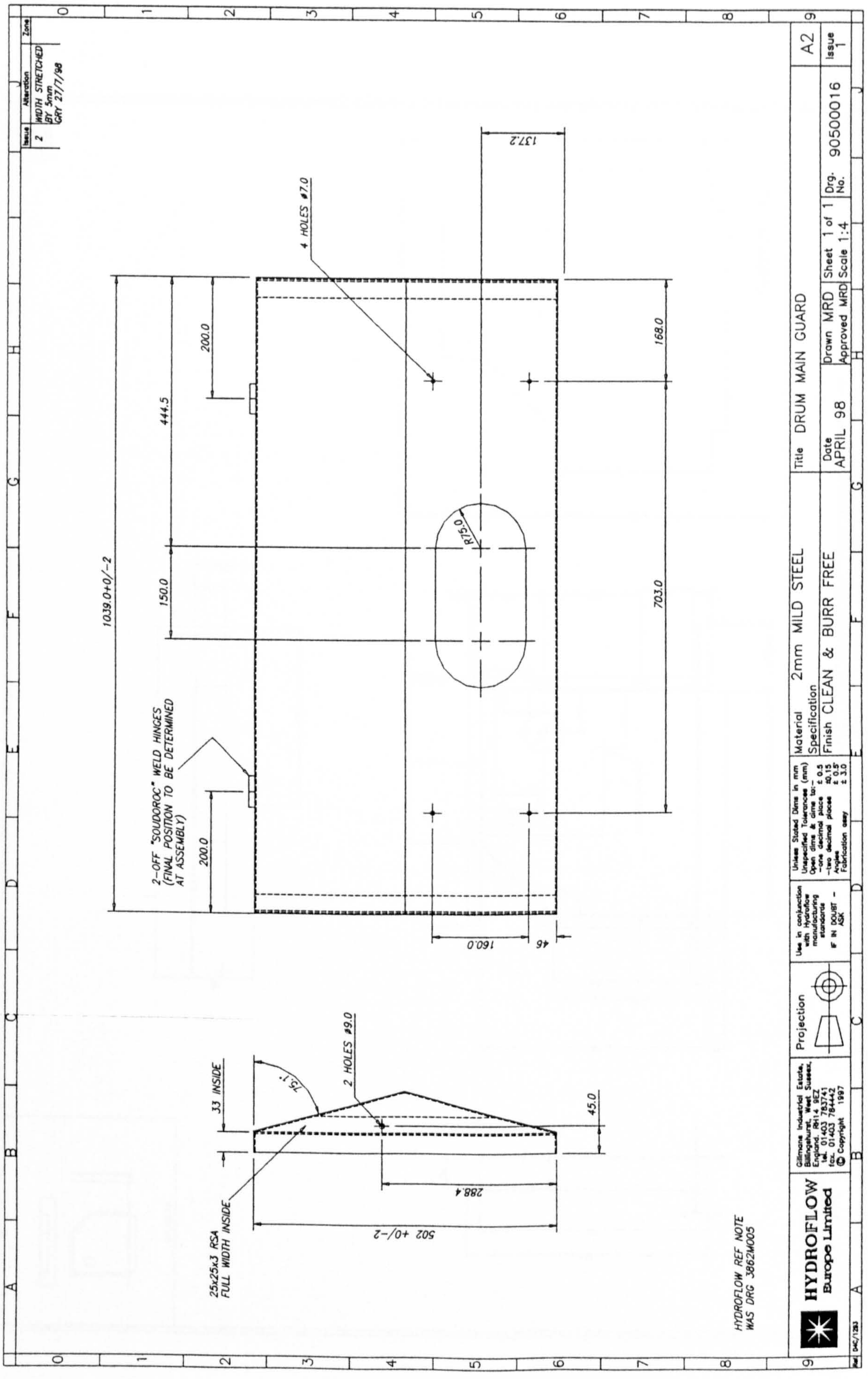


Figure AII.3.4 – Hydroflow Drum Main Guard– Example Manufacturing Drawing

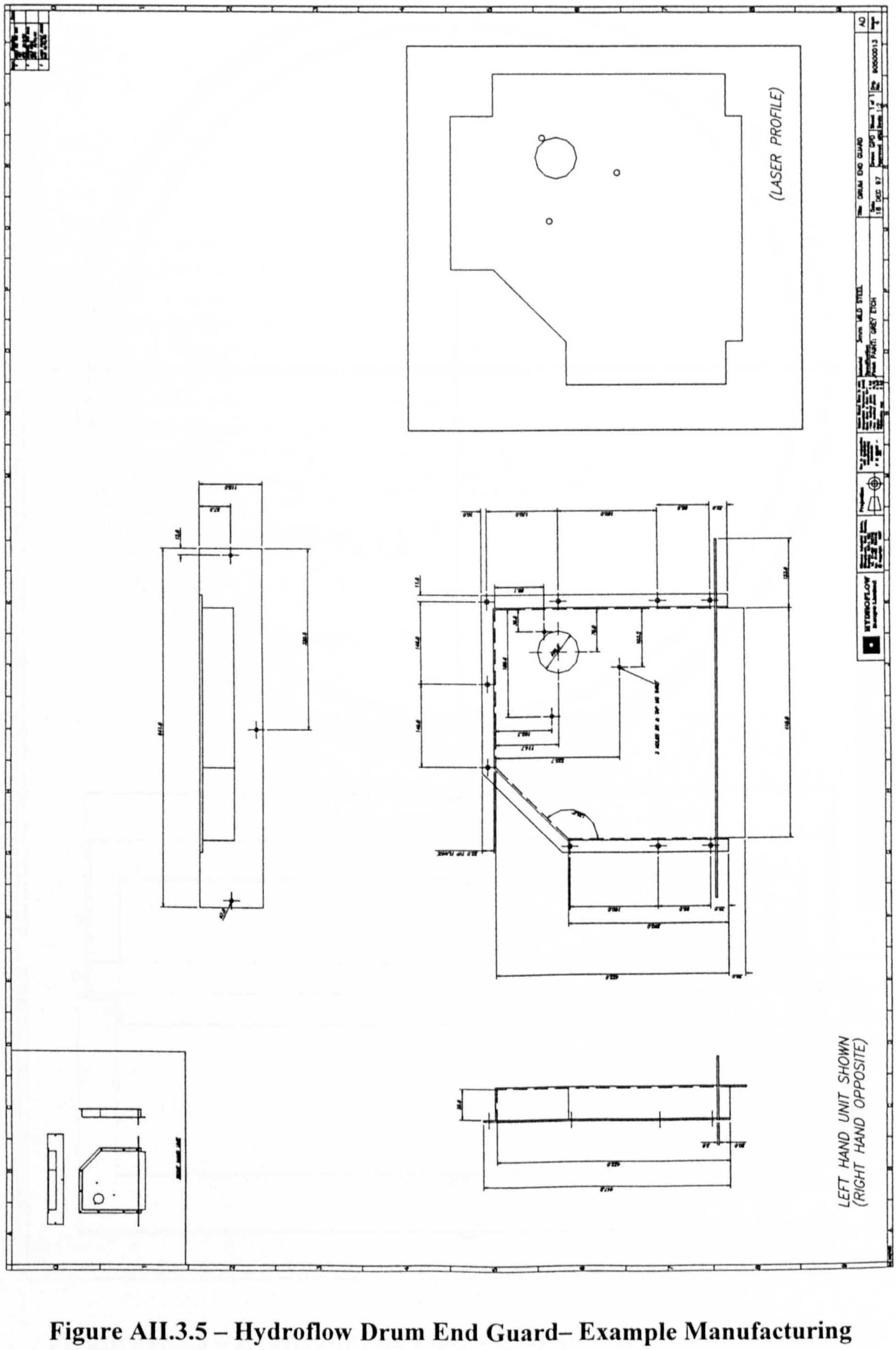


Figure AII.3.5 – Hydroflow Drum End Guard– Example Manufacturing Drawing

