THESIS REPORT

Master's Degree

A Systems Engineering Approach to the Development of an Information System for Creating ISO 9000 Quality Documentation

by A.H. Zhong
Advisor: G.M. Zhang

M.S. 94-1

Sponsored by
the National Science Foundation
Engineering Research Center Program,
the University of Maryland,
Harvard University,
and Industry

Abstract

Title of Thesis:	A Systems Engineering Approach To The Development Of An Information System For Creating ISO 9000 Quality Documentation
Name of Degree Candidate:	Anna Hua Zhong
Degree and Year:	Master of Science, 1994
Thesis Directed By:	Dr. Guangming Zhang Assistant Professor Institute for Systems Research & Department of Mechanical Engineering

ISO 9000 is a series of international quality standards developed by the International Organization for Standardization (ISO) in 1987. It provides a comprehensive set of generic standards that applies to all phases of the product development cycle, including design, manufacturing, and service. Since its establishment, ISO 9000 has gained widespread acceptance by companies as an integral part in achieving total quality management. More and more companies are registering to ISO 9000 to show their commitment to quality.

One of the key components in the ISO 9000 certification process is the quality manual, which deals with the company's business procedures ranging from design to service. With rapid advancements of computer technologies, the task of producing such a quality manual can be done more efficiently with the help of a

well-designed information system. This thesis presents the design and implementation of such an information system where systems engineering principles are incorporated. A survey of relevant information including quality, ISO 9000, information system, database, human factors, user interface, and tradeoff analysis is also presented.

Three unique features of the developed information system are:

- System architecture, which follows the basic framework of the ISO 9000 standards in terms of data storage, user interface and report generation.
- Microsoft Windows and Visual Basic development platform, which makes the prototype ideally suited for small companies such as Compression Telecommunications Corporation (CTEL), an industry sponsor.
- Relational database approach, which offers flexibility and makes the prototype adaptable to the needs of small companies.

The information system prototype developed in this thesis work has been used to produce a quality manual for Compression Telecommunications Corporation (CTEL), and will be used in the ISO 9000 registration process.

A Systems Engineering Approach To The Development Of An Information System For Creating ISO 9000 Quality Documentation

by

Anna Hua Zhong

Thesis submitted to the Faculty of the Graduate School of
The University of Maryland in partial fulfillment of
the requirements for the degree of
Master of Science
1994

Advisory Committee:

Assistant Professor Guangming Zhang, Chairman / Advisor
Associate Professor Thomas Fuja
Associate Professor Michael Pecht
Professor Steven Spivak

Special Thanks To

Dr. Zhang, Michael, and Wing

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 Overview of ISO 9000

Today, quality has emerged as an important strategic weapon in the marketplace. American industry is paying a close attention to this message and responding to the challenge by providing quality products and services at competitive prices. Quality engineering, which focuses on productivity and quality improvements, has become an integral part of the management strategy.

Changes in the global economy have caused American companies to take a hard look at the way they and others have done business in the past. Central to the quality revolution are two issues that continue to receive increasing attention. One of these is a growing awareness and understanding of the roles and responsibilities of management in dealing with quality. The other is an increased understanding of both the needs for and the concepts and methods required to move the quality issue upfront into product planning and the engineering design process. ISO 9000 emphasizes both of these issues.

ISO 9000, a series of quality standards established by the International Organization for Standardization (ISO), has gained widespread acceptance by companies who wish to implement total quality management in products design, customer services, process controls and management practices. More and more

companies are registering to ISO 9000 to show their commitment to quality, and to prove to their customers that their products or services are of the highest quality.

Companies of different sizes and different industries are involved in the ISO 9000 registration process, both as registrars and registees. A recent survey conducted in the eastern region of the United States reveals that a number of quality registrars have actively worked with a significant number of clients for ISO 9000 certification. Big names such as AT&T and Dupont are among registrars in this country. Companies such as IBM and Northern Telecom are proud to advertise that their manufacturing plants are either already ISO 9000 registered, or in the process of getting registered.

The United States national standardization bodies have also recognized ISO 9000. In fact, the American National Standard Institute (ANSI) has urged companies to speed up the ISO 9000 registration process in order to strengthen the competitiveness of U.S. business in the world market. In April, 1993, the National Institute of Standards and Technology (NIST), published a booklet named "Questions and Answers on Quality, the ISO 9000 Standard Series, Quality system Registration, and Related Issues". The booklet provides important information on total quality management in general and ISO 9000 in particular.

1.2 The need for an information systems approach

Almost every company implements some sort of quality systems for design and process control. These systems may succeed and fail due to various reasons, but by far the single most common reason for the failure of a quality system is poor documentation. Documentation is essential because it ensures a continuous operation of the business by facilitating the transition of knowledge from one employee to another. Therefore, ISO 9000 certification places significant emphasis on quality documentation in terms of the quality manual.

The quality manual is a set of tiered documents detailing procedures a company should follow in order to do its business, with the first tier covering high level procedures and referring to lower tiers for specific information. The quality manual for ISO 9000 is unique because it deals with the quality of underlying process controls that are needed in the product development cycle. This hierarchical structure of the quality manual lends itself to the utilization of information systems.

Recent headlines on the information superhighway promise to change the way companies do business. With features such as full internet connection, hand-held computers, and video conferencing, companies can practically send and receive information when they want it and where they want it. However, looking at the way most companies do business today, simple word processors or even

typewriters are still commonly used to produce documentation including the quality manual. Off-the-shelf word processors are inadequate and extremely inefficient if companies want information to be at their fingertips. The following are several of the reasons:

- Information retrieval is often difficult with word processing files because of their sequential rather than hierarchical nature.
- A large number of redundant information may be stored in word processing files. For example, if a certain employee, say John Doe, is responsible for several ISO 9000 elements, his name may have to be repeated several places in the files.
- Redundancy may cause problems in updating information in files. For example, if say Mary Smith takes over John Doe's responsibilities, changes have to be made in several places in the files accordingly.
- Most word processing files are not designed to be shared. Only one person may edit a file at a time. For example, if say John Doe is updating information for element 1, then Mary Smith can not do anything with element 5 until John is done with the file, even though elements 1 and 5 may be totally unrelated.
- Related information are isolated into separate word processing files, and that makes information access difficult. For example, when employee

information, which resides in a separate file, is to be included in the quality manual, a lot of cut-and-paste may be required.

- Mix-and-match information from different word processing files is difficult. Suppose a consulting company that specializes in helping companies produce quality manuals has a set of several model quality manuals. Now they want to create a new one based on those existing models, with some information from each. It may take a significant amount of time and effort to dig out relevant information from previous word processing files and combine them into one.

It is evident that word processors are inadequate for the purpose of creating and maintaining quality manuals. Several of the issues mentioned above such as redundancy and updating problems, can be solved with a database information system. However, off-the-shelf database systems come with their own problems:

- Database tables and queries need to be set up by someone who is familiar with ISO 9000.
- Database information entry forms that come with the application may not be suited for the purpose of creating quality manuals.
- Reports generated may not be in the format required for ISO 9000 registration.

What is needed then is an information system specifically designed to generate quality manuals. This information system includes a central repository designed to store ISO 9000 related information, a user interface that follows the flow of the twenty elements in the ISO 9000 series, and a report generation facility that produces reports in a format specified by ISO 9000. Such an information system can be readily adapted into the information superhighway architecture.

1.3 Scope and organization of the thesis

The purpose of this thesis work is to design and develop an information system for creating ISO 9000 quality documentation using the systems engineering approach. Topics such as methodology, requirements, design, and prototype development are discussed in this thesis. A prototype of the information system is also created and used to help generating a quality manual for Compression Telecommunications Corporation (CTEL), an industry sponsor.

This thesis presentation is organized into six chapters, an appendix, and a separate document. Chapter one gives an introduction and a rationale for this thesis work. Chapter two gives an overview of total quality management (TQM) and ISO 9000. Chapter three gives an introduction of information systems and their evolution.

Chapter four incorporates systems engineering principles in the design and development of information systems. This chapter contains extensive information on topics such as different approaches to information system design, database design, user interface design, and tradeoff analysis. Examples of how these systems engineering principles are used in the design of the information system are also discussed in this chapter.

Chapter five deals with the design and development of the information system. Theoretical information presented in previous chapters are incorporated into the prototype design and development phase. Emphases are place on central repository and user interface design. The central repository is designed using the relational database approach, while the user interface utilizes industry standard Microsoft Windows GUI based interface. Entitiy-relationship models, database tables, and window layouts are included in this chapter. In addition, this chapter covers conceptual design, preliminary design, detailed design, and development of the prototype.

Chapter six concludes with a summary of the information system and recommendations for future improvements. The appendix contains the Visual Basic program source code for the information system prototype. The quality manual for Compression Telecommunications Corporation (CTEL) is attached as a separate document.

Chapter 2 Quality Management And ISO 9000

2.1 Total quality management

2.1.1 Definition of quality

Quality is not a new concept. People have been talking about craftsmanship for centuries. What does it mean by someone having craftsmanship? It means that the person has the skills to produce quality products. Quality is an attribute related to not only products but also people who produce those products.

Everyone has his or her own view of quality. Consumers view quality in terms of what they expect as a fair value for what they have paid. Quality for consumers is therefore related to function, price and service. Producers view quality as a measure of conformance to specifications, standards or contractual agreements. However, mere conformance is not adequate. Producers must also provide assurance on the quality of their products' design and performance. Attributes such as reliability, safety and maintainability should always be on producers' quality list.

Quality is also about people. Producing quality products requires the commitment of not only the employer but also employees. Take a look at McDonald's hamburg outlets. No matter where one goes in the United States or even around the world, one can always expect the same clean restaurant and

friendly and fast service. This is the kind of quality assurance that requires the commitment of thousands of people involved.

2.1.2 History of quality

Many people credit the recent emphasis on quality to Deming, but the modern concept of total quality management can be traced back to Frederick Taylor, when he first separated management from the work force. Management had the responsibility of setting standards, and workers performed activities based on these standards. Independent inspections were held to weed out defective products.

Quality is a relative term, and therefore must be measurable. The concept of statistical quality control came about during World War II, when the quality of weapon systems became an important consideration. After the war, concepts such as probabilities, control charts, sampling and process designs were adapted into civilian manufacturing processes. Scientists who made significant contributions in the area of statistical quality control include H. F. Dodge, H. G. Roming and Walter A. Shewhart.

Another important person in the history of total quality management (TQM) is W. Edwards Deming, who is credited with the dramatic improvements in the quality of Japanese products, and the recent emphasis on quality in the United

States. Deming, as a management consultant, devised fourteen points that were followed faithfully after World War II by Japanese companies and now by a number of American companies. In his fourteen points, Deming heavily emphasized management involvement and employee commitment, thus underscoring the importance of people in total quality management.

2.2 The ISO 9000 series of standards

2.2.1 History of ISO 9000

The ISO 9000 series of standards were originated in Europe in 1987, when the International Organization for Standardization (ISO) published the series. However, the ideas behind these standards date back to 1979, when Geneva based ISO formed Technical Committee (TC) 176. TC 176's mission was to address worldwide customer demands for product quality and the increasing confusion resulting from differences in quality systems. When the ISO 9000 series of standards were published in 1987, they incorporated inputs from many European countries to produce a set of generic, consistent quality standards that can be applied to not only manufacturing but also service industries. The EC92 trust, which combined twelve European countries into one economic community, became an important driving force for the adoption of ISO 9000 into the European

community, and subsequently the rest of the world including the United States, Canada, and Japan.

2.2.2 Elements of ISO 9000

ISO 9000 was intended to be advisory in nature. Companies can devise their own quality plans and procedures based on a set of guidelines. Because of its broad scope, ISO 9000 can be applied to companies of different sizes and in different industries. The basic ISO 9000 series is composed of five standards -- ISO 9000, ISO 9001, ISO 9002, ISO 9003, and ISO 9004. ISO 9000 and ISO 9004 are guidance standards designed to be descriptive in nature, while ISO 9001, ISO 9002 and ISO 9003 are conformance standards with a prescriptive nature that companies can register to. This thesis work is concerned with the ISO 9001 and ISO 9002 standards. ISO 9001 is the most comprehensive in the series, and ISO 9002 applies to the manufacturing industry to which CTEL belongs. The following is a list of the twenty elements in ISO 9001, and elements in ISO 9002 are a subset of those:

- Management Responsibility
- Quality System
- Contract Review
- Design Control

- Document Control
- Purchasing
- Purchaser-Supplied Product
- Product Identification and Traceability
- Process Control
- Inspection and Testing
- Inspection, Measuring and Test Equipment
- Inspection and Test Status
- Control of Nonconforming Product
- Corrective Action
- Storage, Packaging and Delivery
- Quality Records
- Quality Audits
- Training
- Servicing
- Statistical Techniques

2.2.3 The certification process of ISO 9000

Quality systems registration is the assessment and audit of a company's products by a third party. There is recently an increasing number of quality

registrars who offer the services of ISO 9000 certification. The certification process often involves an initial on-site visit by a team from the registrar to document facility and process compliance to the standard. If the registrar believes that the company conforms to the standard, the company is then registered to one of the prescriptive standards in the series -- ISO 9001, ISO 9002 or ISO 9003. Registration is often granted for a period of three years. During the 3-year period, the registrar will conduct additional on-site surveys and inspections.

Since large companies often have a number of manufacturing sites, ISO 9000 allows the separate certification of different sites. This is so that if one site fails the inspection, the other sites may still be certified. The company may, of course, choose to register several sites simultaneously.

The certification process usually involves the following six steps:

- Application -- The company initiates the registration process.
- Document review -- The registrar reviews the company's process documentation, often called the quality manual.
- Pre-Assessment -- The registrar conducts a small audit designed to point out the company's major deficiencies so that the company may correct them before the assessment step.
- Assessment -- The registrar conducts a complete audit to determine whether the company will be registered.

- Registration -- The company will receive one of three possible outcomes - approval, conditional or provisional approval, or disapproval.
- Surveillance -- During the three year registration period, the registrar will conduct on-site inspections to ensure that the company conforms to the standard. These inspections are often held in 6-months intervals. At the end of the registration period, the company may decide whether to register again.

Therefore, in order to be ISO 9000 certified, a company must prepare a set of documentation, and the most important documentation is the company's quality manual. The task of producing such a quality manual can be done more efficiently with the help of a well-designed information system.

Chapter 3 The Evolution Of Information Systems

3.1 The evolution of computers

The history of computer information systems started with the invention of computers. The ancestry of modern computers can be traced back to the seventeenth century, when machines capable of performing the four basic arithmetic operations -- addition, subtraction, multiplication, and division, first appeared. In 1642, the French philosopher and scientist Blaise Pascal built a machine to automatically perform addition and subtraction. Later a German philosopher and mathematician Gottfried Leibniz constructed a similar machine capable of performing also multiplication and division.

An important contribution to the invention of computers is the use of punch cards. Punch cards were originally developed to ease the task of weaving multiple copies of patterned material. In 1801, Joseph Jacquard produced a successful "programmable" loom in which all the power was supplied mechanically and all the control via punch cards.

The next major step came when an Englishman, Charles Babage, designed the Difference Engine and the Analytical Engine. The Difference Engine, like earlier machines, was capable of performing only additions and subtractions. However, using a mathematical technique know as finite differences, the

Difference Engine could be used to compute a large number of formulas -- polynomials, trigonometric functions, using just additions. The Analytical Engine, designed a little bit later by Babage, was more of a general purpose device. It had fundamentally the same components as modern computers, with input/output devices, central processors, and storage devices

In the 1930s, after the invention of electricity, two persons -- Zuse and Aiken, developed separately electromechanical computers. Zuse, a German engineer, conceived the idea of a device that used mechanical relays or switches which could be opened or closed automatically. This design necessitated the use of a binary system. Zuse built a series of general-purpose program-controlled computers, named from Z1 to Z4. At about the same time, Howard Aiken, a physicist and mathematics professor at Harvard University, built an electromechanical device named Mark I.

"Real" computers came about with the invention of vacuum tubes. During World War II, to counter Germany's encryption device Enigma, the British mathematician Alan Turing was given the responsibility of designing a decryption device using vacuum tube technologies. The first such machine, named Colossus, became operational in 1943.

Another famous vacuum tube computer was the ENIAC. The effort started when John Vincent Atanasoff, an associate professor of physics and

mathematics at Iowa State College, designed a special purpose machine for solving simultaneous linear equations. Since Atanosoff built the machine with the help of his student Clifford Berry, the machine became known as the Atanasoff-Berry Computer or the ABC. Later, John Mauchly and John Presper Echert, greatly inspired by the ABC, built ENIAC at the University of Pennsylvania. The ENIAC, completed in 1946, was believed to be the world's first general-purpose electronic digital computer.

After working on ENIAC as a consultant, the mathematician John von Newmann set out to work on the design of a new stored-program computer, referred to as the IAS computer. The essence of the stored-program computer was that programs can be stored in memory alongside data. The IAS computer was the prototype of all subsequent general-purpose computers.

In the 1940s and 50s, computers were also commercialized. Eckert and Mauchly built UNIVAC I and UNIVAC II. IBM Corporation introduced the 700 series of computers, which later established the company as a dominant computer manufacturer.

Transistors marked the beginning of the second generation of computers. In 1947, AT&T's Bell Laboratories invented transistors, and a few years later, NCR, IBM, DEC all began building computers based on this new technology.

The third generation of computers appeared with the invention of integrated circuit boards. Again, large companies such as IBM and DEC played important roles in the development and commercialization of computers using microelectronics technologies.

Later contributions to computer technologies included semiconductor memory and microprocessors. Today, there are mainly three types of computers -- microcomputers, minicomputers and mainframes. However, computer technologies are changing so fast even once overwhelmingly dominant companies such as IBM and UNISYS are having trouble keeping up. The future direction of computers is heading toward the integration of computers with other technologies such as telecommunications. AT&T's latest commercials emphasized these trends with previews of video phones, video conferencing devices, and voice recognition devices, etc. Computer manufacturers are also teaming up with the entertainment industry with a series of mergers to bring viewers multimedia entertainment and in-house shopping among other things. Computers are getting smaller and more powerful by the day. Latest entries in the hand-held computer arena include Apple Computer's Newton, which weighs only a few pounds, fits in the palm of a hand, but is capable of recognizing handwritings and sending faxes. Recent talks of the information superhighway promise to connect all computers into a worldwide network.

3.2 The evolution of computer programming languages

As computer hardware evolved from early mechanical devices to today's integrated circuits, programming languages also evolved from early machine codes to today's code generation tools. When programmers first started to program computers, they literally had to tell the computers what to do using machine codes, which were series of 0s and 1s. Later assembly languages were developed so that people could use some simple instructions such as "load" or "add," and a program would translate these instructions into series of 0s and 1s that the machine could understand. The invention of high-level programming languages revolutionized software development. Early high-level languages included FORTRAN, Lisp and COBOL, and more recent ones included Pascal and Ada. High-level languages enabled the development of large scale quality software using various techniques and methodologies such as top-down design and structured programming. Recent developments in this field include object-oriented programming languages, and code generation tools.

As computer hardware gets smaller and more powerful, software applications get larger and more extensive. Word processor applications such as Word Perfect that used to fit on one or two low density diskettes now require ten high density ones. Packed in these ten diskettes are functions that software

designers and users could only dream about ten years ago. An important feature is the Graphical User Interface (GUI) support with menus, icons, and drag-and-drop. Computer games have always been a good measure of software capabilities. Games in the old days were text based, whereas now they use color graphics, animation, and even multimedia. A computer game manufacture recently announced that it will deliver its future products on CD-ROMs to include features such as digitized movie images and sound tracks.

3.3 The evolution of database information systems

A database system is a special type of information system. Database systems are repositories used to store information in an orderly fashion so that users may access the information later. Databases may be defined by the functions they perform. Any database systems, at the very least, should give users facilities to perform the following functions:

- Adding files to the database
- Deleting files from the database
- Adding data into existing files
- Deleting data from existing files
- Updating data in existing files
- Retrieving data from existing files.

Databases may also be defined by their components. In order to perform those functions mentioned above, a database system will need to have a number of components, including hardware, software, data and users. The following picture shows a simplified view of a database system:

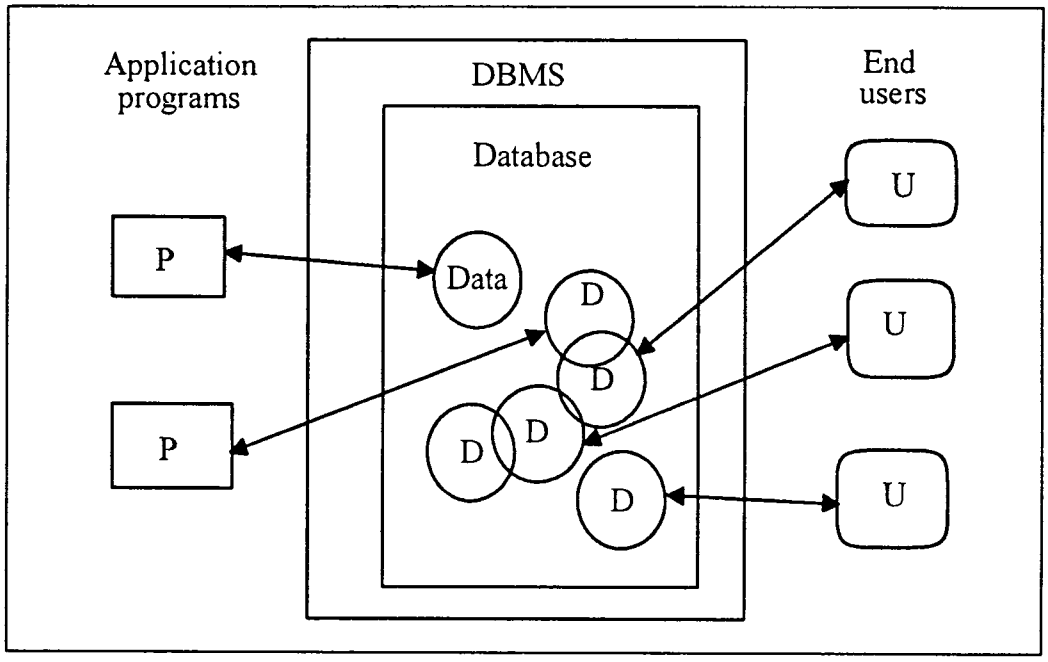

Figure 3.1 A Simplified View Of A Database System

The hardware components of a database consist of storage, I/O devices, device controllers, and processors, etc. The software components are often referred to as database management systems (DBMS). The function of a DBMS is to shield users from the underlying hardware, and facilitate user operations. One

of the most important tasks that a DBMS supports is SQL operation, which simplifies the task of data maintenance.

