Application of Critical Chain to Staged Software Development

by
Ronald Pepin

Master in Business Administration, University of Hartford, 1990

B.S. Electrical Engineering, Western New England College, 1983

Submitted to the System Design and Management Program in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Engineering and Management

at the

Massachusetts Institute of Technology

January 1999
[February, 1999]

Signature of Author [illegible]
System Design and Management
January 15, 1998

Certified by: [illegible]
Steven D. Eppinger, Associate Professor
Sloan School of Management
Thesis Advisor

Accepted by: [illegible]
Thomas L. Magnanti, Co-Director
System Design and Management Program

Application of Critical Chain to Staged Software Development

by
Ronald Pepin

Submitted to the System Design and Management Program in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Engineering and Management

Abstract

One in three IT projects are canceled before they are completed. Of the projects that are completed, over 75% are late, over budget or are released with reduced functionality. Average cost overruns are 189%; average schedule overruns are 222% (The Standish Group). The software development process and the project management techniques are critical components in completing a development project on time and on budget. Critical Chain Project Management techniques and a Staged Development process were designed to address issues that contribute to the large number of schedule and cost overruns.

Critical Chain is based on Theory of Constraint principles developed by Eliyahu Goldratt. Critical Chain offers practical methods for planning, scheduling, tracking and mitigating schedule risk in a development program.

Staged Development, a form of an incremental product development lifecycle, is considered to be a software development best practice. Staged Development promises faster development schedules, increase progress visibility and higher quality.

In this study the author researches, applies and analyzes the Critical Chain and Staged Development methodologies. The combination of the two methodologies created a process that served to increase likelihood of project success.

Thesis supervisor:

Steven D. Eppinger, Associate Professor, Sloan School of Management

Acknowledgments

I would like to acknowledge the following organizations and people for their contributions to my learning effort:

- The Systems Design and Management (SDM) Program for the opportunity. The SDM program provided me with the knowledge and the tools to understand and manage the complexity of defining and leading technical organizations.
- My SDM Classmates for pushing me to set higher goals.
- The leadership at the Otis Engineering Center, specifically Pat Hale and Mick Maurer, for the courage to nourish change within our organization. Pat opened my eyes to the importance of "systems thinking" and Mick provided me with the freedom, the support and the guidance required to implement leading edge concepts.
- The NMS Development Team for their openness to unfamiliar concepts and their unending dedication to the development effort. I am fortunate to be surrounded by peers with the drive to succeed.
- My thesis advisor, Steve Eppinger. Steve provided me with direction and guidance in formulating the concepts presented in this work.
- My wife, Maureen, for her encouragement and understanding over the last two years. Maureen helped me weather the lows and encouraged me to recognize and celebrate the highs.
- My children, Maura, Colin, Brian and Bridget, for the daily cheer they bring to my life.

Table of Contents

List of Figures

1 Critical Chain Fundamentals

1.1 Introduction

Schedule, cost and performance are key metrics in all product development programs. Research conducted by *The Standish Group* compiled the following, not so encouraging statistics on development programs: (Standish Group)

- 30% of IT projects are canceled before they are completed.
- Of those projects that are completed 75% are late, over budget, and/or are implemented with reduced functionality.
- The average cost overrun is 189%
- Average schedule overrun is 222%

Traditional program management practices are not working. Traditional practices encourage bad habits such as individual task padding, multitasking, the Student Syndrome, frequent schedule updates and fire fighting. (Lynch, 1998). Critical Chain techniques were specifically design to address these issues.

The Critical Chain program management methodology is based on practical principles that were developed and applied in manufacturing environments. The underlying fundamental principle is called The Theory of Constrains (TOC). The Theory of Constraints is based on five basic steps (Goldratt, 1992). The five steps are:

1. **Identify** the constraint
2. Decide how to **EXPLOIT** the constraint
3. **SUBORDINATE** everything else to the above decision
4. **ELEVATE** (if necessary) the constraint
5. **GO BACK** to step 1.

Critical Chain applies the 5 basic steps of TOC to program management. Critical Chain replaces individual task padding with strategically placed buffers that protect the critical path. Critical Chain minimizes multitasking by the use of strong resource loading guidelines and reduces the Student Syndrome effect by using 50/50 task duration estimates. Critical Chain techniques also allow the program to accumulate early finishes and to avoid frequent critical path and schedule modifications.

