IMPROVING THE SOFTWARE DEVELOPMENT PROCESS AND SOFTWARE PERFORMANCE THROUGH THE APPLICATION OF SYSTEMS ENGINEERING PRINCIPLES INCLUDING CONCURRENCY

by

Brett S. Barthelmess

Project and Report submitted to the Faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Systems Engineering

APPROVED:

L. Kenneth Harmon

B. S. Blanchard

Rajiv Kapur

DECEMBER, 1993

Blacksburg, Virginia
IMPROVING THE SOFTWARE DEVELOPMENT PROCESS AND SOFTWARE PERFORMANCE THROUGH THE APPLICATION OF SYSTEMS ENGINEERING PRINCIPLES INCLUDING CONCURRENCY

by

Brett S. Barthelmess

Committee Chairman: L. Kenneth Harmon
Systems Engineering

(ABSTRACT)

The ABC Company has been developing human resource administration software for over 10 years. Although this software is not overly technical and ABC has sufficient resources and tools for software development, one significant problem continues to be encountered.

The primary problem faced by ABC software is properly defining and fulfilling user expectations and requirements. This problem can be measured primarily through the growing number of change and maintenance requests from clients upon installation. Numerous hours are spent in these maintenance activities which result in budget overruns and lower profits (or losses).

The purpose of this project was to develop recommendations for alleviating the problem as described above. A study of the problem and possible approaches resulted in the hypothesis that a re-engineering of ABC's software development process to more rigorously follow a systems engineering process, including concurrent engineering (CE) techniques, would help alleviate the problem.

The project methodology was implemented on two similar software development efforts -- one using ABC's current development approach and the other utilizing a systems engineering approach with CE principles. A systems engineering evaluation tool, the objectives matrix, was used to measure each projects' performance in order to test the hypothesis.
The results of the evaluation show that the software development effort utilizing a systems engineering approach with CE techniques will help alleviate the problems encountered. The expected benefits of CE -- allowing analysts, programmers, technical specialists, and users to work simultaneously on a project -- were achieved.

These benefits provide the basis for the recommendations to ABC to continue to employ a systems engineering process which incorporates CE principles within its software development projects and expand its use to larger projects.
ACKNOWLEDGEMENTS

I wish to extend thanks to many people who have contributed to this project. Professor Ken Harmon has provided me with tremendous insight and helped me stay focused on the core of the project. My fellow work colleagues donated many hours to benefit my project as well. I hope the efforts of this project will benefit them and their software development efforts.

Most of all, I would like to acknowledge my parents for the educational opportunities and guidance they provided, and my wife, Angie, and daughter, Erin Kate, who sacrificed much and encouraged me every step of the way.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>SECTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0 PROBLEM DEFINITION</td>
<td>1</td>
</tr>
<tr>
<td>2.0 PROJECT APPROACH</td>
<td></td>
</tr>
<tr>
<td>2.1 Project Objectives</td>
<td>2</td>
</tr>
<tr>
<td>2.2 Project Methodology</td>
<td>2</td>
</tr>
<tr>
<td>3.0 SYSTEMS ENGINEERING PROCESS</td>
<td></td>
</tr>
<tr>
<td>3.1 General Concepts</td>
<td></td>
</tr>
<tr>
<td>3.1.1 Description</td>
<td>8</td>
</tr>
<tr>
<td>3.1.2 Typical SDLC Phases</td>
<td>8</td>
</tr>
<tr>
<td>3.1.3 Incorporation of Concurrency within the Systems Engineering</td>
<td>9</td>
</tr>
<tr>
<td>3.2 ABC’s Software Development Process</td>
<td></td>
</tr>
<tr>
<td>3.2.1 Description</td>
<td>10</td>
</tr>
<tr>
<td>3.2.2 Weaknesses</td>
<td>13</td>
</tr>
<tr>
<td>3.3 Comparison of ABC’s Development Process and Systems Engineering Process Leading to Hypothesis</td>
<td>14</td>
</tr>
<tr>
<td>4.0 CONCURRENT ENGINEERING</td>
<td></td>
</tr>
<tr>
<td>4.1 General Concepts</td>
<td></td>
</tr>
<tr>
<td>4.1.1 Overview</td>
<td>16</td>
</tr>
<tr>
<td>4.1.2 Key Implementation Factors</td>
<td>18</td>
</tr>
<tr>
<td>4.1.3 Tools for Implementation</td>
<td>19</td>
</tr>
<tr>
<td>4.1.4 Potential Difficulties</td>
<td>19</td>
</tr>
<tr>
<td>4.1.5 Case Studies</td>
<td>20</td>
</tr>
<tr>
<td>4.2 Application within Software Development</td>
<td></td>
</tr>
<tr>
<td>4.2.1 Fundamentals and Tools</td>
<td>20</td>
</tr>
<tr>
<td>4.2.2 Team Formation</td>
<td>21</td>
</tr>
<tr>
<td>4.2.3 Critical Factors for Successful Implementation</td>
<td>23</td>
</tr>
<tr>
<td>4.2.4 Requirements Engineering</td>
<td>23</td>
</tr>
<tr>
<td>4.2.5 User Involvement</td>
<td>24</td>
</tr>
<tr>
<td>SECTION</td>
<td>PAGE</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>5.0 Implementation of Software Development Projects</td>
<td></td>
</tr>
<tr>
<td>5.1 Software Description</td>
<td>27</td>
</tr>
<tr>
<td>5.2 Comparative Analysis of Development Processes</td>
<td>31</td>
</tr>
<tr>
<td>5.3 Traditional Development Process (Project A)</td>
<td></td>
</tr>
<tr>
<td>5.3.1 Project Participants and Initial Interaction</td>
<td>31</td>
</tr>
<tr>
<td>5.3.2 Development Phases</td>
<td>32</td>
</tr>
<tr>
<td>5.3.3 User Involvement</td>
<td>37</td>
</tr>
<tr>
<td>5.4 Development Process with CE Principles (Project B)</td>
<td></td>
</tr>
<tr>
<td>5.4.1 Team Formation and Interaction</td>
<td>37</td>
</tr>
<tr>
<td>5.4.2 Development Tools based on Concurrent Engineering</td>
<td>41</td>
</tr>
<tr>
<td>5.4.3 Modified Development Process</td>
<td>43</td>
</tr>
<tr>
<td>5.4.4 Emphasis on Requirements and Users</td>
<td>46</td>
</tr>
<tr>
<td>6.0 EVALUATION OF PROJECT PERFORMANCE</td>
<td></td>
</tr>
<tr>
<td>6.1 Comparison and Selection of Evaluation Methodology</td>
<td>48</td>
</tr>
<tr>
<td>6.2 Data Criteria, Gathering, and Analysis</td>
<td>50</td>
</tr>
<tr>
<td>6.3 Application of Evaluation Tool - Objectives Matrix</td>
<td>54</td>
</tr>
<tr>
<td>7.0 RESULTS</td>
<td></td>
</tr>
<tr>
<td>7.1 Conclusions</td>
<td>58</td>
</tr>
<tr>
<td>7.2 Methodology Analysis</td>
<td>59</td>
</tr>
<tr>
<td>8.0 RECOMMENDATIONS</td>
<td>61</td>
</tr>
<tr>
<td>9.0 REFERENCES</td>
<td>63</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1 - Project Methodology</td>
<td>7</td>
</tr>
<tr>
<td>Figure 2 - Systems Engineering Waterfall Model</td>
<td>15</td>
</tr>
<tr>
<td>Figure 3 - Basic Concurrent Engineering Model</td>
<td>17</td>
</tr>
<tr>
<td>Figure 4 - ABC Enrollment Administration System Overview</td>
<td>30</td>
</tr>
<tr>
<td>Figure 5 - Project A Milestone Diagram</td>
<td>36</td>
</tr>
<tr>
<td>Figure 6 - Project B Milestone Diagram</td>
<td>40</td>
</tr>
<tr>
<td>Figure 7 - Modified SDLC Diagram</td>
<td>47</td>
</tr>
</tbody>
</table>

vii
## LIST OF TABLES

<table>
<thead>
<tr>
<th>TABLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1 - Data Collected from Projects</td>
<td>53</td>
</tr>
<tr>
<td>Table 2 - Objectives Matrix for Project A</td>
<td>56</td>
</tr>
<tr>
<td>Table 3 - Objectives Matrix for Project B</td>
<td>57</td>
</tr>
</tbody>
</table>
1.0 PROBLEM DEFINITION

The ABC Company has been developing Human Resource Administration software for over ten years. However, problems exist that have been plaguing ABC's software developers and management throughout this period. It has been ABC's experience that many projects have been either over budget, behind schedule, difficult to maintain, or deficient in meeting users' expectations and requirements.

The primary problem faced by ABC software is inadequately fulfilling user expectations and requirements. This problem is encountered without exception upon the release of the software for client use. A lack of understanding of user needs during the initial phases of software development contributes greatly to this problem.

The impact of this problem on development projects can be seen in the numerous change and maintenance requests for clients upon installation. Many hours are spent in completing these maintenance tasks. The result is increased costs which erode profits or cause losses to be incurred for the project.

Other problems encountered are projects that fall behind schedule and software that is difficult to maintain. Maintainability and reusability of software code are important to reduce hours spent in the maintenance phase of the life cycle. Again, this is often the area that determines the difference between profits and losses.

The goal of this project is to develop recommendations for alleviating the problems described above. The completion of an in-depth study such as this project can be used to show ABC management the importance of improving their software development process. The results can show quantifiably the effect that implementing a new development process may have on software quality and project profitability.
2.0 PROJECT APPROACH

2.1 Project Objectives

As stated above, the goal of this project is to generate recommendations for ABC to help eliminate the problems they are encountering in their software development efforts.