Another component of a database system is data. Data may be dedicated or shared, depending on whether the system is single-user or multi-user. On a single user system, only one user may access the data at a time, whereas on a multi-user system, several users may access the same piece of data at the same time. Obviously, on a multi-user system, some access control mechanisms need to be built in.

The last component in a database system is user. There are two types of users, applications programmers and end-users. Applications programmers write programs that use database systems. They typically use programming languages such as C or COBOL to perform operations through DBMS. On the other hand, end-users do not normally access databases directly, instead they perform tasks through applications programs.

There are several kinds of database architecture -- inverted list, hierarchic, network, relational, and object-oriented. Each architecture has its advantages and disadvantages. Inverted list and hierarchic systems were popular in the early days of database systems due to their high access speed, but relational databases are currently the most widely used because they are easy to maintain. Virtually all database systems developed in the past few years are based on the relational

model. The latest development in database information systems is object-oriented database architecture, which treats data not as rows and columns, but as objects. However, object-oriented databases are still at early stages of development.

3.4 The evolution of the design and development of information systems

Early information systems were small and could often be designed and developed by a small number of programmers. Persons who wrote the programs were also the ones who maintained them. Documentation was poor and system life cycles were short.

With technological advancements, large scale information systems can no longer be designed and developed by just a few computer hackers. Today's large scale software development teams usually include engineers, computer scientists, usability specialists, testing specialists, and documentation specialists. Designing and developing a large information system often require years of hard work and millions of dollars. With a large number of people involved, it is essential that designers follow systematic approaches throughout the entire system life cycle, including design, development and documentation.

Chapter 4 The Systems Engineering Approach To Information Systems Design And Development

4.1 Overview

There are various approaches to information systems design and development, among them are the systems approach, the waterfall model, and object-oriented design, etc. Each approach has advantages and disadvantages. Several approaches can be used in different phases of the same project to maximize the benefits of each. For example, the waterfall model may be used for high level design, while object-oriented design may be used for prototyping.

A database system is a special type of information system. Designing databases offers unique challenges. Designing a good relational database requires an understanding of entity-relationship diagrams, normalization, and structured query language (SQL), etc.

In addition to various design models, the systems engineering approach to information systems design also requires in-depth understanding of human factors, tradeoff analysis, and cost estimation. Human factors have recently become an important topic in the design of information systems. GUI applications are often developed under the supervision of human factors experts, who review window layouts, proofread help and tutorial texts, and conduct usability studies. These

human factors specialists ensure that the end products not only conform to industry standards, but are also user-friendly.

To design a quality system that is efficient, reliable, maintainable, and yet cost-effective, a number of tradeoff analyses need to be performed. Tradeoff analysis methods relate system design parameters to performance parameters to access the effectiveness of the system. In addition, cost estimations are often performed to access the cost effectiveness of the system.

4.2 The systems approach to information systems design

A system is often defined as a combination of elements that perform a specific function. Systems may be viewed in terms of their subsystems, where each subsystem performs a small set of functions, and in turn combines to perform larger system level functions. Systems may also be viewed in terms of their relationships with outside environment, where some stimuli from the environment acts like inputs to the system, and triggers some system response or outputs. Systems may also be classified as natural or manmade, physical or conceptual, static or dynamic, and closed or open.

No matter how systems are defined, designing quality systems in the information age requires the systems approach. The systems approach dictates that a system is designed for its whole life cycle. The system life cycle starts with a

definition of needs, ends with system disposal, and in between goes through requirements, design, development, test, and utilization. The following figure shows a high level overview of the life cycle approach to systems development:

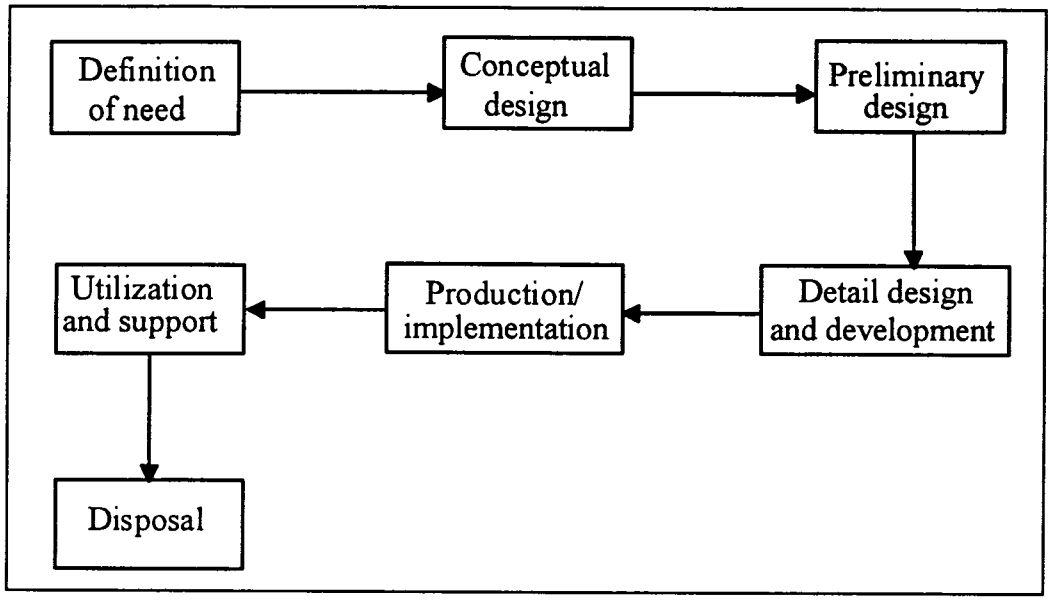

Figure 4.1 The Systems Approach

The systems design process is often not straight forward. An important consideration is the feedback loop. At the end of each design step, evaluations and adjustments are performed before continuing. In order to make good choices, systems designers need to consider the following factors:

- Alternatives and tradeoffs
- Economic evaluations and feasibility

- Optimization
- Process control
- System reliability
- System maintainability
- Human factors

The systems approach is followed in the design and development of the information system prototype. The following table lists the first four steps in the systems approach along with the functions that are perform for the information system prototype:

Steps	Functions
Definition of needs	User requirements
Conceptual design	Requirements analysis High level specification
Preliminary design	Breakdown of subsystem Subsystem functional requirements Detail specification
Detail design and development	Subsystem functional design Prototype development

Table 4.1 Functions performed for the information system prototype

4.3 The waterfall model of software development

The waterfall model of software development became highly influential in software development processes in the 1970s. It emphasizes stage-wise software

development with feedback loops. It later became a standard in most government software procurements. An important aspect of the waterfall model is that certain documentation must be produced for each step in the process. The waterfall model has eight steps with feedback loops between adjacent steps. The following figure shows a picture of the waterfall model:

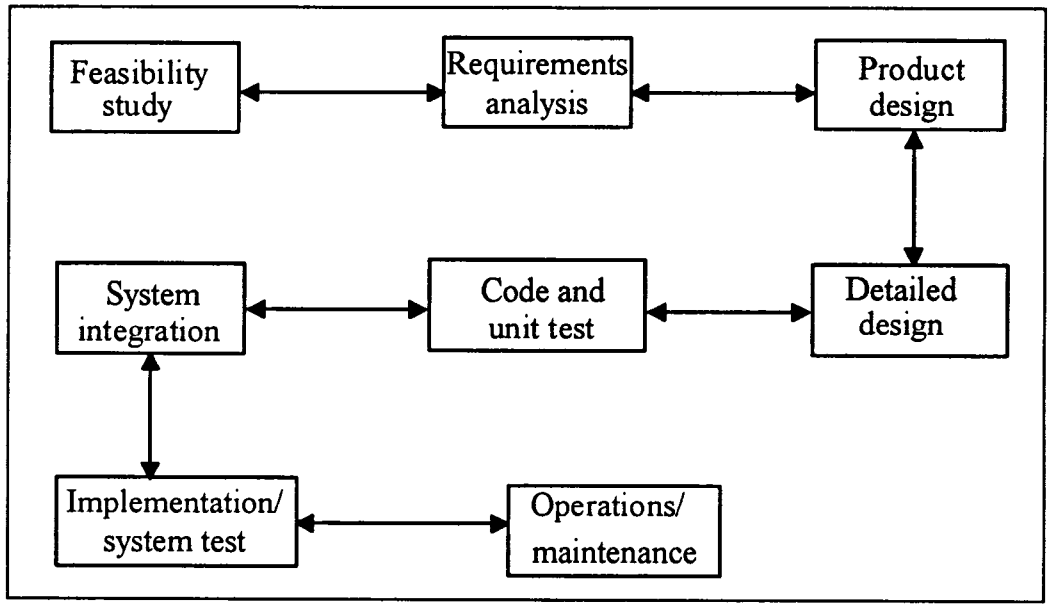

Figure 4.2 The Waterfall Model

The waterfall model requires that certain documentation be produced during the software development cycle. For the ISO 9000 information system prototype, the following list of documentation is produced:

- Requirements documentation

- Design documentation
- Program source code

4.4 The spiral model of software development

Since the waterfall model is document driven, it is not suited for certain types of software development projects. The limitations of the waterfall model become apparent in interactive, Microsoft Windows or OS/2 based applications. Software development under these environments requires rapid prototyping with reusable code, and not necessarily elaborate documentation.

Some have argued that software development should not be document driven, but rather risk driven. Instead of showing off documentation at the end of each stage, risk analyses should be performed to determine whether to proceed further. The spiral model of software development is thus based on prototyping and risk analysis, which make it better suited for developing GUI applications. The spiral model involves essentially the same steps as previous approaches except the following two major differences:

- Risk analyses are performed after major steps
- Prototyping is incorporated into the model

Since a GUI based application is designed and developed in this thesis work, the spiral model is better suited for this purpose than the waterfall model.

To minimize the risks involved, a prototype of the ISO 9000 information system is developed. Future work will include risk analyses and further prototyping before actual implementation begins.

4.5 Box-structured design of information systems

Software development is more than just trial-and-error. The introduction of structured programming demonstrated that program correctness can be mathematically proven. Boxed structured design of information systems is a recent attempt at developing software that has low rate of errors by using mathematical analyses.

According to box structured analysis, any information system can be viewed as a black box, a state box, or a clear box at different stages of development. During requirements gathering, a system is often viewed as a black box, which defines data abstraction in terms of external behaviors. All inputs and outputs of the system are gathered during this stage. The following is a picture of a black box:

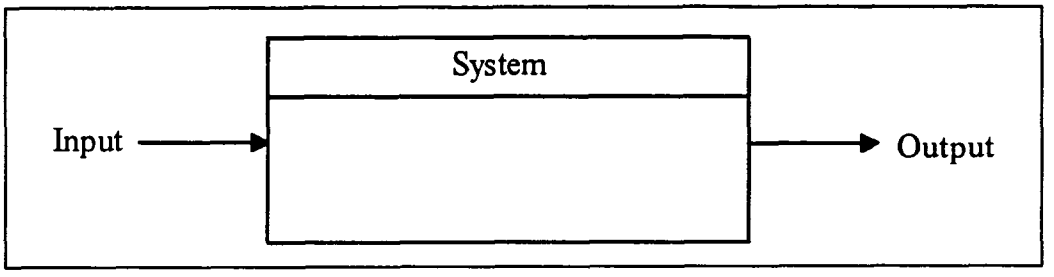

Figure 4.3 A Black Box

Next is the state box, which offers another level of abstraction by utilizing states. In this stage, systems are designed in terms of how data stored in memory is changed by each input. The following is a picture of a state box:

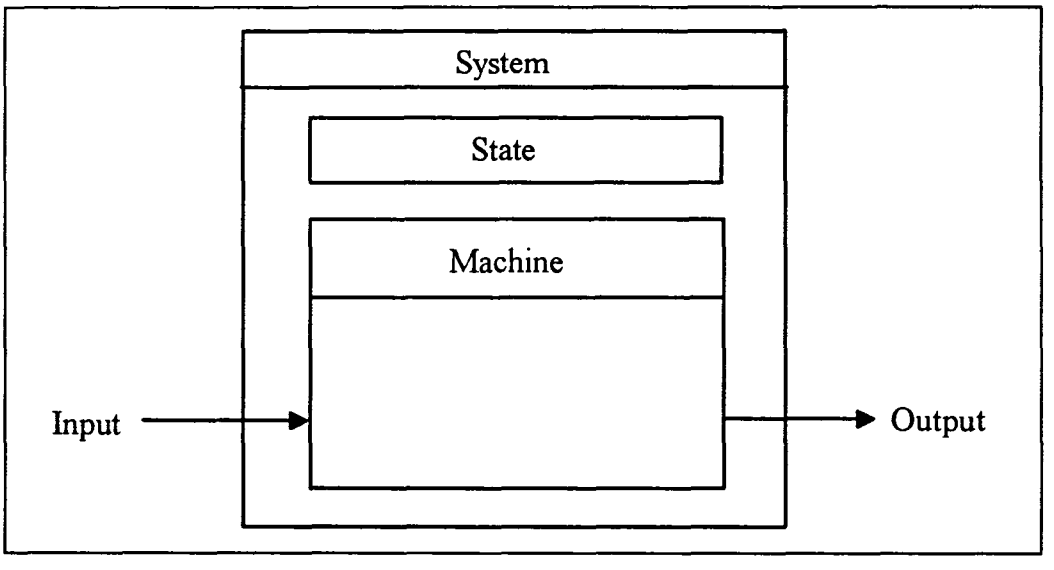

Figure 4.4 A State Box

The clear box is the last step of the transformation, where procedurality is introduced. In this stage, how each input is transformed to each output is designed. Conditions, loops, and concurrence, etc. are incorporated into the clear box. The following is a picture of a clear box:

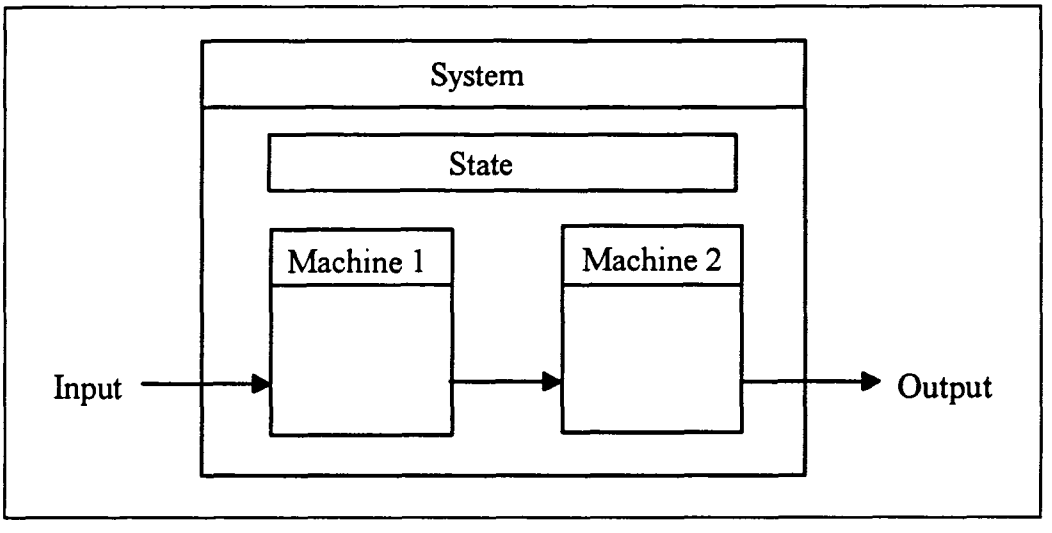

Figure 4.5 A Clear Box

Good information system designs start with black boxes, then go through state boxes, and end with clear boxes. Transformations between the three stages must be done mathematically. Software systems designed using this rigorous box structured method can be mathematically proven to be correct.

Using box structured approach, a high-level design of the ISO 9000 information system prototype can be given. At the black box level, input is defined

as information entered by users, and outputs are information displayed on the screen and reports generated. At the state box level, state is defined as information stored on disk or in memory. At the clear box level, the information system can be broken down into several subsystems -- data entry, data display, data storage, data retrieval, and report generation. To combine similar functions, three subsystems are defined for the purpose of this thesis work -- data entry facility, central repository, and report generation facility.

4.6 The clean-room approach to systems development

The clean-room approach attempts to capture the essence of the systems approach, the spiral model and the box structured design. This methodology is still at early stages of development. It was used in a few small scale software development efforts at various companies including IBM, and the results were promising. This model emphasizes incremental development, correctness verification and feedback. The clean-room approach stresses certification, documentation, and statistical testing, all of which are also emphasized by ISO 9000.

4.7 Object-oriented design of information systems

With the introduction of graphical user interface (GUI), object-oriented design and implementation of information systems have become a hot topic. GUI represents information as icons on the screen, and users choose objects and perform actions on them. Object-oriented design attempts to simulate real world situations, where people often think of objects first and then act on them.

Object-oriented design and implementation are based on the concepts of classes and objects. A class defines a type of objects; it is an abstract data type that describes interactions between the class of objects and their outside environment. On the other hand, an object is an instance of its class; it holds values which may be modified. Classes necessitate hierarchies. The class structure of an object-oriented system is typically a tree structure, with superclasses and subclasses.

The four major advantages of object-oriented design are data encapsulation, inheritance, dynamic binding and polymorphism. Data encapsulation refers to the fact that each object is a black box, whose behavior may only be altered by sending it messages that it understands. The internal data of the object is protected. Inheritance refers to the fact that objects of subclasses inherit all the object behaviors of their parent classes. The advantage of this is reusable code. Dynamic binding is that the system waits until run time rather than

compile time to interpret the messages sent to an object. This affects system performance. Polymorphism defines the ability of most object-oriented systems to send the same messages to objects of different classes. Each object in turn reacts in ways defined in its own class. This also facilitates reuse.

Similar to the spiral model, object-oriented methodology also emphasizes prototyping and feedback. The following figure shows a picture of the object-oriented development model:

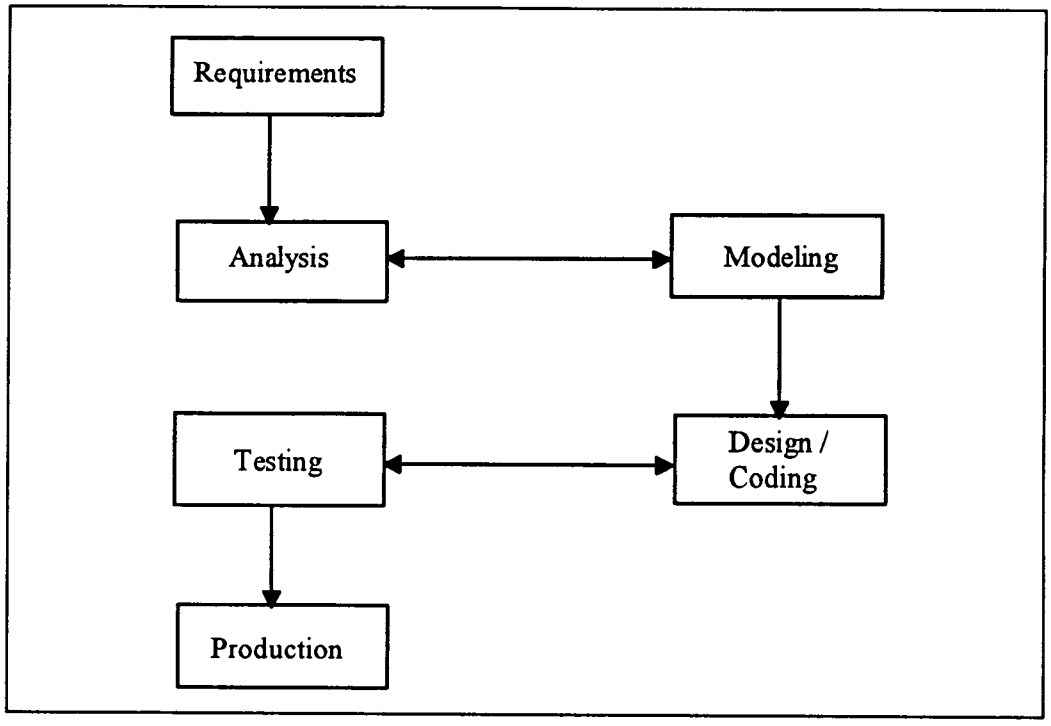

Figure 4.6 Object-oriented Development Model

Since the ISO 9000 information system prototype is a GUI application developed using Visual Basic, the development approach is object-oriented. Controls on windows and dialog boxes are treated as objects, whose actions are programmed. At the system design level, the prototype developed is a part of application modeling, and is used to validate analysis.

4.8 Relational database design

The essence of the relational model is entities and relationships. An entity is a distinguishable object that is represented in the database. Examples of entities include employee, skill, or element. In addition to entities, the relational model includes relationships that link entities together. For example, an employee has a certain set of skills, and a skill may belong to several employees, therefore, "has" and "belongs to" are the relationships between employee and skill. Entities also have properties or attributes. For example, the properties of employee include name, employee number, department, or job title, etc. Which properties of an entity to store in the database depends on user requirements. The relationships between entities may be one-to-one, one-to-many, or many-to-many. For example, the relationship between employee and skill is many-to-many, because each employee can have many skills and each skill can belong to more than one employee. The relationship between company and employee is one-to-many,

because each company has many employees, but each employee normally works for only one company. The relationships between entities may change depending on the circumstances. For example, if a certain company has only one employee, then the relationship between company and employee becomes one-to-one.

Entity-relationship diagrams are used to represent conceptual views of databases, and relational database tables can be constructed based on these diagrams. Normally, each entity has its own table, then additional tables are created linking entities. The number of tables varies based on the complexity of relationships between entities. Tables are arranged horizontally by fields and vertically by records. Each table has keys with which users can access information stored in the table.

After all database tables are designed and built, some means are needed to store and retrieve data. Most relational database products on the market today support structured query language (SQL). Developed by IBM Corporation, SQL is the most widely used database access language. It is powerful, yet English-like and easy to learn. It provides all the necessary table, field and record operations for relational databases.

4.9 User interface design

4.9.1 Definition of user interface

User interface is often defined as a bridge between the machine and the human that is used to facilitate the encoding and decoding of information. For the purpose of a computer information system, user interface is often viewed as a data entry facility where users enter all relevant information and perform specific tasks. It consists of a series of windows and dialog boxes upon which users may type free texts, select options and choose actions. A good user interface guides users through these windows in an intuitive fashion, provides enough controls such as radio buttons or check boxes to minimize the amount of typing required, has consistent action buttons, provides customizable fonts and colors, and has help facilities where needed.