1.2 Critical Chain: Key Concepts and Definitions

An understanding of the Critical Chain terminology is required to understand the Critical Chain method. The Critical Chain terminology that is hereafter presented was adapted from Robert Newbold's text on the subject (Newbold, 1998).

1.2.1 Critical Chain

The Critical Chain is the series of tasks that determine the overall duration of a project. The development pace of the project is determined by the completion rate of the tasks on the Critical Chain. The Critical Chain, which takes resource capacity into account, is typically regarded as the constraint or leverage point of the project (Newbold, 1998). The tasks on the Critical Chain are the focal point in a Critical Chain schedule.

The manufacturing process analogy to the Critical Chain is a bottleneck. Goldratt (Goldratt, 1992) defines a bottleneck as the process (task) that regulates the throughput of the complete manufacturing system. In program management, the tasks on the Critical Chain regulate the speed in which the project completes. The Critical Chain is the program constraint. A one-day gain/slip on the Critical Chain equates to a one-day gain/slip to the project delivery date. A project manager's focus must be on the Critical Chain. All non-Critical Chain tasks and resources are subordinated to the Critical Chain tasks.

1.2.2 Safety

Safety is the excess time that is inserted into a project schedule to protect the project from uncertainty.

Developers, in an effort to protect themselves, have intuitively learned to insert safety by providing pessimistic (conservative) estimates. The net effect of pessimistic estimates is amplified by the lognormal shape of the probability of completion vs. time to complete curve (see figure). Goldratt observed (Goldratt, 1997) that a typical developer will provide an estimate that carries an 80% probability that the task will be completed on time. On a lognormal time scale, the 80% completion time estimate is double the 50% completion time estimate. Therefore, the typical developer provides estimates that are double the time in which there is a 50/50 probability of completing the task. The result: traditional schedules contain significant levels of safety distributed in all of the tasks.

Probability of Completion Curve

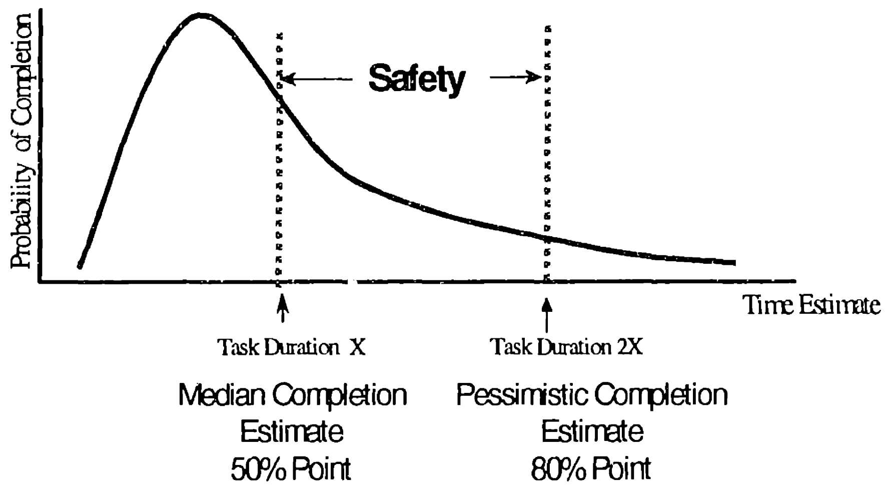


Figure 1: Probability of Completion Curve.
Adapted from Newbold (Newbold, 1998)

First line managers compound the safety adding effect of the individual developers. First line managers add additional safety to prepare for the ever-present threat of upper management's unilateral schedule cuts. In summary, developers add safety to every task and first line managers add additional safety to the end of the schedule.

Along with adding safety, traditional project management techniques also encourage behaviors that waste safety. Traditional project management techniques track and manage programs by the use of milestone dates. Managing projects by milestones wastes safety by encouraging the Student Syndrome and by neglecting potential schedule gains due to early finishes.

The Student Syndrome (see figure) , the behavior in which people do not start a task until they feel they have a 50% probability of completing a task, is a leading factor in safety consumption. Because of the Student Syndrome, even tasks with large amounts of safety built in still may only have 50% chance of meeting the milestone. The Student Syndrome consumes a large portion of the safety that was inserted by the developers pessimistic estimate.

