Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

A COST MODEL FOR MULTI-LIFECYCLE ENGINEERING DESIGN

by
Bin Zhang

Design for Environment aims to yield a product whose aggregate environmental impact is as small as possible. This thesis characterizes a product recovery system and defines the concepts of product lifetime and part lifetime. A multi-lifecycle product recovery model is developed based on the concepts of product tree and disassembly path representation, four product recovery choices, time varying costs, cost comparison. The four product recovery choices of a part in the product tree are: Reconditioning, Part Remanufacturing, Material Recycling, and Landfill. This thesis focuses on how to improve the product design while minimizing environmental impact of the product and introduces an indirect product design improvement method based on the concept of candidate set. A monitor and a Personal Computer are used to illustrate the model and method. A PC Windows 95 software based computer aided DFE tool is partially implemented to demonstrate our method and model. More research and development is needed to implement a complete DFE tool incorporating Multi-lifecycle Engineering concept. The obtained model and results will play an important role in research and development of Multi-lifecycle engineering product design and guide designers in their product and process design.

A COST MODEL FOR MULTI-LIFECYCLE ENGINEERING DESIGN

by
Bin Zhang

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Electrical Engineering

Department of Electrical and Computer Engineering

January 1997

APPROVAL PAGE

A COST MODEL FOR MULTI-LIFECYCLE ENGINEERING DESIGN

Bin Zhang

Dr. MengChu Zhou, Thesis Advisor Associate Professor of Electrical and Computer Engineering, NJIT	Date
Dr. Reggie Caudill, Committee Member Professor of Mechanical Engineering and Executive Director of Multi-Lifecycle Engineering Research Center, NJIT	Date
Dr. Nirwan Ansari, Committee Member Associate Professor of Electrical and Computer Engineering, NJIT	Date

BIOGRAPHICAL SKETCH

Author: Bin Zhang

Degree: Master of Science

Date: January, 1997

Undergraduate and Graduate Education:

- Master of Science in Electrical Engineering,
  New Jersey Institute of Technology, Newark, NJ, 1997

- Bachelor of Engineering in Automatic Control,
  Tsinghua University, Beijing, P. R. China, 1993

Major: Electrical Engineering

Publication:

MengChu Zhou, Bin Zhang, Reggie J. Caudill and Donald Sebastian,
"A Cost Model for Multi-Lifecycle Engineering Design,"
Proceedings of 5th IEEE international Conference on Emerging Technologies and Factory Automation, Hawaii, pp. 385-391, November, 1996.

To my beloved family

ACKNOWLEDGMENT

I would like to express my deepest appreciation to Dr. MengChu Zhou, who not only served as my research supervisor, providing valuable and countless resources, insight, and intuition, but also constantly gave me support and encouragement. Special thanks are given to Dr. Nirwan Ansari and Dr. Reggie Caudill for actively participating in my committee.

Many of my fellow graduate students in the Discrete Event System Laboratory are deserving of recognition for their support.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES
(Continued)

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Design For Environment

Sustainable industrial development seeks to meet current needs of society without compromising the ability of future generations to satisfy their own needs. Unfortunately, our industrial society is not yet on the right path. In order to be able to sustain further the growth and development, we have to make adjustments in the way how we manage the industrial environment interface.

Over the past decade, much research and efforts were put into understanding issues such as waste management and material recovery, as they related to products after they entered the waste stream. Attention is now being focused on the product design. It is known that 70% of the monetary cost of a product is decided in the design stage [3][9]. Once its design is done, almost the whole product life-cycle is set. Design is the most important stage in reducing product environmental impacts and conserving precious resources.

The consideration scope of design activity has been enlarged. Figure 1.1 shows the four main stages of a product lifetime: Manufacturing, Use, Service, and Retirement [5]. Originally, the goal of product design is simple, i.e., Design For Function (DFF). DFF focuses on providing the products which have certain usage. Later on, engineers began to consider more and more about the manufacture and service stages in their design. Design For Manufactureability (DFM) and Design for Serviceability (DFS) are aimed at improvement of convenience for manufacture and service. Most recently the scope was

enlarged again. Design For Environment (DFE), which refers to the practices that are intended to provide products whose aggregate environmental impact is as small as possible [5], is becoming more and more important. It finally entends the design consideration scope to the whole product lifetime.

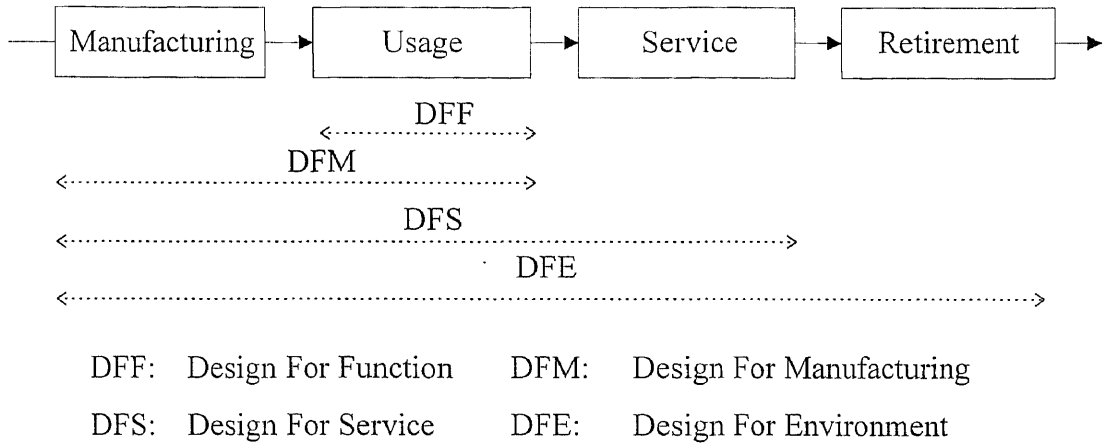


Figure 1.1 The Consideration of Design

The goal of DFE is to generate products which incorporate environmental concerns and to avoid environmental problems before they occur. It is always better to take a proactive approach than a reactive approach which tries to fix problems only after they occur.

1.2 Reasons for Changing to DFE

Why should manufacturers change to DFE? Some of the factors that are pushing companies towards DFE are:

1. Current and anticipated environmental legislation;

2. Corporate image and public perception;
3. Demanding consumers; and
4. Rising waste-disposal and landfill costs.

There are many literature papers which have already described in detail about these factors and here we are not going to discuss them again [1][3][5]. Compared to the above four "have to" reasons, there are also some more important "pulling" factors which are making it financially attractive for companies to invest in DFE projects. DFE can not only save the manufacturers much cost of waste disposal and retired product landfill, but also make them extra money from "waste". For example, when expensive products, such as buses and trains, reach the end of their service life, it is often much less costly to recover old products rather than to purchase new ones. New buses, for example, cost municipalities about $220,000 apiece, while it costs an average of $70,000 to recover one, which brings the old bus back to the same condition as a new bus. Significant price differences create a demand for the recovery of retired products and the research of DFE.

1.3 Product Recovery

Product recovery is an efficient way to cut down the environmental impact. Once a product or part is recovered, the raw material and energy used in it is salvaged.

Several diagrams have been proposed to represent the product recovery process. Figure 1.2 is a general diagram which shows the circular nature of material and energy flows through a product life cycle [2].

A product life cycle can be organized into the following stages:

- Raw Material Acquisition
- Manufacturing
- Assembly
- Use and Service
- Disposal and Landfill

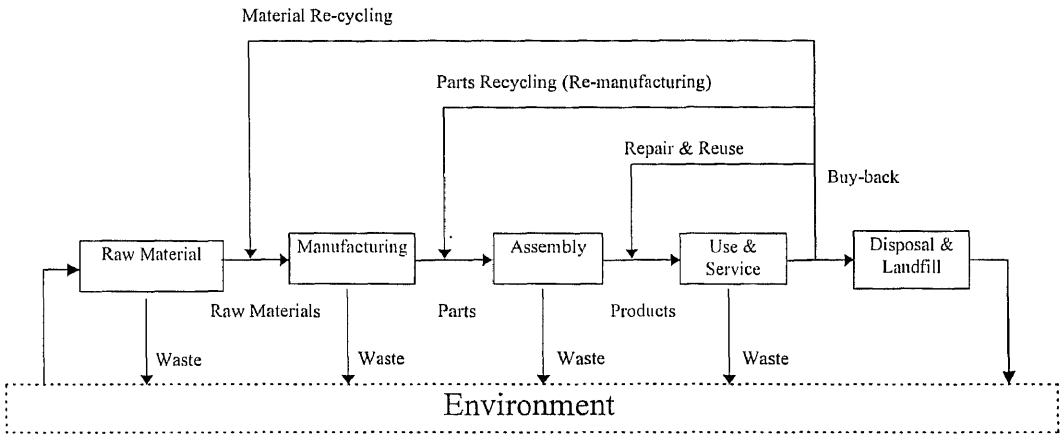


Figure 1.2 Product Recovery Process [2]

According to the condition of a retired product, recovery could be implemented in three forms: Repair and Reuse, Parts Recycling (Re-manufacturing), and Materials Recycling. Most people are familiar with Material Recycling. In a junk yard, hundreds of thousands of cars are waiting to be recycled as metal and go into another product cycle. Both in industry and academia, much attention is given to material recycling. However, more dramatic reduction in environmental impact can be made by product remanufacture and reuse in which the geometrical form of the part and product is retained. With respect to material recycling, remanufacturing and reuse have the two added benefits: 1) not only is

the material waste and amount of landfill reduced, but also energy and matter consumption during manufacture is reduced; and 2) the utilization of existing components reduces an enterprise's monetary cost of producing or acquiring new components. For the products and function blocks which still work well, we may consider to reuse them for the same purpose as during its original life-cycle. Even if some products are not good for the original purpose, we can still consider reusing them for a secondary purpose. For example, we may reuse the automotive tires as mooring cushion in a harbor. Sometimes a product partially fails after its retirement and some of its components are good. We may consider disassembling these components from the block either in parts or in clumps and reassemble them, together with some new replacement parts, to new products.

Recovery is a leveraged process. The original intent of recovery is to relieve the environmental burden. Landfill, in some cases, is not the worst thing. There are many other environmental issues concerned, such as energy consumption, pollution and human consciousness. If we could evaluate all these issues into costs reasonably, we could transform all the criteria into one domain and find an optimum solution by searching the tradeoff of the costs of recovering and discarding. If the overall cost to recover a part is even higher than that to replace it with a new one, obviously it is no longer worthy to recover it. This is the reason why we may not always expect 100% recovery of a product in practice. As shown in Figure 1.2, we have the fourth choice, landfill, for a retired product even though landfill is not what we like. Current research keeps trying to discover new material, new recovery method, and new product design which could reduce the product recovery cost. On the other hand, landfill cost has been increasing and

becomes prohibitive in some countries, e.g., some countries in Europe. As a result, one needs to minimize or completely avoid landfill of products.

If we consider landfill as a special recovery loop, actually we have four recovery loops in total. Repair and reuse is the smallest loop and landfill is the biggest one. Normally, the smaller the loop, the more profit we can gain from the recovery.

1.4 Product Lifetime

In the past, the product lifetime was such a concept that it begins when a product is manufactured and ends when it is out of function. Now the concept is changing. Some high-tech products are out-of-date much earlier than the time when they actually fails. Hence, the endpoint of the lifetime of a product should be the time when it "retires".

In modern industry, manufacturers are responsible for not only providing new products but also disposing their retired products. For example, some manufacturers now have their buyback policy for their products which were sold three or four years ago. This thesis defines lifetime of a product as the time interval between the completion of its manufacture and the time when a manufacturer buys back it for recovery. This definition is useful for designers because the product lifetime is no longer an uncontrollable factor for them. One of the DFE designers' tasks is to set the appropriate product lifetime which results in the most profit of a class of products.

1.5 Computer Aided DFE Tools

Although much research work has been done in this field, DFE is still a new concept to most people. In order to perform effective DFE, a computer aided tool performing DFE

methods transparently is very helpful and may be essential for designers, especially those who have little experience in this area.

DFE will not become widely practised throughout the industry until excellent and easy-to-use computer aided tools are available. The primary purpose of green design is to make practising DFE as simple as possible. Thus, aids or tools ought to present DFE in an easily understandable and usable form. They should enable designers to improve the environmental attributes of products without requiring them to become experts in environmental science and impact analysis. Furthermore, the design advice offered should be presented in such a way as to require little or no additional analysis.

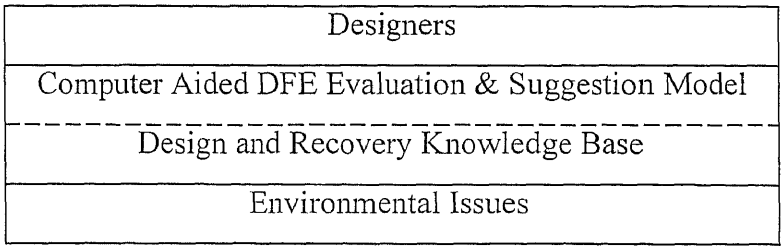


Figure 1.3 Computer Aided Tool and Knowledge Base

Figure 1.3 shows the general idea of a computer aided DFE tool. Using certain evaluation and suggestion model and based on a design and recovery knowledgebase, designers can consider the environment issues and embed the result in their product design.

There are three main challenges in developing good computer aided DFE tools:

1. There is a lack of practical environmental impact assessment methods. Reliable impact assessment methods are needed to transform environmental issues into standard criteria and knowledgebase to ascertain

that design decisions which are based on them will indeed yield products whose aggregate, lifetime impact on the environment is minimized.

2. There is a scarcity of proven environmentally benign material and technology alternatives that can be chosen for designers.
3. The effectiveness of a particular green design solution often depends on an accurate assessment of external parameters, which is hard to predict sometimes.

Some interesting computer aided DFE tools are being developed, such as ReStar [1], LASeR [2], and Cost-Benefit Analysis Model of Product Design for Recyclability [4]. In Chapter 2, we will focus on discussing two of them, ReStar and LASeR. Both of them are powerful DFE Analysis tools. However, they are short of time varying consideration. In Chapter 3, we try to set up a general Multi-Lifecycle product recovery model based on the concept of time-varying product retirement plan. By applying the model on a single part and whole product, Chapters 4 and 5 continue the discussion. Examples are used to explain the ideas. Chapter 6 discusses some of our preliminary computer implementation of the proposed optimization method. Chapter 7 concludes the thesis by summarizing the contribution of the thesis and presenting the limitation of this research and indicating the future research along the line of multi-lifecycle engineering research.

CHAPTER 2

REVIEW OF COMPUTER AIDED DFE METHODS AND TOOLS

As we mentioned in the last chapter, computer aided tools are important for designers to implement Design For Environment (DFE). While no such commercial tools are yet available, some interesting tool development efforts are already under way. In the first two sections of this chapter, we will introduce two typical and successful tools. The first one is a design tool for environmental recovery analysis named ReStar, developed at Carnegie Mellon University. The second one is a service model and recyclability analysis system named LASeR which is developed at Ohio State University. In the third section, we will give a brief review of some other interesting ideas.

There are three key main aspects in a DFE tool: product design representation, environmental issues assessment and evaluation method, and product retirement plan generation and optimization. Our introduction to these tools and methods will focus on these aspects.

2.1 ReStar

ReStar is a powerful computer aided product environmental issue evaluation tool which is developed by Carnegie Mellon University [1]. Navin-Chandra believes that the optimal recovery plan represents a tradeoff between cost, time and environmental distress and one cannot expect that the best product retirement plan is always 100% recycling. The optimal recovery plan search is called the Recovery Problem. ReStar is aimed at detecting

the break-even points which carry the maximum profit. ReStar can help designers find environmentally better design alternatives.

2.1.1 Product Design Representation –ReStar's Disassembly Table

A product design representation is needed for the product environmental impact evaluation. Actually, the product design representation is a portion of the original product design. It ignores the very detailed information and captures the environment related issues of the product design. There are two main reasons why we need a product design representation:

1. Product designs are quite different form each other. It is hard to evaluate the designs in different product representation forms. We need a general product representation form which could be easily used by a DFE analysis model; and
2. We may not need all the detailed information in a product design. What we really want to know are its attributes related to the environmental impact. A simple representation is desired to capture these attributes such that we can check and evaluate the recoverability clearly.

In ReStar, a product assembly graph is represented as a table. Connections are also included in the table as standard parts. The representation is based on specifying, for each part, all the other parts that obstruct it in a particular direction.

Here, we employ an example to show the idea. Figure 2.1 shows a simplified diagram of the assembly of a handlight. The metal head housing is screwed on the same materiel

main housing on which there is a spring welded. The cover is screwed on the head housing to hold the glass. The bulb is screwed on the head housing.

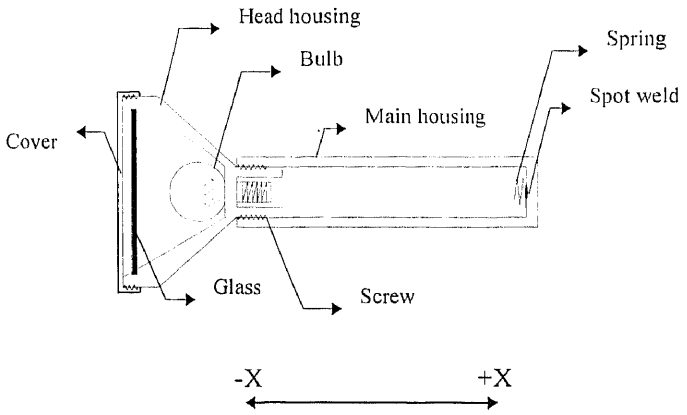


Figure 2.1 A Simple Diagram of a Handlight Assembly

In order to evaluate the recoverability of the handlight, we need its disassembly information. The disassembly feature of this handlight is represented in the form of a disassembly table, as shown in Table 2.1.

Table 2.1 Handlight Disassembly Table

Part	+X	-X
1: Cover	2, 7	Threading
2: Head Housing	1, 7, Threading	1, 3, 7, Threading
3: Bulb	2	1, 7, Threading
4: Main Housing	5, 6, Threading	2, 5, 6, Threading
5: Spring	1, 2, 3, 4, 6, 7	4, 6
6: Spot weld	2	4
7: Glass	2	1

The disassembly table is explained as follows. Consider the first entry of the table. If we want to move the cover in +X direction while all other parts are rigidly held in place, we can see from the diagram that head housing and glass hold it back. These two items are entered in the disassembly table as obstructs of cover moving in +X direction. In this table, an entry which has a threading in any direction means that the entry can be screwed off from that direction. For example, we can screw off the cover from -X direction. The same is done for every part in every direction and we obtain the disassembly information of this headlight assembly.

2.1.2 Automatic Generation of Product Retirement Plan

Once we have this disassembly table, the disassembly algorithm is simple. Two rules are used [1]:

1. if any part is unobstructed in any direction, then it can be removed in that direction, and
2. if any part is held only by a connection in some direction then it can be removed by undoing that connection.

The disassembly rules are applied recursively. Every time a part is removed, all references to that part are removed from the table. For example, if the cover is removed, all its references in rows 2 (Head housing, +X and -X direction), 3 (Bulb, -X direction), and 7 (Glass, -X direction) can be removed. Now we can see the glass is free in -X direction. Thus, next step the glass can be removed and so can all its reference in rows 3 and 4.

Step by step, we can generate the disassembly graph as shown in Figure 2.2.

