Implementation of Axiomatic Design Theory

by

Taejung Kim

B.S., Mechanical Design and Production Engineering
Seoul National University, Seoul, Korea
(1993)

Submitted to the Department of Mechanical Engineering
in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Mechanical Engineering
at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
June 1995

Signature of Author________________________________
Department of Mechanical Engineering
June 9, 1995

Certified by________________________________
Nam P. Suh
Ralph E. and Eloise F. Cross Professor of Manufacturing

Accepted by________________________________
Ain A. Sonin
Chairman, Departmental Graduate Committee

Implementation of Axiomatic Design Theory

by

Taejung Kim

Submitted to the Department of Mechanical Engineering
on June, 1995 in partial fulfillment of the
requirements for the Degree of

Master of Science in Mechanical Engineering

ABSTRACT

Axiomatic design theory was advanced as a systematic approach to the design field. It explains the general features of design activity, extracts common characteristics of good designs and provides a methodology/procedure consistent to them. The common characteristics of good designs is called 'axioms' or 'design axioms'.

ADE(Axiomatic Design Environment) may be defined as a computer software which assists designers in following axiomatic design procedure. ADE satisfies the following specific goals : ADE provides the editing environment which can clearly show the basic concepts of axiomatic design theory (which are domains, functional requirement, design parameter, design matrix, design hierarchy, mapping and decomposition); ADE helps designers to keep axiomatic design procedure by prohibiting the designers from taking undesirable design sequence; ADE helps designers' ideation for appropriate design solutions; ADE has the ability to check whether on not a design is acceptable according to design axioms; ADE has the ability to record a designer's design and retrive it on computer.

A framework for software design, based on axiomatic design theory, was suggested and applied to designing ADE. Software design process is formalized with concepts of axiomatic design theory.

Further research needs to be focused on the ideation process for design solutions.

Thesis Supervisor : **Professor Nam Pyo Suh**
Ralph E. and Eloise F. Cross Professor of Manufacuring
Head of the Department

Acknowledgements

It is always the most exciting experience to finish one step in life, especially when it can results in a kind of physical form like this thesis. At this very exciting moment, I would like to give thanks to those who have contributed much to the completion of my Master's thesis.

I wish to express my sincere thanks to my advisor, Professor Nam Suh. He lead me to the right direction during this work. I appreciate his critical advice, encouragement, understanding and time spent for them.

I gratefully acknowledge the help from Professor Leonard Albano at WPI. From the start of this research, the discussion with him and his advice was a big help to me.

Thanks to the members of Axiomatic Design Group at MIT for their kindness and discussion with me.

I would also like to thank my friends, Jun, Sung, JunHee, Siho, Simon, Kang, Yong, Kevin, Caroline, Bo, Kay and Yu for cheering me up and making morning calls to their lazy friend. They were with me when I needed them.

Thanks to my sister and my brother Yoon for their love and encouragement.

Special thanks to my Parents. Their son in America, so far from his family, made them worry about his health and his life. Without their love, he would have lost the faith for his life.

Table of Contents

List of Figures

List of Tables

CHAPTER 1. Introduction

Axiomatic design theory is advanced by Suh[1,3,7,9] as a systematic approach to the design field. It explains the general features of design activity, extracts common characteristics of good designs and provides a methodology consistent to them.

As the system which designers must deal with becomes large, it is difficult for a designer to follow the axiomatic design methodology. If the knowledge about axiomatic design theory is implemented on digital computer, designers can be lead to follow the methodology and get the information about this theory in an interactive way from computer.

ADE(Axiomatic Design Environment), a computer software to assist designers in following the axiomatic design methodology, was developed in this thesis.

In this thesis, the concept of ADE was introduced and how the ADE was designed with axiomatic design theory was explained. In chapter 2, the background knowledge which is needed to understand other part of this study was summarized. In chapter 3, the concept of ADE was suggested. In chapter 4, how the code for ADE was generated was shown and a framework for software design based on axiomatic design theory was suggested.

CHAPTER 2. Background

Axiomatic design theory is advanced by Suh[1,3,7,9] as a systematic approach to the design field. It explains the general features of design activity, extracts common characteristics of good designs and provides a methodology consistent to them. The common characteristics of good designs are called 'axioms' after which this theory was named. What distinguishes this theory most from other design theories is the assumption that the axioms exist and the fact that these axioms are applied in the methodology.

TDM (Thinking Design Machine) concept was created by Suh[1,2]. Its ultimate goal is to create designs or design concepts that are superior to those currently possible using software based on the axiomatic design theory. Therefore, TDM should have the ability to generate designs and to evaluate them using axiomatic design theory. Much work, related to human creativity and synthesis power, remains to be done to achieve this goal.

In this chapter, the main concepts of axiomatic design theory and the TDM are introduced for those who are unfamiliar to them.

2.1. Review of Axiomatic Design Theory

In this section the main concepts of axiomatic design theory are presented for those who are unfamiliar to this theory. Detailed explanations and examples are given in [1].

2.1.1. Basic Concepts and Facts of Design Activity

In this sub-section, the terminology which is used in axiomatic design theory and the characteristics of design activity are explained.

According to axiomatic design theory, design process is a mapping process between domains. At the highest level of the design hierarchy, each domain has its characteristic vector which can completely characterize the entity of the domain. Using the characteristic vector, the mapping can be expressed in a mathematical form. The equation that represents the relationship between the characteristic vectors is called a design equation. In each domains, the characteristic vector can be decomposed into its own hierarchy. The hierarchy can not be developed without mapping to other domains. The independence of FRs of a design is defined in this section and explained in terms of the design equation. Also, the concept of information content is introduced for a measure of complexity of a design.

Below are the detailed explanations of these concepts :

FRs (*Functional Requirements* **)**

FRs are defined as a minimum set of independent requirements that completely characterize the design objective for a specific need.

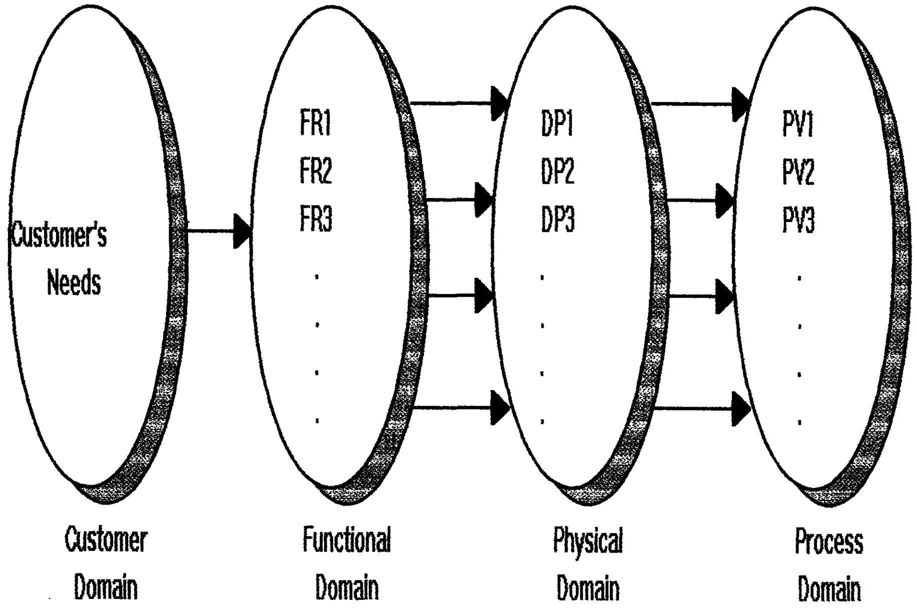
DPs (*Design Parameters*)

DPs are the key variables that characterize the physical entity created by the design process to fulfill the FRs.

PVs (*Process Variables*)

PVs are the key variables that characterize the manufacturing process which is generated to satisfy DPs.

Domains



<Figure 2.1> Four Domains of the Design World

Axiomatic design theory describes the design process as the activity of mapping between domains of the design world. There are four domains : the customer domain, the functional domain, the physical domain and the process domain.

In the customer domain, customer's needs are described. In the functional domain, the customer's needs of the customer domain are characterized in terms of FRs. In the physical domain, a physical solution is generated and the physical solution is characterized by DPs. Finally, in the process domain, manufacturing process is generated to satisfy DPs by finding appropriate process variables, PVs. Although the design process involves the process domain in the case of process design, in this thesis, only the customer domain, the functional domain and the physical domain will be considered.

The suggested domains and mapping between the domains are shown schematically in <Figure 2.1>.

Design

Design may be formally defined as the creation of synthesized solutions in the form of products, processes or systems that satisfy perceived needs through the mapping between the FRs in the functional domain and DPs of the physical domain, through the proper selection of DPs that satisfy FRs.

Design Process

The design process begins with the establishment of FRs in the functional domain to satisfy a given design need. Once the design need is formalized in the form of FRs, ideas are generated in the physical domain to create a product which satisfies the FRs. This product is then analyzed and compared with the FRs. When the product does not fully satisfy the specified FRs, one must generate a new idea or change the FRs to reflect the original need more accurately. This iteration process continues until the designer produces an acceptable result.

Therefore, the overall design process can be expressed as :

Step 1 Definition of FR.
Step 2 Ideation or creation of ideas.
Step 3 Analysis/Test of the proposed ideas.
Step 4 Checking the fidelity of the final solution to the original needs.

Design Equation and Design Matrix

Design is defined as a mapping process between the FRs in the functional domain and DPs in the physical domain. This relationship can be represented by a design equation as:

$$\{\mathbf{FRs}\} = [\mathbf{A}]\,\{\mathbf{DPs}\}$$

where {FRs} is a vector, the components of which are the functional requirements; {DPs} is a vector, the components of which are the design parameters; [A] is a design matrix of the design equation. If the elements of the {FRs} and {DPs} could be represented in the form of numbers, the elements of the design matrix, A_{ij}, are given by

$$A_{ij} = \frac{\partial\, FR_i}{\partial\, DP_j}$$

which must be evaluated at the specific design point in the physical domain. In the case that the variables which can represent the characteristics of the FRs or the DPs are not defined/developed quantitatively, a usual situation at the higher level of the design hierarchy, the components of the design matrix, A_{ij}, can be expressed qualitatively as :

$$A_{ij} = \begin{cases} X \text{ (if } FR_i \text{ is affected by } DP_j \text{)} \\ O \text{ (if not).} \end{cases}$$

Definition of Independence of FRs of a Design

When specific DP of a design can be adjusted to satisfy its corresponding FR without affecting other functional requirements which are already set by other DPs, it is said that the design maintains the independence of FRs.

Classification of Designs(or Design Equations)

According to the form of a design equation and the independence of FRs of the design, designs can be classified as :

Uncoupled Design

When the design matrix of a design is a diagonal matrix, the design is called an 'uncoupled design'. An uncoupled design can maintain the independence of FRs.

Decoupled Design

When the design matrix of a design is a triangular matrix or can be arranged to a triangular matrix by changing the order of FRs and DPs, the design is called a 'decoupled design'. A decoupled design can maintain the independence of FRs if the DPs are adjusted in a particular order. Considering the situation when the number of FRs and DPs is three and the design matrix is triangular, for an example, the design equation may be represented as :

$$\begin{Bmatrix} FR1 \\ FR2 \\ FR3 \end{Bmatrix} = \begin{bmatrix} X & O & O \\ X & X & O \\ X & X & X \end{bmatrix} \begin{Bmatrix} DP1 \\ DP2 \\ DP3 \end{Bmatrix} .$$

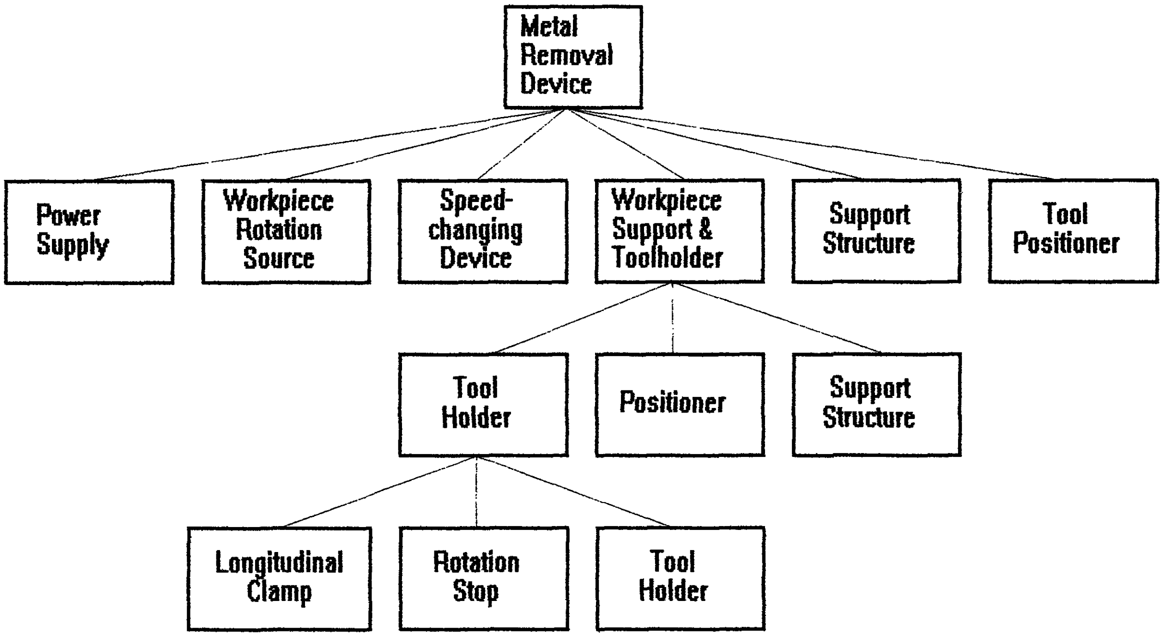
If we vary DP1 first , then the value of FR1 can be set. Although it also affects FR2 and FR3, we can then change DP2 to set the value of FR2 without affecting FR1. Finally, DP3 can be changed to set the value of FR3.

Redundant Design

When the number of DPs is greater than the number of FRs in a design equation and the design can maintain the independence of FRs, the design is called a 'redundant design'.

Coupled Design

If a design is not uncoupled, decoupled nor redundant, the design is called a 'coupled design'. A coupled design cannot maintain the independence of FRs.

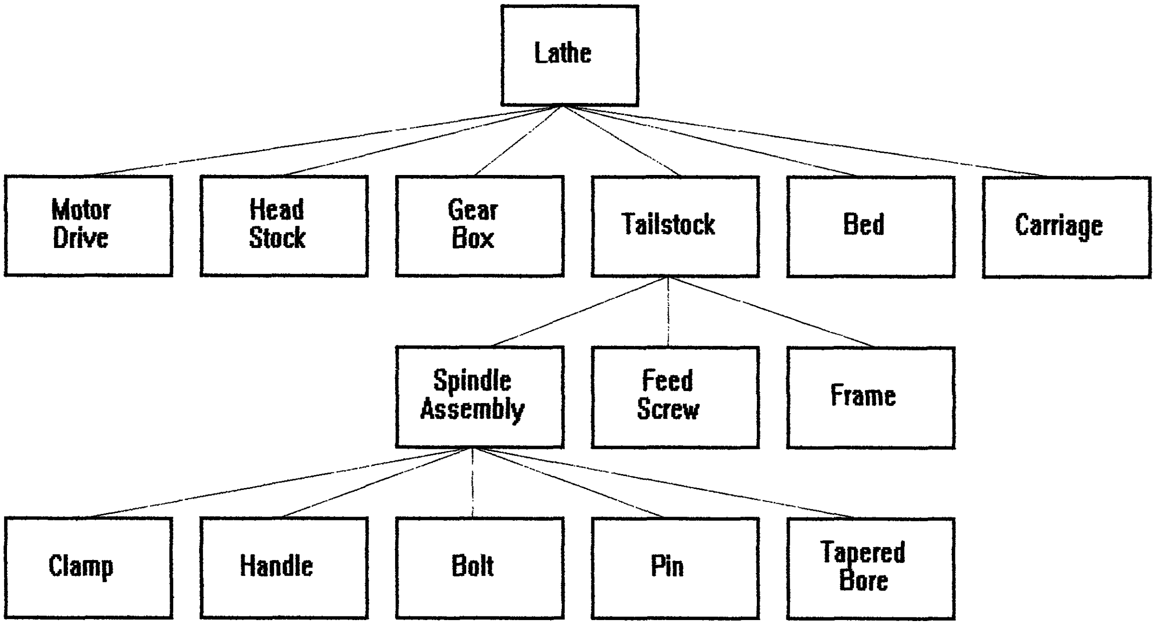


<Figure 2.2> Lathe Functional Hierarchy

Hierarchy of FRs and DPs

FRs and DPs have hierarchies, and they can be decomposed. An example of the functional hierarchy and the physical hierarchy are shown in the <Figure 2.2> and <Figure 2.3>.

However, the decomposition in each domain cannot be made independent of the hierarchies in other domains. That is, FRs at the *i*th level cannot be decomposed into the next level of the FR hierarchy without first going over to the physical domain and developing a solution that satisfies the *i*th level FRs with all the corresponding DPs. Therefore, we have to travel back and forth between the functional domain and the physical domain to develop the FR and DP hierarchies.



<Figure 2.3> Lathe System Physical Hierarchy

Information Content

Information content is defined as the measure of knowledge required to satisfy a given FR set at a given level of the FR hierarchy.

The knowledge required to achieve a task depends on the probability of success. If the physical entity which is generated to satisfy the given FR set has enough knowledge to achieve the given task, no more additional knowledge is required to achieve the task and the probability of success is unity. In some cases, more knowledge is required. That is, unless enough knowledge for the given task is embodied in the physical entity which is generated to satisfy the given FR set, some uncertainty will govern the physical entity. Therefore, the higher the information content is, the more uncertain behavior the physical entity shows. The one of the word of everyday language, complexity, is used when some requirements are hard to achieve with a system or a physical entity. Therefore, the complexity of a system or a physical entity may be, also, understood in terms of the probability of success for achieving functional requirements. In this sense, in axiomatic design theory, the information content can be used as the measure of the uncertainty, the complexity or the required additional knowledge of a system, and the probability of success to achieve the given functional requirements.

There can be various forms of definition of the information content as long as the definition illustrates this situation. One definition frequently used is :

$$I = \ln(1 / p)$$

where I is the information content and p is the probability of success to satisfy the given FR set by the specified DPs.

2.1.2. Design Axioms

A basic assumption of the axiomatic design theory is that there exists a fundamental set of principles that determines good design practice. The set of principles which determines what is good design is called 'axioms' or 'design axioms' in the axiomatic design theory. This assumption, the existence of the axioms, makes axiomatic design theory different from other design theories or design methodologies. The only way to refute these axioms is to uncover counter examples that prove these axioms to be invalid. Axioms may be hypothesized from a large number of observations by noting the common phenomena shared by all cases.

Two design axioms that govern good designs have been developed until now. Axiom 1 deals with the relationship between functional requirements and design parameters, and Axiom 2 deals with the complexity/uncertainty of designs. Theses axioms can be stated in a variety of semantically equivalent forms. The declarative form of axioms is :

Axiom 1 *The Independence Axiom*
Maintain the independence of FRs.

Axiom 2 *The Information Axiom*
Minimize the information content of the design.

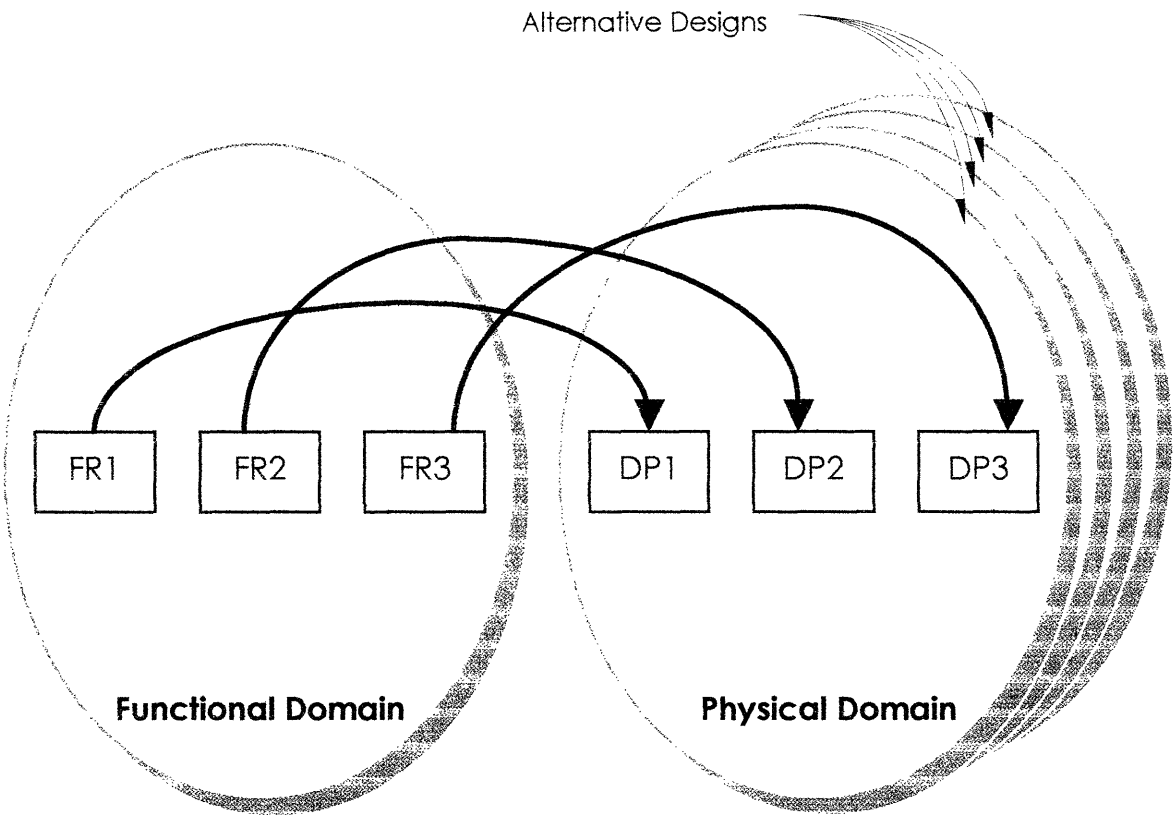
Axiom 1 states that one of the common elements of a good design is maintaining the independence of FRs. Therefore, coupled designs should be avoided. Axiom 2 states that, among all the designs that satisfy the independence axiom, the one with minimum information content is the best one. Therefore, designer must concentrate on maintaining the independence of FRs and minimizing the information content during his mapping process so that he may have a good design. The independence axiom sets the boundary where the optimal design exists and the information axiom provides the criterion for selecting the best design among the suggested designs.

Based on these design axioms, corollaries and theorems can be derived from other propositions which is proven to be true or accepted as empirical facts.

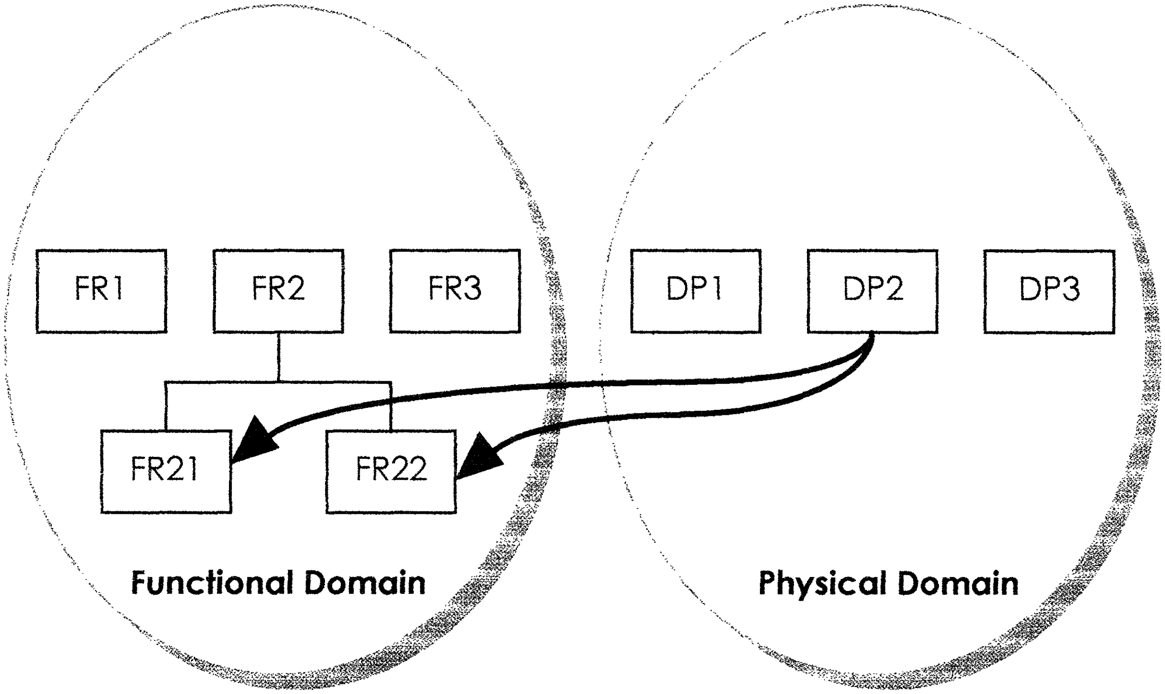
Some corollaries of the design axioms and theorems which were introduced until now can be found in Appendix B.