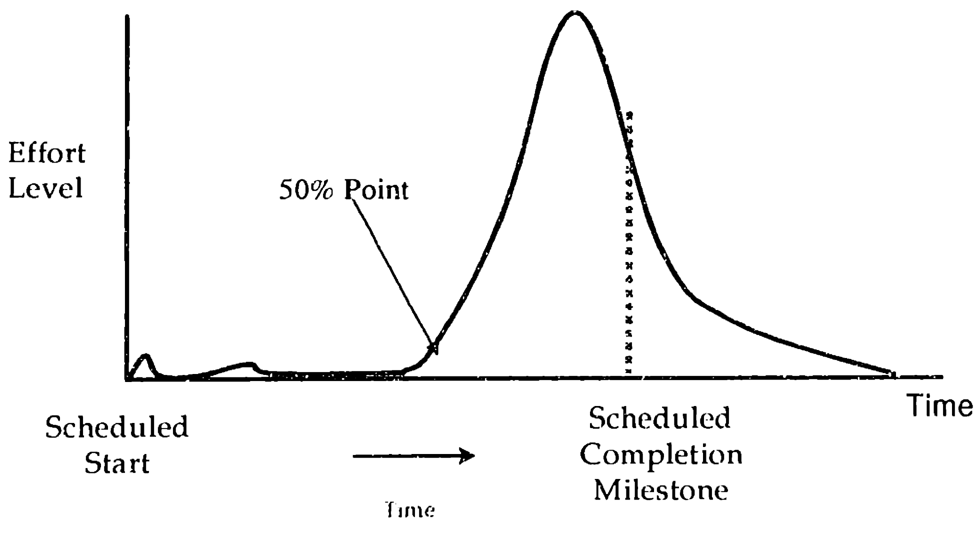


Figure 2: The Student Syndrome. Adapted from Goldratt (Goldratt, 1997).

The second negative effect of managing to milestones is that delays accumulate but gains do not. Simple statistical laws of average indicate that in any reasonably large project there are tasks that finish late and tasks that finish early. For example, Case 1, in the following diagram, demonstrates the net effect of a late finish. In Case 1, the 1-day late finish of task A is directly reflected in a 1-day program schedule delay. All task delays are accumulated in a similar manner.

In Case 2, task A finishes 1-day early. Unlike the delay, an early finish is not automatically reflected in project schedule. The reasons for this phenomenon are based on cultural norms, the overall milestone mentality and human nature. On the cultural front, developers never want to be caught "sandbagging" an estimate. General perception is that early finishes are the result of over estimation. Therefore, from a developer perspective, an early finish will bolster management's suspicion of estimate inflation and will serve to increase the probability and the severity of the heretofore mentioned global cut.

A second milestone factor that prohibits the accumulation of early finishes is the scheduling of start dates. Milestone start dates are hard dates in which a task is to start. Developers coordinate their schedules around these start dates. Therefore, even if task A completes early, there is a high probability that task B is not ready to start until the milestone date.

Early finishes also fall victim to developer's perfectionist characteristics. Developers take ownership and pride in their work. This pride in their work encourages developers, upon finishing early, not to announce the early completion but to continue enhancing the task deliverable until the milestone completion date. This process, which is called Gold-plating, is a major reason that an early task completion is not reflected in the project completion date.