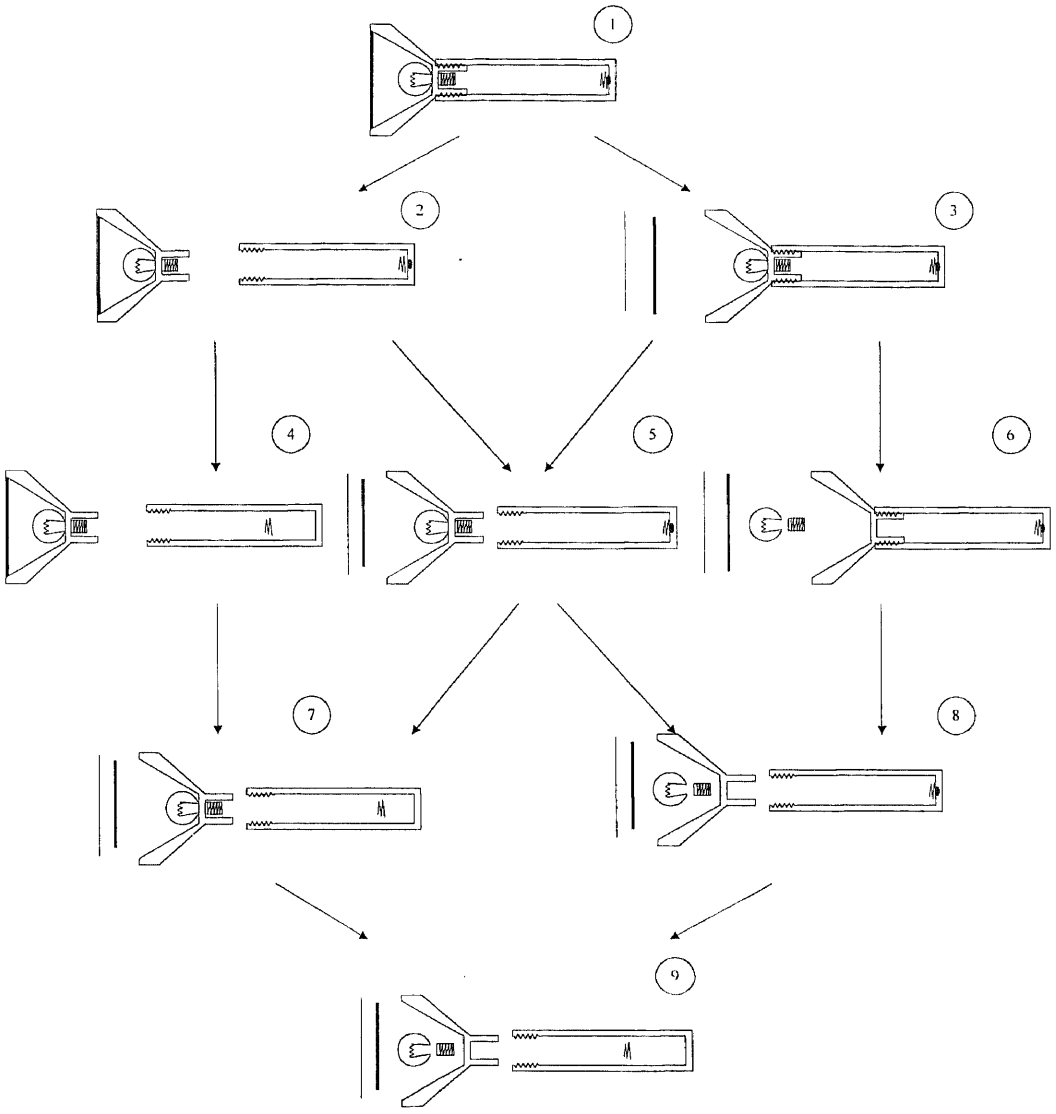
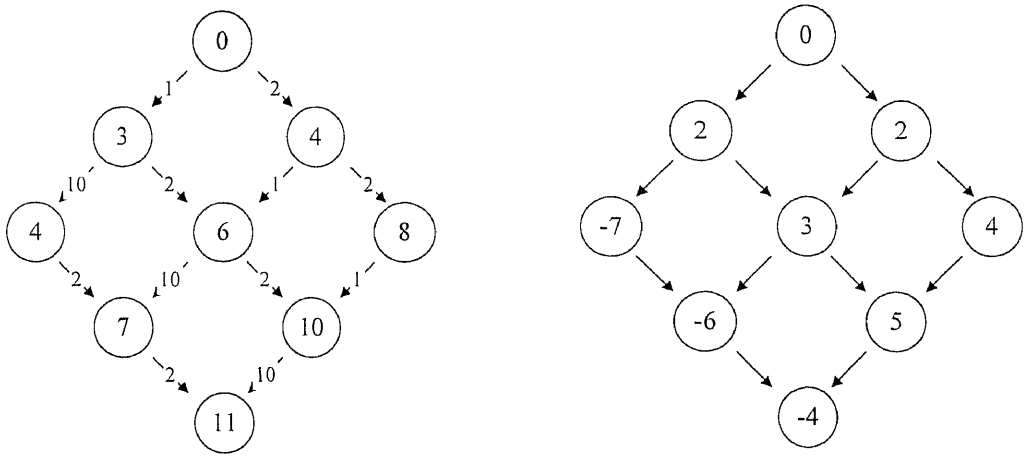


Figure 2.2 Disassembly Process Graph of Handlight

In Figure 2.2, each disassembly status is numbered and each step from status to status denotes one step disassembly procedure which just breaks one connection. Some statuses

have multiple disassembly choices. For example, at status 3, we can either unthread the screw and go to status 5, or unthread the bulb and go to status 6. Therefore, such disassembly process graph may not be unique.



(a) (b)

Figure 2.3 Disassembly Cost Tree for Handlight

Figure 2.3 is the disassembly cost tree of the handlight. The numbers shown in the circles and on the arrows are the revenues and cost of the disassembly procedure, respectively. For example, the number 3 in the left node on the second row means that the revenue from the top to this node is 3. The number 1 on the line between the node with 0 and that with 3 means that the disassembly cost for this step is 1. Figure 2.3(b) shows the net cost of the disassembly process and in this diagram, the number in each circle represents the overall cost. We can see that the best status is the right node on the second row from the bottom and the net revenue is 5. The physical meaning of this optimal node is

disassembling Cover, Glass, Bulb, and Head housing and material-recycling the Main housing and Spring as a clump.

During the process of disassembly, ReStar keeps track of disassembly costs and revenues. It also keeps track of subassemblies of compatible materials. Comparing all the cost and revenues, ReStar can find the best disassembly statues and the best retirement plan.

2.2 LASeR

LASeR, a computer aided Design For Product Retirement (DFPR) tool, is developed by Ohio State University. The name LASeR stands for Life-cycle Assembly, Serviceability, and Retirement. LASeR allows designers to specify a design in terms of a LINKER model, which is a graph of parts and connections. It also allows designers to assess disassembly cost and recycling cost. The designers can change materials and joints to improve their design.

2.2.1 LINKER—The DFPR Product Design Representation Method

The original LINKER is a method used to evaluate layout designs for manufacturing and life-cycle serviceability [2]. Marks et al. [7] modified the LINKER to support DFPR.

LINKER models the structure of a product and captures the necessary data for evaluation. We use another example to explain the idea. Figure 2.4 shows the simple assembly diagram of a food processor.

The food processor has 9 main components. Its structure and recovery intent information is captured in the LINKER model which is shown in Figure 2.5. LINKER is a hierarchical semantic network comprising components, subassemblies (node) and links

which are the relationships between the nodes. Links could be actuarial connections between components or other geometrical relationships, such as supports. In general, nodes contain data for material type, part of material cost, part weight, the name of the item or process, a user-defined part number or code, and the next higher assembly (if applicable). Links contain data for link type, removal and installation time, and fastener type.

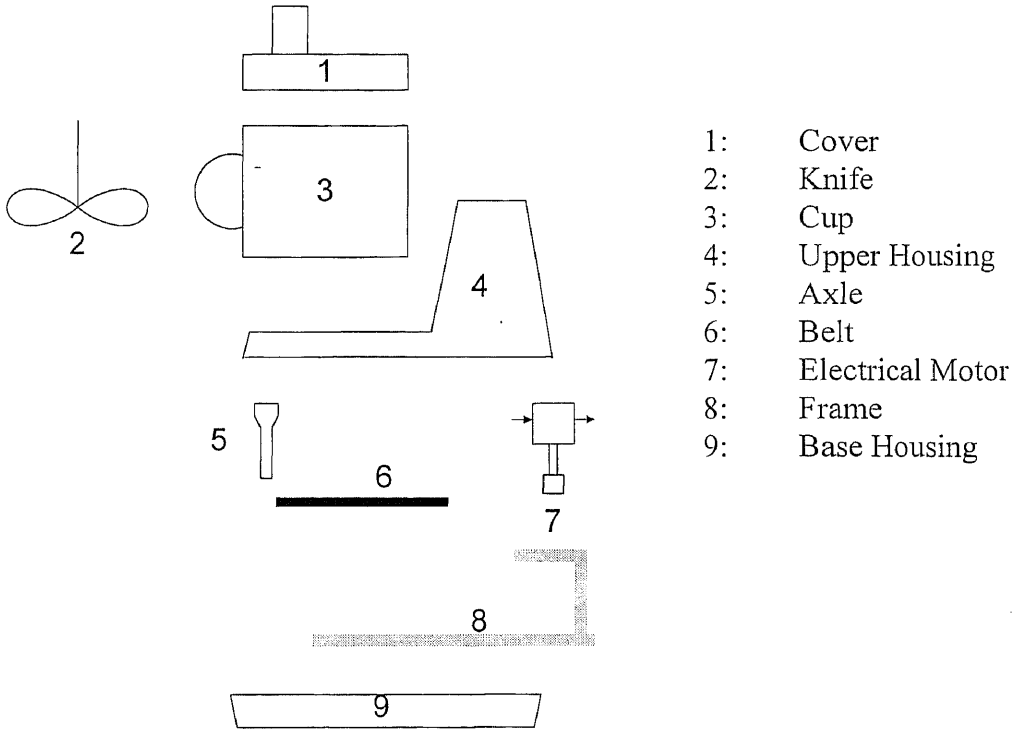


Figure 2.4 Simple Assembly Diagram of a Food Processor

There is a kind of special nodes called "clump" which denotes as the dot-line ellipse in Figure 2.5. A "clump" is a collection of components which share a common characteristic based upon the designer's post-life intent: reuse, remanufacture, and material recycle [2].

The disassembly stops at clumps and it is recovered as a whole. Clump is a powerful idea because it allows the user to consider partial disassembly strategies [1]. For example, in Figure 2.5, cover, knife, and cup are grouped as a clump to be material recycled. Axle, belt, and electrical motor are grouped as another clump to be reused.

LINKER is a kind of structural representations. It represents the geometrical and topological characteristics which are pertinent to recoverability evaluation. Together with the representation, there are a database which records object-oriented environmentally related data of entries for each node and link. Once we have this representation, the inference of disassembly steps becomes a network search that results in a list of links that must be addressed to disassemble a product.

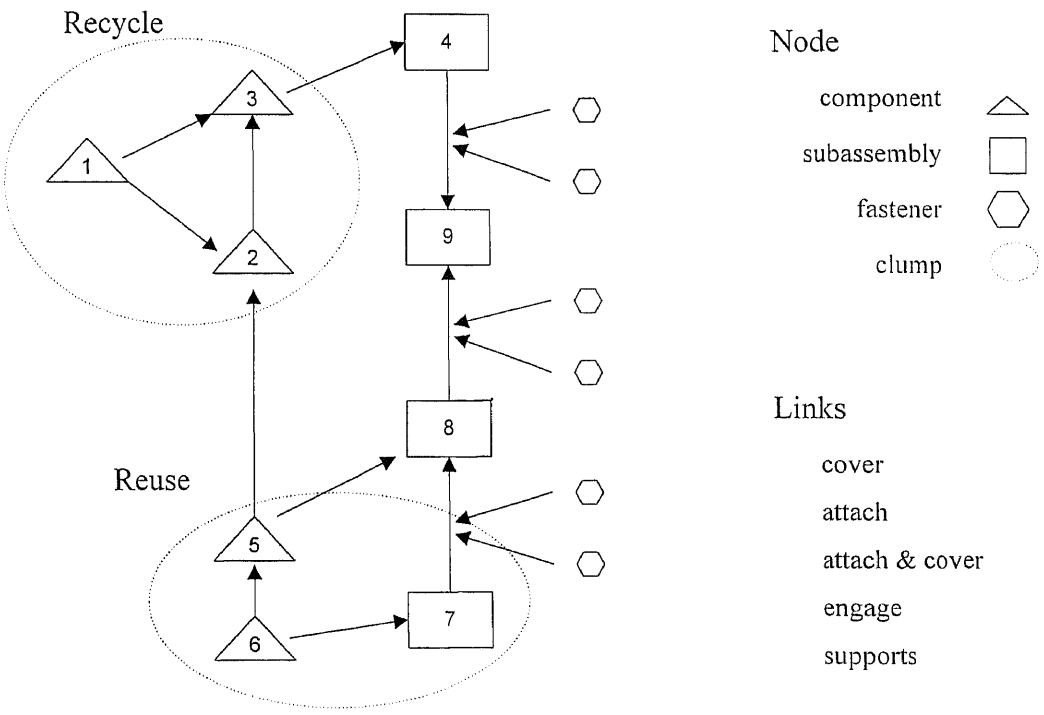


Figure 2.5 LINKER Model for Food Processor

2.2.2 Reverse Fishbone Diagram

Several groups in Stanford's graduate DFM curriculum found that LINKER did not fully capture the details of the retirement specification needed to effectively evaluate the product recoverability. Ishii et al. [7] formalize the reverse fishbone diagram to supplement the LINKER in developing a DFE tool.

Reverse fishbone diagram is an emerging essential analytical tool in the design and evaluation of product retirement processes to minimize the product environmental impact. It helps the designers to identify the disassembly complications and difficulties and ensure that product retirement concerns are addressed up front [7].

Figure 2.5 shows the core idea of a reverse fishbone diagram using the food processor example which is used in the last subsection. As shown in Figure 2.5, the first assembly seperated from the food processor is Cup Assembly which is material recycled as a clump. Then part Upper housing is removed and also recycled as material. The next assembly disassembled is the Electrical Motor assembly and it is reused as a clump. The parts in this assembly include Belt, Axle, Electrical Motor, and Frame. The last part is Base Housing which is recycled as material. Reverse fishbone schematically describes the disassembly steps for a product and specifies the retirement intent for each component and clump.

From Figure 2.6, we can see that the Reverse Fishbone method can show not only the part post-retirement recovery intent, but also the disassembly process of the product.

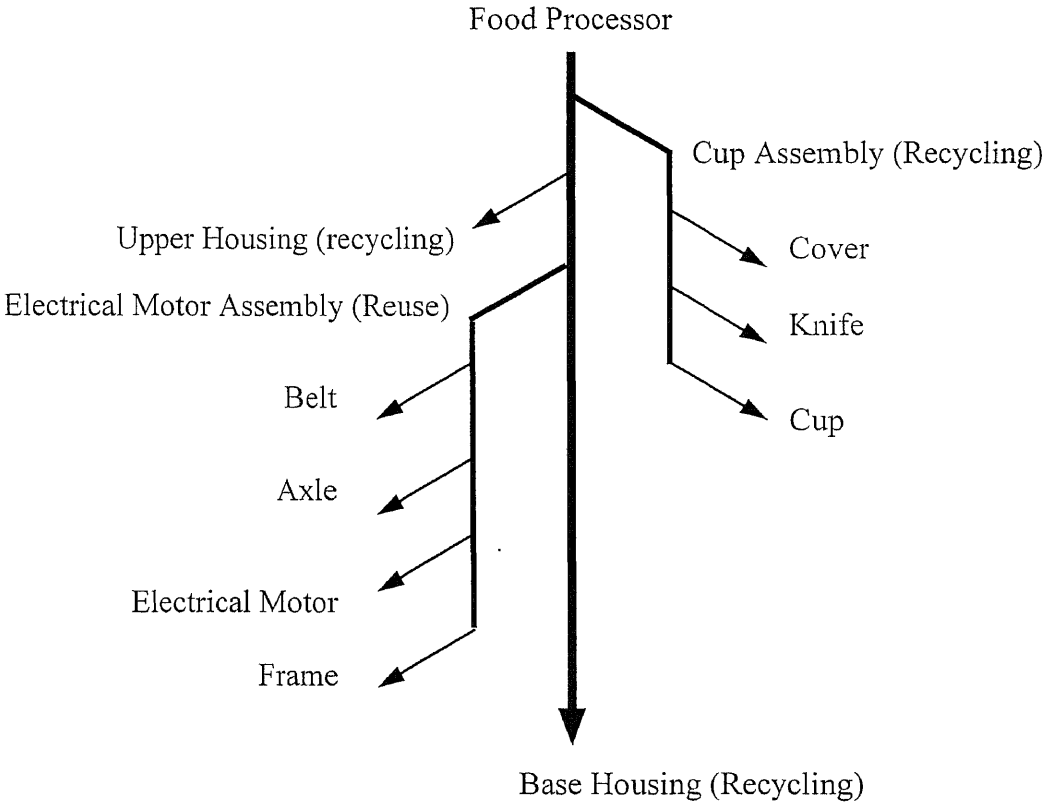


Figure 2.6 Reverse Fishbone Diagram for a Food Processor

2.2.3 DCA -- Environmental Issues Assessment Method

In order to evaluate designs, we have to have an assessment method which could transform different kinds of environmental issues to the domain which we can easily evaluate them.

For example, the residual value of a clump which consists of many different kinds of materials is an important data for calculating the recovery cost of a product. The residual value of a clump is a function of the compatibility of all the materials in it. Thus, we need a method which can reasonably transform the compatibility into residual value. A typical

environmental issues assessment method Design Compatibility Analysis (DCA) is briefly introduced below, which is developed by Ohio State University [2].

DCA can transform the compatibility of a clump to its recovery cost. The transformation is based on a material compatibility expert knowledgebase. The knowledgebase contains compatibility rules, or called C-data. Each C-data contains a compatibility adjective which maps to a [0,1] rating, as shown in Table 2.2.

Table 2.2 DCA Rating Assignments for Material Compatibility Table [2]

Level of compatibility	DCA rating
"Very compatible"	1.0
"Compatible"	0.8
"Some level of compatibility"	0.6
"Incompatible"	0.2
"Hazardous"	0.0
"No_information"	0.5

DCA automatically checks the knowledge for the compatibility of every component with other component, connection, and process in the part or clump. It creates a set of [0,1]. DCA then map the set of [0,1] into a single part or clump compatibility rating $CC(s) \in [0,1]$ for each part or clump s, using the following function:

$$CC(s) = \begin{cases} \text{Max DCA}(x), \text{ if } \forall x \in s \neq \Phi, \text{DCA}(x) \geq 0.5; \\ \text{Min DCA}(x), \text{if } \exists x \in s, \text{ such that. DCA}(s) < 0.5; \\ 0.5 \text{ if } s = \Phi, \text{ indicating neutral compatibility.} \end{cases}$$

Once we obtain CC(s), the part or clump recovery cost is estimated as follows [2]:

$$CRC(s) = LFC(s) \times \frac{\ln(CC(s))}{\ln(0.2)}$$

where

CRC(s) = Part or Clump Recovery Cost

LFC(s) = Land Fill Cost

CC(s) = Part or Clump Compatibility (real number in [0,1])

In summary, DCA is a two-step method. First, it maps design information, the retirement plan and the compatibility knowledge into a rating between 0 and 1 inclusive. Then, it translates the rating to the recovery cost using an experience function.

2.2.4 Product Retirement Plan Generation

The goal of LASeR is to provide a quick analysis and what-if capability based on the system structure and retirement strategy. It focuses on making it easy to change and adjust the structure and strategy representations. In LASeR, the product retirement plan generation relies on designers themselves and designers fully control the analysis of product disassembly and reprocessing.

Compared with LASeR, ReStar focuses on finding the best product retirement plan. It uses a component graph and disassembly table to perform an automated search and analysis for optimal retirement plans based on the level of disassembly and component material compatibility. LASeR pays more attention to evaluation of the existing product design and retirement plan.

2.3 Material Mortgage and Stepped Obsolecence

Beside product recovery, Navin-Chandra also introduces two interesting ideas which are important in reducing the product environmental burden [1].