Specific objectives include:

- improve the quality of software developed in terms of meeting customer expectations and requirements;
- reduce the costs of development by reducing the amount of re-work and iterations through the design and development phases;
- improve the maintainability of the computer code generated and thus extend the useful life of software systems;
- show that code reusability and efficiency can be increased through improving the development process and better understanding of user requirements.

2.2 Project Methodology

The approach undertaken for this project in itself followed the systems engineering process. In essence, what is taking place is a re-engineering of ABC's software development life cycle. The methodology used to execute this project follows an eight step approach which is depicted in figure 1 and explained in the sections below.

2.2.1 Defining the Problem

Changes to the system life cycle approach, in ABC's case, are brought about in response to the identification of a need that develops as a result of a problem which needs fixing. ABC has experienced several problems with their software development process as explained above in section 1.0. Assessing the situation and arriving at the actual problem areas is the first step of this project.

It should be clarified that the purpose of this project
was focused on improving the software development process and not focused on the development of any particular software itself.

2.2.2 Defining Project Scope and Objectives

After the problem has been specified, the next step is to define the scope and objectives of the project to address the problem. These objectives are outlined above in section 2.1.

2.2.3 Research of Systems Engineering Concepts

The next step is to perform research in the area of systems engineering to provide a basis for evaluation of an hypothesis.

The systems engineering process, detailed in section 3.0, used as a foundation for comparison in this project is based on the model as presented in the curriculum within Virginia Tech's Systems Engineering program. The principles of this model were found to be consistent throughout the articles and references researched in the project.

2.2.4 Analysis of ABC's Development Process

A review of ABC's current approach to software development is also needed in order to further develop an understanding of possible sources of the problems being encountered. This is also used as a foundation for generating hypothetical solutions. Section 3.2 outlines ABC's current development process in detail.

2.2.5 Comparison of Systems Engineering Process and ABC's Development Process

At this point, a thorough study of the problem is needed to compare ABC's efforts with a base line methodology of life cycle processes in order to produce a definitive hypothesis. The analysis of ABC's current process and research of systems engineering allow for a comparison to identify areas of possible improvement. Section 3.3 presents this comparison in detail and the logic behind generating the hypothesis tested in the following steps.
as a solution to ABC's problems.

The hypothesis arrived at is to reach a solution through a more thorough application of systems engineering principles including those of concurrent engineering within ABC's software development life cycle. The implementation of concurrency principles and techniques within the development process had not been tried at ABC.

Further research of basic concurrent engineering concepts and techniques, detailed in section 4.0, is performed to gather a solid understanding of how they may be incorporated into the systems engineering process. A general understanding of concurrent engineering is needed before developing particular strategies to be applied within software development. Specific tools and techniques used in the implementation of these principles are presented in detail in section 4.0 as well.

Concurrent engineering efforts at large corporations such as Boeing, AT&T, and Westinghouse provide insight on how these principles have been applied. Specific tools for software development at organizations such as Rockwell and Computer Sciences Corporation (CSC) furnish information more specific to the software development problems faced by ABC.

2.2.6 Implementation of Solution on Software Development Projects

Once the problem definition, a proposed solution, and an understanding of the general principles needed to implement a solution are complete, planning for the implementation of the solution is needed. A project plan is developed to detail a methodology to compare and evaluate the use of systems engineering principles including concurrency to the software development process.

The project methodology is based on the comparison of two similar software development efforts. Project A is based on ABC's current software development process. Project B is based on a more thorough systems engineering process with concurrent engineering principles and tools. The detailed project discussion is presented in section 5.0.

Both projects involve the development of software for a human resources administration system and were chosen to
provide a good basis for comparison because of their similarities.

The importance of choosing two similar projects cannot be understated in order to provide adequate justification that any differences in the results of the development processes are, in fact, due to the implementation of concurrency within the systems engineering process.

The two projects were selected because the following factors were consistent in both:

- each project client are of similar size (1,500 to 1,800 employees);
- each client’s employee benefit programs (which the software is based upon) contain the same number of benefit plans and options;
- each client’s organization structure is similar providing for similar decision making processes;
- the availability of the client’s decision makers is similar in each project;
- the client’s knowledge of the benefit software is identical since it was the initial implementation for each client;
- there are a like number of users of the software (8 to 10 users);
- the ABC project team members are the same for each project;
- the time frames for completion (18 weeks) and the budgets (500 hours) are the same for each project;
- the technical platforms are the same for each project (PC-based applications running as multi-user versions on Novell networks).

Each project was executed based on the two development processes. Data was collected throughout and at the end of each project to be used in the evaluation.
2.2.7 Evaluation of Project Performance

After the specific projects were selected, evaluation criteria and methodology were needed to measure the effectiveness of each project. Evaluation criteria were developed based on the project objectives.

A rigorous systems engineering measurement technique was needed that takes into consideration the criteria and arrives at a quantitative result of how the projects compare. These criteria and the selection process of choosing an evaluation methodology are discussed in detail in section 6.0

2.2.8 Interpretation of Results and Recommendations

The final portion of the project is to critique and deliver findings of the hypothesis of applying systems engineering and concurrency principles to software development. The data collected and performance evaluation results are interpreted and discussed in detail.

As stated above, the primary purpose for executing this project was to provide ABC with recommendations on improving its software development process. Recommendations are made on the continuation of applying these principles to ABC and software practitioners in general. The results of the performance evaluation and recommendations are presented in sections 7.0 and 8.0 respectively.
Figure 1 - Steps of Project Methodology
3.0 SYSTEMS ENGINEERING PROCESS

3.1 General Concepts

3.1.1 Description

Systems engineering can be defined, according to Blanchard and Fabrycky, as the application of science and technology to transform a need into a process to fulfill that need, integrate related technical parameters, and integrate all specialties into a total engineering effort. [1]

Systems engineering is not considered as an engineering discipline per se such as mechanical, electrical, or any other technical specialty. It is a process followed to develop a system from the identification of a need through production and implementation of the system for use by a customer. The process concentrates on carrying out a series of steps in a logical, top-down manner in order to efficiently develop a system. Systems engineering became necessary to ensure that all system elements are considered timely and properly by specialists in conventional engineering areas.

For purposes of this project, systems engineering is defined as an approach to software projects which follow a formalized System Development Life Cycle methodology. The emphasis is on a structured approach which generally follows a serial path through the many phases of the development cycle with feedback occurring throughout.

3.1.2 Typical SDLC Functions

Although there may be variations of the phases of a typical systems engineering life cycle, the following phases are generally applicable in one form or another:

System Planning This phase generally includes market analysis, completion of feasibility studies, and advanced planning to determine and define a specific need.

System Research Research is necessary to discover facts and theories which will result in knowledge necessary to lead system development.

System Design This phase is made up of the conceptual design, preliminary systems design, and detail design.
These processes result in specifications that can be used to produce the desired system configuration.

**Production/Construction** Production or construction begins at the point where system design is frozen. Materials and personnel resources are combined to provide the system output effectively and efficiently.

**System Evaluation** A periodic evaluation is performed to ensure that the system outputs or results conform with the established requirements.

**System Use and Logistic Support** This phase includes the actual use of the system, implementation of system modifications and improvements, provision of logistic support requirements for ongoing operation, and phase-out and disposal of the system.

As mentioned earlier, feedback occurs throughout all phases of the life cycle and plays a critical role throughout especially where phases tend to overlap. For example, the design, production, and evaluation phases can be considered as an iterative loop within itself which is supported by feedback and corrective action.

The entire system engineering process is iterative in nature and feedback is relied upon throughout to ensure convergence. [1]

### 3.1.3 Incorporation of Concurrency within the Systems Engineering Process

As previously mentioned, the phases outlined here are generally followed in a sequential order. Blanchard above, however, emphasized how phases overlap and run iteratively and concurrently. Fabrycky, also, introduces the concept of concurrency within this framework of systems engineering process. [2] This presentation of concurrent engineering principles was a contributing factor leading to the hypothesis of applying these principles as a solution to ABC’s software development process.

Concurrent engineering principles place an even greater emphasis on the formation of multi-disciplinary teams which are formed at the onset of a project. All members of the team are expected to remain heavily involved throughout all phases of development. These teams allow
planners, designers, engineers, customers, and suppliers
to work together simultaneously as a product team and to
share ideas, information, and feedback throughout the
entire life cycle. An objective is to accelerate all
phases of product development by starting them as soon as
possible and running them concurrently as opposed to
serially. [3]

Concurrent engineering concepts, tools, and techniques
are discussed at great length below in section 4.0.

3.2 ABC’s Software Development Process

3.2.1 Description

As stated earlier, ABC’s formalized development process
is intended to be an example of a systems engineering
approach to software development. This formalization is
represented through a SDLC manual which outlines the
phases, outputs, and review points that a software
development effort should follow.

ABC’s software life cycle model is represented as a
series of phases with validation and feedback at each
stage. The ABC SDLC is divided into eight (8) phases.
The development process described in each phase is
required for any software development project.

The phases, as executed by ABC during development
projects, are generally considered to be sequential and
interdependent. Decisions reached and products completed
in one phase are required, in subsequent phases, to
complete successful project development according to
ABC’s SDLC plan.

The eight phases of the SDLC are described in the
paragraphs below:

Planning Phase. An identified system need is reviewed
for feasibility and appropriateness. The requirement may
be the development of a new system or modification of an
existing system. If the requirement is to be satisfied,
the development approach, tasks, resource requirements,
project schedules, feasibility studies, cost/benefit
analysis, and acquisition plans are developed.