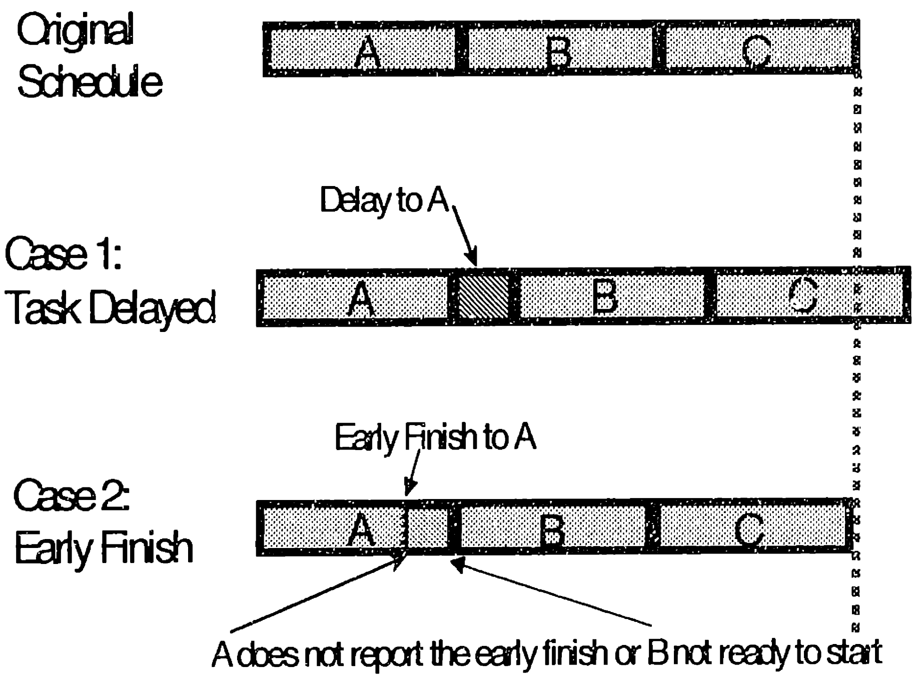


Figure 3: Early Finishes – Adapted from Goldratt (Goldratt, 1997)

Multitasking is also responsible for safety consumption. By definition, multitasking is the practice of giving people more than one task to do at the same time, without having a clear and consistent priority amount among the tasks (Newbold, 1998). Tasks that are multitasked take longer to complete (see figure). A task that has been started, then put on hold is work in process. All work in progress demands some level of attention and serves to distract the developer from the task that should be focused on. This distraction causes the task to take longer to develop. Multitasking also requires the developer to frequently switch between tasks. Switching between tasks carries a time penalty that is similar to the set-up time penalty in a manufacturing environment. Simply stated, if the task is on the Critical Chain is multitasked, the task and the project take longer to complete.

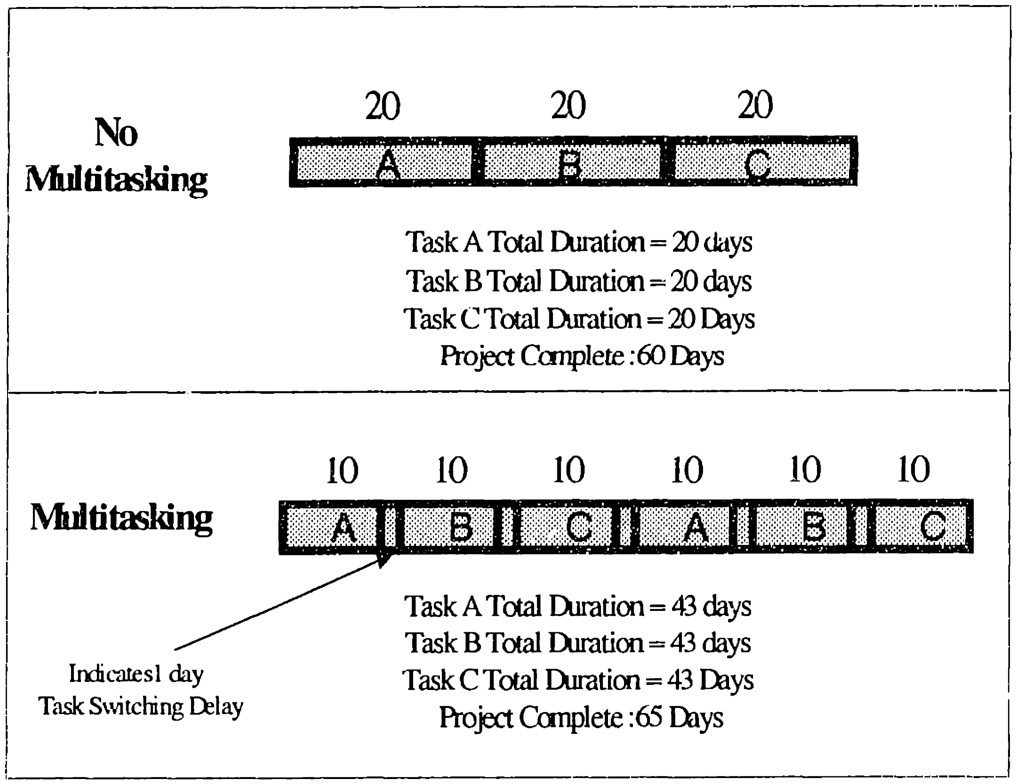

No Multitasking
20
20
20
A
B
C
Task A Total Duration = 20 days
Task B Total Duration = 20 days
Task C Total Duration = 20 Days
Project Complete :60 Days
Multitasking
10
10
10
10
10
10
A
B
C
A
B
C
Task A Total Duration = 43 days
Task B Total Duration = 43 days
Task C Total Duration = 43 Days
Project Complete :65 Days
Indicates1 day
Task Switching Delay

Figure 4: Multitasking – Adapted from Goldratt (Goldratt, 1997)

1.2.3 Buffers

A buffer is time put into a Critical Chain schedule to systematically protect against unanticipated delays and in order to allow for early task starts. Buffers are not slack: they are essential parts of the schedule (Newbold, 1998). At least two types of task buffers exist in the Critical Chain methodology: project buffers and feeding buffers.