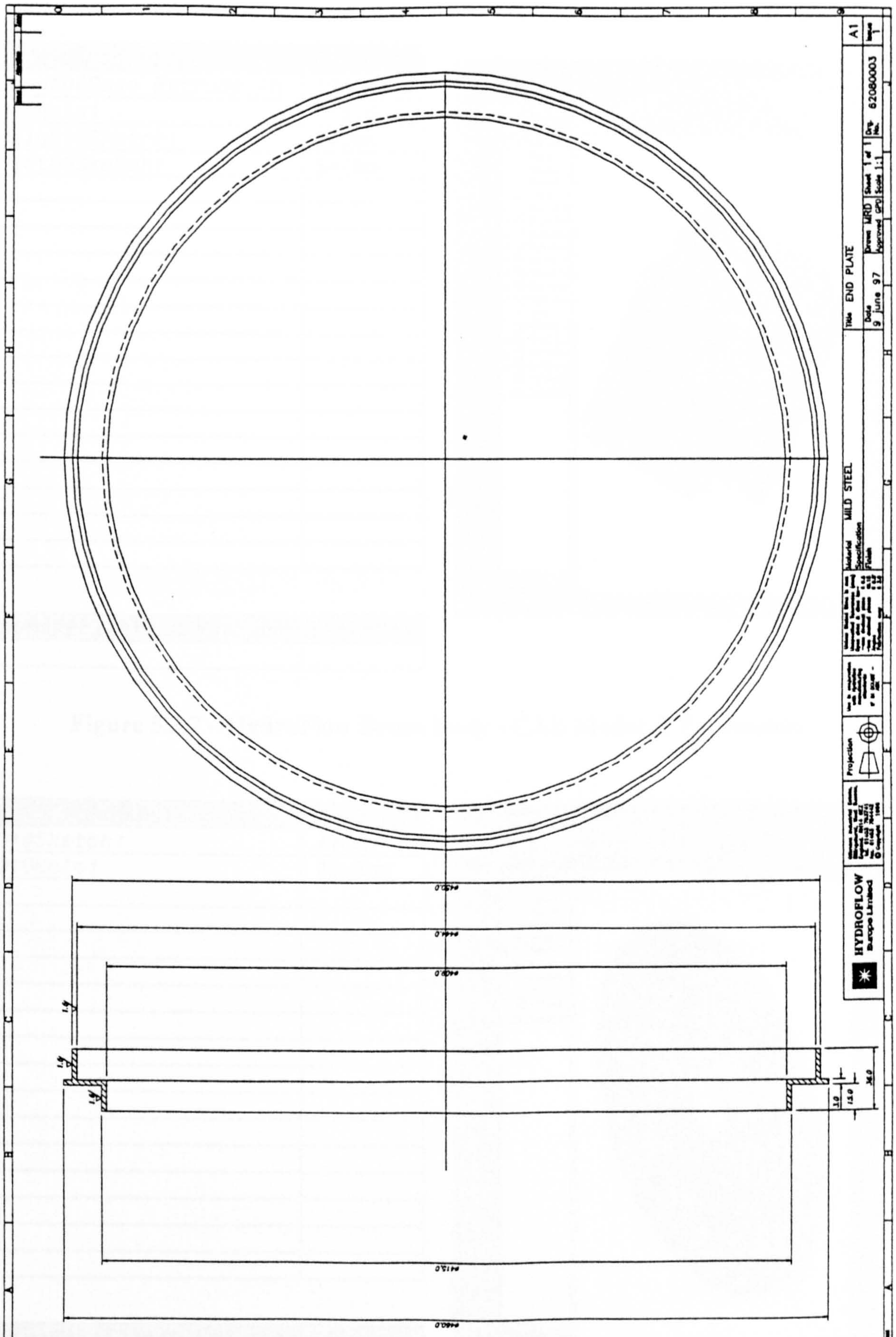


Figure AII.3.6 – Hydroflow End Plate – Example Manufacturing Drawing

Parameter	Value
Length@Base Extrude Th	1043mm
D1@Hole1	115.5mm
Height@Sketch1	540mm
Width@Sketch1	541mm

