WestVirginiaUniversity
THE RESEARCH REPOSITORY @ WVU

Graduate Theses, Dissertations, and Problem Reports

2003

Implementing software engineering practices in small industry with a focus on requirements elicitation

James Clifford Fleming
West Virginia University

Follow this and additional works at: https://researchrepository.wvu.edu/etd

Recommended Citation

Fleming, James Clifford, "Implementing software engineering practices in small industry with a focus on requirements elicitation" (2003). *Graduate Theses, Dissertations, and Problem Reports*. 1372.
https://researchrepository.wvu.edu/etd/1372

This Thesis is protected by copyright and/or related rights. It has been brought to you by the The Research Repository @ WVU with permission from the rights-holder(s). You are free to use this Thesis in any way that is permitted by the copyright and related rights legislation that applies to your use. For other uses you must obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/ or on the work itself. This Thesis has been accepted for inclusion in WVU Graduate Theses, Dissertations, and Problem Reports collection by an authorized administrator of The Research Repository @ WVU. For more information, please contact researchrepository@mail.wvu.edu.

Implementing Software Engineering Practices in Small Industry with a Focus on Requirements Elicitation

by

James Clifford Fleming

Thesis submitted to
The College of Engineering and Mineral Resources
at
West Virginia University
In partial fulfillment of the requirements
for the degree of

Masters of Science
in
Software Engineering

Approved by:
Dr. John Atkins, Committee Chairperson
Dr. Franz X. Hiergeist
Ms. Cynthia D. Tanner

Lane Department of
Computer Science and Electrical Engineering

Morgantown, West Virginia
2003

Keywords: Software Engineering, Project Management, Capability Maturity Model, Configuration Management, Personal Software processes, Software requirements, Data warehousing

Copyright 2003 James Clifford Fleming

Abstract

Implementing Software Engineering Practices in

Small Industry with a Focus on Requirements Elicitation

James Clifford Fleming

I have been involved in small industry for 33 years and I have seen how the evolution of computers and software has affected small companies striving to grow in their market place by trying to take advantage of an evolving technology. Many times an individual is assigned the task of developing software to fit the company's needs and begins the process without any formal training in the practices of Software Engineering. My Thesis will discuss my evolving skills, gained through my Masters in Software Engineering degree work, as a Software Engineer and how I have been able to implement proper Software Engineering in my position in a small company. This has been a worthwhile challenge and the results of my work and study could benefit any person involved in the Software Engineering profession. The focus of this paper will be the challenges of requirements elicitation in a small industry environment.

Acknowledgement

I would like to thank Dr. John Atkins and Ms. Cynthia Tanner for all of their work at establishing the Masters of Software Engineering program at West Virginia University. The program structure has given me the opportunity to pursue a Masters degree that I otherwise would not have had. I want to thank Dr. John Atkins, Ms. Cynthia Tanner, and Dr. Franz X. Hiergeist for their support as my thesis committee. I would like to thank all of my instructors for their professionalism and assistance during my involvement in the program.

Finally I would also like to thank my wife Amy, my children, and grandchildren for patience with me while I have been busy with my studies.

Table of Contents

List of Figures

Acronyms Used

CARA	Criticality and Risk Assessment analysis
CAD	Computer Aided Design
CM	Configuration Management
CMM	Capability and Maturity Model
GUI	Graphical User Interface
IDE	Integrated Development Environment
ISO	International Organization for Standardization 9000-3
LAN	Local Area Network
MSSE	Masters of Software Engineering
CNC	Computerized Numerically Controlled
COTS	Commercial Off the Shelf
OOD	Object Oriented Design
PSP	Personal Software Process
MMS	Manufacturing Management System
V&V	Validation and Verification

Introduction

According to the Small Business Administration, small business accounted for three-quarters of US net jobs between 1999 and 2000. It is estimated that in 2002 there were 550,100 new businesses started but 584,500 businesses failed.[1] In the year 2000 515,977 of the small businesses in the US had between 20 and 99 employees.[2] Many small businesses are family owned and were started on shoestring budgets around a unique idea.

I have spent most of my 33 year working career in a small (50 employees) firm where I first started working with computers and computer software. Computers were a new technology for small firms and this new technology held the promise of being the next marvel that was going to save the business and make everything work better. The shock to company owners was that these computers had to be programmed in order to do anything. My first exposure to the computer was the Radio Shack model 1 personal computer. My first exposure to programming was part programming a numerically controlled, NC, milling machine. Interestingly, I later developed a computer program that generated NC code demonstrating the potential power of the computer.

---

[1] SBA Office of Advocacy latest census data
http://www.sba.gov/advo/stats/sbfaq.html#q3

[2] Statistics of small businesses
http://www.census.gov/epcd/www/smallbus.html#EstabSize

I acquired the knowledge I needed to program computers by studying the programming manuals that came with the equipment. When I wrote programs my work generally had no plan or organization. At the time it all seemed like pure genus sense no one else in the organization knew anything about programming nor wanted to learn, but I was continually bogged down in exhausting programming projects that never fully fulfilled the expectations of the company owners, whom I reported to directly. The requirements were continually changing causing massive rewrites and edits. If an edit was made, it almost always created problems elsewhere in the code. This was a classic case of the ripple effect.

I continued to improve in my abilities to write usable programs as time passed and often networked with other individuals like myself to get the latest hacking news. The problem was that I was not improving in my planning and program design technique since I had no formal consistent design or coding methods. I had not developed a formal process.

I have practiced sound mechanical engineering in my machine design responsibilities but failed to do the same thing in my software engineering work. The problem stems from the fact that I never really viewed my software development as an exercise in engineering. This is a common problem throughout small industry in software development and leads to enormous frustration in developers and users. It can also lead to substantial loss of valuable resources for small companies.

I became aware of software engineering in the mid-1980s but due to many responsibilities of my work I was unable to pursue any formal software engineering training so I continued to produce software that was functional and effective but

looking back I realized my work was totally unacceptable as an engineered product. I began producing programs for some of the Company's customers and the programs performed the required task so the customers appeared happy with the product but the code was difficult to maintain and expand as requirements changed. At some point I realized I must improve my skills so in 1990 I enrolled in a self-paced programming course where I re-learned the Quick Basic programming language and was introduced to Pascal and C++. During this period in time I started a part-time business called DocuPrep. I started developing more sophisticated software and was very quickly confronted with the same old problems brought on by not using any formal process. When I learned of the MSSE. degree program at West Virginia University, I returned to college, completed my Bachelors degree and then enrolled in the Masters in Software Engineering degree program at WVU and immediately improved my software engineering skills as well as other areas of my work.

This paper will examine some of my experiences as I have applied my evolving skills in a real world small industry environment and how what I have learned, through my masters class work, coupled with my work experience has impacted in a very positive way, not only myself but also my fellow coworkers and the company.

The need of a Formal Process

Learning and implementing a formal process is a complicated task but the most difficult part of the implementation is practicing the process. Many software developers in small industry fall into the trap of taking shortcuts to seemingly save time. Many times the pressures to implement a project are so great that the

developer will just start programming on the fly without any plan just to get something working. There is no worse approach to the problem. What generally happens is the scope and requirements are poorly defined and continually drift so the program does not perform any required task well and is abandoned at some point due to lack of user support thus leaving the developer's reputation as a programmer tarnished. Success or failure is often magnified in small industry since there are fewer people to take the credit or the blame. Frederick P. Brooks Jr. spoke of the "Tar Pit"[3] where teams of programmers often struggle, as if stuck in a pit of tar, to no avail to develop software but fail due to the lack of or not following a formal process. This is also true for a small team or the individual developer. The reasons for not following a process are many but the end results are usually the same. A formal process, that is practiced, can help prevent this scenario.

I have worked on and managed projects using both no formal process and the formal processes I have established from my masters degree work. The differences between then and now are dramatic. What is astounding is that I felt that my earlier work, at the time, was exceptional but I had nothing but my work to which a comparison could be made. My previous projects did in fact work and I did have an improvised way of doing things that was evolving, but I now believe that if I had continued on that path eventually I would have experience problems that would be difficult, if not impossible, for me to deal with. If computer technology was not changing so rapidly, I may have gotten away with using my improvised processes but now users expect more from the computers they use and this equates into software project complexity.

---

[3] Frederick P. Brooks, Jr., **The Mythical Man-Month** (Addison Wesley Longman, Inc. 1995) 4.

In the following section I will discuss the improvements I have made as a direct result of the MSSE program and highlight how each class had an impact on the way I think and do my work, the projects I manage, and the people working on those projects. This will set the stage for a study of a major software project I engineered and developed using the skills I have developed as a result of my Masters degree work. The project is development of a manufacturing management and cost evaluation system. It is nearing completion and is a good example of the improvements made in my software engineering practices. The project required extensive domain knowledge and an extensive requirements elicitation process.

The Learning Curve

By definition Software Engineering is "an applied science devoted to improving and optimizing the production of software".[4] My responsibilities as Director of Design and Development at the company I work for include the practice of Conceptual Engineering and Design Engineering of industrial water pumps. I have found that concepts that I have learned for Software Engineering are completely applicable to other engineering disciplines. I have improved my skills as an industrial pump engineer by following the same methodologies I use in software engineering. My many years of experience in the industrial pump engineering field made it easier for me to grasp the principles of software engineering.

---

[4] Bryan Pfaffenberger, **Dictionary of Computer Terms Sixth Edition** (Simon & Schuster, Inc. 1997) 478

The first step to improvement is to realize that one really needs to examine existing processes, be objective and make a critical evaluation of existing talents and not become defensive of past work. My first reaction during the early weeks of my first MSSE course, SENG 330 Validation and Verification, was none of this makes any sense to me in my situation. I also heard this from some of my fellow classmates. This new way of doing things was completely incompatible with my way of doing things. After all, I have been doing it for years. What an awakening I was about to receive. One of the biggest obstacles for the individual, who has worked in the field for years and then returns to school, is to realize that one doesn't know everything. The next largest obstacle is changing the way one thinks and listens. After all, why did I enroll in the MSSE program? Once you get beyond these obstacles a whole new world opens up for you and your coworkers. Your value as a software developer just went up with the realization that it is time to learn new concepts. The days of just hacking it out are coming to an end.

New Knowledge

PSP

Knowledge by definition is a set of propositions about something that is capable of generating additional propositions by means of deduction.[5] When I look back and contemplate all that I have learned in my MSSE class work and how what I have learned has impacted how I perform as a software engineer, one class in particular was at a more personal level. SENG 591A Personal Software Process was a class targeted more to my personal processes and abilities rather than an overall software

[5] Bryan Pfaffenberger, **Dictionary of Computer Terms Sixth Edition** (Simon & Schuster, Inc. 1997) 283

engineering process. This course exposed all of my bad habits, lack of knowledge, and process. There was no way to complete the course without close personal evaluation. The class was disturbing at first and discouraging at times as the class progressed, but there seemed to be a delayed reaction. After the fog of study and assignments raised, it became evident to me that I had made some very valuable discoveries about myself, my abilities, and my deficiencies. It is impossible to properly manage projects if one cannot manage properly oneself. This course created a discipline in me that made it very difficult to continue bad habits, the type of habits that evolve unknowingly over time without a proper process. Even the seemingly simple task of typing code can be enormously frustrating if you misspell variable names or make incorrect variable entries into technically complicated formulas. These types of problems can cost many extra hours of debugging and testing time. This is especially important to the software engineer, such as myself, in small industry who has many other responsibilities and moves from one project to another trying to meet project schedules. Just the awareness that I am prone to simple but costly mistakes when I try to rush my work makes me focus more on the job at hand and make a conscious attempt to avoid those types of problems.

Beginnings of PSP

The development of a personal software process, PSP, that includes a coding standard, that I have started to use faithfully, and a Personal Improvement Proposal form, has helped me overcome many of my deficiencies in the coding process and has helped me to continually focus on improving my skills. PSP is not a replacement for a comprehensive software engineering process but will greatly enhance a small industry software engineer's ability to implement a company wide process. The PSP

also focuses on the importance of acquiring domain knowledge before starting a project. Time spent here will help prevent costly time delays later in the project. Proper domain knowledge will also facilitate accurate requirements elicitation, a unique challenge in small industry projects. PSP is an individual improvement process that I will take with me anywhere I go. It has made me more efficient and more structured in my thinking leading to more efficient and structured design. My only regret is that I took SENG 591A towards the end of my class work instead of the beginning. I believe it would have helped me in my other course work. The following are the Coding standard, Figure 1, and the PIP form, Figure 2, I now use.

Figure 1 Coding Standard

CODING STANDARD Visual Basic 6.0			
Program Headers	Begin all programs with a descriptive header.		
Header Format	********** * Program Assignment: program name * Program Date: * Program Version: * Program Description * Program Author: **********		
Listing Contents	Reuse instructions: Identifiers: Program Comments: Spacing: Indentions: Capitalization:		
Reuse Instructions	All reuse code must be edited to conform to this Coding standard. All reuse code must have origin source identification. All reuse code must show date of reuse.		
Identifiers	Data Type Boolean Byte Collect Obj Currency Date(Time)	Prefix bln byt col cur dtm	Example blnFound bytRasterData colWidgets curRevenue dtmStart

	Double Error Integer Long Object Single String User-Defined-Type Variant	dbl err int lng obj sng str udt vnt	dblTolerance errOrderNum intQuanity lngDistance objCurrent sngAverage strFName udtEmployee vntCheckSum
Program Comments	Add description at the beginning of all sub routines explaining the function of the subroutine. Document the code so that the reader can understand the operation. Comments should explain both the purpose and behavior of the code. Comment variable declarations to indicate their purpose.		
Spacing	Write programs with sufficient spacing so that they do not appear crowded. Separate every program construct with at least one space.		
Indentions	The functional description statement should be indented one space. The highest level statements that follow the functional description should be indented one tab. Each following nested block should be indented one additional tab.		

Code Format

```
Sub LoadBatch()
'*********************************************************************************************
' This sub-routine will load batch data onto form2 (Planning) after it is retrieved
'  from the data base.
'*********************************************************************************************

 Call FileNames(FileLocation1, FileLocation2)  ' data base location

    ProductionBatchName = "BatchPd"              ' data file name


     Close #1          ' Close buffer

                  ' The following Opens a file for processing
```

```
Open FileLocation1 + ProductionBatchName For Random As 1 Len = 256

Lock #1       ‘ Protect against file sharing collisions while processing

   For intPb = 1 To LOF(1) / 256       ‘ Loop through records

       Get 1, intPb, Production_Define         ‘ get record intPb

          ‘ **** Search data for instance of data from Form2.text1.text ****

          If InStr(Production_Define.ProductionName, Form2.Text1.Text) Then

               ‘ Load Form with new data

               Form2.Text2.Text = Production_Define.ProductionPart
               Form2.Text3.Text = Production_Define.ProductionDescription
               Form2.Text4.Text = Production_Define.ProductionDrawing
               Form2.Text5.Text = Production_Define.ProductionBatchTotal
               Form2.Text6.Text = Production_Define.ProductionComment
               Form2.Text7.Text = Production_Define.ProductionMatCode
               Form2.Text8.Text = Production_Define.ProductionDrFile
               Form2.Text9.Text = Production_Define.ProductionBatchPassed
               Form2.Text10.Text = Production_Define.ProductionBatchScraped

             Unlock #1    ‘ Unlock file to permit file sharing

             Close

             Exit Sub

          End If

      Next intPb            ‘ Next iteration of FOR NEXT LOOP

  Unlock #1     ‘ Unlock file to permit file sharing

 Close

End Sub
```


Personal Improvement Process

Figure 2 Personal Process Improvement Proposal form

Process Improvement Proposal	
Date: Program: Version: Author:	
Problem PIP #	Problem Description
Proposal PIP #	Proposal Description
Notes and Comments	

I find it extremely beneficial for self-evaluation to use the Personal Process Proposal form to record problems and solutions as I encounter them. I have found that many times I can review past problem entries and uncover or discover solutions to current problems. Sometimes when stress levels are high I record what I am having

problems with and then at a later time, when I am more relaxed, I start working on the solutions.

One of the benefits of a process is there is a defined way of dealing with issues when they arise which helps everyone who is affected by my work. I also use the form to record successful and failed strategies when working on the requirements phase of a project. Requirements can be very elusive when you are interviewing non-cooperative users or users who really don't understand what they need the software to do for them within their domain. It is important to record what techniques work the best. Because I now work within the disciplines of PSP I can expect my coworkers to become more disciplined in their work. All of this equates to a more structured organization and the beginnings of a corporate wide formal process.

Software Lifecycle and Capability Maturity Model

The SENG 340 Software Lifecycle and Capability Maturity course introduced me to the Capability Maturity Model, CMM. My taking this course has also had a major impact on the company and my fellow co-workers. SENG 340 made me look at the company processes in the same way SENG 591 A Personal Software Process made me look at my personal process. Through the years I had developed a crude configuration management policy, but it was not fully documented and not fully indorsed or practiced by the company. I experienced much resistance when I started implementing proper configuration management procedures. It was met with some skepticism at all levels within the company.

Maturity Levels

There are five maturity levels within the CMM.

1. Initial
2. Repeatable
3. Defined
4. Managed
5. Optimized

Level 1 - Initial Process - Organizations at the initial process level have poorly defined procedures and controls. There is no consistent application of software engineering management of processes and no modern tools or technologies are used.

Level 2 - Repeatable Process - At maturity level 2 organizations will use standard methods and practices for managing software development processes. They will have methods for performing cost estimating, scheduling, requirements change, code changes, and status reviews.

Level 3 – Defined Process - The organization that is at the Defined process level will have the standard methods and practices of a level 2 organization and will also have implemented organizational and methodological improvements. It will have design and code reviews, programmer training programs, review leaders, and increased focus on software engineering. There will be an establishment of a software engineering process group that will focus on software engineering and process implementation.

Level 4 – Managed Process – A maturity level 4 organization bases its operating decisions on a quantitative data. It will conduct extensive analyses of data collected during engineering reviews and testing. Tools are used to control and manage design, data gathering, and analysis. The organization is able to start projecting expected errors.

Level 5 – Optimized Process – The organization that has achieved the optimized Process, will have all of the process controls of the lower levels plus a major focus on improving and optimizing its operation. They will have introduced sophisticated error and cost analyses data and conduct comprehensive error cause and prevention studies.