One is "material mortgage". If there is a durable and expensive part in the product which a customer needs buy every few years, the part can be sold to the customer with a long-term mortgage. For example, sometimes designers use gold on the connectors of a Hi-Fi audio in order to improve the performance. This, however, will raise the product price greatly. Fortunately, the gold connector is durable and can be used for a long time. Suppose that a custom will buy a new Hi-Fi audio about every five years and trade-in the old one. The manufacturer may consider to sell the gold connector to the custom with a long-term mortgage. This method can spread out the large initial investment over several product lifetimes and provide very high-quality products without raising the price.

The other idea is "Stepped Obsolescence". Some customers are called "early adopters" because they look for the latest and best features. Some others who may care more about the price are called "late adopters". The idea of Stepped Obsolescence is to set up a trade-in system where manufacturers can take back the product from the early adopters in a couple of years and re-sell it to the late adopters at a lower price. Stepped Obsolescence could cut down the product environmental burden by extending the lifetime of a product.

CHAPTER 3

COMPUTER AIDED PRODUCT DESIGN SYSTEM

3.1 Computer Aided DFE

As discussed in the first two chapters, on top of functionability, manufacturability, and serviceability, designers are now asked to contribute to reducing the environmental impact of products. Although some DFE guidelines and checklists are available to help designers implement DFE, they are not much helpful for designers to perform DFE efficiently. Most likely, DFE will become widely practised only after good and easy-to-use computer aided DFE tools become available and acceptable for designers.

Product design is such a complicated activity that it is very difficult to find out a general method to improve the product environmental friendliness in a direct way [1][2]. As an alternative, most methods proposed now help designers in an indirect way. Figure 3.1 shows a typical DFE system structure diagram. We can see that it uses a feedback mechanism. There are already many standard CAD tools which can help designers implement the draft design, the first part of this system. In order to make environmentally better draft designs, we can also embed some guidelines and checklists into these standard CAD tools.

The second part of the system is the design improvement which consists of the processes in the dot-line box in Figure 3.1. As the feedback and modification part of the design system, the second part is important for the improvement of product environmental friendliness. However, there is no successful commercial computer aided tool which can fulfill the function of this part.

The design improvement part consists of three processes. The first process is retirement plan generation. The draft design specifies the product issues in manufacture, use, and service stages. Retirement plan generation process completes the whole product recycle lifetime specification by settling down the product issues in the retirement stage. The second process is environment impact evaluation. Using certain evaluation method, we can obtain the environmental impact features of our product design [2]. The feedback information results from the comparision between the actual evaluation results and designers' expected ones. It is used to guide designers to modify and improve the environmental friendliness of their products.

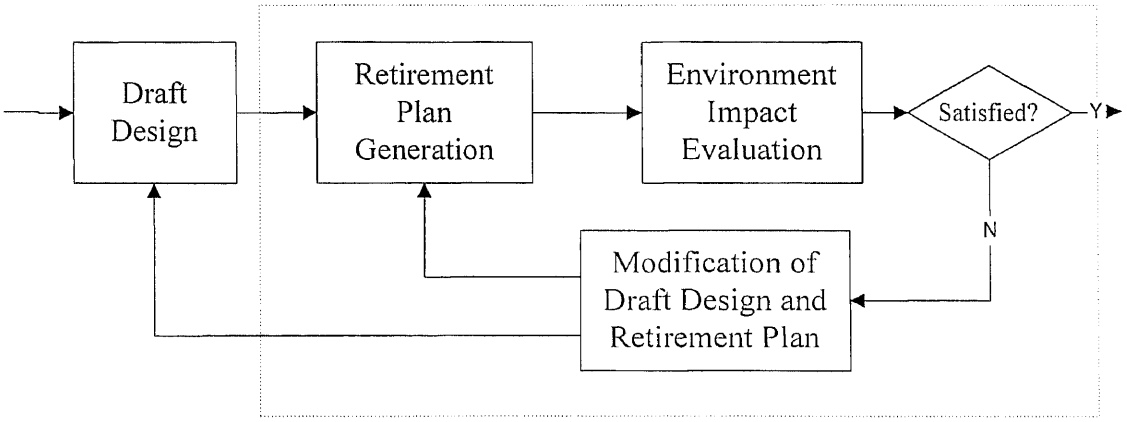
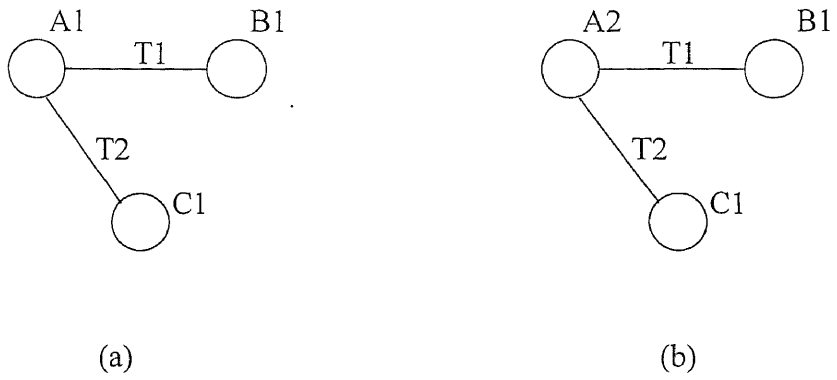


Figure 3.1 A Feedback DFE System Structure

A simple example is employed to show the idea of such a DFE system. Suppose that a simple product consists of only three parts, A, B, and C. Figure 3.2(a) is its draft design. The connection type between Parts A and B is Type 1 (T1). The connection type between Part A and C is Type 2 (T2). In the draft design, designers choose type 1 of all the three parts. As shown in Figure 3.2(a), we denote them as A1, B1, and C1.



(a) (b)

Figure 3.2 A Simple Design and Modification Example

Suppose the retirement plan of this product generated by Retirement Plan Generation process is

- To break T1 between A and B, and reuse part B, and
- To material-recycle part A and C as a whole, since T2 is very hard to break.

Using some evaluation method, we find this draft design and retirement plan is not satisfactory and the problem is that A1 is not compatible with C1. In order to improve the product design designers modify the draft design. They change Part A from type 1 to type 2. The modified design is shown in Figure 3.2(b). This time the design is better and satisfying all the manufacturability, serviceability and environmental considerations. So it passes the design stage.

Most related studies focus on the first two processes of design improvement and many interesting results are reported [1][2][4]. However, the third process as the most important step of design improvement was not well addressed in the previous research. Compared with the first two processes, Modification of Draft Design and Retirement Plan is harder to model and thus more difficult to implement in a computer aided tool. In the last

example, following a certain algorithm and model, designers can use a computer aided tool to evaluate the draft design and the retirement plan and obtain a result. But how can a computer know where the problem is and how the product design can be improved? As we know, a computer is such a machine that it is good at "calculating", but not good at "thinking". It is easier for a computer to search all the possible situations of a product design than to figure out how to improve it. Thus, the alternative solution is to generate the retirement plans of all the possible designs for a product and choose the best one from them. We modify DFE system structure in Figure 3.1 to the candidate set structure as shown in Figure 3.3.

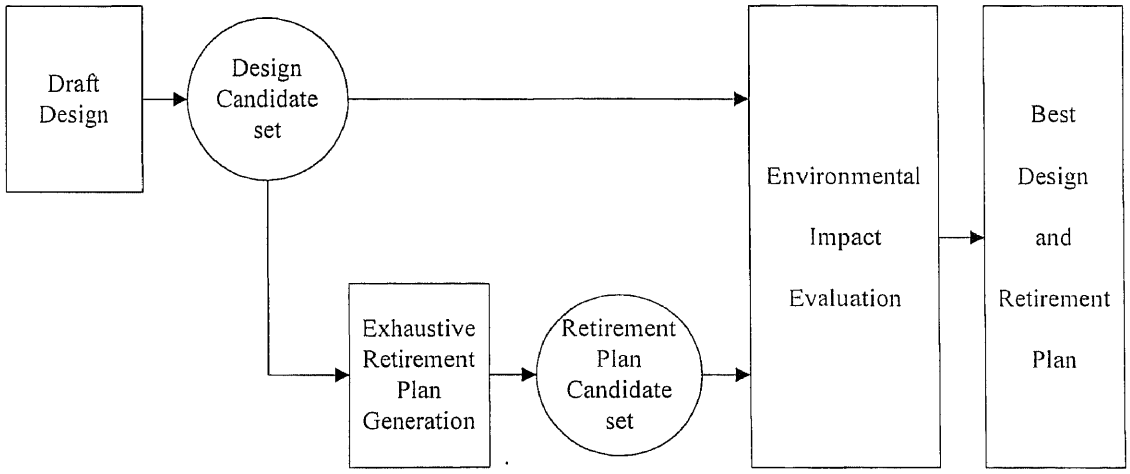


Figure 3.3 A DFE System Structure Using the Concept of Candidate Set

Instead of just one, the draft design process generates a set of product design candidates. All the retirement plans of these design candidates are also generated. Base on the

evaluation of the environmental impacts of these designs and their retirement plans, we can obtain the best one whose environmental impact is the least.

The following sections continue discussing how to use a computer to help designers implement DFE. Section 3.2 focuses on how to represent the DFE design and Section 3.3 proposes a Multi-lifecycle cost model to evaluate designs.

3.2 Product Design Representation

Section 2.1 indicates that direct evaluation of a product's draft design is inconvenient. A product design representation is needed to capture its necessary information for the purpose of environment impact assessment.

In order to evaluate the recoverability of a product, we need to know:

1. how well all the part materials are compatible to each other; and
2. how easily the product can be disassembled;

Thus, a product design representation should cover at least three aspects of information:

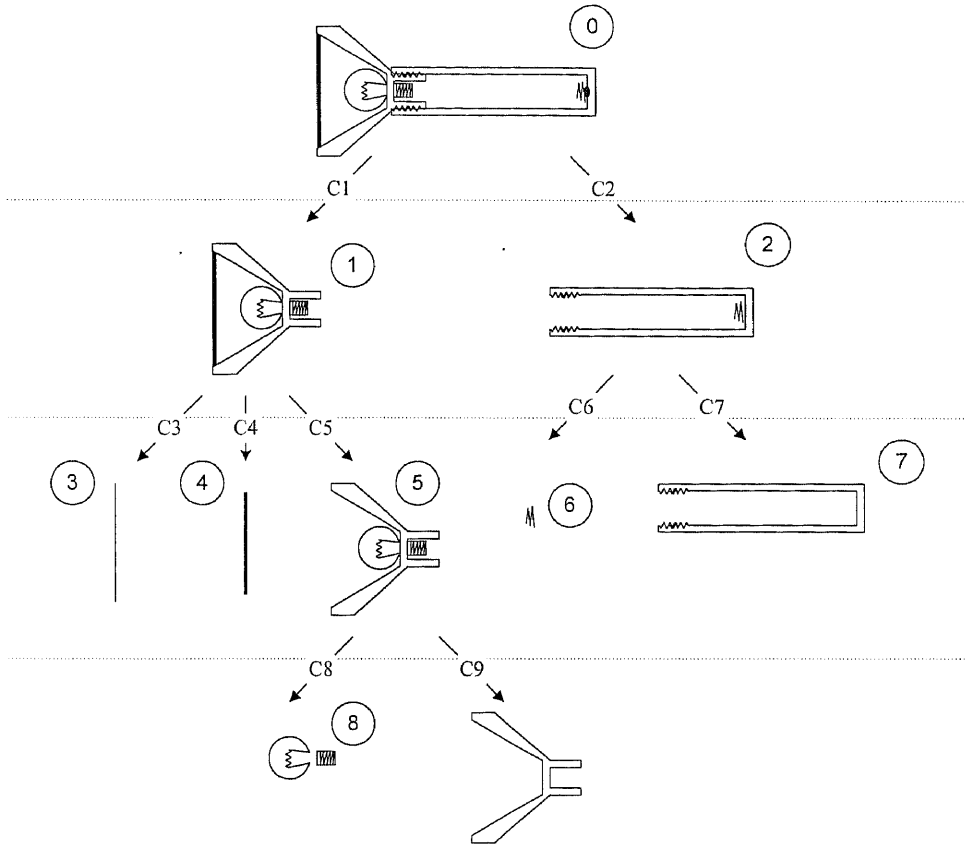
1. each part type and material selection;
2. each connection type among these parts; and
3. the product structure.

3.2.1 Tree Structure and Product Layout

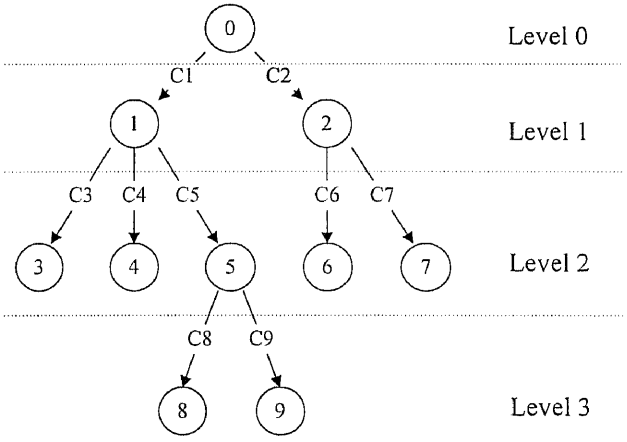
The relationship between parts in a product can be represented as a tree structure as shown in Figure 3.4. The root of the tree is the product itself. It is the most abstract level of a product design. An assembly at level k can be disassembled into several subassemblies at level k+1. As the level goes down, the information becomes more

detailed. For example, at level 1 we can say that the handlight consists of the head assembly and cell housing assembly. At level 2 we have more detailed information and we know the handlight consists of four parts, i.e., Cover, Glass, Spring and Main housing, and one assembly, i.e. Bulb assembly. The leaf nodes in this tree represent the parts which are basic units in the product and cannot or will not be disassembled anymore.

The tree structure is used to set the abstract level of the candidate sets. As we mentioned earlier in this chapter, all the possible design choices of a product can be represented as a candidate set. Actually, when designers specify a part, a connection, and the structure of a product, they have many choices. The choices of each element constitute the candidate set of that element. If there are well developed candidate sets for all the elements (parts, connections, and structure) in a product, product design becomes a process of selection from these candidate sets [12]. Sometimes the candidate set may be not rich enough and there is no suitable candidate, designers need to create another better candidate. The candidate creating activity expands the candidate set and provides more choices for the designers to perform the similar design next time. Currently, much Green Engineering research focuses on generating new material, new manufacture and assembly process, and new structure of products. Its objective is to enrich the candidate sets and give designers more and better choices.



(a)



(b)

Figure 3.4 A Tree Structure for Handlight

Designers may consider constructing a product using candidate sets at different abstract levels. For example, designers may have a candidate set for cell housing assembly at level 1 and candidate sets for spring and housing at level 2. Product designs based on different candidate set considerations are different. Thus, the first design issue needed to be set is the candidate set abstract level.

How to set the candidate set level depends on two factors. The first factor is the amount of available information. Sometimes, designers do not have all the condidate set for all the nodes in the tree. Designers have to conduct their design based on what they have. The second factor is the designers' intent. Sometimes, designers would like to use certain parts or subassemblies and they do not want to change them. A product tree structure clearly shows the relationship of parts and assemblies. Designers can use it to determine the abstract level of candidate sets.

For example, designers may set the abstract level as level 2 for the handlight example in Figure 3.4(a). Thus, they need not consider the disassembly of the bulb assembly. Then, the basic units of the handlight are considered as Cover, Glass, Bulb assembly, Spring, and Main housing.

Once the candidate set abstract level is set, basic units of the product are determined. The connections of these basic units can be represented by a LINKER model. Figure 3.5 shows the LINKER model of the handlight based on abstract level 2.

From Figure 3.5 we can see that there are five part and assembly elements and four connection elements. Suppose that there are N_i candidates in the candidate set of the i^{th}

element which is a part, subassembly or connection, there are $\prod_{i=1}^{9} N_i$ different designs of this handlight.

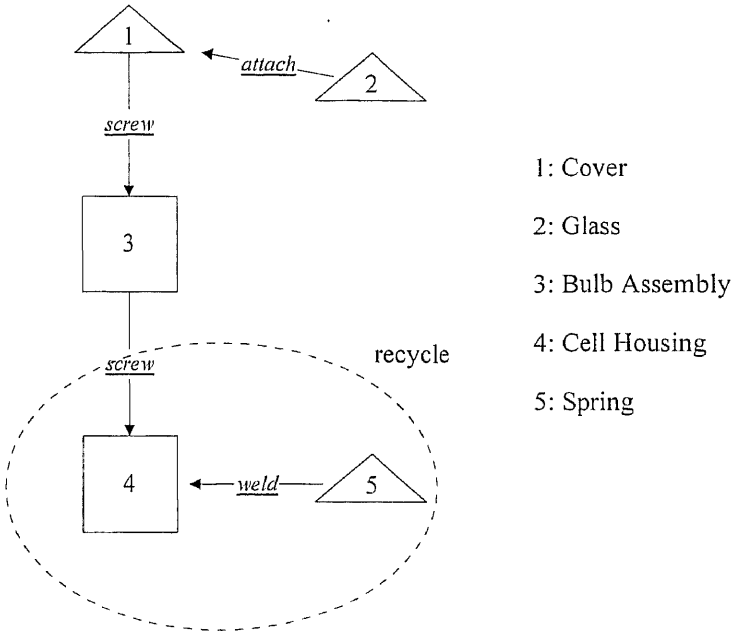


Figure 3.5 LINKER Representation of Handlight

3.2.2 Product Retirement Plan and Disassembly Paths

A LINKER model can capture the necessary product design information. However, it is weak in representing a product retirement plan. A product retirement plan represents the product post-retirement consideration. It defines the retirement stage features of a product. There are two key points in a product retirement plan. One is the product disassembly process. The other is the post-retirement recovery intent of each part and assembly.

Kosuke Ishii, et al., [7] developed a Reverse Fishbone method to represent a product recovery process. In this method, designers set a product retirement plan. If a plan is not satisfactory, designers need to consider modifying and improving it. Here the problem comes again: How can designers modify the plan to improve it and know whether a retirement plan is the best for this product or not? In this subsection, we will introduce the concept of disassembly path and discuss how to find the best retirement plan for a product.

Consider the example shown in Figure 3.2. This simple product consists of three parts, A, B, and C. There are two connections which are T_1 and T_2. Suppose the retirement plan is to totally disassemble this product. Considering the disassembly order, there are two different disassembly processes which are shown in Figure 3.6.

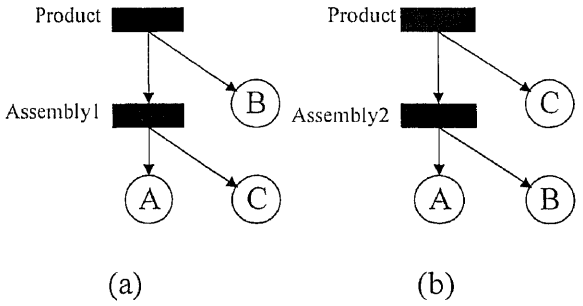


Figure 3.6 Two Disassembly Paths

We call the two disassembly processes "disassembly paths" of the product design. A disassembly path is a possible disassembly process form top (product) to bottom (parts) of the product. We use the word "possible" because the real disassembly process may not necessarily follow the path to the end. It may stop at some nodes.

In order to represent a disassembly situation, we introduce the "stop node" concept. For example, if the recovery intent of Assembly2 in Figure 3.6(b) is part re-manufacturing, the disassembly will continue to next level. If its intent is to landfill, which means that Assembly2 will be landfilled as a clump, the disassembly process will stop at Assembly2 and in this case, we call Assembly2 a "stop node".

A disassembly path has a set of stop nodes. Whether a node is a stop node or not is decided by the comparison of the costs among its choices. We will fulfill this discussion in Chapter 5.