2.1.3. Methodology/Design Procedure

A design methodology/procedure, based on the basic facts and the design axioms which are explained in the previous sub sections, is introduced in this sub section. This methodology can be characterized by the use of the design axioms and the zig-zagging, mapping-decomposition process.



<Figure 2.4> The Mapping Process



<Figure 2.5> The Decomposition Process after Mapping Process

<Figure 2.4> shows the mapping process graphically. After the FRs are defined, designers must generate some ideas which can satisfy the FRs and express the ideas in terms of DPs. During this mapping process the independence axiom should be satisfied. To check the independence of FRs of the design, the designer needs to develop the design equation of the design.

In <Figure 2.5>, the decomposition process is shown schematically. After developing the higher level DPs, the designer is required to decompose some higher level FRs into the lower level FRs based on the developed DPs and the generated idea. It should be noticed that developing the lower level FRs without finding the higher level DPs is prohibited in this procedure.

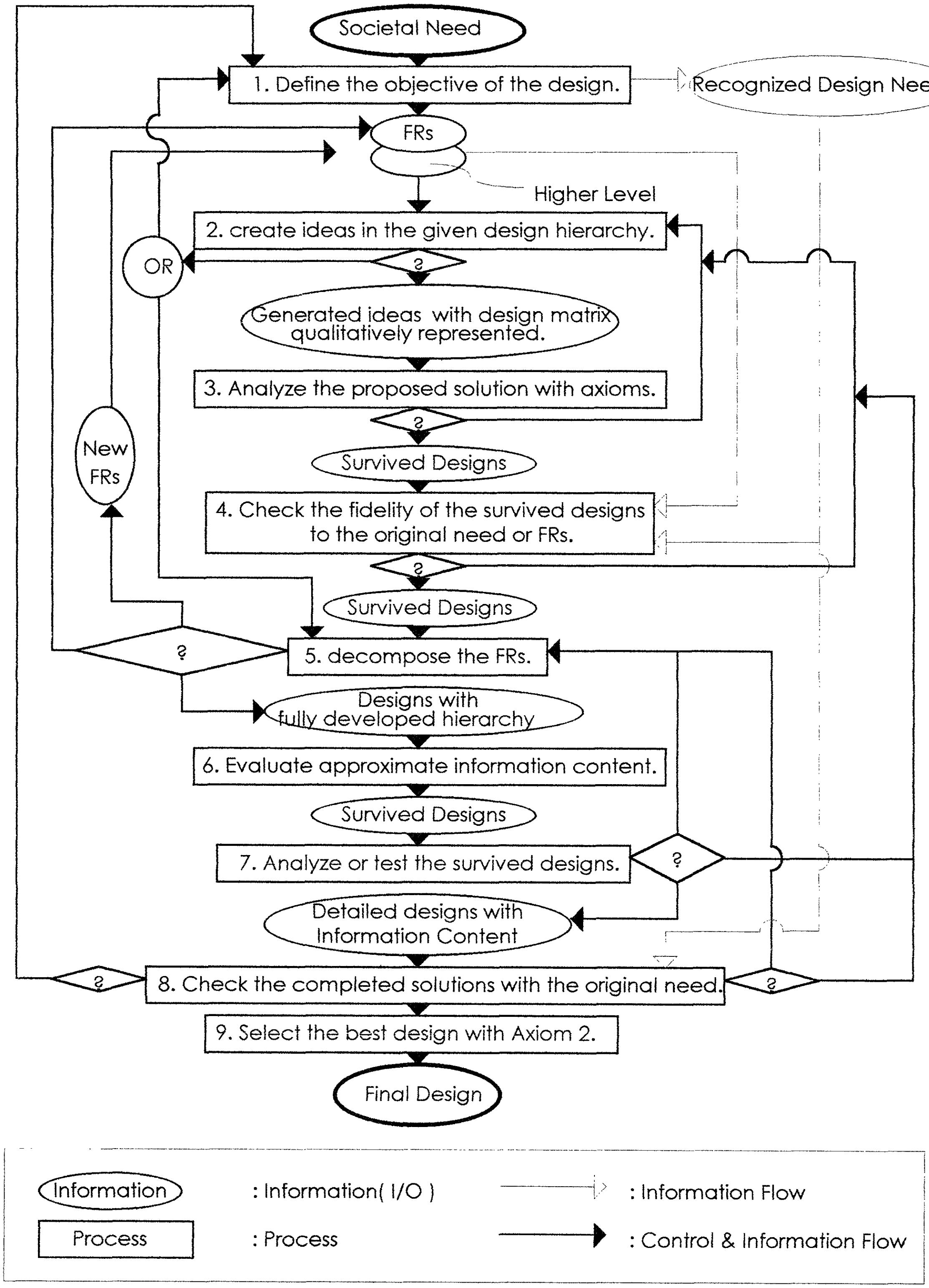
After the decomposition process, the mapping process occurs in the lower level hierarchy again.

Consequently, the overall design procedure is a zig-zagging process, jumping from the functional domain to the physical domain, from the physical domain to the functional domain and from the functional domain to the physical domain again. The decomposition process follows the mapping process and the mapping process follows the decomposition process again. This zig-zagging process continues until the generated DPs in that hierarchy were already fully developed and the relationship between the FRs and the DPs can be understood.

Finally, the information axiom is used to select the best design among the designs which satisfy the independence axiom. To apply the information axiom for the real situation, how to measure the information content of the design or the probability of success is important. The researches on this issue can be found in [1,3,5].

As a result, an overall procedure can be suggested as :

```
(1) Define the objective of the design.
   {
     (1-1) Describe the design need.
     (1-2) Specify the design need in the form of  FRs.
   }

(2) Create ideas in the given design hierarchy level.
   {

     (Option a)
      {
        (2a-1) Try to conceptualize the idea for satisfying the FRs.
                 If fail to get new idea, go to the step (2-3).
        (2a-2) Characterize the idea in terms of DPs.
      }
```

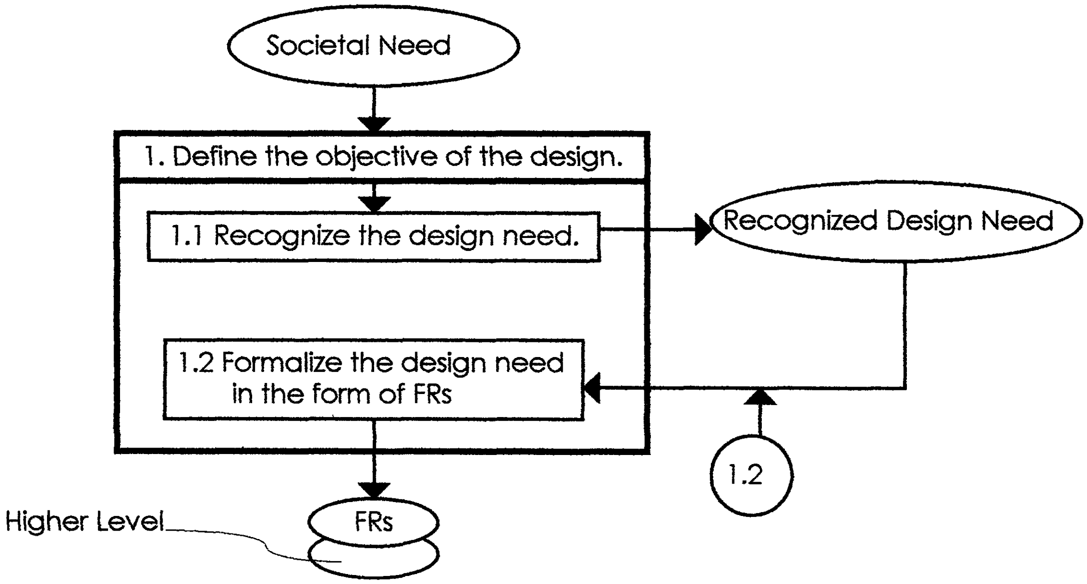

Societal Need
1. Define the objective of the design.
Recognized Design Need
FRs
Higher Level
2. create ideas in the given design hierarchy.
OR
?
Generated ideas with design matrix qualitatively represented.
3. Analyze the proposed solution with axioms.
?
New FRs
Survived Designs
4. Check the fidelity of the survived designs to the original need or FRs.
?
Survived Designs
?
5. decompose the FRs.
Designs with fully developed hierarchy
6. Evaluate approximate information content.
Survived Designs
7. Analyze or test the survived designs.
?
Detailed designs with Information Content
?
8. Check the completed solutions with the original need.
?
9. Select the best design with Axiom 2.
Final Design
Information
: Information(I/O)
: Information Flow
Process
: Process
: Control & Information Flow

<Figure 2.6> Overall Axiomatic Design Procedure

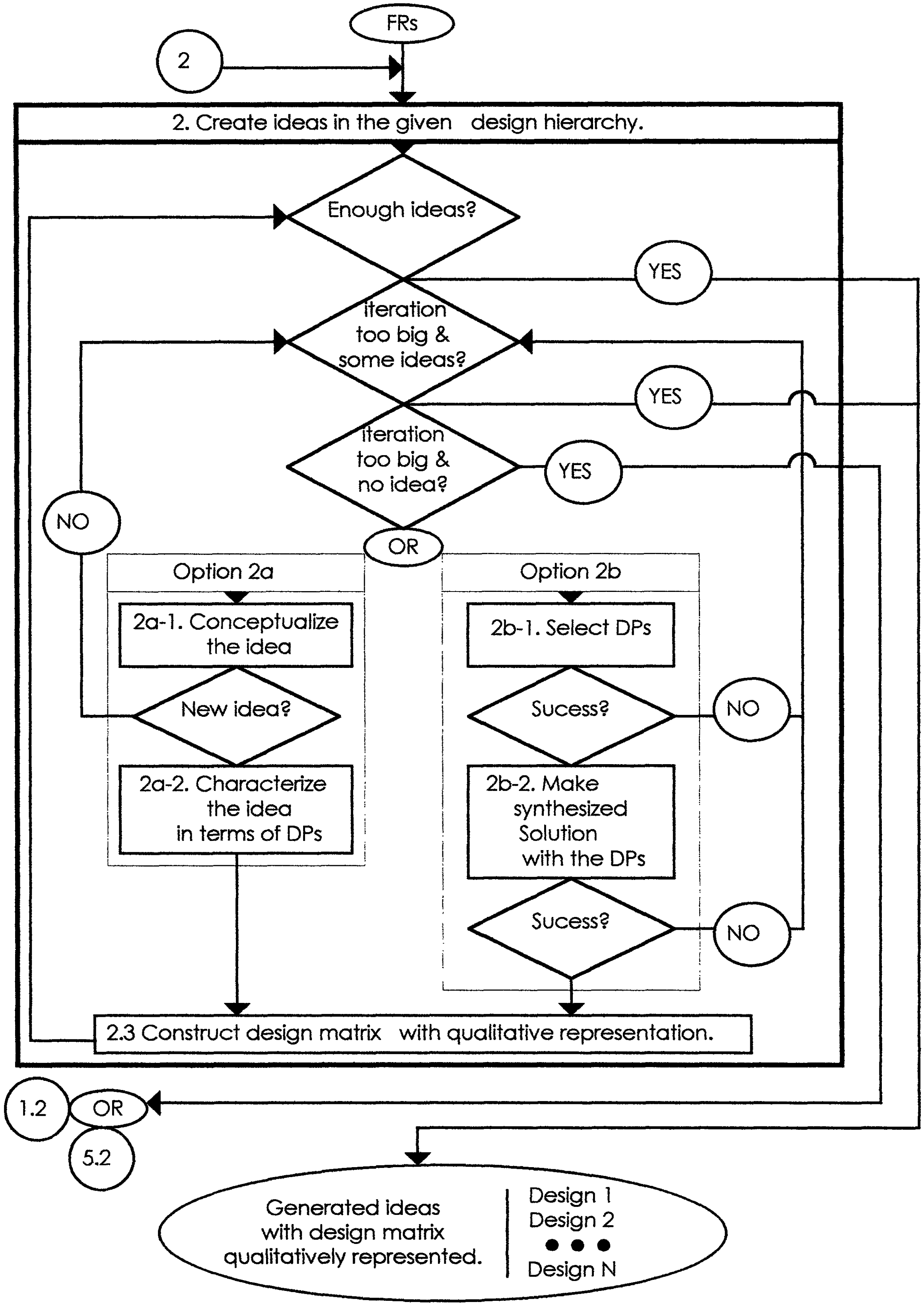
```
    (Option b)
    {
      (2b-1) Try to select the DPs which may satisfy the FRs.
             If fail to get new set of DPs, go to the step (2-3).
      (2b-2) Try to make the synthesized solution to satisfy FRs with the selected DPs.
             If fail, go to the step (2b-1).
    }

    (2-3) Construct the design matrices of the suggested designs. : The qualitative
          representation of the element of the design matrices is enough.
  }

(3) Analyze the proposed solution with axioms.
  {
    (3-1) Analyze the design matrices constructed in the step (2-3) with axiom 1. : Try to
          make the design matrices triangular or diagonal by rearranging the order of
          FRs and DPs.

    (3-2) Discard coupled designs.
    (3-3) Evaluate the approximate information content of the survived designs on
          relative basis. : Designer may use the corollaries or other reasoning based on
          the axioms.[1,4]
    (3-4) Select the designs which have reasonably small information content. :
          Practically, it is needed to restrict the number of the alternative designs to be
          considered at the same time due to the limitation of the memory of human
          brain or computers.
  }

(4) Check the fidelity of the survived designs to the original needs or the higher level FRs
  and DPs.
  {
    (4-1) Check whether the physical parts of the survived designs in the given hierarchy
          can fit to the higher level designs or the original needs.
    (4-2) If fail, discard the designs.
  }

(5) Decompose the FRs.
  {
    (5-1) Find the FRs to be decomposed in the survived designs.
    (5-2) Decompose the FRs.
    (5-3) If the decomposition occurs, go to the step (2), ideation, and do the same
          procedure to generate the next level hierarchy design through the mapping
          process.
    (5-4) If not, finish the decomposition process.
  }
```

```
(6) Evaluate the relative information content of the survived designs.
  {
    (6-1) Evaluate the relative information content of the survived designs like the step
          (3-3).
    (6-2) Discard the designs which have the high information content.
  }

(7) Analyze or Test the survived designs.
  {
    (7-1) Do the research on the design matrix, if the elements of the matrix, which
          represent the relationship between FRs and DPs, are not well known.
    (7-2) Decide the design region or the operating region of the DPs which ensure the
          independence of FRs.
    (7-3) Check the 'O' components of the design matrix.
          {
            (7-3-1) Check whether the 'O' components of the design matrix is really 'O'.
                    : In this checking process, the tolerance of FRs should be considered.
                      (Theorem 8)
                    : Practically, only the components which is critical to the
                      independence of FRs need to be checked.
            (7-3-2) If fail,
                    (Option a) Redesign with the given FRs(Step(2)).
                    (Option b) Backtrack to the higher level design and decompose
                               again(Step(5)).
                    (Option c) Discard the design.
          }
    (7-4) Test/Analyze the design to measure the information content of the survived
          design.
  }

(8) Check the completed design with the original needs.
  : If fail, consider new design or define new set of FRs.

(9) Select the best design which has minimum information content.
```

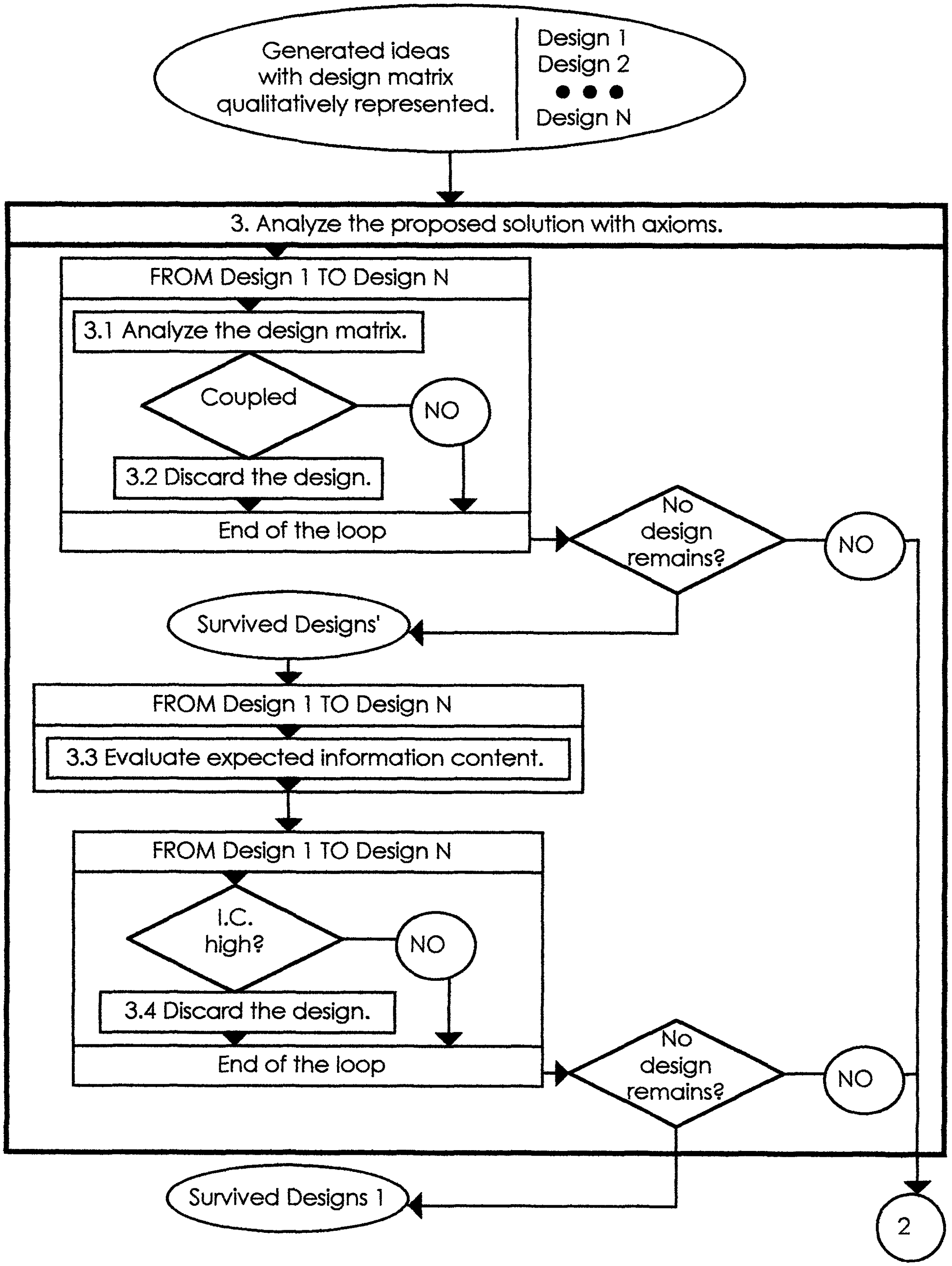
In <Figure 2.6>, the schematic diagram which represents this procedure is shown and the detailed procedures are shown in <Figure 2.7>.


Societal Need
1. Define the objective of the design.
1.1 Recognize the design need.
Recognized Design Need
1.2 Formalize the design need in the form of FRs
1.2
Higher Level
FRs

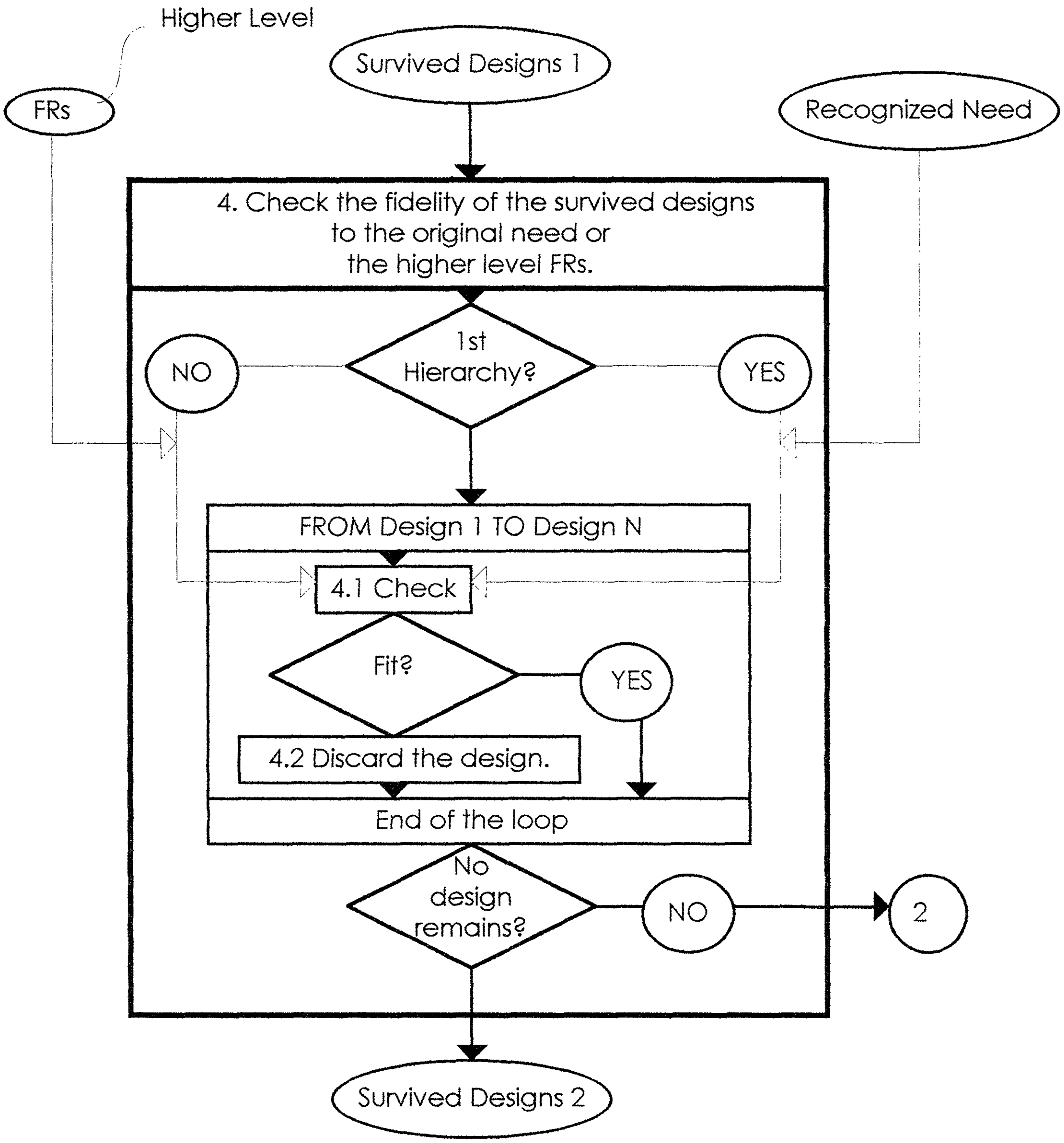
<Figure 2.7-1> Detailed Procedures of Axiomatic Design