The beginnings of CMM

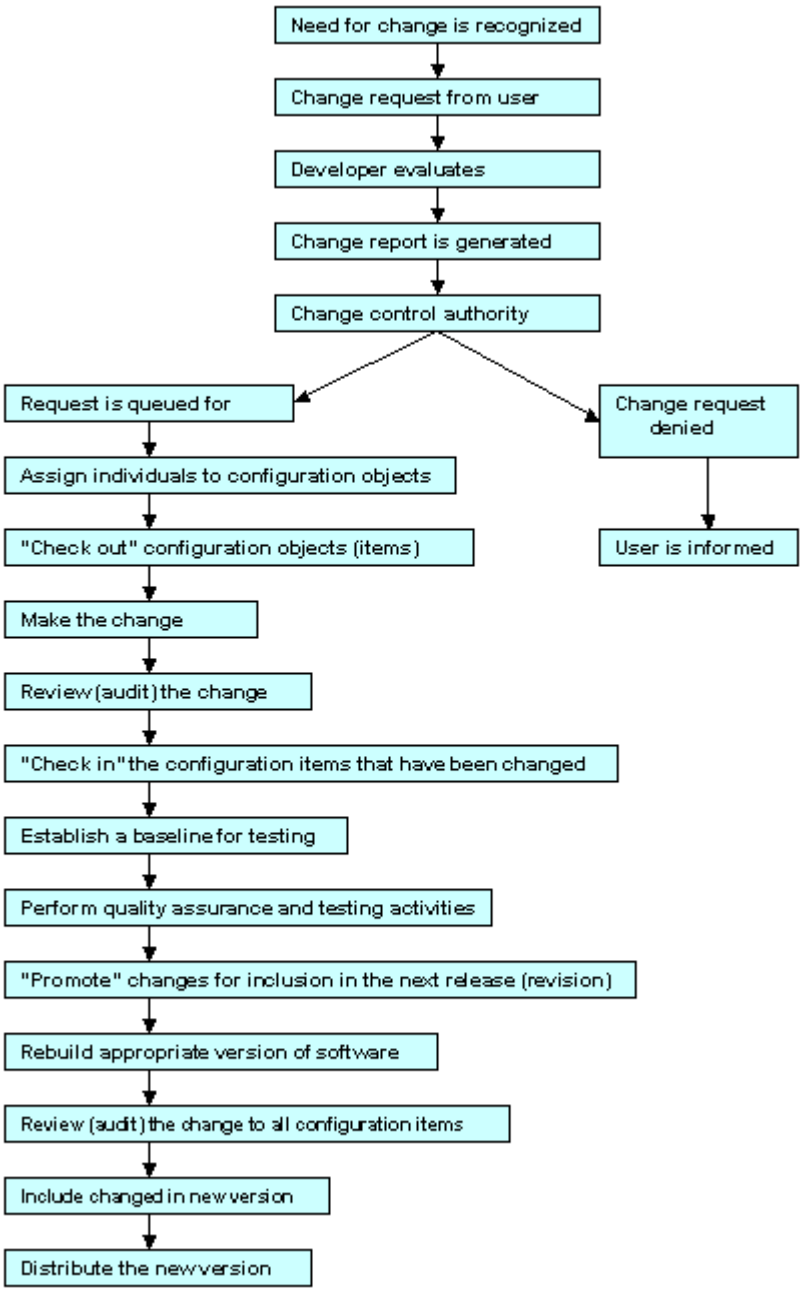
I knew that the organization was at Level one at best. I met with the owners and made the point that there was a lot of work to do. We simply had no formal processes in place and no one was really satisfied with where the company was heading. The goal is to at least improve to a consistent level 2 organization within 18 months.

The capability Maturity Model can also be applied to other areas in a small company. The organization as a whole benefits from the processes implemented. I recognized that if I did not focus totally on the software engineering aspects, and applied the same basic procedures of CMM to the mechanical side of the company I would see greater success and acceptance at all levels of the company. Software is still a mystery to many people but they sometimes understand organization and systems in relation to mechanical design and manufacturing. This was the best way to get owners and other managers to buy in to the concepts of CMM. The entire company must progress as a whole, not just the software development side, otherwise disorganization will erode the efforts to implement any level of CMM.

Gaining Process Control

The president of a small company is the role model but with my position as director of design and my formal education in software engineering I have become the champion, leader. It is my responsibility to move the organization forward in the CMM implementation. My first goal was to implement a new change control process. The following Figure 3 shows the change control process that I have implemented.

Figure 3 Change Control Process[6]



[6] Roger S. Pressman, **Software Engineering: a Practitioner's Approach** (McGraw-Hill Series in Computer Science) 235

The change control process shown in figure 3 also works well on the mechanical side of the company. I introduced configuration management for the many CAD, Computer Aided Design, product part drawings the company has for production. The CAD drawings are data files that need to be controlled and maintained. An unauthorized change in a production drawing can have disastrous results leading to enormous losses. Many small companies in industry do not practice configuration management or have very crude and ineffective methods. Usually individual machine operators will have their own collection of hard copy drawings of parts they produce on their assigned machines and may not be aware of engineering changes and continue to make obsolete parts. The problem hopefully is caught at inspection but many times the customer reports the problem.

One of the first major changes I implemented was establishing a production read only drawing database that houses only approved CAD files and eliminated all paper drawings located at various work stations. All required hard copy drawings are produced from the controlled production database. Many times there is no need for a hard copy thus eliminating unauthorized changes to hard copy drawings. All CAD files are now within configuration management controls. This process improvement alone has saved thousands of dollars in revenue that otherwise could have been lost due to using obsolete drawings to make parts. I also brought all of the CNC, Computerized Numerical Controlled, program production under configuration control. These are just some examples where the process techniques I have learned during my Masters work have led to improvements in the corporate process, not just the software engineering side of the company. It has given the whole CMM project credibility.

Figure 4 represents the Request for Change form I developed for use for both software and mechanical changes. Since people in my organization, as well as other small firms, have responsibilities throughout the organization, I found it an advantage to utilize a common form. This also increases buy-in to the process by individuals who otherwise might be resistive to the process.

Figure 4 Request for Change Form

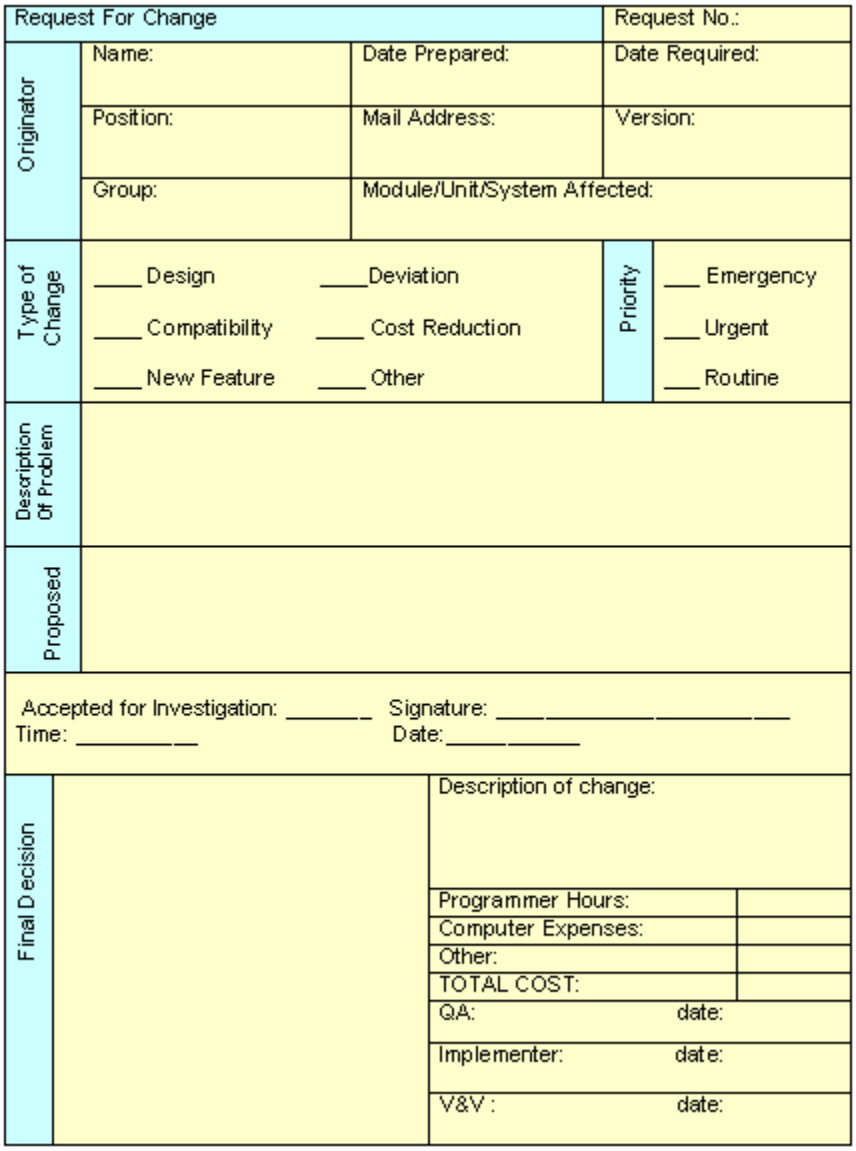
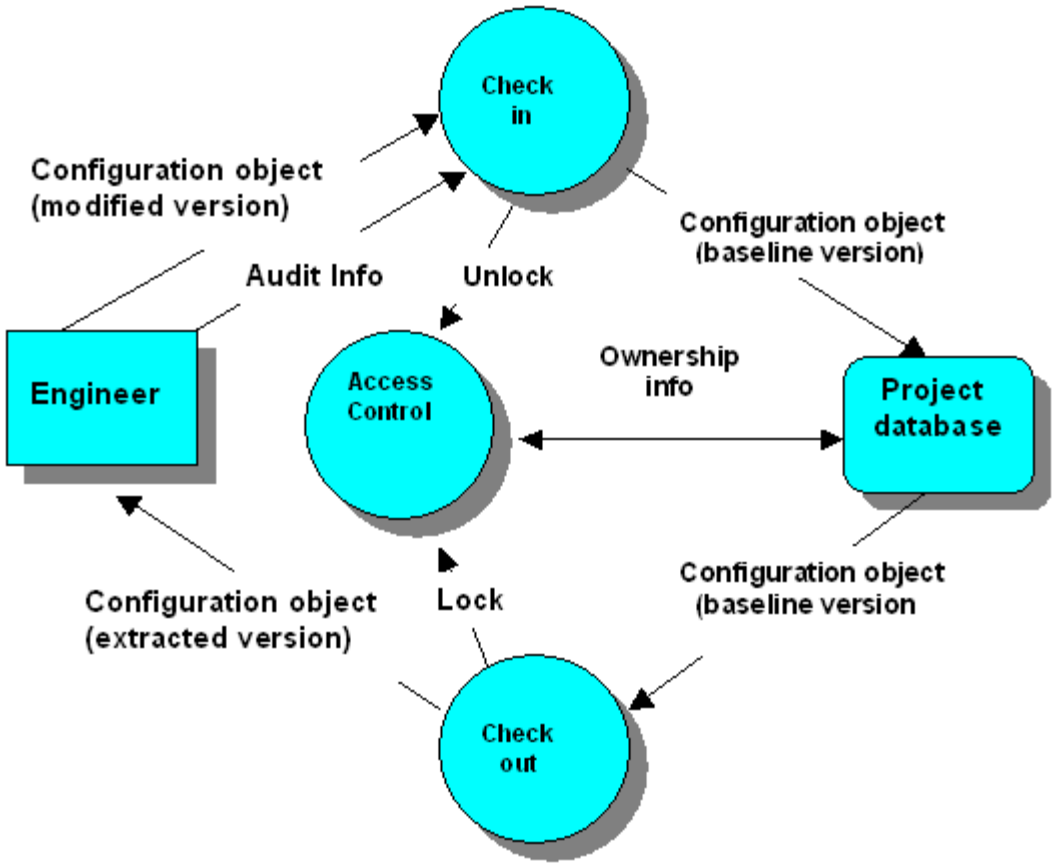


Figure 5 depicts the control procedure used once a change request is accepted and given to engineering to implement the change. The process is intentionally restrictive to prevent simultaneous changes being made to a change request item by several developers or engineers. I control access to the database by utilizing the security and permission levels for all users who require data access.

Figure 5 Access and synchronization control[7]



- The developer or engineer checks out (extracts) a copy of the configuration object.

[7] Roger S. Pressman, **Software Engineering: a Practitioner's Approach** (McGraw-Hill Series in Computer Science) 236

- Security and access controls ensure the individual has the proper access rights to change the item.
- Access control flags the configuration item read only locking out unauthorized changes.

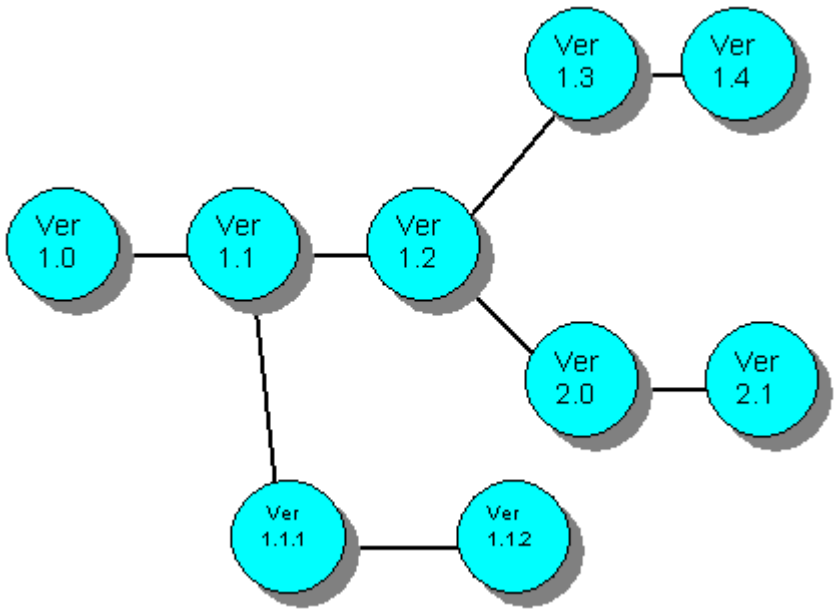
- Changes are implemented and audited creating a new base lined object.

- Access controls allow the original version of the object to be replaced by the new base lined object in the database.

The access control represents the type of disciplined controls that will be required as the organization grows. It has been a very important improvement to the organizational processes and the initial successes have made it easier for further sound software engineering practices.

The next step in the CMM that I have implemented is a better version control system. Figure 6 shows the procedure for version control that I have adopted.

Figure 6 Version Control Tree[8]



[8] Roger S. Pressman, **Software Engineering: a Practitioner's Approach** (McGraw-Hill Series in Computer Science) 232

Version control has not been a tremendous problem for me in the past as far as computer software development is concerned. I had practiced version control on the software that I produced, but when I started to implement CMM and focused on the entire organization then version control became very important. One area that became critical to control was all of the CNC programs developed for the CNC machines used in the manufacturing process. The CNC programs contain axis coordinates, feed and speed codes and machine control codes. CNC programs are downloaded to each individual machine control as production orders require. These programs are particularly vulnerable to problems associated with no formal process controls. The machine operators frequently change CNC programs at the production machines during production of parts creating unauthorized version of the programs. This procedure destroys the base line process each time it happens. This has always been accepted due to changes in materials qualities, stock size variance, and continual tooling changes. This is not acceptable within the scope of CMM.

There was an extreme ownership problem with the CNC programs. All of the CNC programs were housed on one desktop PC without adequate backup procedures. The CNC programmer has been working in his own world disconnected from the rest of the organization. This is a common practice in small companies that use this technology. I began by explaining to the programmer what I was attempting to do by implementing CMM and that it would benefit him in the long run. I wanted him to buy into the process instead of becoming an obstacle to implementation. He finally agreed to cooperate and the very first thing I did was to move all CNC programs into the main corporate database so they could be placed under configuration management controls. This database is backed up every day. I used this point in

discussions with the company president at which time he confessed he was completely unaware of how the CNC programs that are extremely valuable were being managed.

I have started using the new version control process on all of the computer programs that I have developed and those I am now working on. By bringing all software under a common version control as part of the corporate wide CMM I have been able to close many loopholes that, in the past, have led to many problems. As I have previously mentioned, the software engineer in a small firm carries many responsibilities. It is totally due to my initiative that CMM implementation has started. Many people in the organization viewed this as an ego trip on my part so it has been extremely important for me to show the positive impact this implementation has had. CMM is a work in progress.

Project Management

> Project management is the application of knowledge, skills, tools, and techniques to project activities in order to meet or exceed stakeholders' needs and expectations from a project.[9]

One major problem that can occur with managing a project is discipline of the staff. I have encountered this problem many times in the past. Much of the discipline problem was the cause of not practicing any kind of formal process and a lack of communication. I would seldom get the project completed on time and it was always an unstructured process with absolutely no guarantee of success. The system was informal and at times chaotic. There were no adequate procedures in place to properly track progress. It was a problem even on small projects. Many times people

[9] PMI Standards committee, **A Guide to Project Management Body of Knowledge (**William R. Duncan, Director of Standards 1996) 6

would be off doing their own thing with no coordinated way of reaching a common goal of finishing a project. It was also very difficult to report work progress with no system. There was some documentation but because of the fragmented way it was created it was basically useless as a managing tool.

SENG 510 Software Project Management introduced the formal procedures I needed to regain discipline and control of the project management process. The practice of proper management techniques has contributed credibility to my efforts to implement software engineering practices.

Steps to a Software Development Process

There are eight steps to defining a Software Development Process as describe in Managing Software Development Projects[10].

1. Identify the software model.
2. Identify the activities.
3. Identify the relationships among activities.
4. Document other useful information on each activity.
5. Document how to tailor the process.
6. Document on how to improve the process.
7. Obtain buy-in of the process.
8. Continually use and improve the process.

Before my involvement in the MSSE program I did not consider any of the eight steps in defining a process. There are some similarities with my old techniques because the software development process is sort of a natural way of planning and doing work. One always has something in mind and it usually is iterative until finished

---

[10] Neal Whitten, **Managing Software Development Projects** Second Edition (John Wiley & Sons, Inc. 1995) 18

or discarded. There is some documentation, even if it is only the code. Someone is going to buy-in or none of your projects would ever be used. This does not necessarily mean that every project I did was a total disaster because there was no defined process. The problem is I now realize how close to disaster I always was. I have managed software projects for many years and have been successful only because each project was relatively small. Many of the industrial pump design projects I have managed were large and I succeeded because there were drawings, plans, assembly and testing manuals, and procedures that were industry standards. I believe the model is the most important of the steps defining a software development process.

Identify The Software Development Model

All of my Masters courses discussed software development models to some degree but SENG 510 placed the development model into the management perspective of the software development process. My original informal development model resembled the code and fix model. I would just take an idea and start coding. I would then present my work to the effected users, get some feedback, make changes and continue until I ended up with a working program. It worked well on small programs, but was a horrible way to try to develop any substantial programs. Furthermore, I produced very little documentation other than the code. Figure 7 represents the code and fix model that I was using for all of my development prior to my software Engineering courses.