By definition, a project buffer is placed at the end of a project schedule. The purpose of the project buffer is to protect the project completion date from delays along the Critical Chain. The project buffer, which is the redistribution of the safety in individual tasks, accumulates all of the safety at the end of the project. The project buffer is nominally set to be 50% of the length of the Critical Chain.

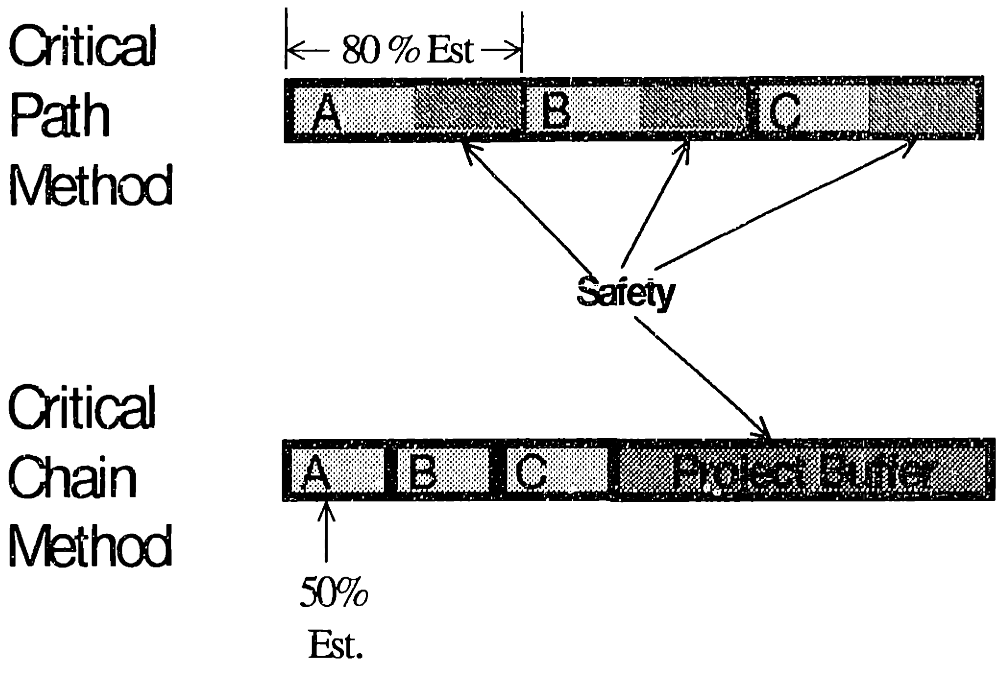


Figure 5: Project Buffer – Adapted from Goldratt (Goldratt, 1997)

Feeding buffers are inserted to ensure non-Critical Chain tasks do not delay the start of a Critical Chain task. Feeding buffers are inserted at the point in which a non-Critical task feeds a critical task. The length of the feeding buffer is generally half the length of the complete chain of non-critical chain of tasks.