FRs
2
2. Create ideas in the given design hierarchy.
Enough ideas?
YES
iteration too big & some ideas?
YES
iteration too big & no idea?
YES
NO
OR
Option 2a
2a-1. Conceptualize the idea
New idea?
2a-2. Characterize the idea in terms of DPs
Option 2b
2b-1. Select DPs
Sucess?
NO
2b-2. Make synthesized Solution with the DPs
Sucess?
NO
2.3 Construct design matrix with qualitative representation.
1.2
OR
5.2
Generated ideas with design matrix qualitatively represented.
Design 1
Design 2
● ● ●
Design N

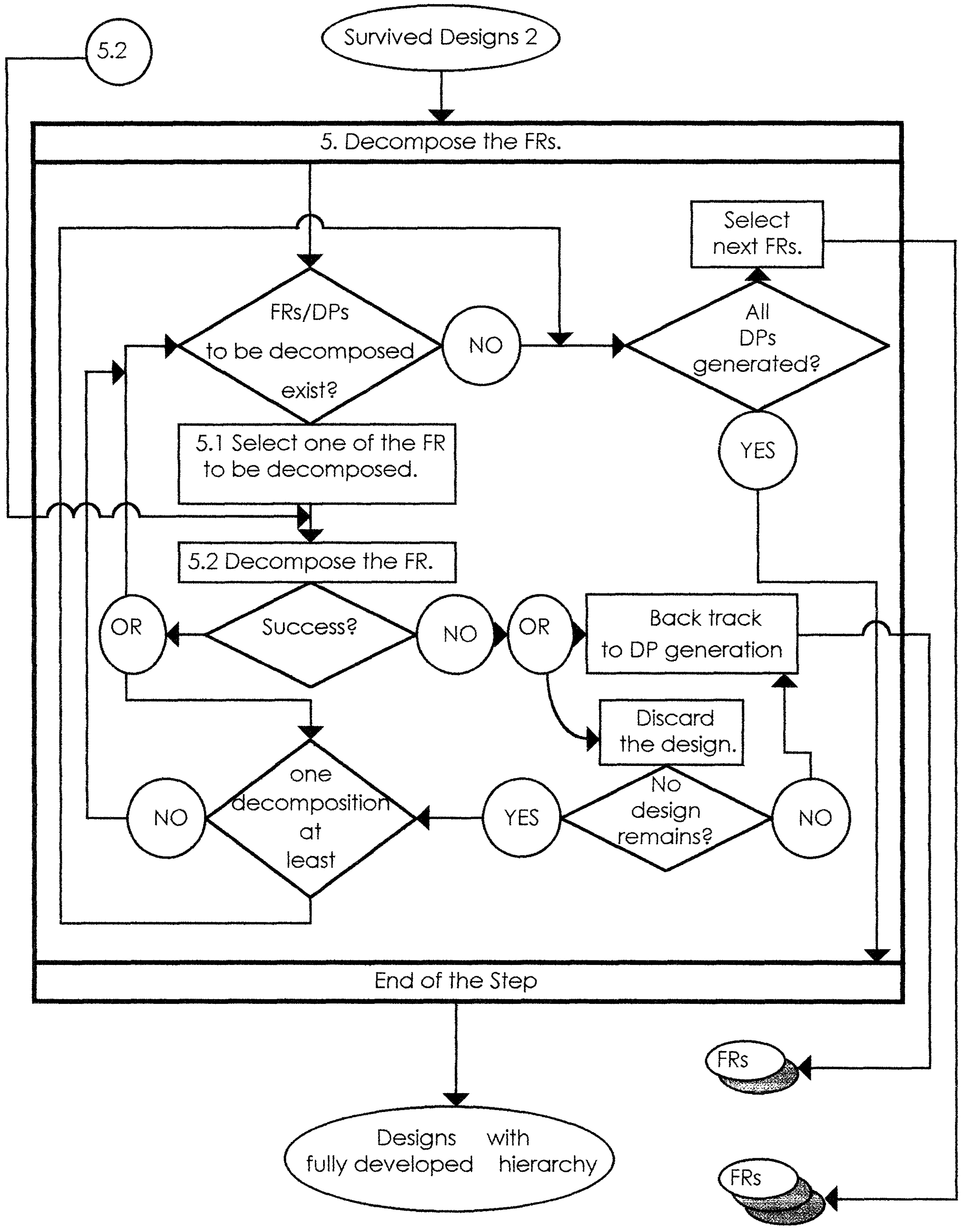
<Figure 2.7-2> Detailed Procedures of Axiomatic Design


Generated ideas with design matrix qualitatively represented.
Design 1
Design 2
● ● ●
Design N
3. Analyze the proposed solution with axioms.
FROM Design 1 TO Design N
3.1 Analyze the design matrix.
Coupled
NO
3.2 Discard the design.
End of the loop
No design remains?
NO
Survived Designs'
FROM Design 1 TO Design N
3.3 Evaluate expected information content.
FROM Design 1 TO Design N
I.C. high?
NO
3.4 Discard the design.
End of the loop
No design remains?
NO
Survived Designs 1
2

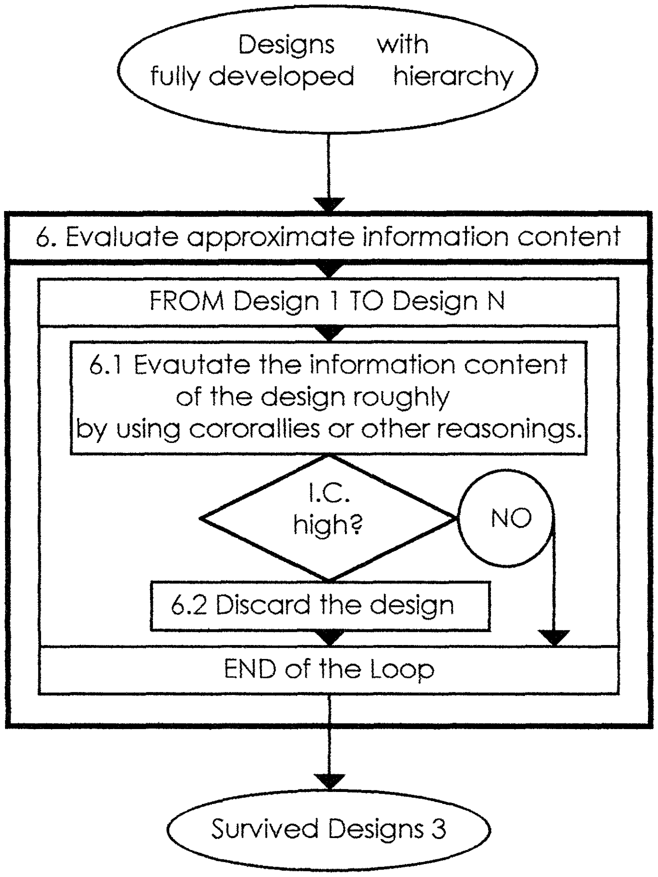
<Figure 2.7-3> Detailed Procedures of Axiomatic Design



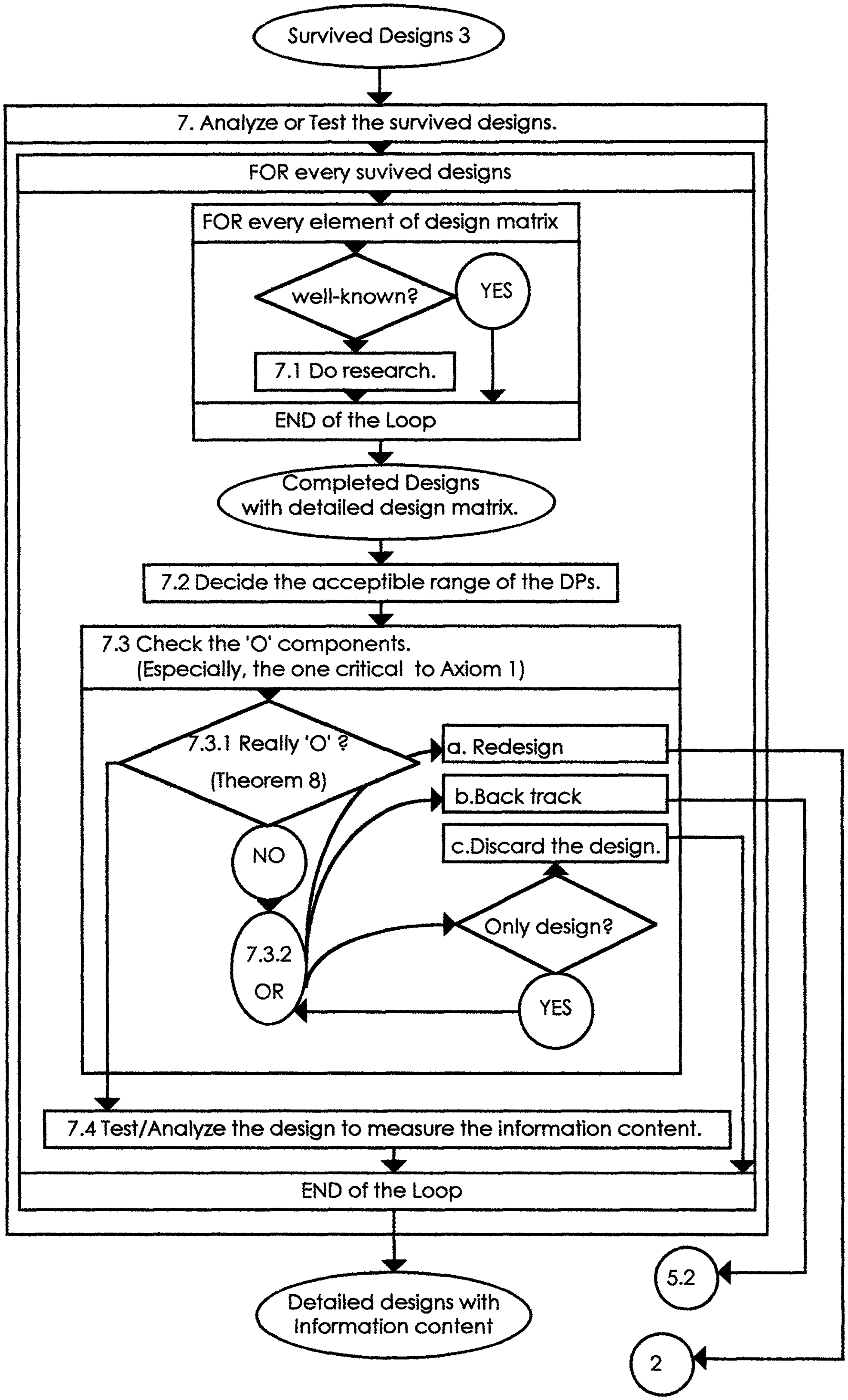
<Figure 2.7-4> Detailed Procedures of Axiomatic Design



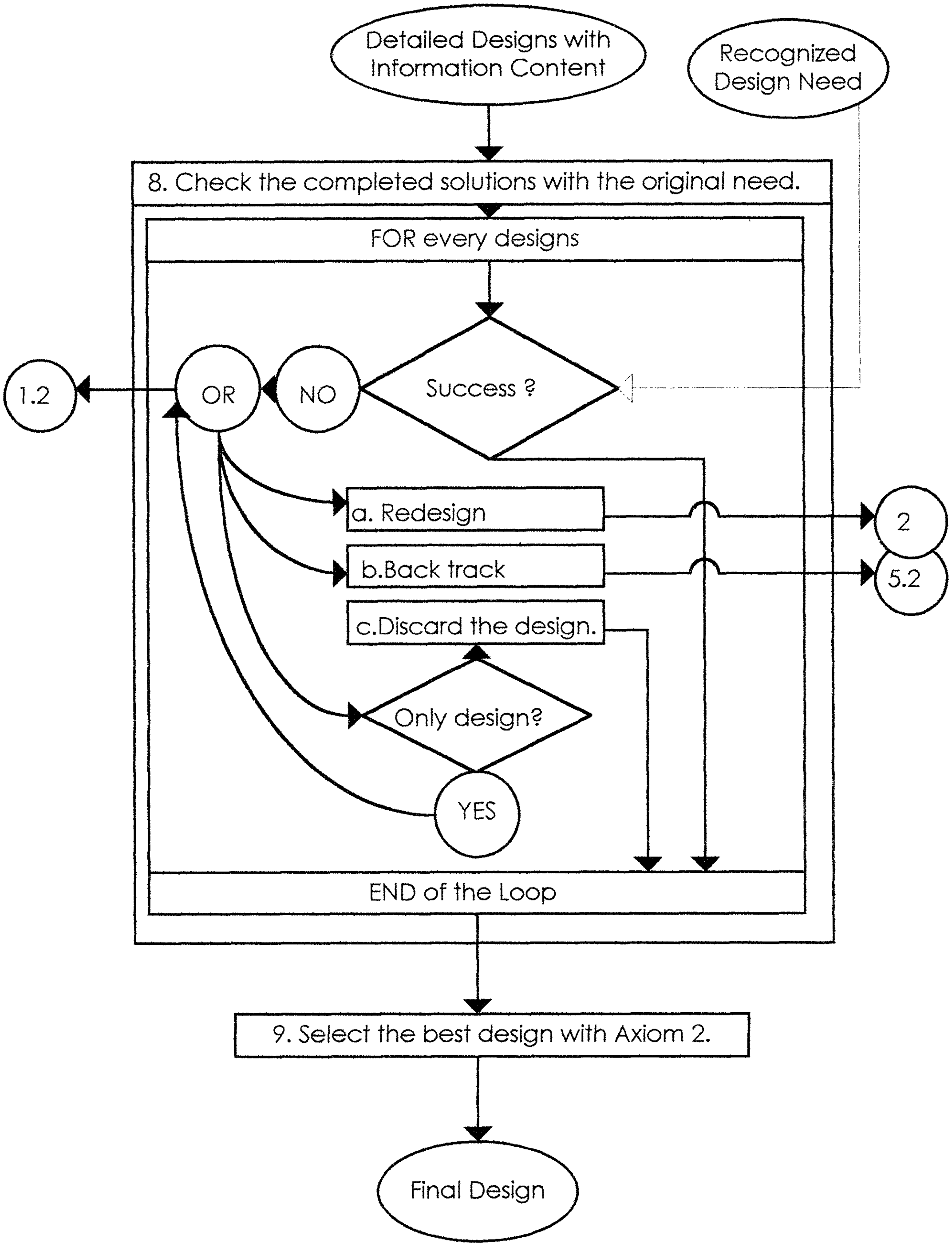
<Figure 2.7-5> Detailed Procedures of Axiomatic Design


Designs with fully developed hierarchy
6. Evaluate approximate information content
FROM Design 1 TO Design N
6.1 Evautate the information content of the design roughly by using cororallies or other reasonings.
I.C. high?
NO
6.2 Discard the design
END of the Loop
Survived Designs 3

<Figure 2.7-6> Detailed Procedures of Axiomatic Design


Survived Designs 3
7. Analyze or Test the survived designs.
FOR every suvived designs
FOR every element of design matrix
well-known?
YES
7.1 Do research.
END of the Loop
Completed Designs
with detailed design matrix.
7.2 Decide the acceptible range of the DPs.
7.3 Check the 'O' components.
(Especially, the one critical to Axiom 1)
7.3.1 Really 'O' ?
(Theorem 8)
a. Redesign
b.Back track
c.Discard the design.
NO
Only design?
7.3.2
OR
YES
7.4 Test/Analyze the design to measure the information content.
END of the Loop
Detailed designs with
Information content
5.2
2

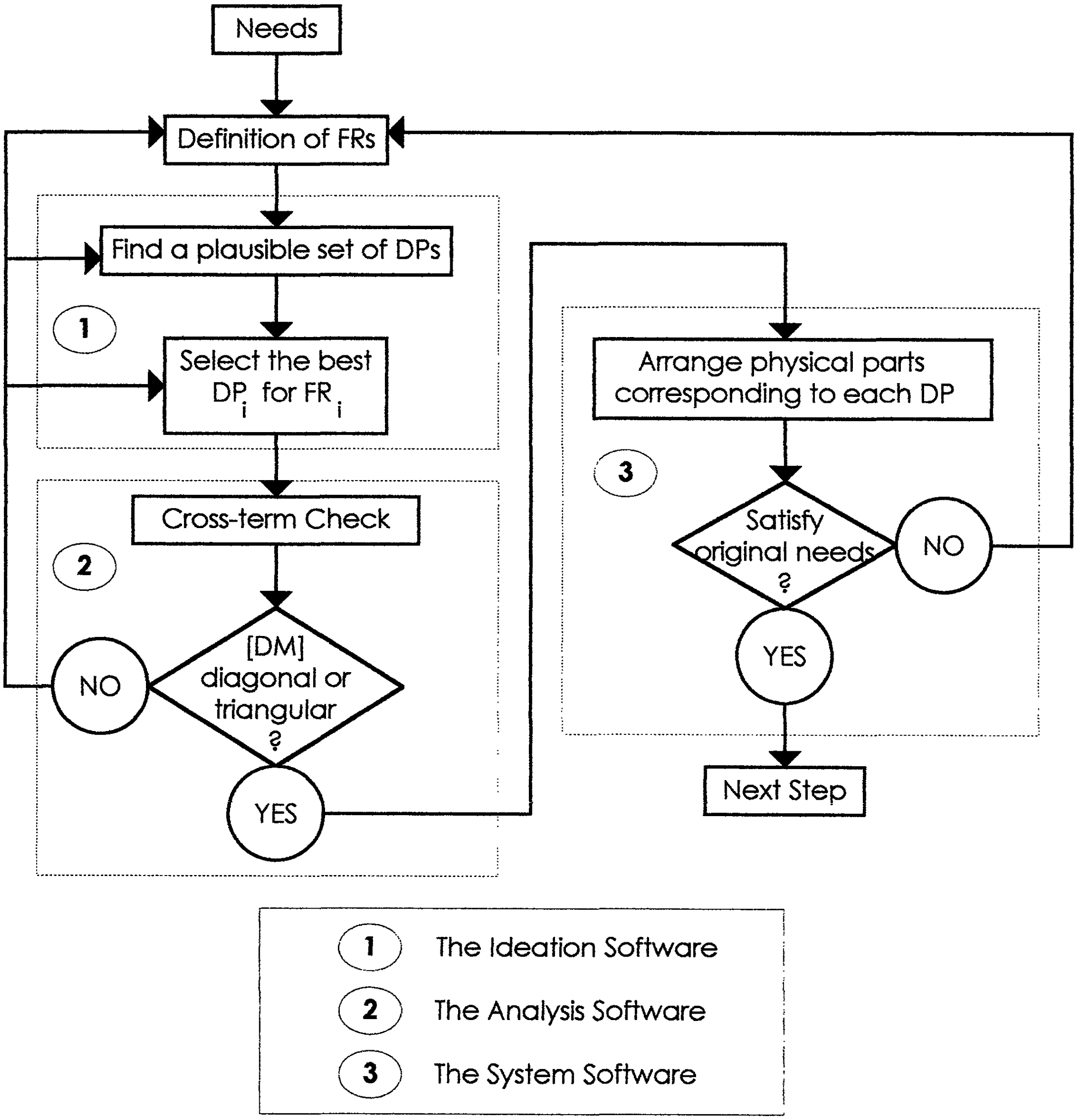
<Figure 2.7-7> Detailed Procedures of Axiomatic Design



<Figure 2.7-8> Detailed Procedures of Axiomatic Design

2.2. TDM (Thinking Design Machine)

In this section, the basic concept of TDM(Thinking Design Machine) is introduced.



<Figure 2.8> Block Diagram for TDM

Based on the axiomatic design theory, the concept of TDM was created by Suh[1,2]. The ultimate goal of design automation is to develop a "thinking design machine" which can create design or design concepts that are superior to those currently possible without the aid of such a machine. A thinking design machine may be defined as an intelligent machine system that can generate creative designs.

This research can be thought as a try for embodying the knowledge of the axiomatic design theory, especially the knowledge like the procedure which is suggested in the section 2.1.3, on computer. To embody the knowledge on computer so that the computer do the design task by itself, the procedure needs to be as detailed as the computer understand it. At a certain stage of detailing the procedure, the research will transit to the field of cognitive science or the artificial intelligence, because it should deal with how to create ideas or synthesized solutions and how to characterize the design with FR/DP concept. Until now, the very high level algorithms/procedures are suggested. The simplified block diagram for TDM is shown in <Figure 2.8>.

The TDM software is decomposed into three software modules ; the ideation software, the analysis software and the system software. The block diagram in <Figure 2.8> corresponds to the step 1,2,3,4 and 5 in <Figure 2.7>.

The ideation software selects a set of plausible DPs that correspond to the set of FRs chosen as a design task. The ideation software may contain a database in the form of morphological relationships between various FRs and DPs, from which appropriate DPs to satisfy given FR set can be selected. After the selection of the DPs, the ideation software constructs the design matrices.

The analysis software has the ability to check the independence of FRs of a given design, to detect the flaws of the design according to the axiom 1 and to suggest the improvement of the design. If the given design is coupled, the analysis software suggests the improvement and discards the design. The results of the analysis software are the uncoupled or decoupled designs which satisfy the given FR set.

The system software arranges the physical components chosen by the ideation software and the analysis software for each DP. The physical parts are arranged not necessarily according to the sequence of the DP control; the arrangement may be based on the sequence of flow of materials in a system, which may or may not be the same as the control sequence.

With the development of the computer ability, many computer aided design(CAD) systems have been developed to date. They usually concentrates on the geometric modeling and representation. As a result, they are lack of the ability to suggest design concepts and evaluate them in a systematic manner. Because TDM concept is based on the axiomatic design theory which provides the criterion for evaluating design concepts, it can help designers' decision making from the very early stage of the design process.

Even though very high level algorithms for TDM was suggested[1,2], much research should be done for the automation of the axiomatic design process.

CHAPTER 3. Axiomatic Design Environment (ADE)

In this chapter, concept of 'Axiomatic Design Environment' is suggested. The 'Axiomatic Design Environment' may be defined as a computer software which can help designers who try to follow the axiomatic design procedure which is explained in the section, 2.1.3. Even though TDM concept is suggested as a ultimate goal of design automation, the modules in the <Figure 2.8> needs to be detailed further to construct TDM. Detailing the modules should be preceded by the research into the creativity or the idea generation, which is not completed yet. In the mean time, computer software can play a role as an assistance tool which is named here as 'Axiomatic Design Environment (ADE)'.

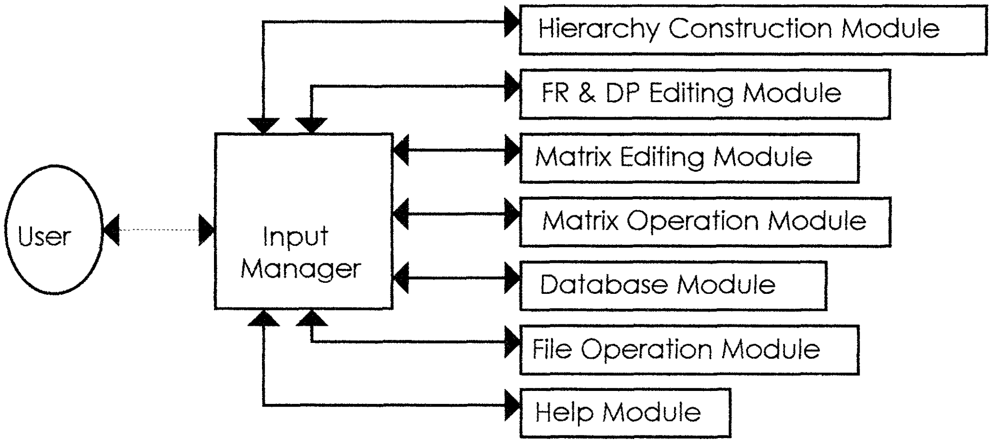
Even though the axiomatic design methodology is suggested in the section 2.1.3, the procedure is not so easy for the novice of the axiomatic design theory to follow. To follow the axiomatic design procedure, a designer is required to have the concrete understanding of the axiomatic design theory.

Furthermore, in the case of complicated system design, recording the design equations and checking the independence of FRs are not simple task.