Requirements Definition Phase. User needs are analyzed,
and user requirements are formally defined. Functional,
data, system performance, security, portability, and maintainability requirements are detailed. All requirements are defined to a level of detail sufficient for systems design to proceed.

**External Design Phase.** The external physical characteristics of the system are designed during this phase. The operating environment is established, major subsystems and their inputs and outputs are defined, and processes are allocated to resources.

**Internal Design Phase.** The internal physical characteristics of the system are designed. Subsystems defined during the External Design Phase are used to create a detailed structure of the system. Each subsystem is partitioned into one or more design units. Detailed logic specifications are prepared for each module.

**Development Phase.** The detailed specifications produced during the Internal Design Phase are translated to an executable system. Software is unit tested, integrated, and retested in a systematic manner. Hardware is assembled and tested.

**Test Phase.** Validation of the functional requirements, as defined in the Functional Requirements Document, occurs to ensure they are satisfied by the developed or modified system and that there are no adverse effects on the overall process.

**Implementation Phase.** This phase is initiated after the system has been tested and accepted by the user. The system or system modifications are installed and made operational in a production environment. The phase continues until the system is operating in production in accordance with the defined user requirements.

**Maintenance Phase.** System operation is ongoing. The system is monitored for continued performance in accordance with user requirements, and needed system modifications are incorporated. This phase continues until the system is retired from use.
3.2.2 Weaknesses

It may appear difficult to question the validity of ABC's approach to software development since it has developed a formalized approach, at least on paper. In reality, however, the most glaring weakness discovered was the lack of a disciplined approach because the SDLC was rarely followed in real development efforts.

Even if followed, ABC's approach was usually very sequential in proceeding from one phase to the next without iterations or feedback. As a result, this approach cannot be considered a systems engineering approach at all since a critical feature of the systems engineering process is feedback and corrective action throughout the life cycle.

ABC was developing software through a short-term "design now, fix later" philosophy which is common within the software industry. This philosophy can further increase chances of poor software quality because of the dynamic nature of user requirements. These requirements can rarely be considered definitive at any one time and without the necessary feedback loop there is no facility for updating user requirements. [4] This is the root of the main problem ABC faces as outlined initially.

Another weakness of ABC's process is that the segregation of phases leads to division of specialties. As these specialties develop the development process follows the serial, "over the wall", approach. Requirements are handed off to become specifications which are in turn tossed "over the wall" to the programmers. [5]

This process inevitably leads to rework. When changes are required, the development process is required to back up to previous phases in serial development usually after implementation has already occurred. Often, only critical, "make or break" changes are made - because it is viewed that it is better to accept the marginal product than to jeopardize proven performance.

Rework also often occurs because ABC's project teams lack the management skills, resources, tracking systems, or communication lines to identify and report deficiencies. Thus, the emphasis is on post-development improvement rather than improving the software development process itself which will yield higher quality software.
Figure 2 displays a traditional software development model, sometimes referred to as the "waterfall" model, which is a representation of ABC’s software development life cycle. [6] This diagram demonstrates the lack of sufficient feedback and corrective action throughout the development life cycle. Feedback was generally limited back one phase at a time. This "one-step" up feedback showed the rigidity of ABC’s development process which required significant rework and increases iterations through the phases.

3.3 Comparison of ABC’s Development Process and Systems Engineering Process Leading to Hypothesis

As previously, stated ABC lacks the discipline of a pure systems engineering process because of the lack of feedback throughout the life cycle. This is especially true in the early phases of the life cycle where requirements definition and preliminary design work is performed.

Customer requirements are extremely volatile. It is naive to believe that requirements can be developed as if users know what they need (and want) in advance. Most requirement specifications are incremental and ever evolving. At no time can a user specify the definitive set of requirements. [5]

Furthermore, few projects had software developers with complete knowledge about the user’s work. This inability to visualize usage scenarios can lead to more difficulty in defining user requirements.

To listen to ABC software managers, one would conclude that they felt the formalized process was in place but they knew something was not working properly. The above description of requirements volatility was recognized and they felt the focus of any new process needed to be on the front-end phases of the life cycle. The managers felt the root of the problem was defining requirements and also felt that user involvement was a key.

However, trying to convince them that the development process in general was inadequate would not work. They felt introducing a systems engineering process was not significant because they already had one. A new paradigm was needed.
This need of a new paradigm lead to the hypothesis of incorporating concurrent engineering principles through the systems engineering process applied within software development. Concurrent engineering was a "new" concept that seemed to provide a mechanism for improving the front-end phases of the life cycle and providing a valuable mechanism for feedback. It also provided a philosophical change that was needed to convince ABC software managers to attempt a change in their development process.

Other possible solutions to ABC’s software development problems may have been generated from various areas. Implementation of computer-aided software engineering (CASE) tools or the hiring of additional experienced software designers may have been possible recommendations to ABC.

These alternatives were not pursued because they concentrate on improving technological or human resources which were not deficient within ABC’s software development process. A solution was needed to improve the development process itself and better define user requirements and how to satisfy them.
Figure 2 - ABC's Software Development Life Cycle Model
4.0 CONCURRENT ENGINEERING

4.1 General Concepts

4.1.1 Overview

Concurrent engineering is becoming a modern organizational buzzword in the U.S. Many manufacturing companies believe that their survival depends on its implementation. In concurrent engineering, the key ingredient is the development of multi-functional teams which have heavy interaction throughout all phases of the life cycle. People from many departments collaborate over the life of a product - from idea to obsolescence - to ensure that it reflects customers' needs and desires and reaches market quicker.

Marketing, engineering, and manufacturing, for example, work together from the outset to anticipate problems and bottlenecks and to eliminate them early on. In so doing, delays in bringing the product to market and costly failures in service are avoided. Accounting and purchasing may also have input and help to ensure low product cost and reliable supplies of parts and materials. Figure 3 shows a typical development cycle under concurrent engineering.

The main results to be gained from concurrent engineering efforts are:

- shorter time for the product introduction cycle,
- improved design quality,
- reduced design iterations, and
- shorter production time.

As stated above, concurrent engineering starts with multi-functional teams. But it also includes concepts and techniques such as: design for manufacture, continuous process improvement, total quality management, and quality function deployment. Integrating all of these features, concurrent engineering strives to create successful new products by bringing together as early as possible in the development cycle a company's resources and its experience in design, development, marketing, manufacturing, service, and sales. [7]

Most importantly to some companies, concurrent engineering can shorten the overall product-development life cycle because the steps along the way are handled in
Figure 3 - Typical Concurrent Engineering Life Cycle - Product Design Draws on Various Disciplines
parallel instead of in series. The time to market is also improved because concurrent engineering reduces the number of product iterations.

Other companies view the concepts not only in their ability to provide shorter development times. Hume sees the greatest value in concurrent engineering is improved product quality. That is, a product which has a more robust specifications, design, and implementation.

The roots of concurrent engineering trace back over a decade, to when Japanese companies stunned their U.S. rivals by producing products of ever-higher quality at lower and lower prices. Even more shocking were the reports of study groups back from Japan. These groups found production lines operated at quality levels 1,000% better than comparable factories in the U.S. This stemmed largely from the fact that the Japanese — emulating the way American companies operated prior to World War II — did not have separate design and manufacturing functions. Their product engineers were equally adept at both. And, they tended to engineer quality into the manufacturing process, while the U.S. relied on assembly-line inspections to weed out defects.

Leading U.S. companies knew they had to make quantum leaps in quality and cycle times. They discovered that all things lead to a few keys to competitiveness - time to market and quality, plus flexibility in responding to changing customer needs and market forces.

4.1.2 Key Implementation Factors

As with any new concept, implementation of new techniques and methodologies can be difficult. Switching from a sequential development flow to a totally concurrent development environment is becoming much more difficult than many companies expected.

Most companies searching for total concurrent engineering find it best to implement the principles in a series of incremental steps. Such incremental steps allow many design teams to operate with a degree of concurrency and produce some of the benefits promised by concurrent engineering. [8]

Implementing concurrent engineering is similar to putting a quality program into effect. It takes commitment by
the entire organization, from top to bottom. It takes education at all levels so that everyone speaks a common language. It takes repetition; managers and engineers must constantly impress upon their peers, and those above and below them, their commitment to concurrent engineering. [9]

Another key implementation factor, and perhaps the most important, is to establish formal and effective lines of communication between the team members. The best solution may be logistical in locating the team members near each other. When a project starts up at NCR, engineers play musical cubicles, so the specialists involved in design, software, hardware, purchasing, manufacturing, and field support all work side by side and compare notes constantly. This encourages ongoing communication and fosters a group spirit of responsibility for the development process. [10]

4.1.3 Tools for Implementation

A major tool used in the concurrent engineering development cycle is prototyping. Prototyping can provide an opportunity to reduce design time for products and can be used as a strategic tool in the concurrent engineering environment.

The goal of using prototyping as a development tool is to eliminate the need to freeze system requirements before design begins. By using a prototype, users can get an "actual feel" of the system, since the interactions with the prototype can enable them to better understand the requirements of the desired system. [11]

Prototyping is an attractive idea for complicated and large systems for which there is no manual process or existing system to help determine the requirements. In such situations, letting the user work with the prototype provides invaluable and intangible inputs which help in determining the system requirements.

4.1.4 Potential Difficulties

Although the rewards of successfully implementing concurrent engineering can be tremendous, considerations must also be made for likely problems to be encountered. Like any new concept introduced in organizations, there
may be a strong resistance to change.