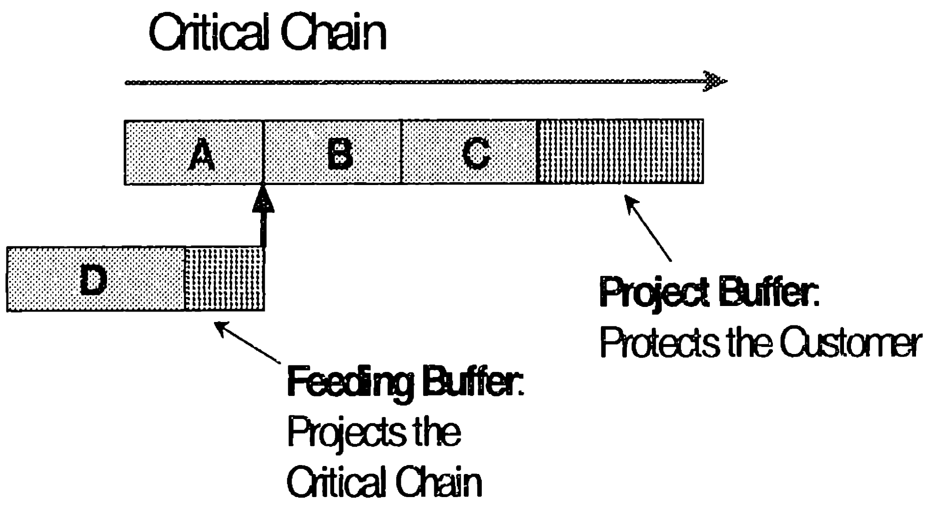


Figure 6: Feeding Buffers – Adapted from Goldratt (Goldratt, 1997)

Along with protecting the schedule completion date, the task buffers are used to track the project. Monitoring the amount of the buffer that has been consumed allows the program manager to identify and track the tasks that are slipping the schedule. Each buffer can be separated into three zones.

The first zone, the "OK Zone", indicates that less than a third of the buffer has been consumed. The second zone, the "Warning Zone", indicates that one third to two thirds of buffer has been consumed and the third zone, over two thirds expended, is the "Act Zone".

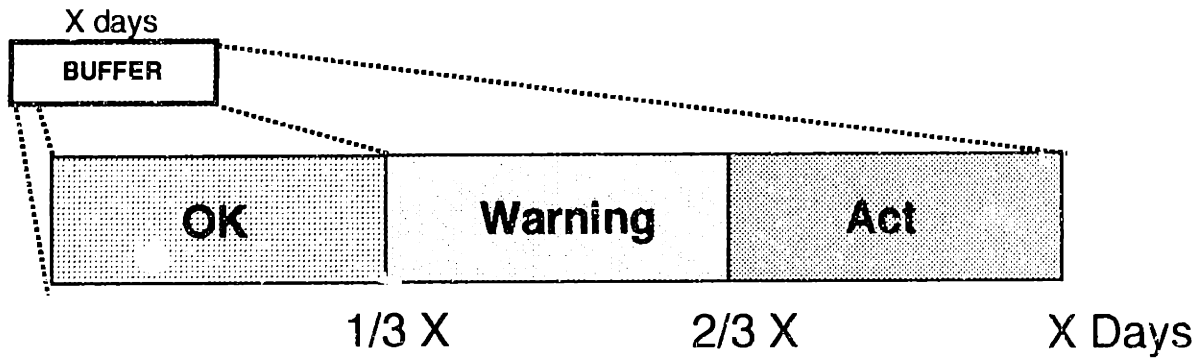


Figure 7: Buffer Warning Ranges – Adapted from Goldratt (Goldratt, 1997)

Path delays that result in delays that are less than 1/3 of the buffer are in the OK Zone should not cause program management concern. Path delays between 1/3 and 2/3 of the buffer are in the Warning Zone and should be on a daily watch list. Path delays above 2/3 of the buffer need immediate program management attention to avoid a schedule delay.

In the Critical Chain methodology, the Buffer Report servers as a concise and complete view of the status of the program. A sample buffer report (ProChain, 1998) is as follows:

Buffer Name	Buffer End	Buffer Length	% of Buffer Used	Buffer Left	Chain Left	Check Task
FB: Implement Test	8/24/98	10	100%	0	24	Test Plan Development
PB: Implement Sub-System Test	11/13/98	19	68%	6.08	46	Design
FB: Unit Integration	10/12/98	6	0%	6	34	

Figure 8: Buffer Report

Buffer Name: The name ProChain™ assigns to the buffer. PB = Project Buffer, FB = Feeding Buffer. Note: The buffer name is determined by the last task in the task path.
Buffer End: The original end date for the buffer.
Buffer Length: The original length of the buffer.
% Of Buffer Used: The percent of the buffer that has been consumed. % Buffer Used is the key metric.
Buffer Left: The number of days left in the buffer
Chain Left: The number of days left to complete the chain of tasks that preceded the buffer.
Check Task: Identifies the task that is most likely causing the buffer consumption.

1.3 ProChain^{TM} Project scheduling.

The ProChain^{TM} Users Guide (ProChain, 1998) was used as the source of information for this section.