4.9.2 Human factors

With the increasing popularity of computers, information systems must be designed to be used by virtually anyone. This places a lot of responsibilities on software designers, who must design applications to be "user-friendly." Fortunately, software designers can benefit from decades of studies done by psychologists.

Much of user interface design is based on human factors, which is the application of relevant information about human capabilities and behavior to the design of systems that people use. Many of the bases for human factors studies result from experiments done by psychologists. Three of the relevant topics of these experiments are sensory modality, coding, and visual display.

Human beings have a number of senses -- visual, auditory, and tactual, etc. Of particular importance in the context of user interface design are visual and auditory sensory modalities. On computer screens, information is usually presented visually, through texts and graphics. However, there are recent studies which suggest that software designers should further utilize the auditory sensory modality using different combinations of music-like tones. In choosing which sensory modality to use, designers need to consider things such as the type of messages, the desired responses, and the users' working environment. The following table shows results from studies done by psychologists and human factors experts regarding when to use the auditory or visual forms of presentation:

Use auditory presentation if:	Use Visual presentation if:
The message is simple.	The message is complex.
The message is short.	The message is long.
The message will not be referred to later.	The message will be referred to later.
The message deals with events in time.	The message deals with location in space.
The visual system of the person is overburdened.	The auditory system of the person is overburdened.
The receiving location is too bright or dark-adaptation integrity is necessary.	The receiving location is too noisy.
The person's job requires moving about continually.	The person's job allows him or her to remain in one position.

Table 4.2 Comparison Of Auditory And Visual Presentations

Most displays present information in coded forms rather than their direct representations or reproductions. Commonly used codes include traffic signs, blips on radar screens, hazard signs, sirens, or icons used in GUI applications. A good coding system has the following characteristics:

- Detectability -- Codes must be seen or heard under the anticipated environment conditions. For example, if the environment is dark, then good lighting may be needed around hazard sign. If the environment is noisy, then sirens must be loud and use a different pitch than the background noise.
- Discriminability -- Every code symbol must be discriminable from other symbols. Studies show that people can identity only 7 ± 2 different codes on

an absolute basis. Therefore, when auditory codes are designed, for example, the ranges between tones need to be spread out.

- Meaningfulness -- Codes must be meaningful to the user so that he or she can easily remember them. A good example of this is a traffic sign, which is meaningful to most people. Another example is an icon used in GUI applications that is intuitive.
- Standardization -- Standardization of codes also facilitates learning and retention.
- Multidimensional codes -- Use of multidimensional codes can increase discriminability. A good example of this is a police cruiser with sirens and turning colored lights, which uses both the visual and auditory sensory modalities. Other examples include the use of both shape and color in hazard signs.
- Compatibility -- Codes must be compatible with the user. For example, use aircraft symbols on a map to denote airports, or arrange knobs in the same way displays are arranged.

Screen design is particularly important in software development. Users must be able to see and understand what is on the screen with ease. For most of the VGA displays currently in existence, texts with font sizes of between 9 and 12 are adequate. Screen density is another factor to consider. Screens should have

adequate white spaces so that they do not appear "too busy". A good design uses rows and columns to group information if applicable; it also uses charts and graphs whenever possible to minimize the amount of reading necessary. With the introduction of GUI, standardized symbols are available to effectively code information.

4.9.3 Windows and dialog boxes

Windows in GUI present views on objects. A typical window often has a title bar at the top, a system menu on top left corner, minimize/maximize buttons on top right corner, a menu bar below the title bar, a presentation space to hold controls that convey information, and a frame that surrounds the window. Windows may be classified as primary or secondary. Secondary windows, sometimes called child windows, are clipped by the parent or primary window. Closing the primary window causes all its secondary windows to be closed. Windows may also be classified as modal or modeless. A modal window keeps the focus and does not allow users to interact with other windows until it is closed. On the other hand, a modeless window does not keep the focus, and users may interact with other windows at will.

Dialog boxes are similar to windows, except that they usually do not have menu bars and are not sizable. Dialog boxes may also be modal or modeless like windows.

There are a number of standard controls that are often used in the presentation space of a window or a dialog box. Windows, dialog boxes, and controls are currently standardized by big companies in the computer industry, including IBM and Microsoft. The following is a list of commonly used standard controls, most of which are used in the data entry facility of the ISO 9000 information system prototype:

- Static text -- This control is used to display labels.
- Entry field -- Users may type free text into entry field controls. An entry field may be single line, or multiple line with scroll bars.
- Radio button -- This control is used when a selection needs to be made for a small set of values.
- Check box -- This control is also used for selection, but there can be only two, sometimes three, choices.
- List box -- This control is used when selections need to be made for a large and variable set of values. A list box control may be single selection or multiple selection. List boxes usually have scroll bars attached.

- Push button -- This control is sometimes call action buttons. It is used when users tell the computer to perform some actions. Typical push buttons include OK, Cancel, Help, Add, Delete, etc.
- Combination boxes -- A combination box control may be used when the designer want to combine several controls into one. Typical combination boxes include drop down entry fields, drop down lists, and spin buttons.
- Group box -- A group box may be used when the designer wants to group certain controls on a window for effective presentation. A typical use of a group box is around a set of radio buttons.
- Picture -- A picture control is used to preserve space for graphics or bitmaps.
- Container -- With object-oriented implementations, container controls are introduced to hold icon objects, and to facilitate drag-and-drop actions.
- Notebook -- A notebook control is a collection of dialog boxes. It has the appearance of a notebook with tabs. Users may click on tabs to go to a specific dialog box or notebook page.

4.9.4 Message boxes

Message boxes are small dialog boxes used to remind users to perform certain actions, or to warn users of certain error conditions. Message boxes are

standardized so that programmers don't need to code for all the controls on them. Programmers usually access message boxes by calling a predefined function using a message box number. Message boxes fall into four categories -- error, warning, query and information, each with its predefined icons and push buttons. There may be variations on the icons and push buttons used, but they serve the same purpose. The following is a list of the four types of message boxes, all of which are used in the data entry facility of the ISO 9000 information system prototype:

- Error -- This box is used when the user performs some action incorrectly, and the program cannot continue. It usually comes with a stop sign icon and push buttons OK, Cancel, or Retry, Cancel. It may also come with a beep to catch the user's attention.
- Warning -- This box is used when the user performs some action that may cause problems later, but the program can still continue at the time of the warning. It comes with an exclamation mark icon and push buttons OK, Cancel. It may also come with a beep.
- Query -- This box is often used to ask the user to confirm some action they requested. It comes with a question mark icon and push buttons Yes, No. It may also come with a beep.

- Information -- This box is used to present some information or as a reminder. It comes with a letter "i" icon and push button OK. This box usually does not have a beep associated with it.

4.9.5 Fonts and colors

The introduction of color monitors and GUI have offered software designers and users vast opportunities in terms of software fonts and colors. The word-processing software that is used to generate this thesis paper offers 68 different fonts, from Arial to WingDings. Font sizes range from 4 to 72. Most VGA monitors support 256 colors, which means designers can design software that lets users customize their windows' background and foreground using any combination of the 256 colors.

All this computer power does not mean that designers can choose fonts and colors according to their own preferences. On the contrary, this places greater responsibilities on designers to consider human factors. Typically, popular fonts such as Helvetica, Times, Chicago, and Courier are good choices, whereas Script or WingDings are often not acceptable. Font sizes between 9 and 12 are good for most applications. A neutral color is always a good choice. OS/2, for example, comes with a soft gray color. If possible, applications should give users capability to choose fonts and colors they like.

4.9.6 Help facility and tutorial

A good help facility is an important aspect of a good application. It sometimes takes as much time and effort to write the help facility as the application itself. Help facilities usually come with the following elements:

- Using help -- This is the help for help. It shows users how to use the help facility.
- Help index -- This gives users an index of the help facility. Users may choose a topic to view and study.
- Help contents -- This gives users a table of contents for the help facility. Users may choose a part to view and study.
- Search -- This gives users a way to search for certain key words in the help facility.

Help facilities may be programmed using a technique know as hypertext. Using this technique, users may go to one part of the help facility, highlight certain keywords they want to get more information on, and click on those keywords to go to another part of the help facility.

Another important part in assisting users in using the application is to develop a tutorial. The tutorial is often developed as a separate application. It guides users through the application using simple examples.

4.10 Tradeoff analysis

4.10.1 System effectiveness

Since World War II, mathematical and statistical concepts have been applied to the evaluation of system effectiveness. System effectiveness measures how well a system achieves its objective under resource constraints. System effectiveness, presented as a probability, is the product of system readiness, system reliability and system design adequacy, that is, $P_{se} = P_{sr} P_r P_{da}$. System readiness, also called availability, is a measure of whether the system is available and ready to use when needed. System reliability, also called dependability is a measure of whether the system will perform as designed when used. System design adequacy, also called capability, is a measure of whether the system will achieve the mission objective.

To evaluate the effectiveness of a system, each of its elements must be evaluated separately. Many of the tradeoff issues come into play when evaluating availability, dependability and capability. For example, cost may play an important role in system capability.

4.10.2 Cost effectiveness

To make any system cost effective, cost estimations must be performed. The methodologies for cost estimations are well established. They invariably include establishing a work breakdown structure, and then estimating labor hours, labor rate, material count, and material unit price for each item in the work breakdown structure. Overhead costs are subsequently added to arrive at the total cost of the system.

Just as cost often affects system effectiveness, system effectiveness also affects system cost. Systems with more capability and higher dependability often cost more to build, because better parts and more skilled people are involved.

4.10.3 Tradeoff analysis methods

Tradeoff analysis is not guessing, every choice made must be based on quantifiable arguments. To perform a tradeoff analysis, alternatives and their selection criteria must be defined and evaluated using decision making tools.

A number of mathematical, statistical and economical tools have been proven to be useful in performing tradeoff analyses. The following is a partial list of these tools:

- Decision evaluation function
- Decision evaluation matrix

- Break-even economic evaluations
- Optimization theories
- Probabilities
- Queuing theories
- Process control theories

4.10.4 Tradeoff analysis of information systems

A good tradeoff analysis relates each system design element to each system performance parameter and tries to quantify every element. For the ISO 9000 information systems prototype, elements such as user, hardware and software platform, application speed, learning curves are considered. The following table shows high-level subsystems and performance parameters for the information system. Subsystems are shown vertically on the left and performance parameters are show horizontally on the top. A "Yes" is shown in the box wherever the corresponding subsystem design parameter has an impact on the performance parameter. For the purpose of this analysis, hardware and operating systems used are considered part of the overall system because they have major impact on the performance of the prototype.

	Application functionality	Application quality	Application speed	User-friendliness	Learning curve	Application maintainability	Application portability	Development cycle
1. Hardware								
1.1 CPU			Yes				Yes	Yes
1.2 Memory	Yes		Yes					
1.3 Fixed drive	Yes		Yes					
2. Software								
2.1 Operat. system	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
2.3 Program. language	Yes	Yes	Yes			Yes		Yes
3. ISO 9000 application								
3.1 Database	Yes	Yes	Yes			Yes	Yes	Yes
3.2 User interface	Yes		Yes	Yes	Yes	Yes	Yes	Yes
4. User				Yes	Yes			
5. Developer		Yes				Yes		Yes

Table 4.3 Software Performance vs. Subsystem Design Parameters

Chapter 5 Design And Development Of The Information System For ISO 9000 Quality Documentation

5.1 Requirements analysis

Using box structured design approach, the ISO 9000 information system can be defined as having the following inputs:

- Company information including employees, skills, and resources.
- ISO 9000 elements information.
- Company specific ISO 9000 procedure information including verification methods, responsible employees, skills required, and resources required.

The system also has the following outputs:

- Reports generated on information entered into the system, i.e., the quality manual.
- Information that may be viewed on the screen.

In addition, users have requested a GUI application to be run on a typical personal computer configured with DOS and Microsoft Windows.

5.2 Design

5.2.1 Conceptual design

Requirements analysis has revealed that the ISO 9000 information system must have the following elements:

- A central repository
- A user-friendly data entry facility
- A report generation facility

The central repository must be flexible enough to store several kinds of information such as employee information, skill information, and company resource information. In addition, generic ISO 9000 guidelines and company specific ISO 9000 procedures must also be stored. The report generation facility must be able to generate reports on information stored in the central repository. For the data entry facility, the standard Microsoft Windows user interface is to be used based on user preference.

5.2.2 Central repository preliminary and detailed design

The central repository is designed to be a relational database that answers the following questions:

- What does ISO 9000 recommend?
- What needs to be done to meet that recommendation?

- Whose responsibility is it?
- What skills are required?
- What methods are used for verification?
- What resources are needed?

To answer these questions, the relational database is designed to have the following elements or entities:

- ISO 9000 Element or sub-element
- Company specific procedure
- Employee
- Skill
- Verification method
- Resource
- Company

The following entity-relationship diagram shows the relationships between these entities.

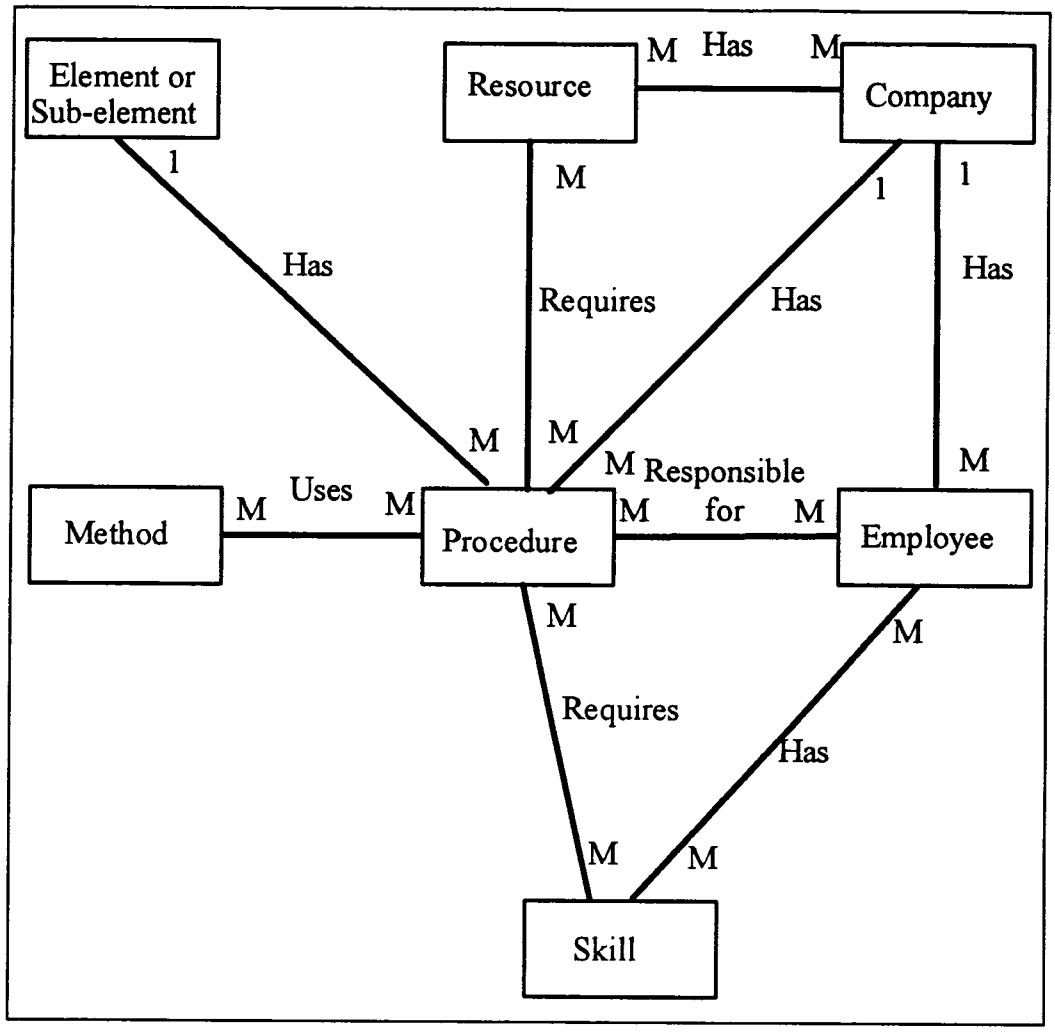

Figure 5.1 Entity-Relationship Diagram

Several points are illustrate by the information presented in the chart.

- Each element or sub-element can have more than one company specific procedures, but each procedure is governed by only one element or sub-element.

- Each company has many procedures, but each procedure belongs to only one company.
- For any combination of element and company, there is only one procedure, that is, each company has one procedure for every ISO 9000 element or sub-element.
- Each procedure may be 1 or more employees' responsibility, and each employee may be responsible for more than one procedure.
- Each procedure may require many types of skills, such as control charts or sampling, and each of these skills applies to more than one procedure.
- Each employee may possess many skills, and more than one employee may have the same skill.
- Each company has a set of resources, and each resource may belong to several companies.
- Each procedure may be verified using several methods, and each method may apply to several procedures.

Since Compression Telecommunications Corporation (CTEL) is the only company involved at this time, the entity-relationship diagram may be simplified to the one shown in the following figure.

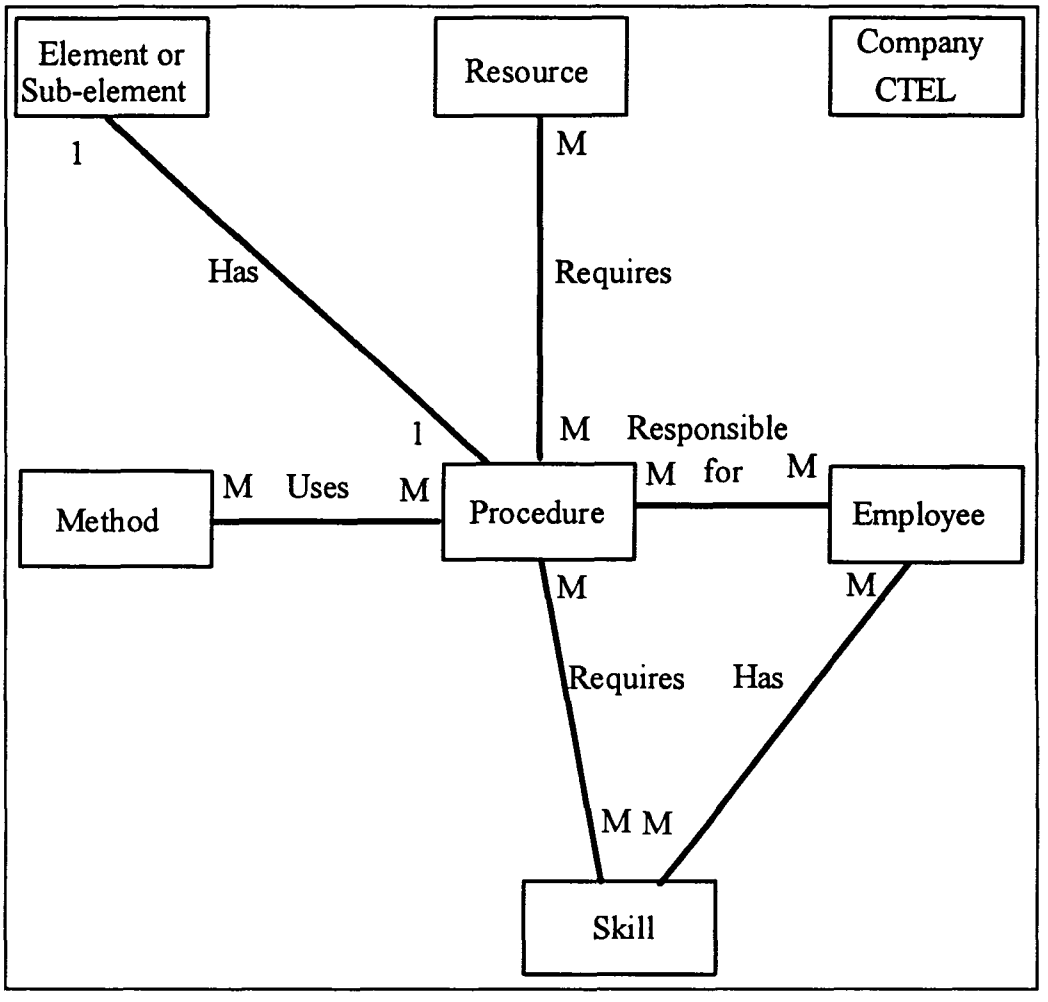

Figure 5.2 Simplified Entity-Relationship Diagram

In the above diagram, company is a table by itself used to store information such as CTEL's address, phone number, etc. Since only one company is involved, the relationship between element and procedure has been reduced to one to one, with each element or sub-element dealing with only one company specific

procedure. In addition, the relationship between company and resource has been eliminated with the understanding that all resources belong to CTEL.

Using the design specified in the above entity-relationship diagram, detailed relational database tables can be constructed. The following are the 11 relational database tables:

- Company table
- Element-procedure table
- Employee table
- Method table
- Resource table
- Skill table
- Procedure-Employee table
- Procedure-Method table
- Procedure-Resource table
- Procedure-Skill table
- Employee-Skill table

The following chart shows the details for each of the 11 tables, and the definitions for each field in the tables.

Table name	Field name	Field detail
Company	Company name	String (50)
	Company address	String (50)
	City	String (20)
	State	String (2)
	Zip code	String (5) - must be numeric
	Area code	String (3) - must be numeric
	Phone number	String (3) - must be numeric
	Phone extension	String (4) - must be numeric
Element - Procedure	Procedure number	Integer - key
	Element number	Integer - between 1 and 20
	Element sub-number 1	Integer - must be >= 1
	Element sub-number 2	Integer - must be >= 1
	Element name	String (50)
	Element description	String (300)
	Procedure text	String (500)
Employee	Employee number	Integer - key
	Last name	String (20)
	First name	String (15)
	Middle initial	String (1)
	Suffix	String (3)
	Title	String (4)
Method	Method number	Integer - key
	Method name	String (50)
	Method description	String (300)
Resource	Resource number	Integer - key
	Resource name	String (50)
	Resource description	String (300)
Skill	Skill number	Integer - key
	Skill name	String (50)

	Skill description	String (300)
Procedure - Employee	Procedure number	Integer - key
	Employee number	Integer - key
Procedure - Method	Procedure number	Integer - key
	Method number	Integer - key
Procedure - Resource	Procedure number	Integer - key
	Resource number	Integer - key
Procedure - Skill	Procedure number	Integer - key
	Skill number	Integer - key
Employee - Skill	Employee number	Integer - key
	Skill number	Integer - key

Table 5.1 Database Tables And Fields

5.2.3 User interface preliminary and detailed design

The prototype has a primary window and a number of secondary windows and dialog boxes. Each window can be considered as a view on some objects in the central repository. The following are the major windows and dialog boxes:

- Primary window lets users open or close files, open secondary windows and dialog boxes, generate reports, and access the help facility.
- Company dialog box lets users enter company information including name, address, and phone number.
- Element or sub-element and procedure dialog box let users enter element number, element name, element description and procedure text. It also provides access to a lower level dialog box for users to select methods used,

resources needed, skills required and employees responsible for that particular element and procedure. A list of elements in the central repository is displayed alongside to facilitate easy browsing and editing.