Figure 7 Code and Fix Model[11]

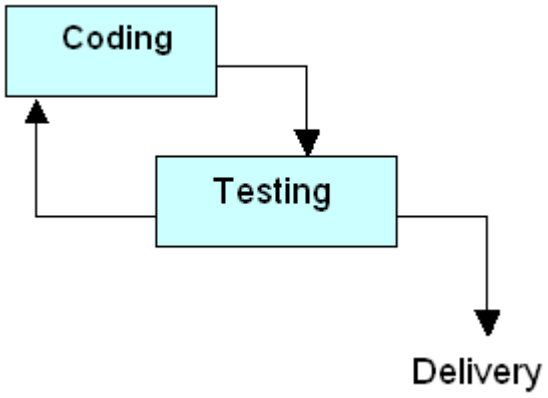
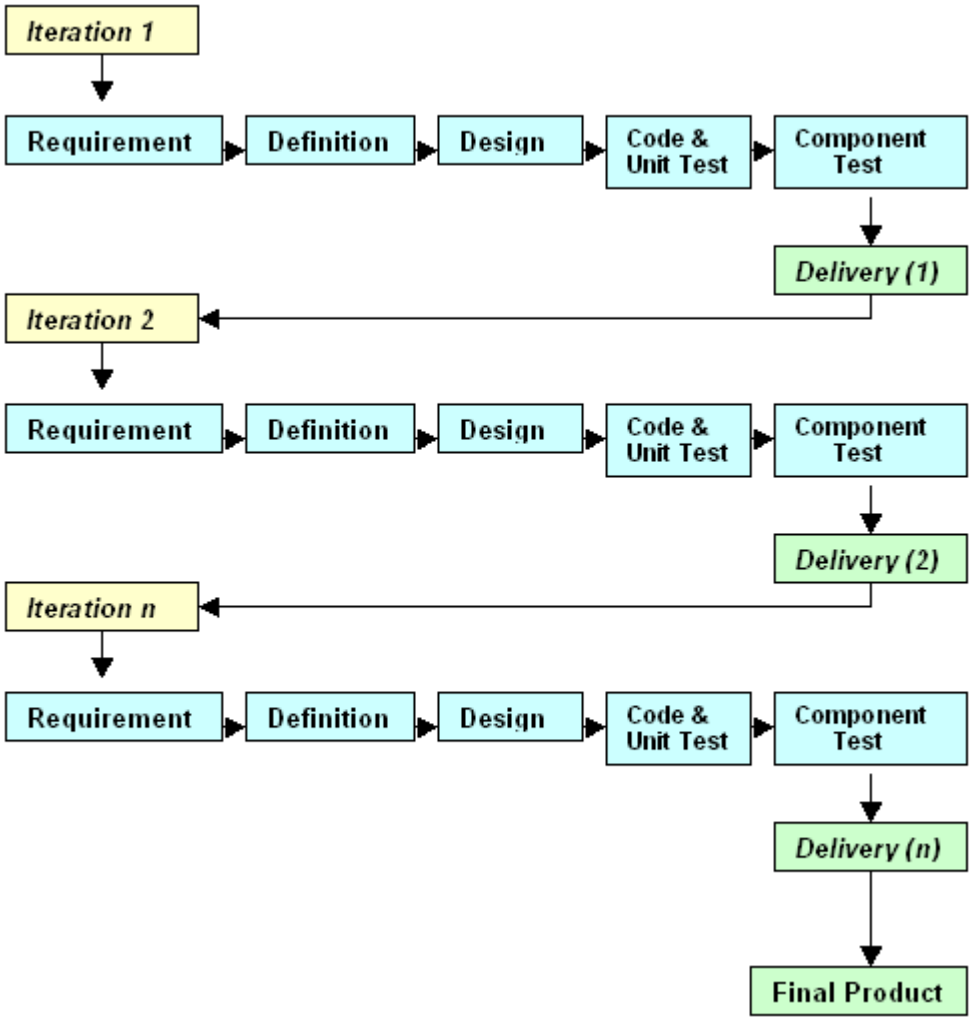


Figure 8 represents the Iterative model that I now use. This model is used when the requirements and product definitions are not necessarily known, a major issue in a small company, and may continue to be defined as the development process continues. This is accomplished through continued iterations through the development phases. This allows for early user involvement, an absolute must, in helping to define requirements and develop early versions of the product to insure the right product is being developed. The model diagram, Figure 8, shows separation of testing into two phases, unit testing in the coding phase and then component testing. A classic such as the Waterfall model discourages iteration and generally requires each phase to be completed before the next begins with the final step being delivery of the product with very little flexibility to add or change requirements. This can become a serious problem if the user injects additional requirements towards the end of the product development. The Iterative model addresses requirements creep

[11] Neal Whitten, **Managing Software Development Projects** Second Edition (John Wiley & Sons, Inc. 1995) 19

by addressing new requirements and product definitions in the next Iteration through the model phases. The main thing to be aware of is that the Iteration model allows for continued iterations through the phases and therefore can introduce additional complexity that can extend the development schedule, lead to gold plating, and added cost.

Figure 8 Iterative Development Model[12]



[12] Neal Whitten, **Managing Software Development Projects** Second Edition (John Wiley & Sons, Inc. 1995) 22

One major problem I have experienced over the years is requirement elicitation. With most of the projects with which I have been involved, the users really had no idea what they really needed the software to do. Even though they worked in a particular domain they really had little understanding of their domain or simply could not communicate their needs. Most times they understood the one area in which they were working, but could not relate to other areas of the company processes.

Requirements elicitation is a problem in large industry as well as in small industry and is not always the user or customers fault. The software engineer must gain adequate domain knowledge and use sound software engineering processes so that adequate requirements are gathered and the right product is produced. This simple concept of getting information can be a very complex and frustrating process. Before my MSSE classes I would make a simple list of some features of the basic idea and then using the code and fix model, figure 7, I would hack away until I had something that seemed to work. This approach can work on very small projects but will fail on larger ones. I know now that this was a very poor approach. The Iterative model, figure 8, is a good model to help find those elusive undefined requirements as the project progresses. The introduction of a development model has also helped me keep discipline within my staff since now there is a map to follow. It takes time for everyone to buy into the process but if you follow the phases of the model then its value becomes evident to all involved.

Identify the Activities

Once I selected the Iterative model as the model that works best for my organization, I had to develop all the procedures to implement the model.

The following is a list of activities that should occur as described in Managing Software Development Projects[13]

- Requirements
- Objectives
- Specifications
- High-level design
- Publication content plans
- Test plans
- Low-level design
- Code
- Unit and function test
- Component test
- First-draft publications
- System test
- Second Draft publications
- Regression test
- Packaging
- Delivery

Implementing a software development model in a small firm is a difficult task. People involved tend to look at the list of activities and believe they will be overwhelmed with paperwork and procedure. This is a real concern for my coworkers and myself. I have limited resources in time and personnel and have to be careful about how I utilize these resources so I have to be conservative as to what I ask my staff to do.

I believe the requirements component of the above list is the most important step in the process. If the requirements are not properly extracted and defined nothing that follows will be complete. The iterative nature of the iterative model greatly facilitates this process but the number of iterations is dependent on how well the requirements are developed. It is important to keep iterations as low as possible to maintain

[13] Neal Whitten, **Managing Software Development Projects** Second Edition (John Wiley & Sons, Inc. 1995) 35

control. Figure 9 shows the document I created that helps my staff with defining the scope and requirements definition of projects.

Figure 9 Project Concept Document

Project Concept Document	
Investigate the type of business requesting the software project.	**Try to get some preliminary domain knowledge. Take some time to learn about the client and what the client's company does and what processes are used. This will help in asking the proper questions to the users about their requirements.**
Purpose of the software and the target audience.	**1. What is the fundamental concept, purpose, and what are the goals of the proposed software?** **2. Identify the users of the software and what level their usage will be. President, Engineer, Managers, Clerical ect.** **3. Schedule meetings with a representative user of each class of user and discuss what they expect from the software to be developed. Confirm that all users understand the defined purpose of the software.** **4. Develop a list of required features and their benefits for the software, starting with the most essential. Develop this list based on information gathered in the user meetings.**
Develop a requirements list.	**1. Create a requirements list based on the user input from the user meetings. Keep each requirement description in as simple terms as possible.** **2. Insure that each requirement fits within the software concept scope. If a requirement develops from the analysis of the user**

	meetings that creeps from the concept scope then present it to the effected user representative to validate its value to the software. **3. Prioritize the requirements list.** 4. **Separate the requirements into Functional, Performance, and system response timing requirements.**
Develop use cases.	**1. Develop uses case for each requirement on the requirements list.** **2. Present the use cases to each user class representative for their review. This may lead to iteration of the requirements gathering process but will help define additional requirements missed in the meetings. Insure that any new requirements fit within the concept scope of the software.**

Figure 10 represent the process template for developing use cases for the various requirements. The use case is an excellent method to insure that requirements collected are valid and fit within the scope of the project. Before my MSSE work I never developed a use case. I would create flow charts showing program flow, make drawings depicting the different GUIs, and write notes on what users said they wanted their software to do, but I never developed a formal use case. I stated earlier in this paper that the changes before and after my involvement in the MSSE program are dramatic.

Figure 11 is the Test Report Template I developed and now use for program testing. This replaces an informal system of just running the program and fixing the problems as they are encountered. This form adds consistency, repeatability, documentation and discipline to our testing.

Figure 10 Use Case Template

Use Case Template		
Requirement #:		
Project #:	Developer:	Date:
Use Case #:		
CHARACTERISTIC INFORMATION		
Goal in Context:		
Scope:		
Level:		
Preconditions:		
Successful End Condition:		
Failed End Condition:		
Primary Actor:		

Trigger:	
MAIN SUCCESS SCENARIO:	
EXCEPTIONS:	

Figure 11 Test Report Template

Test Report Template			
Test ID: Revision #:	Program:	Test Conducted By:	Date:
Test Setup:			
Test Platform Description:			
Constraints:			
Test Objective:			
Test Description:			
Test Conditions:			
Example:			
Expected Results:			

Actual Results:	

Software Analysis and Design

One of my biggest problems over the years with my software development was my limited knowledge of proper software design. Every day was full of problems and issues and even though I understood that there must be a better way of developing a program, there was just no time at the end of the day to learn or try anything new. The problem was a paradox in that I needed to improve my skills but yet I had no time to enhance my skills and many of the problems I was having were generated by my lack of proper software engineering knowledge.

One of my major design problems of the past was requirements development. Without formal training it is all but impossible to extract adequate requirements. I always felt that I could figure them out as I went and the requirements I did get from the users were only marginally useful. This was the type of flawed thinking that was brought on by working for a small firm where there was no real emphasis on continued education or self-improvement. This is not intended to be a poor reflection of a small company, it is merely reality in small industry.

Everyday is a challenge for small business owners to keep their companies in business and software is just a small part of the problem, so they think. The problem is that today software is an intricate part of almost every operation conducted within

a company and people, such as I, continue to do their best at developing software products that will never be the caliber and quality necessary in today's business world. The small businesses that survive and grow will most assuredly require better software products, COTS or custom applications, in order to stay in business.

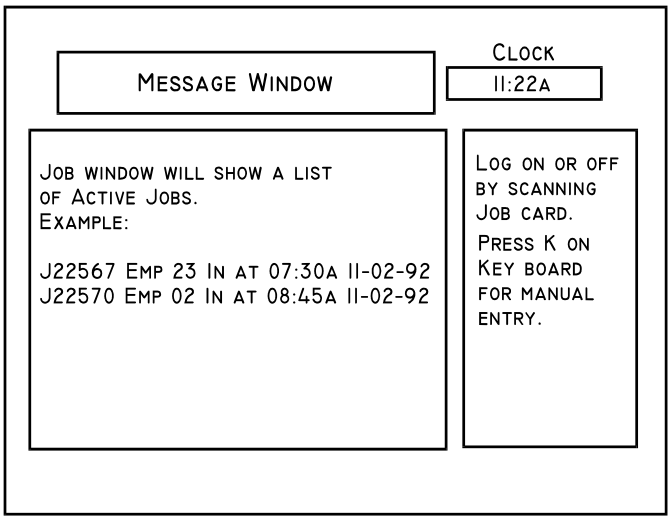
SENG 520 Software Analysis and Design, as with all of my MSSE class work helped me break out of the paradoxical situation I found myself in. This class continued the study of requirements elicitation and focused on the more intricate details of requirements. I learned how to utilize use cases to help refine individual requirements. The Use Case Template, figure 10, is a direct result of my SENG 520 class work.

GUI Design

The class also concentrated on proper program design. One area of design that I found helpful was a study of the User interface design process. This created a formal repeatable process for developing the various GUIs, Graphical User Interface, required by most of the software projects I have worked on. Figure 12 is a sample of the process I used for the software GUI design before the MSSE program.

My previous GUI design process was basically designing the GUIs on the fly and not really getting any user input. It is another one of those areas that, without a formal process, old habits just seem to linger on. I felt that I could interpret the GUI requirements as I developed the program so I would not worry about GUI design until I reached a point I had to implement them. There really was no process involved.

Figure 12 Pencil sketch of main GUI for the TimeCal



TimeCal is a program that I developed several years before my involvement in the MSSE program. It represents the extent of my GUI design prior to coding. There were times when I did not do any GUI sketches but merely programmed the GUI. I now use Visual Basic 6.0[14] for most of my software development and I use the form tools for most of the GUI prototype designs. I will occasionally make a sketch during a requirements interview to clarify GUI questions, but I always go back with a prototype I designed using the VB tools. Figure 13 is the repeatable formal process I now use.

[14] Microsoft Press, **Visual Basic 6.0** (Microsoft Corporation 1998)