In this sense, ADE is designed with the following specific goal in mind :

1. guide designers to follow axiomatic design procedure by prohibiting designers from taking undesirable design sequence,
2. give information about the design stage,
3. provide basic concepts which are axioms, domains, hierarchy, mapping and decomposition,
4. have the ability to record and retrieve a design equation,
5. have the ability to check the independence of FRs, and
6. help designers to generate design concepts.

<Figure 3.1> shows the schematic diagram which illustrates how ADE works.



<Figure 3.1> Structure of the ADE

Input Manager has the main control of the program. It detects user input and operates other modules according to the user input. If a user violates the axiomatic design procedure, input manager gives a warning and prohibits the attempt.

Hierarchy Construction Module has the ability to make FR and DP decomposition. This module should allow users to add FR and DP in the selected hierarchy, to eliminate a FR or a DP and to change the content of a FR and a DP. FR decomposition should be prohibited without constructing the higher level DP hierarchy and making decision on the higher level design among the alternatives.

FR & DP Editing Module gets the FR and DP from users. When this module is called, program should provide the information about the content of the higher level FR and DP so that a user can understand the design stage.

Matrix Editing Module gets the information about the design matrix elements from users. It is desirable that this module provides information about FR and DP which are related with the selected matrix element.

Matrix Operation Module has the ability to check the independence of FRs and rearrange the FRs and DPs.

Database Module operates database environment. It provides the data which can help designers to select the appropriate DPs.

File Operation Module has the ability to save a design and retrieve a design which is already stored in the computer disk.

Help Module has two functions. Help function provides the database which has the program usage and the explanation of the basic terminology. Hint function provides the information, which is related to the current stage of a designer, automatically.

Basically, ADE forces designer to follow the design sequence like below.

1. FR editing
2. DP editing
3. matrix editing
4. checking independence
5. selecting the best alternatives
6. decomposition

During this basic sequence, 'help module' and 'database module' can be called for assistance.

ADE, which is suggested in this section, leads designers to follow axiomatic design procedure, provides the knowledge of the axiomatic design theory and helps designers to find appropriate DPs.

CHAPTER 4. Axiomatic Design of ADE

In this chapter, how ADE code is generated following axiomatic design procedure is shown. The main purpose of this chapter is to formalize programming process using concepts of axiomatic design theory. What is FR and DP in computer programming and how programmer can follow zig-zagging process of axiomatic design methodology is shown. This approach can help program management by providing additional information about programmer's intention and revealing program structure more useful way, not just showing either the function call relationship or the relationship among objects.

The code was generated using the programming language, C++, one of the object-oriented programming language. For the readers who understand this specific programming language, the important code is presented in appendix A and the explanation is given with the terminology of C++ in this chapter. For the readers who do not have the knowledge about this programming language, the sentences which require the understanding of this programming language is written in italic characters. But readers are required to have, at least, the knowledge about structured programming. Although readers can understand the content of this chapter without understanding the sentences which is written in italic characters and the programming code given in appendix A, author inserted the sentences and the code for the readers who want to see what is the part of the code which corresponds to each step of axiomatic design procedure.

4.1. Small Example

In this section, a framework which can formalize software design process based on the axiomatic design theory is introduced with a simple example.

In the case of software design, FR can be thought of as a function of the structured programming. As an example, consider a sorting program. The program is required to read five integer numbers from the disk file, sort the numbers and print out the numbers in the sorted order. In this case, the program objective is satisfied if the next three tasks are satisfied.

FR1. read the data
FR2. sort the data
FR3. print the result

And, DPs can be thought of as information which is required to satisfy the given FR set. There can be two kinds of forms of information. One is the data type required to achieve a given FR and the other form is algorithm or detailed module/function which is needed to satisfy a given FR. Taking the above example, DPs can be specified like below.

DP1. the data in the input file
DP2. numbers in the sorted way
DP3. algorithm for printing function

The element of the design matrix is noted as 'X' when the corresponding FR makes a reference of the value of the corresponding DP or affects the value of the DP in the case that the DP has the data type form. In the case that the corresponding DP has the algorithm form, the 'X' in the design matrix means that the corresponding FR calls/uses the corresponding DP. In this example, the DP3 has the algorithm form.

The construction of the design matrix and selection of the DPs should be made at the same time because the same set of DPs can represent a different program structure. In other words, both the design matrix and the DP set represent a program code. In this example, author meant by the DPs that FR1(read data) open the file where DP1(input data) is specified, FR2(sort data) reads the value of the DP1 and sorts the data, and FR3(print the result) prints out the value of the DP2. This situation can be represented like below design equation.

$$\begin{Bmatrix} FR1 \\ FR2 \\ FR3 \end{Bmatrix} = \begin{bmatrix} X & O & O \\ X & X & O \\ O & X & X \end{bmatrix} \begin{Bmatrix} DP1 \\ DP2 \\ DP3 \end{Bmatrix}$$

The X's in design matrix can be thought as a part of the program which makes a link between FRs and DPs.

If either 'X' in the design matrix or the DP is too complicated to be coded at one level, the decomposition of the FR is needed and the mapping process occurs again at the next level of design hierarchy.

4.2. 1^{st} Hierarchy

TABLE 4.1 Design Equation of 1st Level

FR 1 : Help Axiomatic Designer. (o Provide Basic Concept. o Give hint about Axiom, Corollary. o Give hint about appropriate DP. o Help designer when using Theorems. o Lead designer in axiomatic manner. o Provide editing tool. o Provide ability to save/retrieve designs.)	X	**DP 1 :** Computer Program Package ADE.

In this hierarchy, FR is specified as 'Help Axiomatic Designer'.

This statement of FR is too ambiguous because of ambiguity of the word 'Help'. For example, a calculator can help a designer when he wants operation of numbers and a design catalog can help a designer when he tries to find out some mechanism which will be used in his design. In that sense, the meaning of the FR is explained in more detailed manner in the above parenthesis. In a sense the content of the parenthesis can be thought of as tolerances or constraints and in the other sense this can be thought of as just a part of FR. Frequently the statement of FR set cannot be finished just in one sentence. It contains many facts and concept and it contains some ambiguity. Therefore, it is necessary to explain the meaning of the statement. FR should be specified as in detail as designer/design group can fully understand the FR, especially in group design this fact is important. Each member of the group who is involved in that hierarchy should understand the FRs fully.

After this, DP was specified as 'Computer Program Package ADE', which is explained below :

Below are the explanation of the DP - Computer Program. It is the stage to determine program appearance.

Some decisions about program development

- Machine : DOS machine
- Interface : Menu system based on keyboard and mouse input
- Program can save designer's working file which contains his design decision.
- Program can open already existing file.
- Program allows designer to open new file.

Implementation of Basic Concept

- The concept of domain and hierarchy will be implemented by providing editing environment.
- The program allows selecting each domain - CA, FR, DP and give a designer notice which domain he is in.
- The program provides editing environment which can deal with hierarchy.
- The program should provide editing environment for CA, FR, DP.
- Customer domain is simplified to editor where designer can describe his design need.

Implementation of Design Axioms, Corollaries and Theorems

In this section, the program features related to axioms and theorems is specified.

- Include all axioms, corollaries, and theorems in HELP MENU item.

Axiom 1 : Independence Axiom

Maintain the independence of FRs.

1. Remind user that Independence of FRs is achieved by independent relationship between FRs.
   - Give Hint about this fact if menu item HINT is called when designer is in the DP domain and Mapping Window.
2. Provide Mapping Window.
   - Give CAUTION about design matrix state(coupled or not).
   - Matrix operation(or Re-ordering FR & DP).
   - Keep main diagonal.

Axiom 2 : Information Axiom

Minimize the information content.

1. Include definition and explanation of information content in the HELP
2. Encourage intuitive reasoning about information content in the given hierarchy.
   - After generating design solutions, give CAUTION for using information content.
3. Provide multiple design generation.
4. Give HINT about corollaries related to reducing information content.

Corollary 1 : Decoupling of Coupled Design

1. Include this corollary in HELP in Corollary section.
2. Give HINT when Coupled Design is generated.

Corollary 2 : Minimization of FRs

- Give HINT in FR domain.

Corollary 3 : Integration of Physical Parts

- Give HINT in DP domain.

Corollary 4 : Use of Standardization

- Give HINT in DP domain.

Corollary 5 : Use of Symmetry

- Give HINT in DP domain.

Corollary 6 : Largest Tolerance

- Give HINT in FR domain.

Corollary 7 : Uncoupled Design with Less Information Content

- Already achieved in Axiom 2 Part.

Corollary 8 : Effective Reangularity of a Scalar

- In high level, reangularity is not applicable.

- This program will not provide reangularity calculation.

Theorem 1 : Coupling Due to Insufficient Number of DPs

- Achieved by not allowing this situation.
- Considered in Axiom 1 part.

Theorem 2 : Decoupling of Coupled Design

- Do not allow less No. of DPs than No. of FRs.
- Give CAUTION about this fact.
- Provide working environment suggesting decoupled design. - This will be omitted in this first version.

Theorem 3 : Redundant Design

- Actually do not allow more No of DPs than No of FRs.
- Give CAUTION in this attempt.

Theorem 4 : Ideal Design

- Achieved already in Axiom 1, Theorem 2 and Theorem 3 part.

Theorem 5 : Need for New Design

- When designer try to add more FRs in the hierarchy which is already edited and marked as complete set, force designer to erase DPs related to that FR set.

Theorem 6 : Path Independency of Uncoupled Design
Theorem 7 : Path dependency of Coupled and Decoupled Design.

- Considered in Axiom 2 section.

Theorem 8 : Independence and Tolerance

- Give HINT in Mapping Window recalling that X and O mark should be made after considering tolerance.

Theorem 9 : Design for Manufacturability

- Allow PV domain.
- Show [A][B] matrix and determine manufacturability.
- Will not be achieved in this stage.

Theorem 10 : Modularity of Independence Measure
Theorem 11 : Invariance

- Related to reangularity.
- Excluded in this program.

Theorem 12 : Sum of Information
Theorem 13 : Information Content of Total System

- Give Explanation when 'Intuitive Reasoning HINT' is generated.

Theorem 14 : Information Content of Coupled versus Uncoupled Design

- Already considered.
- Just include this fact in HELP MENU

Theorem 15 : Design-Manufacturing Interface

- HINT in DP domain.

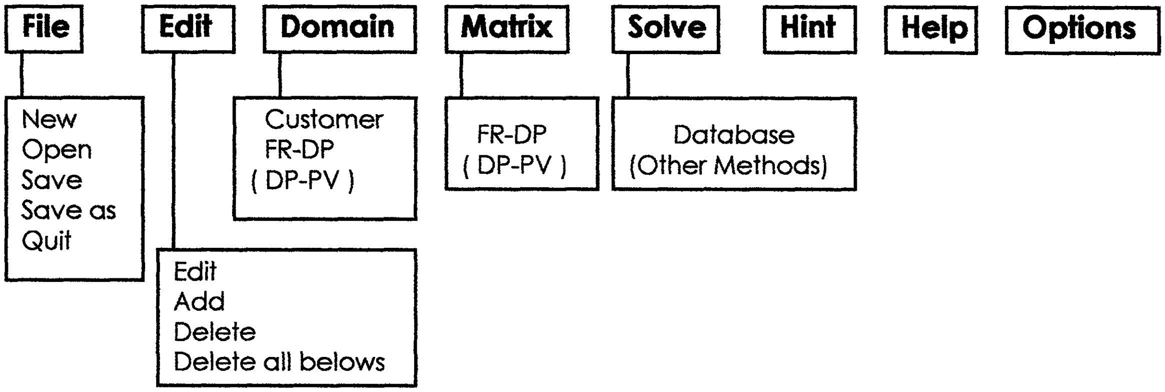
Theorem 16 : Equality of Information Content

- Remind designer this fact.
- Give designer HINT not to use weighting function when calculating Information Content.

Hint about appropriate DP

- Provide a database which links various FRs and design concepts.

Screen Display



<Figure 4.1> Menu Structure of the ADE

After above rough description of the program, the screen display was decided in each situation.

<Figure 4.1> shows the menu structure of the ADE. Main menu consists of eight items - (1)File, (2)Edit, (3)Domain, (4)Matrix, (5)Solve, (6)Hint, (7)Help and (8)Options. Each main menu item has its sub-menus as shown in <Figure 4.1>.

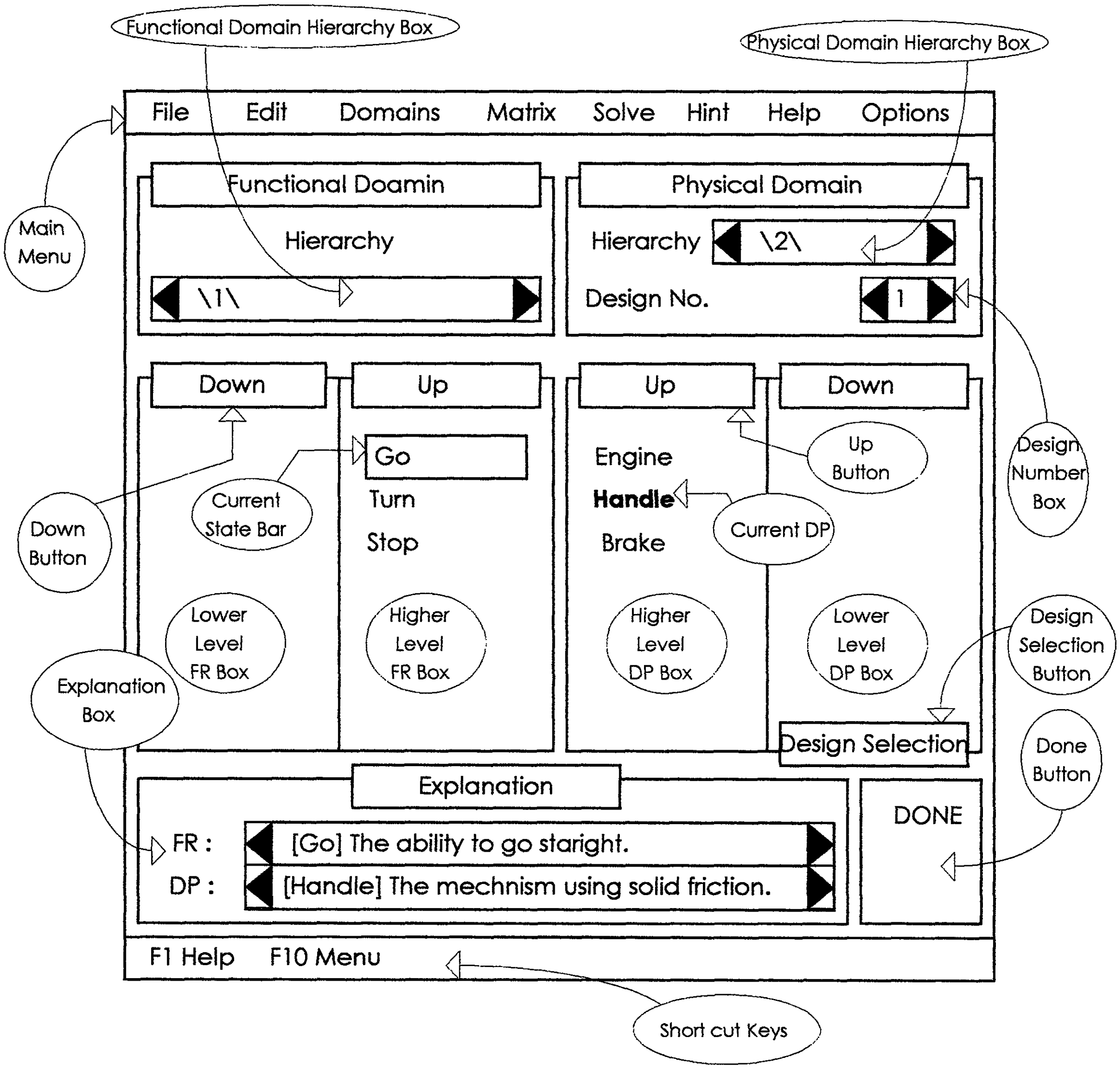
The main menu item, **File**, has the commands which are related to the disk file operations. If the New command is selected, ADE arrange the program environment so that a user may start a new design. The Open command is used when a user wants to load an already existing file for revision and addition. The Save command can be used when a user wants to save the currently-loaded design in a disk file. The Save as command saves the currently-loaded design in a disk file after imposing a new file name. The Quit command terminates the program and returns the control of the computer to the operating system.

The main menu item, **Edit**, is made up of the commands which are needed to construct the design hierarchy. The Edit command is used for the purpose of revising the content of either a FR or a DP. If the Add command is selected, the ADE provides the editing environment for typing a new FR or DP. After a user finishes typing the FR/DP, the program adds the FR/DP in the lower level of design hierarchy and shows the result. The Delete command eliminates a FR/DP from the design hierarchy. The Delete all bellows command differs from the Delete command in that the former eliminates all the FRs and DPs which are decomposed from a certain FR/DP.

The **Domain** menu item has the commands which provide the working environment of each domain.

The **Matrix** menu item has the commands which call the design matrix editing environment.

The **Solve** menu item provides the tools which help designers' idea generation.



FR : [Go] The ability to go staright.

DP : [Handle] The mechnism using solid friction.

<Figure 4.2> Editing Environment for FR & DP

Although only the database which contains links between various FRs and DPs is included in this first version of ADE, other methods can be added in this menu item.

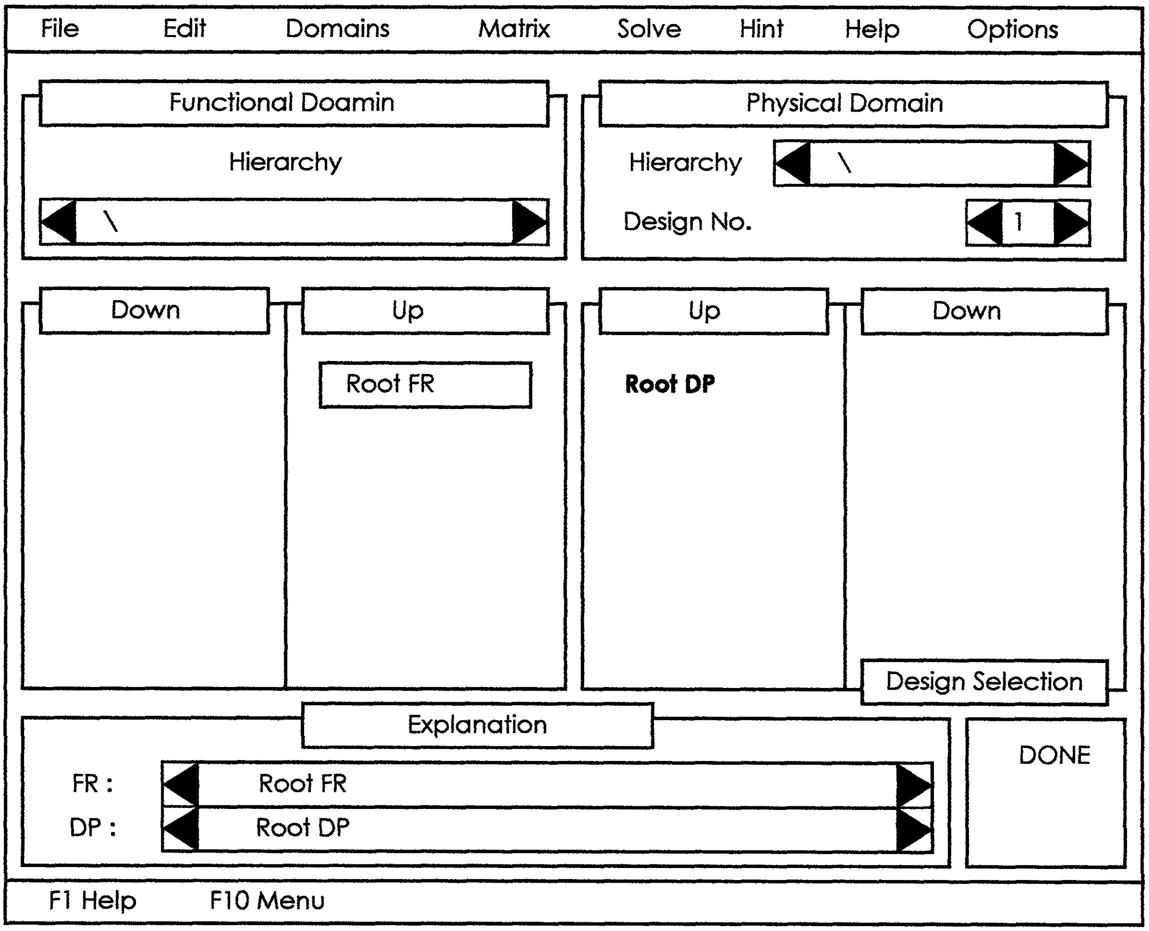
When the **Hint** menu item is called, ADE provides the information which can be used by the users in the current design situation.

The **Help** menu item provides the information related to both the program usage and the axiomatic design theory like a manual book.

The **Options** menu item can have the commands which can adjust the ADE to a certain user's preference.

Design process with ADE starts from describing the design objective in the customer domain. To do that, a user should select the Customer command in the **Domain** menu. If the Customer command is selected, ADE provides the editing environment for the customer domain. ADE prohibits a user from selecting other domains before customer domain is finished.