- Method dialog box lets users enter method name and method description. A list of methods is also displayed so that users may select and update any method information.
- Resource dialog box lets users enter resource name and resource description. A list of resources is also displayed so that users may select and update any resource information.
- Skill dialog box lets users enter skill name and skill description. A list of skills is also displayed so that users may select and update any skill information.
- Employee dialog box lets users enter employee information such as last name, first name, title, etc. It also provides access to a lower level dialog box where users may associate skills with employees. A list of employees currently in the central repository is also displayed so that users may easily select and update any employee information.
- The process quality manual dialog box lets users view all relevant information in the central repository, and choose to produce a quality manual for the company, which is CTEL in this case.

In order for the application to have a consistent look and feel, the following design rules are used for windows and dialog boxes:

- Every window and dialog box has a title bar with a system menu, including the maximize and minimize buttons.
- The main window has a menu bar with the usual File and Help menu items in addition to Edit and Report.
- Every dialog box has standard push buttons - OK, Cancel and Help or Close and Help.
- The OK push button performs the standard function of saving information and closing the dialog box.
- The Cancel or Close push buttons perform the standard function of ignoring changes made since the last time a push button is pressed, and closing the dialog box.
- The Help push button performs the standard function of bringing up help texts.

Another issue to consider is the naming conventions used for controls on windows and dialog boxes. Naming convention deals with the variable names that are assigned to each control. These variable names are used in programs to associate actions with appropriate controls. For the prototype, the following popular naming conventions are used:

- Variable names for all controls have a prefix followed by the control's name, e.g., stEmployee.
- For variable names, each word in a control's name is initial capped with spaces removed and no hyphens or underscores, e.g., lbSkillList.

The following table shows all types of controls used in the prototype and their naming conventions:

Control name	Other name	Prefix	Example
Static text	Label	st	stSkillName
Entry field	Text field	ef	efSkillName
List box	Selection list	lb	lbSkillSet
Group box	Frame	gb	gbSkillList
Combo box	Drop down list	cb	cbTitle
Push button	Action button	pb	pbOk
Menu	Menu drop down	mn	mnFile
Sub-menu	Menu item	smn	smnExit
Picture box	Bitmaps	bmp	bmpProduct
Form	Window or dialog box	fm	fmSkill

Table 5.2 Window Controls' Naming Conventions

The following pages show window layouts along with the detailed design of each window and dialog box. Emphases are placed on each window's push buttons and associated actions. Details for most of the static fields and group boxes are omitted, since they are merely labels with no actions involved.

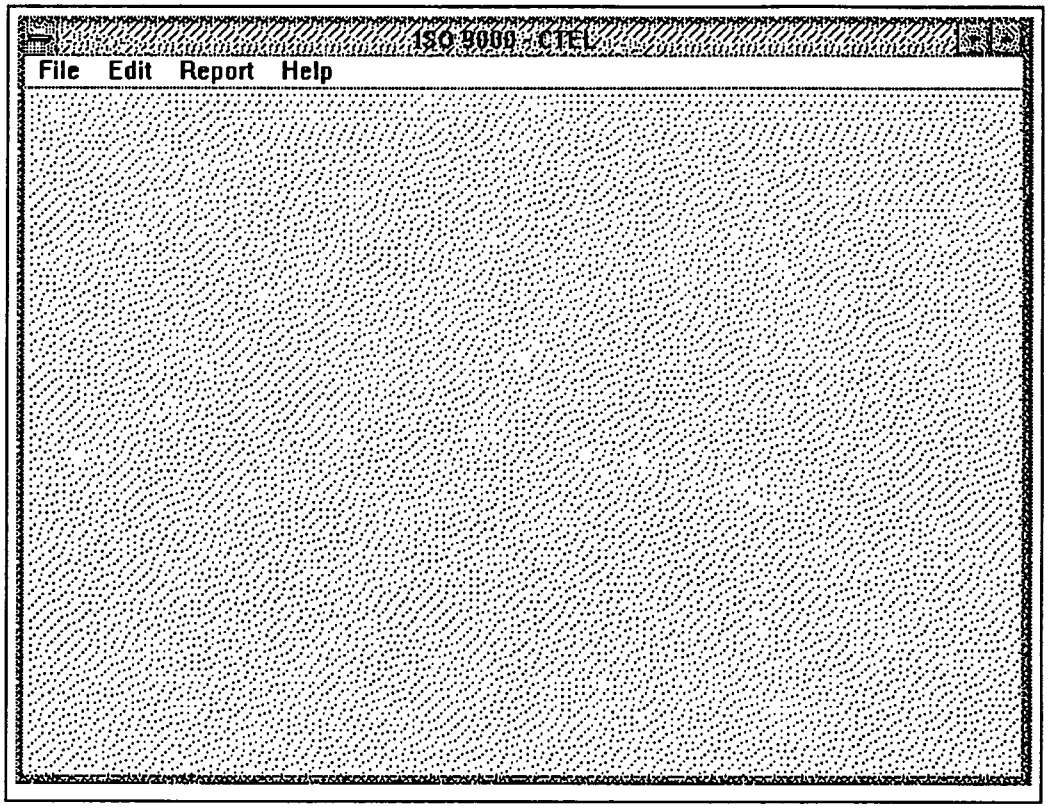

Figure 5.3 Main Window

- The main window has a menu bar with menu items "File", "Edit", "Report" and "Help".
- The "File" menu item has sub-menus "New", "Open", "Delete", and "Exit".
- When File-New is selected, open the "New file" dialog box.
- When File-Open or File-Delete is selected, open the "File list" dialog box.
- When File-Exit is selected, make sure all other windows are closed and shut down the application.

- The "Edit" menu item has "Company", "Element", "Method", "Resource", "Skill", and "Employee" sub-menu items. Selecting these sub-menu items opens up the appropriate dialog box, e.g., "Company" dialog box, "Skill" dialog box.
- The "Report" menu item has "Quality manual" sub-menu item. Selecting the "Quality manual" sub-menu item opens up the "Process quality manual" dialog box.
- The "Help" menu-item has "Contents", "Using help", Help Index, and "Product information" sub-menu items. Selecting these sub-menu items brings up the appropriate information.

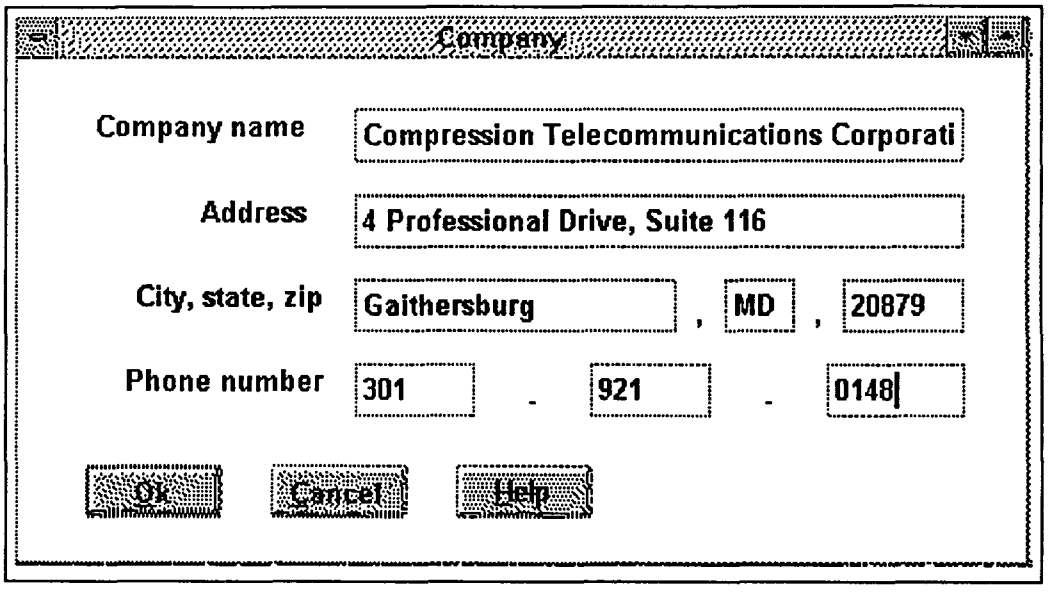

Figure 5.4 Company Dialog Box

- When window opens up, display appropriate company (CTEL) information.
- Selecting "OK" push button causes any changes made to be saved.
- Selecting "Cancel" causes any changes made to be discarded.

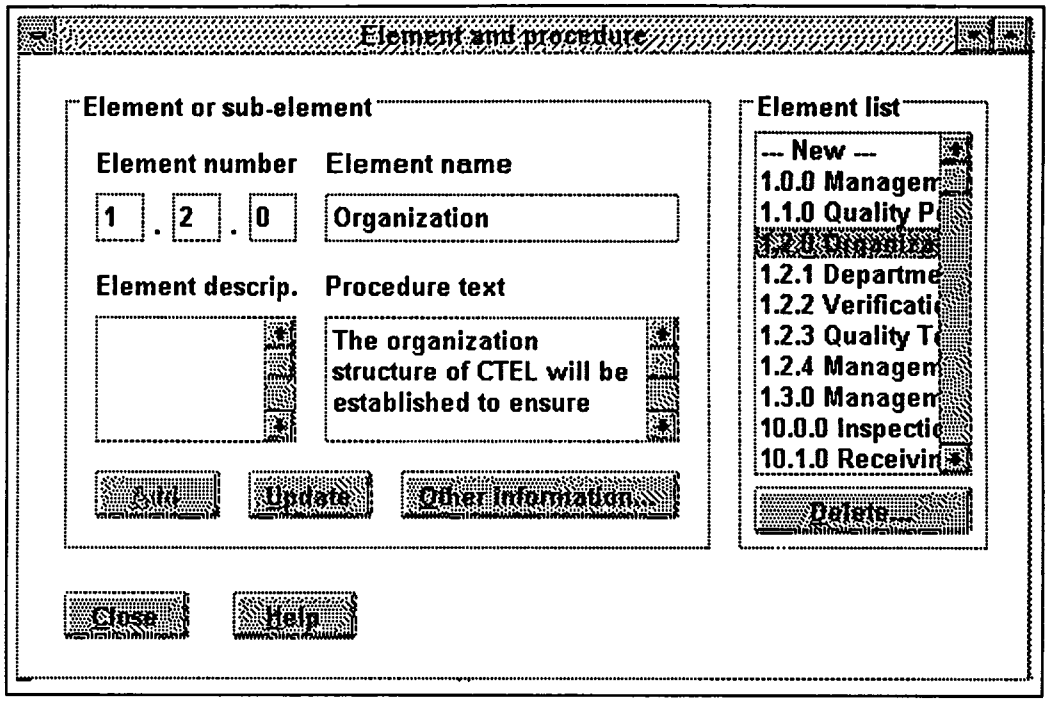

Figure 5.5 Element And Procedure Dialog Box

- When window opens up, the list on the right contains all elements saved previously. The item labeled "New" is selected in the list box on the right, and entry fields on the left are blank. Push buttons "Update" and "Delete" are disabled.

- When an item other than "New" is selected in the list box, display the appropriate information for that item in entry fields on the left. Enable push buttons "Update" and "Delete", disable push button "Add".
- When "Add" or "Update" push buttons are clicked, save changes made in entry fields, and refresh the list on the right to reflect the changes.

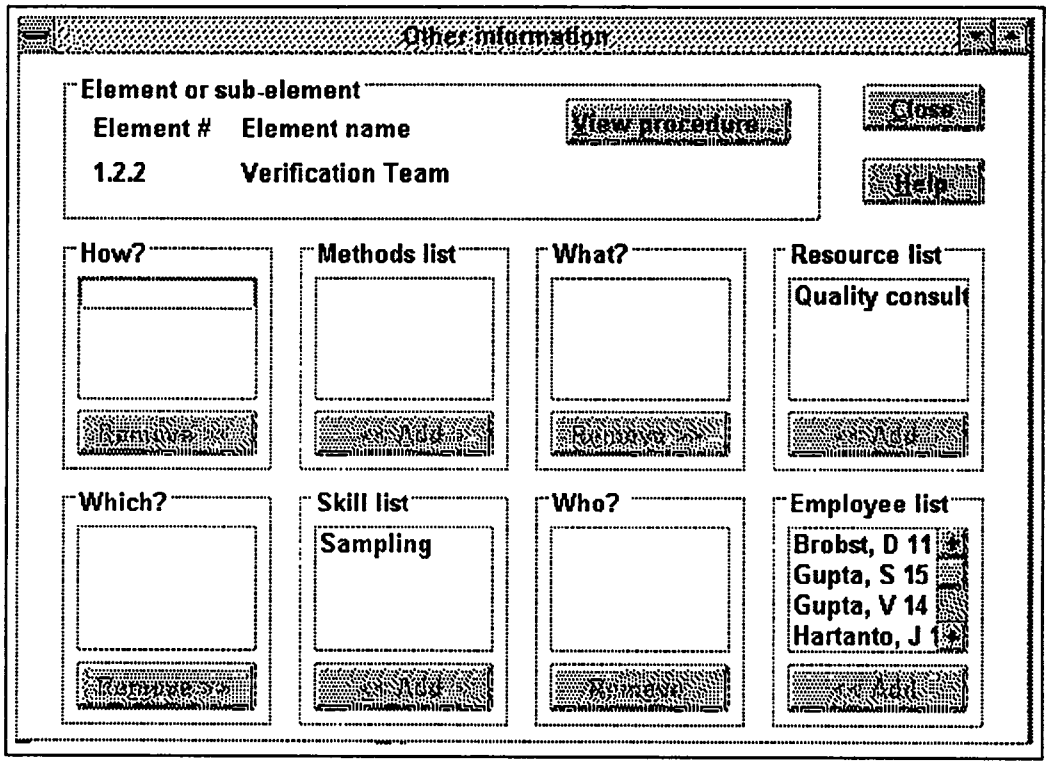

Figure 5.6 Procedure - Other Information Dialog Box

- When the window opens up, list boxes lbSet1, lbSet2, lbSet3 and lbSet4 contain all methods, resources, skills and responsible employees associated

with the element, and lbList1, lbList2, lbList3 and lbList4 contain all methods, resources, skills and employees not associated with the element but are available.

- When push button "Remove" is clicked, remove selected items from the particular lbSet and place them into the lbList to the right. That is, disassociate those items from the element and remove them from the appropriate database tables, e.g., procedure-method table.
- When push button "Add" is clicked, remove selected items from the particular lbList and place them into the lbSet to the left. That is, associate those items with the element and save them into the appropriate database tables, e.g., procedure-method table.
- When push button "View" procedure is clicked, open up the "View procedure" dialog box.

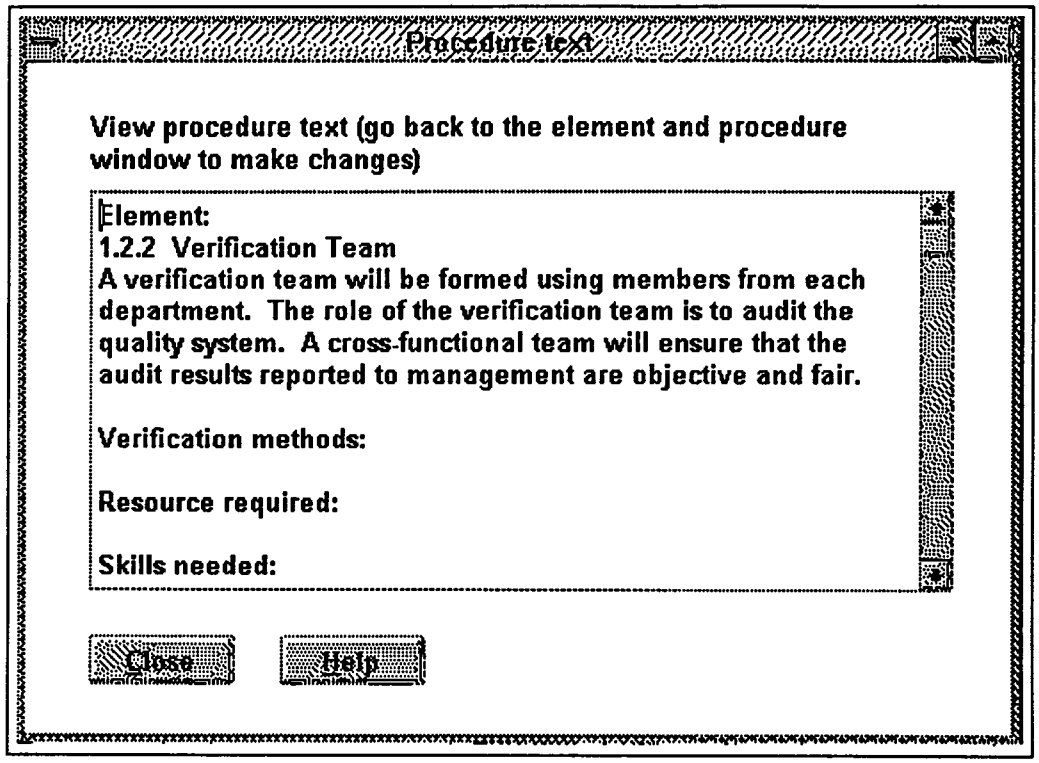

Figure 5.7 View Procedure Text Dialog Box

- When the dialog box opens up, display information for the particular element in the entry field. This information includes procedure text, methods used, resources required, skills required, and employees responsible.

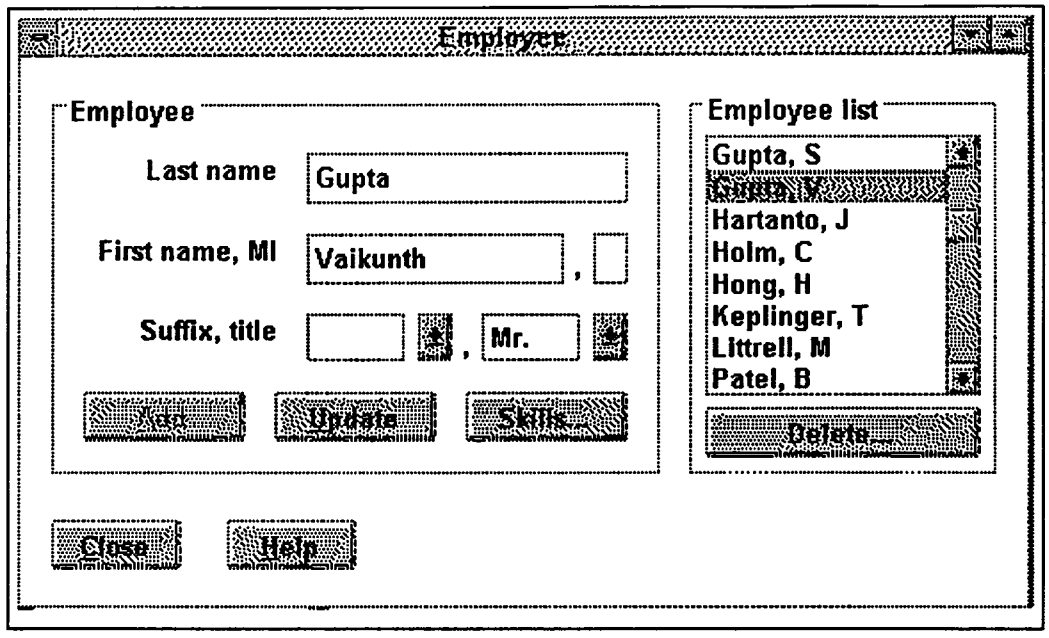

Figure 5.8 Employee Dialog Box

- Controls on this dialog box behave the same way as those in the Element-procedure dialog box.
- The list on the right contains all employee in the company. In this case, the employees of CTEL are displayed.

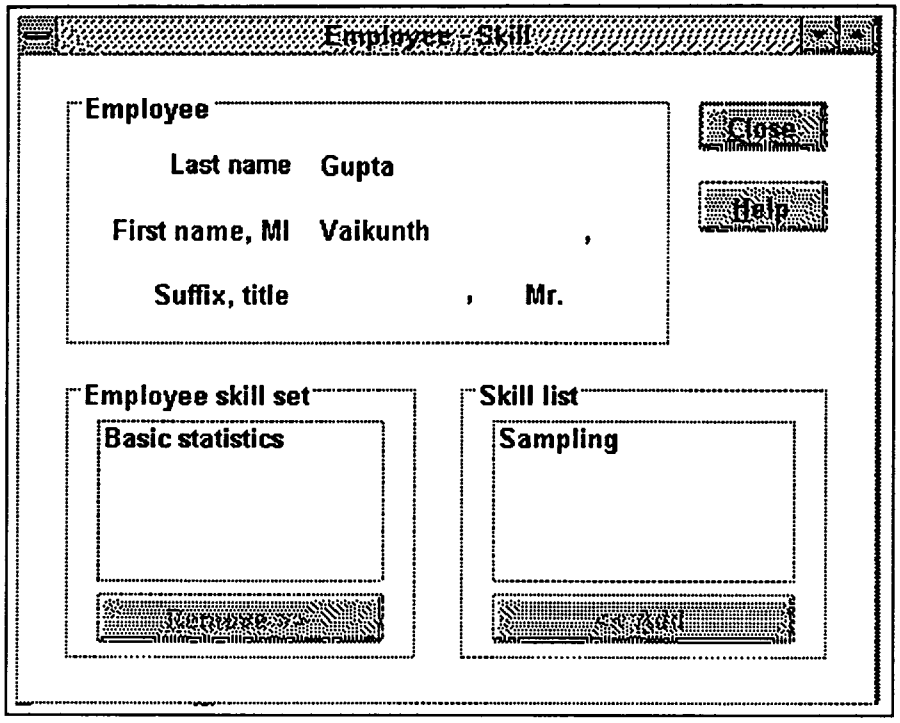

Figure 5.9 Employee - Skill Dialog Box

- Controls on this dialog box behave the same way as those in the Procedure-other dialog box.
- List box lbSet contains all skills the employee possesses, and list box lbList contains available skills.