Non-Persistent Feature	Status

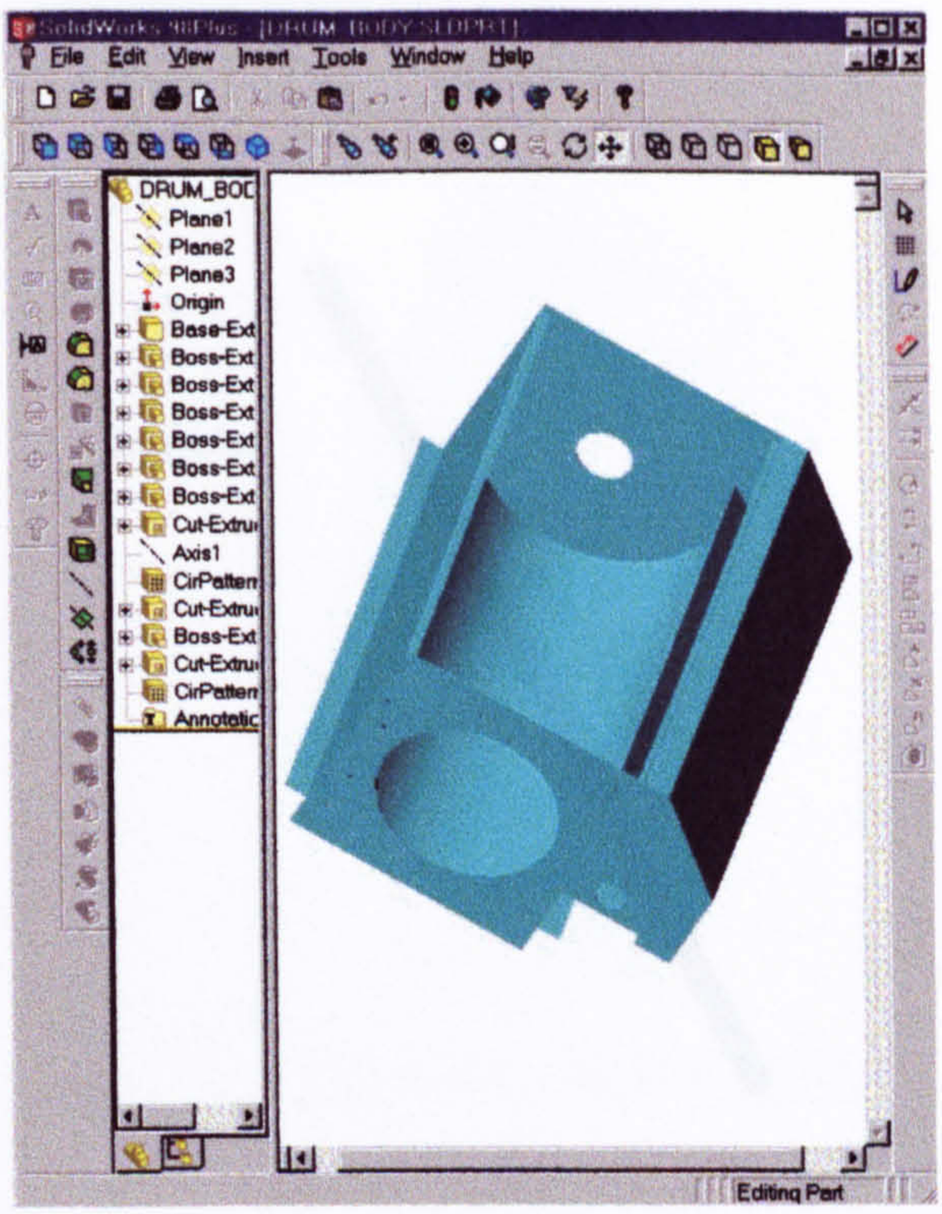


Figure 5.3.7 – HydroFlow Drum Body - CAD Model & Parameters

Parameter	Value
D1@Sketch1	444mm
D1@Hole1	70.5mm

Non-Persistent Feature	Status

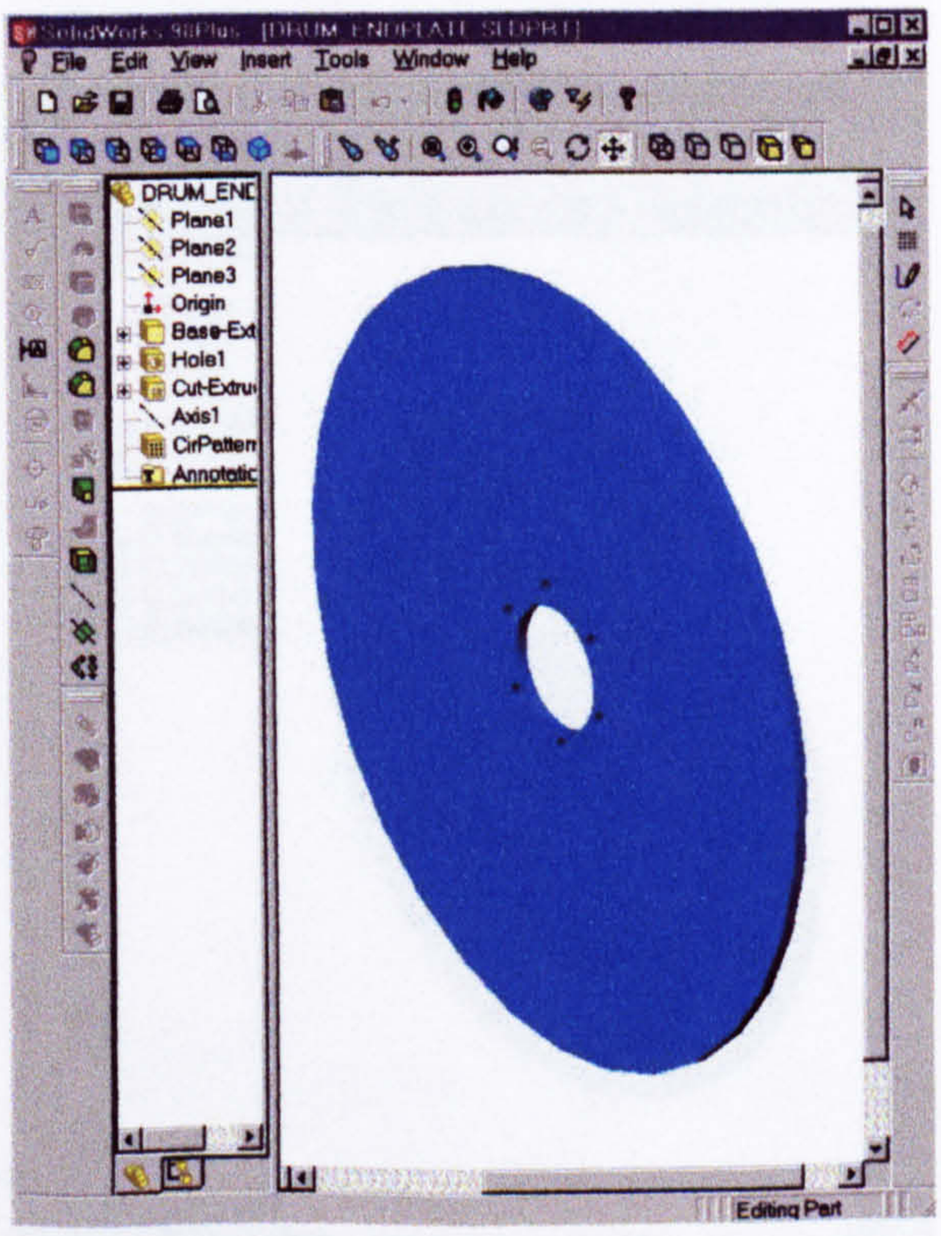


Figure 5.3.8 – HydroFlow Drum Endplate - CAD Model & Parameters

Parameter	Value
D1@Sketch1	1inch
D1@Base Extrude1	1030mm
D1@Pattern1	23
D3@Pattern1	43.5mm