It is important to provide the teams working under these new concepts with the resources and training needed to be able to work together effectively. Boeing found that forming "togetherness" can be difficult. Some teams expressed that they lacked needed resources or skills, some people were adamantly opposed at the start to sharing data, and some team leaders were inexperienced at running interdepartmental meetings. Working together requires much management, care, and nurturing of individuals. [10]

4.1.5 Case Studies

There are many examples that can be reviewed to display the successes of concurrent engineering on particular products and industries. Below are a few examples of how these principles are being used:

- Boeing Commercial Airplane Group is using concurrent engineering techniques to develop the giant 777 transport. They expect to release design drawings a year and a half earlier than happened with the 767 and save as much as 20% of the 777's estimated $4 billion to $5 billion development cost. [10]

- AT&T latched onto the concept when it redesigned its main electronic phone-switching systems. The total "cycle time" from conception to production was trimmed by more than half from the normal three years. Additionally, manufacturing defects plunged as much as 87%. [12]

- John Deere & Company began using concurrent engineering in the late 1980s and has slashed the previous seven-year cycle time for construction and forestry equipment by 60%, thus, saving 30% of the usual development costs. [12]

4.2 Application within Software Development

4.2.1 Fundamentals and Tools

Software organizations are looking beyond patching up the old ways of doing business and are re-engineering their
software development processes to develop more effective ways of working. One of the ways they are doing this is using concurrent engineering principles and processes. These techniques can be powerful in improving the quality and speed of software design and development.

Many tools and techniques are being used in software development to apply concurrent engineering principles. Joint Application Development (JAD) is a relatively new technique that provides a mechanism for managing the politics of a project, increasing user commitment and involvement through consensus building and objective and unbiased leadership. JAD focuses on improving the front-end life cycle phases and documentation. The technique is centered around workshops in which eight to 20 team members make decisions through the consensus building leadership of a specially trained and unbiased facilitator who is not a member of the project team. [13]

JAD techniques are often integrated with other tools such as information engineering (IE) methodology and CASE tool automation to aid in the new systems development environment.

Software prototyping, another often-used tool, is similar to engineering prototypes in that they are intended to be working models of software systems or part of a system. Because software systems can be as complex as more traditionally engineered systems, it makes sense that prototyping has a useful role in software development.

Prototyping can help considerably when the user's requirements for the system are not clearly understood or are difficult to document. This situation is quite common, and the traditional life cycle model does not adequately accommodate this uncertainty about system needs. Prototyping eases the communications between developers and users by allowing their discussions to focus on real behavior and actual user experiences in exercising the prototype.

4.2.2 Team Formation

The beginning step of implementing concurrent engineering in a software development environment is the formation of a multi-functional team.

In a manufacturing environment, the "business team" is
usually comprised of representatives from design engineering, manufacturing, customer service, finance, and marketing. The team is generally managed by a "product boss", typically a senior design engineer, who has the ultimate responsibility to the corporation for delivering the product on time, and at cost. The organizational structure of the business team is predominantly a matrix organization. [14]

Within a software development environment, the team formation parallels that of manufacturing. Concurrent engineering techniques deliver results only if the right people are involved and committed to their roles and responsibilities in the process. A typical software development multi-functional team may include the following:

- **Project Manager** - The person leading the project team and is operationally responsible for delivering the project results. Depending on the organizational culture, the project manager may be a systems person, a user, or a team made up of one person from each "side of the house."

- **Facilitator** - A consultant to the project manager who guides team meetings and directs discussions using behavioral and group dynamics techniques within the structure of soundly engineered methodology.

- **Users** - The users of the software systems have a primary and vital decision making role in the development process. Users can be represented in two ways. Some users must be "representative," in that they are responsible for the strategic direction of the operation from a commonization or standards view. Other users must be "real end users," those people who must live with the project results on a day to day basis.

- **Systems Experts** - Have a direct decision making role in the development process to ensure that the project is realistic, can be delivered when the users require it, and takes advantage of the available technology most effectively.

- **Outside Technical Experts** - Provide the business expertise needed for the particular project. External technical experts play more of an advisory
role in the development process. [13]

4.2.3 Critical Factors for Successful Implementation

As stated earlier, major change within organizations often meets great resistance. This is also true when companies introduce concurrent engineering principles for the first time. Organizations committed to improving software development speed and quality must tackle the issues of organizational political and cultural change.

This includes the commitment to the use of concurrent engineering tools and methodology. Resources must be trained in new team building principles and in leadership and facilitator techniques. Effort and patience must be given to helping people understand that a new approach to software development is being created and that project team membership will be different in this new systems environment.

Corbin has identified other critical success factors in the formation of teams to implement concurrent engineering principles in software development. They are:

- Team members need to be those who "can't be spared" from their current jobs. It is essential to get the best team members available. If they won't be missed from their jobs, they aren't the right people.

- A good facilitator can be the key to success. Only a hand full of people have the right mix of Information Systems and people skills to be a successful facilitator. The facilitator must be able to keep the team on track and resolve problems quickly.

- The information systems staff should be seen, not heard. In many cases only one systems person is placed on requirements teams. The role of the analyst is to listen, not talk. Requirements are to focus on user needs, not computers. [15]

4.2.4 Requirements Engineering

The task of requirements engineering focus on the early
stages of software development. The goal is to have well
developed requirements, especially in regards to the
maintenance phase of the development life cycle. The
maintenance phase is often overlooked in software
development, but it represents up to 50% of software
system costs. Defining requirements that reduce
maintenance requests and activities can contribute
significant cost savings. [16]

Reasons for focusing on the upstream portion of the life
cycle include:

- Research on software engineering has recognized
  that the maintenance of software should be done at
  the level of specifications, rather than at the
  level of code to achieve high-quality software.
  Specifications are the closest in form to the
  user's conceptual model of the system, the least
  complex, and most localized. Accordingly, the
  process of maintenance should be viewed as one of
  redesign at the level of specifications.

- Researchers have concluded that from the
  maintenance view point of reusability, reuse of
  components at the design stage has more potential
  for payoff than reuse of code, and that design
  reuse is the only way to achieve orders-of-
magnitude increase in software development
  productivity.

4.2.5 User Involvement

The emphasis of concurrent engineering within software
development is on early and thorough user involvement
throughout the development life cycle. Users, in this
area, can often be thought of analogous to designers in
a manufacturing environment. Software users are
generally the most knowledgeable of how processes work
and what requirements need to be fulfilled.

With this knowledge, users should be included in the
development cycle and considered as analysts. For this
to become a reality, the end users must be given a means
of ensuring that the specified system matches their
intent, a means that avoids the delay and expense of a
flawed or incomplete implementation.

Gould and Lewis go so far as to create a usability
engineering model which includes: early focus on users, user participation in the design, prototyping, and empirical user testing. [17]

The first stage of their model aims at understanding the target user population and user tasks. Developers should visit a user site to gain a feel for how the software will be used. Individual user differences and variability in tasks are the two factors with largest impact on system design. The concept of "user" should be defined to include everyone whose work is affected by the product.

The next stage of the usability model is participatory design. Instead of guessing, representative users should be included as a part of the design team. Users often raise questions that others may not even dream of asking. Users, however, are not designers, so they should not be expected to come up with design ideas from scratch. However, they are very good at reacting to concrete designs they do not like or will not work in practice. To get the full benefit of user involvement, suggested system designs should be presented in a form the user can understand.

The usability model's next phase is experimental prototyping. As discussed above, this is highly recommended for early stages of the development process. In many software engineering models, most of the development time is devoted to refining various intermediate work products, and executable programs are produced at the latest possible moment.

A problem with this approach is that there is no user interface to test with real users until the last possible moment, because the "intermediate work products" do not explicitly separate the user interface in a prototype with which users can interact. Experience also shows that it is not advisable to involve the users in the design process by showing them abstract specifications documents they do not understand documents nearly as well as concrete prototypes.

Empirical user testing is the model's final phase where it is important to have the test users perform tasks representative of the eventual use indicated by the user requirements.

A final note is injected here concerning what some
information systems professionals refer to as "power users." These are users who have somewhat accomplished technical skills and often embark on creating applications on their own. Often these systems are used by others extensively but are produced through shoddy design and poor programming techniques. [18]

If the power user leaves the company, information systems is stuck supporting applications it didn't build. But out of the chaos that frequently accompanies power-user-developed systems, so come solutions. Often these applications turn into innovative, high-value solutions that solve real business problems.

According to von Hippel's research on innovations, these users often develop in unmet needs that are months or years ahead of the general market. In some cases, software manufacturers who partnered with the so-called "lead users" cut product development time to a third. It is important learn how to identify "lead users" and learn from their efforts.
5.0 IMPLEMENTATION OF SOFTWARE DEVELOPMENT PROJECTS

5.1 Software Description

The Enrollment Administration software developed by ABC is used by their clients to administer flexible benefit programs for their employees. Flexible benefit programs allow employees to select options in many different benefit areas (i.e. medical, dental, life insurance, etc.) according to their family's needs. In essence, employees can select a "custom" benefit plan with the advantages of pre-tax deductions for any employee costs resulting in tax savings.

The software is customized for each client to handle the specifics of their employee benefit plans and requires annual updates to accommodate any changes in plans. The software is made up of two basic subsystems:

- The annual enrollment subsystem is used during a specific enrollment period so employees have the opportunity to change their benefit coverage annually. This annual enrollment opportunity is required by federal regulations.

- The ongoing administration subsystem is used to enroll new hires and make changes for employees who have allowable status changes (marriage, birth of child, etc.) throughout the year.

The Enrollment Administration system performs the following functions:

- **Data Collection/Edit**
  Collect and edit employee and dependent data required for the production of personalized enrollment materials. Each employee receives an information packet containing personalized benefit options and costs based on age, location, and other indicative data.

- **Calculation**
  Calculate individual available options, price tags, and tax savings projections designed to conform with the client's benefits plan and strategy.