After describing the design objective, a user can select the FR-DP command in the **Domain** menu. This command provides the environment to make the design hierarchies



Hierarchy: \

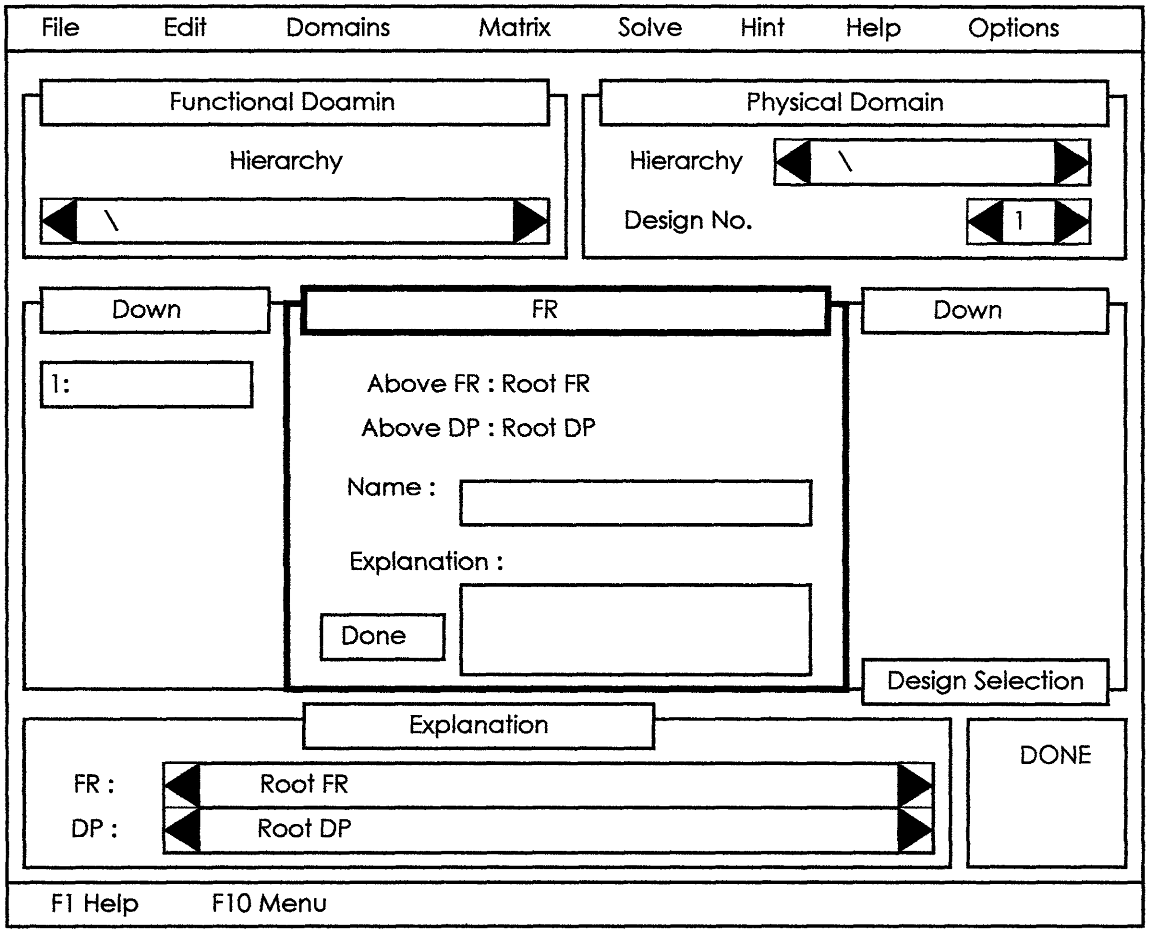
Hierarchy: \

Design No. 1

FR : Root FR

DP : Root DP

<Figure 4.3> Editing Environment for FR & DP at Initial State



Above FR : Root FR

Above DP : Root DP

Name :

Explanation :

FR : Root FR

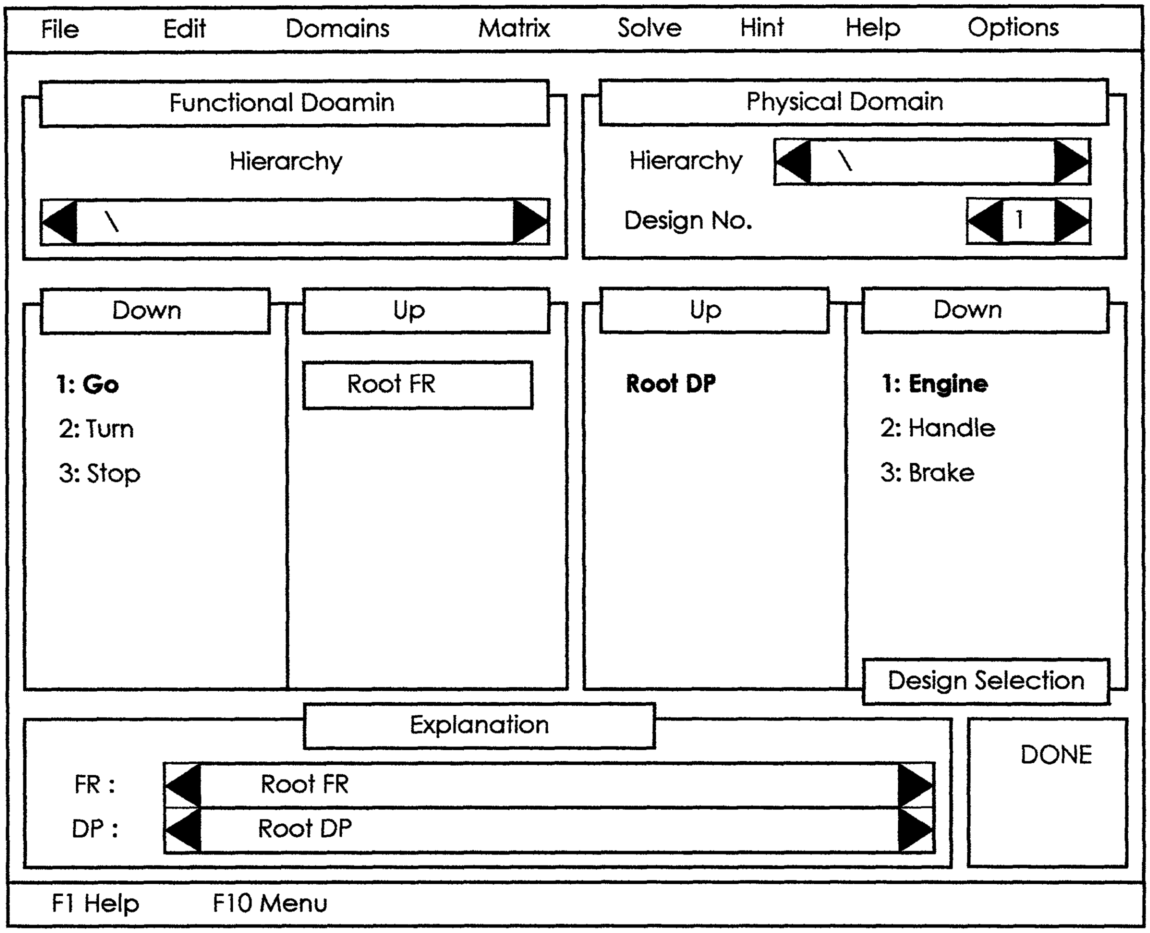
DP : Root DP

<Figure 4.4> FR Editing Environment

in the functional domain and physical domain like <Figure 4.2>.

The hierarchy boxes display the series of numbers which indicates the hierarchical position of the current FR and DP of the higher level FR/DP box. The explanation box displays the content of the current FR and DP. The current state bar is used to indicate the FR or DP which a user concerns. As the current state bar moves by user inputs, the hierarchy boxes and the explanation box are updated according to the movement. The FR/DP in the higher level FR/DP box whose decomposition is shown in the lower level FR/DP box is indicated by a different color. The Up/Down button is used for moving up and down in the design hierarchy.

If a user uses the New command in the **File** menu and selects the FR-DP command in the **Domain** menu after editing customer domain. The computer screen will be like <Figure 4.3>. To define FRs, It is required to select the Add command in the **Edit** menu. By the Add command, the program provides the editing environment as shown in <Figure 4.4>. A user can define a short name in the name box for the FR and he can also describe the FR in the explanation box as long as he wants. In the same way, the other FRs and DPs can be defined. After that, the screen will be like <Figure 4.5>. ADE prohibits users



\

\

1

1: Go
2: Turn
3: Stop

Root FR

Root DP

1: Engine
2: Handle
3: Brake

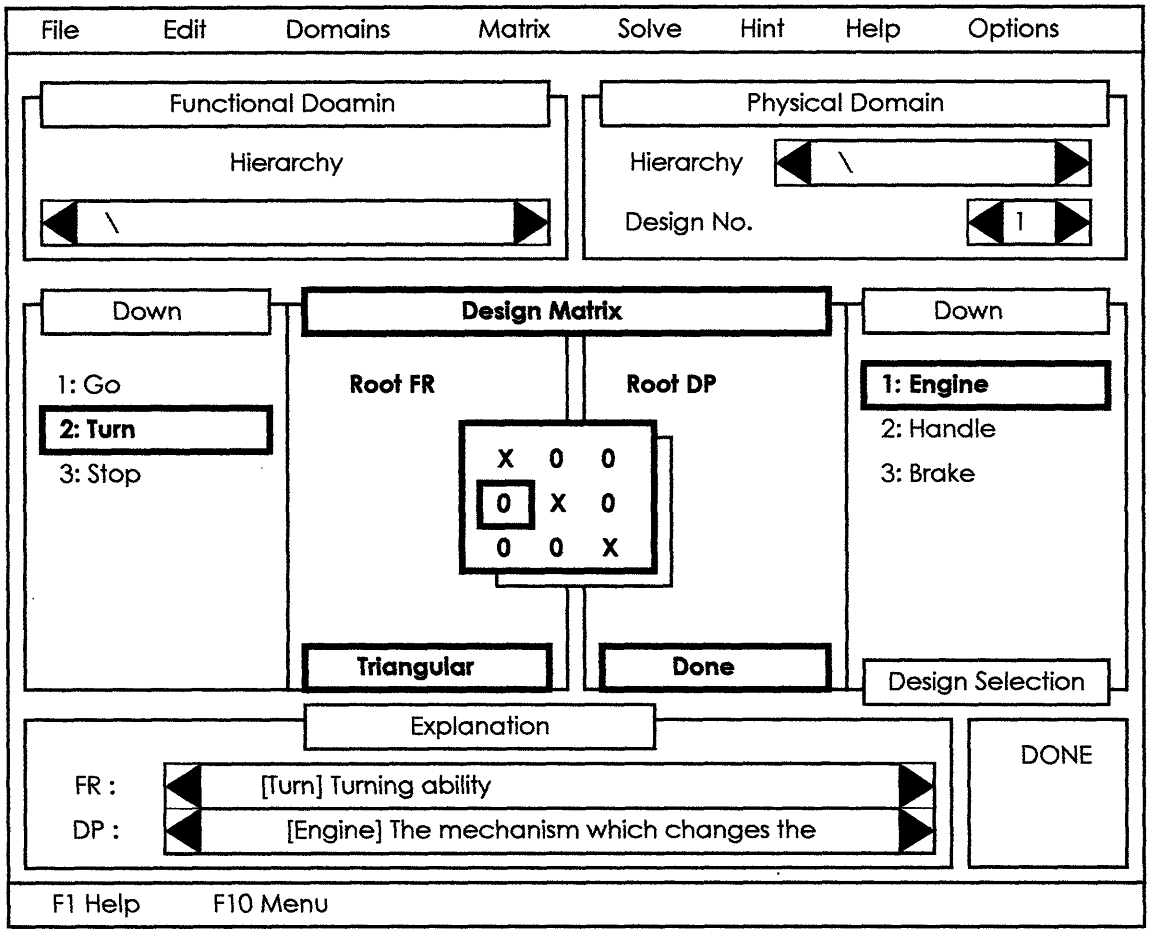
FR : Root FR
DP : Root DP

<Figure 4.5> The Screen Display after editing FRs and DPs

from defining more DPs than FRs. The arrow boxes in the design number box is used to change the lower level DP box for editing alternative designs.

After editing FRs and DPs, a user is allowed to define the design matrix. If the FR-DP command in the **Matrix** menu is selected, ADE provides the editing environment for the design matrix as shown in <Figure 4.6>. A user can decide the design matrix moving to each element of the design matrix. As a user moves in the design matrix editing environment, ADE highlights the FR and DP which are related to the design matrix element. If the Triangular button is selected in this situation, ADE rearrange the order of the FRs and DPs so that the design matrix has the form of triangular matrix.

After editing the design matrix, a user should decide which design he want to decompose. This is done by selecting the design selection button. If the button is selected, ADE changes the color of the design number so that it can indicate the selected design. Selecting a design is prohibited if the design matrix of that design is not decided.



1: Go
2: Turn
3: Stop

X 0 0
0 X 0
0 0 X

1: Engine
2: Handle
3: Brake

FR : [Turn] Turning ability
DP : [Engine] The mechanism which changes the

<Figure 4.6> The Editing Environment for Design Matrix

If a user selects the down button after selecting the design, ADE shows next level of the design hierarchy as shown in <Figure 4.3>. Moving to the next level of design hierarchy is prohibited if the design which will be decomposed is not selected.

In that way, design hierarchy can be developed to the terminal FRs and DPs.

Although more detailed screen display should be decided in this stage of the design process, it was not presented in this thesis.

Since the meaning of the DP is clear after deciding screen display, this level of design can be finished.

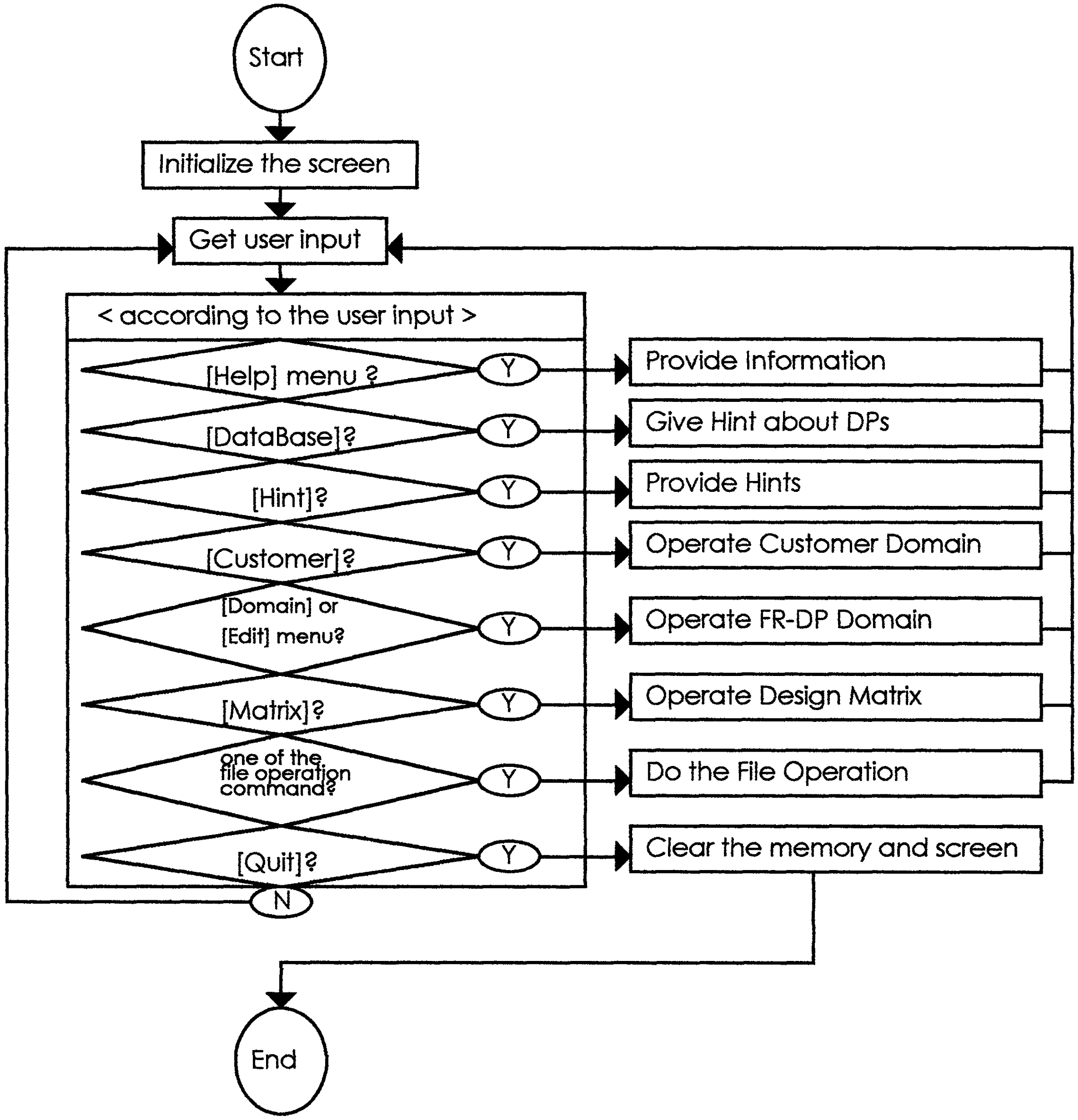
4.3. 2nd Hierarchy

TABLE 4.2 2nd level FRs

Functional Requirements	Explanation	*Explicit Expression in C++*
FR1 Provide HELP.	o Provide information about axioms, corollaries, theorems and how to use programs.	o void Help(void)
FR2 Provide Database.	o This database links various FRs and DPs.	o void DataBase(void)
FR3 Provide HINT.	o Provide information according to the position of designer.	o void Hint(void)
FR4 Operate Customer Domain.	o By providing editing environment.	o void CaDomain(int&)
FR5 Operate FR-DP Domain.	o By providing editing environment & operation of hierarchical structure.	o void FrDpDomain(void) o void Edit(void) o void Delete(void) o void DeleteAll(void) o void Add(void)
FR6 Operate Design Matrix.	o Provide matrix input environment. o Change matrix to triangular matrix.	o void FrDpMatrix(void)
FR7 End	o Arrange Screen to DOS. o Arrange Memory.	o void END(void)
FR8 File-related funtions	o Save current design. o Read existing file. o Create new file. o Assign New Name.	o void Save(void) o void Open(void) o void New(void) o void Saveas(void)
FR9 Initialize.	o Arrange memory for use.	o void Initialize(void)

Decomposition

Based on the above hierarchy decision, 1st level FR was decomposed into 2nd level FRs in TABLE 4.2. Of course, if decomposed FRs are satisfied, original FR should be satisfied. This means that designer should have the concrete and fixed method to satisfy original FR by the decomposed FRs. In the case of computer program code design, the programmer should have the algorithm which can combine decomposed FRs. In <Figure 4.7>, the simplified block diagram for the algorithm which combines the decomposed



<Figure 4.7> 2nd Level Decomposition

FRs is shown. The block, [Get user input], is assumed to be designed before. Other blocks need to be developed further.

In addition, the code which represents this algorithm was generated(See Appendix A1). *In that code (*Appendix A1*) the functions which need to be decomposed further were indicated by the symbol '/**/'.(See* FILE : TDM.CPP *in* Appendix A1). *The other functions were fully decided/defined.* If the undefined functions/blocks are defined, the program will work and the original need will be satisfied. From the other point of view, we can say that original need should be represented by the combination of some basic elements.

This decomposition process is not unique, because there can be many algorithms which can satisfy original need. But as long as the original need satisfied, it is acceptable.

TABLE 4.3 2nd level DPs

Design Parameters	Explanation	*Explicit Expression for Members in C++*

DP1
HELP Data CLASS
	o Have the information about files which have the explanation about axioms, corollaries, theorems and how to use programs.	o static char *HeaderFileName;
		o int NoOfMenu;
		o MenuType *M;
		o char *FileName;

DP2
Database File Written CLASS
	o Have Data members which can read database file.	o static char *HeaderFileName;
		o int NoOfMenu;
		o MenuType *M;
		o char *FileName;

DP3
Hint CLASS
	o Base number of file.
	o Number of files in that situation.
		o int Base;
		o int n;

DP4
CA CLASS
	o Whether or not CA domain edited.
	o Pointer of the Text
		o int AssCa;
		o char *Cp;

DP5
FrDp CLASS
	o Pointer to Roots
	o top left corner
	o top right corner
	o higher level
	o lower level
	o Current Position
	o Cursor Position in Hierarchy Box & Explanation Box
		o Unit *RootFR,*RootDP;
		o Unit *TopLeftFR ,*TopLeftDP;
		o Unit *TopRightFR,*TopRightDP;
		o Unit *AboveFR ,*AboveDP;
		o Unit *BelowFR ,*BelowDP;
		o UnShort CurrentSection;
		o int Hp[3],ExplSp[3];

DP6
MatrixScreen CLASS
	o Current Position
	o Top-left Position
		o Point Mp;
		o Point TopLeftMp;

DP7 END Function	o Do not require further data. o Decided not to be decomposed.	o void END(void);

DP8
File CLASS
	o Current file name.
	o File pointer
	o Whether or not file assigned.
		o char *FileName[N];
		o File *FileP;
		o int AssFile;

DP9 Initialize Function	o Do not require further data. o Decided not to decompose.	o void Initialize(void);

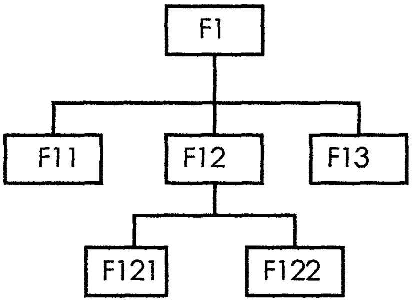
The top-down function tree which is usually used in structured programming is not equivalent to the functional requirement hierarchical structure. As an example, let's consider below situation.

Function **F1** *calls* Functions **[F11,F12,F13].**

Function **F12** *calls* Functions **[F121,F122].**

Above expression directly represents a function tree structure like <Figure 4.8>. But, functional requirement structure can be

F1 *can be decomposed into* **[F11,F121,F122,F13].**
F1 *can be decomposed into* **[F11,F121].**
etc.



<Figure 4.8> A Function Tree Structure

The FR decomposition is valid as long as designer can decide/define the other part of the program.

Selecting DPs

The corresponding DPs are explained in TABLE 4.3.

In the case of program design, DPs can be thought of as information which is needed to satisfy funtional requirements. For example, the information about wheather or not the customer domain was edited before and the information about the position of the cursor of the customer domain editor are needed to satisfy the **FR4**(=Operate Customer Domain). In that sense, the information is assigned as **DP4**.

In the first column of TABLE 4.3, the terminology 'class' was used to give names to DPs, *but this does not mean that the class which has the name will be generated.* That terminology was used just to make it clear that DP(or information) can be either data or algorithms. *Actual object was generated after making a full design matrix.*

The DPs (required information for FRs) can be expressed as either data members or member functions (which are usually called methods in object-oriented programming). In structured programming, DPs can be either data type/structure or defined function. *In the last column of TABLE 4.3, what the DPs explicitly mean was indicated in C/C++ programming language. In object-oriented programming, message philosophy is supported, which means that data members can not be changed directly by functions which are in the outside of that object. As a result, the part of DP which are represented by data type can be converted to the form of member functions if programmer use that philosophy thoroughly, because data member will not be known outside the object and only the functions which is operated by the object will be known to outside. This results in the fact that all the DPs can be member functions. However, for the purpose of identifying the relationship between FR and DP, DP was represented in the form of data type.*

TABLE 4.4 Design Matrix for 2nd Hierarchy

	DP1	DP2	DP3	DP4	DP5	DP6	DP7	DP8	DP9
FR1	X11								
FR2		X22							
FR3			X33						
FR4			X43	X44					
FR5			X53	X54	X55				
FR6			X63		X65	X66			
FR7					X75		X77		
FR8				X84	X85			X88	
FR9			X93	X94	X95				X99

Design Matrix

The design matrix corresponding to these FRs and DPs was generated in TABLE 4.4.

The **X** in the design matrix(TABLE 3.4) means that the corresponding FR uses the corresponding DPs or makes references/changes of the value of the DPs. *In object-oriented programming, the* ***X*** *means that the FR gives message to that DP.* If **X** has the meaning like above explanation, design matrix represents relationship between functions and the information which is required to achieve the functions. Here, information has the form of either data or algorithm.

The process of deciding design parameters and the design matrix should be done at the same time as explained in the section 4.1. To achieve an acceptable design (diagonal or triangular matrix), some iterations may be needed. Ideally, designer should try to have the DP set which can result in an acceptable design for a given FR set. But if it is too hard to achieve an acceptible design with the given FR set, it can be allowed to decompose again the higher level FR-DP to get new FR set because decomposition process is not unique one.

Setting DP Value

In the axiomatic design procedure, there are the step to set DP value to make the design entity have the acceptible performace which is expressed in the form of FR. The first axiom means that the design whose FRs can be set in independent way is an acceptible design.

In the case of software design, this step can be thought as the step to define the expression of the DP in a specific programming language and clarify the meaning of the DPs. The meaning of DPs is known by the overall context of the design. In other words, programmer should decide how each function which is declared as FRs deals with the DPs.

TABLE 4.5 Class Definition & Design Matrix Element

Class Name	**Matrix Element**
HelpClass	X11
DataBaseClass	X22
HintClass	X33
CaClass	X33 X43 X44
FrDpClass	X33 X43 X44 X53 X54 X55
DoMatrixClass	X33 X43 X44 X53 X54 X55 X63 X65 X66
DoFileClass	X33 X43 X44 X53 X54 X55 X63 X65 X66 X84 X85 X88
END () in TDMCLASS(Super Class)	X75 X77
Initialize() in TDMCLASS(Super Class)	X93 X94 X95 X99

If a program is as simple as the example in the section, 4.1, this step looks very evident. In contrast, as the software requirement is complicating, this step turns out to be a hard task. Design matrix is useful at this stage of design because design matrix indicates the sequence of the decision making.

After the decision making, it is desirable to define code to a degree so that the code can reveal why the **X** was checked in design matrix, i.e., how the DP set can be connected to the given FR set. The code which represents this situation was generated in Appendix A2. *File TDM.HPP, which has the class definition of TDMCLASS and was defined at the decomposition process, was redefined in more detailed manner to represent this situation.*

This representation of the relationship between function and information is different from language to language. In a sense, programming languages have evolved by developing the grammar which can express this relationship.