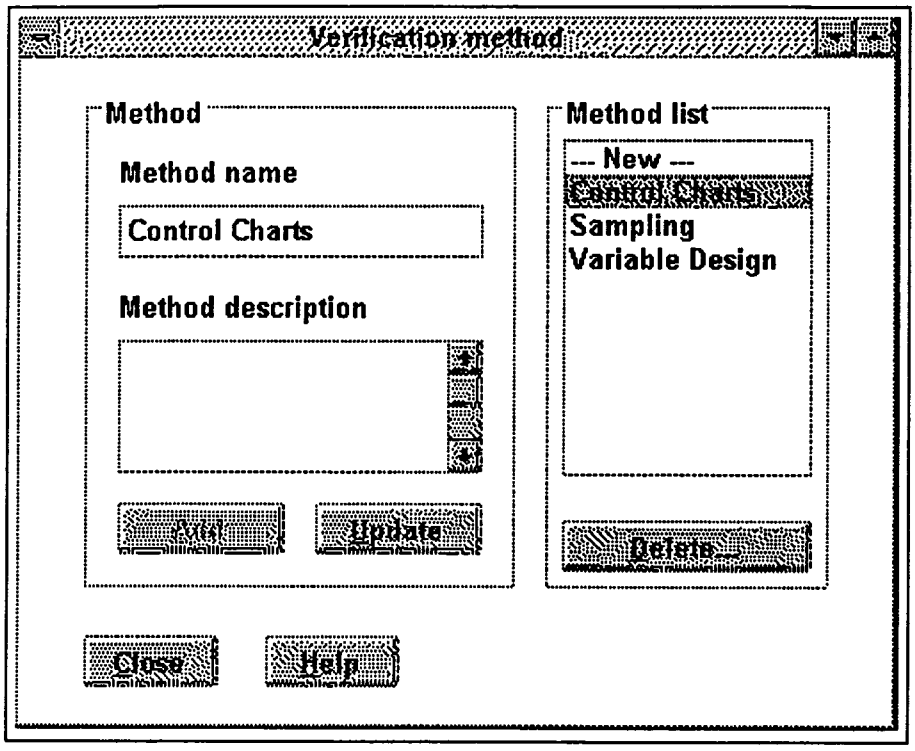

Figure 5.10 Method Dialog Box

- Controls on this dialog box behave the same way as those in the Element-procedure dialog box.
- The list on the right contains all methods entered previously.

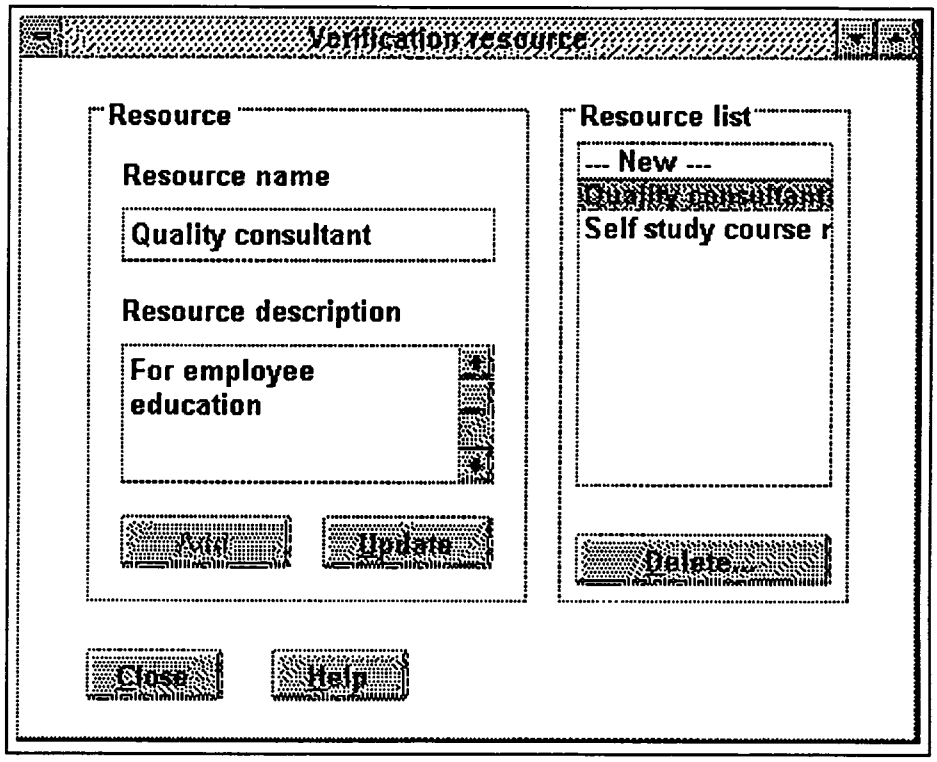

Figure 5.11 Resource Dialog Box

- Controls on this dialog box behave the same way as those in the Element-procedure dialog box.
- The list on the right contains all resources entered previously.

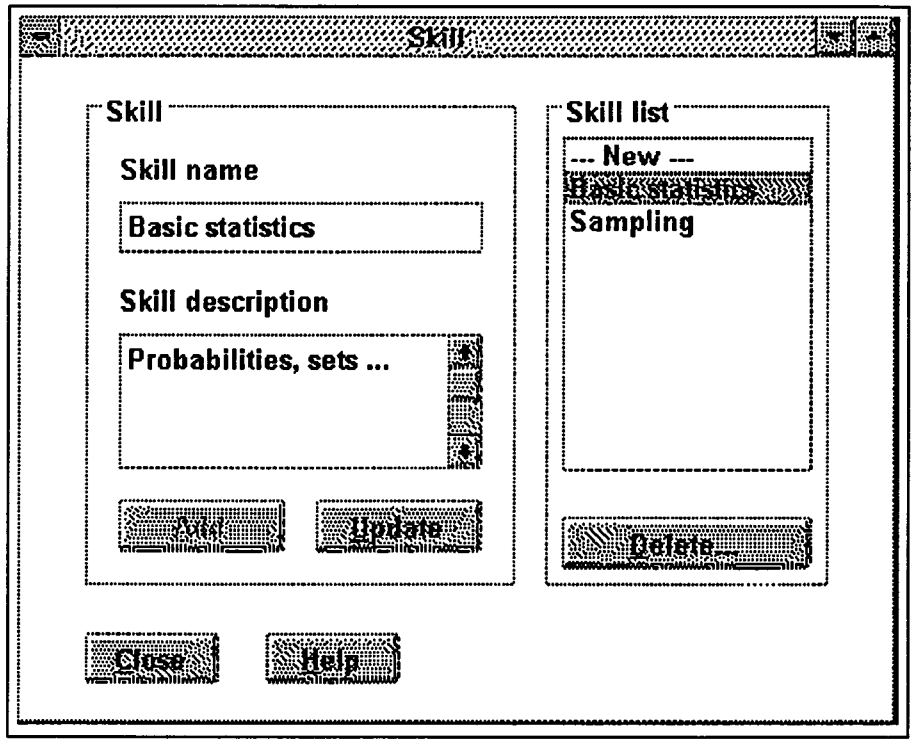

Figure 5.12 Skill Dialog Box

- Controls on this dialog box behave the same way as those in the Element-procedure dialog box.
- The list on the right contains all skills entered previously.

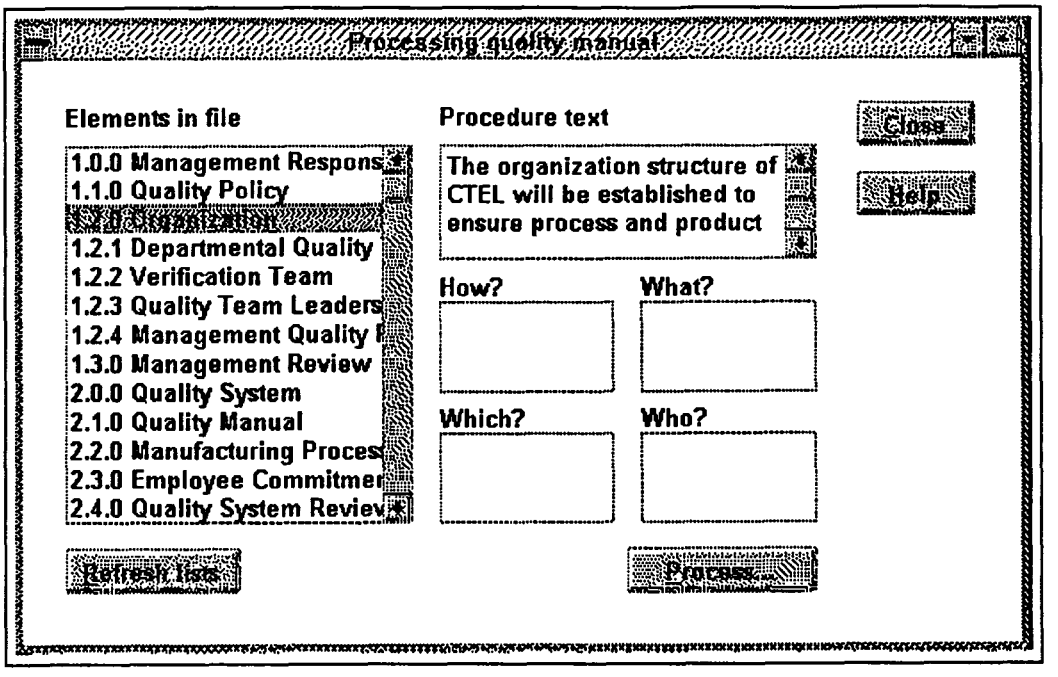

Figure 5.13 Process Quality Manual Dialog Box

- When the dialog box opens up, display all elements saved previously in the list box on the left.
- When an item is selected in the list box, display appropriate information for the element on the right.
- When push button "Refresh list" is clicked, refresh the list on the left
- When push button "Process" is clicked, loop through all elements in the list box on the left and write all information to an ASCII text file.

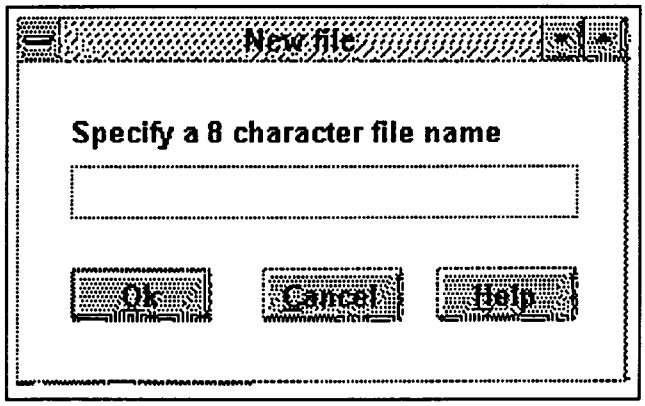

Figure 5.14 New File Dialog Box

- When push button OK is selected, create a new sub-directory under C:\ISO9000 for the specified company. Editing will be done to ensure that sub-directory names are valid.
- Cancel push button closes the dialog box.

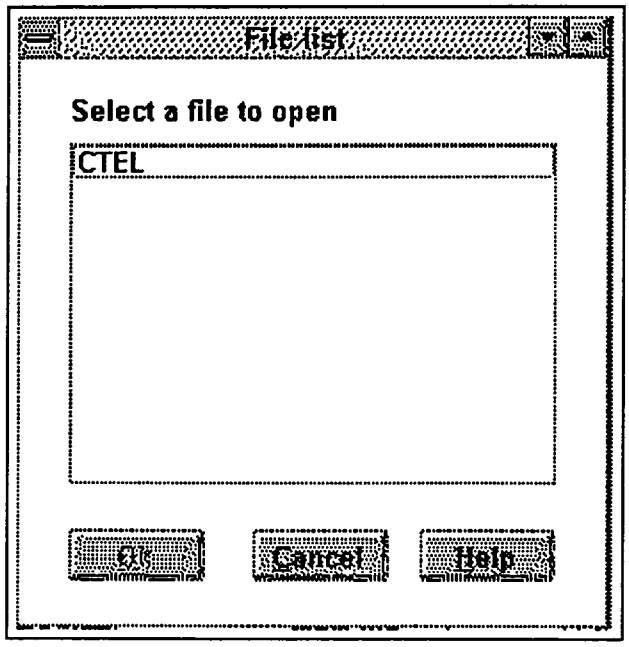

Figure 5.15 File Selection Dialog Box

- The list box contains all sub-directory names under C:\ISO9000, where central repositories reside. Information for different companies are stored in separate directories.
- When push button OK is clicked, either load or delete information for the selected company depend on whether File-open or File-delete has been selected previously on the main window.
- The "Cancel" push button closes the window.

Standard message boxes are used in the information system prototype. They are used in the following situations:

- Query message boxes are used when the user chooses to delete something or shut down the application.
- Error message boxes are used when there is a file access error.
- Information message boxes are used to inform the user that a file has been generated.

The font used in the prototype is Arial with a size of 9.25. Default colors, which is black text on white background, are used for secondary windows and dialog boxes. The primary window inherits the Microsoft Windows color scheme that the user has set previously.

A small help facility consisted of window level help is designed to be used with future prototypes. The following is a list of help texts for each window:

- Main window -- Use the File menu pull-down to open and delete files, or to exit the application. Use the Edit menu pull-down to enter or modify company, element, method, resource, skill and employee information. Use the Report menu pull-down to create quality manual for the company.
- Company dialog box -- Enter company information on this dialog box. Select OK to save the information in the database, or select Cancel to discard the changes made.
- Element and procedure dialog box -- Enter element and procedure information on this dialog box. Use the list on the right to select an element

to edit. Select New in the list box to add an element. Select Delete to delete the selected element from the list. Select Other information to bring up the Element-other dialog box. Select Close to close the dialog box.

- Employee dialog box - Enter employee information on this dialog box. Use the list on the right to select an employee to edit. Select New in the list box to add an employee. Select Delete to delete the selected employee from the list. Select Close to close the dialog box.
- Method dialog box -- Enter method information on this dialog box. Use the list on the right to select a method to edit. Select New in the list box to add a method. Select Delete to delete the selected method from the list. Select Close to close the dialog box.
- Resource dialog box -- Enter resource information on this dialog box. Use the list on the right to select a resource to edit. Select New in the list box to add a resource. Select Delete to delete the selected resource from the list. Select Close to close the dialog box.
- Skill dialog box -- Enter skill information on this dialog box. Use the list on the right to select a skill to edit. Select New in the list box to add a skill. Select Delete to delete the selected skill from the list. Select Close to close the dialog box.

- Procedure-other dialog box -- Use this dialog box to associate methods, resources, skills, and employees with the current procedure. Select Add or Remove to add or remove items from appropriate lists. Select View procedure to bring up the View procedure text dialog box. Select Close to close the dialog box.
- Employee-skill dialog box -- Use this dialog box to associate skills with the current employee. Select Add or Remove to add or remove items from appropriate lists. Select Close to close the dialog box.
- View procedure dialog box -- Use this dialog box to view text for the current procedure. Changes made on this dialog box will not be saved.
- Process quality manual dialog box -- Use this dialog box to check information for all procedures before creating the ASCII text file. Use the list on the left to select and view items. Select Refresh to refresh the list. Select Process to create the quality manual for the company. Select Close to close the dialog box.
- File new dialog box -- Enter a sub-directory name in the entry field. Select OK to create the sub-directory for a new company. Select Cancel to disregard the request.
- File list dialog box -- Select a company in the list to open or delete. Select OK to process the request. Select Cancel to disregard the request.

5.2.4 Report generation facility design

A quality manual for Compression Telecommunications Corporation (CTEL) is generated as a part of this thesis work. The quality manual consists of the twenty ISO 9000 elements. It is created as an ASCII text file using the ISO 9000 information system prototype. Users may use an editor or a word processor to view, modify, add graphics, and print. The quality manual that is attached to this thesis report is developed using a combination of the prototype and a popular word processor.

5.3 Prototype development

5.3.1 Development platform

Since one of the requirements is that the final product must be run on a commonly available personal computer equipped with DOS and Microsoft Windows, a development environment that is especially designed for this platform is used. Among the available choices are Microsoft Visual Basic, Borland C++ with Microsoft Windows support, and Smalltalk/V for Windows. Some tradeoff analyses are done based on factors such as platform compatibility, development time, and execution speed, etc. Microsoft Visual Basic is chosen as the implementation platform for the prototype as a result of these analyses.

Microsoft Visual Basic has its foundation in Basic, which is the most popular programming language. In addition, it introduces a new concept in software development termed visual programming, which lets programmers create GUI applications with ease. It has been suggested that with GUI applications, 90 percent of the programs written deal with user interface, and only 10 percent go to the meat of the applications. Visual programming enables programmers to dramatically decrease the time spent creating user interface. Microsoft Visual Basic is also object-oriented in terms of window handling. Each window control is viewed as an object, and programmers code actions for these objects.

5.3.2 Development detail

The actual source codes of Visual Basic programs are included in the appendix.

Chapter 6 Conclusion And Recommendations

6.1 Conclusion

ISO 9000 is a set of generic standards for quality management and assurance. It applies to all products and services, because it defines requirements for a system to manage quality during and after product development. Due to its broad scope, ISO 9000 registration requires elaborate documentation. A well-designed information system utilizing recent technologies such as graphical user interface (GUI) and relational database can be used to facilitate the task of producing quality documentation for ISO 9000 certification.

A prototype of such an information system has been developed in this thesis work to demonstrate the effectiveness of using the systems engineering approach to produce a quality manual. It employs Microsoft Visual Basic on the popular DOS/Microsoft Windows platform. Several unique features of the implemented information system prototype are:

- System architecture, which structures data entry, data storage and output according to the framework of the ISO 9000 standards.
- Microsoft Windows and Visual Basic development platform, which makes the prototype not only easy to use but also suitable for small companies such as Compression Telecommunications Corporation (CTEL).

- Relational database approach, which offers easy maintenance and expansion of data storage for present and future needs.

The utilization of the information system prototype has enabled a systematic and rapid creation of the quality manual for Compression Telecommunications Corporation (CTEL).

6.2 Recommendations

The information system has been specifically designed for creating quality documentation for ISO 9000 certification. Several enhancements may be needed to further expand this thesis work. These enhancements may transform the developed prototype to a more useful and marketable tool. The following is a list of possible future enhancements:

- Utilize a full relational database with structured query language (SQL) capabilities, or even consider using an object-oriented database for large companies.
- Use a more powerful programming language such as C++ to utilize full object-oriented implementations.
- Enhance the user interface with customizable fonts and color, a tutorial, and hypertext, context sensitive and field level help.

- Give users the capability to import forms and organization charts into the information system, and include them as part of the reports.

These enhancements may require substantial studies on usability and cost estimation. Issues such as learning curve, installation, and user training must also be considered.

Bibliography

1. Arter, D. R. "Demystifying the ISO 9000 / Q90 Series Standards" Quality Progresss, November 1992

2. Blanchard, B. S. and Fabrycky, W. J. Systems Engineering and Analysis Prentice Hall, New Jersey, 1990

3. Boehm, B. W. "A Spiral Model of Software Development and Enhancement" IEEE Computer, May 1988

4. Buckler, G. "ISO Designation Shows Quality A Priority" Computing Canada September 1, 1993

5. Canter, S. "Microsoft Visual Basic For Windows, Version 3.0, Professional Edition" PC Magazine November 9, 1993

6. Date, C. J. Database Systems Addison-Wesley Publishing Company, Massachusetts, 1991

7. Dichter, C. "Software Audits: How Good Really?" UNIX Review January, 1994

8. Digitalk Smalltalk/VPM Tutorial and Programming Handbook Digitalt, Los Angeles, 1989

9. Editors "Database Applications" LAN Times August 16, 1993

10. Editors "Database Management Systems" Database Programming & Design October 15, 1993

11. Editors "Microcomputer DBMSs And Application Development Systems" DBMS June 15, 1993

12. Elliot, S. "Management Of quality In Computing Systems Education: ISO 9000 Series Quality Standards Applied" Journal of Systems Management September, 1993

13. Garver, R. "What Are The ISO Series Standards" Industrial Engineering September, 1993

14. Habayeb, A. R. Systems Effectiveness Pergamon Press, Oxford, 1987

15. Haverson, D. "Shedding Light On ISO 9000" MIDRANGE Systems August 10, 1993

16. Haverson, D. "The Road To World Class" MIDRANGE Systems September 14, 1993

17. Hevner, A. R. and Mills, H. D. Box-Structured Requirements Determination Methods 1992

18. IBM Common User Access Advanced Interface Design Reference IBM Corporation, North Carolina, 1991

19. Inwood, C. "Developers Still Lagging In ISO Preparation" Computing Canada August 16, 1993

20. McKague, A. "The Danger Zone" Computing Canada October 25, 1993

21. Mills, H. D., Linger, R. C. and Hevner, A. R. "Box Structured Information" Systems IBM Systems Journal, Vol 26, No 4, 1987

22. Microsoft Corporation Microsoft Visual C++ Development System For Windows User's Guide Microsoft Corporation, 1993

23. National ISO 9000 Support Group "The Price of Quality" ComputerWorld, June 1993

24. Orvis, W. J. Visual Basic for Windows SAMS Publishing, Indiana, 1992

25. Ottey, A. "Keys To Quality" MIDRANGE Systems July 13, 1993

26. Peach, R. W. The ISO 9000 Handbook CEEM Information Services, Virginia, 1992

27. Peters, T. J. and Waterman, R. H., Jr. In Search of Excellence Lessions from America's Best-Run Companies Warner Books, New York, 1982

28. Pratt, T. W. Programming Languages Design and Implementation Prentice-Hall, Inc., New Jersey, 1984

29. Ricciuti, M "Visual Tools Give COBOL A New Look" Datamation, November 15, 1993

30. Rosen, C. "ISO Auditor Tells How To Pass An ISO 9000 Audit" Electronic Business Buyer October, 1993

31. Rothery, B. ISO 9000 Gower Press, Hampshire, 1993

32. Sanders, M. S. and McCormick, E. J. Human Factors In Engineering And Design McGraw-Hill, Inc., New York, 1993

33. Saracelli, K. and Bandat, K. "Process Automation In Software Application Development" IBM Systems Journal September, 1993

34. Sinha, M. N. and Willborn, W. O. The Management of Quality Assurance John Wiley & Sons, New York, 1985

35. Stallings, W. Computer Organization and Architecture Principles of Structure and Function Mcmilliam Publishing Company, New York, 1990