- **Personalized Enrollment Materials**
  Produce personalized enrollment materials ranging from one-page fact sheets to multi-page enrollment
workbooks. Develop complex distribution sorting algorithms for ease of enrollment material distribution.

• **Election Processing**
  Collect and edit employees’ election plan choices through the use of an Integrated Voice Response system. Acknowledge/confirm participant elections through personalized statements.

• **Integrated Voice Response**
  Provides an automated enrollment facility through the use of touch-tone telephones that allow employees to enroll at their convenience. Provides on-line editing which ensures error-free enrollments.

• **Interface to External Systems**
  Provide automatic interfaces to external systems such as the client’s payroll/benefit system and insurance carriers concerning enrollment.

Additionally, the ongoing administration system must perform the following:

• **Employee Database Maintenance**
  Establish an employee database with all relevant indicative, election and dependent data. Perform file maintenance on this database to keep the information current.

• **Process New Hires and Status Changes**
  Provide complete enrollment functions as outlined above for new hires throughout the year. Provide a mechanism to record life-event status changes and allow changes in benefit coverage as indicated in the client’s status change guidelines.

• **Administrative Reporting**
  Generate administrative reports that detail the employees in the various categories defined by the system such as new hires, terminations, etc.

The Enrollment Administration system is PC-based software designed to run on IBM compatible personal computers. The software is typically installed on client’s networks for a multiple user setup. The primary users of the system are client benefit administrators (BAs) who use the system to monitor enrollment and answer employee
inquiries. Customization and maintenance programming is performed by internal ABC staff.

Figure 4 provides a visual representation of the system and its interactions.
5.2 Comparative Analysis of Development Processes

ABC's approach to flexible benefits administration is to customize software solutions to fit the needs of the client, offering maximum flexibility in the administration system. The sections below provide a detailed description of two projects following different development approaches.

Project A is the development of an Enrollment Administration system using ABC's current approach to the software development process. Project B is a similar development effort that places an emphasis on a more formalized systems engineering approach which includes concurrent engineering principles as well.

The projects are very similar in size and scope. Each software system is being developed for a client with 1,000 to 2,000 employees and comparable benefit plans. The plan designs represent like complexity and the personalized enrollment materials are exactly the same size. The similarities in the projects provide a sound basis for comparison.

5.3 Current Development Process (Project A)

5.3.1 Project Participants and Initial Interaction

The formation of the project team was the initial step in beginning the development process. ABC's project team consisted of:

- Project Manager - Person leading the project with overall responsibility for implementation. Maintains contact with head client contacts.

- Systems Lead - The head system consultant responsible for software development. Manages analysts and programmers.

- Analysts/Programmers - The technical consultants performing the day-to-day development and programming activities. Includes areas of administration system software and interactive voice response software.

- Technical Experts - Other ABC consultants with expertise in benefits plan design and employee
communications.

- Lead Client Personnel - Persons responsible for implementation from the client side. Includes high level personnel in the areas of employee benefits, communications, and systems (specifically payroll).

The team also included ABC vendors who supply design, art work, video, and production capabilities primarily from the employee communications standpoint.

The communication among the team was critical for the software development process. Communication throughout the process relied on weekly status meetings and electronic mail.

The weekly status meetings were used to gather the entire team to review progress according to the project schedule, discuss known issues, and attempt to reveal potential problems. The meetings were lead by the Project Manager with presentations from each area to review status. No specific facilitation techniques were used.

Electronic mail was relied on heavily to keep other team members informed of meetings, decisions, and other happenings in specific areas. All messages were filed electronically in a client folder for historical, audit, and billing purposes.

The initial project meeting was used to establish a complete and detailed project schedule including key milestones. This schedule was agreed upon by all team members and was used by the Project Manager and lead consultants to monitor progress. Figure 5 shows the milestone schedule for Project A.

5.3.2 Development Phases

Project A followed ABC's development process as discussed in section 3.2 above. The following is a description of the activities, outputs (documentation, etc.), and review points of each phase. Review points are defined as milestone points requiring approval by a specified individual or group of individuals.

Planning Phase
Activities: The primary activity performed was the
planning meeting. This meeting was held with the entire project team to finalize the project scope and schedule, budget, and outline the preliminary benefit plan design.

Outputs: Project schedule, final budget, preliminary plan design document, detailed list of deliverables.

Review points: Approval of plan design document by client.

Requirements Definition Phase
Activities: ABC analysts conducted interviews with four users for their input to the functional requirements of the software. The concept of flexible benefits was totally new to most users; therefore, the analysts questions needed to be specific as possible. Particular emphasis was placed on existing processes that the client used to enroll and administer the current benefit plan.

A system planning meeting was held with ABC systems staff and key client system personnel to specify key system milestones. This meeting was also used to identify preliminary test conditions.


Review points: Approval of functional requirements document by lead user.

External Design Phase
Activities: A series of three external design meetings with ABC, client payroll, and external groups were held. Specifications for interfaces were developed for the initial data collection, payroll, and insurance carrier interfaces.

Outputs: Interface specifications.

Internal Design Phase
Activities: A series of five internal design meetings with ABC systems staff were held. The primary input to the design process was the functional requirements document.

The user interface was specified as well as the logic needed for the calculation of personalized benefit options and costs. The script used for the interactive Voice Response election entry system was developed.
Hardware requirements were produced and recommendations on hardware were given to the client to have available for the implementation phase.

Outputs: Internal design document which includes user interface specifications, calculation manual, Voice Response system script, and hardware configuration.

Review points: Approval by client.

Development Phase
Activities: The customized modules were programmed and unit tested. Modules were integrated into system test area upon completion of unit testing. Code review meetings and walk-throughs were held to identify unresolved issues.

Hardware acquisition and installation was made at the client site.

Outputs: Program modules and programmers' issues log to present issues at next status meeting.

Review points: Passing of unit testing of each module.

Test Phase
Activities: ABC performed system testing of integrated software as it was to be delivered to the client. The test plan was followed to capture the entire range of scenarios that were identified.

Client testing included the review of all system output including personalized enrollment materials. Calls were placed to the Voice Response system to test the script and the election capturing module.

Outputs: Tested system and test enrollment materials.

Review points: Passing of system testing and user testing sign off on system outputs and script.

Implementation Phase
Activities: Customized software was installed and made operational at the client site and a set of revalidation tests were performed to ensure the system is capable of performing all key functions. Two training sessions were given to all users (Benefit Administrators) by ABC analysts.
Outputs: Operational system and users manual.

Review points: Revalidation of key control functions.

**Maintenance Phase**

Activities: Two primary programmers and one backup programmer were assigned to monitor the software performance and provide support in response to client calls. The initial calls for change requests were heavy over the first two months of system operations.
<table>
<thead>
<tr>
<th>Project Phase</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Planning</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Requirements Def.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>External Design</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Internal Design</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Development</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maintenance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5 - Milestone Diagram for Project A
Support programmers logged all maintenance calls. Change requests were prioritized by the System Lead with direction for the client and returned to the programmers for implementation.

Outputs: Change request log kept by programmers.

Review points: Client sign off on changes implemented.

5.3.3 User Involvement

A special note is applied here to emphasize the areas of involvement of the users throughout the development process of Project A as outlined above.

Actual client users, as opposed to higher level client personnel, were involved minimally throughout the development process. The user involvement was limited mainly to the user interviews by ABC analysts in the Requirements Definition phase and the reviewing of system outputs in the Testing Phase.

User involvement is a major difference in the development processes of the two phases. Project B emphasizes including the actual users on the project team and stresses their input and involvement throughout the development life cycle as outlined below.

The need for additional user involvement, as mentioned above, stems from the volatility of user requirements. Including the users more throughout the phases of development provides for feedback which allows developers to control and validate changes to requirements. [19]

5.4 Development Process with Concurrent Engineering Principles (Project B)

5.4.1 Team Formation and Interaction

As with Project A, team formation was the initial step in the development process for Project B. The project team included the same members as those in Project A: project manager, systems lead, analysts and programmers, technical experts, and lead client personnel.

Additionally, other team members were included that became critical to the success of the project. These
members were:

- **Facilitator** - A ABC consultant with no direct project responsibility was utilized to conduct requirements definition and system design meetings.

- **Key Users** - Unlike Project A, this project emphasized to the client the importance of including key benefit systems users (benefit administrators). Three of most knowledgeable (and hard to replace) users were included in all requirements definition activity, design activity, and throughout the entire development life cycle.

The inclusion of these additional team members provided for a more "cross-functional" team. The combination of facilitator and users provided a mechanism to spur more creativity, feedback, and communication within the team.

The inclusion of the "real end" users provided the detailed knowledge of the client's operations; while the lead client personnel furnished the strategic guidance that made the project implementable.

Upon completion of the formation of the team, another initial step was to provide the team members with training for some of the concurrent engineering tools that would be used. The primary tool requiring training was the techniques of Joint Application Design (JAD), which is outlined below.

As stated earlier, effective and efficient team communications is essential to the success of the project. The emphasis was switched from the weekly status meeting to regular working sessions where team members worked together throughout the development effort.

Logistically, the team members were grouped together in a set of cubicles in order to facilitate informal communications as well. The three key users were provided working spaces within this area also and spent three or four days a week at ABC's office during the development process.

Electronic mail was utilized heavily for communications in the project also, similar to Project A. An electronic bulletin board was established for publishing meeting schedules and other information with significant
importance. Varying groups within the project were set up electronically as well to prevent from requiring all members to weed through messages that did not impact them.

As with Project A, a detailed project schedule was established and agreed upon by the team members. Figure 6 was the milestone schedule for Project B.
Figure 6 - Milestone Diagram for Project B
5.4.2 Development Tools based on Concurrent Engineering

Project B utilized two distinct tools based on concurrent engineering principles throughout the development process. These tools were: Joint Application Design (JAD) sessions and prototyping.