For example, in C++, this relationship can be declared in the class definition. *But we cannot say one design matrix represents only one object structure. The information in the design matrix can help designer by just preventing undesirable class definition. Of course, this representation of relationship among objects is not unique one and other representations are possible. In TABLE 4.5, the relationship between the declared class and corresponding **X**'s in design matrix is shown. In this case, most of this process was achieved by using inheritance characteristic of C++ language.* Even though the class definition which can exactly corresponds to the design matrix in TABLE 4.4 is desirable, the class in the code shown in Appendix A2 does not corresponds to the TABLE 4.4 because of some limitation in the grammar of the language. TABLE 4.6 shows the design matrix which the actual code represents. The matrix elements in bold characters are the elements which are not intended by the designer. Designer should be careful of such

TABLE 4.6 Design Matrix which Class Definition represents

	DP1	DP2	DP3	DP4	DP5	DP6	DP7	DP8	DP9
FR1	X11								
FR2		X22							
FR3			X33						
FR4			X43	X44					
FR5			X53	X54	X55				
FR6			X63	**X64**	X65	X66			
FR7	**X71**	**X72**	**X73**	**X74**	X75	**X76**	X77	**X78**	
FR8			**X83**	X84	X85	**X86**		X88	
FR9	**X91**	**X92**	X93	X94	X95	**X96**	**X97**	**X98**	X99

Bold Characters : The Element which is undesirable/not-intended but compiler cannot detect and so the designer should be careful of.

elements because the compiler will not give warnings about the mistakes on those elements. That is, class declarnation in Appendix A2 represents the design matrix in TABLE 4.6 but designer intended the design matrix in TABLE 4.4. Programmer should be careful not to violate the design matrix in TABLE 4.4 when he makes details of the functions because the *(C++)* compiler will not give warnings when he violates the design matrix.

Decomposition to the Next Level

If how the FRs deal with the DPs is decided, the role of the functions declared as FRs has the concrete meaning which is expressed in terms of data structure and algorithm. After that, the functions can be coded in a specific programming language. In this step, the meaning of the **X**'s in the design matrix is clarified in a specific programming language.

In some cases, it is difficult to express the meaning of the **X**'s at once and the decomposition of the FR is needed.

Since FR7 and FR9 was decided not to be decompose further, the function END() and Initialize() are defined fully in this level (*See* Appedix A2).

4.4. 3rd Hierarchy

In the same way, designer can decompose continuously until he can decide whole code.

In this section, the decomposition of **FR4-[DP3,DP4]** and **FR5-[DP3,DP4,DP5]** was presented as an example of decomposition. In Appendix A3, the code which represents this decomposition is shown. When decomposed, **FR4** was found not to call any functions which is unknown(*See* FILE : TDM.HPP, TDM_CA.CPP, CA.HPP, CA.CPP in Appendix A3). This means that **X43** and **X44** can be coded without further decomposition. Hence, only

FR5 was decomposed and is shown in TABLE 4.7. *As same as in Appendix A1, '/**/' symbol used to indicate that the indicated functions are functional requirements.*

TABLE 4.7 Decomposition of [FR5]

Functional Requirement	Explanation	Explicit Expression in C++
FR5/1 Initialize FR-DP Domain Screen.	o Draw Boxes. o Set Cursor Bar. o Fill Boxes with Information.	o void InitializeFdWholeScreen (...)
FR5/2 Check whether or not mouse is one of FRs or DPs.	o when Left Button clicked.	o Bool MouseOneOfUnit(...)
FR5/3 Change State Values	o according to mouse position when Left Button clicked.	o void DoMouseChange(...)
FR5/4 Reprint Structure Box.	o when needed.	o void InitFdScr(...)
FR5/5 Do Actions according to mouse position when one of Arrow Boxes chosen.	o Hierarchy Boxes. o Explanation Boxes. o Text Move.	o void DoArrowBoxes(...)
FR5/6 Do Actions when Done Box chosen.	o Arrange Screen. o Return Control.	o void DoFdEnd(...)
FR5/7 Do Actions when Design Select Box chosen.	o Screen Arrange o Set State Values.	o void DoDesignSelect(...)
FR5/8 Do Actions when Left Key chosen.	o Screen Arrange o Set State Values.	o void DoFdLeft(...)
FR5/9 Do Actions when Right Key chosen.	o Screen Arrange o Set State Values.	o void DoFdRight(...)
FR5/10 Do Actions when Up Key chosen	o Screen Arrange o Set State Values.	o void DoFdUp(...)
FR5/11 Do Actions when Down Key chosen	o Screen Arrange o Set State Values.	o void DoFdDown(...)
FR5/12 Provide FR Editing Environment.	o Name Box o Explanation Box o Done Box o Provide Above FR. o Provide Above DP.	o ... DoEditFR(...)
FR5/13 Provide DP Editing Environment.	o Name Box o Explanation Box	o ...DoEditDP(...)

	o Done Box o Provide FR. o Provide Above FR. o Provide Above DP.	
FR5/14 Find Real Above DP		o Unit *FindRealAboveDP(...)
FR5/15 Find Real Above FR		o Unit *FindRealAboveFR(...)
FR5/16 Provide Ability to deal Unit class of FR & DP.		o Member Functions of Unit class

4.5. Conclusion

In this chapter, one framework for designing computer software was developed taking ADE program package as an example, based on axiomatic design methodology. Though this framework was explained in C++ language - one of the objected-oriented programming languages, it can be easily applied to other programing languages.

In this framework, FRs can have the form of functions and DPs can have the form of either functions or data types.

The **X**'s in design matrix menas that the corresponding FR(function) either calls the corresponding DPs(if they have the form of functions) or make changes/references of the value of DPs(if they have the form of data type). Deciding DPs and design matrix should be done at the same time.

When programmer has the concrete algorithm to achieve higher level FR by decomposed FRs, the decomposition is valid.

The procedure for software design can be like below.

Step 1. Define FRs : Declanation of functions with the algorithm to achieve original need or higher level FR.
Step 2. Find appropriate DPs and decide corresponding design matrix.
Step 3. Make codes which represent design equation.
Step 4. Decide how FRs deal with the DPs.
Step 5. Make codes which express the meaning of the step 4.
Step 6. Decompose FRs if neccessary.

Appendix A : Important Program Codes List

APPENDIX A1 :

Decomposition of [1st Hierarchy] → [2nd Hierarchy]

FILE : MENUNO.H

```
// Typical Number of Menu---------

 #define __MENU_ESCAPE       -1
 #define __MENU_NEW           0
 #define __MENU_OPEN          1
 #define __MENU_SAVE          2
 #define __MENU_SAVEAS        3
 #define __MENU_END           4
 #define __MENU_EDIT          5
 #define __MENU_ADD           6
 #define __MENU_DELETE        7
 #define __MENU_DELETEALL     8
 #define __MENU_CA_DOMAIN     9
 #define __MENU_FR_DP_DOMAIN 10
 #define __MENU_MATRIX       11
 #define __MENU_DATABASE     12
 #define __MENU_HINT         13
 #define __MENU_HELP         14
 #define __MENU_OPTION1      15
 #define __MENU_OPTION2      16

// End Of FILE : MENUNO.H---------
```

FILE : NOMENU.H

```
// NO OF MENUS ----------------------------------------
 #define NO_OF_MAIN        8 // No Of        Main Menu
 #define NO_OF_SUB        17 // No Of         Sub Menu
 #define NO_OF_BOTTOM      5 // No Of Bottom Line Hint
// End Of FILE : NOMENU.H ---------------------------
```

FILE : TOP.CPP

```
#ifndef __TDM_HPP
 #define __TDM_HPP
 #include "TDM.HPP"
#endif

void main()
{
 TDMCLASS TDM;
 TDM.GenerateShell();
}

// End of FILE : TOP.CPP ---------------------------------
```

FILE : TDM.HPP

```
#ifndef __CAUTION_HPP
 #define __CAUTION_HPP
 #include "CAUTION.HPP"
#endif
          // GENERAL SCREEN,MENU,EDITOR + CAUTION

#ifndef __DO_VAL_H
```

```
 #define __DO_VAL_H
 #include “DoVal.h”
#endif
         // FILE WHICH HAS DOMAIN VALUE DEFINITION

#ifndef __MENUNO_H
 #define __MENUNO_H
 #include “MenuNo.h”
#endif
         // FILE WHICH HAS MENU NUMBER DEFINITION

#ifndef __NOMENU_H
 #define __NOMENU_H
 #include “NoMenu.h”
#endif
         // FILE WHICH HAS NO OF MENU DEFINITION


class TDMCLASS : virtual private CautionClass
{
 private : // Static Data Const
          static MenuType     Main[NO_OF_MAIN  ];
          static MenuType      Sub[NO_OF_SUB   ];
          static UnShort       Key[NO_OF_SUB+1 ];
          static BarType    Bottom[NO_OF_BOTTOM];
          static int        Count [NO_OF_MAIN+1];
          static Point      MPoint[NO_OF_MAIN  ];

           // Static Data Variable
          static int                    InitMenu;
          static Bool                  YesInvoke;
          static Bool                   YesCheck;
          static int                    __choice;

 private : // RELATED TO DATA STRUCURE

          static UnShort Domain;

 private :
          void SetScreen    (void); // TDM_SCR .CPP

          Bool Logo         (void); // TDM_LOGO.CPP
          Bool GoStop
             (Point,int /* color*/,int /* rev */ );

          void EndEarly     (void); // TDM_EN_E.CPP
          void MemoryError  (void); // TDM_ME_E.CPP

          void DealEditMenu(void); // TDM_DE_E.CPP
 public   :

          void GenerateShell(void);
};

// End Of FILE : TDM.HPP----------------------------------------
```

FILE : CAUTION.HPP

```
#ifndef __EDIT_HPP
 #define __EDIT_HPP
 #include "edit.hpp"
#endif
         // GENERAL LIBRARY CLASS FOR EDIT, MENU, SCREEN CONTROL.
         // OF COURSE, THIS CLASS CAN BE ONE OF DPs, IN 2ND HIERARCHY.
         // BUT, I ASSUMED, THIS CLASS ALEADY DEVELOPED.

class CautionClass : virtual private Edit
{
 public :

 void Message(    char *);
 void Message(int,char *);
};

// End Of FILE : CAUTION.HPP ---------------
```

FILE : CAUTION.CPP

```
#ifndef __CAUTION_HPP
 #define __CAUTION_HPP
 #include "CAUTION.HPP"
#endif

void CautionClass::Message(char *p)
{
 OK(ColorTable[9],ColorTable[19],p);
}

void CautionClass::Message(int yo,char *p)
{
 OK(yo,ColorTable[6],ColorTable[7],p);
}

// End Of FILE : CAUTION.CPP -----------------
```

FILE : DOVAL.H

```
// Definition Of Domain Value -----

#define __NO_DOMAIN 0
#define __CA_DOMAIN 1
#define __FD_DOMAIN 2

// End Of FILE : DOVAL.H ----------
```

FILE : TDMEXT.CPP

```
#ifndef __TDM_HPP
 #define __TDM_HPP
 #include "tdm.hpp"
#endif

MenuType TDMCLASS::Main[NO_OF_MAIN]
                          = {
                             {"File    ","F"},
                             {"Edit    ","E"},
                             {"Domain  ","D"},
                             {"Matrix  ","M"},
                             {"Solve   ","S"},
                             {"Tips    ","T"},
                             {"Help    ","H"},
                             {"Options","O"}
                            };
MenuType TDMCLASS::Sub[NO_OF_SUB]
                 =
                  {
                   {"New          ","N"},
                   {"Open       F3","O"},
                   {"Save       F2","S"},
                   {"Save as...  ","a"},
                   {"Quit   Alt-X","Q"},

                   {"Edit             F4","E"},
                   {"Add              F7","A"},
                   {"Delete          DEL","D"},
                   {"Delete  All Belows","B"},

                   {"Customer   Ctrl-C","C"},
                   {"FR-DP      Ctrl-D","D"},

                   {"FR-DP      Ctrl-M","F"},

                   {"Data Base","D"},

                   {""," "},

                   {""," "},
                   {"Skill ","S"},
                   {"Domain","D"}
                  };

UnShort  TDMCLASS::Key[NO_OF_SUB+1]
               = {
                   NulKey,F3,F2,NulKey,AltX,
                   F4,F7,Del,NulKey,
                   CtrlC,CtrlD,
                   CtrlM,
                   NulKey,
                   F9,
                   F1,
                   NulKey,NulKey,
                   NoKey
                 };
BarType TDMCLASS::Bottom[NO_OF_BOTTOM]
                 = {
                    {"F1 Help   ","F1"},
                    {"F2 Save   ","F2"},
                    {"F3 Open   ","F3"},
                    {"F9 Tips   ","F9"},
                    {"F10 Menu  ","F10"}
                   };

int   TDMCLASS::Count [NO_OF_MAIN+1] = { NO_OF_MAIN,5,4,2,1,1,1,1,2 };
Point TDMCLASS::MPoint[NO_OF_MAIN]
               = {
                  { 4,2 },{13,2},{22,2},{31,2},{40,2},{48,2},{57,2},{67,2}
                 };

int   TDMCLASS::InitMenu =    0;
Bool  TDMCLASS::YesInvoke=FALSE;
Bool  TDMCLASS::YesCheck =TRUE ;
int   TDMCLASS::__choice =   -1;

// End Of FILE : TDMEXT.CPP --------------------------------------------
```

FILE : TDM.CPP

```
// THE FUNTION WHICH IS ONE OF FRS
// ARE INDICATED BY SYMBOL - /**/

#ifndef __TDM_HPP
 #define __TDM_HPP
 #include "tdm.hpp"
#endif

void TDMCLASS::GenerateShell(void)
{

SetScreen();
if (!Logo()) { EndEarly();return; }

if (!Initialize()/**/) { MemoryError();EndEarly();return; }


EventType e;
for(;;)
{
event.GetEvent(e);
if(YesCheck)

          YesInvoke = YesInvokeFullDown
                        (e,Main,MPoint[0],Count[0],Key,F10,InitMenu);

else     YesCheck  = TRUE;

if(YesInvoke)
         {
          __choice = FullDown(Main,Sub,Key,MPoint,InitMenu,Count,
                              ColorTable[19],ColorTable[9]);

        switch(__choice)
         {
          case __MENU_ESCAPE      :                  DealEditMenu();break;
          case __MENU_NEW         :       New()/**/;DealEditMenu();break;
          case __MENU_OPEN        :      Open()/**/;DealEditMenu();break;
          case __MENU_SAVE        :      Save()/**/;DealEditMenu();break;
          case __MENU_SAVEAS      :    Saveas()/**/;DealEditMenu();break;
          case __MENU_END         :       END()/**/;return;

          case __MENU_EDIT        :      Edit()/**/;DealEditMenu();break;
          case __MENU_ADD         :       Add()/**/;DealEditMenu();break;
          case __MENU_DELETE      :    Delete()/**/;DealEditMenu();break;
          case __MENU_DELETEALL   : DeleteAll()/**/;DealEditMenu();break;

          case __MENU_CA_DOMAIN   :  CaDomain()/**/;break;
          case __MENU_FR_DP_DOMAIN:FrDpDomain()/**/;break;

          case __MENU_MATRIX      : FrDpMatrix()/**/;DealEditMenu();break;

          case __MENU_DATABASE    :   DataBase()/**/;DealEditMenu();break;

          case __MENU_HINT        :       Hint()/**/;DealEditMenu();break;

          case __MENU_HELP        :       Help()/**/;DealEditMenu();break;

           /* -------------------------------------------------------------
          case __MENU_OPTION1     :    Option1()/**/;DealEditMenu();break;
          case __MENU_OPTION2     :    Option2()/**/;DealEditMenu();break;
           */ // IN THIS VERSION OPTION FUNCTION WAS OMITTED. ------------

           default: break;
         } // End of switch()
        }

} // End of for(;;)

} // End of FUNCTION GenerateShell()
// End of Main FILE : TDM.CPP -------------------------------------
```

FILE : TDM_LOGO.CPP

```
#ifndef __TDM_HPP
 #define __TDM_HPP
 #include "TDM.HPP"
#endif

Bool TDMCLASS::Logo(void)
{
BoxType BOX = { 17,5,54,16 };
Point     P = { 2, 20 };

event.mouse.Hide();

char *s;

 s=(char far *)malloc((BOX.x2-BOX.x1+2)*(BOX.y2-BOX.y1+2)*2);
  gettext(BOX.x1+1,BOX.y1+1,BOX.x2+2,BOX.y2+2,s);
   DrawBox(KSC_DOUBLE,SHADOW,BOX,ColorTable[3]);
   PutStrInBox("  TDM  ",                BOX,0,ColorTable[2]);
   PutStrInBox("THINKING DESIGN MACHINE",BOX,2,ColorTable[1]);
   PutStrInBox("Version 1.0",            BOX,3,ColorTable[4]);
   PutStrInBox("Application of",         BOX,5,ColorTable[1]);
   PutStrInBox("AXIOMATIC DESIGN THEORY",BOX,6,ColorTable[4]);
   PutStrInBox("MAR 10/1995"            ,BOX,8,ColorTable[2]);
   PutStrInBox("Dept. of Mechanical Engineering, MIT",
                                          BOX,10,ColorTable[4]);

   event.mouse.Show();
   Bool B = GoStop(P,ColorTable[6],ColorTable[7]);
   event.mouse.Hide();
  puttext(BOX.x1+1,BOX.y1+1,BOX.x2+2,BOX.y2+2,s);
 free(s);
   event.mouse.Show();
return B;
}


Bool TDMCLASS::GoStop(Point P,int color,int rev )
```

```
{
MenuType GOSTOP[2] =
{
{
"PROCEED ---------------------------------------------------------",
"P"
},
{
"RETURN TO DOS(Esc)----------------------------------------------",
"D"
}
};

UnShort Key[2] = { AltP, AltD };

Bool B;

int choice = PopUp(GOSTOP,Key,P,2,color,rev);
switch(choice)
      {
       case -1 : B=FALSE;break;
       case  0 : B=TRUE ;break;
       case  1 : B=FALSE;break;
      }
return B;

}

// End Of FILE : TDM_LOGO.CPP ------------------------
```

FILE : TDM_SCR.CPP

```
#define __TDM_HPP
 #define __TDM_HPP
 #include "TDM.HPP"
#endif

void TDMCLASS::SetScreen(void)
{
Point P0400           =       { 4, 0 } ;
Point P0224           =       { 2, 24} ;
BoxType UpLine        = { 0, 0,79, 0 } ;
BoxType DnLine        = { 0,24,79,24 } ;
BoxType MiddleFullBox = { 0, 1,79,23 } ;

SizeCursor(9,0);
event.mouse.Hide();
ClrScreen( ColorTable[5] );
FillBox  ( 0xb2 ,MiddleFullBox );
ClrScreen( UpLine,ColorTable[19]);
ClrScreen( DnLine,ColorTable[19]);
BarString( (BarType *)Main,P0400,
                 Count[0],ColorTable[19],ColorTable[9]);
BarString(          Bottom,P0224,
             NO_OF_BOTTOM,ColorTable[19],ColorTable[9]);
event.mouse.Show();
}

// End Of FILE : TDM_SCR.CPP --------------------------
```

FILE : TDM_EN_E.CPP

```
#ifndef __TDM_HPP
 #include "TDM.HPP"
 #define __TDM_HPP
#endif

void TDMCLASS::EndEarly(void)
{
event.mouse.Hide();
ClrScreen();
SizeCursor(8,9);
}
```

```
// End Of FILE : TDM_EN_E.CPP -----------
```

FILE : TDM_ME_E.CPP

```
#ifndef __TDM_HPP
 #define __TDM_HPP
 #include “TDM.HPP”
#endif

void TDMCLASS::MemoryError()
{
 Message( "MEMORY ERROR : NOT SUFFICIENT MEMORY.");
}

// End Of FILE : TDM_ME_E.CPP-------------------------
```

FILE : TDM_DE_E.CPP

```
#ifndef __TDM_HPP
 #define __TDM_HPP
 #include “TDM.HPP”
#endif

void TDMCLASS::DealEditMenu(void)
{
if      (Domain==__CA_DOMAIN) CaDomain()/**/;
else if (Domain==__FD_DOMAIN) FrDpDomain()/**/;
}

// End Of FILE : TDM_DE_E.CPP ---------------------
```

APPENDIX A2 :

Design Matrix & Design Parameter [2^{nd} Hierarchy]

FILE : TDM.HPP

```
// WHAT IS CHANGED WILL BE INDICATED BY ITALIC & BOLD CHARACTER.


// #ifndef __CAUTION_HPP
//  #define __CAUTION_HPP
//  #include "CAUTION.HPP"
// #endif
          // GENERAL SCREEN,MENU,EDITOR + CAUTION :
          // OMITTED HERE BECAUSE TRANSFERED TO LOWER CLASS.

#ifndef __HELP_HPP
 #define __HELP_HPP
 #include “HELP.HPP”
#endif

#ifndef __DATABASE_HPP
 #define _DTABASE_HPP
 #include “DATABASE.HPP”
#endif

#ifndef __DOFILE_HPP
 #define __DOFILE_HPP
 #include “DOFILE.HPP”
#endif

#ifndef __DO_VAL_H
 #define __DO_VAL_H
 #include “DoVal.h”
#endif
         // FILE WHICH HAS DOMAIN VALUE DEFINITION

#ifndef __MENUNO_H
 #define __MENUNO_H
 #include “MenuNo.h”
#endif
         // FILE WHICH HAS MENU NUMBER DEFINITION

#ifndef __NOMENU_H
 #define __NOMENU_H
 #include “NoMenu.h”
#endif
         // FILE WHICH HAS NO OF MENU DEFINITION


class TDMCLASS ://virtual private CautionClass,// DECIDED AT DECOMPOSITION PROCESS
                                               // TRANSFERED TO LOWER LEVEL OBJECT

                                               // HAVE DEFINITION
                                               // FOR A FUNCTION IN A FILE.
                 private HelpClass,            //     HELP.HPP : Help()
                 private DataBaseClass,        // DATABASE.HPP : DataBase()
                 public  DoFileClass           //   DOFILE.HPP :
                                               // Save(),Open(),New(),Saveas()
                                               // FrDpMatrix(),FRDpDomain(),CaDomain()
{
 private : // Static Data Const
          static MenuType     Main[NO_OF_MAIN  ];
          static MenuType      Sub[NO_OF_SUB   ];
          static UnShort       Key[NO_OF_SUB+1 ];
          static BarType    Bottom[NO_OF_BOTTOM];
          static int        Count [NO_OF_MAIN+1];
          static Point      MPoint[NO_OF_MAIN  ];
          // Static Data Variable
          static int                    InitMenu;
          static Bool                  YesInvoke;
          static Bool                   YesCheck;
          static int                    __choice;

 private : // RELATED TO DATA STRUCURE

          static UnShort Domain;

 private :
          void SetScreen    (void); // TDM_SCR .CPP

          Bool Logo         (void); // TDM_LOGO.CPP
          Bool GoStop
             (Point,int /* color*/,int /* rev */ );

          void EndEarly     (void); // TDM_EN_E.CPP
          void MemoryError  (void); // TDM_ME_E.CPP

          void DealEditMenu(void); // TDM_DE_E.CPP

 private : // FUNCTIONAL FEQUIREMENT

          int Initialize(void);     // TDM_INIT.CPP
          void END(void);           // TDM_END.CPP

 public   :

          void GenerateShell(void);
};

// End Of FILE : TDM.HPP----------------------------------------
```

FILE : TDM_INIT.CPP

```
#ifndef __TDM_HPP
 #include "TDM.HPP"
 #define __TDM_HPP
#endif

#include <ALLOC.H>

int TDMCLASS::Initialize()
{
  char *__RootFR=(char *)malloc(8);
  if(!__RootFR) return 0;
  strcpy(__RootFR,"ROOT FR");

  char *__RootDP=(char *)malloc(8);
  if(!__RootDP) return 0;
  strcpy(__RootDP,"ROOT DP");

  char *__RootExplFR=(char *)malloc(8);
  if(!__RootExplFR) return 0;
  strcpy(__RootExplFR,"ROOT FR");

  char *__RootExplDP=(char *)malloc(8);
  if(!__RootExplDP) return 0;
  strcpy(__RootExplDP,"ROOT DP");

   Unit *D1 = new Unit; if(!D1) return 0;
   Unit *D2 = new Unit; if(!D2) return 0;

   D1->AllAssZero();
   D2->AllAssZero();

   D1->AssignName(__RootFR);
   D2->AssignName(__RootDP);

   D1->AssignExpl(__RootExplFR);
   D2->AssignExpl(__RootExplDP);
   AssignRootFR(D1);
   AssignRootDP(D2);

   AssignAboveFR(GetRootFR());
   AssignAboveDP(GetRootDP());

   AssignBelowFR
    ((GetAboveFR())->GetFirstBelow( (GetAboveFR())->GetCurrentDesignNo() ));
   AssignBelowDP
    ((GetAboveDP())->GetFirstBelow( (GetAboveDP())->GetCurrentDesignNo() ));

   AssignTopLeftFR(GetRootFR());
   AssignTopLeftDP(GetRootDP());

   AssignTopRightFR(GetBelowFR());
   AssignTopRightDP(GetBelowDP());

   AssignAssFile(0);
// AssignOption (0x00);

   AssignAssCa(0);

   Domain=__NO_DOMAIN;
   AssignCurrentSection(0x00);

   for (int i=0;i<__NO_OF_DOMAIN;i++)
                        {
                         AssignHp(i,0);
                         AssignSp(i,0);
                        }
   SetHint(__NODOMAIN_BASE,__NODOMAIN_NO);

   return 1;
}

// End Of FILE : TDM_INIT.CPP  --------------------------------------------------------
```