The disassembly path and stop node set capture the disassembly features of a product. The stop node set represents the best recovery plan in its disassembly path. For one product layout, there may be many possible disassembly paths and we can find the best recovery plan for each of them. Comparing the overall best plans for all possible disassembly paths of a product layout, we can find out the optimal retirement plan for this product design layout. This is the basic idea of the optimal retirement plan generation.

3.3 Product Recovery System and Product Design Evaluation

In this section, we will discuss how to evaluate a product design and its retirement plan.

As we mentioned before, once the design of a product is done, its whole lifetime is almost set. The objective of DFE is to provide an optimal product design whose environmental impact in its overall lifetime is as little as possible.

In every stage of product lifetime, environment resources are consumed and wastes may be generated. In order to provide products, manufacturers have to mine renewable and nonrenewable material, purify the bulk materials into engineered materials, manufacture

the materials into parts, and assembly the parts into products. All these activities require resource and energy. In the usage and service stages, resource and energy may also be required to operate them, repair defects and maintain their performance.

In addition to problems created by resource and energy consumption themselves, wastes and residuals which are produced by these problems also generate significant environmental impacts. Many residuals are temporarily concentrated in landfills. Both the disposal cost and the cost for landfill space are rising.

As we discussed earlier in this chapter, for a certain product, a manufacturer may have a set of design candidates. Since designs are different, their environmental impacts are different. Certainly we want to choose the one whose overall environmental impact is the least. Thus, a method to evaluate the environmental impact of product designs is needed.

The product environmental impact is complicated. There are many aspects of impacts and they are often related to each other. For example, sometimes if we want less toxic waste to enter the environment, we have to spend more resource and energy in the waste disposal process. It is hard to compare these different aspects of issues in a direct way. In order to evaluate a product's overall environmental impact, we should reasonably translate all the criteria of different domains into a single domain.

Some criteria can be easily translated into costs. For example, the environmental cost of raw material and energy used by manufacturing and serving a product can be represented by their prices because the price, to some extent, is an environmental impact index. The environmental cost of a kind of waste generated in a process can be represented by the disposal cost and government fine on it. However, some criteria are very difficult to estimate and translate to costs because there is lack of the accurate date for waste,

especially the waste gas . Sometimes the manufacturers do not know the type and amount of waste which are generated by a manufacturing process and the accurate information of their environmental impact.

Although the research of product environment impact is still in its infancy, some interesting results have already come out, such as the Netherlands VNCI system and the IVL/VOLVO EPS system [13]. In IVL/VOLVO system, first the environmental indices for each raw material and emission are calculated by a team of environmental scientists ecologists, and materials specialists. Then, according to these indices, the product environmental impact are translated into Environmental Load Unit (ELU), which is a special kind of cost.

To develop a reasonable method to translate all the environmental issues into costs requires the research results of chemistry , physics, meteorology, climatology, biology, political science, economics and a variety of more specialized disciplines. This is an important topic for the future DFE research.

The first thing to find out the best design is to set the optimization scope. Two issues may be considered when designers set the scope. One is the optimization level. Product design optimization could be done at part, assembly, product, and product set levels. The level selection is based on designers' intent. The other is the time interval. The optimal design for a product intent for two year use may be different from that for four year use. Time interval setting also depends on designers' intent.

Once we set the optimizaton and evaluation scopes, we can consider the whole product recycle process as a blackbox system. As shown in Figure 3.7, the product environmental impact in its time scope is the system input minus the system output. System inputs

include the costs of material, energy and landfill space which are used to manufacture, use, maintain, and recover a product and disposal the industry waste. System output is the residual value of components which go out of the system and enter the environment.

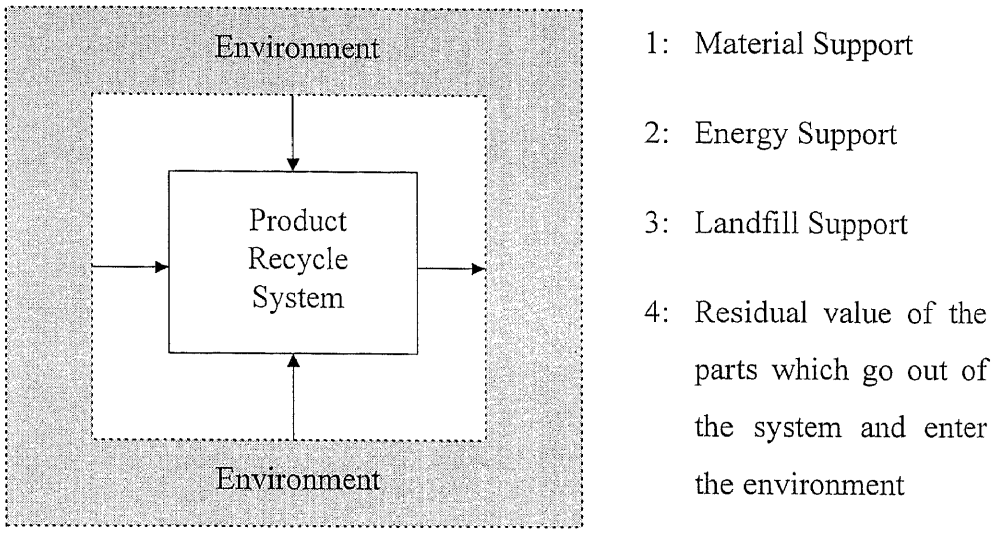


Figure 3.7 Product Recycle System

In a product recovery cycle, some parts may go out of the system. The residual value of these parts can be considered as a special kind of system revenue. The part may leave the system in the following three forms:

1. At the end of the time scope we set, some parts are still worthy to recover and can be used into another lifecycle of the same type of products. The recover choice could be reconditioning and material recycling;
2. The product recovery shown in Figure 1.2 is a closed loop. But actually, some parts which no longer satisfy the requirements of certain level products may

still be usable for a lower level product. The product recycle could be an open loop as shown in Figure 3.8;

3. If the value of a retired part reused or recycled for a secondary purpose is higher than that for the original purpose, this part may leave the original product recovery cycle and enter the secondary product cycle. For example, in Section 1.3 we mentioned the automotive tire example. Suppose that the value of the material recycled from a retired tire is $4 and the recycle processing cost is $2. Then, the residual value of this retired tire for its original purpose is $2. The retired tire can also be reuse as a mooring cushion. If its value for this second purpose is $6, it is higher than $2. For such cases, the retired tire should go out of the tire recovery system and its residual value is treated as $6.

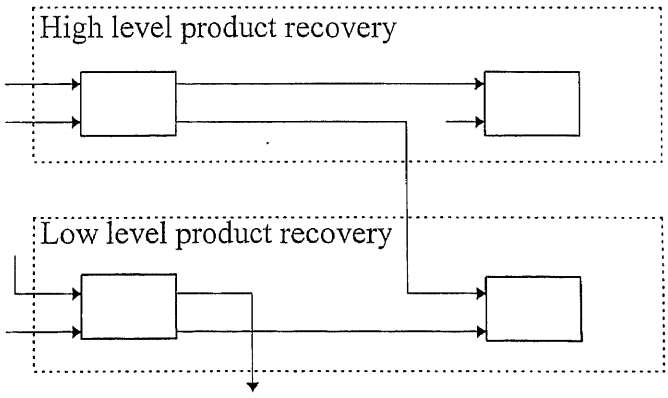


Figure 3.8 Open-loop Part Recovery

Therefore, we have the following index to represent the product system's overall environment impact density

$$\Delta = \frac{\sum \text{Material Cost} + \sum \text{Energy Cost} + \sum \text{Landspace Cost} - \sum \text{Residual Value}}{\text{Considered Time Period}}$$

3.4 Time Varying Cost and Multi-Lifecycle Product Recovery Model

This section introduces the concept of time varying cost and a Multi-lifecycle product Model. In order to know the inside situation of the black box in Figure 3.7, consider how a single part is recycled in the system first. Then the product recovery system is discussed.

3.4.1 Part Lifetime

When a product retires, its lifetime ends. But the lifetimes of the parts in this product may not end yet. Recovery expands the part lifetime.

What is the part lifetime? We define it as the service time of a part. Ignoring a part's further disassembly, we have three choices for a retired part which are reconditioning, material recycling, and landfilling. When it is recycled for material, a part "disappears" and the recycled material is used in a brand new part next time. Only the part reconditioning has the service time memory. The part lifetime ends when the part is landfilled or recycled, whichever comes first.

There are several recovery choices for a retired part. Which is the best depends on the comparison of the costs related to each choice. Thus, after each product lifetime the part recovery choice may be different. This time the reuse is the best for a part, but next time

material-recycle may be better. This is an important idea because it means that after each product life the retirement plan for its parts and subassemblies may be different.

The model shown in Figure 1.2 is a representation of product recovery. It shows all the recovery choices clearly. However, it is short of time consideration. Since the material will degrade with time to a greater or less extent, all the cost and revenue items should be a function of time. It is easy to understand that the residual value of a part which has served for four years is lower than that of the one which has just used for a few months. As an example, consider Figure 3.9. At the ends of first and second product periods, a part may have high residual value so that it is worthy to disassemble it from the whole product and reuse it in a new product. But at the end of the third product period, it is no longer worthy to recover it. It is either landfilled or material recycled.

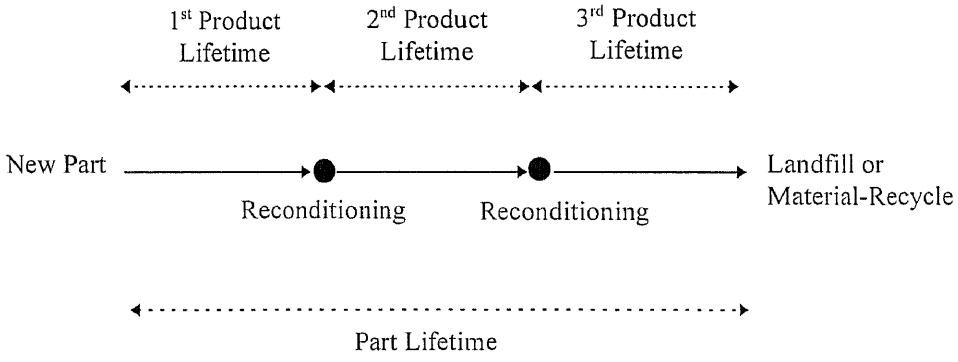


Figure 3.9 Part lifetime and product lifetime

3.4.2 Multi-Lifecycle Product Recovery

Parts have lifetimes and their lifetimes are different. Therefore, the recovery situation is different from one product lifetime to another. We should extend our views to consider a few product lifecycles instead of only one. Although the model shown in Figure 1.2

considers the product recovery, it still focuses on single product life. In fact, the meaning of Multi-Lifecycle is quite different from several same single lifecycles. Therefore, we should modify the product recovery model accordingly.

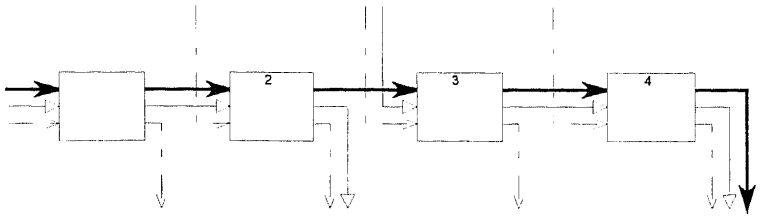


Figure 3.10 An Example of a Multi-Lifecycle Product Recovery Process

As an example of multi-lifecycle consideration, in Figure 3.10 the lines in different styles represent different parts in a product, i.e., Parts A, B and C are represented by bold solid, solid and dotted lines, respectively. Each block represents a product period or lifecycle. As we see, at the end of the first lifecycle, two of the three parts, Parts A and B, are recovered into a new product lifecycle. Part C, whose "life time" is just one product lifetime, is landfilled and replaced by a new part. After the second lifecycle, Part B, at the time after two product periods, is discarded while Part A can still be reused in another new product. At the end of the fourth product lifetime, all the parts are landfilled.

Figure 3.10 shows the way how components are recovered and recycled in the multi-lifecycle product recovery system. As we mentioned before, once a product design is set, its recovery system is determined. In order to evaluate this system, first we should set the candidate abstract level and the time consideration scope. The environmental impact of this product is the summation of all the material, energy, and landfill space consumed by

the product in the scope. In the following two chapters, we will continue discussing the multi-lifecycle product recovery model in a more rigorous mathematical language.

CHAPTER 4

PART SELECTION OPTIMIZATION

In this section, we will focus on a single node in a product tree and continue the discussion about time-varying cost.

4.1 Costs and Degradation Rate

There are many aspects of a part, which relate to its cost, reliability and recyclability. One of the most important ones is degradation rate λ. It is known that some features of a part will degrade with its service time. The degradation reflects in the residual value of a part and reconditioning cost as well. Normally, the longer the service time, the worse the condition of a part and the higher we have to pay if we want to recondition it such that it can be used in a new product.

Consider the following situation. After a product period, n same electronic parts are sent back to their manufacturer through certain buyback policy. From the standpoint of probability, among these n parts, the expected number of "good" parts among n is nP[good]. Here P[good] is the probability for a part being good, and a function of time. For each part, if it is good, we perform only some simple treatment such as cleaning or repainting. This cost C_{good} is relatively low. If the part fails, we must take some inspection and replace some parts. This cost $C_{failure}$ is very high because the failure inspection of an electronic part is relatively difficult and the repair is more expensive. Thus, the whole expense to recondition the retired parts to new ones is

$$E = C_{good} nP[good] + C_{failure} nP[failure]$$

As we know, the failure probability of many electronic parts obeys the exponential distribution. P[good]=$e^{-\lambda t}$, where λ is the degradation rate of a product and t is the service time.

Thus

$$E = ne^{-\lambda t} C_{good} + n(1 - e^{-\lambda t}) C_{failure}$$

The average cost for each product $C_{recondition}$ is:

$$C_{recondition} = E/n = C_{good} e^{-\lambda t} + C_{failure}(1 - e^{-\lambda t})$$

The curve of an example reconditioning cost is shown in Figure 4.1 when C_{good}=4, $C_{failure}$=70, and λ=0.02888.

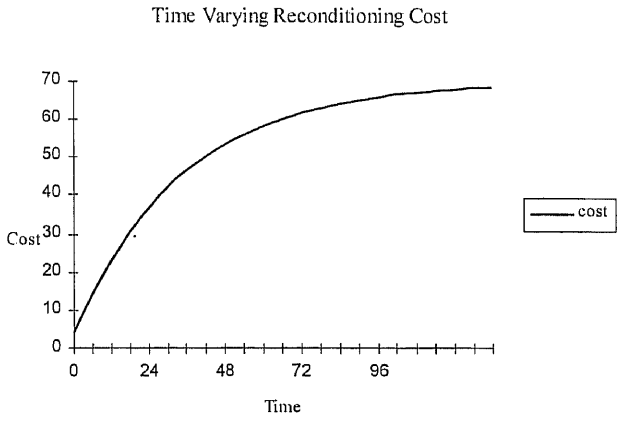


Figure 4.1 Product Reconditioning Cost

It is easy to understand that all the costs are functions of λ and there is always a tradeoff between λ and part costs. The smaller the degradation rate λ, the better the part quality,

and the longer the part lifetime. However, its original new part cost is also higher. In another word, with λ going down, the cost goes up [6]. Figure 4.2(a) shows the relationship between reliability and cost, and Figure 4.2(b) shows the relationship between λ and cost.

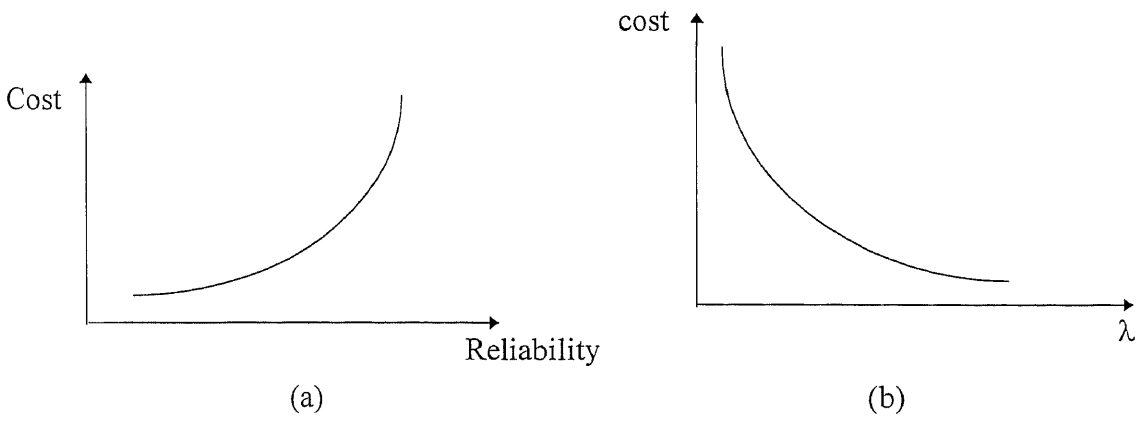


Figure 4.2 **(a)** Relationship between Reliability and Cost [6]
(b) Relationship between λ and Cost

4.2 Part Selection Optimization

Based on the concept of degradation rates of parts, how can we evaluate and select the best part whose environmental impact is the least? In Chapter 3 we discussed that the first thing to evaluate a product system was to set up the consideration scope. In this chapter, we consider the lifetime of a part. It starts from the manufacture of the virgin material and ends at the landfill of the part. We want to find the best λ which carries the least overall cost in this scope.

There are four choices for a retired node (part) for whose disassembly is not applicable:

Choice 1: landfilled and replaced with a new one;

Choice 2: recycled as material and replaced with a new part;

Choice 3: reconditioned to a new part (repaired or reused); and

Choice 4: reused for a second purpose replace with a new one.

The costs related to these choices are:

Cost 1: $C_{replacement} = C_{landfill} + C_{newpart}$, the cost for landfilling the used part and replacing with a new one. The cost of a new part includes material and manufacturing costs;

Cost 2: $C_{recycle} = C_{material\ recycling} - R_{material} + C_{newpart}$, the cost should be the difference between the cost for recycling the used part back to material and the residual value of the material plus the cost of a new part;

Cost 3: $C_{recondition}$, the cost to recondition the old part to an equivalent new part; and

Cost 4: $C_{sp} = C_{newpart} - R_{sp}$, the cost to reuse the retired part for a second purpose and replace with a new part.

For some products, especially computer products, we have to consider another important issue: out-of-date. Even though an IBM 386 personal computer may still function, in normal case, no one would consider reconditioning it any more. Of course, we can still recover it as material. In the reconditioning cost of these products we should add an out-of-date punishment cost, $C_{out\text{-}of\text{-}date}$.

$$C_{recondition} = C_{good} e^{-\lambda t} + C_{failure}(1 - e^{-\lambda t}) + C_{out-of-date}$$

In the following discussion, we will set up an optimum problem to find the best λ for a part.

Recall the concept of part lifetime. If the lifetime of a part is *I*, it has been reconditioned *I*-1 times before it is replaced with a new part (choices 1, 2, and 4). I is actually the total

number of times for which a part is used in a product. The part overall multi-lifecycle cost in its lifetime is

$$\sum_{i=1}^{I-1} C_{recondition}(i, \lambda) + \min\{C_{replacement}(i, \lambda), C_{recycle}(i, \lambda), C_{sp}(i, \lambda)\}$$

where λ is the degradation rate of a part. As we mentioned before, each part has its own λ. For a mechanical part, λ is mainly related to its material and for an electronic part, to its type. Cost which relates to this part is the function of its degradation rate λ. Therefore, if we change the degradation rate, the whole cost changes as well. Our idea is to find an optimum λ^* which gives us the lowest cost per product lifecycle.

This λ^* can be found by solving the following optimization problem.