In order to incorporate the knowledge of the key users, JAD sessions were utilized primarily to address a critical issue in the software development: user involvement in requirements definition, systems design, and development.

JAD sessions replaced the archaic method of defining system requirements by having systems analysts interview users. The interviewing method took considerable time and left the development team always hoping that the right people were interviewed and the real needs of the system were identified.

Three JAD sessions were held. The first was a four day session which was aimed at building a team-oriented partnership among the ABC systems professionals and the users. Its initial focus was on defining the scope of the system. Then, the focus shifted to defining the system requirements. At the end of the session, a requirements definition document was produced.

The second session, held for three days, concentrated on the analysis of how to satisfy the requirements defined. The software functionality was specified as well as the user interface. Working with the ABC systems staff, the specifications which made up the internal design document were completed.

The ABC staff tried to allow the users to take on as much of the design tasks as possible. They interjected to help in answering technical questions specific to flexible benefits. They also aided the users in resolving how needed functionality may best be customized into the core software. The result of the second workshop were an updated requirements definition document and a preliminary design documents that could be used to develop a working prototype.

The third JAD session was a two day session that allowed the user to work "hands on" with the developed prototype.
This resulted in the alteration of the requirements document as well as the design documents. Numerous suggestions and changes were identified in the user interface and the logical flow of the software that otherwise may not have been identified until implementation.

There were numerous other informal JAD sessions used as design review and development sessions. Each were used to review prototype versions and further the development process.

Prototyping is the second tool used in conjunction with JAD sessions to aid in ABC’s software development process. The prototyping methodology used by ABC was of the extensible variety meaning that ABC continued to enhance versions of prototypes into the actual production software.

ABC analysts and programmers used the JAD workshops to begin experimental prototyping. The first prototype consisted mainly of the customized user interface with no true processing capabilities.

Subsequent sessions were held for the users to work with the prototype and provide further input to the requirements and design of the system. Each version and review session contributed to the more detailed functionality needed for day-to-day activity. Specifications were written and updated after each prototype version.

Software must evolve as user needs are better understood. This inevitably leads to changes in requirements with provisions made for validation and evolution. Prototyping is an extremely effective technique for getting the design right. It encourages analysis of system design. [19]

By utilizing the prototype reviews as design meetings, the users were very active participants in the decision-making processes. The cross-functional combination of users and ABC systems expertise became an effective development technique. By using this combination, operational issues became systems issues, and systems issues became operational issues. The concurrency of these events resulted in a more thorough task of requirements definition, analysis, and design.
5.4.3 Modified Development Process

In completing Project B, ABC utilized the above concurrent engineering tools to supplement its development process. The result is a modified development life cycle.

The phases of this modified SDLC are no longer as discrete as those in Project A. The initial phases, especially, tend to overlap which is the principle of concurrent engineering. Figure 7 represents the development process as followed by Project B.

The phases, or mixtures thereof, are described below along with outputs and review points.

Planning Phase
Activities: The formation of the team was completed and training for team members was held in JAD techniques. A planning meeting, similar to Project A, was held to establish the project schedule and final budget. Preliminary plan design was also outlined. Communication strategies were implemented in the form of electronic mail groups and a common work area was set up for the team including client staff.

Outputs: Project schedule, final budget, preliminary plan design document, project team trained in JAD techniques.

Review points: Approval of plan design document by client benefits personnel.

Requirements/Design/Prototyping Phase
Activities: This phase, or interaction of phases, was where the true concurrency took place in the development. As outlined above, the project team, including key users, used the JAD workshops to concurrently define requirements, conduct preliminary design, and develop the software through of prototyping.

The JAD sessions and informal prototype review sessions provided a mechanism for hands on review of the software including the user interface. Requirements were identified throughout the phase that otherwise would not have been found until post-implementation. Most issues were able to be resolved on the spot since the project team was located in the same work area.
These sessions and review meetings reiterate the fact that "freezing" requirements at any point in a systems engineering development approach is very difficult. Software requirements, especially with applications new to the user, are very dynamic and often change throughout the development life cycle.

As with Project A, hardware requirements were produced. One of the users on the project team took responsibility for the procurement of the hardware needed at the client site for the live system.

Code review sessions were held among the programmers, and in some cases with users, with an emphasis on identifying modules which could be coded for reusability. A senior programmer was assigned the responsibility of improving the code modularity. The goal was to increase code usability on this particular project and to also build in modularity into the core software to improve subsequent systems maintainability.

This concurrent effort also provided an effective preliminary testing effort. It also supplied a means to develop a user acceptance test plan to be performed in the testing phase.

After several versions, the system was, in effect, developed and ready to proceed to the next phases.

Outputs: Requirement and design specifications through the use of prototyping, test plan and scripts, prototype versions.

Review points: Review by users of each prototype version and sign-off to begin system testing.

Test Phase
Activity: The system testing phase was radically different from Project A in that the emphasis was shifted from the ABC system professionals to the users. Additional users were brought in for a three day testing session.

Empirical user testing was performed to ensure that the user interface was effective and easy to use for all users. Individual users were allowed to work with the software to become familiar with the "look and feel" for half of the first day. During the second half, they were given test scripts to execute to test the user interface.
For the second and third days, users were grouped in a method referred to as constructive interaction. This approach of two users working together was utilized to encourage more verbalization about the software. The user groups were given test scripts to execute that represented daily tasks.

At the end of the third day, time was reserved for the testers to comment and make suggestions of changes to the system.

Although ABC systems staff was not directly related in carrying out the system test, the project team members were involved by providing a brief training to get the users' started. They were also on hand at all times to answer in questions. The programmers worked a late shift in order to correct any problems or make requested changes that the users had identified during the day. Most changes were made the same evening as they were identified and retested.

**Implementation Phase**
Activities: The customized software was installed and made operational at the client site. Revalidation test were performed to ensure installation was completed properly.

A single, much shorter, training session was held by ABC for the Benefit Administrators. This was possible because of the involvement of the three key users in the development life cycle as well as the participation of other users in the testing phase. The three key users also were able to serve as valuable resources for the other users when the system began operating.

Outputs: Operational system an user manual.

Review points: Revalidation of key control functions.

**Maintenance Phase**
Activities: Support was reduced from two primary programmers in Project A to one primary programmer. A backup programmer was provided as well. The support programmer was the contact point for any operational questions and maintenance or change requests.

The request levels were very low compared to Project A. Changes were able to be applied to the client system
electronically with the capability of ABC staff to dial into the clients' network and make changes. An electronic mailbox was also set up for problem identification and questions.

Outputs: Change/maintenance request log.

Review points: Client sign off on changes implemented.

5.4.4 Emphasis on Requirements and Users

The emphasis on user involvement in the early phases of Project B should not be understated. This focus is aimed at improving software quality by meeting user expectations and requirements as well as reducing software maintenance costs.

The concept of user participatory design was relied on heavily in Project B. Going beyond "getting to know the user", the project team encouraged users to raise issues. Many of these issues would not have been uncovered through user interviews. Users reacted well to the prototypes which were much more effective than reviewing voluminous specifications. It was vitally important to have the actual users to participate in the design, not just their managers.

Also, as highlighted in section 4.2.5 above, the project team included a "power user." One of the key users, often referred to as "radical" by many on the client staff, had developed an application used within the benefits department. The application was a benefit tracking database that he extracted from a mainframe database. The database was used as an ad hoc reporting function which was very popular in the benefits department.

This functionality was maintained in ABC's software by providing a similar extract capability from the Enrollment Administration system database. The client's Information System department took over the responsibility of maintaining the ad hoc reporting tools.
Figure 7 - Modified Software Development Process Followed by Project B
6.0 EVALUATION OF PROJECT PERFORMANCE

6.1 Comparison and Selection of Evaluation Methodology

Several systems engineering evaluation models were examined to find the appropriate one for comparing the data and results of these two projects. A thorough, rigorous systems engineering tool is needed to judge the performance of each project in a scientific, objective, and consistent manner.

Sink details three differing evaluation techniques which were considered. Two other models, one specific to software quality measurement, were also explored.

The first of Sink's techniques is the normative productivity measurement methodology (NPMM). The NPMM is a process by which measures, ratios, or indexes of productivity can be participatively identified and developed into a measurement system. This approach is a partial-factor approach which is best suited for smaller units of analysis such as the group level. The participatory and development-oriented nature of the NPMM ties the measurement and evaluation processes into a system includes planning, control, and improvement processes. [20]

The second measurement approach presented by Sink is titled the multifactor productivity measurement model (MFPMM). It is a more consultative, less participatory approach. The MFPMM is actually a decision support system model that operates with organizational system data on prices/costs and quantities of output and input resources. It is best suited for larger units of analysis such as the plant or firm level. The MFPMM is a total-factor productivity model.

The third procedure is actually utilized within NPMM and is known as the multicriteria performance/productivity measurement technique (MCP/PMT) or objectives matrix. This objectives matrix is also defined by Riggs and West. This technique allows for the development of an aggregate, single indicator of performance through partial-factor criteria as used in NPMM. The MFPMM does this implicitly but there substantial costs in its development and certain situations in which the approach is not appropriate.

Another comparison or evaluation technique presented by
Riggs and West is the priority decision table. The table provides a format for conducting and recording comparisons of criteria that lead to decisions. The table accommodates mixed dimensions and utilizes importance ratings to distill a one-number priority rating from all of the criteria evaluations. This single summary score, similar to the objectives matrix, allows competing courses of action to be compared on an equivalent basis. [21]

A final measurement technique considered was presented by Cave and Maymon and applies specifically to the measurement of software quality. The technique defines software quality in terms of the relationship between system availability and the cost of maintaining the availability. [22]

The above techniques were all studied and considered for use in this project. The factors of an evaluation methodology which were considered important for this project were:

- rigorous, systems engineering approach,
- multicriteria based,
- weighted approach based on criteria, and
- arrives in a single, quantitative indicator of performance.