FILE : TDM_END.CPP

```
#ifndef __TDM_HPP
 #include "TDM.HPP"
 #define __TDM_HPP
#endif

#include <ALLOC.H>

void TDMCLASS::END(void)
{

// Free Dynamically  Allocated Memory

if (GetAssCa()) free(GetCp());

(GetRootFR())->DeleteSelf();
(GetRootDP())->DeleteSelf();

// Return Mouse and Cursor to Dos Enviroment

event.mouse.Hide();
ClrScreen();
SizeCursor(8,9);

}

// End Of FILE : TDM_END.CPP ---------------
```

FILE : HELP.HPP

```
#ifndef __CAUTION_HPP
 #define __CAUTION_HPP
 #include "CAUTION.HPP"
#endif

class HelpClass : virtual CautionClass
```

```
{
 private :
         static char *HeaderFileName;
         int                NoOfMenu;
         MenuType                 *M;
         char              *FileName;

 protected : // WILL BE INSERTED AFTER DECOMPOSITION.

 public  :
         void Help(void);
};

// End Of FILE : HELP.HPP ---------------------------
```

FILE : DATABASE.HPP

```
#ifndef __CAUTION_HPP
 #define __CAUTION_HPP
 #include "CAUTION.HPP"
#endif

class DataBaseClass : virtual public CautionClass
{
 private :
         static char *HeaderFileName;
         int                NoOfMenu;
         MenuType                 *M;
         char              *FileName;

 private   :
 protected : // WILL BE INSERTED AFTER DECOMPOSITION.

 public  :
         void DataBase(void);
};

// End Of FILE : DATABASE.HPP ---------------------------
```

FILE : DOFILE.HPP

```
#ifndef __DOMAT_HPP
 #define __DOMAT_HPP
 #include “DOMAT.HPP”
#endif

#define __LEN_FILENAME 13

class DoFileClass : public DoMatrixClass
{
 private :
         char    *FileName[__LEN_FILENAME];
         File    *FileP;
         int      AssFile;

 public :
         void   Save(void);
         void   Open(void);
         void    New(void);
         void Saveas(void);
};

// End Of FILE : DOFILE.HPP --------------------
```

FILE : DOMAT.HPP

```
#ifndef __FRDP_HPP
 #define __FRDP_HPP
 #include “FRDP.HPP”
#endif

class DoMatrixClass : public FrDpClass
{
 private :
           Point Mp;
           Point TopLeftMp;
 public :
           void FrDpMatrix(void);
};

// End Of FILE : DOMAT.HPP ------------
```

FILE : FRDP.HPP

```
#ifndef __CA_HPP
 #define __CA_HPP
 #include "CA.HPP"
#endif

#ifndef __UNIT_HPP
 #define __UNIT_HPP
 #include "UNIT.HPP"
#endif

#define __NO_OF_DOMAIN 3


class FrDpClass : public CaClass
{
 private :
          Unit      *RootFR,      *RootDP;
          Unit   *TopLeftFR,   *TopLeftDP;
          Unit  *TopRightFR,  *TopRightDP;
          Unit     *AboveFR,     *AboveDP;
          Unit     *BelowFR,     *BelowDP;
          UnShort CurrentSection;
          int  Hp[__NO_OF_DOMAIN];
          int  Sp[__NO_OF_DOMAIN];

 protected :
          void AssignRootFR(Unit*);
          void AssignRootDP(Unit*);
          void AssignTopLeftFR(Unit*);
          void AssignTopLeftDP(Unit*);
          void AssignTopRightFR(Unit*);
          void AssignTopRightDP(Unit*);
          void AssignAboveFR(Unit*);
          void AssignAboveDP(Unit*);
          void AssignBelowFR(Unit*);
          void AssignBelowDP(Unit*);
          void AssignCurrentSection(UnShort);
          void AssignHp(int/*Domain NO */,int/*Value*/);
          void AssignSp(int/*Domain NO */,int/*Value*/);

          Unit      *GetRootFR();
          Unit      *GetRootDP();
          Unit      *GetTopLeftFR();
          Unit      *GetTopLeftDP();
          Unit      *GetTopRightFR();
          Unit      *GetTopRightDP();
          Unit      *GetAboveFR();
          Unit      *GetAboveDP();
          Unit      *GetBelowFR();
          Unit      *GetBelowDP();
          UnShort    GetCurrentSection();
          int        GetHp(int/*Domain NO */);
          int        GetSp(int/*Domain NO */);

public :
          // Functions called by :
          // FrDpDomain(),Edit(),Delete(),DeleteAll(),Add();


};

// End Of FILE : FRDP.HPP ----------------------------------
```

FILE : CA.HPP

```
#ifndef __HINT_HPP
 #define __HINT_HPP
 #include “HINT.HPP”
#endif

class CaClass : public HintClass
{
 private :
           int AssCa;
           char *Cp;

 protected :
           void  AssignAssCa(int);
           void  AssignCp(char *);
           int   GetAssCa();
           char *GetCp();

 public :
           // Functions called by CaDomain()

};

// End Of FILE : CA.HPP -------------------
```

FILE : HINT.HPP

```
#ifndef __CAUTION_HPP
 #define __CAUTION_HPP
 #include “CAUTION.HPP”
#endif

#define __NODOMAIN_BASE 0
#define __NO_DOMAIN_NO  0


class HintClass : public CautionClass
{
 private :
           static BoxType HintBox;
           static UnShort HintKey;

           int Base;
           int n;

 public  :
           SetHint(int /*Base*/,int /*No of Hint*/);
           Hint();
};
```

APPENDIX A3 :

Decomposition [3rd Hierarchy] : [FR4] & [FR5]

FILE : TDM.HPP

```
// WHAT IS CHANGED WILL BE INDICATED BY BOLD CHARACTER.


// #ifndef __CAUTION_HPP
//  #define __CAUTION_HPP
//  #include "CAUTION.HPP"
// #endif
          // GENERAL SCREEN,MENU,EDITOR + CAUTION :
          // OMITTED HERE BECAUSE TRANSFERED TO LOWER CLASS.

#ifndef __HELP_HPP
 #define __HELP_HPP
 #include “HELP.HPP”
#endif

#ifndef __DATABASE_HPP
 #define __DTABASE_HPP
 #include “DATABASE.HPP”
#endif

#ifndef __DOFILE_HPP
 #define __DOFILE_HPP
 #include “DOFILE.HPP”
#endif

#ifndef __DO_VAL_H
 #define __DO_VAL_H
 #include “DoVal.h”
#endif
         // FILE WHICH HAS DOMAIN VALUE DEFINITION

#ifndef __MENUNO_H
 #define __MENUNO_H
 #include “MenuNo.h”
#endif
         // FILE WHICH HAS MENU NUMBER DEFINITION

#ifndef __NOMENU_H
 #define __NOMENU_H
 #include “NoMenu.h”
#endif
         // FILE WHICH HAS NO OF MENU DEFINITION


class TDMCLASS ://virtual private CautionClass,// DECIDED AT DECOMPOSITION PROCESS
                                               // TRANSFERED TO LOWER LEVEL OBJECT

                                               // HAVE DEFINITION
                                               // FOR A FUNCTION IN A FILE.
                 private HelpClass,            //     HELP.HPP : Help()
                 private DataBaseClass,        // DATABASE.HPP : DataBase()
                 public  DoFileClass           //   DOFILE.HPP :
                                               // Save(),Open(),New(),Saveas()
                                               // FrDpMatrix(),FRDpDomain(),CaDomain()
{
 private : // Static Data Const
          static MenuType     Main[NO_OF_MAIN  ];
          static MenuType      Sub[NO_OF_SUB   ];
          static UnShort       Key[NO_OF_SUB+1 ];
          static BarType    Bottom[NO_OF_BOTTOM];
          static int        Count [NO_OF_MAIN+1];
          static Point      MPoint[NO_OF_MAIN  ];

           // Static Data Variable
          static int                    InitMenu;
          static Bool                  YesInvoke;
          static Bool                   YesCheck;
          static int                    __choice;

 private : // RELATED TO DATA STRUCURE

          static UnShort Domain;

 private :
          void SetScreen   (void); // TDM_SCR .CPP

          Bool Logo        (void); // TDM_LOGO.CPP
          Bool GoStop
             (Point,int /* color*/,int /* rev */ );

          void EndEarly    (void); // TDM_EN_E.CPP
          void MemoryError (void); // TDM_ME_E.CPP

          void DealEditMenu(void); // TDM_DE_E.CPP

 private : // FUNCTIONAL FEQUIREMENT

          int Initialize(void);    // TDM_INIT.CPP
          void END(void);          // TDM_END.CPP
          void CaDomain(void);     // TDM_CA.CPP
          void FrDpDomain(void);   // TDM_FRDP.CPP
          void Edit(void);
          void Delete(void);
          void DeleteAll(void);
          void Add(void);
 public  :

           void GenerateShell(void);
};

// End Of FILE : TDM.HPP----------------------------------------
```

FILE : TDM_CA.CPP

```
#ifndef __TDM_HPP
 #define __TDM_HPP
 #include "TDM.HPP"
#endif

void TDMCLASS::CaDomain(void)
{

int CaseNumber = EditCaDomain
                 (Domain,Main,MPoint[0],Count[0],Key,F10,InitMenu);

if(CaseNumber==-2)
  {
    YesCheck=FALSE;
    YesInvoke=TRUE;
  }
else YesCheck=TRUE;

}

// End of FILE : TDM_CA.CPP ------------------------------------
```

FILE : CA.HPP

```
#ifndef __HINT_HPP
 #define __HINT_HPP
 #include "HINT.HPP"
#endif

#ifndef __DOVAL_H
 #define __DOVAL_H
 #include "DOVAL.H"
#endif
```

```
#define __CA_BOX            { 1, 5,78,19 }
#define __CA_EDIT_BOX       { 3, 6,55,17 }
#define __CA_BELOW_BOX      { 1,20,78,22 }
#define __CA_HEAD_BOX       { 1, 2,78, 4 }
#define __CA_DONE_BOX       {57, 6,76, 8 }
#define __CA_CANCEL_BOX     {57,11,76,13 }
#define __MID_FULL_BOX      { 0, 1,79,23 }

#define __CA_EDIT_PAGE 5

class CaClass : public HintClass
{
 private :
          int AssCa;
          char *Cp;

 protected :
          void  AssignAssCa(int); // CA.CPP
          void  AssignCp(char *); // CA.CPP
          int   GetAssCa();       // CA.CPP
          char *GetCp();          // CA.CPP

 public :
          int EditCaDomain(
                           int&,              /*Domain*/
                           MenuType*,         /*M*/
                           Point,             /*P*/
                           int,               /*NoMain*/
                           UnShort*,          /*HotKeys*/
                           UnShort,           /*MenuInvokeKey*/
                           int&               /*MenuInit*/
                         ); // CA.CPP

};

// End Of FILE : CA.HPP -------------------
```

FILE : CA.CPP

```
#ifndef __CA_HPP
 #define __CA_HPP
 #include "CA.HPP"
#endif

//============================================================================
// FUNCTION NAME :  EDIT CA DOMAIN ( METHOD OF CLASS CaClass )
//----------------------------------------------------------------------------
int CaClass::EditCaDomain
//----------------------------------------------------------------------------
// INPUT  :
//----------------------------------------------------------------------------
 (
  int           &Domain,    // DOMAIN
                            // MENU SYSTEM INFORMATION :
  MenuType           *M,    // SERIES OF MAIN MENU
  Point               P,    // MENU POSITION
  int            NoMain,    // NO. OF MAIN MENU
  UnShort      *HotKeys,    // HOT KEY SERIES
  UnShort MenuInvokeKey,    // MENU INVOKE KEY
  int         &MenuInit     // MENU INITIAL CODE
                              // HAS MEANING WHEN RETURN VALUE IS ASSIGNED
 )
//----------------------------------------------------------------------------
// OUTPUT :
//----------------------------------------------------------------------------
//       RETURN VALUE  : -2 : MENU WAS INVOKED
//                       -1 : ESC WAS SELECTED
//                      >=0 : DEFINED MENU BOX SELECTED
//                       -3 : MEMORY ERROR OCCURRED
//       MENUINIT      : HAS MEANING WHEN R.T.V.= -2
//                       MENU SYSTEM INITIAL CODE
//----------------------------------------------------------------------------
// EXPLANATION :  DEAL CUSTOMER DOMAIN EDITING ENVIROMENT.
//----------------------------------------------------------------------------
```

```
//--------------------------------------------------------------------------
// DATE :
//--------------------------------------------------------------------------
// RELATED PROJECT : TDM / AXIOMATIC DESIGN GROUP / DEPT. MECH / MIT
//==========================================================================
{
  BoxType CaBox     = __CA_BOX;
  BoxType CaEditBox = __CA_EDIT_BOX;
  BoxType CaHeadBox = __CA_HEAD_BOX;
  BoxType CaBelowBox= __CA_BELOW_BOX;
  BoxType MidFullBox= __MID_FULL_BOX;
  BoxType CaDoneBox = __CA_DONE_BOX;
  BoxType CaCancelBox=__CA_CANCEL_BOX;
  BoxType MenuBox[2] = {__CA_DONE_BOX,__CA_CANCEL_BOX};

  event.mouse.Hide();
  if (Domain) CLEAR_MID_BOX(MidFullBox);
  Domain=__CA_DOMAIN;

  DrawBox(KSC_DOUBLE,!SHADOW,CaHeadBox,ColorTable[7]);
  DrawBox(KSC_DOUBLE,!SHADOW,CaBox,ColorTable[6]);
  DrawBox(KSC_DOUBLE,!SHADOW,CaBelowBox,ColorTable[7]);
  DrawBox(KSC_SINGLE,!SHADOW,CaEditBox,ColorTable[2]);
  DrawBox(KSC_SINGLE,SHADOW,CaDoneBox,ColorTable[2]);
  DrawBox(KSC_SINGLE,SHADOW,CaCancelBox,ColorTable[2]);

  PutStrInBox("CUSTOMER DOMAIN",CaHeadBox,1,ColorTable[8]);
  PutStrInBox("Describe Design Need.",CaBelowBox,1,ColorTable[5]);
  PutStrInBox("CANCEL",CaCancelBox,1,ColorTable[3]);
  PutStrInBox("DONE",CaDoneBox,1,ColorTable[3]);

  event.mouse.Show();

  char  *p;
  int MenuReturn;
  if(GetAssCa())  // IF IT HAS FREE ASSIGNED TEXT
    {
      SizeCursor(8,9);
      long ep =  Editer(
                         CaEditBox,&p,
                         ColorTable[2],ColorTable[3],__CA_EDIT_PAGE,
                         GetCp(),1,
                         MenuBox,2,MenuReturn,
                         M,P,NoMain,HotKeys,MenuInvokeKey,MenuInit
                       );
       if (ep<0)  {
                   Message("Out of Memory");
                   return -3;
                  }
       SizeCursor(9,0);

       if(MenuReturn==0 || MenuReturn==-2) // IF "DONE" SELECTED
         {
          free(GetCp());
          if(ep)                  {
                                   AssignCp(p);
                                   AsssignAssCa(1);
                                  }
          else                    {
                                   free(p);
                                   AssignAssCa(0);
                                  }
          if(MenuReturn==0)                 // IF "DONE"  BOX SELECTED
                                  {
                                   Domain=__NO_DOMAIN;
                                   event.mouse.Hide();
                                   CLEAR_MID_BOX(MidFullBox);
                                   event.mouse.Show();
                                  }
         }
       else if(MenuReturn==1 || MenuReturn==-1) // IF "CANCEL" SELECTED
         {
          free(p);
          Domain=__NO_DOMAIN;
          event.mouse.Hide();
          CLEAR_MID_BOX(MidFullBox);
          event.mouse.Show();
```

```
        }
       }
    else  // IF IT  DOESN'T HAVE PRE-ASSIGNED TEXT
       {
        SizeCursor(8,9);
         char *p;
        long ep = Editer(
                        CaEditBox,&p,
                        ColorTable[2],ColorTable[3],__CA_EDIT_PAGE,
                        MenuBox,2,MenuReturn,
                        M,P,NoMain,HotKeys,MenuInvokeKey,MenuInit
                        );
         AssignCp(p);
        if (ep<0)   {
                    Message("Out of Memory");
                    return -3;
                   }

        SizeCursor(9,0);
        if          (MenuReturn==0 || MenuReturn==-2) // IF "DONE" SELECTED
                   {
                    if (ep)                   AssignAssCa(1);
                    else                      free(GetCp());
                    if (MenuReturn==0)  // IF DONE BOX SELECTED
                                       {
                                        Domain=__NO_DOMAIN;
                                        event.mouse.Hide();
                                        CLEAR_MID_BOX(MidFullBox);
                                        event.mouse.Show();
                                       }
                   }
        else if     (MenuReturn==1 || MenuReturn==-1) // IF "CANCEL" SELECTED
                   {
                    free(GetCp());
                    Domain=__NO_DOMAIN;
                    event.mouse.Hide();
                    CLEAR_MID_BOX(MidFullBox);
                    event.mouse.Show();
                   }
       }

return MenuReturn;
}

void  CaClass::AssignCp(char *p)      { Cp=p;          }
void  CaClass::AssignAssCa(int yesno) { AssCa=yesno; }
char *CaClass::GetCp()                { return Cp;     }
int   CaClass::GetAssCa()             { return AssCa;}

// End Of FILE : CA.CPP ----------------------------------------------
```

FILE : TDM_FRDP.CPP

```
#ifndef __TDM_HPP
 #define __TDM_HPP
 #include “TDM.HPP”
#endif

void TDMCLASS::FrDpDomain(void)
{

if ( GetAssCa())
   {
    Message("YOU SHOULD FINISH CUSTOMER DOMAIN BEFORE FR-DP DOMAIN.");
    YesCheck=FALSE;
    YesInvoke=TRUE;
    InitMenu = -1 - __MENU_CA_DOMAIN;
    return;
   }

int CaseNumber = EditFrDpDomain
                     (Domain,Main,MPoint[0],Count[0],Key,F10,InitMenu);
if(CaseNumber==-1)
  {
    YesCheck=FALSE;
    YesInvoke=TRUE;
  }
else if(CaseNumber==1)
  {
    YesCheck=FALSE;
    YesInvoke=TRUE;
    InitMenu = -1 - __MENU_ADD;
  }
else YesCheck=TRUE;

}

void TDMCLASS::Edit(void)
{
if (Domain!=__FD_DOMAIN)
   {
    Message("YOU SHOULD SELECT FR-DP DOMAIN TO USE EDIT MENU.");
    return;
   }

DoEdit();
}

void TDMCLASS::Add(void)
{
if (Domain!=__FD_DOMAIN)
   {
     Message("YOU SHOULD SELECT FR-DP DOMAIN FIRST TO ADD ITEM.");
     return;
   }

DoAdd();
}

void TDMCLASS::Delete(void)
{
if (Domain!=__FD_DOMAIN)
   {
     Message("YOU SHOULD SELECT FR-DP DOMAIN FIRST TO DELETE ITEM.");
     return;
   }

DoDelete();
}

void TDMCLASS::DeleteAll(void)
{
if (Domain!=__FD_DOMAIN)
   {
     Message("YOU SHOULD SELECT FR-DP DOMAIN FIRST TO DELETE BELOW ITEMS.");
     return;
   }

DoDeleteAll();
}

// End Of FILE : TDM_FRDP.CPP
```

FILE : FRDP.HPP

```
#ifndef __CA_HPP
 #define __CA_HPP
 #include “CA.HPP”
#endif

#ifndef __UNIT_HPP
 #define __UNIT_HPP
 #include “UNIT.HPP”
#endif

#define __NO_OF_DOMAIN 3

#define __ARROW_1_BOX { 2,4, 2,4}
#define __ARROW_2_BOX {38,4,38,4}
#define __ARROW_3_BOX {53,4,53,4}
```

```
#define __ARROW_4_BOX {77,4,77,4}
#define __ARROW_5_BOX  { 5,21, 5,21}
#define __ARROW_6_BOX  {71,21,71,21}
#define __ARROW_7_BOX  { 5,22, 5,22}
#define __ARROW_8_BOX  {71,22,71,22}
#define __ARROW_9_BOX  {74, 3,74, 3}
#define __ARROW_10_BOX {77, 3,77, 3}
#define __SELECT_DESIGN_BOX      { 60,19, 77,19}
#define __FD_DONE_BOX { 73,20,78,22 }

#define __MOUSE_DELAY  1000

#define __FD_TITLE_1    { 21, 6,38, 6 }
#define __FD_TITLE_2    {  2, 6,19, 6 }
#define __FD_TITLE_3    { 41, 6,58, 6 }
#define __FD_TITLE_4    { 60, 6,77, 6 }



class FrDpCalss : public CaClass
{
 private :
          Unit      *RootFR,     *RootDP;
          Unit   *TopLeftFR,   *TopLeftDP;
          Unit  *TopRightFR,  *TopRightDP;
          Unit     *AboveFR,     *AboveDP;
          Unit     *BelowFR,     *BelowDP;
          UnShort CurrentSection;
          int  Hp[__NO_OF_DOMAIN];
          int  Sp[__NO_OF_DOMAIN];

 protected :
          void AssignRootFR(Unit* U)            { RootFR = U; }
          void AssignRootDP(Unit* U)            { RootDP = U; }
          void AssignTopLeftFR(Unit* U)      { TopLeftFR = U; }
          void AssignTopLeftDP(Unit* U);     { TopLeftDP = U; }
          void AssignTopRightFR(Unit* U);   { TopRightFR = U; }
          void AssignTopRightDP(Unit* U);   { TopRightDP = U; }
          void AssignAboveFR(Unit*U);          { AboveFR = U; }
          void AssignAboveDP(Unit*U);          { AboveDP = U; }
          void AssignBelowFR(Unit*U);          { BelowFR = U; }
          void AssignBelowDP(Unit*U);          { BelowDP = U; }
          void AssignCurrentSection(UnShort u);{ CurrentSection = u; }
          void AssignHp(int domain ,int value);{ Hp[domain] = value; }
          void AssignSp(int domain ,int value);{ Sp[domain] = value; }

          Unit      *GetRootFR();               { return RootFR; }
          Unit      *GetRootDP();               { return RootDP; }
          Unit      *GetTopLeftFR();            { return TopLeftFR; }
          Unit      *GetTopLeftDP();            { return TopLeftDP; }
          Unit      *GetTopRightFR();           { return TopRightFR; }
          Unit      *GetTopRightDP();           { return TopRightDP; }
          Unit      *GetAboveFR();              { return AboveFR; }
          Unit      *GetAboveDP();              { return AboveDP; }
          Unit      *GetBelowFR();              { return BelowFR; }
          Unit      *GetBelowDP();              { return BelowDP; }
          UnShort    GetCurrentSection();       { return CurrentSection; }
          int        GetHp(int domain);         { return Hp[domain]; }
          int        GetSp(int domain);         { return Sp[domain]; }

public :
          int        EditFrDpDomain(

                           int&,               /*Domain*/
                           MenuType*,          /*M*/
                           Point,              /*P*/
                           int,                /*NoMain*/
                           UnShort*,           /*HotKeys*/
                           UnShort,            /*MenuInvokeKey*/
                           int&                /*MenuInit*/
                          );
          void       DoEdit(void);
          void       DoDelete(void);
          void       DoDeleteAll(void);
          void       DoAdd(void);
};
// End Of FILE : FRDP.HPP ----------------------------------
```