Denote

$C_1(i, \lambda) = C_{recondition}$ the cost of recovering a used part which has served i product lifetimes into a new product; this cost includes disassembly cost and reconditioning cost;

$C_2'(i, \lambda) = C_{replacement}$ the cost of replacing a used part with a new one; this cost includes new part and landfill cost;

$C_2''(i, \lambda) = C_{recycle}(i, \lambda)$ The cost of material recycling a used part. It is the difference between the material recycle processing cost and the residual value of the recycled material;

$C_2'''(i, \lambda) = C_{sp}(i, \lambda)$ The cost to reuse the retired part for a second purpose;

$C_3(i, \lambda) = C_{newpart}(i, \lambda)$ The cost of a new part;

$C_4(i, \lambda) = C_{landfill}(i, \lambda)$ The cost of disposing and landfilling a used part;

$C_5(i, \lambda)=C_{good}(i, \lambda)$ The cost of reconditioning a used part when it is good;

$C_6(i, \lambda)=C_{fail}(i, \lambda)$ The cost of reconditioning a used part when it fails;

$C_7(i, \lambda)=C_{processing}(i, \lambda)$ The processing cost of recycling a used part into material;

$R_8(i, \lambda)=R_{material}(i, \lambda)$ The residual value of recycled material from a used part

Then we can represent the optimization problem as:

Objective Function:

$$\min\{\frac{\sum_{i=1}^{I-1} C_1(i,\lambda)+C_2(i,\lambda)}{I}\}$$

Subject to:

$$C_2'(i,\lambda) = C_3(i,\lambda) + C_4(i,\lambda) \quad (1)$$

$$C_2''(i,\lambda) = C_3(i,\lambda) + C_7(i,\lambda) - R_8(i,\lambda) \quad (2)$$

$$C_2'''(i,\lambda) = C_3(i,\lambda) - R_{sp}(i,\lambda) \quad (3)$$

$$C_2(i,\lambda) = \min\{C_2'(i,\lambda), C_2''(i,\lambda), C_2'''(i,\lambda)\} \quad (4)$$

$$C_1(i,\lambda) = C_5 e^{-\lambda \times \sum_{j=1}^{i} t_j} + C_6(1 - e^{-\lambda \times \sum_{j=1}^{i} t_j}) \quad (5)$$

$$C_1(I,\lambda) \geq C_2(i,\lambda) \quad (6)$$

where t_j is the j^{th} product lifetime, and I is the part lifetime, i.e. the number of lifecycles of the same type of products. After I product cycles, the recovery cost is equal to or greater than the replacement cost.

This problem to obtain optimal λ leads to no analytical solution in general because of the complexity of all the functions of time. Actually, some costs, such as the material recycle processing cost and recycled material residual value, do not change significantly and can be considered as constants. Sometimes t_j is also a constant. For example, some

manufacturers have certain policy to buy back retiring products which were sold a couple of years ago. For these products, all the t_j's are equal to t_p which is the product buy-back time.

By considering the above facts, Equation (4) is reduced to

$$C_1 = C_5 e^{-\lambda \times i \times t_p} + C_6(1 - e^{-\lambda \times i \times t_p}) = C_6 - (C_6 - C_5)e^{-\lambda \times i \times t_p} \qquad (7)$$

Combining (5) and (6), we obtain

$$I = \text{Ceiling}\left(\frac{\ln\left(\frac{C_2 - C_6}{C_5 - C_6}\right)}{-\lambda t_p}\right) \qquad (8)$$

where Ceiling(x) is the smallest integer u such that $u \geq x$.

Therefore,

$$\Delta = \frac{\sum_{i=1}^{I-1} C_1(i,\lambda) + C_2(i,\lambda)}{I}$$

$$= \frac{C_2}{I} + \frac{I-1}{I} C_6 + \frac{e^{-\lambda t_p} - \frac{C_2 - C_6}{C_5 - C_6}}{1 - e^{-\lambda t_p}} \times \frac{C_5 - C_6}{I} \qquad (9)$$

When material changes, the cost changes as well. $C_2(\lambda)$, $C_5(\lambda)$, and $C_6(\lambda)$ are specified in each product. Now we can see that all the items in Δ in Equation (8) are functions of λ. Thus we can find the best λ by solving this optimization problem.

In most cases, we need to select the best one among a limited number of choices. This is much simpler than finding a theoretic solution if the number of choices is small. Using this model, we simply compute all the Δ's for all λ's and choose the λ which carries the smallest Δ. Sometimes we know all the cost data and we can use them directly.

Sometimes we do not know all of them but we know the function type, we can also figure out all the data and obtain the result.

4.3 Example

A computer monitor is a typical consumer electronic product. Since it consists of many incompatible materials and it is difficult to disassemble, we do not consider material recycling or re-manufacturing it. Thus, there are two choices for a retired monitor. One is to recondition, and the other is to landfill.

Suppose the costs and failure rate of a monitor are given in Table 4.1. The product lifetime is 24 months, i.e., t_p=24 months.

Table 4.1 Costs and Failure Rate of a Computer Monitor

Item	Value	Meaning
C_3	100	New part cost
C_4	50	Landfill cost
C_5	15	Reconditioning cost if the monitor functions
C_6	200	Reconditioning cost if the monitor malfunctions
C_7	80	Material recycle processing cost
R_8	10	Recycled material residual value
R_{sp}	N/A	
λ	0.01199	Failure rate

Therefore, $C_2=\min\{C_3+C_4, C_3+C_7-R_8\}=\min\{100+50, 100+80-10\}=150$

Thus, the part lifetime

$$I = \text{Ceiling}\left(\frac{\ln\left(\frac{C_3 + C_4 - C_6}{C_5 - C_6}\right)}{-\lambda t_p}\right) = \text{Ceiling}\left(\frac{\ln\left(\frac{100 + 50 - 200}{15 - 200}\right)}{-0.01199 \times 24}\right)$$

$$= \text{Ceiling}(4.547) = 5 \text{ (product lifetime)}$$

and the average cost of the monitor is its overall cost divided by its lifetime, i.e.,

$$\Delta = \frac{\sum_{i=1}^{I-1} C_1(i,\lambda) + C_2(\lambda)}{I}$$

$$= \frac{C_3(\lambda) + C_4}{I} + \frac{I-1}{I} C_6 + \frac{e^{-\lambda t_p} - \frac{C_3 + C_4 - C_6}{C_5 - C_6}}{1 - e^{-\lambda t_p}} \times \frac{C_5 - C_6}{I} = 114.15$$

Table 4.2 shows the comparison of reconditioning cost and replacement cost after each product lifetime.

Table 4.2 Comparison of Reconditioning Cost and Replacement Cost

Product lifetime	1	2	3	4	5
Reconditioning Cost C_1	61.25	96.4	122.3	140.8	155.6
Replacement Cost C_2'	150	150	150	150	150
Material Recycle Cost C_2''	170	170	170	170	170

The costs in Table 4.2 is obtained according to:

$$C_1 = C_5 e^{-\lambda \times i \times t_p} + C_6(1 - e^{-\lambda \times i \times t_p})$$

$$C_2' = C_3 + C_4$$

$$C_2'' = C_3 + C_7 - R_8$$

From Table 4.2 we can see that after first four product lifetimes, the reconditioning cost is lower than the replacement cost. After the fifth product lifetime, the reconditioning cost becomes higher than the replacement cost, which means that the monitor is no longer worthy to recondition.

Discussion 1: The Need to Continue Product Recovery

Since (C_5-C_6) is definitely negative, Equation (5) has no solution if $(C_3+C_4-C_6)$ is positive which means that the replacement cost is higher than the reconditioning cost when a part fails. With new recovery and disassembly methods emerging, the reconditioning cost C_6 can drop. Meanwhile, with the environmental problem becoming more and more serious, the landfill cost C_4 is rising. These show the need to continue the product recovery.

Discussion 2: Design Selection

Suppose that we have two design candidates for the computer monitor. Since the designs are different, they have different costs and failure rate. The data of these two designs is shown in Table 4.3.

Table 4.3 The Data of Two Different Designs

	C_3	C_4	C_5	C_6	C_7	R_8	λ
Design 1	100	50	15	200	80	10	0.01199
Design 2	80	60	15	200	90	5	0.02398

In Design 2, we use some cheap components. The new part cost of Design 2 is 20 lower than that of Design 1, but the failure rate is twice as that of Design 1 and the landfill cost rises from 50 to 60 due to the different materials and part volumes.

We have already calculated the result of Design 1. Its lifetime is 5 and average cost per product lifetime is 114.15.

For Design 2,

$$C_2 = \min\{C_2', C_2''\} = \min\{80+60, 80+90-5\} = 140$$

$$I = \text{Ceiling}\left(\frac{\ln\left(\frac{C_2 - C_6}{C_5 - C_6}\right)}{-\lambda t_p}\right) = \text{Ceiling}\left(\frac{\ln\left(\frac{140-200}{15-200}\right)}{-0.02398 \times 24}\right)$$

$$= \text{Ceiling}(1.956) = 2 \text{ (product lifetime)}$$

and

$$\Delta=(140+96.4)/2=118.2$$

Although the new part cost of Design 2 is cheap than that of Design 1, it has higher average cost. In this case, designers will certainly choose Design 1.

CHAPTER 5

PRODUCT DESIGN OPTIMIZATION

In the last section, we focused on a single part and did not consider part disassembly. The model discussed above is suitable for those simple products which just have a few relatively independent components,. However, most products, especially the electronic products, are very complicated and there are tight relationships between components. The part optimum is not independent of each other. We have to take the whole product into our consideration.

In Chapter 3, we discussed the product design optimization. In this chapter, we employ a personal computer example to fulfill the discussion.

The draft structure of a personal computer is shown in Figure 5.1.

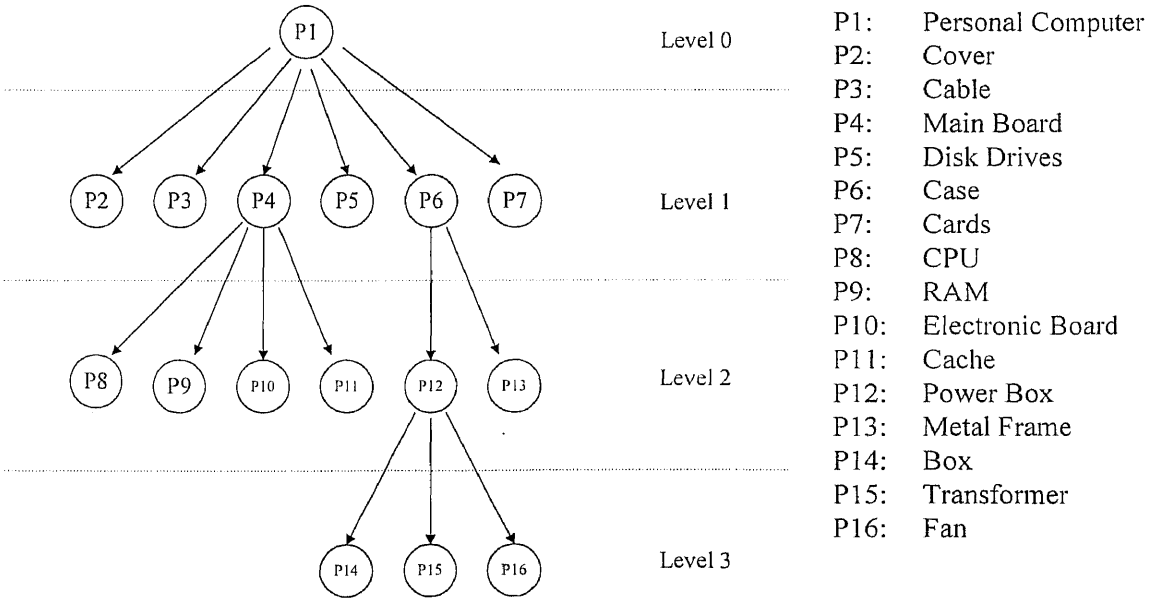


Figure 5.1 The Tree Structure of A Personal Computer

We take abstract level 2 and do not consider the disassembly of the power box. The costs and rates of all the components are assumed in Table 5.1. Since the data for a part used for a second purpose is not available, we do not consider this recovery choice in this example.

Table 5.1 Costs and Rates of Parts in Personal Computer

	Name	C_{new}	$C_{landfill}$	C_{MR}	C_{good}	$C_{failure}$	λ	η
P1	PC	790	50	25	80	1300	0.007	0.0002
P2	Cover	10	4	1	2	30	0	0.0002
P3	Cable	5	2	1	2	50	0.0005	0.0002
P4	M. B.	330	20	10	30	700	0.003	0.0002
P5	D.D.	250	15	5	20	500	0.001	0.0002
P6	Case	50	25	1	5	100	0.001	0.0002
P7	Cards	100	20	6	10	250	0.0015	0.0002
P8	CPU	100	1	2	15	1000	0	0.0002
P9	RAM	120	1	2	10	1000	0.0005	0.0002
P10	E. B.	65	15	8	5	150	0.002	0.0002
P11	Cache	30	1	2	5	100	0.0005	0.0002
P12	Power Box	20	5	2	5	30	0.001	0.0002
P13	Metal Frame	15	2	1	3	25	0	0.0002

Here, C_{MR} equals material recycle processing cost minus recycled material value. η is the out-of-date punishing factor. In this example, we assume that the punishing cost is equal to $\eta \times t^2 \times C_{newpart}$. All the data above are based on the assumption that product lifetime is 24 months. In this example, we also consider the inflation of landfill cost and the rate is assumed 9.2% [13]. The landfill cost consists of tipping, transportation, management,

and treatment fees. Since the accurate data is not available for this present research, the landfill costs in this example are assumed based on the data given in [12]. Considering that PC as a whole contains hazardous material, its landfill cost is significantly higher than that of the municipal waste. Note that the tipping fee of a ton waste in Newark area was $100 in 1984 [12].

Using a LINKER model and disassembly table, we can represent the position structure of the product design and generate a set of disassembly paths. Figure 5.2 shows two of these disassembly paths.

In Figure 5.2, "■" represents the part and "—" represents the assembly.

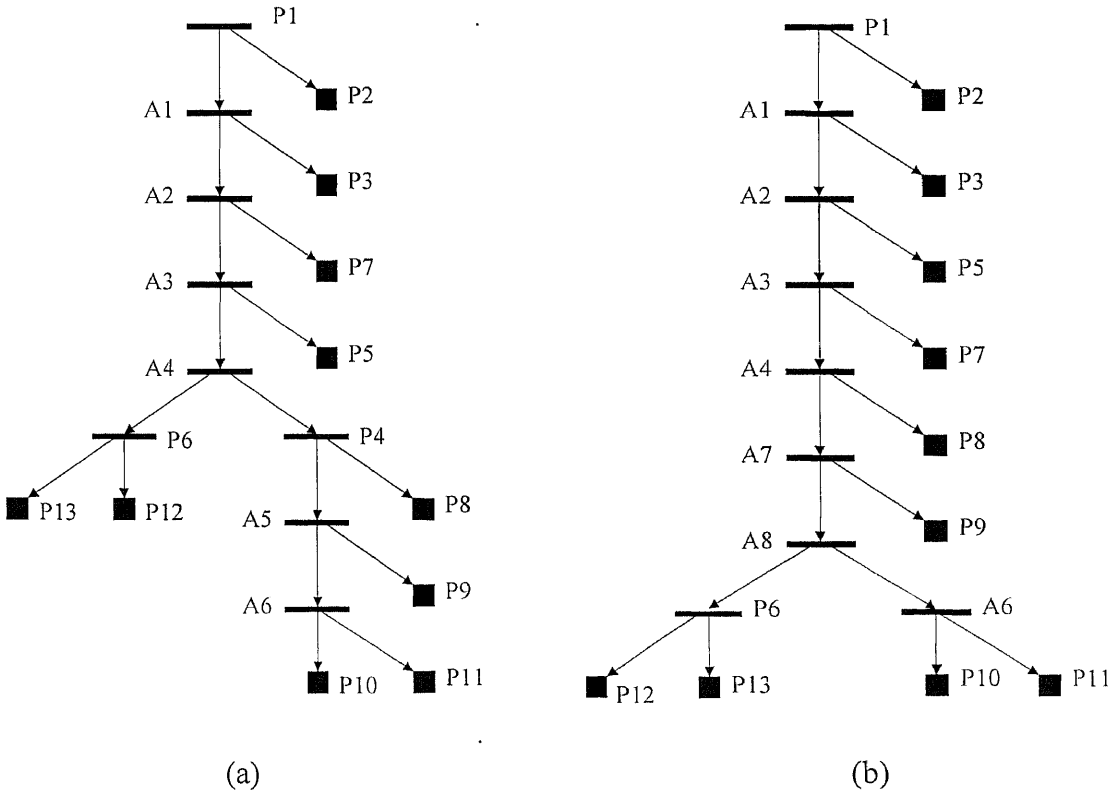


Figure 5.2 Two Possible Disassembly Paths

Although a disassembly path is similar to the Reverse Fishbone diagram, there are two differences between these two:

1. A Reverse Fishbone represents the product retirement plan which is set by the designers and it may not be the best retirement plan for the product. A disassembly path represents a possible disassembly order from the top to the bottom. It does not set the disassembly depth and the retirement intent of each node. The best retirement plan in this path can be obtained by search all the possible situations. We will discuss how to find the best plan later.
2. A disassembly path shows not only the parts but also all the assemblies.

As we mentioned before, there are five recovery choices for each non-leaf node in the disassembly path. In Section 4.2 we discussed four of them and the costs related to the four choices are $C_{replacement}$, $C_{recycle}$, C_{sp}, and $C_{recondition}$. The fifth choice is part re-manufacturing which means to disassemble the subassembly represented by the node and recover the children of that subassembly. Its cost is $C_{remanufacture}$. The best recovery form of this node should be the one carries the least cost. If we denote the recovery cost of this node at level k as $C^k_{recovery}$, then

$$C^k_{recovery} = \min\{C^k_{replacement}, C^k_{recycle}, C^k_{recondition}, C^k_{sp}, C^k_{remanufacture}\} \quad (5.1)$$

Where

$$C^k_{remanfacture} = C^k_{disassembly} + \sum_{i=1}^{N} C^{k+1}_{recovery} \quad (5.2)$$

Equation 5.2 means that the remanufacturing cost of a node at level k is the summation of all the recovery costs of its children at level k+1 plus the disassembly cost at level k. N is the number of children.

For example, Let us look at node A6 in disassembly path 5.2 (a). Suppose the costs of A6, P10 and P11 after the first product lifetime are given in Table 5.2.

Table 5.2 Data for A6, P10 and P11 After the First Product Lifetime

	$C_{replacement}$	$C_{recycle}$	$C_{recondition}$
A6	117.8	108	50.1
P10	32	31.2	9.6
P11	90.2	73	19.3

The least cost of P10 and P11 are their reconditioning cost 9.6 and 19.3, respectively. If the disassemlby cost is 5, the remanufacture cost of A6 is 5+19.3+9.6=33.9. It is less than any of other three costs. Therefore, the recovery cost of A6 is 33.9 and the recovery intent of A6 after the first product lifetime is part re-manufacturing.

Table 5.3 shows the data of the three nodes after the third product lifetime. Since time is longer, the reconditioning costs are higher as shown in Table 5.3.

Table 5.3 Data for A6, P10 and P11 after the Third Product Lifetime

	$C_{replacement}$	$C_{recycle}$	$C_{recondition}$
A6	125.2	108	195.5
P10	31.6	32	39.6
P11	90.2	73	92.3

Now the recovery costs of P10 and P11 are 31.6 and 73, respectively based on the calculation results in the 2nd and 3rd rows in Table 5.3. Thus $C_{remanufacture}$ of A6 is 5+31.6+73=109.6. We can see that the least cost of A6 changes to material recycle cost, i.e. 108. Thus, the best recovery intent of A6 is material recycle and the recovery cost is 108. This means that A6 is no longer worthy to be disassembled and the product disassembly process stops at A6.

If we keep doing the calculation on all the nodes in the disassembly path, we can find the best recovery intent for each node and the stop node set for this disassembly path.

In order to calculate the result, we need not only the data for each part node but also the data for each assembly node. Table 5.1 lists the data for all parts and Table 5.4 lists the data for all the assembly nodes.