The technique which best satisfies these factors is the objectives matrix, therefore it was chosen as the evaluation methodology. The objectives matrix is defined further and applied to the project in section 6.3.

Although the other methodologies are proven, there were reasons for not using them here. The MFPM was felt to rely too heavily on cost and revenue figures which were difficult to define and more applicable to manufacturing measurements.

The priority decision table is very similar to the objectives matrix but uses a rating technique which scales against the highest alternative as opposed to a defined optimum level. This provides no flexibility for ranking in terms of improvement based on levels higher than any already achieved.

The definition of quality in the software quality measurement process is not consistent with the objectives of this project. System availability is not a pertinent
criterion for quality within this project's context.

6.2 Data Criteria, Gathering, and Analysis

Following the NPM technique, a form of the nominal group technique was used by ABC software development professionals to determine appropriate criteria for evaluation.

Pertinent data was identified in order to evaluate the effect of supplementing ABC's software development approach with concurrent engineering principles. The data was collected in order to make a quantitative analysis of the two projects. This analysis will lead to the results and conclusion of the effectiveness of the concurrent engineering principles.

The criteria was chosen in order to reflect the effectiveness of the projects meeting the objectives as stated in Section 2.1. The criteria are described below and are broken up in four groups representing the each objective. They are:

Objective:
Improve the quality of software developed in terms of meeting customer expectations and requirements.

Criteria:
1. **Number of client-requested modifications after implementation.** Data collected over a two month period. Collected to judge the effectiveness of the requirements definition and meeting client expectations.

2. **Number of informational requests from users (not requiring system changes).** Data also collected over a two month period.

3. **Number of training hours needed.** Used to reflect the effectiveness of the user interface and software usability.

4. **Number of user hours during testing phase.** Gathered under the assumption that increased user activity in the testing phase will further improve meeting the expectations.

Objective:
Reduce the costs of development by reducing the amount of re-work and iterations through the design and development phases.

Criteria:
5. Total number of hours spent on project by ABC personnel. Gauges the development costs on ABC's side.
6. Total hours spent on maintenance changes. Used to determine the costs of the maintenance activity over a two month period.
7. Number of "rework" issues identified after the development phase. These are issues where major time is spent because the requirements were incorrect - not simple changes for user interface or other reasons.
8. Number of hours spent fixing "rework" issues. Used to determine the costs of the rework.
9. Ratio of project costs to budget. Overall measure of project costs versus estimates developed.

Objectives:
Improve the maintainability of the computer code generated and thus extend the useful life of software systems.

Show that code reusability and efficiency can be increased through improving the development process and better understanding of user requirements.

Criteria:
10. Number of reusable modules. Used as a measure of code efficiency and reusability.
11. Total number of modules. Gathered as a gauge of the code maintainability.
12. Average hours spent per maintenance item. Collected to estimate the efficiency of the code.
14. Number of user hours in the early life cycle phases - requirements definition, design, development.
This data was gathered as indication of the emphasis of maintainability being "built in" at the requirements level.

Table 1 displays the raw data of each project used as input in the evaluation process. The importance, or weight, of each criteria is also listed which will be used in the evaluation as well.
### Table 1 - Data Collected from Projects

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Project A</th>
<th>Project B</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Number of client-requested modifications</td>
<td>15</td>
<td>7</td>
<td>15</td>
</tr>
<tr>
<td>2. Number of informational requests</td>
<td>33</td>
<td>19</td>
<td>10</td>
</tr>
<tr>
<td>3. Number of training hours</td>
<td>28</td>
<td>16</td>
<td>5</td>
</tr>
<tr>
<td>4. Number of user hours in testing phase</td>
<td>30</td>
<td>96</td>
<td>5</td>
</tr>
<tr>
<td>5. Total Wyatt hours on project</td>
<td>450</td>
<td>575</td>
<td>10</td>
</tr>
<tr>
<td>6. Total hours spent on maintenance changes</td>
<td>83</td>
<td>32</td>
<td>10</td>
</tr>
<tr>
<td>7. Number of &quot;rework&quot; issues</td>
<td>6</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>8. Number of hours spent on &quot;rework&quot; issues</td>
<td>98</td>
<td>34</td>
<td>5</td>
</tr>
<tr>
<td>9. Ratio of project costs to budget</td>
<td>1.10</td>
<td>1.40</td>
<td>10</td>
</tr>
<tr>
<td>10. Number of reusable modules</td>
<td>16</td>
<td>36</td>
<td>5</td>
</tr>
<tr>
<td>11. Total number of modules</td>
<td>86</td>
<td>52</td>
<td>5</td>
</tr>
<tr>
<td>12. Average hours spent on maintenance items</td>
<td>5.53</td>
<td>4.57</td>
<td>5</td>
</tr>
<tr>
<td>13. Total lines of code</td>
<td>25,000</td>
<td>18,000</td>
<td>5</td>
</tr>
<tr>
<td>14. Number of user hours in early phases</td>
<td>78</td>
<td>220</td>
<td>5</td>
</tr>
</tbody>
</table>

Total: 100
6.3 Application of Evaluation Tool - Objectives Matrix

As stated above, the systems engineering tool known as the objectives matrix (OMAX) was utilized as an evaluation methodology. OMAX is a measurement method that uses indicators and a weighting procedure to develop a total effectiveness indicator. [20]

As applied to performance measurement, the objectives matrix is a compact format within which goals originate and from which a profile of accomplishments take form. The objectives matrix is made of seven portions. These portions are:

- **Measurement Criteria** - These are the characteristics identified that collectively delineate the success of a project. The criteria are entered as headings for the columns of the matrix.

- **Performance** - The actual raw data registered as a single number are entered in the next line for all criteria.

- **Scales** - The body of the matrix is composed of levels of achievement for all the criteria. There are eleven levels, ranging from unsatisfactory performance at level 0 to a realistic objective for superior accomplishment at level 10.

  When an objectives matrix is initiated, the prevailing level of performance is considered to be at level 3. Each higher score is achieved by passing a hurdle defined. Several ABC consultants with much experience in similar projects had input in developing the scales for this evaluation.

- **Score** - In the line immediately below the body of the matrix, the performance number, registered in the performance line immediately above the body, is converted to a score. This is done by comparing the recorded performance number with the hurdle scores.

- **Weight** - Importance ratings are attached to all the criteria to indicate their relative impact on the objectives. These ratings were also developed by an experienced group of ABC consultants. The sum of individual criterion weights totals to 100,
which corresponds to 100 percent meeting of the objectives being evaluated.

- **Value** - The score multiplied by the weight for the criterion in which the score was recorded is the value of performance for that period.

- **Performance Indicator** - The sum of the values (weighted scores) is the performance indicator for the period. The initial indicator would have a value of 300, because all the criteria rate a score of 3 for the level of achievement at the start of the monitoring process.

Although the objectives matrix format is simple, its application includes ingredients of sound management practices. Its development follows advice to identify goals and objectives and assign priorities. Its versatility is also a valued feature. A weighted indicator method is perhaps most attractive for non-manufacturing activities, where measurement is generally more difficult.

Weight assignment is an important factor within the methodology. It provides an opportunity to emphasize activities that have the greatest potential for reaching the goals defined. The final phase of the matrix combines the criteria scores and weights to determine a single performance indicator. This single number represents the composite performance of an individual project.

Tables 2 and 3 apply the objectives matrix to both projects arriving at performance indicators which are compared in the results section.
### Table 2 - Objective Matrix for Project A