Figure 13 User Interface Design Process

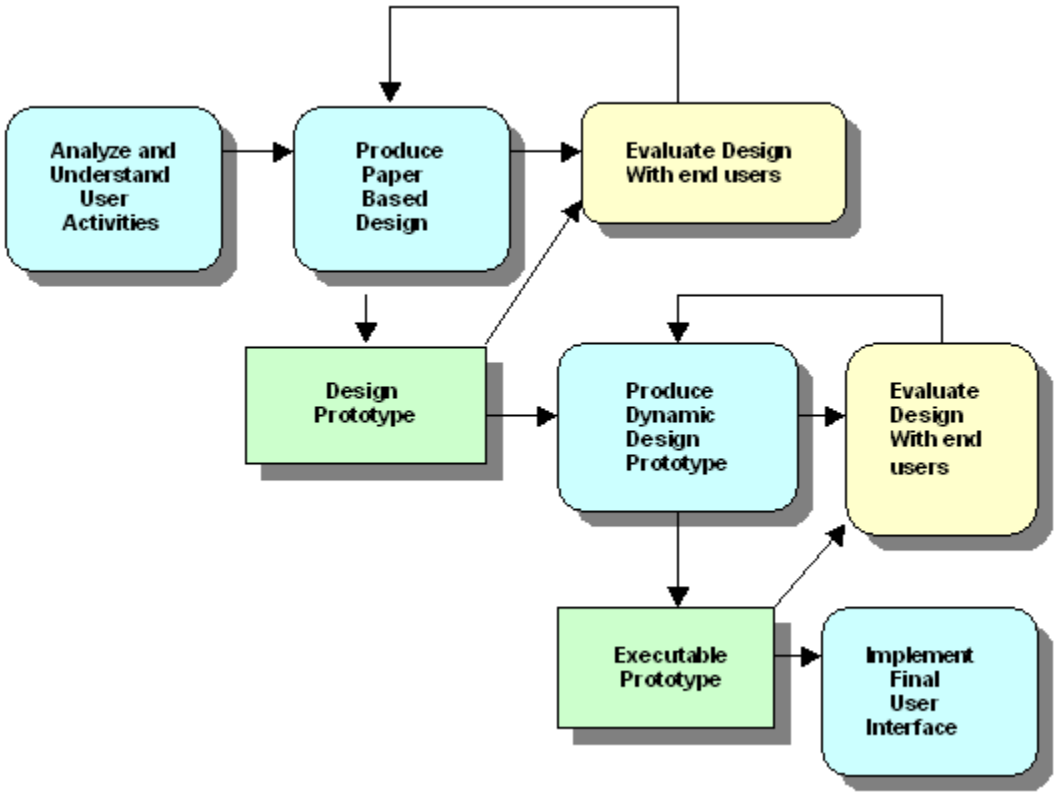


Figure14 Re-Engineered Jtrack main consol GUI

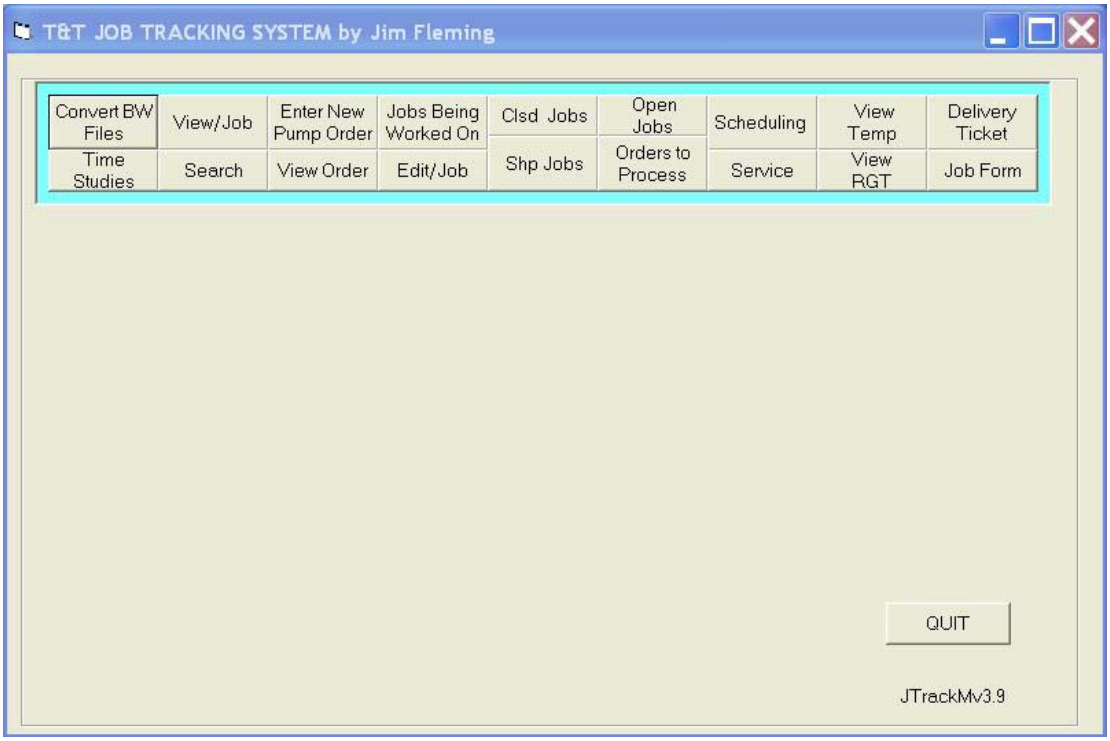


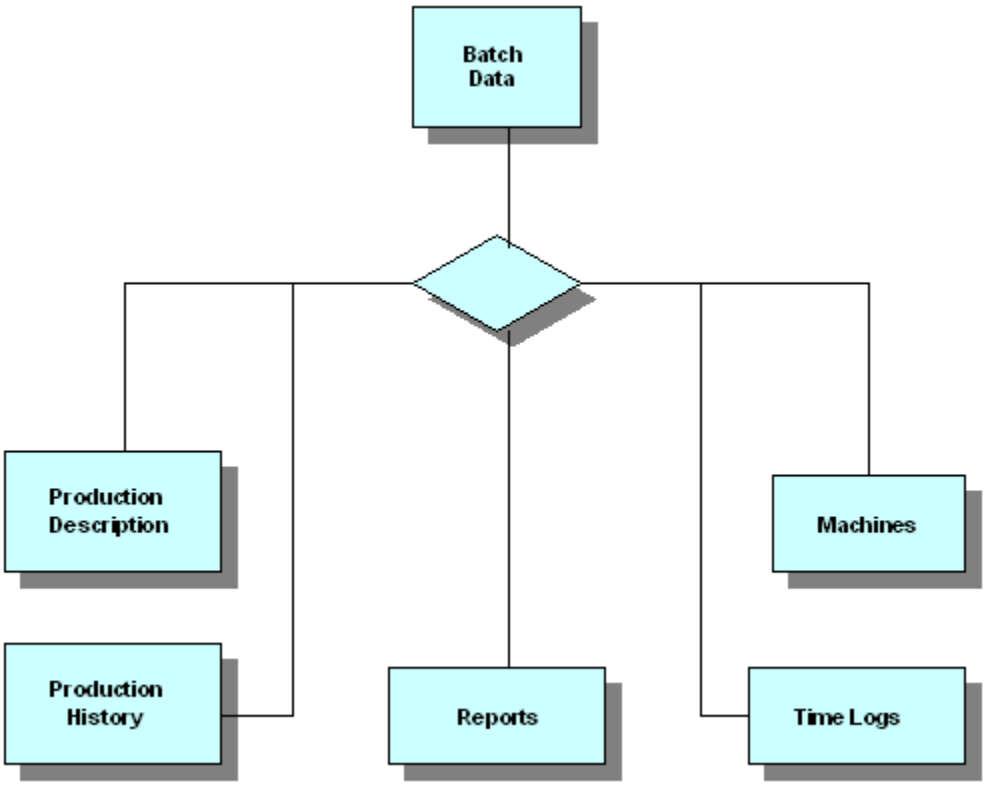
Figure 14 is the main consol GUI for a program I developed called Jtrack. I started the Jtrack project before my MSSE class work but I went back and re-engineered parts of the software as I gained more software engineering knowledge as my class work progressed. The Jtrack main consol is an example of my design using the formal User Interface Design model.

GUI design is one area of software design in which almost anyone can see improvements made. All of the stakeholders within the company find the new GUI designs more modern and intuitive that my old techniques. It also helps with developer credibility and buy-in of the project and the new processes being introduced since the users are so closely involved.

Data Modeling

SENG 520 introduced me to proper data modeling techniques. I had used flow charts to trace data flow, but had never modeled the data structure. Figure 15 represents a Entity Relationship Diagram, ERD, of the data object Batch Data decomposed into it various objects. Batch Data is a data object within my MMS project.

Figure 15 Entity Relationship Diagram

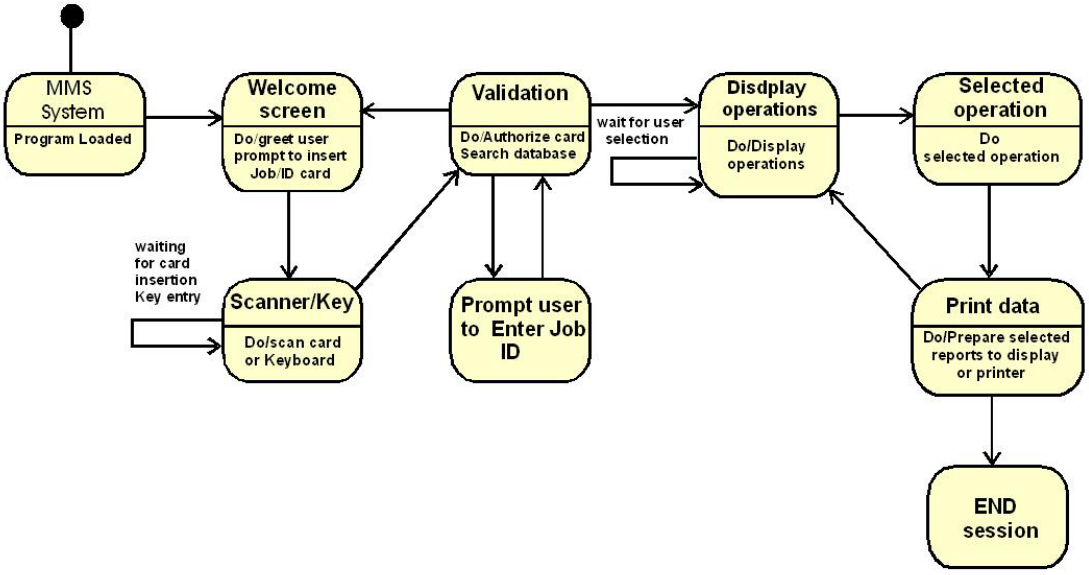


I use the ERD to help me model the relational database tables for the software I design and develop. This method provides a higher-level design capability than I have used in the past and is an effective tool for database discussions with co-workers. My previous method was simply creating a list of the data variables and just working from the list when I started setting up the tables. I never considered having to discuss issues at the database design level with anyone in the organization. When

I did so people seemed to have problems grasping what I was presenting. Using entity relationship diagrams has greatly improved my database design and presentation techniques and provides a formal repetitive database design tool. It again, adds to the credibility of my implementation of software engineering processes, so with each newly accepted method I introduce it becomes easier to continue introducing new methods.

The state transition diagram, figure 16, is a method that I did not use in the past but have found to be very useful in my work now. I use it to present the design to the user or client. The advantage of this technique is that you can make a visual presentation of a particular state of a program function or feature. The diagram aids in the understanding of a program state rather than confusing the client with a complicated text format presentation. Users get a better understanding of how their software will work by looking at a diagram so they feel more involved. I always had a problem explaining program transition to users because most times they were not programmers and really did not understand what I was talking about. It also helps in extracting additional requirements before the project enters the coding phase of the model. This is very beneficial and can result in substantial cost savings. Figure 16 is the state diagram for a card scanning procedure for logging onto the Manufacturing Management system.

Figure 16 State Transition Diagram



Object Oriented Design

The concept of Object Oriented Design[15], which I studied in the class SENG 591 Object Oriented Design, is very similar to the Product Types of Pumps[16] and sub-type assemblies concepts used in industrial pump design and technology. The process sub-divides a machine type into various sub-types of the machine and then divides the sub-types into various sub-assemblies. The OOD Methodology used in software engineering uses the same divide and conquer approach to software engineering as machine design does. A design is decomposed into classes that are composed of objects.

---

[15] Cay S. Hortsmann, **Practical Object-Oriented Development in C++ and Java** (John Wiley & Sons, Inc. 1997) 7

[16] Hydraulic Institute, Hydraulic Institute Standards (Library of Congress Card No. A82-84047) 8

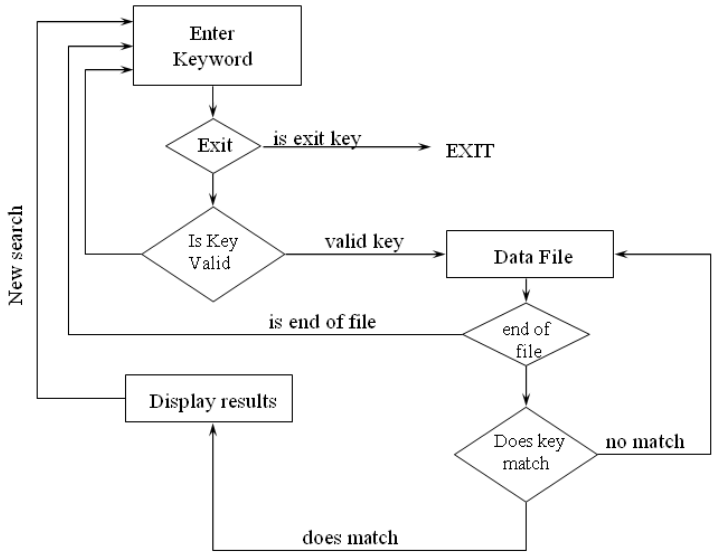
Figure 18 represents a high-level class relationship diagram of the manufacturing management module of the Manufacturing Management and cost evaluation system, MMS, which will be studied further in this paper. Breaking the problem down, decomposing it, into class relationships significantly reduces the complexity of project discussions and requirements definition early in the projects design. Users get a better understanding of the system and a better grasp of their needs when they can see the overall picture and how they fit in. A program developed using the OOD methodology will also be easier to maintain and expand as new functionality is added and requirements change. A person who, as myself, has worked his way up to or is assigned to the position of software engineer in a small firm and has never been exposed to or had formal study of this technique, can find his work much more difficult. In the past I have experienced many frustrations when I had to expand functionality of a software project because I did not use the OOD methodology.

Many meetings are informal in the small company environment. I have had impromptu meetings with a company president in a hall that could have led to the canceling of a crucial module or an entire project. My task was to convince him to continue the project. Now, most times, I am successful because I am able to present a lot of information visually and quickly. That has not always been the case. In a larger organization there would be scheduled formal meetings with department managers to present project ideas or discuss a set of requirements. In my case I have not had the luxury of a large staff to help me present my software ideas. This does not mean that it is acceptable for me not to practice sound software engineering; on the contrary, it becomes even more important to follow sound industry proven methods since there may be less room for error in a small staff.

I have experienced many frustrations in my career simply because I was not familiar with the object oriented technique in software development. My method was strictly procedural and many times just simply total chaos. The procedural technique is not completely wrong. It is fairly efficient for very small projects but as project complexity increases the OOD approach is better. The MMS system is a good example of a system that I, as a single developer, could not have developed using the procedural technique.

In the past I used flow charts, figure 17, and outlines that depicted the logic of the code and some graphic semblance of the points I was trying to make. Generally, however, I lost the interest of those whom I was presenting to and therefore compromised my work and profitability for the company. The problem I had with this method was I completely understood the charts and outlines I was presenting, but could not understand why no one else understood. Sometimes programmers drift off into their own world and lose touch with reality. The reality is most users don't know anything about computer program code and technically complicated diagrams.

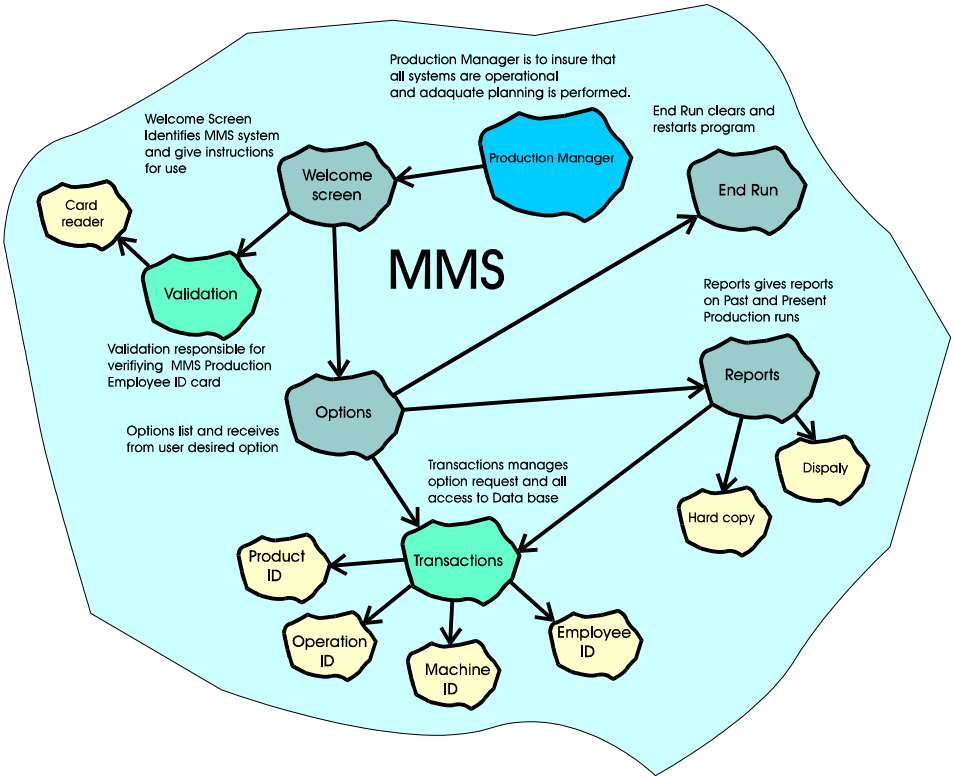
Figure 17 Past Methodology Flow Chart



The small company software engineer has an enormous responsibility in a small firm. Many times bad decisions are made to purchase off the shelf, OTS, software advertised to be the silver bullet[17] that will do everything the company needs. There is no silver bullet and it is the small company software engineer's responsibility to ensure that the software developed or purchased fulfills the requirements and not the ego of high-powered salesman or a friend of the owners. You must be able to make convincing arguments showing how developing a properly engineered custom piece of software to efficiently perform a required task or purchasing a software product that may or may not perform that task, is good or bad for the company. OOD has helped me in my design, presentation, and implementation of software products and updates. As I mentioned earlier, the position of software engineer in a small firm carries a lot of responsibility.

[17] Frederick P. Brooks, Jr., **The Mythical Man-Month** (Addison Wesley Longman, Inc. 1995) 179

Figure 18 Manufacturing Management Class Relationship Diagram



My work, as a professional software engineer, should be understandable by any professional software engineer. It is important to have processes in place should anyone have to take my place. Small companies may have limited resources and should not have to spend them on someone redoing work due to poor software engineering methods.

Requirements

Past Method Failure

What was not working with my past method for requirements gathering was the lack of the detail of the requirements. My requirements document was typically a single hand written page of notes and never a detailed analysis of the requirements. I was not able to work properly with the users to uncover anything other than the most obvious requirements and this always led to problems later in the project. As stated previously, most requirement meetings I attended were one on one with users and there wasn't anyone else to review my work. This was an unavoidable situation that perpetuated poor requirements elicitation.

In a small firm time is limited. When I need to have a requirements interview I will usually ask when a person will have some time to talk to me about requirements on a new project or a new feature on an existing project. Sometimes I will just watch and listen to people doing their work. I can get a feel for the problems they encounter when they are working on their every day jobs. Many of their procedures have evolved over time but aren't necessarily the best way of doing things and sometimes they just don't know what they want or need.

One observation I made once was a purchasing agent who was having problems with a search utility in a program his company had purchased. He could not do multiple searches without exiting the module he was working in and then reentering the module and restarting his search queue. He became very frustrated and finally got up from his desk to get away from the problem. He took 10 minutes to collect himself and then returned to the same frustrating exercise. He looked at me and said

he had a lot to do and maybe we could schedule a meeting later. He did not realize how much he actually had already said. I discovered that everyone using that software product experienced similar problems. When the company bought the product there was no formal requirements meeting with all of the stakeholders to discuss whether the product had all of the functionality they required. The result is a substantial investment that is actually raising the cost of their operation.

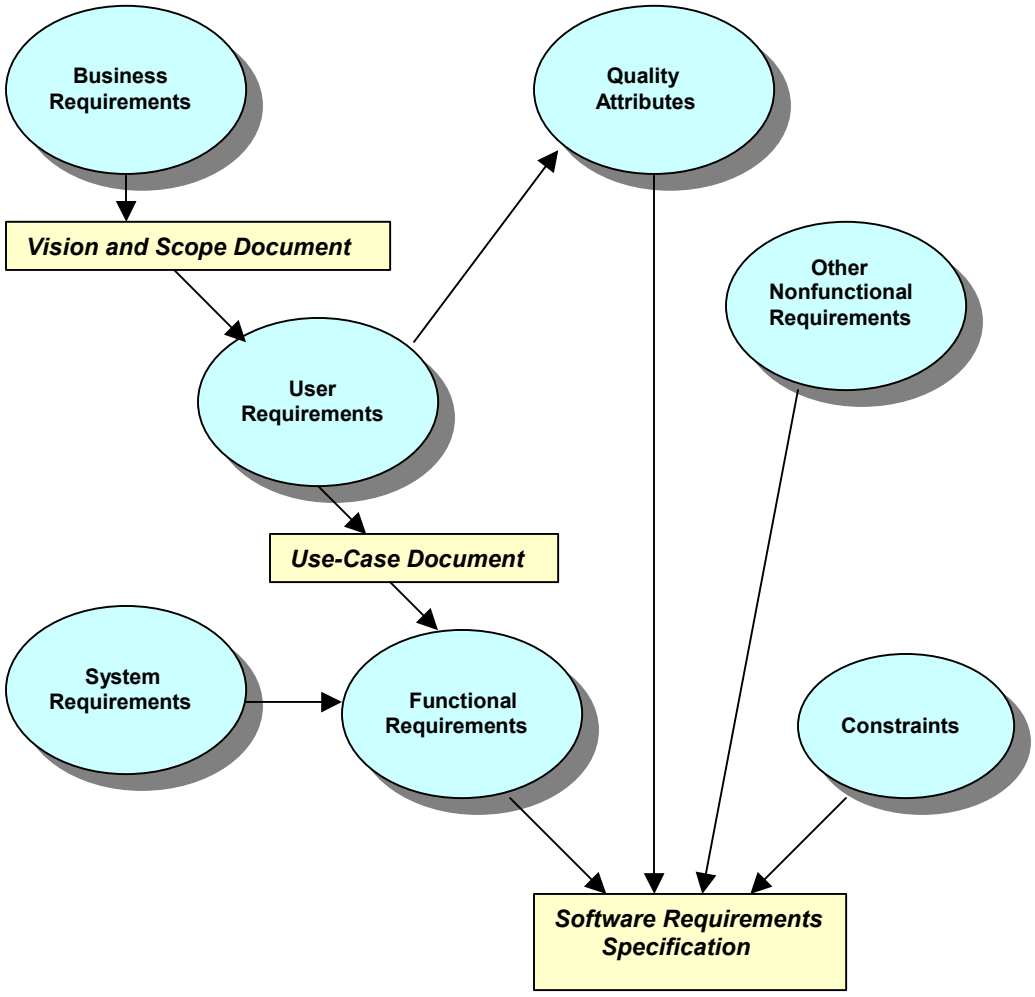
At another office I noticed a young secretary reading a 2 1/2 inch thick manual trying to use a program she needed for doing some task, but she could not understand the procedure as it was written in the manual and she was afraid of making a mistake. She said she had never had to do anything like that before and she really did not understand computers anyway.