36. Stewart, R. D. Cost Estimating John Wiley & Sons, Inc, New York, 1991

Appendix Visual Basic Source Code For The Prototype

A.1 Global

```
Declarations

'window status

' false closed

' true open

Global WINfmCompany As Integer

Global WINfmElement As Integer

Global WINfmEmployee As Integer

Global WINfmSkill As Integer

Global WINfmResource As Integer

Global WINfmMethod As Integer

' return code from window

' true ok is clicked

' false cancel is clicked

Global RCfmNewFile As Integer

Global RCfmFileList As Integer

Global RCfmOther As Integer

Global RCfmEmpSkl As Integer

Global FileAction As String
```

```
Global CurrentFile As String
Global CurrentEmployee As Integer
Global CurrentProcedure As Integer
Global ProcedureText As String
Global Const BaseDirectory = "C:\ISO9000"
Global Const NewItem = "--- New ---"
Type CompanyRecType    ' 137
   Name As String * 50
   Address As String * 50
   City As String * 20
   State As String * 2
   Zip As String * 5
   AreaCode As String * 3
   Number As String * 3
   Extension As String * 4
End Type
Global CompanyRec As CompanyRecType
Type ElementRecType   '856
   Number As Integer
   Number1 As Integer
   Number2 As Integer
```

```
    Name As String * 50
    Description As String * 300
    Procedure As String * 500
End Type
Global ElementRec As ElementRecType
Type EmployeeRecType        ' 43
    Last As String * 20
    First As String * 15
    MI As String * 1
    Suffix As String * 3
    Title As String * 4
End Type
Global EmployeeRec As EmployeeRecType
Type SkillRecType     ' 350
    Name As String * 50
    Description As String * 300
End Type
Global SkillRec As SkillRecType
Type ResourceRecType     ' 350
    Name As String * 50
    Description As String * 300
```

```
End Type
Global ResourceRec As ResourceRecType
Type MethodRecType    ' 350
  Name As String * 50
  Description As String * 300
End Type
Global MethodRec As MethodRecType
Type ProcedureSkillRecType   ' 4
  ElementIndex As Integer
  SkillIndex As Integer
End Type
Global ProcedureSkillRec As ProcedureSkillRecType
Type ProcedureResourceRecType   ' 4
  ElementIndex As Integer
  ResourceIndex As Integer
End Type
Global ProcedureResourceRec As ProcedureResourceRecType
Type ProcedureMethodRecType   ' 4
  ElementIndex As Integer
  MethodIndex As Integer
End Type
```

```
Global ProcedureMethodRec As ProcedureMethodRecType
Type EmployeeSkillRecType     ' 22
   EmployeeIndex As Integer
   SkillIndex As Integer
End Type
Global EmployeeSkillRec As EmployeeSkillRecType
Type ProcedureEmployeeRecType   ' 4
   ElementIndex As Integer
   EmployeeIndex As Integer
End Type
Global ProcedureEmployeeRec As ProcedureEmployeeRecType
Sub InitFlags ()
   WINfmCompany = False
   WINfmElement = False
   WINfmEmployee = False
   WINfmSkill = False
   WINfmResource = False
   WINfmMethod = False
   WINfmManul = False
   RCfmNewFile = False
   RCfmFileList = False
```

```
    RCfmEleEmp = False
    RCfmEleSkl = False
    RCfmEmployee = False
    RCfmEmpSkl = False
End Sub
Sub OpenFiles ()
    Open "COMPANY.DAT" For Random As #1 Len = 137
    Open "ELEMPROC.DAT" For Random As #2 Len = 856
    Open "EMPLOYEE.DAT" For Random As #3 Len = 43
    Open "METHOD.DAT" For Random As #4 Len = 350
    Open "SKILL.DAT" For Random As #5 Len = 350
    Open "RESOURCE.DAT" For Random As #6 Len = 350
    Open "PROCSKIL.DAT" For Random As #9 Len = 4
    Open "EMPLSKIL.DAT" For Random As #10 Len = 4
    Open "PROCEMPL.DAT" For Random As #11 Len = 4
    Open "PROCRESO.DAT" For Random As #12 Len = 4
    Open "PROCMETH.DAT" For Random As #13 Len = 4
End Sub
```

A.2 Main window

```
Function saveFile () As Integer
```

```
   If WINfmCompany = True Or WINfmElement = True Or WINfmEmployee =
True Or WINfmSkill = True Or WINfmMethod = True Or WINfmResource =
True Or WINfmInspection = True Then
      UserResponse% = MsgBox("There is a file open.  Either save it by clicking
Ok  on all open dialog boxes or  Cancel  out.", 16, "Error")
      If WINfmCompany = True Then
         fmCompany.Show
      End If
      If WINfmElement = True Then
         fmElement.Show
      End If
      If WINfmEmployee = True Then
         fmEmployee.Show
      End If
      If WINfmSkill = True Then
         fmSkill.Show
      End If
      If WINfmMethod = True Then
         fmMethod.Show
      End If
      If WINfmResource = True Then
```

```
        fmResource.Show
      End If
      If WINfmManual = True Then
        fmManual.Show
      End If
      saveFile = False
    Else
      Close
      ChDir BaseDirectory
      CurrentFile = ""
      Call InitFlags
      saveFile = True
    End If
End Function
Sub MDIForm_Load ()
    On Error Resume Next
    ChDir BaseDirectory        ' change directory
    If Err = 76 Then           ' if directory does not
      MkDir BaseDirectory      ' exist, make one
      ChDir BaseDirectory
    End If
```

```
    mnEdit.Enabled = False      ' disable menu dropdowns
    mnReport.Enabled = False
    CompanyFlag = False         ' set flags to false for not open
    ElementFlag = False
    EmployeeFlag = False
    SkillFlag = False
    InspectionFlag = False
End Sub
Sub smnCompany_Click ()
    fmCompany.Show
End Sub
Sub smnDelete_Click ()
    If saveFile() = True Then
        Close
        mnEdit.Enabled = False
        mnReport.Enabled = False
        ChDir BaseDirectory
        fmMain.Caption = "ISO 9000 - Main Window"
        FileAction = "D"
        fmFileList.Show 1
    End If
```

```
End Sub
Sub smnElement_Click ()
   fmElement.Show
End Sub
Sub smnEmployee_Click ()
   fmEmployee.Show
End Sub
Sub smnExit_Click ()
   If saveFile() = True Then
      ChDir "C:\"
      Unload fmMain
   End If
End Sub
Sub smnMethod_Click ()
   fmMethod.Show
End Sub
Sub smnNew_Click ()
   On Error Resume Next
   If saveFile() = True Then
      mnEdit.Enabled = False
      mnReport.Enabled = False
```

```
        fmNewFile.Show 1
        If RCfmNewFile = True Then
            mnEdit.Enabled = True
            mnReport.Enabled = True
            fmMain.Caption = "ISO 9000 - " + CurrentFile
            Call smnCompany_Click
        End If
    End If
End Sub
Sub smnOpen_Click ()
    If saveFile() = True Then
        FileAction = "O"
        mnEdit.Enabled = False
        mnReport.Enabled = False
        fmFileList.Show 1
        If RCfmFileList = True Then
            mnEdit.Enabled = True
            mnReport.Enabled = True
            fmMain.Caption = "ISO 9000 - " + CurrentFile
            Call smnCompany_Click
        End If
```

```
    End If
End Sub
Sub smnProductInformation_Click ()
    fmProduct.Show 1
End Sub
Sub smnQualityManual_Click ()
    fmManual.Show
End Sub
Sub smnResource_Click ()
    fmResource.Show
End Sub
Sub smnSkill_Click ()
    fmSkill.Show
End Sub
```

A.3 Company dialog box

```
Function editControls () As Integer
Dim msg As String
    msg = ""
    If Trim$(efName.Text) = "" Then
        msg = msg + "Company name cannot be blanks."
```

```
End If
If Trim$(efAddress.Text) = "" Then
   msg = msg + "  Address cannot be blanks."
End If
If Trim$(efCity.Text) = "" Then
   msg = msg + "  City cannot be blanks."
End If
If Len(Trim$(efState.Text)) <> 2 Then
   msg = msg + "  State code must be 2 characters."
End If
If Not IsNumeric(efZip.Text) Then
   msg = msg + "  Zip code must be numeric."
End If
If Len(Trim$(efZip.Text)) <> 5 Then
   msg = msg + "  Zip code must be 5 digits."
End If
If Not IsNumeric(efAreaCode.Text) Then
   msg = msg + "  Area code must be numeric."
End If
If Len(Trim$(efAreaCode.Text)) <> 3 Then
   msg = msg + "  Area code must be 3 digits."
```

```
    End If
    If Not IsNumeric(efNumber.Text) Then
       msg = msg + "  Number must be numeric."
    End If
    If Len(Trim$(efNumber.Text)) <> 3 Then
       msg = msg + "  Number must be 3 digits."
    End If
    If Not IsNumeric(efExtension.Text) Then
       msg = msg + "  Extension must be numeric."
    End If
    If Len(Trim$(efExtension.Text)) <> 4 Then
       msg = msg + "  Extension must be 4 digits."
    End If
    If msg <> "" Then
       UserResponse% = MsgBox(msg, 16, "Error")
       editControls = False
    Else
       editControls = True
    End If
End Function
```

```
Sub Form_Load ()
   Get #1, 1, CompanyRec
   If Not EOF(1) Then
      efName.Text = CompanyRec.Name
      efAddress.Text = CompanyRec.Address
      efCity.Text = CompanyRec.City
      efState.Text = CompanyRec.State
      efZip.Text = CompanyRec.Zip
      efAreaCode.Text = CompanyRec.AreaCode
      efNumber.Text = CompanyRec.Number
      efExtension.Text = CompanyRec.Extension
   End If
   WINfmCompany = True
End Sub
Sub Form_Unload (Cancel As Integer)
   WINfmCompany = False
End Sub
Sub pbCancel_Click ()
   Unload fmCompany
End Sub
Sub pbOk_Click ()
```

```
    If editControls() = True Then
        CompanyRec.Name = efName.Text
        CompanyRec.Address = efAddress.Text
        CompanyRec.City = efCity.Text
        CompanyRec.State = efState.Text
        CompanyRec.Zip = efZip.Text
        CompanyRec.AreaCode = efAreaCode.Text
        CompanyRec.Number = efNumber.Text
        CompanyRec.Extension = efExtension.Text
        Put #1, 1, CompanyRec
        Unload fmCompany
    End If
End Sub
```

A.4 Element-procedure dialog box

```
Declarations
Dim unUsed() As Integer, maxUnUsed As Integer, maxLoc As Integer, fileInd As
Integer, unUsedInd As Integer, expression  As String, actionSuccessful As Integer
Sub clearEF ()
    efNumber.Text = ""
    efNumber1.Text = ""
```

```
    efNumber2.Text = ""
    efName.Text = ""
    efDescription.Text = ""
    efProcedure.Text = ""
    pbAdd.Enabled = True
    pbUpdate.Enabled = False
    pbDelete.Enabled = False
End Sub
Sub clearRec ()
    ElementRec.Number = 0
    ElementRec.Number1 = 0
    ElementRec.Number2 = 0
    ElementRec.Name = ""
    ElementRec.Description = ""
    ElementRec.Procedure = ""
End Sub
Function editControls () As Integer
Dim msg As String
    If Trim$(efNumber.Text) = "" Then
        efNumber.Text = "0"
    End If
```

```
If Trim$(efNumber1.Text) = "" Then
   efNumber1.Text = "0"
End If
If Trim$(efNumber2.Text) = "" Then
   efNumber2.Text = "0"
End If
msg = ""
If Not IsNumeric(Trim$(efNumber.Text)) Then
   msg = msg + "  Element number must be numeric."
Else
   efNumber.Text = Trim$(Str$(Val(efNumber.Text)))
   If Val(efNumber.Text) <= 0 Or Val(efNumber.Text) > 20 Then
      msg = msg + "  Element number must be between 1 and 20."
   End If
End If
If Not IsNumeric(Trim$(efNumber1.Text)) Then
   msg = msg + "  Sub-element number must be numeric."
Else
   efNumber1.Text = Trim$(Str$(Val(efNumber1.Text)))
End If
If Not IsNumeric(Trim$(efNumber2.Text)) Then
```

```
      msg = msg + "  Sub-sub-element number must be numeric."
   Else
      efNumber2.Text = Trim$(Str$(Val(efNumber2.Text)))
   End If
   If Trim$(efName.Text) = "" Then
      msg = msg + "  Element name cannot be blanks."
   End If
   If msg <> "" Then
      userResponse% = MsgBox(msg, 16, "Error")
      editControls = False
   Else
      editControls = True
   End If
End Function
Function elementExists () As Integer
Dim flag As Integer, userResponse As Integer
   flag = False
   listInd = 0
   While (flag = False) And (listInd < lbList.ListCount)
      expression = efNumber.Text + "." + efNumber1.Text + "." +
efNumber2.Text + " " + Trim$(efName.Text)
```

```
      If expression = Trim$(lbList.List(listInd)) Then
         flag = True
      Else
         listInd = listInd + 1
      End If
   Wend
   If flag = True Then
      userResponse = MsgBox("Element number and name has already been
used.", 16, "Error")
   End If
   elementExists = flag
End Function
Sub refreshlbList ()
   lbList.Clear
   lbList.AddItem (NewItem)
   lbList.ItemData(0) = 0
   fileInd = 1
   unUsedInd = 0
   Get #2, fileInd, ElementRec
   While Not EOF(2)
```

```
        expression = Trim$(Str$(ElementRec.Number)) + "." +
Trim$(Str$(ElementRec.Number1)) + "." + Trim(Str$(ElementRec.Number2)) +
" " + Trim$(ElementRec.Name)
        If expression <> "0.0.0 " Then
           lbList.AddItem (expression)
           lbList.ItemData(lbList.NewIndex) = fileInd
        Else
           unUsedInd = unUsedInd + 1
           ReDim Preserve unUsed(unUsedInd)
           unUsed(unUsedInd) = fileInd
        End If
        fileInd = fileInd + 1
        Get #2, fileInd, ElementRec
    Wend
    maxLoc = fileInd - 1
    maxUnUsed = unUsedInd
    unUsedInd = 1
    lbList.Selected(0) = True
    Call lbList_Click
End Sub
Sub updateEF ()
```

```
    efNumber.Text = Trim$(Str$(ElementRec.Number))
    efNumber1.Text = Trim$(Str$(ElementRec.Number1))
    efNumber2.Text = Trim$(Str$(ElementRec.Number2))
    efName.Text = Trim$(ElementRec.Name)
    efDescription.Text = Trim$(ElementRec.Description)
    efProcedure.Text = Trim$(ElementRec.Procedure)
    pbAdd.Enabled = False
    pbUpdate.Enabled = True
    pbDelete.Enabled = True
End Sub
Sub updateRec ()
    ElementRec.Number = efNumber.Text
    ElementRec.Number1 = Val(efNumber1.Text)
    ElementRec.Number2 = Val(efNumber2.Text)
    ElementRec.Name = Trim$(efName.Text)
    ElementRec.Description = Trim$(efDescription.Text)
    ElementRec.Procedure = Trim$(efProcedure.Text)
End Sub
Sub Form_Load ()
    Call refreshlbList
    WINfmElement = True
```

```
End Sub
Sub Form_Unload (Cancel As Integer)
   WINfmElement = False
End Sub
Sub lbList_Click ()
   expression = Trim$(lbList.Text)
   fileInd = lbList.ItemData(lbList.ListIndex)
   If expression = NewItem Then
      Call clearEF
   Else
      Get #2, fileInd, ElementRec
      Call updateEF
   End If
End Sub
Sub pbAdd_Click ()
   If editControls() = True And Not elementExists() Then
      Call updateRec
      If unUsedInd <= maxUnUsed Then
         Put #2, unUsed(unUsedInd), ElementRec
         CurrentProcedure = unUsed(unUsedInd)
         unUsedInd = unUsedInd + 1
```

```
        Else
            Put #2, maxLoc + 1, ElementRec
            CurrentProcedure = maxLoc + 1
            maxLoc = maxLoc + 1
        End If
        Call refreshlbList
        actionSuccessful = True
    Else
        actionSuccessful = False
    End If
End Sub
Sub pbCancel_Click ()
    Unload fmElement
End Sub
Sub pbDelete_Click ()
    fileInd = lbList.ItemData(lbList.ListIndex)
    Call clearRec
    Put #2, fileInd, ElementRec
    Call refreshlbList
End Sub
Sub pbOther_Click ()
```

```
    If pbAdd.Enabled = True Then
        Call pbAdd_Click
    Else
        Call pbUpdate_Click
    End If
    If actionSuccessful = True Then
        fmOther.Show 1
    End If
End Sub
Sub pbUpdate_Click ()
    expression = Trim$(efNumber.Text) + "." + Trim$(efNumber1.Text) + "." +
Trim$(efNumber2.Text) + " " + Trim$(efName.Text)
    fileInd = lbList.ItemData(lbList.ListIndex)
    If editControls() = True Then
        If expression = Trim$(lbList.Text) Then
            Call updateRec
            Put #2, fileInd, ElementRec
            CurrentProcedure = fileInd
            Call refreshlbList
            actionSuccessful = True
        Else
```

```
        If Not elementExists() Then
            Call updateRec
            Put #2, fileInd, ElementRec
            CurrentProcedure = fileInd
            Call refreshlbList
            actionSuccessful = True
        Else
            actionSuccessful = False
        End If
    End If
  Else
    actionSuccessful = False
  End If
End Sub
```

A.5 Employee dialog box

Declarations

```
Dim unUsed() As Integer, maxUnUsed As Integer, maxLoc As Integer, fileInd As
Integer, unUsedInd As Integer, expression  As String, actionSuccessful As Integer
Sub clearEF ()
  efLast.Text = ""
```

```
    efFirst.Text = ""
    efMI.Text = ""
    cbSuffix.Text = ""
    cbTitle.Text = ""
    pbAdd.Enabled = True
    pbUpdate.Enabled = False
    pbDelete.Enabled = False
End Sub
Sub clearRec ()
    EmployeeRec.Last = ""
    EmployeeRec.First = ""
    EmployeeRec.MI = ""
    EmployeeRec.Suffix = ""
    EmployeeRec.Title = ""
End Sub
Function editControls () As Integer
Dim msg As String, userResponse As Integer
    msg = ""
    If Trim$(efLast.Text) = "" Then
        msg = msg + "Last name cannot be blanks."
    End If
```

```
    If msg <> "" Then
        userResponse = MsgBox(msg, 16, "Error")
        editControls = False
    Else
        editControls = True
    End If
End Function
Sub refreshlbList ()
    lbList.Clear
    lbList.AddItem (NewItem)
    fileInd = 1
    unUsedInd = 0
    Get #3, fileInd, EmployeeRec
    While Not EOF(3)
        expression = Trim$(Trim$(EmployeeRec.Last) + ", " +
Left$(EmployeeRec.First, 1) + EmployeeRec.MI)
        If expression <> "," Then
            lbList.AddItem (expression)
            lbList.ItemData(lbList.NewIndex) = fileInd
        Else
            unUsedInd = unUsedInd + 1
```

```
            ReDim Preserve unUsed(unUsedInd)
            unUsed(unUsedInd) = fileInd
        End If
        fileInd = fileInd + 1
        Get #3, fileInd, EmployeeRec
    Wend
    maxLoc = fileInd - 1
    maxUnUsed = unUsedInd
    unUsedInd = 1
    lbList.Selected(0) = True
    Call lbList_Click
End Sub
Sub updateEF ()
    efLast.Text = Trim$(EmployeeRec.Last)
    efFirst.Text = Trim$(EmployeeRec.First)
    efMI.Text = Trim$(EmployeeRec.MI)
    cbSuffix.Text = Trim$(EmployeeRec.Suffix)
    cbTitle.Text = Trim$(EmployeeRec.Title)
    pbAdd.Enabled = False
    pbUpdate.Enabled = True
    pbDelete.Enabled = True
```

```
End Sub
Sub updateRec ()
   EmployeeRec.Last = Trim$(efLast.Text)
   EmployeeRec.First = Trim$(efFirst.Text)
   EmployeeRec.MI = Trim$(efMI.Text)
   EmployeeRec.Suffix = Trim$(cbSuffix.Text)
   EmployeeRec.Title = Trim$(cbTitle.Text)
End Sub
Sub Form_Load ()
   Call refreshlbList
   cbSuffix.AddItem ("Sr.")
   cbSuffix.AddItem ("Jr.")
   cbSuffix.AddItem ("III")
   cbSuffix.AddItem ("IV")
   cbSuffix.AddItem ("V")
   cbTitle.AddItem ("Mr.")
   cbTitle.AddItem ("Mrs.")
   cbTitle.AddItem ("Ms.")
   cbTitle.AddItem ("Miss")
   cbTitle.AddItem ("Dr.")
   WINfmEmployee = True
```

```
End Sub
Sub Form_Unload (Cancel As Integer)
   WINfmEmployee = False
End Sub
Sub lbList_Click ()
   expression = Trim$(lbList.Text)
   fileInd = lbList.ItemData(lbList.ListIndex)
   If expression = NewItem Then
      Call clearEF
   Else
      Get #3, fileInd, EmployeeRec
      Call updateEF
   End If
End Sub
Sub pbAdd_Click ()
   If editControls() = True Then
      Call updateRec
      If unUsedInd <= maxUnUsed Then
         Put #3, unUsed(unUsedInd), EmployeeRec
         CurrentEmployee = unUsed(unUsedInd)
         unUsedInd = unUsedInd + 1
```

```
      Else
         Put #3, maxLoc + 1, EmployeeRec
         CurrentEmployee = maxLoc + 1
         maxLoc = maxLoc + 1
      End If
      Call refreshlbList
      actionSuccessful = True
   Else
      actionSuccessful = False
   End If
End Sub
Sub pbCancel_Click ()
   Unload fmEmployee
End Sub
Sub pbDelete_Click ()
   fileInd = lbList.ItemData(lbList.ListIndex)
   Call clearRec
   Put #3, fileInd, EmployeeRec
   Call refreshlbList
End Sub
Sub pbSkills_Click ()
```

```
    If pbAdd.Enabled = True Then
        Call pbAdd_Click
    Else
        Call pbUpdate_Click
    End If
    If actionSuccessful = True Then
        fmEmpSkl.Show 1
    End If
End Sub
Sub pbUpdate_Click ()
    fileInd = lbList.ItemData(lbList.ListIndex)
    If editControls() = True Then
        Call updateRec
        Put #3, fileInd, EmployeeRec
        CurrentEmployee = fileInd
        Call refreshlbList
        actionSuccessful = True
    Else
        actionSuccessful = False
    End If
End Sub
```