Table 5.4 Costs and Rates of Assembly Nodes in Personal Computer

Name	C_{new}	$C_{landfill}$	C_{MR}	C_{good}	$C_{failure}$	λ	η	Conection
A1	775	95	48	77	1290	0.007	0.0002	Type 1
A2	765	85	42	75	1280	0.0065	0.0002	Type 2
A3	655	65	30	60	1200	0.005	0.0002	Type 3
A4	390	60	30	40	800	0.004	0.0002	Type 2
A5	205	35	18	20	500	0.0025	0.0002	Type1
A6	100	32	17	10	500	0.0025	0.0002	Type1

According to the data given as Tables 5.1 and 5.4, we obtain the best retirement plan for the disassembly path shown in Figure 5.2 (a). Table 5.5 shows the result after the first

product lifetime, which is 24 months in this example. Tables 5.6 and 5.7 show the results after second and third product lifetimes.

As we mentioned before, from one product design we could generate a set of disassembly paths. Using the above method, we can obtain the best retirement plan and the overall cost for each of them. The optimal disassembly path for this product design is the one whose overall cost is the least. Therefore, we can find the optimal retirement plan and overall environmental impact for each of the product candidates and finally find the best product design. For example, suppose that a PC only have two possible disassembly paths, Path A and B, as shown in Figure 5.2 (a) and (b) respectively. We have already known that the least overall cost for Path A for the first three product lifetime is 249.9+512+782.4=1544.3. We can do the same calculation for Path B. Suppose that its overall cost in the same time period is 1600. This means Path A is better than Path B and the best retirement plan is following Path A and going to the stop node set. The overall environmental impact of this product design in this time period is 1544.3.

Using this method, we can obtain all the overall environmental impacts of the design candidates and therefore, we can find the best design.

Table 5.5 Retirement Plan of Personal Computer after the First Product Lifetime

Part and Assembly	Optimal Retirement Intent	Optimal Retirement Cost
1:PC	Disassembly	249.9
2:Assembly 1	Disassembly	241.7
3:Cover	Recondition	3.2
4:Assembly 2	Disassembly	233.8
5:Cable	Recondition	2.9
6:Assembly 3	Disassembly	193.8
7:Cards	Recondition	30
8:Assembly 4	Disassembly	118.6
9:Disk Drives	Recondition	60.2
10:Case	Recondition	13.0
11:Main Board	Recondition	95.6
12:Power Box	Recondition	7.9
13:Metal Frame	Recondition	3.9
14:Assembly 5	Recondition	71.6
15:RAM	Recondition	26.5
16:Assembly 6	Disassembly	33.9
17:CPU	Recondition	35.6
18:Electronic Board	Recondition	19.3
19:Cache	Recondition	9.6

Table 5.6 Retirement Plan of Personal Computer after the Second Product Lifetime

Part and Assembly	Optimal Retirement Intent	Optimal Retirement Cost
1:PC	Disassembly	512
2:Assembly 1	Disassembly	500.4
3:Cover	Recondition	6.6
4:Assembly 2	Disassembly	491.1
5:Cable	Recondition	4.3
6:Assembly 3	Disassembly	407.9
7:Cards	Recondition	73.1
8:Assembly 4	Disassembly	234.8
9:Disk Drives	Recondition	158.2
10:Case	Disassembly	27.0
11:Main Board	Recondition	197.8
12:Power Box	Recondition	15.5
13:Metal Frame	Recondition	5.5
14:Assembly 5	Disassembly	168.4
15:RAM	Recondition	61.1
16:Assembly 6	Disassembly	74.5
17:CPU	Recondition	88.9
18:Electronic Board	Recondition	48.4
19:Cache	Recondition	21.1

Table 5.7 Retirement Plan of Personal Computer after the Third Product Lifetime

Part and Assembly	Optimal Retirement Intent	Optimal Retirement Cost
1:PC	Disassembly	782.4
2:Assembly 1	Disassembly	766.4
3:Cover	Material Recycling	11
4:Assembly 2	Disassembly	755.4.7
5:Cable	Material Recycling	6.0
6:Assembly 3	Disassembly	639.4
7:Cards	Material Recycling	106
8:Assembly 4	Disassembly	369.4
9:Disk Drives	Material Recycling	255
10:Case	Disassembly	37.8
11:Main Board	Recondition	321.6
12:Power Box	Material Recycle	22
13:Metal Frame	Recondition	10.8
14:Assembly 5	Material Recycling	215
15:RAM	Landfill	101.6
16:Assembly 6	Material Recycling	108
17:CPU	Landfill	121.6
18:Electronic Board	Material Recycling	73
19:Cache	Landfill	31.6

CHAPTER 6

COMPUTER AIDED TOOL IMPLEMENTATION

6.1 Ideal Computer Aided DFE Tool

We should not assume that all the designers are DFE experts. An ideal computer aided DFE tool can support designers to implement DFE transparently. The data input and product layout representation should be similar to the standard CAD tools which are commonly used by designers.

There are three key issues in the development of a DFE tool. The first one is the import of design information. The information needed to evaluate a product layout, in a general case, includes

- the structure of a product,
- the material or type of each part in the product,
- each connection method, and
- manufacturing processes.

For simple products which just have a few components, designers can input all the information by drawing the structures and filling out each part information form. However, for a complicated product, this process becomes impractical. One solution is to develop an import system which can read all the data from the product design layout database of some standard CAD tool. Another solution is to integrate the DFE module into the standard CAD tool which lets designers implement DFE directly in their design process.

The second issue is the environmental impact assessment tool which supports transferring the standard product information and its environment impact into cost domain so that designers can evaluate different aspects of their design using indexes in one single domain. There are two main parts in this tool, which are the environmental impact assessment method and the environmental impact database. This database is also a knowledge base which is based for one to develop design improvement suggestion systems.

The third issue is the optimal design search method. As we mentioned before, there are three steps in optimal design search. The first step is to generate design candidates. Currently this task relies on designers themselves. The second step is to generate disassembly paths. From each product design layout generated in the first step, a DFE tool should automatically generate all the possible disassembly paths. The third step is to generate and find the optimal retirement plan for each disassembly path.

6.2 Tool Implementation

Currently, a computer aided tool is under development to demonstrate our model and method. It is developed on PC Windows 95 platform using Object-Oriented method and Microsoft Visual C++ 4.0 IDE. The coding follows the Microsoft Application Frame and is supported by Microsoft Foundation Class Library (MFC) 4.0.

Among the three steps we just discussed, we have implemented the third one. Currently, an automatic disassembly path generation algorithm is still under development. Our automatic optimum search starts from a disassembly path and designers have to input the

product information by filling out all the part forms one by one according to their position order. Figure 6.1 shows one of such input forms.

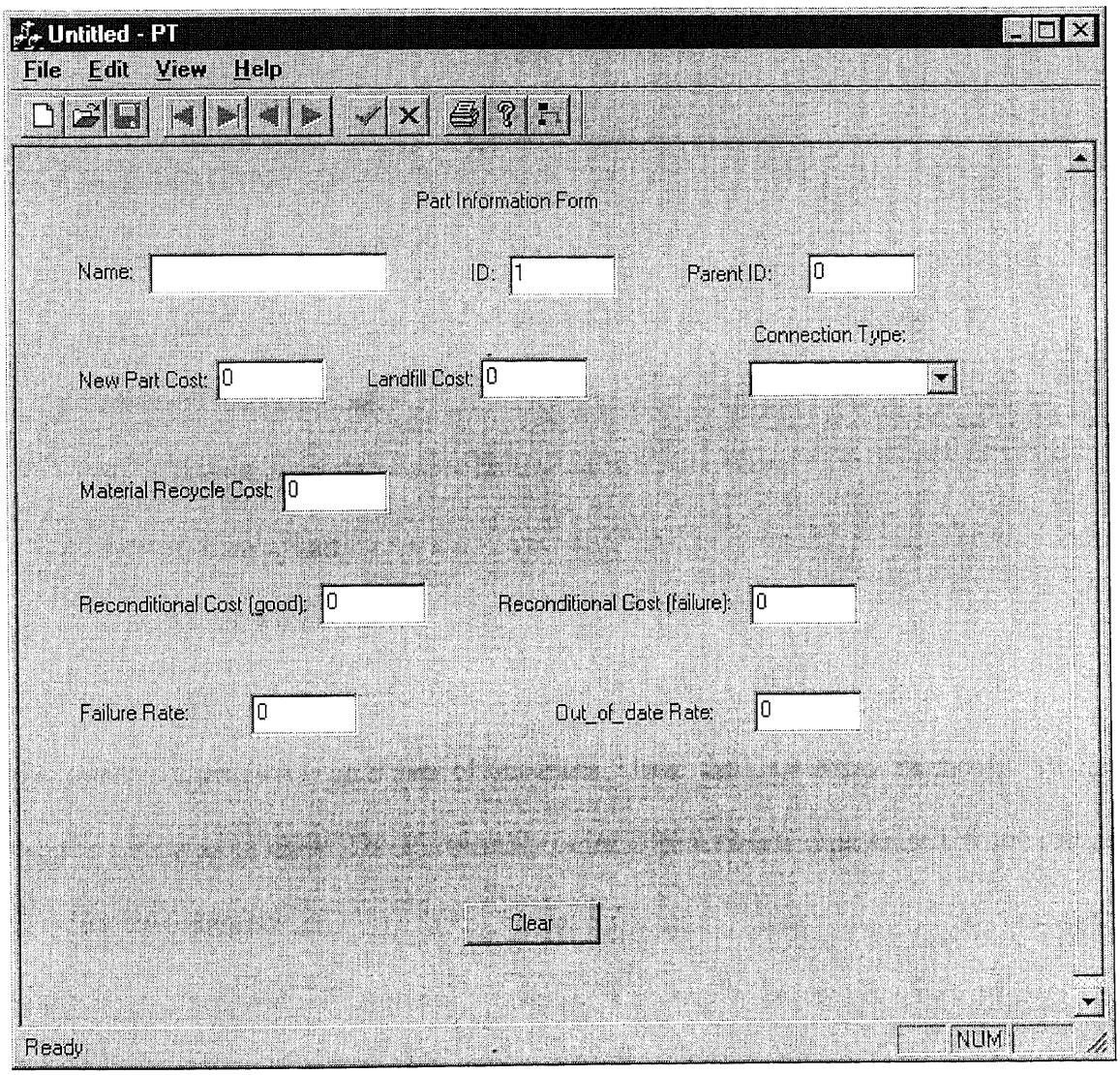


Figure 6.1 Part Information Input Form

To represent a disassembly path, first we define a part class. The data members of the class include:

- Name: the part name;
- Part ID.: each part is numbered according to its position order;
- Parent ID.: If a part could be disassembled into several subparts, we call the subparts as children of the part. The part is the parent of these subparts;
- New Part Cost;
- Landfill Cost;
- Connection Type: According to the disassembly difficulty, we set four types of connections. This data member is belong to non-leaf (parent) node;
- Material Recycle Cost;
- Reconditioning Cost;
- Failure Rate λ; and
- Out-of-date Rate.

All the information above can be obtained from the input form as shown in Figure 6.1.

Each part in a product is an object of this class. Using the C++ Array Template, we can organize them as an array. The disassembly order information is represented by the parent part No. data member.

Once we have all the information, we can follow the following algorithm to find the optimal retirement plan on this disassembly path.

Step 1: Calculate the $C_{replacement}$, $C_{material\ recycle}$, $C_{recondition}$ for each node based on Equations (1), (2), and (4). Set the recovery cost of each node as the least of the three costs of that node. Move the node pointer to the end of Array.

Step 2: If this node is a leaf node, go to Step 3. Otherwise, add disassembly cost to part re-manufacturing cost. If remanufacturing cost is less than recovery cost, set recovery cost as re-manufacturing cost.

Step 3: Add recovery cost of this node to the part remanufacturing cost of its parent and move up one node.

Step 4: If not finished, go to Step 2. Otherwise go to Step 5.

Step 5: Move down from the top node and set the node as a stop node if the its recovery cost is not equal to its re-manufacturing cost.

Step 6: finished.

The algorithm flow chart is shown in Figure 6.2.

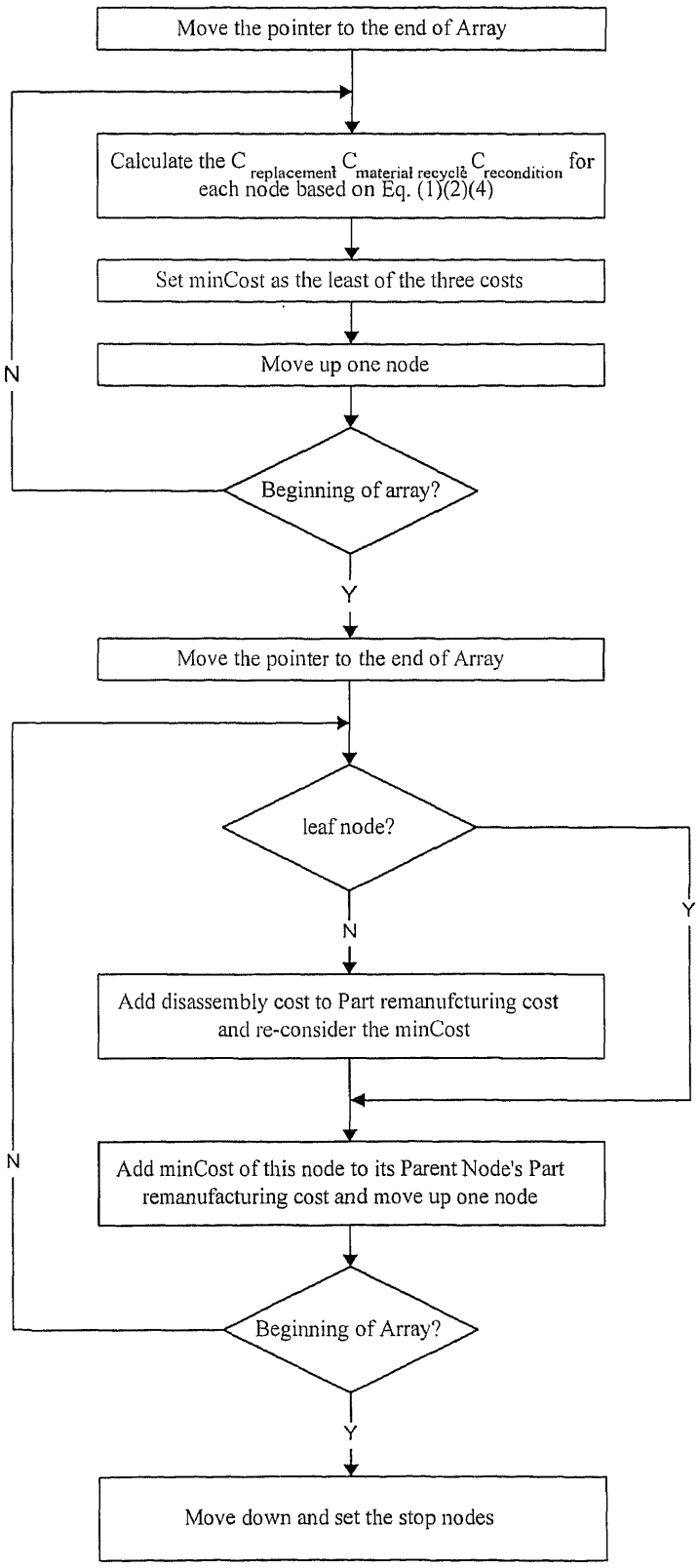


Figure 6.2 Optimal Retirement Plan Search Algorithm Flow Chart

CHAPTER 7

CONCLUSION

7.1 Contributions of this Thesis

Product design is the most important stages in reducing product environmental impacts and conserving precious environmental resource. The design of a product determines its recovery system. This thesis discusses the concept Design For Environment (DFE) and the method of evaluating a product recovery system. The ideas and results of this thesis are helpful for designers to improve their product design with environmental issues considered.

The contributions of this thesis are summarized as follows:

1. It introduced the concept of product design based on candidate sets and indirect design improvement.
2. It proposed the method of disassembly path which can capture the product disassembly features and automatically find the best product retirement plan based on each disassembly path.
3. It introduced the concept of time varying cost and builds a multi-lifecycle product recovery model based on this concept.
4. It defined the Multi-lifecycle part lifetime. The part lifetime begins at the manufacture of the part and ends at the time when the part is material-recycled or landfilled. Among the four recovery choices, which choice is the best for a retired part depends on the comparison of the costs related to the four choices.

5. It applied the proposed method to a monitor and PC to decide their part life times and costs given their new part, landfill, reconditioning, and residual material costs and the part degrading and out-of-date rates.
6. It discussed the components of a Computer Aided DFE Tool.
7. A PC Windows 95 based Software tool is partially developed to demonstrate our method and model.

7.2 Limitations and Future Research

The improvement of product design could be done in many levels. The optimization can be performed at the levels of part, assembly, product, and product set. In the low level optimization, i.e., the part selection, the problem is clear and we can well define a mathematical programming model to find a solution. In the higher level, i.e. product structure, the activity of improvement becomes more complicated. This thesis has not discussed how to model product structure improvement. Research is needed to deal with a complex electronic product to explore the applicability of our model and method. We also hope to improve our model and method by giving more consideration at the higher levels. Currently, a method which can reasonably translate product environmental impact into cost is still not available. Based on the research results on the environment related disciplines, such a method to support our model needs to be developed.

Some ideas proposed in this thesis, such as product design based on candidate set and DFE design suggestion system, are not well developed. We will continue the research on these ideas.

Much work should be directed to development of a complete computer aided DFE tool with a more friendly interface to a user. A product design representation import system and automatic disassembly path generation algorithm are needed. Embedding our DFE tool as a module into standard CAD tools should be performed.