New Approach

Both of the above scenarios relate to a poor requirements process that failed to uncover necessary requirements that would have, if implemented, prevented the problems in the above examples from occurring. These sort of problems are brought on by the fact that the software industry is still evolving and has not yet reached the point, as traditional engineering disciplines have, where there is always a procedure, plan or drawings, that works for all situations, to show how to elicit requirements. The problem is that with poor requirements engineering, poor projects are produced. Many times programs are accepted “as is” and people continually struggle to use sub-standard software. Figure 19 is a diagram of an accepted requirements development process that I have adopted and now use. The end result is a software requirements specification that represents the product to be built. Following this procedure will not always guarantee successful requirements elicitation but if no

process is followed one can guarantee a correct product that fits the clients needs, will never be produced.

Figure 19 Components of Software Requirements[18]



[18] Karl E. Wiegers, **Software Requirements** (Microsoft Press, 1999) 8

The software requirement process, as shown in figure 19 gives the software engineer the tools necessary for proper requirements engineering but it still remains a very personal process. The software engineer becomes a requirements analyst and must gain adequate domain knowledge in order to ask the right questions and recognize key requirements. A software engineer working with a small industry client must learn how users do their work and gain understanding into what their particular individual requirements are. He must learn how to deal with users who will not cooperate due to a fear of the new software affecting their jobs. People get used to doing things a certain way and do not necessarily want to change.

Gaining domain knowledge is the key to requirements engineering. I have achieved considerable domain knowledge at my company that has helped me develop the MMS project. Because I have that knowledge, it has helped me uncover vague requirements that I otherwise may have missed. I have an in-depth understanding of the various department responsibilities within the company and I understand what the software is intended to achieve. By using the requirements model I am able to build a library of techniques that I can use with other projects.

Now when I am working on a project where I am not domain literate, I conduct research into the type of industry or problem, to learn enough to start requirement interviews. It is vital that I speak the business language of my clients since it is my responsibility to communicate with them and not theirs with me. I really don't have anyone who can critique my interview results, so I must be confident that my methods are sound. I strive to build a list of business requirements first and then develop a vision and scope document that is reviewed by the client for approval. I do not proceed further until I have an approval of the vision and scope document.

In the past I would not have created the vision and scope document and I would not have used any formal process to develop the business rules. I would have created a list of requirements but it would not have been organized into requirements types. I usually just took notes of an informal requirements meeting and worked from that list by-passing many software engineering steps. As I have already mentioned I really did not view software development as an engineering exercise. Many developers in small industry fit into this category. Figure 19 is the system I have adopted for requirements elicitation.

Figure 20 Software Requirements Specification Template[19]

Software Requirements Specification Template
1. **Introduction:** Presents an overview of the Software Requirements Specification to help readers understand the document. 1.1 Purpose (Identify the product.) 1.2 Document Conventions (Describe any standards or typographical conventions.) 1.3 Intended Audience and Reading Suggestions (List the readers to whom the SRS is directed.) 1.4 Product Vision and Scope (Provide a short description of the software being specified and its purpose.) 1.5 References (List any documents or other resources to which this SRS refers.)
2. **Overall Description:** Presents a high-level overview of the product being specified.

[19] Karl E. Wiegers, **Software Requirements** (Microsoft Press, 1999) 154

2.1 Product Perspective (Describe whether this product is new or part of an existing larger system.)

2.2 Product Functions (Summarize the major functions the software must perform.)
2.2.1 User Classes and Characteristics (Describe the users who will be using this software and their characteristics.)

2.3 Operating Environment (Describe the environment this software will be used in including operating systems and hardware.)

2.4 Design and Implementation Constraints (List any constraints that will impede developers in the development of this software.)

2.5 Assumptions and Dependencies (List any assumptions that may affect the listed requirements of the SRS.)

3. External Interface Requirements: Specifies any requirements that ensure the new product will connect properly to external components.

3.1 User Interfaces (List any components on the GUIs that will be required.)
Example: The following corporate users will access their user functions through secure GUIs that cannot be accessed by the general public users.

3.2 Hardware Interfaces (List any hardware interfaces that will be required with this software.)
Example: A broadband or 56K modem internet connection will be required.

3.3 Software Interfaces (Describe connections between this software and other existing or planned software.)

3.4 Communication Interfaces (Describe the requirements that that will require any communications. Network, Web, ect)
Example: The internet connection will require a broadband or 56 K modem connection.

4. **System Features:** Shows the functional requirements organized by system features and the major services provided by the product.
4.1 System Feature:

4.1.1 Description and priority.
4.1.2 Stimulus/Response Sequences.
4.1.3 Functional Requirements.

5. Other Nonfunctional Requirements: List any nonfunctional requirements

other than external interface requirements and constraints.

5.1 Performance Requirements
5.2 Safety Requirements
5.3 Security Requirements
5.4 Software Quality Attributes
5.5 Business Rules
5.6 Business requirements
5.7 Functional Requirements

6. Other Requirements: Define any other requirements not covered by this SRS. (List any requirements that are not directly covered in the SRS. Expansion, Ect)

Appendix A: Glossary
This section is for defining terms pertinent to the SRS document.

Appendix B: Analysis Models
This section will house any required analysis models and diagrams.

The following is the Purpose and Product Vision and Scope for the MMS system.

1.1 Purpose.

The analytical purpose of the Manufacturing Management System project is to provide a system that corporate and production planners can use to perform research utilizing the historical data acquired by the system. This research will facilitate in-depth cost reduction studies of the manufacturing system and processes to ultimately improve profitability. The warehoused data will also be used to analyze production techniques that can be incorporated into new products during design. This will help reduce proto-type analysis and testing time for individual components and entire mechanical systems.

1.4 Product Vision and Scope.

The manufacturing company I work for needs a system to continually collect manufacturing data on production items. The system will require a database that will store all manufacturing data as production parts are produced. The data will be used for analysis of trends in production times and processes at various stages of production. The resulting analysis will be used to continually improve all phases of the production process from tooling, materials used, and product engineering. Manufacturing personnel will log onto a production job by scanning a production card, at their workstation, for a specified production part. Any relevant information about recommended improvements will be entered into the database at each workstation. As the production item

> progresses to each required process the data for that process will be entered. At completion the historical data will be incorporated with previous collected data for future analysis. The goal is to incorporate data warehouse technologies into the design of this system.

I never considered the Purpose or the Vision and Scope document as an important part of development. I felt that it should be very obvious what the project was and what it was going to include. I have had many meetings where the vision of a project was discussed informally and the order given to proceed. This seems like the easy way to begin but what usually will happen are problems later with misunderstandings of what the project actually is. I have been there many times. I did take notes during these meetings but never placed the context of my notes in a formal document for approval. Using a formal process and getting approval will usually get the project off to a good start. It is important to point out here that, at the time I was not intentionally trying to save time by not collecting project information. I was just using the method I had been using for years. This is one of those times during which a small industry software engineer has to be self-critical and non-defensive about past work and methodologies and move forward.

In the past while working on internal projects, many times it was assumed, by my supervisors, that I knew what they wanted and they left all of the definitions in my hands. This was a poor position in which to be placed, but I accepted the challenge without a second thought. Now, after my MSSE class work, I would never begin without the formal process. In retrospect, I can see times when I have actually increased the cost of projects to the company because of my lack of proper software engineering practices. I was not alone in this practice. I have met many people in similar positions doing the same thing. It seemed fast and expeditious.

I recently encountered a problem from not using a formal requirements process on the Jtrack project I discussed earlier. I was asked to add territory functionality to the order entry module. It seems simple enough but will actually require considerable work on my part. Had I used proper software engineering procedures from the beginning I would have identified the territory requirement as a business rule that would have been describe in the vision and scope document. It would have been a much simpler implementation had it been addressed in the beginning of the project or at least in one of the first iterations of the development model. This work will be added expense to the company. In my defense the Jtrack system has saved the company a great deal of money. Nevertheless, the reality is it could have been much better design.

Use cases

A very important requirements development tool is the use case. Each of the MSSE classes discussed use cases but SENG 691K Software Requirements Engineering expanded on the use of use cases and use case diagrams. Figure 21 is Use Case #2 for the MMS project. It demonstrates my use of the standard Use Case Template for an in-depth description of a requirement.

Figure 21 Use Case for MMS

Use Case Template		
Project # : MMS10	Developer: Jim FLeming	Date: 08-15-2003
Requirement #: 2		
Use Case #: 2		
CHARACTERISTIC INFORMATION		

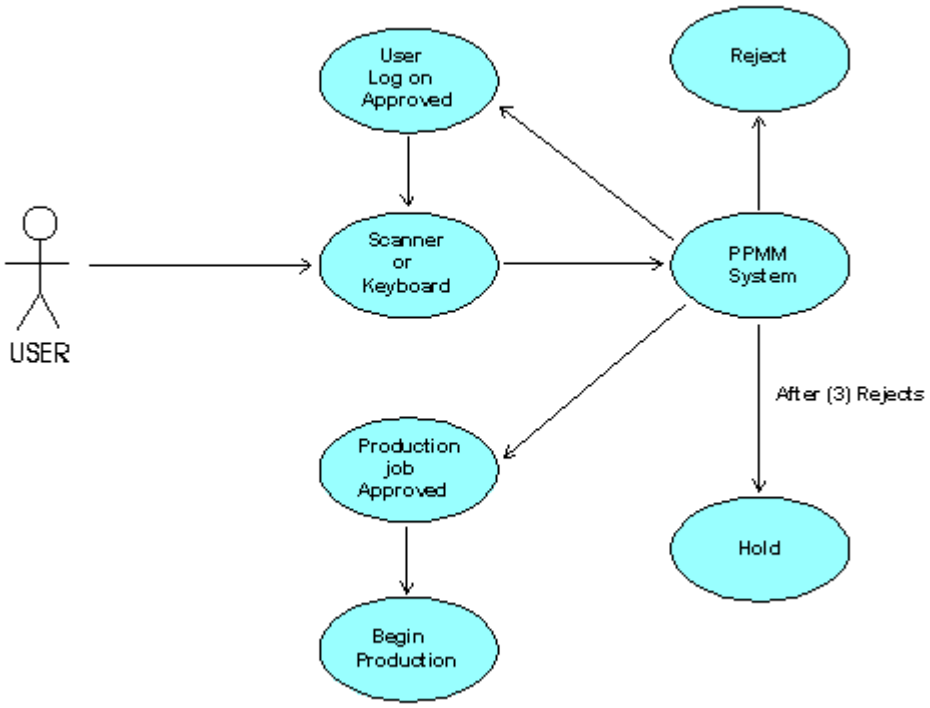
Goal in Context:	A method for machine operators (users) to log onto the MMS system to start a production job using a bar coded ID and job card or the keyboard.
Scope:	Production log on module of MMS.
Level:	User type is Machine Operator.
Preconditions:	1. Each workstation computer must be connected to the company LAN. 2. Each workstation must have a bar code reader. 3. Each machine operator must have a valid employee number. 4. Each machine operator must have a valid production job card to scan. 5. Company database must have machine operator's employee number filed. 6. Company database must have the production job filed.
Successful End Condition:	Machine operator has valid employee number and successfully logs onto the MMS system to start production on scanned job.
Failed End Condition:	Machine operator cannot log onto MMS system and production is not started.
Primary Actor:	Machine operator.
Trigger:	Machine operator must logon to the MMS system to start a production job.
MAIN SUCCESS SCENARIO:	1. Production job issued. 2. Machine operator scans his employee card with card scanner and awaits approval. 3. The MMS system displays to the user acknowledgement of the scanning operation. 4. The MMS system verifies the employee number in the database. 5. Machine operator is logged onto the MMS system. 6. The machine operator now scans in the production job.

	7. The MMS verifies the production job scanned. 8. Machine operator is successfully logged onto system and production job. 9. The machine operator can now start production.
EXCEPTIONS:	2a. The machine operator will be given (3) attempts to log onto the system. 2a1. After (3) failed attempts the system will prompt for manual entry of the machine operator's employee number. 2a2. If scanning and manual logon attempts fail then the MMS system will send an alert to the supervisor station and display a holding security screen at the machine operator's workstation. 6a. The machine operator will be given (3) attempts to log onto a production job. 6a1. If after (3) attempts the system will prompt for manual entry of the production job.

Figure 22 is a use case diagram for the above use case. The use case diagram is a good tool for presenting a use case scenario to a client for clarification of a requirement or just plotting a solution for my own understanding. One of the problems I have encountered in the past when working with users on a requirements list is that I assumed that the user understood what I was talking about when I would ask them about a particular requirement. They would acknowledge my question but were not that cooperative. Consequently, I often felt that I did not get all the information that I needed. I would set and pencil sketch crude flowcharts but produce nothing of professional caliber that I could submit in a formal document. Requirements engineering is a complicated task especially if you are working with limited domain knowledge. I have found that the use case diagram depicting a formal

use case scenario description has helped me to take an engineering approach to the requirements elicitation process and to produce a more defined requirements document. This is a formal process that I did not use in the past and is a direct result of my MSSE work.

Figure 22 Use Case Diagram for MMS Use Case # 2



Often when I present a use case diagram to a user to insure that I got the requirement correct they have additional facts about the requirement. This leads to less development delays and allows me to proceed in a professional manner giving me more credibility as I continue to improve all of the company's processes.

When I have to present a requirement in more technical detail for analysis I now use an analytical class model. The following table, figure 23, breaks the use case down into class objects that can then be placed into a model of the use case.

Figure 23 Class object table for use case 2 for MMS

Studies of use case 2 suggest that the following will be part of the system.	
Controller objects:	Production Timer. The MMS system.
Boundary objects:	Work station. Bar code scanner. Company database. Menu GUI.
Entity objects:	Employee ID bar code card. Production job ID bar code card.

Breaking complicated use cases down to class object levels greatly enhances the analysis of the use case and can uncover hidden details of a requirement that otherwise may have been missed. I am still sometimes amazed at how poor my original methods of requirements engineering were after all of the years I have worked in software engineering. I do believe however, that all of my years of experience allowed me to grasp these formal procedures easier than if I had never worked in the profession of software engineering before. The only problem is that I continually have to explain why I am changing my methods now. It was awkward at times explaining that I really was not doing it right all of these years.

Figure 24 is the analytical class model of use case 2 for the MMS project.

Figure 24 Analytical Class model for use case 2

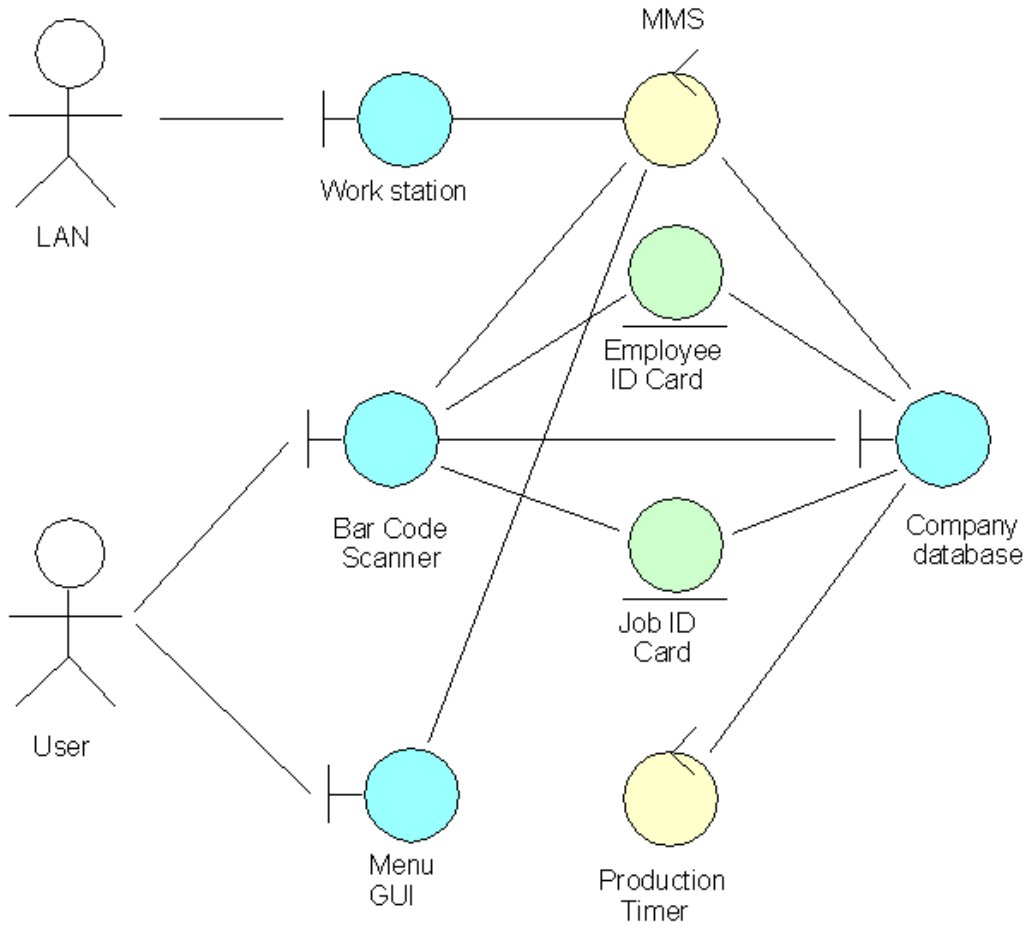
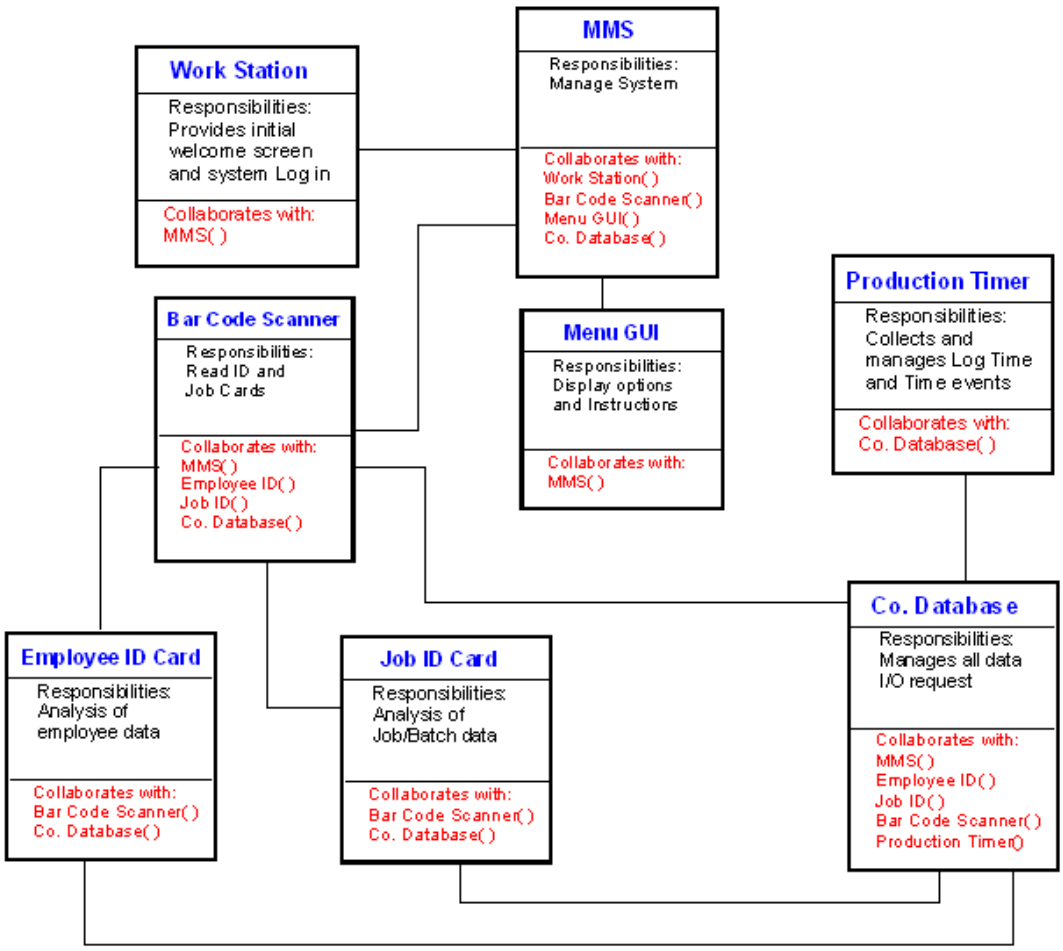


Figure 25 shows the class object responsibility and collaboration model for use case 2 of the MMS. This technique is a good way to map out and understand how the different objects interact with each other. It is a more technical model that I use to help me with implementing a requirement. In the past I would use, at this point, only flow charts at code level thus missing a complete engineering step that could have help me with the requirements definition. This model is definitely an improvement over my past method.