Non-Persistent Feature	Status

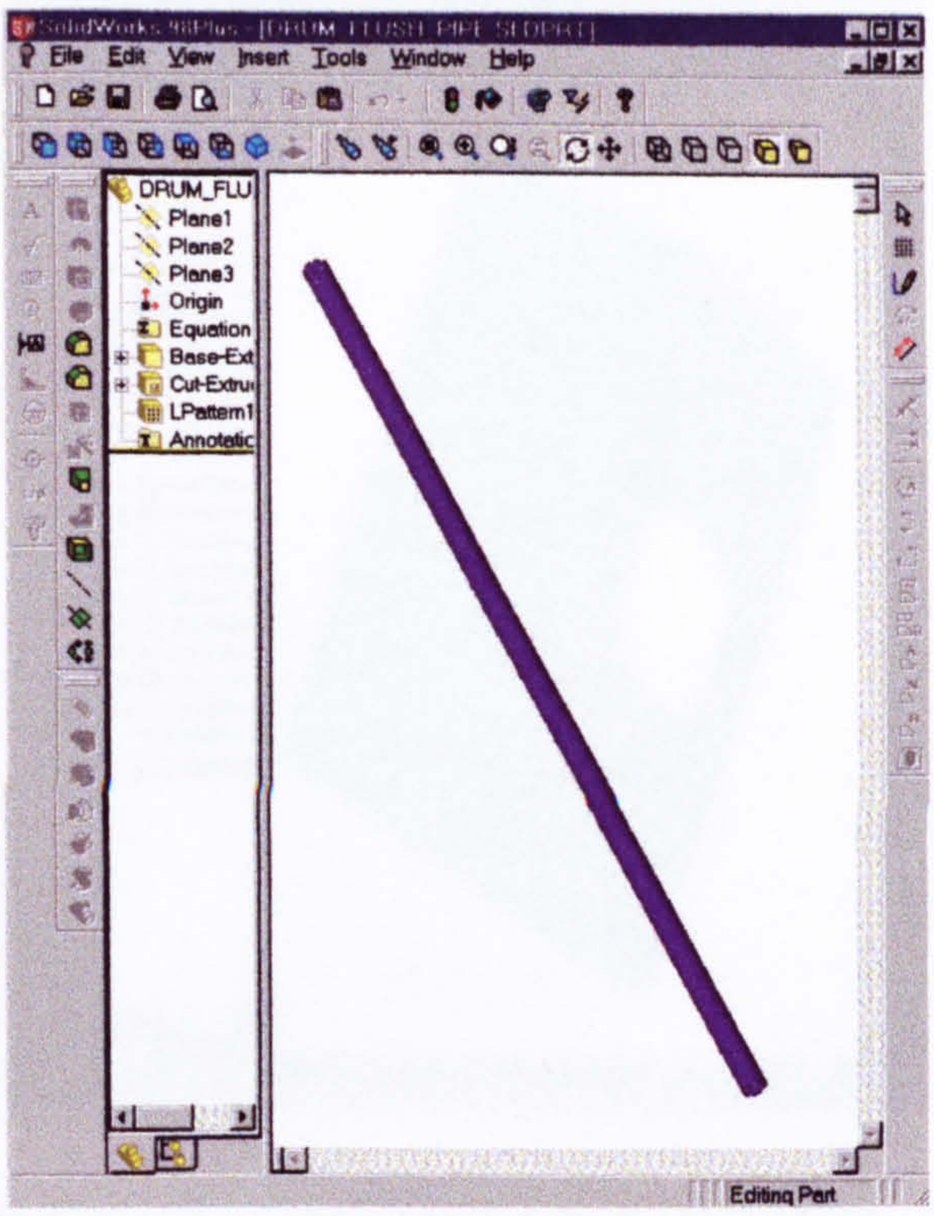


Figure 5.3.9 – HydroFlow Drum Flush Pipe - CAD Model & Parameters

Parameter	Value
D1@Sketch1	200mm
D1@Extrude Base	5mm
Offset@Sketch2	50mm
D1@Sketch2	1inch
PCD@Sketch3	174mm
Flush Angle@Sketch3	30°