APPENDIX

SOFTWARE SOURCE CODE

```
////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                                //
// File Name:   InputDialog.h                                                                     //
//                                                                                                //
// Description: The file is the defination of the class InputDialog                               //
//              The dialog is used to input the product lifetime                                  //
//                                                                                                //
// Update date: 10/19/1996                                                                        //
//                                                                                                //
////////////////////////////////////////////////////////////////////////////////////////////////////


/////////////////////////////////////////////////////////////////////////////
// CInputDialog dialog

class CInputDialog : public CDialog
{
// Construction
public:
        CInputDialog(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
        //{{AFX_DATA(CInputDialog)
        enum { IDD = IDD_TIMEINPUT };
        UINT   m_nTime;
        //}}AFX_DATA


// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CInputDialog)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:

        // Generated message map functions
        //{{AFX_MSG(CInputDialog)
```

```
        // NOTE: the ClassWizard will add member functions here
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                                    //
// File Name:   InputDialog.cpp                                                                       //
//                                                                                                    //
// Description: The file is the implementation of the class InputDialog                               //
//              The dialog is used to input the product lifetime                                      //
//              The default value is 24 (months)                                                      //
//                                                                                                    //
// Update date: 10/19/1996                                                                            //
//                                                                                                    //
////////////////////////////////////////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "PT.h"
#include "InputDialog.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CInputDialog dialog


CInputDialog::CInputDialog(CWnd* pParent /*=NULL*/)
    : CDialog(CInputDialog::IDD, pParent)
{
    //{{AFX_DATA_INIT(CInputDialog)
    m_nTime = 24;
    //}}AFX_DATA_INIT
}


void CInputDialog::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CInputDialog)
    DDX_Text(pDX, IDC_PTIME, m_nTime);
```

```
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CInputDialog, CDialog)
	//{{AFX_MSG_MAP(CInputDialog)
		// NOTE: the ClassWizard will add message map macros here
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CInputDialog message handlers
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                                //
// File Name:   MainFrm.h                                                                         //
//                                                                                                //
// Description: The file is the interface of the CMainFrame class                                 //
//              It is the main window of the Application                                          //
//                                                                                                //
// Update date: 10/19/1996                                                                        //
//                                                                                                //
////////////////////////////////////////////////////////////////////////////////////////////////////


/////////////////////////////////////////////////////////////////////////////

class CMainFrame : public CFrameWnd
{
protected: // create from serialization only
        CMainFrame();
        DECLARE_DYNCREATE(CMainFrame)

// Attributes
public:

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CMainFrame)
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        virtual void ActivateFrame(int nCmdShow);
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CMainFrame();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:  // control bar embedded members
        CStatusBar  m_wndStatusBar;
        CToolBar    m_wndToolBar;
```

```
// Generated message map functions
protected:
        //{{AFX_MSG(CMainFrame)
        afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////


/////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                                               //
// File Name:   MainFrm.cpp                                                                                      //
//                                                                                                               //
// Description: The file is the implementation of the CMainFrame class                                           //
//              It is the main window of the application                                                         //
//                                                                                                               //
// Update date: 10/19/1996                                                                                       //
//                                                                                                               //
/////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//:
//

#include "stdafx.h"
#include "PT.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
        //{{AFX_MSG_MAP(CMainFrame)
        ON_WM_CREATE()
        //}}AFX_MSG_MAP
```

```
END_MESSAGE_MAP()

static UINT indicators[] =
{
        ID_SEPARATOR,           // status line indicator
        ID_INDICATOR_CAPS,
        ID_INDICATOR_NUM,
        ID_INDICATOR_SCRL,
};

/////////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
        // TODO: add member initialization code here

}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
        if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
                return -1;

        if (!m_wndToolBar.Create(this) ||
                !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
        {
                TRACE0("Failed to create toolbar\n");
                return -1;      // fail to create
        }

        if (!m_wndStatusBar.Create(this) ||
                !m_wndStatusBar.SetIndicators(indicators,
                  sizeof(indicators)/sizeof(UINT)))
        {
                TRACE0("Failed to create status bar\n");
                return -1;      // fail to create
        }

        // TODO: Remove this if you don't want tool tips or a resizeable toolbar
        m_wndToolBar.SetBarStyle(m_wndToolBar.GetBarStyle() |
```

```
                CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC);

        // TODO: Delete these three lines if you don't want the toolbar to
        //  be dockable
        m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
        EnableDocking(CBRS_ALIGN_ANY);
        DockControlBar(&m_wndToolBar);

        return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
        // TODO: Modify the Window class or styles here by modifying
        //  the CREATESTRUCT cs

        return CFrameWnd::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
        CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
        CFrameWnd::Dump(dc);
}

#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers

void CMainFrame::ActivateFrame(int nCmdShow)
{
        CFrameWnd::ActivateFrame(nCmdShow);
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                                //
// File Name:   Part.h                                                                            //
//                                                                                                //
// Description: The file is the interface of the CPart class                                      //
//              It represents the general part in the product                                     //
//                                                                                                //
// Update date: 10/19/1996                                                                        //
//                                                                                                //
////////////////////////////////////////////////////////////////////////////////////////////////////


class CPart:public CObject
{
        DECLARE_SERIAL(CPart)
public:
        CString m_strName;
        CString m_strConnectType;

        UINT m_nID;
        UINT m_nParentID;

        int m_nChildNumber;                     //child number is limited in 5;

        UINT m_nChildID[5];

        int m_nCostRG;
        int m_nCostRF;
        int m_nCostLandfill;
        int m_nCostNew;
        int m_nCostMR;

        double m_fRateDate;
        double m_fRateFailure;
        double m_fCostR;
        double m_fCostM;
        double m_fCostC;
        double m_fCostP;
        double minCost;

        BOOL IsLeaf;
        BOOL IsStopNode;
```

```
CPart(){
        m_nID=1;
        m_nParentID=0;
        m_nChildNumber=0;

        for(int j=0;j<5;j++){
                m_nChildID[j]=0;
        }

        m_nCostRG=0;
        m_nCostRF=0;
        m_nCostLandfill=0;
        m_nCostNew=0;
        m_nCostMR=0;
        m_fRateDate=0.0;
        m_fRateFailure=0.0;
        m_fCostR=0.0;
        m_fCostM=0.0;
        m_fCostC=0.0;
        m_fCostP=0.0;
}

CPart(const char* szName, UINT nID, UINT nParentID, long nCostNew, long
nCostMR, long nCostLandfill
        , long nCostRG, long nCostRF, double fRateDate, double fRateFailure,
const char* szConnectType):m_strName(szName)
{
        m_strConnectType=szConnectType;
        m_nID=nID;
        m_nParentID=nParentID;
        m_nCostNew=nCostNew;
        m_nCostRG=nCostRG;
        m_nCostRF=nCostRF;
        m_nCostLandfill=nCostLandfill;
        m_nCostMR=nCostMR;
        m_fRateDate=fRateDate;
        m_fRateFailure=fRateFailure;
}

CPart(const CPart& s):m_strName(s.m_strName){
        m_strConnectType=s.m_strConnectType;
        m_nID=s.m_nID;
        m_nParentID=s.m_nParentID;
        m_nCostNew=s.m_nCostNew;
        m_nCostMR=s.m_nCostMR;
```

```
        m_nCostLandfill=s.m_nCostLandfill;
        m_nCostRG=s.m_nCostRG;
        m_nCostRF=s.m_nCostRF;
        m_fRateDate=s.m_fRateDate;
        m_fRateFailure=s.m_fRateFailure;
}

const CPart& operator=(const CPart& s){
        m_nID=s.m_nID;
        m_nParentID=s.m_nParentID;
        m_strConnectType=s.m_strConnectType;
        m_strName=s.m_strName;
        m_nCostNew=s.m_nCostNew;
        m_nCostMR=s.m_nCostMR;
        m_nCostLandfill=s.m_nCostLandfill;
        m_nCostRG=s.m_nCostRG;
        m_nCostRF=s.m_nCostRF;
        m_fRateDate=s.m_fRateDate;
        m_fRateFailure=s.m_fRateFailure;
        return *this;
}

BOOL operator ==(const CPart& s) const{
        if ((m_strName==s.m_strName)&&
                (m_nID==s.m_nID)&&
                (m_nParentID==s.m_nParentID)&&
                (m_strConnectType==s.m_strConnectType)&&
                (m_nCostNew==s.m_nCostNew)&&
                (m_nCostMR==s.m_nCostMR)&&
                (m_nCostLandfill==s.m_nCostLandfill)&&
                (m_nCostRG==s.m_nCostRG)&&
                (m_nCostRF==s.m_nCostRF)&&
                (m_fRateDate==s.m_fRateDate)&&
                (m_fRateFailure==s.m_fRateFailure)){
                return TRUE;
        }
        else{
                return FALSE;
        }
}

BOOL operator !=(const CPart& s) const{
        return !(*this==s);

}
```

```
        void CPart::Serialize(CArchive& ar)
        {
                if (ar.IsStoring()){
                        ar << m_strName << (long)m_nCostNew<<
(long)m_nCostLandfill
                                << (long)m_nCostMR << (long)m_nCostRG <<(
long)m_nCostRF
                                << m_fRateFailure<<
m_fRateDate<<m_strConnectType<<m_nID
                                <<m_nParentID;
                }
                else {
                        ar >> m_strName>> (long&)m_nCostNew>>
(long&)m_nCostLandfill
                                >> (long&)m_nCostMR>> (long&)m_nCostRG >>
(long&)m_nCostRF
                                >> m_fRateFailure >>
m_fRateDate>>m_strConnectType>>m_nID
                                >>m_nParentID;
                }
        }

#ifdef _DEBUG
        void Dump(CDumpContext& dc) const;
#endif

};


typedef CTypedPtrArray<CObArray, CPart*> CPartArray;




////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                                        //
// File Name:   Part.cpp                                                                                  //
//                                                                                                        //
// Description: The file is the implementation of the class part.                                         //
//              The method to operate the data members can be put in this file                            //
//                                                                                                        //
// Update date: 10/19/1996                                                                                //
//                                                                                                        //
////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
#include "StdAfx.h"
#include "part.h"
IMPLEMENT_SERIAL(CPart, CObject,0)

#ifdef _DEBUG
void CPart::Dump(CDumpContext& dc) const
{
        CObject::Dump(dc);
        dc<<"\nm_strName="<<m_strName;
}
#endif
```

```
////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                            //
// File Name:   PT.h                                                                          //
//                                                                                            //
// Description: The file is the defination of the class PT.                                   //
//              It defines the class attributions for the application.                        //
//                                                                                            //
// Update date: 10/19/1996                                                                    //
//                                                                                            //
////////////////////////////////////////////////////////////////////////////////////////////////



#ifndef __AFXWIN_H__
        #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"       // main symbols

/////////////////////////////////////////////////////////////////////////////
// CPTApp:
// See PT.cpp for the implementation of this class
//

class CPTApp : public CWinApp
{
public:
        CPTApp();

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CPTApp)
        public:
        virtual BOOL InitInstance();
        //}}AFX_VIRTUAL

// Implementation

        //{{AFX_MSG(CPTApp)
        afx_msg void OnAppAbout();
                // NOTE - the ClassWizard will add and remove member functions here.
                //    DO NOT EDIT what you see in these blocks of generated code !
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
```

```
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                              //
// File Name:   PT.cpp                                                                          //
//                                                                                              //
// Description: The file is the implementation of the class PT.                                 //
//          It defines the class behaviors for the application.                                 //
//                                                                                              //
// Update date: 10/19/1996                                                                      //
//                                                                                              //
////////////////////////////////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "PT.h"

#include "MainFrm.h"
#include "part.h"
#include "PTDoc.h"
#include "PTView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CPTApp

BEGIN_MESSAGE_MAP(CPTApp, CWinApp)
        //{{AFX_MSG_MAP(CPTApp)
        ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
                // NOTE - the ClassWizard will add and remove mapping macros here.
                //    DO NOT EDIT what you see in these blocks of generated code!
        //}}AFX_MSG_MAP
        // Standard file based document commands
        ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
        ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
```

```
        // Standard print setup command
        ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CPTApp construction

CPTApp::CPTApp()
{
        // TODO: add construction code here,
        // Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CPTApp object

CPTApp theApp;

/////////////////////////////////////////////////////////////////////////////
// CPTApp initialization

BOOL CPTApp::InitInstance()
{
        // Standard initialization
        // If you are not using these features and wish to reduce the size
        //  of your final executable, you should remove from the following
        //  the specific initialization routines you do not need.

#ifdef _AFXDLL
        Enable3dControls();                     // Call this when using MFC in a shared
DLL
#else
        Enable3dControlsStatic();       // Call this when linking to MFC statically
#endif
        SetRegistryKey("Inside Visual C++");
        LoadStdProfileSettings();  // Load standard INI file options (including MRU)

        // Register the application's document templates.  Document templates
        //  serve as the connection between documents, frame windows and views.

        CSingleDocTemplate* pDocTemplate;
        pDocTemplate = new CSingleDocTemplate(
                IDR_MAINFRAME,
                RUNTIME_CLASS(CPTDoc),
                RUNTIME_CLASS(CMainFrame),       // main SDI frame window
```

```
        RUNTIME_CLASS(CPTView));
    AddDocTemplate(pDocTemplate);

    // Parse command line for standard shell commands, DDE, file open
    CCommandLineInfo cmdInfo;
    ParseCommandLine(cmdInfo);

    // Dispatch commands specified on the command line
    if (!ProcessShellCommand(cmdInfo))
        return FALSE;

    return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

// Dialog Data
    //{{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
    //}}AFX_DATA

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CAboutDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
    //{{AFX_MSG(CAboutDlg)
        // No message handlers
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
    //{{AFX_DATA_INIT(CAboutDlg)
    //}}AFX_DATA_INIT
```

```
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		// No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CPTApp::OnAppAbout()
{
	CAboutDlg aboutDlg;
	aboutDlg.DoModal();
}

/////////////////////////////////////////////////////////////////////////////
// CPTApp commands
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                                //
// File Name:   PTDoc.h                                                                           //
//                                                                                                //
// Description: The file is the interface of the CPTDoc class                                     //
//              It define the storage behavior of the application                                 //
//                                                                                                //
// Update date: 10/19/1996                                                                        //
//                                                                                                //
////////////////////////////////////////////////////////////////////////////////////////////////////


/////////////////////////////////////////////////////////////////////////////

class CPTDoc : public CDocument
{
private:
        CPartArray m_partArray;
protected: // create from serialization only
        CPTDoc();
        DECLARE_DYNCREATE(CPTDoc)

// Attributes
public:

        CPartArray* GetArray() {
                return &m_partArray;
        }
// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CPTDoc)
        public:
        virtual BOOL OnNewDocument();
        virtual void Serialize(CArchive& ar);
        virtual void DeleteContents();
        virtual BOOL OnOpenDocument(LPCTSTR lpszPathName);
        virtual BOOL OnSaveDocument(LPCTSTR lpszPathName);
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CPTDoc();
```

```
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
        //{{AFX_MSG(CPTDoc)
        afx_msg void OnEditClearAll();
        afx_msg void OnUpdateEditClearAll(CCmdUI* pCmdUI);
        afx_msg void OnUpdateFileSave(CCmdUI* pCmdUI);
        afx_msg void OnReport();
        //afx_msg void OnUpdateReport(CCmdUI* pCmdUI);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////



/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                                                           //
// File Name:   PTDoc.cpp                                                                                                    //
//                                                                                                                           //
// Description: The file is the implementation of the CPTDoc class                                                           //
//              It define the storage behavior of the application                                                            //
//                                                                                                                           //
// Update date: 10/19/1996                                                                                                   //
//                                                                                                                           //
/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////


#include "stdafx.h"
#include "PT.h"

#include "part.h"
#include "ReportDialog.h"
#include "InputDialog.h"
#include "PTDoc.h"
#include "math.h"

#ifdef _DEBUG
```

```
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CPTDoc

IMPLEMENT_DYNCREATE(CPTDoc, CDocument)

BEGIN_MESSAGE_MAP(CPTDoc, CDocument)
        //{{AFX_MSG_MAP(CPTDoc)
        ON_COMMAND(ID_EDIT_CLEAR_ALL, OnEditClearAll)
        ON_UPDATE_COMMAND_UI(ID_EDIT_CLEAR_ALL,
OnUpdateEditClearAll)
        ON_UPDATE_COMMAND_UI(ID_FILE_SAVE, OnUpdateFileSave)
        ON_COMMAND(ID_REPORT, OnReport)
        //ON_UPDATE_COMMAND_UI(ID_REPORT, OnUpdateReport)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CPTDoc construction/destruction

CPTDoc::CPTDoc()
{
        // TODO: add one-time construction code here
#ifdef _DEBUG
        afxDump.SetDepth(1);
#endif

}

CPTDoc::~CPTDoc()
{
}

BOOL CPTDoc::OnNewDocument()
{
        TRACE("Entering CPartDoc::OnNewDocument\n");
        if (!CDocument::OnNewDocument())
                return FALSE;

        // TODO: add reinitialization code here
        // (SDI documents will reuse this document)
```

```
        return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CPTDoc serialization

void CPTDoc::Serialize(CArchive& ar)
{
        if (ar.IsStoring())
        {
                // TODO: add storing code here
        }
        else
        {
                // TODO: add loading code here
        }
        m_partArray.Serialize(ar);
}

/////////////////////////////////////////////////////////////////////////////
// CPTDoc diagnostics

#ifdef _DEBUG
void CPTDoc::AssertValid() const
{
        CDocument::AssertValid();
}

void CPTDoc::Dump(CDumpContext& dc) const
{
        CDocument::Dump(dc);
        dc<<"\n"<<m_partArray<<"\n";
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CPTDoc commands

void CPTDoc::DeleteContents()
{
#ifdef _DEBUG
        Dump(afxDump);
#endif
        //while(m_partList.GetHeadPosition()) {
```

```
        //      delete m_partList.RemoveHead();}
        m_partArray.RemoveAll();
}

void CPTDoc::OnEditClearAll()
{
        // TODO: Add your command handler code here
        DeleteContents();
        UpdateAllViews(NULL);
}

void CPTDoc::OnUpdateEditClearAll(CCmdUI* pCmdUI)
{
        // TODO: Add your command update UI handler code here
        pCmdUI->Enable(m_partArray.GetSize()!=0);
}


BOOL CPTDoc::OnOpenDocument(LPCTSTR lpszPathName)
{
        if (!CDocument::OnOpenDocument(lpszPathName))
                return FALSE;

        // TODO: Add your specialized creation code here

        return TRUE;
}

BOOL CPTDoc::OnSaveDocument(LPCTSTR lpszPathName)
{
        // TODO: Add your specialized code here and/or call the base class

        return CDocument::OnSaveDocument(lpszPathName);
}

void CPTDoc::OnUpdateFileSave(CCmdUI* pCmdUI)
{
        // TODO: Add your command update UI handler code here
        pCmdUI->Enable(IsModified());
}

void CPTDoc::OnReport()
{
        CPart* pPart;
        CPart* pParentPart;
```

```
CString stopNodeName;

BOOL stopFlag;
UINT productTime;
int nCostDisassembly,i;
double p;

CInputDialog inputDlg;
inputDlg.DoModal();
productTime=inputDlg.m_nTime;

for(i=(m_partArray.GetSize()-1);i>=1;i--)
{
        pPart=m_partArray.GetAt(i);
        pPart->m_fCostR=(pPart->m_nCostNew)+(pPart->m_nCostLandfill);
        pPart->minCost=pPart->m_fCostR;

        pPart->m_fCostM=(pPart->m_nCostMR)+(pPart->m_nCostNew);
        if (pPart->m_fCostM<pPart->minCost){
                pPart->minCost=pPart->m_fCostM;
        }

        p=exp(-(pPart->m_fRateFailure)*productTime);
        pPart->m_fCostC=(pPart->m_nCostRG)*p+(pPart->m_nCostRF)*(1-p)
                +(pPart->m_fRateDate)*productTime*productTime*(pPart-
>m_nCostNew);
        if (pPart->m_fCostC<pPart->minCost){
                pPart->minCost=pPart->m_fCostC;
        }

        pPart->m_fCostP=0.0;
        pPart->IsLeaf=TRUE;
        pPart->IsStopNode=FALSE;
        pPart->m_nChildNumber=0;

}

for(i=m_partArray.GetSize()-1;i>=1;i--)
{
        pPart=m_partArray.GetAt(i);

        if (!pPart->IsLeaf){
                if(pPart->m_strConnectType=="Type1"){
                        nCostDisassembly=5;
                }
```

```
                else if(pPart->m_strConnectType=="Type2"){
                        nCostDisassembly=10;
                }
                else if(pPart->m_strConnectType=="Type3"){
                        nCostDisassembly=15;
                }
                else {
                        nCostDisassembly=20;
                }

                pPart->m_fCostP+=nCostDisassembly;

                if (pPart->m_fCostP<pPart->minCost){
                        pPart->minCost=pPart->m_fCostP;
                }

        }

        if (pPart->m_nParentID!=0) {
                pParentPart=m_partArray.GetAt(pPart->m_nParentID);
                pParentPart->m_fCostP+=pPart->minCost;
                pParentPart->IsLeaf=FALSE;
                pParentPart->m_nChildID[pParentPart-
>m_nChildNumber++]=pPart->m_nID;
        }

}

i=1;
stopFlag=FALSE;
while((i<=m_partArray.GetSize()-1) && (!stopFlag))
{
        pPart=m_partArray.GetAt(i);
        if(!pPart->IsLeaf){
                if(pPart->minCost!=pPart->m_fCostP){
                        pPart->IsStopNode=TRUE;
                        stopNodeName=pPart->m_strName;
                        stopFlag=TRUE;
                }
        }

        i++;
}

if(!stopFlag){
```

```
            stopNodeName="No stop";
        }

        CReportDialog dlg;
        dlg.m_strStopNode=stopNodeName;
        dlg.m_fCostFinal=(m_partArray.GetAt(1))->minCost;
        dlg.m_pArray=&m_partArray;
        int ret=dlg.DoModal();

}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                                //
// File Name:   PTView.h                                                                          //
//                                                                                                //
// Description: The file is the interface of the CPTView class                                    //
//                                                                                                //
// Update date: 10/19/1996                                                                        //
//                                                                                                //
////////////////////////////////////////////////////////////////////////////////////////////////////

class CPTView : public CFormView
{
protected:
        int m_position;
        CPartArray* m_pArray;
protected: // create from serialization only
        CPTView();
        DECLARE_DYNCREATE(CPTView)

public:
        //{{AFX_DATA(CPTView)
        enum { IDD = IDD_PT_FORM };
        CString         m_strName;
        int             m_nCostNew;
        int             m_nCostRF;
        int             m_nCostMR;
        int             m_nCostLandfill;
        int             m_nCostRG;
        UINT  m_nID;
        double m_fRateDate;
        double m_fRateFailure;
        CString         m_strConnectType;
        UINT  m_nParentID;
        //}}AFX_DATA

 // Attributes
 public:
        CPTDoc* GetDocument();

 // Operations
 public:

 // Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CPTView)
```

```
        public:
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        virtual void OnInitialUpdate();
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);   // DDX/DDV support
        virtual void OnUpdate(CView* pSender, LPARAM lHint, CObject* pHint);
        virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
        virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
        virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
        virtual void OnPrint(CDC* pDC, CPrintInfo*);
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CPTView();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
        //{{AFX_MSG(CPTView)
        afx_msg void OnCommandHome();
        afx_msg void OnUpdateCommandHome(CCmdUI* pCmdUI);
        afx_msg void OnCommandEnd();
        afx_msg void OnUpdateCommandEnd(CCmdUI* pCmdUI);
        afx_msg void OnCommandNext();
        afx_msg void OnUpdateCommandNext(CCmdUI* pCmdUI);
        afx_msg void OnCommandPrev();
        afx_msg void OnUpdateCommandPrev(CCmdUI* pCmdUI);
        afx_msg void OnCommandDel();
        afx_msg void OnCommandIns();
        afx_msg void OnClear();
        //}}AFX_MSG
protected:
        virtual void GetEntry(int position);
        virtual void InsertEntry(int position);
        virtual void ClearEntry();

        DECLARE_MESSAGE_MAP()
};
```

```
#ifndef _DEBUG  // debug version in PTView.cpp
inline CPTDoc* CPTView::GetDocument()
   { return (CPTDoc*)m_pDocument; }
#endif

/////////////////////////////////////////////////////////////////////////////



/////////////////////////////////////////////////////////////////////////////
//                                                                         //
// File Name:   PTView.cpp                                                 //
//                                                                         //
// Description: The file is the implementation of the CPTView class        //
//              It define the display behavior of the application          //
//                                                                         //
// Update date: 10/19/1996                                                 //
//                                                                         //
/////////////////////////////////////////////////////////////////////////////


#include "stdafx.h"
#include "PT.h"

#include "part.h"
#include "PTDoc.h"
#include "PTView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CPTView

IMPLEMENT_DYNCREATE(CPTView, CFormView)

BEGIN_MESSAGE_MAP(CPTView, CFormView)
	//{{AFX_MSG_MAP(CPTView)
	ON_COMMAND(ID_PART_HOME, OnCommandHome)
	ON_UPDATE_COMMAND_UI(ID_PART_HOME, OnUpdateCommandHome)
```

```
        ON_COMMAND(ID_PART_END, OnCommandEnd)
        ON_UPDATE_COMMAND_UI(ID_PART_END, OnUpdateCommandEnd)
        ON_COMMAND(ID_PART_NEXT, OnCommandNext)
        ON_UPDATE_COMMAND_UI(ID_PART_NEXT, OnUpdateCommandNext)
        ON_COMMAND(ID_PART_PREV, OnCommandPrev)
        ON_UPDATE_COMMAND_UI(ID_PART_PREV, OnUpdateCommandPrev)
        ON_COMMAND(ID_PART_DEL,OnCommandDel)
        ON_COMMAND(ID_PART_INS,OnCommandIns)
        ON_BN_CLICKED(IDC_CLEAR, OnClear)

        //}}AFX_MSG_MAP
        // Standard printing commands
        ON_COMMAND(ID_FILE_PRINT, CFormView::OnFilePrint)
        ON_COMMAND(ID_FILE_PRINT_DIRECT, CFormView::OnFilePrint)
        ON_COMMAND(ID_FILE_PRINT_PREVIEW,
CFormView::OnFilePrintPreview)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CPTView construction/destruction

CPTView::CPTView()
        : CFormView(CPTView::IDD)
{
        TRACE("Entering CPartView constructor\n");
        //{{AFX_DATA_INIT(CPTView)
        m_strName = _T("");
        m_nCostNew = 0;
        m_nCostRF = 0;
        m_nCostMR = 0;
        m_nCostLandfill = 0;
        m_nCostRG = 0;
        m_nID = 1;
        m_fRateDate = 0.0;
        m_fRateFailure = 0.0;
        m_strConnectType = _T("");
        m_nParentID = 0;
        m_position=0;
        //}}AFX_DATA_INIT
        // TODO: add construction code here

}

CPTView::~CPTView()
{
```

```
}

void CPTView::DoDataExchange(CDataExchange* pDX)
{
	CFormView::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CPTView)
	DDX_Text(pDX, IDC_NAME, m_strName);
	DDV_MaxChars(pDX, m_strName, 30);
	DDX_Text(pDX, IDC_COST_NEW, m_nCostNew);
	DDX_Text(pDX, IDC_COST_RF, m_nCostRF);
	DDX_Text(pDX, IDC_COST_MR, m_nCostMR);
	DDX_Text(pDX, IDC_COST_LANDFILL, m_nCostLandfill);
	DDX_Text(pDX, IDC_COST_RG, m_nCostRG);
	DDX_Text(pDX, IDC_ID, m_nID);
	DDX_Text(pDX, IDC_RATE_DATE, m_fRateDate);
	DDX_Text(pDX, IDC_RATE_FAILURE, m_fRateFailure);
	DDX_CBString(pDX, IDC_CONNECT, m_strConnectType);
	DDX_Text(pDX, IDC_PARENTID, m_nParentID);
	//}}AFX_DATA_MAP
}

BOOL CPTView::PreCreateWindow(CREATESTRUCT& cs)
{
	// TODO: Modify the Window class or styles here by modifying
	//  the CREATESTRUCT cs

	return CFormView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CPTView printing

BOOL CPTView::OnPreparePrinting(CPrintInfo* pInfo)
{
	// default preparation
	return DoPreparePrinting(pInfo);
}

void CPTView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
	// TODO: add extra initialization before printing
}

void CPTView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
```

```
	// TODO: add cleanup after printing
}

void CPTView::OnPrint(CDC* pDC, CPrintInfo*)
{
	// TODO: add code to print the controls
}

/////////////////////////////////////////////////////////////////////////////
// CPTView diagnostics

#ifdef _DEBUG
void CPTView::AssertValid() const
{
	CFormView::AssertValid();
}

void CPTView::Dump(CDumpContext& dc) const
{
	CFormView::Dump(dc);
}

CPTDoc* CPTView::GetDocument() // non-debug version is inline
{
	ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CPTDoc)));
	return (CPTDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CPTView message handlers

void CPTView::OnCommandHome()
{
	TRACE("Entering Home\n");
	if ((m_pArray->GetSize())!=0) {
		m_position=1;
		GetEntry(m_position);
	}
}
void CPTView::OnCommandNext()
{
	TRACE("Entering Next\n");
	if (++m_position<=m_pArray->GetSize()-1) {
		GetEntry(m_position);
```

```
        }
}

void CPTView::OnCommandPrev()
{
        TRACE("Entering Next\n");
        if (--m_position>=1) {
                GetEntry(m_position);
        }
}

void CPTView::OnCommandEnd()
{
        int tailIndex;
        TRACE("Entering End\n");
        if ((tailIndex=m_pArray->GetSize())!=0) {
                m_position=tailIndex-1;
                GetEntry(m_position);
        }
}

void CPTView::OnCommandIns()
{
        TRACE("Entering OnCommandIns\n");
        InsertEntry(m_position);
        GetDocument()->SetModifiedFlag();
        GetDocument()->UpdateAllViews(this);
}

void CPTView::OnCommandDel()
{
        CPart* pP;
        if ((pP=m_pArray->GetAt(m_position))!=NULL){
                m_pArray->RemoveAt(m_position);
                delete pP;

                GetEntry(1);
                m_position=1;
                GetDocument()->SetModifiedFlag();
                GetDocument()->UpdateAllViews(this);
        }
}


void CPTView::OnUpdateCommandHome(CCmdUI* pCmdUI)
```

```
{
        pCmdUI->Enable( (m_position>1) && (m_position<=m_pArray->GetSize()-1 )
);
}

void CPTView::OnUpdateCommandEnd(CCmdUI* pCmdUI)
{
        pCmdUI->Enable((m_position<=m_pArray->GetSize()-2)&&(m_position>=0));

}

void CPTView::OnUpdateCommandPrev(CCmdUI* pCmdUI)
{
        pCmdUI->Enable((m_position>1)&&(m_position<=m_pArray->GetSize()-1));

}

void CPTView::OnUpdateCommandNext(CCmdUI* pCmdUI)
{
        pCmdUI->Enable( (m_position>=0) && (m_position<=m_pArray->GetSize()-2)
);

}


void CPTView::OnInitialUpdate()
{
        TRACE("Entering OnInitialUpdate\n");
        CFormView::OnInitialUpdate();

        // TODO: Add your specialized code here and/or call the base class

}


void CPTView::OnClear()
{
        // TODO: Add your command handler code here
        TRACE("Entering Onclear\n");
        ClearEntry();
}

void CPTView::GetEntry(int m_position)
{
        if ((m_position<=m_pArray->GetSize()-1)&&(m_position>=1)) {
```

```
                CPart* pPart=m_pArray->GetAt(m_position);
                m_strName=pPart->m_strName;
                m_strConnectType=pPart->m_strConnectType;
                m_nID=pPart->m_nID;
                m_nParentID=pPart->m_nParentID;
                m_nCostNew=pPart->m_nCostNew;
                m_nCostMR=pPart->m_nCostMR;
                m_nCostRG=pPart->m_nCostRG;
                m_nCostRF=pPart->m_nCostRF;
                m_nCostLandfill=pPart->m_nCostLandfill;
                m_fRateFailure=pPart->m_fRateFailure;
                m_fRateDate->pPart->m_fRateDate;
        }
        else {
                ClearEntry();
        }
        UpdateData(FALSE);
}

void CPTView::InsertEntry(int position)
{
        if (UpdateData(TRUE)){
                CPart* pPart=new CPart;
                pPart->m_strName=m_strName;
                pPart->m_strConnectType=m_strConnectType;
                pPart->m_nID=m_nID;
                pPart->m_nParentID=m_nParentID;
                pPart->m_nCostNew=m_nCostNew;
                pPart->m_nCostMR=m_nCostMR;
                pPart->m_nCostLandfill=m_nCostLandfill;
                pPart->m_nCostRF=m_nCostRF;
                pPart->m_nCostRG=m_nCostRG;
                pPart->m_fRateFailure=m_fRateFailure;
                pPart->m_fRateDate=m_fRateDate;
                m_pArray->InsertAt(++m_position,pPart);
        }
}

void CPTView::ClearEntry()
{
        m_strName="";
        m_strConnectType="";
        m_nID=1;
        m_nParentID=0;
        m_nCostNew=0;
```

```
        m_nCostMR=0;
        m_nCostLandfill=0;
        m_nCostRF=0;
        m_nCostRG=0;
        m_fRateFailure=0;
        m_fRateDate=0;

        UpdateData(FALSE);
        ((CDialog*) this)->GotoDlgCtrl(GetDlgItem(IDC_NAME));
}

void CPTView::OnUpdate(CView* pSender, LPARAM lHint,CObject* pHint)
{
        TRACE("Entering OnUpdate\n");
        m_pArray=GetDocument()->GetArray();
        m_position=0;
        GetEntry(m_position);
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                                //
// File Name:   ReportDialog.h                                                                    //
//                                                                                                //
// Description: The file is a  header file. It is the  defination of the class                    //
//              ReportDialog                                                                      //
//              The dialog is used to output the result                                           //
//                                                                                                //
// Update date: 10/19/1996                                                                        //
//                                                                                                //
////////////////////////////////////////////////////////////////////////////////////////////////////


/////////////////////////////////////////////////////////////////////////////
// CReportDialog dialog

class CReportDialog : public CDialog
{
// Construction
public:
        CReportDialog(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
        //{{AFX_DATA(CReportDialog)
        enum { IDD = IDD_REPORT };
        double m_fCostFinal;
        CString         m_strStopNode;
        CImageList m_imageList;
        CPartArray* m_pArray;
        //}}AFX_DATA


// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CReportDialog)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:

        // Generated message map functions
        //{{AFX_MSG(CReportDialog)
        virtual void OnOK();
```

```
        virtual BOOL OnInitDialog();
        void insertNode(UINT x,TV_INSERTSTRUCT* ptv, CTreeCtrl* pTree);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                                        //
// File Name:   ReportDialog.cpp                                                                          //
//                                                                                                        //
// Description: The file is the implementation of the class ReportDialog                                  //
//              The dialog is used to output the result                                                   //
//                                                                                                        //
// Update date: 10/19/1996                                                                                //
//                                                                                                        //
////////////////////////////////////////////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "PT.h"
#include "part.h"
#include "ReportDialog.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CReportDialog dialog


CReportDialog::CReportDialog(CWnd* pParent /*=NULL*/)
        : CDialog(CReportDialog::IDD, pParent)
{
        //{{AFX_DATA_INIT(CReportDialog)
        m_fCostFinal = 0.0;
        m_strStopNode = _T("");
        //}}AFX_DATA_INIT
}


void CReportDialog::DoDataExchange(CDataExchange* pDX)
{
```

```
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CReportDialog)
	DDX_Text(pDX, IDC_FINALCOST, m_fCostFinal);
	DDX_Text(pDX, IDC_STOPNODE, m_strStopNode);
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CReportDialog, CDialog)
	//{{AFX_MSG_MAP(CReportDialog)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CReportDialog message handlers

void CReportDialog::OnOK()
{
	// TODO: Add extra validation here

	CDialog::OnOK();
}

BOOL CReportDialog::OnInitDialog()
{
	CDialog::OnInitDialog();

	// TODO: Add extra initialization here
	HICON hIcon[3];

	m_imageList.Create(16,16,0,0,8);

	hIcon[0]=AfxGetApp()->LoadIcon(IDI_RED);
	hIcon[0]=AfxGetApp()->LoadIcon(IDI_BLUE);
	hIcon[0]=AfxGetApp()->LoadIcon(IDI_GREEN);

	for (int n=0;n<3;n++) {
		m_imageList.Add(hIcon[n]);
	}

	CTreeCtrl* pTree=(CTreeCtrl*) GetDlgItem(IDC_TREEVIEW1);
	pTree->SetImageList(&m_imageList,TVSIL_NORMAL);

 //tree structure common values
	TV_INSERTSTRUCT tvinsert;
```

```
	tvinsert.hParent=NULL;
	tvinsert.hInsertAfter=TVI_LAST;
	tvinsert.item.mask=TVIF_IMAGE|TVIF_SELECTEDIMAGE|TVIF_TEXT;
	tvinsert.item.hItem=NULL;
	tvinsert.item.state=0;
	tvinsert.item.stateMask=0;
	tvinsert.item.cchTextMax=8;
	tvinsert.item.iSelectedImage=1;
	tvinsert.item.cChildren=0;
	tvinsert.item.lParam=0;

//
	/*strcpy(tvinsert.item.pszText,LPCTSTR(m_pArray->GetAt(1)->m_strName)) ;
	tvinsert.item.iImage=1;
	HTREEITEM hDad=pTree->InsertItem(&tvinsert);

	tvinsert.hParent=hDad;

	strcpy(tvinsert.item.pszText,LPCTSTR(m_pArray->GetAt(2)->m_strName)) ;
	tvinsert.item.iImage=2;
	HTREEITEM hD=pTree->InsertItem(&tvinsert);

	strcpy(tvinsert.item.pszText,LPCTSTR(m_pArray->GetAt(3)->m_strName)) ;
	tvinsert.item.iImage=2;
	HTREEITEM hDa=pTree->InsertItem(&tvinsert);

	*/
	insertNode(1,&tvinsert,pTree);
	return TRUE;  // return TRUE unless you set the focus to a control
		// EXCEPTION: OCX Property Pages should return FALSE
}

void CReportDialog::insertNode(UINT x,TV_INSERTSTRUCT* ptv,CTreeCtrl* pTree)
{

	strcpy(ptv->item.pszText,LPCTSTR(m_pArray->GetAt(x)->m_strName));
	if(m_pArray->GetAt(x)->IsStopNode){
		ptv->item.iImage=2;
	}
	else {
		ptv->item.iImage=1;
	}
	HTREEITEM hP=pTree->InsertItem(ptv);

	if(m_pArray->GetAt(x)->m_nChildNumber==0){
```

```
                return;
        }
        else {
                ptv->hParent=hP;
                for(int k=0;k<m_pArray->GetAt(x)->m_nChildNumber;k++) {
                        insertNode(m_pArray->GetAt(x)->m_nChildID[k],ptv,pTree);
                }
        }
}
```