Figure 25 Class object Responsibility and Collaboration model for use case 2 of the MMS



All of the processes I have discussed so far have been an immense help to me in several areas. My abilities to communicate at a more professional level with clients has helped me project a more professional organizational image and I now have the respect of my coworkers and the company owners. One can teach old dogs new tricks. My software engineering abilities are now at a professional engineering level as opposed to before my MSSE involvement when everything I did was improvised and not completely effective. My software also works and serves the users better.

My project management skills are also greatly improved. This has helped me on all of the projects I manage, not just the software projects.

Data Management

Data is the life force of any company. Programs are stored and served, Reports are generated, Payrolls are managed, Graphics are produced, Engineering drawings are developed and distributed, and Documents are created, all using existing or newly generated data files that must be maintained.

Implementing Data Management in the MMS Project

I am also in charge of the operation and maintenance of a 32 station Local Area Network at the company where I work. The network is configured as a client server 100 MBPS Ethernet LAN. Some of the data is located on individual computer hard drives and removable storage media but the bulk of the data is stored on two network servers though it is not housed in a way that the data can be analyzed in any real useful way.

The class SENG 691L Data Warehousing addressed data management and data warehouse technologies. I had been working at consolidating all of the various data sources into a common database but after completing SENG 961L I have decided to take another approach that has led to the main concept of the MMS project.

The vision and scope document for the MMS project that I previously discussed covers the business rules for creating a database using data warehouse technologies. I am extremely interested in this technology as the analysis of

manufacturing data has enormous cost saving potential for a small industrial equipment manufacturer.

In the following section I will discuss the current infrastructure of the company's LAN and data storage. I will also discuss the future infrastructure that I am now working towards.

Current System Infrastructure

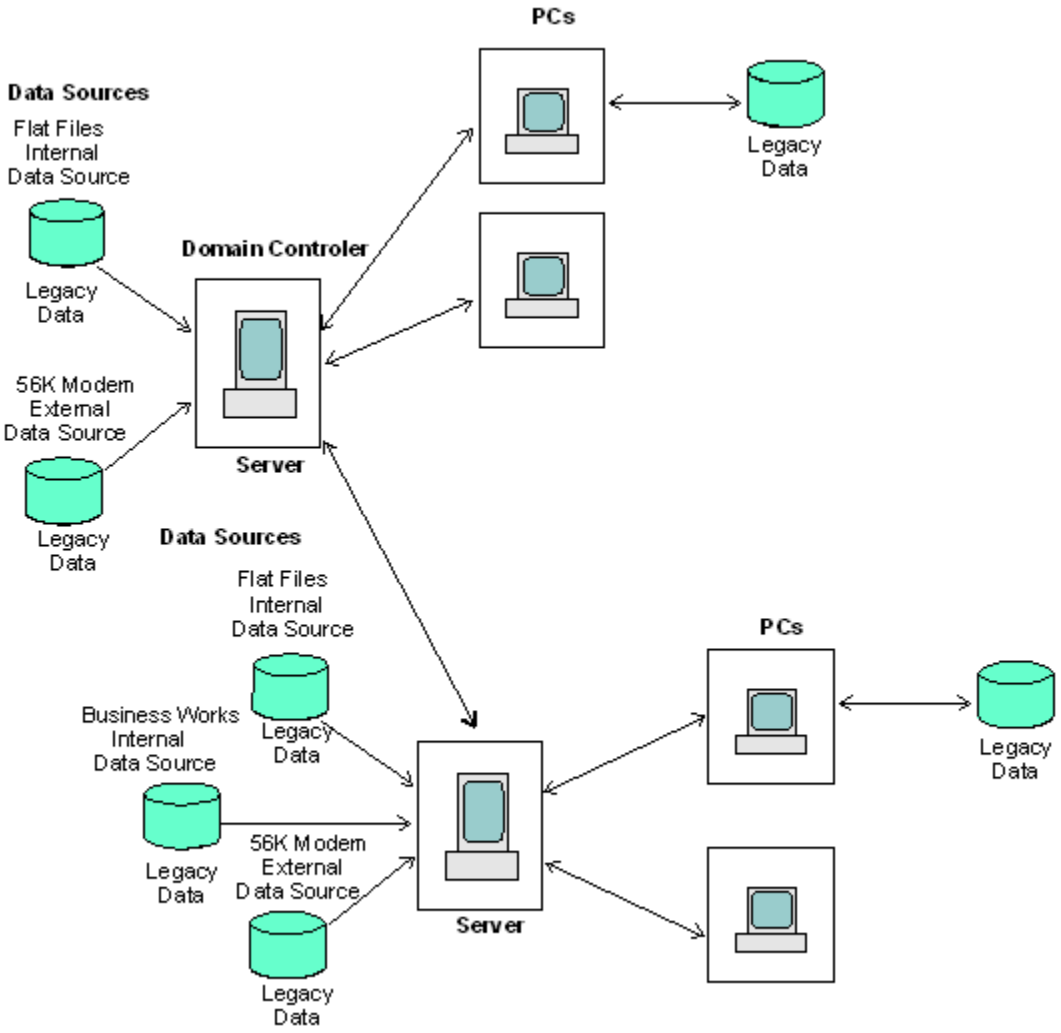
> The existing infrastructure is fragmented in that data is collected from multiple non-conformed sources and is never processed. There is no way to perform any analysis unless special programs are developed to retrieve legacy data and perform analysis only on that specific data. This has led to a scenario where data is collected and stored but never used. Reports that are generated are often inaccurate due to dirty data.

Hardware and system capabilities currently in place:

- There is a multiple processor Dell 2500 Power Edge server running the Windows 2000 server operating system with 78-gigabyte storage capacity and a multiple processor Dell Power Edge 2400 server running the Windows NT operating system with 20-gigabyte storage capacity. Both of these systems will be used in the data staging area to capture and prepare the data to be used in the warehouse.
- There are 30 client Desktop PCs running the Windows Millennium or XP professional operating system. All PCs are members of the network.
- Internal data sources connect over a 100 MBPS Ethernet LAN using the client server configuration.
- External data sources connect over a 56K phone line or removable media.
- Sporadically place hubs with hard to trace cables.

Figure 26 shows the current architecture of the company LAN and computer systems.

Figure 26 Current company network architecture



The LAN in its original configuration has evolved over time and was not constructed with any particular plan in mind. When more functionality was required it was just added to the existing system. The result has been a system that has a very fragmented database structure. Users store some data on their local hard drives that other users may need. When the request comes to access these independent data

stores I have to link the users so the data can be accessed. Because there is no design or formal process, the practice has continued. The two servers in the LAN house the inventory and accounting databases. The inventory and accounting data is also fragmented in that there are duplicated data files that are not properly managed or maintained.

Implementing Data Warehouse Technologies

When I was placed in charge of the LAN, the system was already established. My first objective was to start setting up some security measures for the database. Security was basically non-existent and the entire database system was at risk. There were no formal backup procedures in place and much of the localized legacy data was never backed up. I was in the process of centralizing some of the legacy data structures so those who needed access could get it without me having to link computers to perform data transfers when I became involved with the MSSE program. One big problem I had trying to implement changes was that people had an extreme ownership of the legacy data stored on their local drives and got really nervous when I told them that I was going to move it to one of the servers.

The New Infrastructure

The way in which the LAN was setup and being used was totally unacceptable. I started to work on a new concept of restructuring the entire LAN and databases to conform with the migration to data warehouse technology. Migrating to data warehouse technology will take time and money. It will also take educating the users to the advantages of data warehousing.

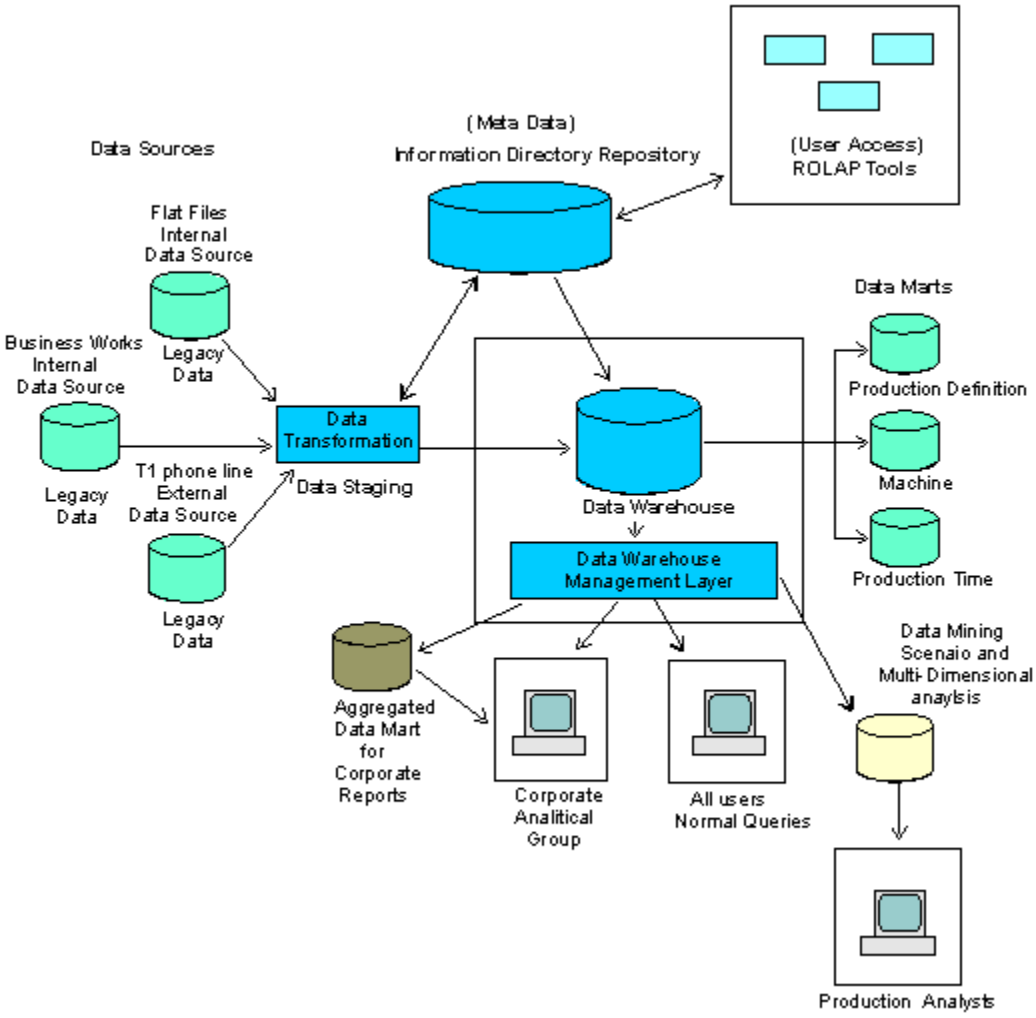
It is not practical to try to implement a data warehouse all at once. Data warehouses are a collection of data marts so my plan is to start with basic data marts and gradually build the system into a data warehouse configuration. The following is a list of additional resources that will be required to begin implementation of data warehouse technologies.

Additional hardware and system capabilities required:

- One additional 2600 Dell Power Edge server with a minimum storage capacity of 500 Gigabytes. This system will be used as the information directory repository and application server. It will utilize the Windows 2000 server operating system.
- A Broadband Internet connection ported through a router and connected to the LAN with firewall protection.
- All the hubs will be have to be relocated to a common area.
- All cabling must be upgraded to high-speed category 5 UTP certified.
- Warehouse design will keep the client server configuration.
- User access will be through the new 2600 Power Edge application server where all meta-data and data marts will be housed.

Of all the changes I have made in the corporate processes that I have discussed in this paper, the data warehouse project is the most aggressive and challenging. This project requires a real commitment by the corporate executives and a total acknowledgement of the viability of software engineering practices. The decision to move forward is a milestone in my efforts to put into effect what I have learned in my MSSE work. Figure 27 is the proposed architecture for the data warehouse project.

Figure 27 Proposed Data Warehouse Architecture model



Data Warehouse Process Description

The following is a brief description of the processes of the proposed data warehouse.

- Data will be collected from the various legacy sources and placed into the staging area of the warehouse. This data will be cleansed and placed into SQL relational data tables.

- Processed tables will reside on a 2500 Dell Power Edge server where further processing will occur before these tables are used to populate the data marts located on the 2600 Dell Power Edge application server.

- Users will access the data through the (Meta Data) Information Directory Repository housed on the 2600 Dell Power Edge application server. All user access will be controlled through the Data Warehouse Management layer.

Relational On-Line Analytical Process, ROLAP, tools will be used for all user data queries.

- There will be (1) Aggregated data mart that will be used exclusively for corporate reports. The data in this data mart will be replicated from pertinent data housed in the (3) main data marts and then aggregated at the proper grain for the necessary reports.

Phase 1 of the implement of the data warehouse

Phase 1 Scope Definition

Based on requirements gathered from the business, 4 data marts will be required to fulfill the business requirements.

Data Mart (1) Production Definition, will house the product definition data.

- Product part number and name dimension.
- Product drawing file name and descriptions dimension.
- Material codes and requirements dimension.
- Unit of measures and preparations dimension.

Data Mart (2) Machine, will house the production machines data.

- Machine descriptions and ID codes dimension.
- Production sequence numbers dimension.
- Machine production charge dimension.
- Tool package dimension.
- Operator dimension.

Data Mart (3) Production Time, will house the historical and collected production times data.

- Historical production run time including setup, sequence number, cycle, and inspection times and dates dimension.
- Current production sequence dimension.
- Current production cycle time and date dimension.
- Current inspection time and date dimension.

Data Mart (4) Reports, will house aggregated data for executive reports

- Machine dimension.
- Employee Dimension.
- Product Dimension.

The data mart matrix, Figure 28, shows the relationship between the data marts and dimensions.

Figure 28 Data Mart Matrix

Data Marts	Product # and name	Material Codes	Units of measure	Drawing files		Machine ID & description	Production charge	Tool package	Operator	Production sequence		Historical production cycle data	Sequence cycle time	Inspection	Sequence definition		Machine	Employee	Product	
Product definition	X	X	X	X																
Machine	X					X	X	X	X				X	X	X					
Production Time		X	X	X		X	X		X	X		X	X	X	X					
Reports	X					X			X			X					X	X	X	
					Dimensions ➔															

The following is a description of all the data mart star schemas.

Data Mart (1) Product definition Dimension Descriptions:

Schema Fact table:

- The Part_Number_Key joins the Product# and Name dimension.
- The Drawing_Key joins the Drawing Files dimension.
- The Material_Key joins the Material Codes dimension.
- The Unit_Measure_Key joins the Units of Measure dimension.

Schema Fact Table Facts:

- Product# fact captures the product part#.
- Product description captures the product name.
- Cad File captures the cad file data for the part drawing.
- Part Role captures the criticality of the part.

- Material Code captures the unique code for the type of material used to produce the part.
- Material Certs. captures the certification data for the materials used.
- Measure units captures the unit of measure for stock lengths or quantities.
- Preparations captures unique data about special material preparations prior to manufacturing.

Schema Dimension (Product part number and name dimension).
Attributes:

- The product part number will be a 1 to 10 character alphanumeric code that will be unique to each part. (example: 7304Brg)
- The name will be from 1 to 50 character length code representing the physical name of the part. (example: Ball type 30 degree angular thrust bearing)
- The part role is a criticality indicator as to the importance of the part in relation to other manufactured items. (This product must be finished on schedule).
- Bin Location is the physical storage area for the product. (example: BL1023)
- Last Batch Code represents the last production run of this product. This will be for historical reference.

Schema Dimension (Product drawing file name and descriptions dimension.)
Attributes:

- The product drawing file name will be a 1 to 50 character file name representing a unique drawing name.
- CAD file is a unique data file name for the engineering drawing representing the part.
- Drawing approval date is the authorization date representing when the part cad file was released for production.
- Drawing signature records who authorized the release of the cad file to production.
- The CNC File name is the file name for the CNC program used to produce the part on the CNC production machines.
- The CNC approval date is the date that the CNC program was released to production.
- The CNC signature records who authorized the release of the CNC program for production.

Schema Dimension (Unit of measures and preparations dimension).
Attributes:

- The Units of measure code will be 1 to 30 character alphanumeric code representing the quantity of a material used to make a particular part.
- (example: cut length 10.500 inches)

- The stock length code designates the length or quantity of stock, using the Units of Measure code, is required for each part piece.

Schema Dimension (Material codes and requirements dimension).
Attributes:

- The material code will be a 1 to 10 character alphanumeric code representing specific material type a part is made of. (example: M1023)
- Material description is textual description of the material.
- The Material Certs. will describe any special material inspections or certifications to be confirmed before proceeding with manufacturing.
- The preparations field will contain any special material preparations that must be performed before manufacturing the part.
- The Vendor code will identify the vendor who supplied the material.
- The Bin location represents the physical location the material is stored. (example BRM102).