1.3.1 Creating the Schedule

ProChain^{TM} Project Scheduling is an add-on feature to Microsoft Project. ProChain^{TM} is designed to help the program manager follow the principles of the Critical Chain. The ProChain^{TM} software was designed and developed by *Creative Technology Labs* (www.prochain.com). ProChain^{TM} imposes the following six steps to developing a project schedule (ProChain, 1998).

1. Create the initial project network
2. Level Load the schedule
3. Identify the Critical Chain
4. Create the Buffers
5. Insert the Buffers into the network.
6. Review Resultant Schedule – Go to step 1 if necessary

All of the above steps are implemented in ProChain^{TM} except Step 1 and step 6.

A well thought out project network is fundamental to developing a solid program schedule. The key tasks in developing a project network are:

- Establishing the Program Deliverable
- An Analysis of the Required Tasks
- Establishing the Inter-Task Dependencies
- Allocation of Resources
- Estimation of the Task Durations

The initial step in creating the network is to ensure that there is a clear and concise definition of the program's final deliverable. The program deliverable is the reason for doing the program. Once the end point is established, the network building process is laid out from right to left, with the rightmost task being the final deliverable. Task identification phase starts by asking, "What tasks are absolutely necessary to meet the deliverable?" The first pass through the network development should be kept at a high level. Detailed tasks identification occurs in the second or third pass of the network. An example of the first pass of building the project network is shown below.