REFERENCE

1. D. Navin-Chandra, "The Recovery Problem in Product Design", *Journal of Engineering Design*, Vol. 5(1), pp. 67-87, 1994.

2. K. Ishii, C. F. Eubanks and P. D. Marco, "Design for Product Retirement and Material Life-cycle", *Materials & Design*, Vol. 15, Number 4, pp. 225-233, 1994.

3. G. A. Keoleian and D. Menerey, "Sustainable Development by Design: Review of Life Cycle Design and Related Approaches", *AIR & WASTE* , Vol. 44, pp. 645-668, May 1994.

4. R. W. Chan, D. Navin-Chandra and F. B. Prinz, "A Cost-Benefit Analysis Model of Product Design for Recyclability and its Application" , *IEEE Transactions on Components Packaging, and Manufacturing Technology-Part A*, Vol. 17, No. 4, pp. 502-507, December 1994. .

5. W. J. Glantschnig, " Green Design: An Introduction to Issues and Challenges", *IEEE Transactions on Components Packaging, and Manufacturing Technology-Part A*, Vol. 17, No. 4, pp. 508-513, December 1994.

6. R. A. Bones, " Design for Reliability " , *Engineering*, pp. 798-800, Nov. 1976.

7. K. Ishii and B. Lee, "Reverse Fishbone Diagram: A Tool in Aid of Design for Product Retirement", *1996 ASME Design Technical Conference.*

8. T. Amezquita, R. Hammond, M. Salazar, and Bert Bras, "Characterizing the Remanufacturablity of Engineering System", *1995 Design Engineering Technical Conferences,* Volume 1, pp. 271-278, ASME 1995.

9. J. Emblemsvag and B. Bras, "Activity-based Costing in Design for Product Retirement", *Advances in Design Automation,* Vol. 2, pp. 351-361, ASME 1994.

10. J. F. Scheuring, B. Bras, and K. M. Lee, "Effects of Design for Disassembly on Integrated Disassembly and Assembly Processes", working paper, G.I.T., 1994.

11. Bras and J. Emblemsvag, "The Use of Activity-based Costing, Uncertainty, and Disassembly Action Charts in Demanufacture Cost Assessments", *1995 Design Engineering Technical Conference*, Vol. 1, pp. 285-292, ASME 1995.

12. G. A. Hazelrigy, *Systems Engineering: An Approach to Information-Based Design*, Prentice Hall, Upper Saddle River, New Jersey, 1996.

13. H. L. Rishel, T. M. Boston, and C. J. Schmidt, *Costs of Remedial Response Actions at Uncontrolled Hazardous Waste Sites,* Noyes Publications, Park Ridge, New Jersey, 1984.

14. T. E. Graedel and B. R. Allenby, *Industrial Ecology,* Prentice Hall, Englewood Cliffs, New Jersey 07632.