Data Mart (2) Machine Dimension Description:

Schema Fact table:

- Machine_Key joins the Machine ID & description dimension.
- Sequence_Key joins the Production Sequence dimension.
- Operator_Key joins the Operator dimension.
- Tools_Key joins the tools package dimension.
- Charge_Key joins the Production charge dimension.

Schema Fact Table Facts:

- Machine code captures the unique identifier for a particular production machine.
- Machine description captures the textual description of a production machine.
- Manf. Sequence captures the position in the manufacturing process the identified machine is used.
- Operator Identification captures the operators unique ID number.
- Tool Package ID captures the unique code identifying the tools used on the CNC machine to produce the product.
- Tool Grade captures the unique tool insert grade used with the identified tool package.
- Machine charge captures the charge rate of the CNC machine identified.
- Machine overhead captures unique additional charges for the identified CNC machine tool.

Schema Dimension (Machine descriptions and ID codes dimension).
Attributes:

- The ID code will be a unique 1 to 20 digit code representing a particular machine. (example: Mach10)
- The Machine description is a unique text description of any machine that is used in the manufacturing of a given part. This description will also contain any special attachments required for the manufacturing of the part. (example: 3040 Sheldon horizontal CNC milling machine with hyperdex indexing head attachment installed.)
- The Control type code represents a particular type of machine control system being used.
- Last Maintenance represents the last date that maintenance was performed on a particular machine.

Schema Dimension (Production sequence numbers dimension).
Attributes:

- The Manufacturing sequence number field is a placement number in the manufacturing sequence. (example: 5 , the Mach10 is the 5^{th} machine in the manufacturing process of a particular part.)
- The Production Code is a unique code identifying the type of process a for which a machine is used.

Schema Dimension (Operator dimension).
Attributes:

- The Employee code field is a 1 to 5-digit code that represents an operator's unique employee code. (example: EMP52)
- Experience code represents the experience grade of an operator.
- The Evaluation field is a textual rendering of the operator's last evaluation.

Schema Dimension (Tool package dimension).
Attributes:

- The Tool package field is a 1 to 10 character alphanumeric code that represents a tool configuration package that describes the type of tool holders. (example: TPK10, representing tool package configuration 10).
- Tools field indicates insert grade required for a specific manufacturing sequence.
- Validated is the date the tools package was released to production.
- Special instructions represent any special instructions pertaining to the selected tools package.
- Tool technician is an identification of the personnel who prepared the tools package.

Schema Dimension (Machine production charge dimension).
Attributes:

- The Setup rate field represents a unique charge for setup time.
- The Machine production charge is a value placed on a particular machine, production rate, for a given process in the manufacturing of a part. (example: $120.00PH for 120.00 per hourly charge rate.) This charge represents a cost of overhead for a particular machine and is a constant attached to a machine in all calculations of part manufacturing cost.

Data Mart (3) Production Time Dimension Description:

Schema Fact Table:

- Product_History_Key joins the Historical Production Cycle Data dimension.
- Production_Sequence_Key joins the Sequence Definition dimension.
- Inspection_Key joins the Inspection dimension.
- Step_Cycle_Key joins the Sequence Cycle Time dimension.

Schema Fact table Facts:

- Historical cycle time captures the last actual time to complete an identified production cycle.
- Set-Up data captures pertinent set-up information for preparing for production.
- Current cycle time capture the current cycle time of production.
- Production sequence captures the sequence number of different stages of production.
- Step cycle time captures the time to complete the current production phase.

Schema Dimension (Historical production run time).
Attributes:

- The Part# field represents a unique number representing a part.
- The Part description field represents a textual description of a part.
- The Cad file filed is a unique file name for the cad data for the part drawing.
- The CNC file filed houses the file name of the CNC program used to produce the part.
- The Last Operator code field represents the identification of the last person who produced a part at a specified step cycle.

Schema Dimension (Sequence Definition).
Attributes:

- The Machine code field is a unique identifier for a particular machine.
- Production sequence captures the sequence number of different stages of production.
- The Interplant Transport time filed represents the elapsed time between step cycles.
- The Batch issue date field houses the date of Batch job creation.

- The Batch Completion date field houses the date that a batch job was completed.

Schema Dimension (Inspection).
Attributes:

- The Inspection code field represents the type of inspection that will be performed on a manufactured item.
- The Inspection Cycle time field houses the last inspection step cycle time.
- The Inspection Interval field is the interval of inspection on a production item during the manufacturing process. Is it inspected at each step or only at the end of production.
- The Specifications field houses procedures for inspection.
- The Comments field stores any pertinent information from last inspection.

Schema Dimension (Sequence Cycle Time).
Attributes:

- The last setup time field is the total accumulated step setup time of the last production of a particular part.
- The last cycle time field represents the last step cycle time for a production item.
- The current setup time field represents the current step cycle setup time.
- The current cycle time field is the current accumulated time on the current production item.

Data Mart (4) Reports Dimension Description

Schema Fact Table:

- Machine_Key joins the Machine dimension.
- Product_Key joins the Product dimension.
- Operator_Key joins the employee dimension.

Schema Fact Table Facts:

- Machine cost captures the overhead for a selected machine.
- Product cost captures the manufacturing cost of a production item.
- Operator cost captures the overhead in reference to a selected operator in reference to a production run.

Schema Dimension (Machine).
Attributes:

- The Machine code field contains a unique code representing a particular machine.
- The Machine rate is the cost per minute for a particular machine.

Schema Dimension (Product).
Attributes:

- The Product# field is a unique part identifier.
- The Description filed houses a textual description of the part.
- The Average cost field holds a pre-calculated average cost of a part.

Schema Dimension (Employee).
Attributes:

- The employee code field is a unique employee identifier.
- The Experience code field represents the selected employees experience in relation to their involvement in a particular production run.
- The evaluation field is the latest employee evaluation in regards to the current production run.

Figures 29, 30, 31, and 32 are the Star Schema Models for the described data marts.

Figure 29 Production Definition Star Schema Model

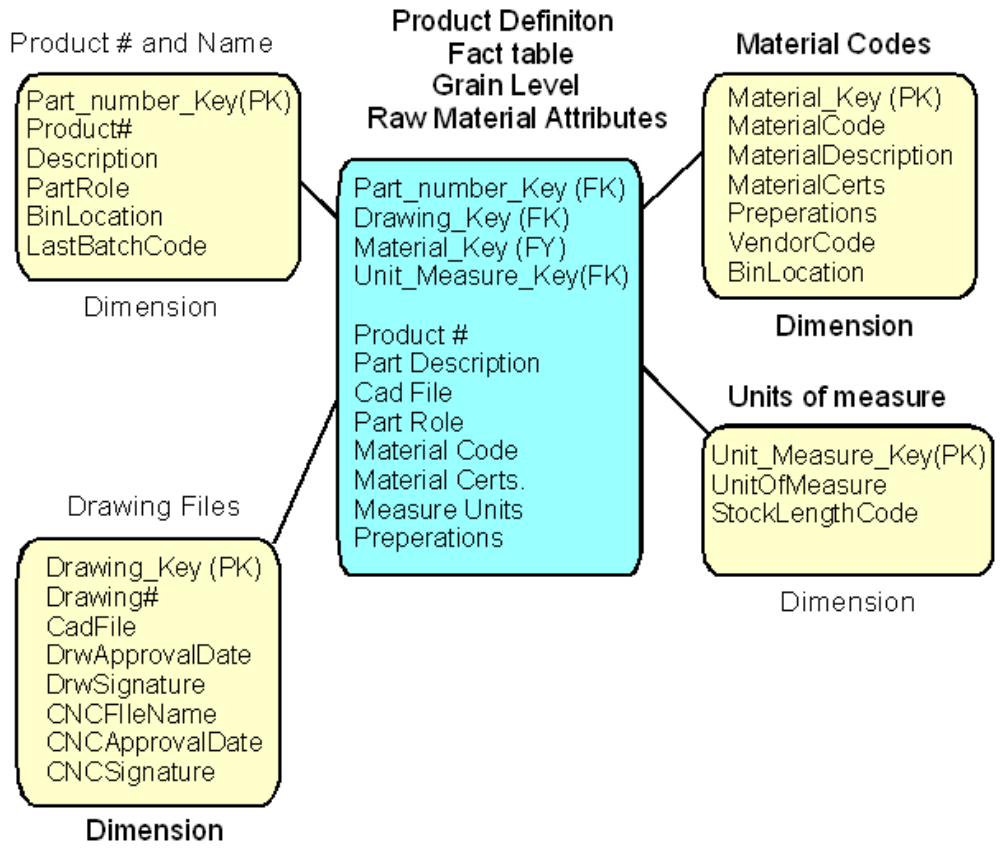


Figure 30 Production Machines Star Schema Model

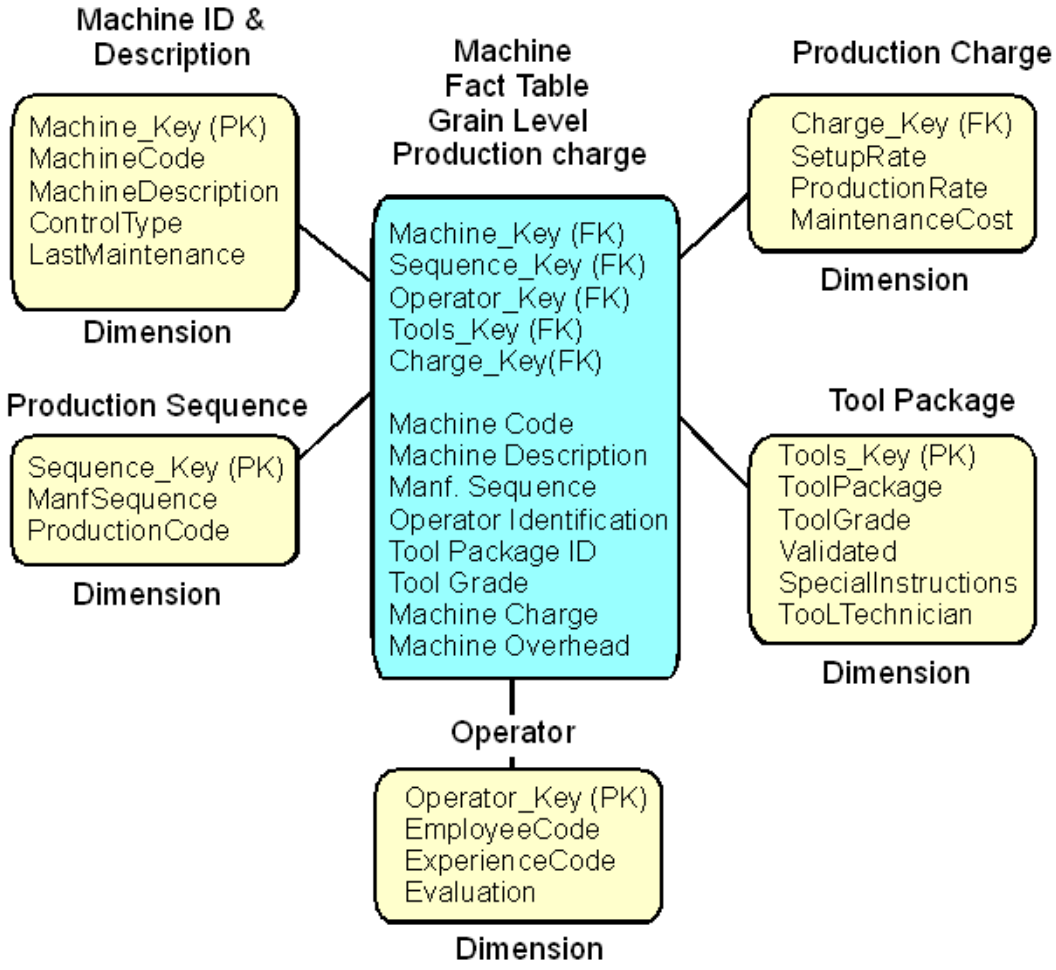


Figure 31 Production Time Star Schema Model

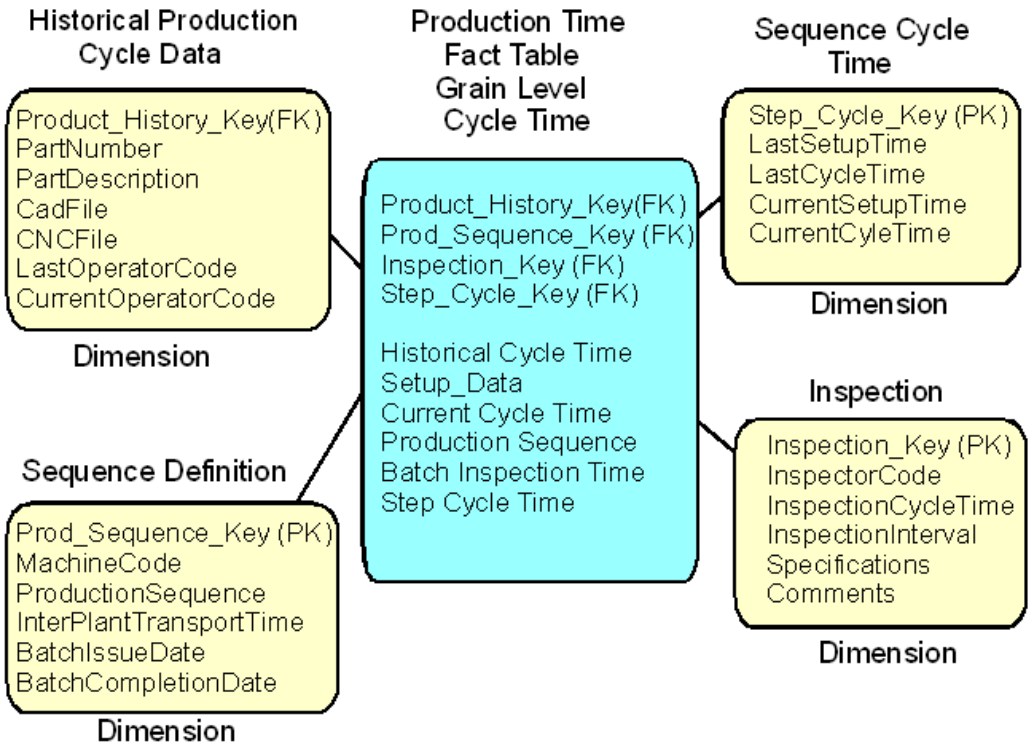
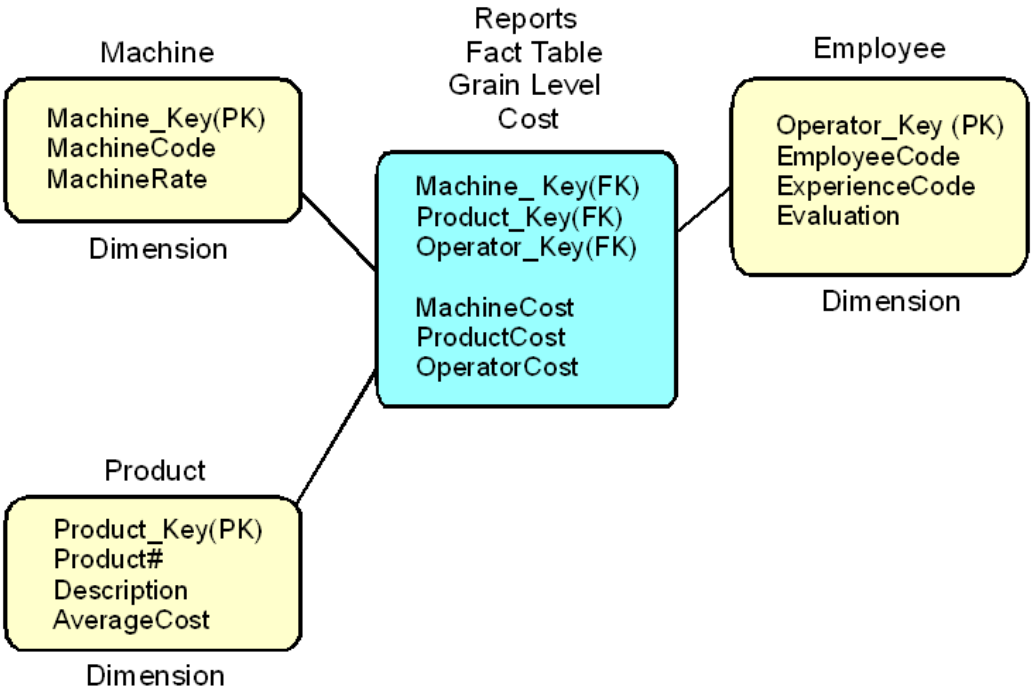


Figure 32 Reports Star Schema Model



Data Sources

Internal Legacy sources include:

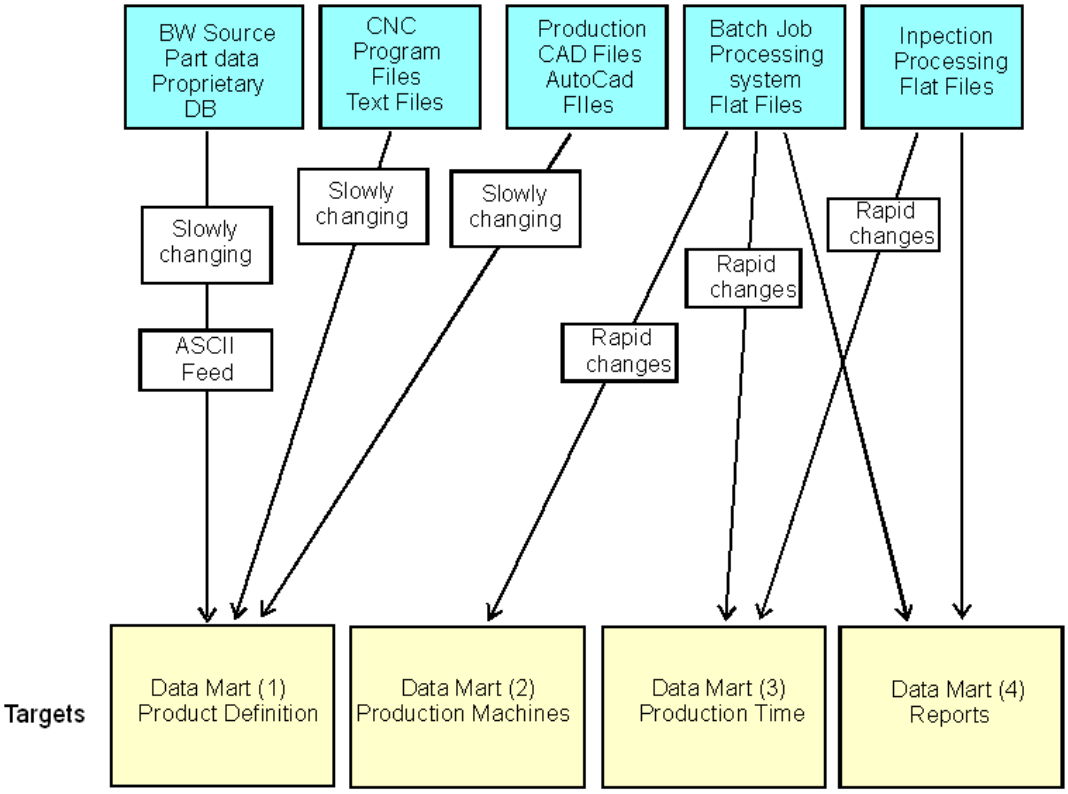
- Data collected during the manufacturing process at various data entry stations. This data is in the form of flat files.
- Product name and descriptions will be retrieved from a proprietary database via downloaded text files that will be translated.
- Engineering CAD data, drawing numbers ect., from existing fragmented data housed on various engineering workstations.
- Production data, tooling, NC program code, existing cycle time data, housed on various production planning stations. Some data is resident in Excel spread sheets and flat files.