A.6 Method dialog box

```
Dim unUsed() As Integer, maxUnUsed As Integer, maxLoc As Integer, fileInd As
Integer, unUsedInd As Integer, expression  As String
Sub clearEF ()
   efName.Text = ""
   efDescription.Text = ""
   pbAdd.Enabled = True
   pbUpdate.Enabled = False
   pbDelete.Enabled = False
End Sub
Sub clearRec ()
   MethodRec.Name = ""
   MethodRec.Description = ""
End Sub
Function editControls () As Integer
Dim msg As String
   msg = ""
   If Trim$(efName.Text) = "" Then
      msg = msg + "Method name cannot be blanks."
   End If
```

```
  If msg <> "" Then
    userResponse% = MsgBox(msg, 16, "Error")
    editControls = False
  Else
    editControls = True
  End If
End Function
Function methodExists () As Integer
Dim flag As Integer, userResponse As Integer
  flag = False
  listInd = 0
  While (flag = False) And (listInd < lbList.ListCount)
    expression = Trim$(efName.Text)
    If expression = Trim$(lbList.List(listInd)) Then
      flag = True
    Else
      listInd = listInd + 1
    End If
  Wend
  If flag = True Then
```

```
        userResponse = MsgBox("Method name has already been used.", 16,
"Error")
    End If
    methodExists = flag
End Function
Sub refreshlbList ()
    lbList.Clear
    lbList.AddItem (NewItem)
    lbList.ItemData(0) = 0
    fileInd = 1
    unUsedInd = 0
    Get #4, fileInd, MethodRec
    While Not EOF(4)
        expression = Trim$(MethodRec.Name)
        If expression <> "" Then
            lbList.AddItem (expression)
            lbList.ItemData(lbList.NewIndex) = fileInd
        Else
            unUsedInd = unUsedInd + 1
            ReDim Preserve unUsed(unUsedInd)
            unUsed(unUsedInd) = fileInd
```

```
        End If
        fileInd = fileInd + 1
        Get #4, fileInd, MethodRec
    Wend
    maxLoc = fileInd - 1
    maxUnUsed = unUsedInd
    unUsedInd = 1
    lbList.Selected(0) = True
    Call lbList_Click
End Sub
Sub updateEF ()
    efName.Text = Trim$(MethodRec.Name)
    efDescription.Text = Trim$(MethodRec.Description)
    pbAdd.Enabled = False
    pbUpdate.Enabled = True
    pbDelete.Enabled = True
End Sub
Sub updateRec ()
    MethodRec.Name = Trim$(efName.Text)
    MethodRec.Description = Trim$(efDescription.Text)
End Sub
```

```
Sub Form_Load ()
   Call refreshlbList
   WINfmMethod = True
End Sub
Sub Form_Unload (Cancel As Integer)
   WINfmMethod = False
End Sub
Sub lbList_Click ()
   expression = Trim$(lbList.Text)
   fileInd = lbList.ItemData(lbList.ListIndex)
   If expression = NewItem Then
      Call clearEF
   Else
      Get #4, fileInd, MethodRec
      Call updateEF
   End If
End Sub
Sub pbAdd_Click ()
   If editControls() = True And Not methodExists() Then
      Call updateRec
      If unUsedInd <= maxUnUsed Then
```

```
            Put #4, unUsed(unUsedInd), MethodRec
            unUsedInd = unUsedInd + 1
        Else
            Put #4, maxLoc + 1, MethodRec
            maxLoc = maxLoc + 1
        End If
        Call refreshlbList
    End If
End Sub
Sub pbCancel_Click ()
    Unload fmMethod
End Sub
Sub pbDelete_Click ()
    fileInd = lbList.ItemData(lbList.ListIndex)
    Call clearRec
    Put #4, fileInd, MethodRec
    Call refreshlbList
End Sub
Sub pbUpdate_Click ()
    expression = Trim$(efName.Text)
    fileInd = lbList.ItemData(lbList.ListIndex)
```

```
    If editControls() = True Then
      If expression = Trim$(lbList.Text) Then
        Call updateRec
        Put #4, fileInd, MethodRec
        Call refreshlbList
      Else
        If Not methodExists() Then
          Call updateRec
          Put #4, fileInd, MethodRec
          Call refreshlbList
        End If
      End If
    End If
End Sub
```

A.7 Resource dialog box

```
Dim unUsed() As Integer, maxUnUsed As Integer, maxLoc As Integer, fileInd As
Integer, unUsedInd As Integer, expression  As String
Sub clearEF ()
    efName.Text = ""
    efDescription.Text = ""
```

```
    pbAdd.Enabled = True
    pbUpdate.Enabled = False
    pbDelete.Enabled = False
End Sub
Sub clearRec ()
    ResourceRec.Name = ""
    ResourceRec.Description = ""
End Sub
Function editControls () As Integer
Dim msg As String
    msg = ""
    If Trim$(efName.Text) = "" Then
        msg = msg + "Resource name cannot be blanks."
    End If
    If msg <> "" Then
        userResponse% = MsgBox(msg, 16, "Error")
        editControls = False
    Else
        editControls = True
    End If
End Function
```

```
Sub refreshlbList ()
   lbList.Clear
   lbList.AddItem (NewItem)
   lbList.ItemData(0) = 0
   fileInd = 1
   unUsedInd = 0
   Get #6, fileInd, ResourceRec
   While Not EOF(6)
      expression = Trim$(ResourceRec.Name)
      If expression <> "" Then
         lbList.AddItem (expression)
         lbList.ItemData(lbList.NewIndex) = fileInd
      Else
         unUsedInd = unUsedInd + 1
         ReDim Preserve unUsed(unUsedInd)
         unUsed(unUsedInd) = fileInd
      End If
      fileInd = fileInd + 1
      Get #6, fileInd, ResourceRec
   Wend
   maxLoc = fileInd - 1
```

```
    maxUnUsed = unUsedInd
    unUsedInd = 1
    lbList.Selected(0) = True
    Call lbList_Click
End Sub
Function resourceExists () As Integer
Dim flag As Integer, userResponse As Integer
    flag = False
    listInd = 0
    While (flag = False) And (listInd < lbList.ListCount)
        expression = Trim$(efName.Text)
        If expression = Trim$(lbList.List(listInd)) Then
            flag = True
        Else
            listInd = listInd + 1
        End If
    Wend
    If flag = True Then
        userResponse = MsgBox("Resource name has already been used.", 16,
"Error")
    End If
```

```
    resourceExists = flag
End Function
Sub updateEF ()
    efName.Text = Trim$(ResourceRec.Name)
    efDescription.Text = Trim$(ResourceRec.Description)
    pbAdd.Enabled = False
    pbUpdate.Enabled = True
    pbDelete.Enabled = True
End Sub
Sub updateRec ()
    ResourceRec.Name = Trim$(efName.Text)
    ResourceRec.Description = Trim$(efDescription.Text)
End Sub
Sub Form_Load ()
    Call refreshlbList
    WINfmResource = True
End Sub
Sub Form_Unload (Cancel As Integer)
    WINfmResource = False
End Sub
Sub lbList_Click ()
```

```
    expression = Trim$(lbList.Text)
    fileInd = lbList.ItemData(lbList.ListIndex)
    If expression = NewItem Then
        Call clearEF
    Else
        Get #6, fileInd, ResourceRec
        Call updateEF
    End If
End Sub
Sub pbAdd_Click ()
    If editControls() = True And Not resourceExists() Then
        Call updateRec
        If unUsedInd <= maxUnUsed Then
            Put #6, unUsed(unUsedInd), ResourceRec
            unUsedInd = unUsedInd + 1
        Else
            Put #6, maxLoc + 1, ResourceRec
            maxLoc = maxLoc + 1
        End If
        Call refreshlbList
    End If
```

```
End Sub
Sub pbCancel_Click ()
   Unload fmResource
End Sub
Sub pbDelete_Click ()
   fileInd = lbList.ItemData(lbList.ListIndex)
   Call clearRec
   Put #6, fileInd, ResourceRec
   Call refreshlbList
End Sub
Sub pbUpdate_Click ()
   expression = Trim$(efName.Text)
   fileInd = lbList.ItemData(lbList.ListIndex)
   If editControls() = True Then
      If expression = Trim$(lbList.Text) Then
         Call updateRec
         Put #6, fileInd, ResourceRec
         Call refreshlbList
      Else
         If Not resourceExists() Then
            Call updateRec
```

```
            Put #6, fileInd, ResourceRec
            Call refreshlbList
          End If
        End If
    End If
End Sub
```

A.8 Skill dialog box

```
Declarations
Dim unUsed() As Integer, maxUnUsed As Integer, maxLoc As Integer, fileInd As
Integer, unUsedInd As Integer, expression As String
Sub clearEF ()
    efName.Text = ""
    efDescription.Text = ""
    pbAdd.Enabled = True
    pbUpdate.Enabled = False
    pbDelete.Enabled = False
End Sub
Sub clearRec ()
    SkillRec.Name = ""
    SkillRec.Description = ""
```

```
End Sub
Function editControls () As Integer
Dim msg As String
   msg = ""
   If Trim$(efName.Text) = "" Then
      msg = msg + "Skill name cannot be blanks."
   End If
   If msg <> "" Then
      userResponse% = MsgBox(msg, 16, "Error")
      editControls = False
   Else
      editControls = True
   End If
End Function
Sub refreshlbList ()
   lbList.Clear
   lbList.AddItem (NewItem)
   lbList.ItemData(0) = 0
   fileInd = 1
   unUsedInd = 0
   Get #5, fileInd, SkillRec
```

```
    While Not EOF(5)
      expression = Trim$(SkillRec.Name)
      If expression <> "" Then
        lbList.AddItem (expression)
        lbList.ItemData(lbList.NewIndex) = fileInd
      Else
        unUsedInd = unUsedInd + 1
        ReDim Preserve unUsed(unUsedInd)
        unUsed(unUsedInd) = fileInd
      End If
      fileInd = fileInd + 1
      Get #5, fileInd, SkillRec
    Wend
    maxLoc = fileInd - 1
    maxUnUsed = unUsedInd
    unUsedInd = 1
    lbList.Selected(0) = True
    Call lbList_Click
End Sub
Function skillExists () As Integer
Dim flag As Integer, userResponse As Integer
```

```
    flag = False
    listInd = 0
    While (flag = False) And (listInd < lbList.ListCount)
      expression = Trim$(efName.Text)
      If expression = Trim$(lbList.List(listInd)) Then
        flag = True
      Else
        listInd = listInd + 1
      End If
    Wend
    If flag = True Then
      userResponse = MsgBox("Skill name has already been used.", 16, "Error")
    End If
    skillExists = flag
End Function
Sub updateEF ()
    efName.Text = Trim$(SkillRec.Name)
    efDescription.Text = Trim$(SkillRec.Description)
    pbAdd.Enabled = False
    pbUpdate.Enabled = True
    pbDelete.Enabled = True
```

```
End Sub

Sub updateRec ()

   SkillRec.Name = Trim$(efName.Text)

   SkillRec.Description = Trim$(efDescription.Text)

End Sub

Sub Form_Load ()

   Call refreshlbList

   WINfmSkill = True

End Sub

Sub Form_Unload (Cancel As Integer)

   WINfmSkill = False

End Sub

Sub lbList_Click ()

   expression = Trim$(lbList.Text)

   fileInd = lbList.ItemData(lbList.ListIndex)

   If expression = NewItem Then

      Call clearEF

   Else

      Get #5, fileInd, SkillRec

      Call updateEF

   End If
```

```
End Sub
Sub pbAdd_Click ()
   If editControls() = True And Not skillExists() Then
      Call updateRec
      If unUsedInd <= maxUnUsed Then
         Put #5, unUsed(unUsedInd), SkillRec
         unUsedInd = unUsedInd + 1
      Else
         Put #5, maxLoc + 1, SkillRec
         maxLoc = maxLoc + 1
      End If
      Call refreshlbList
   End If
End Sub
Sub pbCancel_Click ()
   Unload fmSkill
End Sub
Sub pbDelete_Click ()
   fileInd = lbList.ItemData(lbList.ListIndex)
   Call clearRec
   Put #5, fileInd, SkillRec
```

```
    Call refreshlbList
End Sub
Sub pbUpdate_Click ()
    expression = Trim$(efName.Text)
    fileInd = lbList.ItemData(lbList.ListIndex)
    If editControls() = True Then
        If expression = Trim$(lbList.Text) Then
            Call updateRec
            Put #5, fileInd, SkillRec
            Call refreshlbList
        Else
            If Not skillExists() Then
                Call updateRec
                Put #5, fileInd, SkillRec
                Call refreshlbList
            End If
        End If
    End If
End Sub
```

A.9 Procedure-other dialog box

```
Declarations
Dim unUsed1() As Integer, maxUnUsed1 As Integer, maxLoc1 As Integer,
fileInd1 As Integer, unUsedInd1 As Integer, expression1 As String,
tempMethodRec As MethodRecType
Dim unUsed2() As Integer, maxUnUsed2 As Integer, maxLoc2 As Integer,
fileInd2 As Integer, unUsedInd2 As Integer, expression2 As String,
tempResourceRec As ResourceRecType
Dim unUsed3() As Integer, maxUnUsed3 As Integer, maxLoc3 As Integer,
fileInd3 As Integer, unUsedInd3 As Integer, expression3 As String, tempSkillRec
As SkillRecType
Dim unUsed4() As Integer, maxUnUsed4 As Integer, maxLoc4 As Integer,
fileInd4 As Integer, unUsedInd4 As Integer, expression4 As String,
tempEmployeeRec As EmployeeRecType
Sub clearRec1 ()
   ProcedureMethodRec.ElementIndex = 0
   ProcedureMethodRec.MethodIndex = 0
End Sub
Sub clearRec2 ()
   ProcedureResourceRec.ElementIndex = 0
   ProcedureResourceRec.ResourceIndex = 0
End Sub
```

```
Sub clearRec3 ()
   ProcedureSkillRec.ElementIndex = 0
   ProcedureSkillRec.SkillIndex = 0
End Sub
Sub clearRec4 ()
   ProcedureEmployeeRec.ElementIndex = 0
   ProcedureEmployeeRec.EmployeeIndex = 0
End Sub
Function employeeInSet (Selected As String) As Integer
Dim flag As Integer, tempInd As Integer
   flag = False
   tempInd = 0
   While (flag = False) And (tempInd < lbSet4.ListCount)
      If Selected = Trim$(lbSet4.List(tempInd)) Then
         flag = True
      Else
         tempInd = tempInd + 1
      End If
   Wend
   employeeInSet = flag
End Function
```

```
Function methodInSet (Selected As String) As Integer
Dim flag As Integer, tempInd As Integer
   flag = False
   tempInd = 0
   While (flag = False) And (tempInd < lbSet1.ListCount)
      If Selected = Trim$(lbSet1.List(tempInd)) Then
         flag = True
      Else
         tempInd = tempInd + 1
      End If
   Wend
   methodInSet = flag
End Function
Sub refreshlbList1 ()
   lbList1.Clear
   fileInd1 = 1
   Get #4, fileInd1, tempMethodRec
   While Not EOF(4)
      expression1 = Trim$(tempMethodRec.Name)
      If expression1 <> "" And Not methodInSet(expression1) Then
         lbList1.AddItem (expression1)
```

```
        lbList1.ItemData(lbList1.NewIndex) = fileInd1
      End If
      fileInd1 = fileInd1 + 1
      Get #4, fileInd1, tempMethodRec
    Wend
    pbAdd1.Enabled = False
End Sub
Sub refreshlbList2 ()
    lbList2.Clear
    fileInd2 = 1
    Get #6, fileInd2, tempResourceRec
    While Not EOF(6)
      expression2 = Trim$(tempResourceRec.Name)
      If expression2 <> "" And Not resourceInSet(expression2) Then
        lbList2.AddItem (expression2)
        lbList2.ItemData(lbList2.NewIndex) = fileInd2
      End If
      fileInd2 = fileInd2 + 1
      Get #6, fileInd2, tempResourceRec
    Wend
    pbAdd2.Enabled = False
```

```
End Sub
Sub refreshlbList3 ()
   lbList3.Clear
   fileInd3 = 1
   Get #5, fileInd3, tempSkillRec
   While Not EOF(5)
      expression3 = Trim$(tempSkillRec.Name)
      If expression3 <> "" And Not skillInSet(expression3) Then
         lbList3.AddItem (expression3)
         lbList3.ItemData(lbList3.NewIndex) = fileInd3
      End If
      fileInd3 = fileInd3 + 1
      Get #5, fileInd3, tempSkillRec
   Wend
   pbAdd3.Enabled = False
End Sub
Sub refreshlbList4 ()
   lbList4.Clear
   fileInd4 = 1
   Get #3, fileInd4, tempEmployeeRec
   While Not EOF(3)
```

```
      expression4 = Trim$(tempEmployeeRec.Last) + ", " +
Trim$(Left$(tempEmployeeRec.First, 1)) + Trim$(tempEmployeeRec.MI) +
Str$(fileInd4)
      If (expression4 <> (", " + Str$(fileInd4))) And Not
employeeInSet(expression4) Then
         lbList4.AddItem (expression4)
         lbList4.ItemData(lbList4.NewIndex) = fileInd4
      End If
      fileInd4 = fileInd4 + 1
      Get #3, fileInd4, tempEmployeeRec
   Wend
   pbAdd4.Enabled = False
End Sub
Sub refreshlbSet1 ()
   lbSet1.Clear
   fileInd1 = 1
   unUsedInd1 = 0
   Get #13, fileInd1, ProcedureMethodRec
   While Not EOF(13)
      If ProcedureMethodRec.ElementIndex = CurrentProcedure Then
         Get #4, ProcedureMethodRec.MethodIndex, tempMethodRec
```

```
            expression1 = Trim$(tempMethodRec.Name)
            lbSet1.AddItem (expression1)
            lbSet1.ItemData(lbSet1.NewIndex) = fileInd1
        Else
            If ProcedureMethodRec.ElementIndex = 0 Then
                unUsedInd1 = unUsedInd1 + 1
                ReDim Preserve unUsed1(unUsedInd1)
                unUsed1(unUsedInd1) = fileInd1
            End If
        End If
        fileInd1 = fileInd1 + 1
        Get #13, fileInd1, ProcedureMethodRec
    Wend
    maxLoc1 = fileInd1 - 1
    maxUnUsed1 = unUsedInd1
    unUsedInd1 = 1
    pbRemove1.Enabled = False
End Sub
Sub refreshlbSet2 ()
    lbSet2.Clear
    fileInd2 = 1
```

```
unUsedInd2 = 0
Get #12, fileInd2, ProcedureResourceRec
While Not EOF(12)
   If ProcedureResourceRec.ElementIndex = CurrentProcedure Then
      Get #6, ProcedureResourceRec.ResourceIndex, tempResourceRec
      expression2 = Trim$(tempResourceRec.Name)
      lbSet2.AddItem (expression2)
      lbSet2.ItemData(lbSet2.NewIndex) = fileInd2
   Else
      If ProcedureResourceRec.ElementIndex = 0 Then
         unUsedInd2 = unUsedInd2 + 1
         ReDim Preserve unUsed2(unUsedInd2)
         unUsed2(unUsedInd2) = fileInd2
      End If
   End If
   fileInd2 = fileInd2 + 1
   Get #12, fileInd2, ProcedureResourceRec
Wend
maxLoc2 = fileInd2 - 1
maxUnUsed2 = unUsedInd2
unUsedInd2 = 1
```

```
    pbRemove2.Enabled = False
End Sub
Sub refreshlbSet3 ()
    lbSet3.Clear
    fileInd3 = 1
    unUsedInd3 = 0
    Get #9, fileInd3, ProcedureSkillRec
    While Not EOF(9)
        If ProcedureSkillRec.ElementIndex = CurrentProcedure Then
            Get #5, ProcedureSkillRec.SkillIndex, tempSkillRec
            expression3 = Trim$(tempSkillRec.Name)
            lbSet3.AddItem (expression3)
            lbSet3.ItemData(lbSet3.NewIndex) = fileInd3
        Else
            If ProcedureSkillRec.ElementIndex = 0 Then
                unUsedInd3 = unUsedInd3 + 1
                ReDim Preserve unUsed3(unUsedInd3)
                unUsed3(unUsedInd3) = fileInd3
            End If
        End If
        fileInd3 = fileInd3 + 1
```

```
      Get #9, fileInd3, ProcedureSkillRec
   Wend
   maxLoc3 = fileInd3 - 1
   maxUnUsed3 = unUsedInd3
   unUsedInd3 = 1
   pbRemove3.Enabled = False
End Sub
Sub refreshlbSet4 ()
   lbSet4.Clear
   fileInd4 = 1
   unUsedInd4 = 0
   Get #11, fileInd4, ProcedureEmployeeRec
   While Not EOF(11)
      If ProcedureEmployeeRec.ElementIndex = CurrentProcedure Then
         Get #3, ProcedureEmployeeRec.EmployeeIndex, tempEmployeeRec
         expression4 = Trim$(tempEmployeeRec.Last) + ", " +
Left$(tempEmployeeRec.First, 1) + tempEmployeeRec.MI +
Str$(ProcedureEmployeeRec.EmployeeIndex)
         lbSet4.AddItem (expression4)
         lbSet4.ItemData(lbSet4.NewIndex) = fileInd4
      Else
```

```
        If ProcedureEmployeeRec.ElementIndex = 0 Then
          unUsedInd4 = unUsedInd4 + 1
          ReDim Preserve unUsed4(unUsedInd4)
          unUsed4(unUsedInd4) = fileInd4
        End If
      End If
      fileInd4 = fileInd4 + 1
      Get #11, fileInd4, ProcedureEmployeeRec
    Wend
    maxLoc4 = fileInd4 - 1
    maxUnUsed4 = unUsedInd4
    unUsedInd4 = 1
    pbRemove4.Enabled = False
End Sub
Function resourceInSet (Selected As String) As Integer
Dim flag As Integer, tempInd As Integer
    flag = False
    tempInd = 0
    While (flag = False) And (tempInd < lbSet2.ListCount)
      If Selected = Trim$(lbSet2.List(tempInd)) Then
        flag = True
```

```
      Else
         tempInd = tempInd + 1
      End If
   Wend
   resourceInSet = flag
End Function
Sub setStaticText ()
   Get #2, CurrentProcedure, ElementRec
   stVarNumber.Caption = Trim$(Str$(ElementRec.Number)) + "." +
Trim$(Str$(ElementRec.Number1)) + "." + Trim$(Str$(ElementRec.Number2))
   stVarName.Caption = Trim$(ElementRec.Name)
End Sub
Function skillInSet (Selected As String) As Integer
Dim flag As Integer, tempInd As Integer
   flag = False
   tempInd = 0
   While (flag = False) And (tempInd < lbSet3.ListCount)
      If Selected = Trim$(lbSet3.List(tempInd)) Then
         flag = True
      Else
         tempInd = tempInd + 1
```

```
        End If
    Wend
    skillInSet = flag
End Function
Sub Form_Load ()
    Call setStaticText
    Call refreshlbSet1
    Call refreshlbSet2
    Call refreshlbSet3
    Call refreshlbSet4
    Call refreshlbList1
    Call refreshlbList2
    Call refreshlbList3
    Call refreshlbList4
End Sub
Sub lbList1_Click ()
    pbAdd1.Enabled = True
End Sub
Sub lbList2_Click ()
    pbAdd2.Enabled = True
End Sub
```

```
Sub lbList3_Click ()
   pbAdd3.Enabled = True
End Sub
Sub lbList4_Click ()
   pbAdd4.Enabled = True
End Sub
Sub lbSet1_Click ()
   pbRemove1.Enabled = True
End Sub
Sub lbSet2_Click ()
   pbRemove2.Enabled = True
End Sub
Sub lbSet3_Click ()
   pbRemove3.Enabled = True
End Sub
Sub lbSet4_Click ()
   pbRemove4.Enabled = True
End Sub
Sub pbAdd1_Click ()
   listInd1 = 0
   While listInd1 < lbList1.ListCount
```

```
      If lbList1.Selected(listInd1) = True Then
         ProcedureMethodRec.ElementIndex = CurrentProcedure
         ProcedureMethodRec.MethodIndex = lbList1.ItemData(listInd1)
         If unUsedInd1 <= maxUnUsed1 Then
            Put #13, unUsed1(unUsedInd1), ProcedureMethodRec
            unUsedInd1 = unUsedInd1 + 1
         Else
            Put #13, maxLoc1 + 1, ProcedureMethodRec
            maxLoc1 = maxLoc1 + 1
         End If
      End If
      listInd1 = listInd1 + 1
   Wend
   Call refreshlbSet1
   Call refreshlbList1
End Sub
Sub pbAdd2_Click ()
   listInd2 = 0
   While listInd2 < lbList2.ListCount
      If lbList2.Selected(listInd2) = True Then
         ProcedureResourceRec.ElementIndex = CurrentProcedure
```

```
        ProcedureResourceRec.ResourceIndex = lbList2.ItemData(listInd2)
        If unUsedInd2 <= maxUnUsed2 Then
          Put #12, unUsed2(unUsedInd2), ProcedureResourceRec
          unUsedInd2 = unUsedInd2 + 1
        Else
          Put #12, maxLoc2 + 1, ProcedureResourceRec
          maxLoc2 = maxLoc2 + 1
        End If
      End If
      listInd2 = listInd2 + 1
    Wend
    Call refreshlbSet2
    Call refreshlbList2
End Sub
Sub pbAdd3_Click ()
    listInd3 = 0
    While listInd3 < lbList3.ListCount
      If lbList3.Selected(listInd3) = True Then
        ProcedureSkillRec.ElementIndex = CurrentProcedure
        ProcedureSkillRec.SkillIndex = lbList3.ItemData(listInd3)
        If unUsedInd3 <= maxUnUsed3 Then
```

```
            Put #9, unUsed3(unUsedInd3), ProcedureSkillRec
            unUsedInd3 = unUsedInd3 + 1
          Else
            Put #9, maxLoc3 + 3, ProcedureSkillRec
            maxLoc3 = maxLoc3 + 1
          End If
        End If
        listInd3 = listInd3 + 1
      Wend
      Call refreshlbSet3
      Call refreshlbList3
End Sub
Sub pbAdd4_Click ()
      listInd4 = 0
      While listInd4 < lbList4.ListCount
        If lbList4.Selected(listInd4) = True Then
          ProcedureEmployeeRec.ElementIndex = CurrentProcedure
          ProcedureEmployeeRec.EmployeeIndex = lbList4.ItemData(listInd4)
          If unUsedInd4 <= maxUnUsed4 Then
            Put #11, unUsed4(unUsedInd4), ProcedureEmployeeRec
            unUsedInd4 = unUsedInd4 + 1
```

```
        Else
          Put #11, maxLoc4 + 1, ProcedureEmployeeRec
          maxLoc4 = maxLoc4 + 1
        End If
      End If
      listInd4 = listInd4 + 1
    Wend
    Call refreshlbSet4
    Call refreshlbList4
End Sub
Sub pbCancel_Click ()
    Unload fmOther
End Sub
Sub pbRemove1_Click ()
    listInd1 = 0
    While listInd1 < lbSet1.ListCount
      If lbSet1.Selected(listInd1) = True Then
        fileInd1 = lbSet1.ItemData(listInd1)
        Call clearRec1
        Put #13, fileInd1, ProcedureMethodRec
      End If
```

```
      listInd1 = listInd1 + 1
    Wend
    Call refreshlbSet1
    Call refreshlbList1
End Sub
Sub pbRemove2_Click ()
    listInd2 = 0
    While listInd2 < lbSet2.ListCount
      If lbSet2.Selected(listInd2) = True Then
        fileInd2 = lbSet2.ItemData(listInd2)
        Call clearRec2
        Put #12, fileInd2, ProcedureResourceRec
      End If
      listInd2 = listInd2 + 1
    Wend
    Call refreshlbSet2
    Call refreshlbList2
End Sub
Sub pbRemove3_Click ()
    listInd3 = 0
    While listInd3 < lbSet3.ListCount
```

```
    If lbSet3.Selected(listInd3) = True Then
      fileInd3 = lbSet3.ItemData(listInd3)
      Call clearRec3
      Put #9, fileInd3, ProcedureSkillRec
    End If
    listInd3 = listInd3 + 1
  Wend
  Call refreshlbSet3
  Call refreshlbList3
End Sub
Sub pbRemove4_Click ()
  listInd4 = 0
  While listInd4 < lbSet4.ListCount
    If lbSet4.Selected(listInd4) = True Then
      fileInd4 = lbSet4.ItemData(listInd4)
      Call clearRec4
      Put #11, fileInd4, ProcedureEmployeeRec
    End If
    listInd4 = listInd4 + 1
  Wend
  Call refreshlbSet4
```

```
    Call refreshlbList4
End Sub
Sub pbView_Click ()
Dim eol As String, text As String, ind As Integer, eRec As ElementRecType,
mRec As MethodRecType, rRec As ResourceRecType, sRec As SkillRecType,
yRec As EmployeeRecType, pmRec As ProcedureMethodRecType, prRec As
ProcedureResourceRecType, psRec As ProcedureSkillRecType, pyRec As
ProcedureEmployeeRecType
    eol = Chr$(13) + Chr$(10)
    text = ""
    Get #2, CurrentProcedure, eRec
    text = text + "Element:" + eol + Trim$(Str$(eRec.Number)) + "." +
Trim$(Str$(eRec.Number1)) + "." + Trim$(Str$(eRec.Number2)) + "  " +
Trim$(eRec.Name) + eol
    text = text + Trim$(eRec.Procedure) + eol + eol
    ind = 0
    text = text + "Verification methods:" + eol
    While ind < lbSet1.ListCount
      Get #13, lbSet1.ItemData(ind), pmRec
      Get #4, pmRec.MethodIndex, mRec
      text = text + Trim$(mRec.Name) + eol
```

```
    ind = ind + 1
Wend
text = text + eol
ind = 0
text = text + "Resource required:" + eol
While ind < lbSet2.ListCount
    Get #12, lbSet2.ItemData(ind), prRec
    Get #6, prRec.ResourceIndex, rRec
    text = text + Trim$(rRec.Name) + eol
    ind = ind + 1
Wend
text = text + eol
ind = 0
text = text + "Skills needed:" + eol
While ind < lbSet3.ListCount
    Get #9, lbSet3.ItemData(ind), psRec
    Get #5, psRec.SkillIndex, sRec
    text = text + Trim$(sRec.Name) + eol
    ind = ind + 1
Wend
text = text + eol
```

```
    ind = 0
    text = text + "Employees responsible:" + eol
    While ind < lbSet4.ListCount
      Get #11, lbSet4.ItemData(ind), pyRec
      Get #3, pyRec.EmployeeIndex, yRec
      text = text + Trim$(yRec.Last) + ", " + Trim$(yRec.First) + " " + yRec.MI +
eol
      ind = ind + 1
    Wend
    text = text + eol
    ProcedureText = text
    fmProcedureText.Show 1
End Sub
```