Non-Persistent Feature	Status

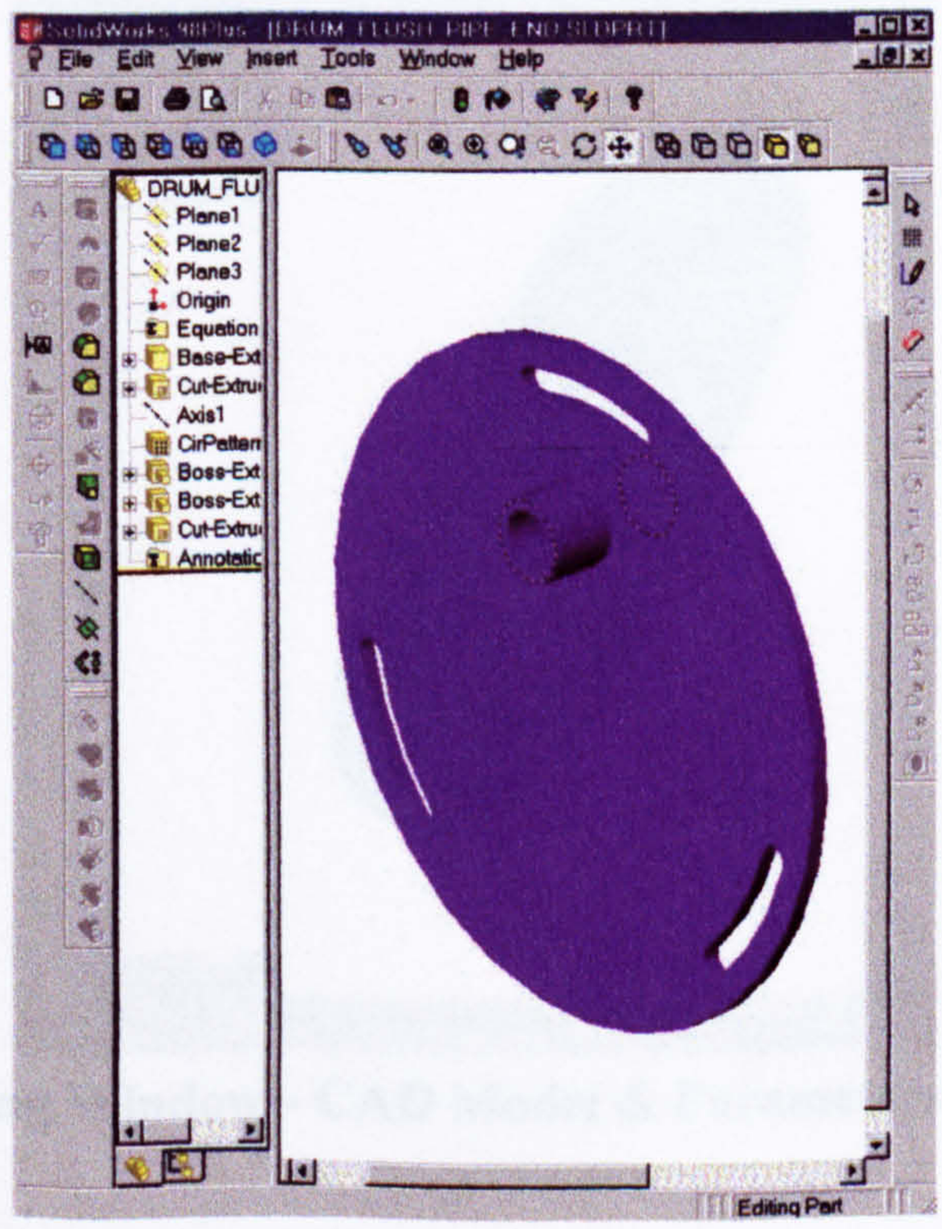


Figure 5.3.10 – HydroFlow Drum Flush Pipe End - CAD Model & Parameters

Parameter	Value
D1@Base Extrude Thin	1039mm
Depth@Sketch1	502mm
Roof Angle@Sketch1	75.1°
D1@Sketch3	150mm
D2@Sketch3	75mm