External Legacy sources include:

- Research data from various engineering and design web pages.

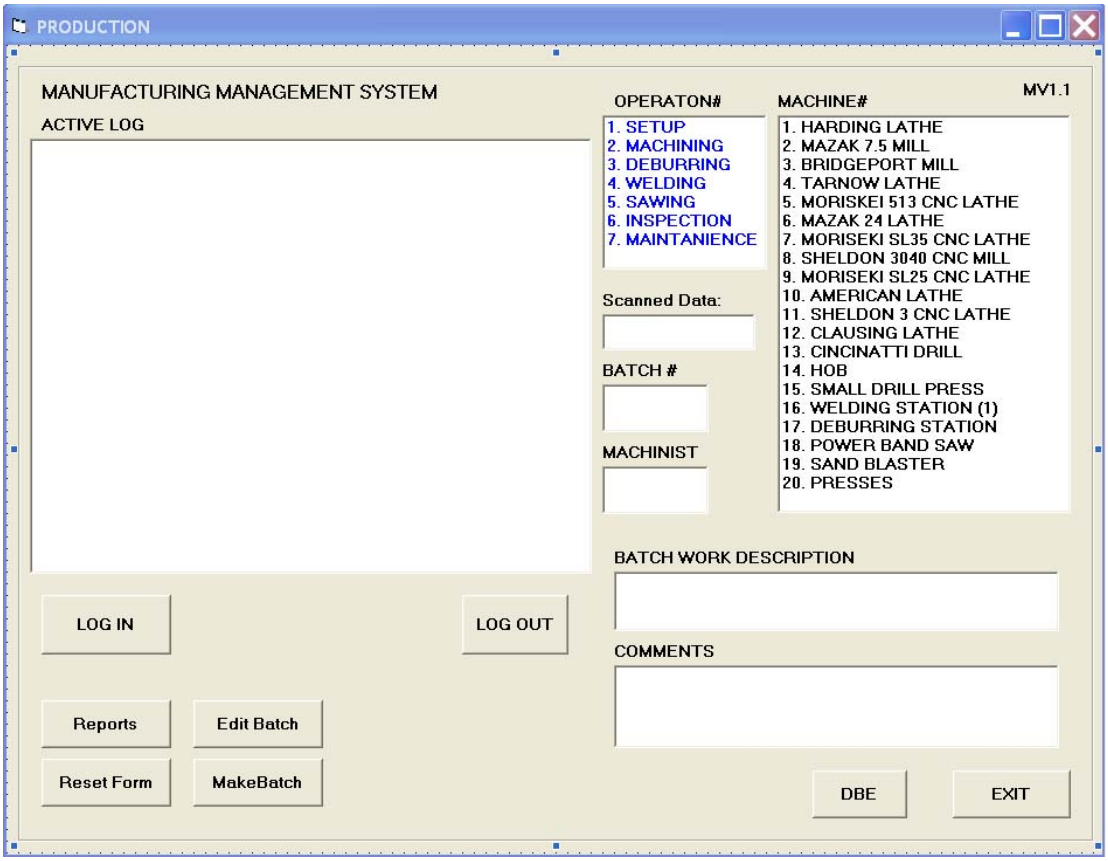
Figure 33 is the Legacy Source Map that maps the legacy data to a corresponding data mart. The map also shows the data activity, rapid or slow changing.

Figure 33 Legacy source Map



The data warehouse project has revolutionized the way my coworkers and I work with the company data. The MMS project has opened up new ways of managing production, and the data warehouse part of the project has made an enormous impact on controlling production cost. Figure 34 is a image of the main GUI for the MMS system.

FIGURE 34 MMS MAIN GUI



Validation & Verification

I would like to briefly touch on SENG 330 Validation and Verification. SENG 330 was my first Masters class. Most of the other classes covered V&V to some degree but one of the most important lessons I learned in SENG 330 was the type of discipline that is required for proper software engineering. The class covered development models and different verification and validation techniques. The class prepared me for the study habits and work dedication that is required to complete the Masters Degree in Software Engineering.

One of the most interesting topics covered was CARA, Criticality and Risk Assessment analysis[20]. I have not extensively utilized the method but I am presenting a CARA study on the MMS project to demonstrate the magnitude of change in my software development processes. I had never encountered a CARA study before but I believe it will be one of the tools that I will use in the future.

The following figures 35 and 36 are the Criticality and Risk Assessment analysis for the MMS system to determine the degree of testing that will be required for the listed functions. The knowledge I have gained in my Masters program techniques such as the CARA, have helped me make decisions about the amount of testing that can reasonably be done on larger systems.

[20] Robert O. Lewis, **Independent Verification and Validation** (John Wiley & Sons, Inc., 1992) 265

Figure 35 CARA Score Matrix for MMS project

			CARA Score Matrix for Manufacturing Management System						CARA score
	Criticality Score						Risk Score		Integer CxR
Function	perf/op	saf./sec.	cost/schd.	1	2	3	4	5	
1	3	2	2	2	2	2	2	2	4.666667
1.1	3	2	2	1	1	2	1	2	3.266667
1.2	1	1	1	2	2	1	1	2	1.6
2	4	4	4	3	2	3	3	3	11.2
2.1	3	3	4	2	2	3	2	3	8
2.2	2	2	3	2	3	2	2	3	5.6
3	3	4	3	3	2	2	2	2	7.333333
3.1	4	4	4	3	2	3	2	3	10.4
4	3	4	3	2	2	1	2	1	5.333333
5	2	3	2	2	1	1	1	2	3.266667
6	3	4	3	2	1	3	2	2	6.666667
7	3	3	2	2	1	2	2	2	4.8
8	3	1	2	1	2	2	2	1	3.2
	Functions List								
	1 User Interface								
		1.1 Screen layout and appearance consistent with exiting business system							
		1.2 Use multiple windows on screens when required							
	2 Data Security								
		2.1 Data base location.							
		2.2 Data file password protection							
	3 Data Access								
		3.1 File structure and type							
	4 Data Attributes								
	5 Program Scaleability								
	6 Network Compatibility								
	7 Error Handling								
	Risk List								
	1 Complexity								
	2 Maturity of technology								
	3 Requirements definition and stability								
	4 Testability								
	5 Developer experience								

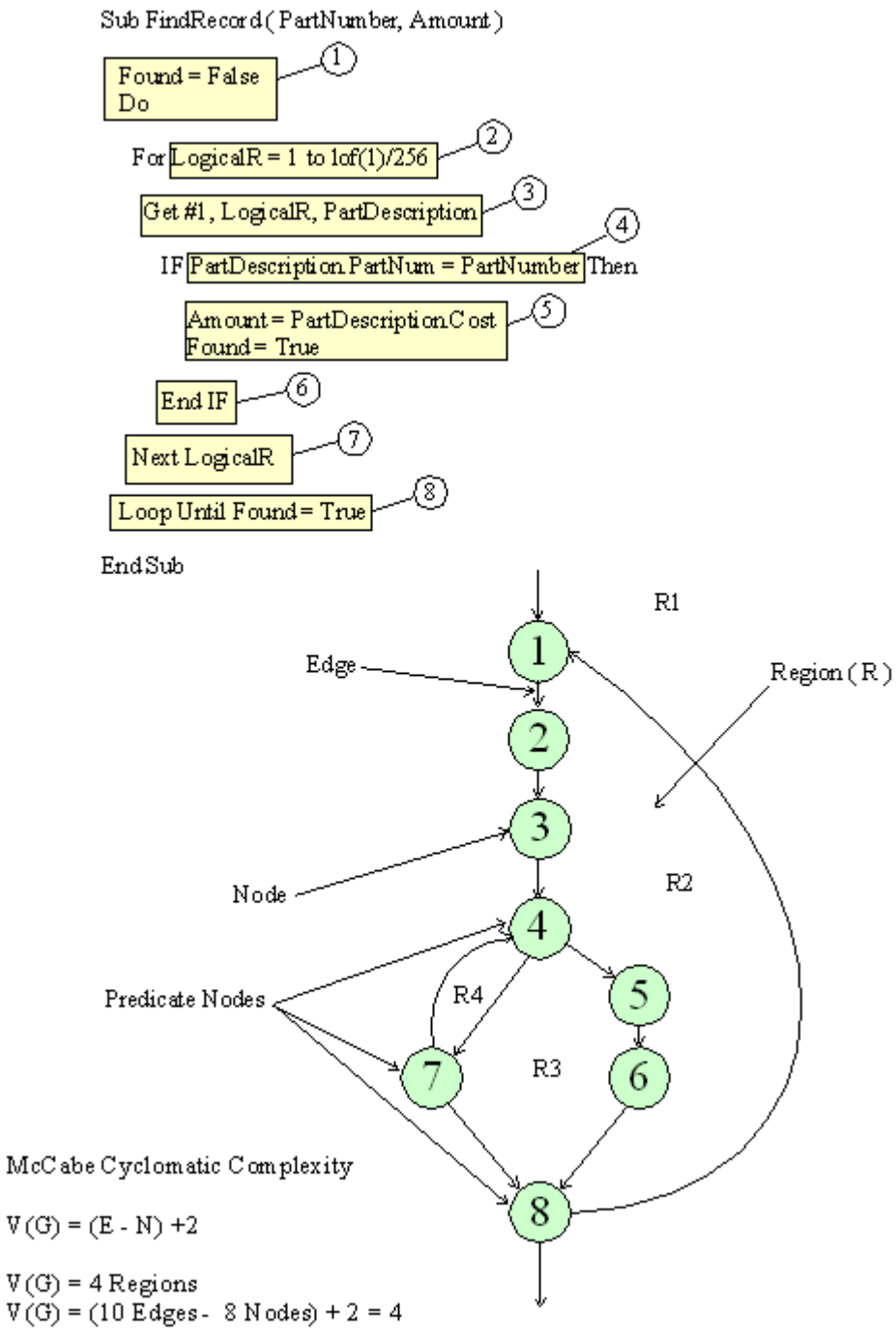
Figure 36 CARA analysis Summery

	CARA Score	Activity
Functions List		
1 User Interface	4.666667	Minor Review
1.1 Screen layout and appearance consistent with exiting business system	3.266667	Minor Review
1.2 Use multiple windows on screens when required	1.6	No Review
2 Data Security	11.2	Comprehensive Review
2.1 Data base location.	8	Focused Review
2.2 Data file password protection	5.6	Limited Review
3 Data Access	7.333333	Focused Review
3.1 File structure and type	10.4	Comprehensive Review
4 Data Attributes	5.333333	Limited Review
5 Program Scaleability	3.266667	Minor Review
6 Network Compatibility	6.666667	Focused Review
7 Error Handling	4.8	Minor Review

The final technique that I will mention is the MaCabe Cyclomatic Complexity[21]. The MaCabe Cyclomatic Complexity is another technique that I learned in SENG 330 that I have not started to use extensively. It again represents a quantum leap over my old processes. Figure 36 represents the Cyclomatic Complexity of the FindRecord sub-routine of the MMS project. It is a method for determining the complexity of code. It is most useful for the study of loops and nested loops to determine if the code is becoming overly complex and may need to revision.

[21] Steven R. Rakitin, **Software Verification and Validation** (ARTECH HOUSE, INC., 1997) 102

Figure 37 MaCabe Cyclomatic Complexity model



Conclusion

My involvement in the MSSE program over the last three years has made a profound difference in me professionally and personally. On a professional level my MSSE experience brought about dramatic changes in the way my company performs everyday business. I believe that my work at implementing Software Engineering practices, at this point in time, may have very well prevented enormous losses in revenue for the company that may have put the company at risk. Even though there must be investment made to introduce the changes I am making, the fact that the methodologies can span across the different engineering disciplines used within the organization brings a rapid return on their investment.

The work I am doing implementing data warehouse technologies is adding enormous value to the company resources. The new ability to get almost real time cost analysis on production items has brought about a new way of production planning and engineering. We now have hard factual data about production methods that work and those that don't. We don't re-invent the wheel each time we produce parts. The analysis group now can use the new data to make tooling and setup changes that can have an extreme effect on cost. This type of information has not been available to the corporate owners and managers in this magnitude before.

The processes I have introduced, and those yet to come, have changed the company culture. One of my goals is for my company to move up to ISO 9000 quality standards for their manufacturing. The introduction of CMM will lead to a corporate wide discipline required by these quality standards. The introduction of process also

empowers others in the organization to improve and buy-in to the changes being made. This will have a very positive effect on all future projects.

On a personal level the MSSE has given me the credibility on the software side of the organization that I have on the mechanical design side. Although I have held a management position for many years, I have not always been taken seriously by my supervisors. Without credentials it is sometimes difficult to gain credibility. You get into a position in a small firm and you tend to remain there. It takes confidence to bring about changes that are not always within the scope of your position. The MSSE program has given me the confidence and the professional credentials required to implement corporate wide process changes.

Being involved in the MSSE program where my classmates were all professional people gave me insight into how these processes work in different types of organizations. It was interesting to hear many of the same types of problems that I have experienced and how they went about resolving those problems. Work related discussions were always welcomed and I believe it added to classes. The Instructors respected everyone as professional rather than just students and I believe that greatly enhanced the learning experience.

List of Works Cited

Bryan Pfaffenberger, **Dictionary of Computer Terms Sixth Edition** (Simon & Schuster, Inc. 1997)

Cay S. Hortsmann, **Practical Object-Oriented Development in C++ and Java** (John Wiley & Sons, Inc. 1997)

Frederick P. Brooks, Jr., **The Mythical Man-Month** (Addison Wesley Longman, Inc. 1995)

Hydraulic Institute, Hydraulic Institute Standards (Library of Congress Card No. A82-84047)

Karl E. Wiegers, **Software Requirements** (Microsoft Press, 1999)

Microsoft Press, **Visual Basic 6.0** (Microsoft Corporation 1998)

Neal Whitten, **Managing Software Development Projects** Second Edition (John Wiley & Sons, Inc. 1995)

PMI Standards committee, **A Guide to Project Management Body of Knowledge (**William R. Duncan, Director of Standards 1996)

Robert O. Lewis, **Independent Verification and Validation** (John Wiley & Sons, Inc., 1992) 265

Roger S. Pressman, **Software Engineering: a Practitioner's Approach** (McGraw-Hill Series in Computer Science)

SBA Office of Advocacy latest census data
http://www.sba.gov/advo/stats/sbfaq.html#q3

Statistics of small businesses
http://www.census.gov/epcd/www/smallbus.html#EstabSize

Steven R. Rakitin, **Software Verification and Validation** (ARTECH HOUSE, INC., 1997)

Vitae

James Clifford Fleming

SOFTWARE INTEGRATION EXPERIENCE

I am responsible for all software engineering and development at the company I work for. I program primarily in Visual Basic 6.0. I have also developed control software utilizing programmable logic control systems for industrial equipment. My latest project is the engineering and development of a manufacturing management system. This system will give production managers the ability to do analytical production studies to improve overall product production efficiency. The database implementation will be based on data warehouse technologies utilizing the current relational database and other legacy data structures. I have engineered, developed, and implemented inventory management systems, engineering software, and personal process software. I am proficient in the use of Auto Cad Mechanical, Microsoft Word, Excel, Corel Draw, and Microsoft Power Point.

I have been maintaining a 32 station Local Area Network and all the computers on the network for the company for the last five years. I am responsible for maintaining the efficiency and security of the network and installing all of the software that I develop and all of the off the shelve software that is purchased by the company. The network has a Dell Power Edge 2400 dual processor domain controller server utilizing the Microsoft Small Business Server operating system running on the NT platform and a Dell Power Edge 2500 backup domain controller server utilizing the Microsoft Windows 2000 Server operating system. The PCs on the network utilize Windows 98, Millennium, and Windows XP professional operating systems. The network is linked to the Internet through a router using Win Route firewall software. I completely setup, including installation of the Windows 2000 server operating system, the Dell Power Edge 2500 dual processor server. The network is based on the client server network model. I am responsible for all trouble shooting and training on the use of the network and its resources.

PRODUCT DEVELOPMENT

The company I work for manufactures industrial water pumps. My current position as Director of Design and Development also requires me to supervise the engineering department and develop conceptual designs for future industrial pump products. My latest design achievement is a line of Ultra Pure high pressure water pumps for the Reverse Osmosis Industry. My pump designs are in use around the world in water purification systems.

MANAGERIAL EXPERIENCE

Director of Design and Development: Current position

My current position of Director of Design and Development requires me to engineer and develop all software and industrial pump design projects. I manage all of the projects from conception to final proto-type testing. This requires me to supervise the engineering and production staffs during the development of these types of projects. I also become involved with customers when technical issues arise that require product expertise. I am continually teaching the engineers and technicians that I supervise when issues occur that require my personal involvement.

Plant Superintendent:

The position of Plant Superintendent required me to supervise all factory employees and production activities. I was responsible for all production scheduling and the efficient operation of the manufacturing facilities. This position also required extensive interaction with company customers and vendors. I became an effective liaison between the customer and the company.

INSTRUCTIONAL SEMINARS

My position as Director of Design and Development has required me to give industrial pump technology seminars to some of our largest customers engineering staffs. The scope of the seminars is generally a technical introduction to various types of pumping technologies. I have given these seminars to US Filter, Culigan International, Glegg Water and Pentair water to name a few.

EDUCATION

Currently pursuing a *Master of Science in Software Engineering Degree*, West Virginia University.

Received *Certificate of Software Engineering*, WVU December 2002

Received *Regents B.A. Degree Focused on Industrial Technology,* Fairmont State College December 2000

Enrolled in The American College of Computer & Information Sciences. June 1998
Course completed:
CS 100 Intro to computers and Information Processes
CS Computer Programming Using Pascal

References on Request