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Number of client-requested modifications</th>
<th>Number of informational requests</th>
<th>Number of training hours</th>
<th>Number of user hours in testing phase</th>
<th>Total Wyatt hours on project</th>
<th>Total hours spent on maintenance changes</th>
<th>Number of &quot;rework&quot; issues</th>
<th>Number of hours spent on &quot;rework&quot; issues</th>
<th>Ratio of project costs to budget</th>
<th>Number of reusable modules</th>
<th>Total number of modules</th>
<th>Average hours spent on maintenance items</th>
<th>Total lines of code</th>
<th>Number of user hours in early phases</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Performance</strong></td>
<td>15</td>
<td>33</td>
<td>28</td>
<td>30</td>
<td>450</td>
<td>83</td>
<td>6</td>
<td>98</td>
<td>1.10</td>
<td>16</td>
<td>86</td>
<td>5.5</td>
<td>25,000</td>
<td>78</td>
</tr>
<tr>
<td><strong>Scales</strong></td>
<td>10</td>
<td>0</td>
<td>12</td>
<td>4</td>
<td>140</td>
<td>250</td>
<td>0</td>
<td>0</td>
<td>0.75</td>
<td>50</td>
<td>10</td>
<td>1</td>
<td>10,000</td>
<td>275</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>3</td>
<td>18</td>
<td>8</td>
<td>120</td>
<td>300</td>
<td>15</td>
<td>1</td>
<td>10</td>
<td>0.80</td>
<td>45</td>
<td>20</td>
<td>2</td>
<td>12,000</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>6</td>
<td>18</td>
<td>12</td>
<td>100</td>
<td>350</td>
<td>30</td>
<td>2</td>
<td>20</td>
<td>0.85</td>
<td>40</td>
<td>40</td>
<td>3</td>
<td>14,000</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>9</td>
<td>22</td>
<td>16</td>
<td>80</td>
<td>400</td>
<td>45</td>
<td>3</td>
<td>30</td>
<td>0.90</td>
<td>35</td>
<td>50</td>
<td>4</td>
<td>16,000</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>12</td>
<td>26</td>
<td>20</td>
<td>60</td>
<td>425</td>
<td>60</td>
<td>4</td>
<td>40</td>
<td>0.95</td>
<td>30</td>
<td>60</td>
<td>5</td>
<td>18,000</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>15</td>
<td>30</td>
<td>24</td>
<td>50</td>
<td>450</td>
<td>75</td>
<td>5</td>
<td>50</td>
<td>1.00</td>
<td>25</td>
<td>70</td>
<td>7</td>
<td>20,000</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>18</td>
<td>34</td>
<td>28</td>
<td>40</td>
<td>475</td>
<td>90</td>
<td>6</td>
<td>75</td>
<td>1.10</td>
<td>20</td>
<td>80</td>
<td>10</td>
<td>24,000</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>21</td>
<td>38</td>
<td>32</td>
<td>30</td>
<td>500</td>
<td>105</td>
<td>7</td>
<td>100</td>
<td>1.20</td>
<td>15</td>
<td>90</td>
<td>13</td>
<td>28,000</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>24</td>
<td>42</td>
<td>36</td>
<td>20</td>
<td>550</td>
<td>120</td>
<td>8</td>
<td>125</td>
<td>1.35</td>
<td>10</td>
<td>100</td>
<td>20</td>
<td>32,000</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>27</td>
<td>46</td>
<td>40</td>
<td>10</td>
<td>600</td>
<td>135</td>
<td>9</td>
<td>150</td>
<td>1.50</td>
<td>5</td>
<td>125</td>
<td>25</td>
<td>36,000</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>30</td>
<td>50</td>
<td>44</td>
<td>0</td>
<td>650</td>
<td>150</td>
<td>10</td>
<td>200</td>
<td>2.00</td>
<td>0</td>
<td>150</td>
<td>30</td>
<td>40,000</td>
</tr>
<tr>
<td><strong>Score</strong></td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td><strong>Weight</strong></td>
<td>15</td>
<td>10</td>
<td>5</td>
<td>5</td>
<td>10</td>
<td>10</td>
<td>5</td>
<td>5</td>
<td>10</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td>75</td>
<td>40</td>
<td>20</td>
<td>15</td>
<td>50</td>
<td>40</td>
<td>20</td>
<td>15</td>
<td>40</td>
<td>15</td>
<td>15</td>
<td>25</td>
<td>15</td>
<td>15</td>
</tr>
</tbody>
</table>

Performance Indicator: 400
### Table 3 - Objective Matrix for Project B

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Performance</th>
<th>Scales</th>
<th>Score</th>
<th>Weight</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Number of client-requested modifications</td>
<td>Number of informational requests</td>
<td>Number of training hours</td>
<td>Total Wyatt hours on project</td>
<td>Total hours spent on maintenance changes</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>19</td>
<td>16</td>
<td>96</td>
<td>575</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>0</td>
<td>12</td>
<td>4</td>
<td>140</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>3</td>
<td>16</td>
<td>8</td>
<td>120</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>6</td>
<td>18</td>
<td>12</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>9</td>
<td>22</td>
<td>16</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>12</td>
<td>26</td>
<td>20</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>15</td>
<td>30</td>
<td>24</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>18</td>
<td>34</td>
<td>28</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>21</td>
<td>38</td>
<td>32</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>24</td>
<td>42</td>
<td>36</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>27</td>
<td>46</td>
<td>40</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>30</td>
<td>50</td>
<td>44</td>
<td>0</td>
</tr>
<tr>
<td>Performance Indicator</td>
<td>575</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
7.0 RESULTS

7.1 Conclusions

As can be seen by the overall performance indicators from Tables 2 and 3, Project B performs considerably better based on the criteria of the objectives matrix. This indicates that the stated objectives have been met by enhancing ABC's software development approach through a more rigorous application of systems engineering principles including concurrency.

The emphasis on user involvement in the early phases of the software life cycle is reflected in the data, specifically the number of user hours in the early phases. The number of modifications, hours spent on modifications, "rework" issues, and hours spent on "rework" issues were lower in Project B. This indicates that user requirements were defined thoroughly and incorporated in the design of the software upon its initial release.

These numbers also represent an opportunity to control software maintenance costs by reducing the amount of rework and client-requested modifications after implementation. This is an area of importance for ABC, and other companies, since the maintenance area can often contribute to 50% of a project's cost. Controlling these costs often means the difference between profits and losses in most ABC projects.

The reduced number of training hours needed and informational requests as well as the increased number of user hours in the testing phase indicate the user involvement throughout the entire life cycle. These factors also lead to improving software quality through meeting customer expectations and reduced maintenance costs.

Improvement in maintainability also results from Project B techniques. Although there are many technical issues that may contribute to code reuse, an organization must address nontechnical issues as well. Issues such as developer training and better user requirements also contribute to greatly to being able to code reusable modules.

Increasing the number of reusable modules not only reduces the total number of modules but the total number
of lines of code as well. These factors attribute to improving code efficiency, maintainability, and reusability.

There are areas in the matrix where Project B did not rate as highly as Project A. The total number of ABC hours increased in Project B and the ratio of cost overrunning the budget also increased in Project B.

At first glance, these may seem to be critical discrepancies that run contrary to the objectives of the project. However, the large number of hours spent on Project B by ABC staff, and in turn the costs overrun, can be accounted for. These numbers primarily represent an investment in training the ABC staff in concurrent engineering techniques as well as getting accustomed to working in a new way.

For these reasons, it could be expected that subsequent projects using concurrent engineering techniques should improve performance in the areas of ABC hours. Therefore, ABC’s software project costs should be lowered resulting in increased profits. Projects continuing to operate using the current development approach, such as Project A, would have no basis to expect improvements in any of the criteria.

7.2 Methodology Analysis

The methodology used to investigate ABC’s software development process was effective. Comparing two similar ABC projects provided a sound basis to examine the successfulness of meeting the objectives of applying these principles.

The experience and input of many ABC professionals helped establish solid criteria to judge the projects. These criteria were used as a basis to collect data and as an input to the evaluation. The use of the objectives matrix provides a solid systems engineering tool to perform a thorough evaluation of the two projects in meeting the stated objectives.

An argument could be made that the conclusions drawn above could result from other independent variables of each project other than the use of concurrent engineering principles. Even with the considerable experience provided by ABC professionals, it is admittedly difficult
to prove conclusively that the criteria measured were affected only by concurrent engineering principles and not by other variables.

Section 2.2.6 above listed several factors that could be viewed as independent variables. It is impossible for these variables to be identical across the two projects and, therefore, have no influence on the results. However, these factors vary minimally from Project A to Project B. Therefore, one can conclude that the bulk of the difference in the results shown can and should be attributed to the application of a thorough systems engineering approach.

Other factors not listed in Section 2.2.4 that were outside of the control of this project could attribute to the results. Again, however, none of these are estimated to affect the results considerably enough to disprove the hypothesis. Some of these factors and the reason for considering their impact minimal may be:

- **Client project team members:** Although it is impossible to have the same client team members on two different companies' project team, the number and roles of the team members were identical. The more important factors were the availability of the client's decision makers and its organizational structure.

- **Internal knowledge of application:** A lack of technical knowledge pertaining to the application on part of the analysts and technical experts can also contribute to varying results of a software development effort. However, within each of these projects, the same ABC professionals were utilized. These individuals had a thorough knowledge of the flexible benefit application.

- **Technical skills of programmers:** Often, the skill level of the developers with differing programming languages and technologies greatly affects the results of development projects. As stated above, though, the computer programming within these projects is not complex. Furthermore, the projects were being developed for identical PC platforms and hardware by the same ABC programmers.
8.0 Recommendations

The software industry has spent much of its early years focused on technology itself and not how to make the technology effectively utilized. The industry is realizing that a change of focus is necessary to improve the way software is developed. ABC's software focus is similar to these industry trends.

Based on the results of this project, the following recommendations have been developed for ABC:

- Continue with and expand the use of applying systems engineering processes with concurrent engineering techniques to its current software development approach. This should be viewed as an investment in a process with great potential to produce long term savings in costs and improvement in software quality.

- Increase the number of ABC personnel involved in these projects and train them in the use of techniques such as group interaction, prototyping, and joint application design (JAD) sessions. The positive results found in this project will continue as ABC's professionals become more familiar and comfortable with these techniques.

- Provide a forum for client involvement from a user standpoint. Include users in training sessions and encourage involvement throughout the development process.

- Expand the application of these techniques to larger software development endeavors. These long term opportunities may have even greater impact on ABC's larger projects. The projects discussed herein are relatively small compared to other ABC efforts. The potential results of these techniques could be much larger yielding bigger increases in profits on larger projects.

Larger projects require a greater emphasis on multi-functional teams. Changing and more complex requirements place an increased focus on the early portion of the life cycle in order to alleviate extreme costs in the maintenance phase.

- Modify the existing Systems Development Life Cycle.
(SDLC) manual to help formalize and encourage the concurrency that can take place within the life cycle phases. This manual should be viewed as an evolving document that provides software development process guidelines but is continually open for improvement.

Other software developers and the software industry as a whole could also benefit from incorporating a more rigorous systems engineering approach with concurrency principles in their development process. It should be viewed as a new paradigm to software development that can be tried and modified to best suit a particular companies' structure and development philosophy.

Compared to manufacturing and other engineering disciplines, software engineering and development is a relatively young industry. New techniques and processes need to be utilized, tested, implemented, modified, and discarded in an effort to continuously improve the software development process.
9.0 REFERENCES