Non-Persistent Feature	Status

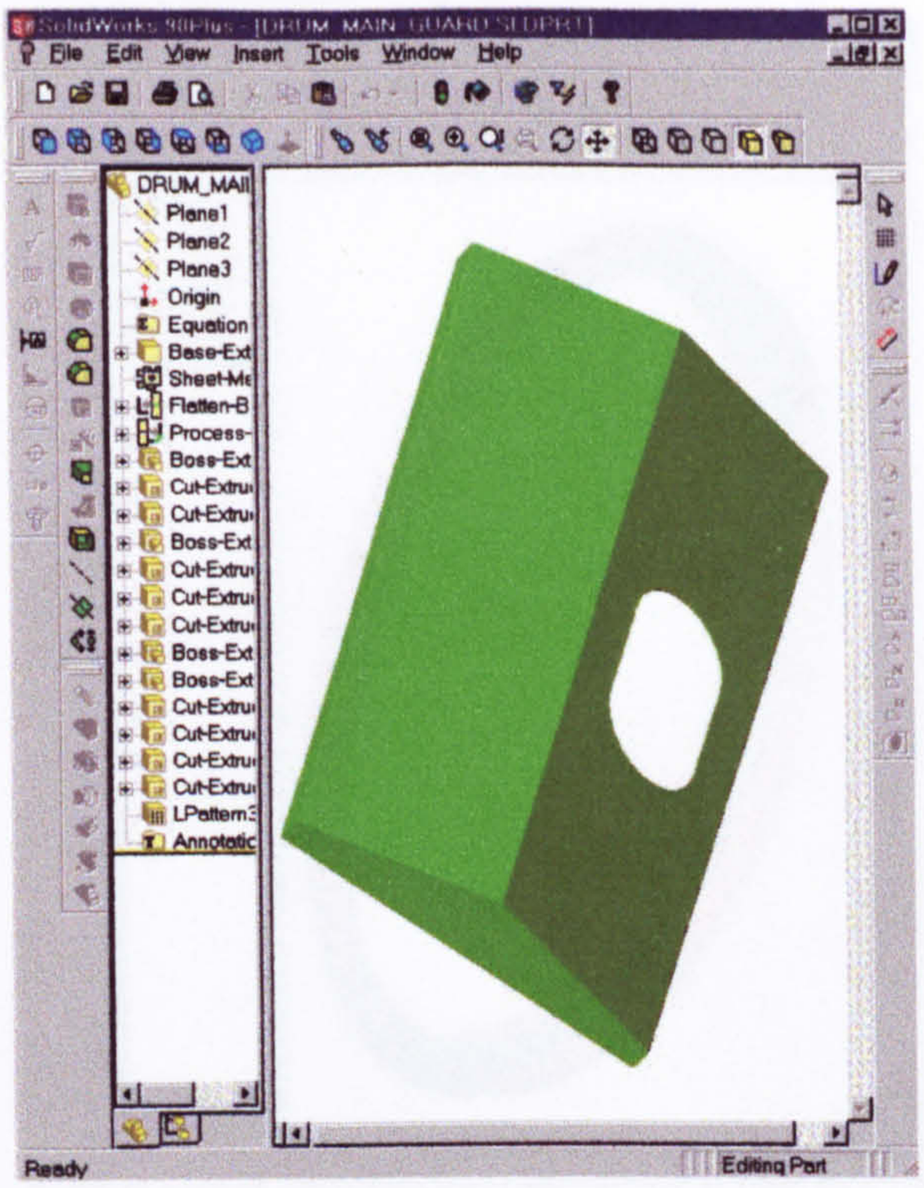


Figure 5.3.11 – HydroFlow Drum Main Guard - CAD Model & Parameters

Parameter	Value
Length@Sketch1	150mm
Corner Radius@Sketch1	68.5mm
Thickness@Base Extrude	3mm