FILE : FRDP.CPP

```
#ifndef __FRDP_HPP
 #define __FRDP_HPP
 #include "FRDP.HPP"
#endif


int FrDpClass::EditFrDpDomain
(
  int           &Domain,

  MenuType           *M,
  Point               P,
  int            NoMain,
  UnShort      *HotKeys,
  UnShort MenuInvokeKey,
  int         &MenuInit
)
{

BoxType SelectDesignBox=__SELECT_DESIGN_BOX;
BoxType DoneBox=__FD_DONE_BOX;
BoxType  ArrowBoxes[10]={
                               __ARROW_1_BOX,__ARROW_2_BOX,
                               __ARROW_3_BOX,__ARROW_4_BOX,
                               __ARROW_5_BOX,__ARROW_6_BOX,
                               __ARROW_7_BOX,__ARROW_8_BOX,
                               __ARROW_9_BOX,__ARROW_10_BOX
                        };
BoxType TitleBoxes[4]={
                             __FD_TITLE_1,__FD_TITLE_2,
                             __FD_TITLE_3,__FD_TITLE_4
                      };


 event.mouse.Hide();
 if (Domain!=__FD_DOMAIN) InitialrizeFDwholeScreen()/**/;
 event.mouse.Show();

 EventType e;
 for(int Pause=0;;Pause++,Pause%=__MOUSE_DELAY)
 {
  event.GetEvent(e);

  if(YesInvokeFullDown(e,M,P,NoMain,HotKeys,MenuInvokeKey,MenuInit))
  return -1;


  if(e.what==MouseDown&&e.mouse.buttons==LeftButton) // ONE THOUCH MOUSE
    {
     if(e.mouse.DoubleClick) // IF DOUBLE CLICKED
       {
       UnShort UnitNo,TitleNo;
       if(MouseOneOfUnit(e.mouse.where,TitleNo,UnitNo)/**/)
          {
          DoMouseChange(TitleNo,UnitNo)/**/;
          event.mouse.Hide();InitFdScr()/**/;event.mouse.Show();
          DoEdit();
          event.mouse.Hide();InitFdScr()/**/;event.mouse.Show();
          }
       }  // END OF MOUSE DOUBLE CLICK
     else    // IF SIMPLELY CLICKED
       {
       // IF ONE OF UNIT
       UnShort UnitNo,TitleNo;
       if(MouseOneOfUnit(e.mouse.where,TitleNo,UnitNo))
          {
          DoMouseChange(TitleNo,UnitNo)/**/;
          event.mouse.Hide();InitFdScr()/**/;event.mouse.Show();
```

```
         }
       // IF ARROW SELECTED
       int BoxSelected=MatchBoxNo(ArrowBoxes,e.mouse.where,9);
       if(BoxSelected!=-1)
         {
          DoArrowBoxes(MatchBoxNo(ArrowBoxes,e.mouse.where,9))/**/;
          MouseDelay();
         }
       // IF DONE BOX SELECTED
       if(MouseInBox(e.mouse.where,DoneBox))
         { DoFdEnd()/**/; return 0; }
       // DESIGN SELECT BOX
       if(MouseInBox(e.mouse.where,SelectDesignBox))

           DoDesignSelect()/**/;

       // TITLE BOXES
       int TitleBoxNo=MouseOneOfBoxes(e.mouse.where,4,TitleBoxes);
       switch(TitleBoxNo)
        {
          case 0 : break;
          case 1 : DoTitleUpFR()/**/;break;
          case 2 : DoTitleDownFR()/**/;break;
          case 3 : DoTitleUpDP()/**/;break;
          case 4 : DoTitleDownDP()/**/;break;
          default: break;
        }
       } // END OF SIMPLE CLICK
    }  // END OF MOUSE ONE THOUCH LEFT

  else if(e.what==MouseDown&&e.mouse.buttons==RightButton) // ONE TOUCH RIGHT
    {
     if(e.mouse.DoubleClick); // IF DOUBLE CLICKED
     else                     // ONE TOUCH
       {
       UnShort UnitNo,TitleNo;
       if(MouseOneOfUnit(e.mouse.where,TitleNo,UnitNo)/**/)
         {
          DoMouseChange(TitleNo,UnitNo)/**/;
          event.mouse.Hide();InitFdScr()/**/;event.mouse.Show();
          return 1;
         }
       }

    }


  else if(e.what==EvKeyDown)   // ONE THOUCH KEY ACTION
    {
      if     (e.key.keyCode==Esc)  { DoFdEnd()/**/; return 0; }
      else if(e.key.keyCode==Left) { DoFdLeft()/**/;}
      else if(e.key.keyCode==Right){ DoFdRight()/**/;}
      else if(e.key.keyCode==Up)   { DoFdUp()/**/;}
      else if(e.key.keyCode==Down) { DoFdDown()/**/;}
      else if(e.key.charScan.charCode=='\r')
             {
              DoEdit();
              event.mouse.Hide();InitFdScr()/**/;event.mouse.Show();
             }

    }

  if( e.mouse.buttons==LeftButton && !Pause)
          // CONTINUOUS MOUSE PUSH
    {
       DoArrowBoxes(MatchBoxNo(ArrowBoxes,e.mouse.where,9))/**/;
    }

 } // END OF FOR

} // END OF FUNCTION : EDIT FR DP DOMAIN()


void FrDpClass::DoAdd(void)
{
```

```
Unit *New;
int error;
UnShort T,d; // T:TITLE NO // d:SECTION BUFFER
T=CurrentSection>>3;
if(T==1||T==3)
{
 Message("YOU CAN'T ADD UNIT IN THE DOWN BOX.");
 return;
}

if(T==0) // FR DOMAIN
{
int *Series        =AboveFR->FindSeries();
Unit *REAL_ABOVE_DP=RootDP->FindPointer(Series);
free(Series);
if (REAL_ABOVE_DP->GetSelectedDesignNo()!=-1)
 {
  Message("YOU'VE ALEADY DECIDE THIS LEVEL DESIGN. ");
  Message("TO ADD MORE FRS, DELETE RELATED DPS.");
  Message("DIFFERENT SET OF FR IS SATISFIED BY DIFFERENT SET OF DP");
  return;
 }


New=AboveFR->MakeNext(0,error);
switch(error)
  {
    case 1 : Message("Decide Above FR at first.");return;
    case 3 : Message("Out of Memory.");return;
  }
New->MakeDesignSelected();

int ESpush=Sp[0];
int HSpush=Hp[0];
Unit *Bpush=BelowFR;
UnShort Cpush=CurrentSection;
Unit *Tpush=TopRightFR;


Sp[0]=0;
Hp[0]=0;
BelowFR       =                                   New;
CurrentSection=(1<<3)|(1<<2)|(CurrentSection&0x03);
if(!(New->GetPrev())) TopRightFR=New;

for(
    ;BelowFR->FindGap()-TopRightFR->FindGap()>11
    ;TopRightFR=TopRightFR->GetNext()
   );

event.mouse.Hide();
InitFdScr()/**/;
event.mouse.Show();
if ( (DoEditFr(BelowFR,RootDP)/**/)==2 ) // IF CANCEL SELECTED
    {
     New->DeleteUnit();

     Sp[0]=ESpush;
     Hp[0]=HSpush;
     BelowFR=Bpush;
     CurrentSection=Cpush;
     TopRightFR=Tpush;

     event.mouse.Hide();
     InitFdScr()/**/;
     event.mouse.Show();
    }

else  { event.mouse.Hide();InitFdScr()/**/;event.mouse.Show(); }
}

if(T==2) // DP DOMAIN
{
// CHECKING WHETHER OR NOT FR IS ASSIGNED.

{ // THIS BRACKET WAS SELECTED TO CONSTRAINT RANGE OF INT VAR NO OF FR
int  *FR_SERIES=AboveDP->FindSeries();
Unit *ABOVE_FR =S->RootFR(FR_SERIES);
free(FR_SERIES);

int NoOfFR=ABOVE_FR->GetFirstBelow(ABOVE_FR->GetCurrentDesignNo())->FindNoOfUnit();
if ( !NoOfFR )
     {
       Message("BEFORE EDITING DP, FR SHOULD BE EDITED.");
       return;
     }

if (
     NoOfFR                         // NO OF FR
     <=                             // LESS OR EQUAL
     TopRightDP->FindNoOfUnit()     // NO OF DP
   ) {
       Message("YOU ARE NOT ALLOWED TO MAKE REDUNDANT DESIGN.");
       return;
     }
} // RANGE OF INT VAR NO OF FR

New=AboveDP->MakeNext(AboveDP->GetCurrentDesignNo(),error);
switch(error)
  {
    case 1 : Message("Decide Above FR at first.");return;
    case 3 : Message("Out of Memory.");return;
  }

int ESpush=Sp[1];
int HSpush=Hp[1];
Unit *Bpush=BelowDP;
UnShort Cpush=CurrentSection;
Unit *Tpush=TopRightDP;

Sp[1]=0;
Hp[1]=0;
BelowDP       =                                    New;
CurrentSection=(3<<3)|(1<<1)| (S->CurrentSection&0x05);

if(!(New->GetPrev())) TopRightDP=New;

for(
    ;BelowDP->FindGap()-TopRightDP->FindGap()>11
    ;TopRightDP=TopRightDP->GetNext()
   );

event.mouse.Hide();
InitFdScr()/**/;
event.mouse.Show();
if ( (DoEditDp(BelowDP,RootFR)/**/)==2 ) // IF CANCEL SELECTED
    {
     New->DeleteUnit();

     Sp[1]=ESpush;
     Hp[1]=HSpush;
     BelowDP=Bpush;
     CurrentSection=Cpush  ;
     TopRightDP=Tpush;
     event.mouse.Hide();
     InitFdScr()/**/;
     event.mouse.Show();
    }

else { event.mouse.Hide();InitFdScr()/**/;event.mouse.Show(); }

}

}

void FrDpClass::DoEdit(void)
{

switch((int)(CurrentSection>>3))
{
 case 0 : DoEditFr(AboveFR,RootDP/* ACTUALLY THIS IS DP SO LATER DECIDED */ )/**/;break;
 case 1 : DoEditFr(BelowFR,RootDP/* ACTUALLY THIS IS DP SO LATER DECIDED */ )/**/;break;
 case 2 : DoEditDp(AboveDP,RootFR/* ACTUALLY THIS IS DP SO LATER DECIDED */ )/**/;break;
 case 3 : DoEditDp(BelowDP,RootFR/* ACTUALLY THIS IS DP SO LATER DECIDED */ )/**/;break;
}

}


#define __DELETE_UNIT_SCREEN_CHANGE(B,T,I,J,K,L)\
  {\
    Unit *B_NEXT=B->GetNext();\
    Unit *B_PREV=B->GetPrev();\
    switch(B->DeleteUnit())  /* SWITCH TO DELETE RESULT */\
      {\
       case 0 : /* CHILDREN EXIST */\
              Message("YOU CANNOT DELETE THIS UNIT. IT HAS CHILDREN.");\
              return;\
       case 1 : /*  FIRST UNIT */\
              B=B_NEXT;\
              T=B;\
              break;\
       case 2 : /*   ONLY UNIT */\
              B=NULL;\
              T=NULL;\
              CurrentSection=(I<<3)|(J<<L)|(S->CurrentSection & K);\
              break;\
       case 3 : /* MIDDLE UNIT */\
              B=B_NEXT;\
              T=B->GetTop();\
              break;\
       case 4 : /* LAST UNIT   */\
              B=B_PREV;\
              T=B->GetTop();\
              break;\
      } /* END OF SWITCH(DELETE RESULT) */\
  } /* END OF __DELETE_UNIT_SCREEN_CHANGE(B,T,I,J,K,L)  DEFINITION */\


void FrDpClass::DoDelete(void)
{

 switch(CurrentSection>>3) // SWITCH ACCORDING TO [CURRENT TITLE BAR]
  {
   case 0 :
   case 2 : // IF IN THE              UP BOX // SWITCH TITLE BAR
    Message("YOU CAN'T DELETE UNIT IN THE UP BOX.");
    return;
   case 1 : // IF IN THE FR DOMAIN DOWN BOX  // SWITCH TITLE BAR
    int *Series=AboveFR->FindSeries();
    Unit *REAL_ABOVE_DP=RootDP->FindPointer(Series);
    free(Series);
    for(int ii=0;ii<MAX_DESIGN_CHOICE;ii++)
    {
    int NoOfDP
        =
       (REAL_ABOVE_DP->GetFirstBelow(ii))->FindNoOfUnit();

    if(BelowFR->FindNoOfUnit()    <= NoOfDP)
      {
       Message("YOU CANNOT REMOVE THIS FR. FIRST DELETE RELATED DP.");
       return;
      }
    }
    __DELETE_UNIT_SCREEN_CHANGE(BelowFR,TopRightFR,0x00,0x00,0x03,2);
    break;  // BREAK OF [IF TITLE BAR IN FR DOWN BOX]
   case 3 : // IF IN THE DP DOWN BOX // SWITCH TITLE BAR

    if(AboveDP->GetSelectedDesignNo()==AboveDP->GetCurrentDesignNo())
      {
       int *Series=AboveDP->FindSeries();
       Unit *REAL_ABOVE_FR=RootFR->FindPointer(Series);
       int NoOfFR=(REAL_ABOVE_FR->GetFirstBelow(0))->FindNoOfUnit();
       free(Series);
       Bool HasChild=FALSE;
       Unit *Dummy=REAL_ABOVE_FR->GetFirstBelow(0);
       for (int i=0;i<NoOfFR && Dummy;i++,Dummy=Dummy->Next)
            if ((HasChild=Dummy->YesHasChildren())==TRUE) break;
       if(HasChild)
         {
          Message("THE GROUP OF RELATED FRS HAS CHILDREN. REMOVE THE FR FIRST.");
          return;
         }
       HasChild=FALSE;
       Dummy=AboveDP->GetFirstBelow(AboveDP->GetCurrentDesignNo());
       for (    i=0;i<BelowDP->FindNoOfUnit() && Dummy;i++,Dummy=Dummy->Next)
            if ((HasChild=Dummy->YesHasChildren())==TRUE) break;

       if(HasChild)
         {
          Message("THIS GROUP OF DPS HAS CHILDREN. REMOVE THEM FIRST.");
          return;
         }
       else
         {
          AboveDP->AssignSelectedDesignNo(-1);
         }
      }
    __DELETE_UNIT_SCREEN_CHANGE(BelowDP,TopRightDP,0x02,0x00,0x05,1);
    break;  // BREAK OF [IF TITLE BAR IN DP DOWN BOX]
  } // END OF SWITCH [TITLE BAR]
 event.mouse.Hide();
 InitFdScr()/**/;
 event.mouse.Show();
} // END OF FUNCTION DELETE_FD()


void FrDpClass::DoDeleteAll(void)
{
 switch(CurrentSection>>3) // SWITCH ACCORDING TO [CURRENT TITLE BAR]
  {
   case 1 :
   case 3 : // IF IN THE DOWN BOX
    Message("YOU CANNOT DELETE IN THE DOWN BOX. SELECT UP BOX");
    return; // END OF CASE(1 AND 3)
   case 0 : // IF IN THE UP FR BOX
    Unit *REAL_DP=FindRealAboveDP()/**/;
    AboveFR->DeleteBelows();
    REAL_DP->DeleteBelows();
    if( (Unit huge*)(AboveDP)==(Unit huge*)(REAL_DP) )
     {
      BelowDP=NULL;
      TopRightDP=NULL;
     }

    BelowFR=NULL;
    TopRightFR=NULL;

    break;  // END OF CASE(0)
   case 2 : // IF IN THE UP DP BOX
    Unit *REAL_FR=FindRealAboveFR()/**/;
    REAL_FR->DeleteBelows();
    AboveDP->DeleteBelows();
    if( (Unit huge*)(AboveFR)==(Unit huge*)(REAL_FR) )
     {
      BelowFR=NULL;
      TopRightFR=NULL;
     }

    BelowDP=NULL;
    TopRightFR=NULL;

    break;  // END OF CASE(2)

  } // END OF SWITCH [CURRENT TITLE BAR]

 event.mouse.Hide();
 InitFdScr()/**/;
 event.mouse.Show();
} // END OF FUNCTION DELETE_ALL_FD()

// End of FILE : FRDP.CPP -------------------------------------------------
```

Appendix B : Axioms/Corollaries/Theorems

Axioms

Axiom 1 : *The Independence Axiom*

Maintain the independence of FRs.

Axiom 2 : *The Information Axiom*

Minimize the information content.

Corollaries

Corollary 1 : *Decoupling of Coupled Designs*

Decouple or separate parts or aspects of a solution if FRs are coupled or become interdependent in the design proposed.

Corollary 2 : *Minimization of FRs*

Minimize the number of FRs and constraints.

Corollary 3 : *Integration of Physical Parts*

Integrate design features in a single physical part if FRs can be independently satisfied in the proposed solution.

Corollary 4 : *Use of Standardization*

Use standardized or interchangeable parts if the use of this parts is consistent with FRs and constraints.

Corollary 5 : *Use of Symmetry*

Use symmetrical shapes and/or components if they are consistent with the FRs and constraints.

Corollary 6 : *Largest Tolerance*

Specify the largest allowable tolerance in stating FRs.

Corollary 7 : *Uncoupled Design with Less Information Content*

Seek an uncoupled design that requires less information than coupled designs in satisfying a set of FRs.

Corollary 8 : *Effective Reangularity of a Scalar*

The effective reangularity R for a scalar coupling “matrix” or element is unity.

Theorems

Theorem 1 : *Coupling Due to Insufficient Number of DPs*

When the number of DPs is less than the number of FRs, either a coupled design results, or the FRs cannot be satisfied.

Theorem 2 : *Decoupling of Coupled Design*

When a design is coupled due to the greater number of FRs than DPs (i.e., $m > n$), it may be decoupled by the addition of new DPs so as to make the number of FRs and DPs equal to each other, if a subset of the design matrix containing $n \times n$ elements constitutes a triangular matrix.

Theorem 3 : *Redundant Design*

When there are more DPs than FRs, the design is either a redundant design or a coupled design.

Theorem 4 : *Ideal Design*

In an ideal design, the number of DPs is equal to the number of FRs.

Theorem 5 : *Need for New Design*

When a given set of FRs is changed by the addition of a new FR, or substitution of one of the FRs with a new one, or by selection of a completely different set of FRs, the solution given by the original DPs cannot satisfy the new set of FRs. Consequently, a new design solution must be sought.

Theorem 6 : *Path Independency of Uncoupled Design*

The information content of an uncoupled design is independent of the sequence by which the DPs are changed and on the specific paths of change of these DPs.

Theorem 7 : *Path dependency of Coupled and Decoupled Design*

The information contents of coupled and decoupled designs depend on the sequence by which the DPs are changed and on the specific paths of change of these DPs.

Theorem 8 : *Independence and Tolerance*

A design is an uncoupled design when the designer-specified tolerance is greater than

$$\left(\sum_{j \neq i, j=1}^{j=n} \frac{\partial FR_i}{\partial DP_j} \Delta DP_j \right)$$

in which case the non diagonal elements of the design matrix can be neglected from design consideration.

Theorem 9 : *Design for Manufacturability*

For a product to be manufacturable, the design matrix for the product, **[A]** (which relates the **FR** vector for the product to the **DP** vector of the product) times the design matrix **[B]** (which relates the **DP** vector to the **PV** vector of the manufacturing process) must yield either a diagonal or triangular matrix. Consequently, when any one of these design matrices; that is, either **[A]** or **[B]**, represents a coupled design, the product cannot be manufactured.

Theorem 10 : *Modularity of Independence Measure*

Suppose that a design matrix **[DM]** cannot be partitioned into square submatrices that are nonzero only along the main diagonal. Then the reangularity and semangularity for **[DM]** are equal to the products of their corresponding measures for each of the nonzero submatrices.

Theorem 11 : *Invariance*

Reangularity and semangularity for a design matrix **[DM]** are invariant under alternative orderings of the FR and DP variables, as long as orderings preserve the association of each FR with its corresponding DP.

Theorem 12 : *Sum of Information*

The Sum of information for a set of events is also information, provided that proper conditional probabilities are used when the events are not statistically independent.

Theorem 13 : *Information Content of Total System*

If each DP is probablistically independent of other DPs, the information content of the total system is the sum of information of all individual events associated with the set of FRs that must be satisfied.

Theorem 14 : *Information Content of Coupled versus Uncoupled Design*

When the state of FRs is changed from one state to another in the functional domain, the information required for the change is greater for a coupled process than for an uncoupled process.

Theorem 15 : *Design-Manufacturing Interface*

When the manufacturing system compromises the independence of the FRs of the product, either the design of the product must be modified, or a new manufacturing process must be designed and/or used to maintain the independence of the FRs of the products.

Theorem 16 : *Equality of Information Content*

All information contents that are relevant to the design task are equally important regardless of their physical origin, and no weighting factor should be applied to them.

References

[1] Suh, Nam P., 1990, *The Principles of Design*, Oxford University Press.

[2] Suh, Nam P. and Sekimoto, Shinya, 1990, "Design of Thinking Design Machine", *Annals of the CIRP*, v.39/1/1990, pp. 145-148.

[3] Albano, Leonard D. and Suh, Nam P., 1992, "Axiomatic Approach to Structual Design", *Research in Engineering Design*, 4:171-183, 1992.

[4] Kim, Sun-Jae, Suh, Nam P., and Kim, Sang-Gook, 1991, "Design of Software System based on Axiomatic Design", *International Journal of Robotics and Computer Integrated Manufacturing*, Vol. 8, No. 4, pp.243-255, 1991.

[5] Wallace, David R. and Suh, Nam P., 1993, "Information-Based Design for Environmental Problem Solving", *Annals of the CIRP*, Vol. 42/1/1993, pp. 175-180.

[6] Gebla, David A. and Suh, Nam P., 1992, "An Application of Axiomatic Design", *Research in Engineering Design*, 3:149-146, 1992.

[7] Suh, Nam P., 1984, "Development of the Science Base for the Manufacturing Field Through the Axiomatic Approach", *Robotics and Computer Integrated Manufacturing*, vol. 1, no. 3/4, 1984, pp. 399-415.

[8] Suh, N.P., Kim, S.H., Bell, A.C., Wilson, D.R., Cook, N.H., and Lapidot, N., 1978, "Optimizatiotion of Manufacturing Systems Through Axiomatics", *Annals of CIRP*, vol. 27, 1978b.

[9] Suh, N.P., Bell, A.C., and Gossard, D., 1978, " On an Axiomatic Approach to Manufacturing Systems", *Journal of Engineering and Industry*, Transactions of the A.S.M.E., vol. 100, 1978a, pp. 127-130.

[10] Hill, Percy H., 1970, *The Science of Engineering Design*, Holt, Rinehart and Winston, Inc..

[11] Coyne, R.D., Rosenman, M.A., Radford, A.D., Balachandran, M., and Gero, J.S., 1990, *Knowledge-based design systems*, Addison-Wesley Publishing Company, Inc..

[12] Kolski, C., Strugeon, Le E., and Tendjaoui, M., 1993, "Implementation of AI Techniques for Intelligent Interface Development", *Engng Applic. Artif. Intell.*, Vol 6, No. 4, pp 295-305, 1993.

[13] Cross, Niegel, 1993,"Science and Design Methodology: A Review", *Research in Engineering Design*, 5:63-69, 1993.

[14] Mengguang, Wang, Zhining, Liu, and Zihou, Yang, 1993, "Knowlwdge Representation for Problem Processing in a Decision Support System", *Engng Applic. Artif. Intell.*, Vol. 6, No. 4, pp 381-386, 1993.

[15] Henderson, Mark R. and LeRoy, E. Tayor, 1993, "A Meta-Model for Mechanical Products Based Upon the Mechanical Design Process", *Research in Engineering Design*, 5:140-160, 1993.

[16] Bieniawski, Z. T., 1993, "Principles and Methodology of Design for Excavations in Geologic Media", *Research in Engineering Design*, 5:49-58, 1993.

[17] Ladd, Scott R., 1990, *C++ Techniques and Application*, Prentice Hall Inc., UK.

[18] Scildt, Herbert, 1992, *Turbo C/C++: The Complete Reference, Second Edition*, McGraw-Hill.