A.10 Employee-skill dialog box

```
Declarations
Dim unUsed() As Integer, maxUnUsed As Integer, maxLoc As Integer, fileInd As
Integer, unUsedInd As Integer, expression  As String, tempSkillRec As
SkillRecType
Sub clearRec ()
  EmployeeSkillRec.EmployeeIndex = 0
```

```
    EmployeeSkillRec.SkillIndex = 0
End Sub
Sub refreshlbList ()
    lbList.Clear
    fileInd = 1
    Get #5, fileInd, tempSkillRec
    While Not EOF(5)
        expression = Trim$(tempSkillRec.Name)
        If expression <> "" And Not skillInSet(expression) Then
            lbList.AddItem (expression)
            lbList.ItemData(lbList.NewIndex) = fileInd
        End If
        fileInd = fileInd + 1
        Get #5, fileInd, tempSkillRec
    Wend
    pbAdd.Enabled = False
End Sub
Sub refreshlbSet ()
    lbSet.Clear
    fileInd = 1
    unUsedInd = 0
```

```
Get #10, fileInd, EmployeeSkillRec
While Not EOF(10)
   If EmployeeSkillRec.EmployeeIndex = CurrentEmployee Then
      Get #5, EmployeeSkillRec.SkillIndex, tempSkillRec
      expression = Trim$(tempSkillRec.Name)
      lbSet.AddItem (expression)
      lbSet.ItemData(lbSet.NewIndex) = fileInd
   Else
      If EmployeeSkillRec.EmployeeIndex = 0 Then
         unUsedInd = unUsedInd + 1
         ReDim Preserve unUsed(unUsedInd)
         unUsed(unUsedInd) = fileInd
      End If
   End If
   fileInd = fileInd + 1
   Get #10, fileInd, EmployeeSkillRec
Wend
maxLoc = fileInd - 1
maxUnUsed = unUsedInd
unUsedInd = 1
pbRemove.Enabled = False
```

```
End Sub
Sub setStaticText ()
   Get #3, CurrentEmployee, EmployeeRec
   stVarLast.Caption = Trim$(EmployeeRec.Last)
   stVarFirst.Caption = Trim$(EmployeeRec.First)
   stVarMI.Caption = Trim$(EmployeeRec.MI)
   stVarSuffix.Caption = Trim$(EmployeeRec.Suffix)
   stVarTitle.Caption = Trim$(EmployeeRec.Title)
End Sub
Function skillInSet (Selected As String) As Integer
Dim flag As Integer, tempInd As Integer
   flag = False
   tempInd = 0
   While (flag = False) And (tempInd < lbSet.ListCount)
      If Selected = Trim$(lbSet.List(tempInd)) Then
         flag = True
      Else
         tempInd = tempInd + 1
      End If
   Wend
   skillInSet = flag
```

```
End Function

Sub Form_Load ()

   Call setStaticText

   Call refreshlbSet

   Call refreshlbList

End Sub

Sub lbList_Click ()

   pbAdd.Enabled = True

End Sub

Sub lbSet_Click ()

   pbRemove.Enabled = True

End Sub

Sub pbCancel_Click ()

   Unload fmEmpSkl

End Sub

Sub pbRemove_Click ()

   listInd = 0

   While listInd < lbSet.ListCount

      If lbSet.Selected(listInd) = True Then

         fileInd = lbSet.ItemData(listInd)

         Call clearRec
```

```
        Put #10, fileInd, EmployeeSkillRec
      End If
      listInd = listInd + 1
    Wend
    Call refreshlbSet
    Call refreshlbList
End Sub
```

A.11 View procedure dialog box

```
Sub Form_Load ()
    efText = ProcedureText
End Sub
Sub pbCancel_Click ()
    Unload fmProcedureText
End Sub
```

A.12 File-new dialog box

```
Function editControls () As Integer
Dim msg As String
    msg = ""
    If Trim$(efFileName.Text) = "" Then
```

```
      msg = msg + "File name cannot be blanks."
   End If
   If msg <> "" Then
      UserResponse% = MsgBox(msg, 16, "Error")
      editControls = False
   Else
      editControls = True
   End If
End Function
Sub pbCancel_Click ()
   Unload fmNewFile
   RCfmNewFile = False
End Sub
Sub pbOk_Click ()
   On Error Resume Next
   If editControls() = True Then
      MkDir BaseDirectory + "\" + Trim$(efFileName.Text)
      If Err <> 0 Then
         UserResponse% = MsgBox("File name not valid or already exists.", 16,
"Error")
      Else
```

```
        ChDir BaseDirectory + "\" + Trim$(efFileName.Text)
        Call OpenFiles
        CurrentFile = Trim$(efFileName.Text)
        RCfmNewFile = True
        Unload fmNewFile
      End If
    End If
End Sub
```

A.13 File list dialog box

```
Sub Form_Load ()
Dim count, d(), i, dirName      ' Declare variables.
    If FileAction = "O" Then    ' set text
      stFileList.Caption = "Select a file to open"
    Else
      stFileList.Caption = "Select a file to delete"
    End If
                        ' get subdirectories
    dirName = Dir$(BaseDirectory + "\*.*", 16)
    Do While dirName <> ""
      If dirName <> "." And dirName <> ".." Then
```

```
        If GetAttr(BaseDirectory + "\" + dirName) = 16 Then
          If (count Mod 10) = 0 Then
            ReDim Preserve d(count + 10)
          End If
          count = count + 1
          d(count) = dirName
        End If
      End If
      dirName = Dir$
    Loop
    lbFileList.Clear
    For i = 1 To count      ' add items and select current
      lbFileList.AddItem d(i)
      If CurDir = BaseDirectory + "\" + d(i) Then
        lbFileList.Selected(i - 1) = True
      End If
    Next i
    pbOk.Enabled = False
End Sub
Sub lbFileList_Click ()
    pbOk.Enabled = True
```

```
End Sub
Sub lbFileList_DblClick ()
   Call lbFileList_Click
   Call pbOk_Click
End Sub
Sub pbCancel_Click ()
   RCfmFileList = False
   Unload fmFileList
End Sub
Sub pbOk_Click ()
   On Error Resume Next
   If FileAction = "O" Then
      ChDir BaseDirectory + "\" + lbFileList.Text
      Call OpenFiles
      CurrentFile = Trim$(lbFileList.Text)
      RCfmFileList = True
      Unload fmFileList
   Else
      Kill BaseDirectory + "\" + Trim$(lbFileList.Text) + "\*.*"
      RmDir Trim$(lbFileList.Text)
      If Err <> 0 Then
```

```
        UserResponse% = MsgBox("Error removing directory.  File not deleted.",
16, "Error")
      End If
      RCfmFileList = True
      Unload fmFileList
   End If
End Sub
```

A.14 Process quality manual dialog box

```
Declarations
Dim curProc As Integer, listInd As Integer, fileInd As Integer, expression  As
String, tempElementRec As ElementRecType, tempMethodRec As
MethodRecType, tempResourceRec As ResourceRecType, tempSkillRec As
SkillRecType, tempEmployeeRec As EmployeeRecType, pmRec As
ProcedureMethodRecType, prRec As ProcedureResourceRecType, psRec As
ProcedureSkillRecType, pyRec As ProcedureEmployeeRecType
Sub refreshlbList ()
   lbList.Clear
   fileInd = 1
   Get #2, fileInd, tempElementRec
   While Not EOF(2)
```

```
      expression = Trim$(Str$(tempElementRec.Number)) + "." +
Trim$(Str$(tempElementRec.Number1)) + "." +
Trim(Str$(tempElementRec.Number2)) + " " + Trim$(tempElementRec.Name)
      If expression <> "0.0.0 " Then
         lbList.AddItem (expression)
         lbList.ItemData(lbList.NewIndex) = fileInd
      End If
      fileInd = fileInd + 1
      Get #2, fileInd, tempElementRec
   Wend
End Sub
Sub refreshlbSet1 ()
   lbSet1.Clear
   fileInd = 1
   Get #13, fileInd, pmRec
   While Not EOF(13)
      If pmRec.ElementIndex = curProc Then
         Get #4, pmRec.MethodIndex, tempMethodRec
         expression = Trim$(tempMethodRec.Name)
         lbSet1.AddItem (expression)
         lbSet1.ItemData(lbSet1.NewIndex) = fileInd
```

```
        End If
        fileInd = fileInd + 1
        Get #13, fileInd, pmRec
    Wend
End Sub
Sub refreshlbSet2 ()
    lbSet2.Clear
    fileInd = 1
    Get #12, fileInd, prRec
    While Not EOF(12)
        If prRec.ElementIndex = curProc Then
            Get #6, prRec.ResourceIndex, tempResourceRec
            expression = Trim$(tempResourceRec.Name)
            lbSet2.AddItem (expression)
            lbSet2.ItemData(lbSet2.NewIndex) = fileInd
        End If
        fileInd = fileInd + 1
        Get #12, fileInd, prRec
    Wend
End Sub
Sub refreshlbSet3 ()
```

```
    lbSet3.Clear
    fileInd = 1
    Get #9, fileInd, psRec
    While Not EOF(9)
      If psRec.ElementIndex = curProc Then
        Get #5, psRec.SkillIndex, tempSkillRec
        expression = Trim$(tempSkillRec.Name)
        lbSet3.AddItem (expression)
        lbSet3.ItemData(lbSet3.NewIndex) = fileInd
      End If
      fileInd = fileInd + 1
      Get #9, fileInd, psRec
    Wend
End Sub
Sub refreshlbSet4 ()
    lbSet4.Clear
    fileInd = 1
    Get #11, fileInd, pyRec
    While Not EOF(11)
      If pyRec.ElementIndex = curProc Then
        Get #3, pyRec.EmployeeIndex, tempEmployeeRec
```

```
        expression = Trim$(tempEmployeeRec.Last) + ", " +
Left$(tempEmployeeRec.First, 1) + tempEmployeeRec.MI +
Str$(ProcedureEmployeeRec.EmployeeIndex)
        lbSet4.AddItem (expression)
        lbSet4.ItemData(lbSet4.NewIndex) = fileInd
      End If
      fileInd = fileInd + 1
      Get #11, fileInd, pyRec
   Wend
End Sub
Sub updateEF ()
   Get #2, curProc, tempElementRec
   efProcedure.Text = Trim$(tempElementRec.Procedure)
End Sub
Sub Form_Load ()
   WINfmManual = True
   Call refreshlbList
   lbList.Selected(0) = True
   Call lbList_Click
End Sub
Sub Form_Unload (Cancel As Integer)
```

```
    WINfmManual = False
End Sub
Sub lbList_Click ()
    curProc = lbList.ItemData(lbList.ListIndex)
    Call updateEF
    Call refreshlbSet1
    Call refreshlbSet2
    Call refreshlbSet3
    Call refreshlbSet4
End Sub
Sub pbCancel_Click ()
    Unload fmManual
End Sub
Sub pbProceed_Click ()
Dim eol As String, Text As String, ind As Integer
    Open "MANUAL.TXT" For Output As #14
    eol = Chr$(13) + Chr$(10)
    Print #14, "Quality Manual for "; CurrentFile; eol; eol
    Print #14, CompanyRec.Name
    Print #14, CompanyRec.Address
```

```
   Print #14, Trim$(CompanyRec.City); ", "; CompanyRec.State; "  ";
CompanyRec.Zip
   Print #14, "("; CompanyRec.AreaCode; ") "; CompanyRec.Number; " - ";
CompanyRec.Extension; eol; eol
   listInd = 0
   While listInd < lbList.ListCount
      lbList.Selected(listInd) = True
      curProc = lbList.ItemData(listInd)
      Text = ""
      Get #2, curProc, tempElementRec
      Text = Text + Trim$(Str$(tempElementRec.Number)) + "." +
Trim$(Str$(tempElementRec.Number1)) + "." +
Trim$(Str$(tempElementRec.Number2)) + "  " + Trim$(tempElementRec.Name)
+ eol
      Text = Text + Trim$(tempElementRec.Procedure) + eol + eol
      If lbSet1.ListCount > 0 Then
         ind = 0
         Text = eol + Text + "Verification methods:" + eol
      End If
      While ind < lbSet1.ListCount
         Get #13, lbSet1.ItemData(ind), pmRec
```

```
    Get #4, pmRec.MethodIndex, tempMethodRec
    Text = Text + Trim$(tempMethodRec.Name) + eol
    ind = ind + 1
Wend
If lbSet2.ListCount > 0 Then
    ind = 0
    Text = eol + Text + "Resource required:" + eol
End If
While ind < lbSet2.ListCount
    Get #12, lbSet2.ItemData(ind), prRec
    Get #6, prRec.ResourceIndex, tempResourceRec
    Text = Text + Trim$(tempResourceRec.Name) + eol
    ind = ind + 1
Wend
If lbSet3.ListCount > 0 Then
    ind = 0
    Text = eol + Text + "Skills needed:" + eol
End If
While ind < lbSet3.ListCount
    Get #9, lbSet3.ItemData(ind), psRec
    Get #5, psRec.SkillIndex, tempSkillRec
```

```
            Text = Text + Trim$(tempSkillRec.Name) + eol
            ind = ind + 1
        Wend
        If lbSet4.ListCount > 0 Then
            ind = 0
            Text = eol + Text + "Employees responsible:" + eol
        End If
        While ind < lbSet4.ListCount
            Get #11, lbSet4.ItemData(ind), pyRec
            Get #3, pyRec.EmployeeIndex, tempEmployeeRec
            Text = Text + Trim$(tempEmployeeRec.Last) + ", " +
Trim$(tempEmployeeRec.First) + " " + tempEmployeeRec.MI + eol
            ind = ind + 1
        Wend
        Print #14, Text
        listInd = listInd + 1
    Wend
    msg$ = "Quality manual has been created as 'MANUAL.TXT' in
'C:\ISO9000\" + CurrentFile + "'.  Use an editor to view and print."
    userResponse% = MsgBox(msg$, 64, "Information")
    Close #14
```

```
End Sub

Sub pbRefresh_Click ()

   Call refreshlbList

End Sub
```