Non-Persistent Feature	Status

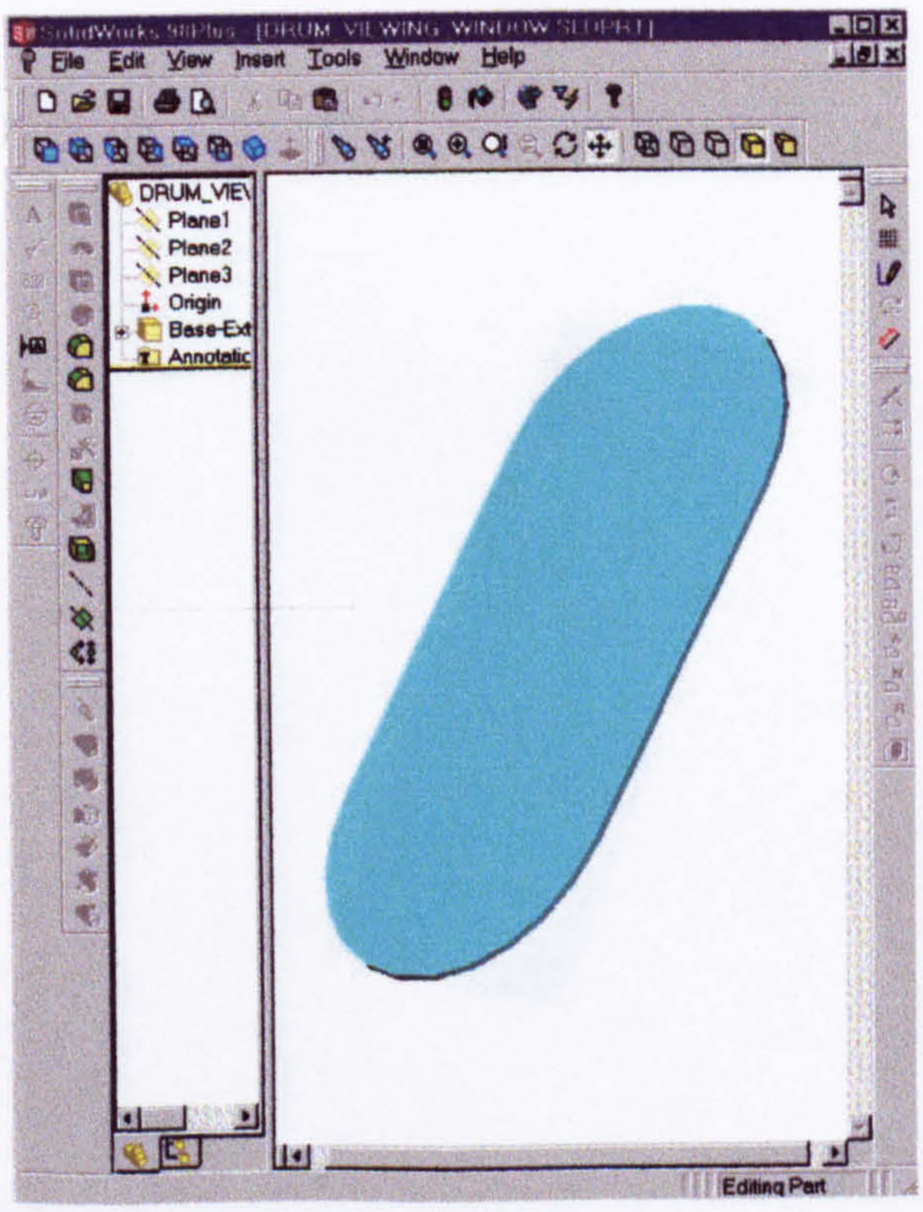


Figure 5.3.12 – HydroFlow Drum Viewing Window - CAD Model & Parameters

Parameter	Value
Outer Dia@Sketch1	460mm
Bore Diameter@Sketch1	409mm
Thickness@Sketch1	3mm
Lip Diameter@Sketch1	450mm
D5@Sketch1	36mm

Non-Persistent Feature	Status

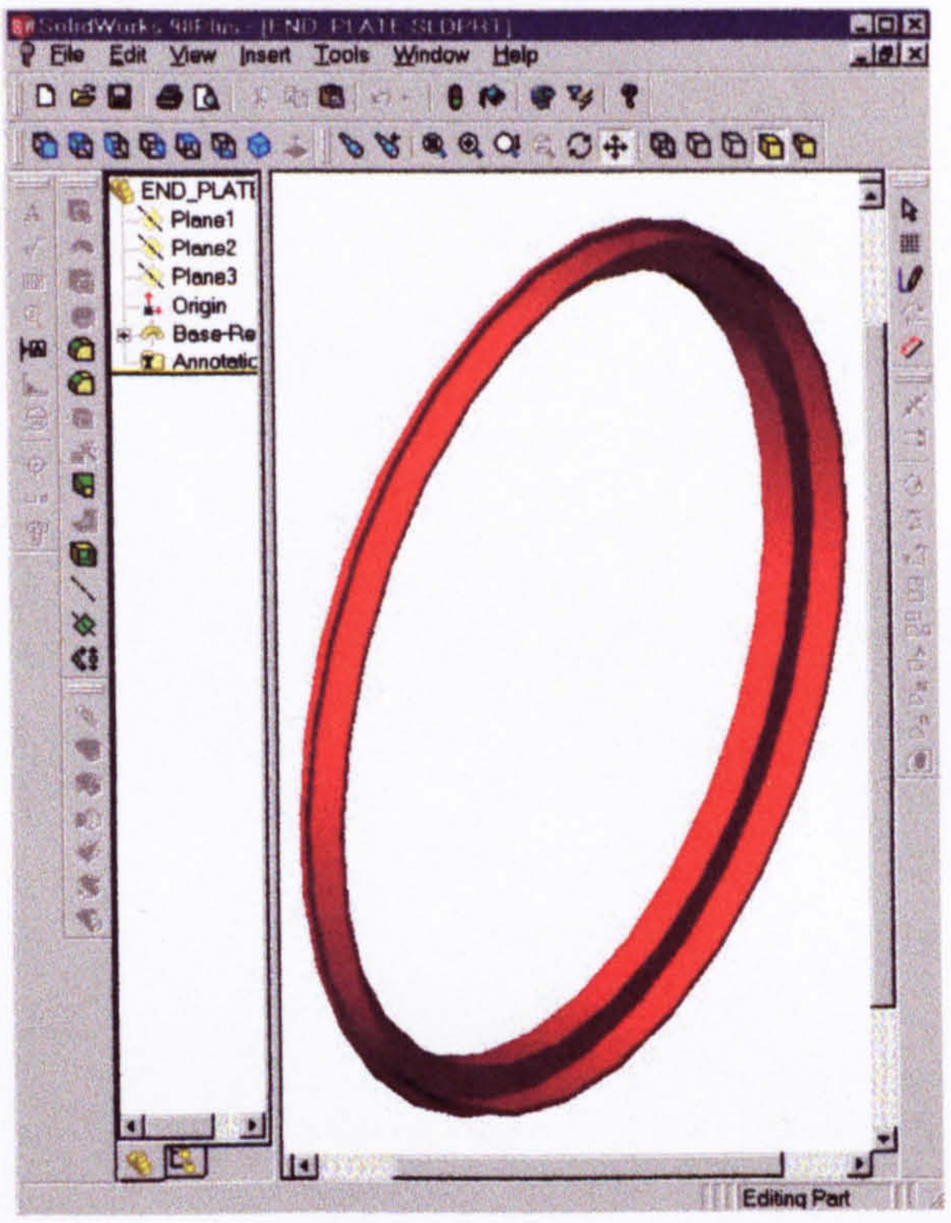


Figure 5.3.13 – HydroFlow End Plate - CAD Model & Parameters

Parameter	Value
D1@Base Extrude	970mm
Lip Height@Sketch1	5mm
Lip Angle@Sketch1	5°

Non-Persistent Feature	Status

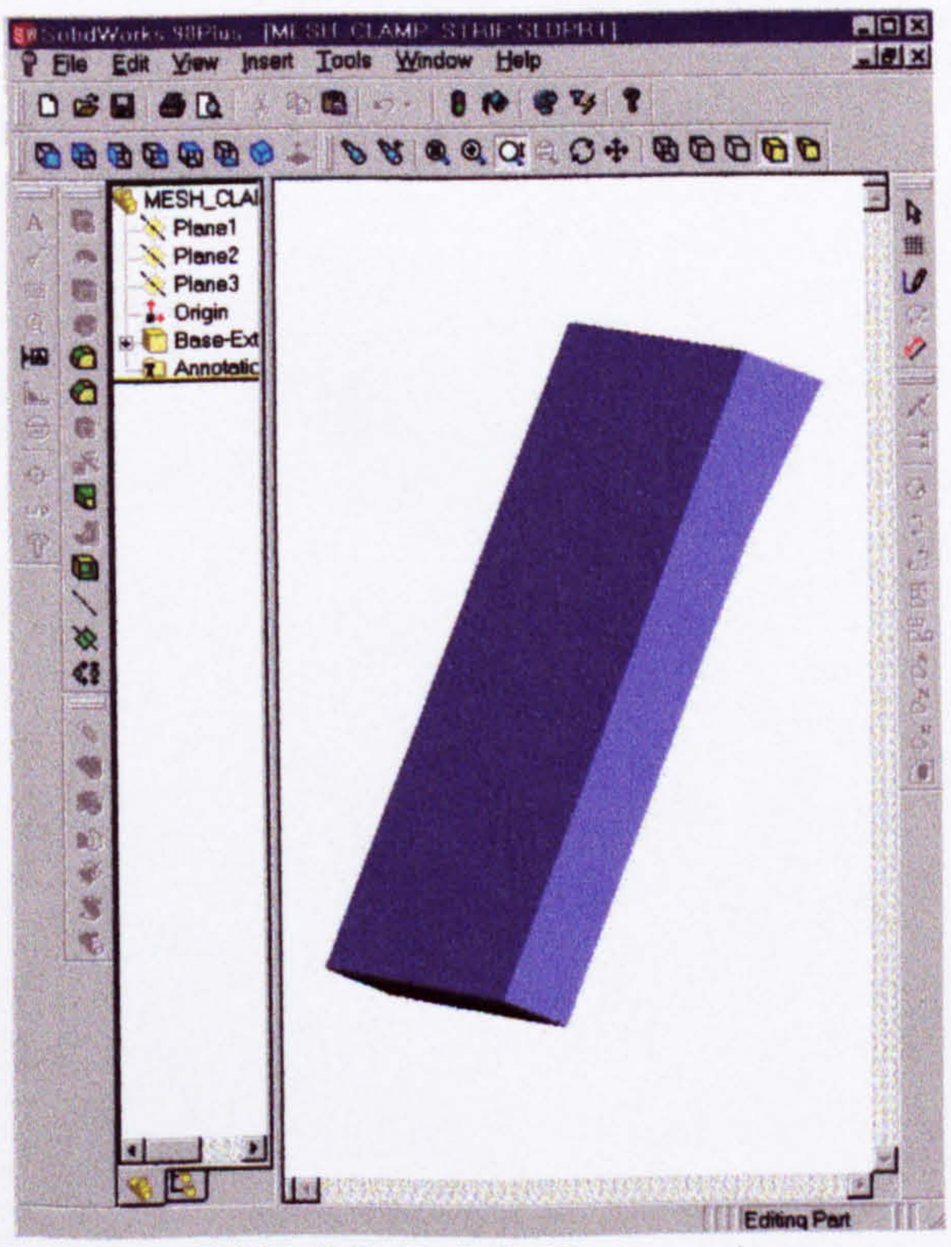


Figure 5.3.14 – HydroFlow Mesh Clamp - CAD Model & Parameters