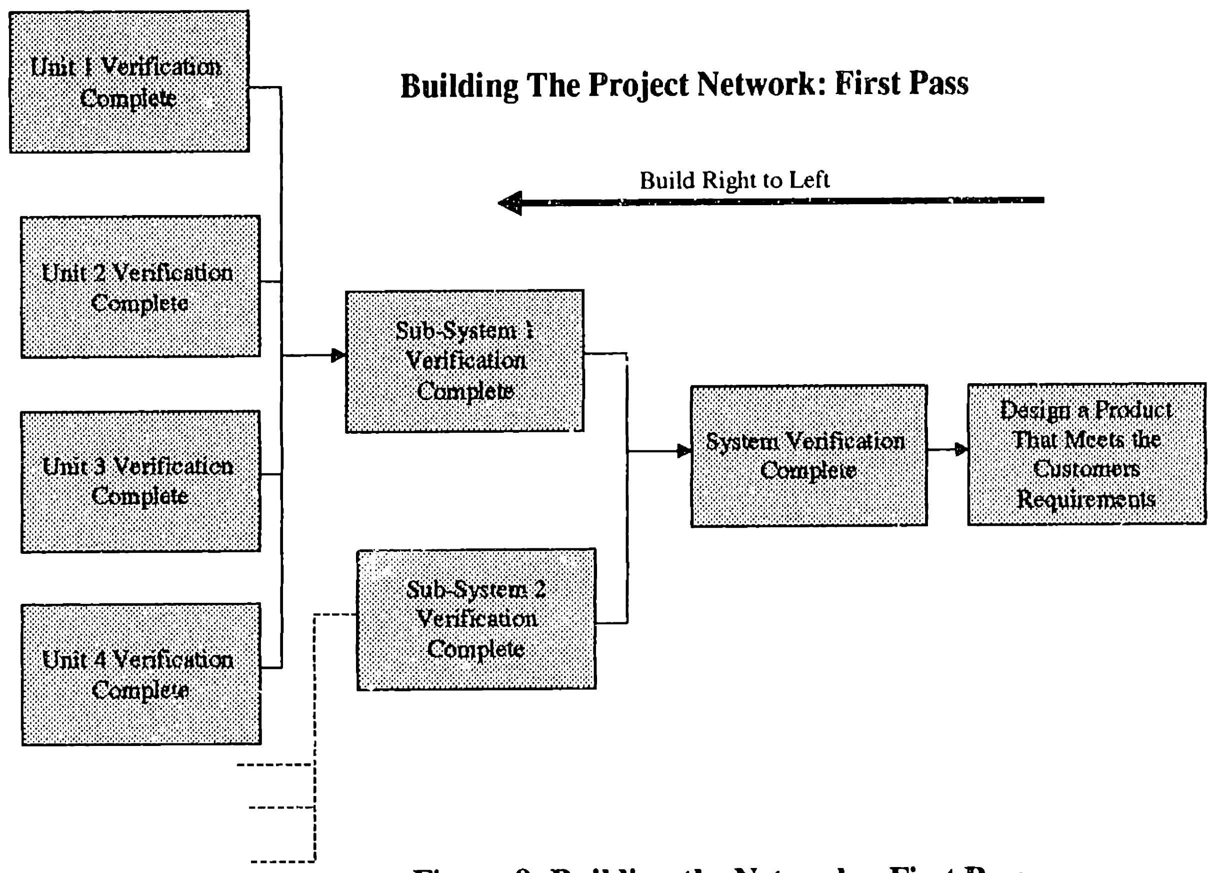

Building The Project Network: First Pass
Build Right to Left
Unit 1 Verification Complete
Unit 2 Verification Complete
Unit 3 Verification Complete
Unit 4 Verification Complete
Sub-System 1 Verification Complete
Sub-System 2 Verification Complete
System Verification Complete
Design a Product That Meets the Customers Requirements

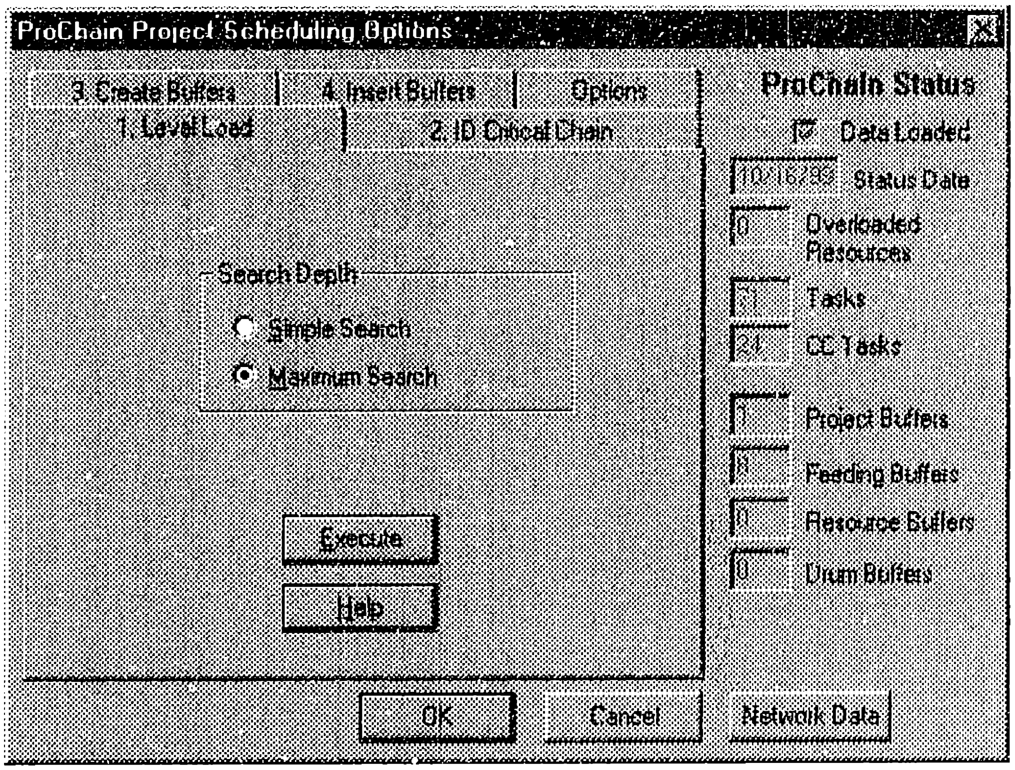
Figure 9: Building the Network – First Pass

Building the network establishes a list of the required tasks and the task dependencies. All tasks and the dependencies should be carefully reviewed to ensure that they are absolutely necessary. The next step is to assign resources to the tasks. For proper leveling loading can only be accomplished if each task has resources allocated to it. In order for ProChain^{TM} to level load the network each task must have resources assigned to it.

Task estimation is the last step before the network is entered into Microsoft Project. Ideally, the resource(s) assigned to the task provide the task duration estimates. The estimator must understand that they are being asked to provide estimates based on Critical Chain principles. Task estimates must adhere to the following guideline in mind:

- Estimate the duration as if it is the only task to work on (no multitasking)
- Assume all proceeding tasks will complete on time
- Estimate the duration such that there is a 50% probability of successfully completing the task in the estimated time.
- Late Finishes will not be punished.

ProChain^{TM} requires that the tasks, task dependencies, resource allocations and task estimates be entered into a Microsoft Project schedule.



Search Depth: Simple Search [✓ Maximum Search]

[✓] Data Loaded

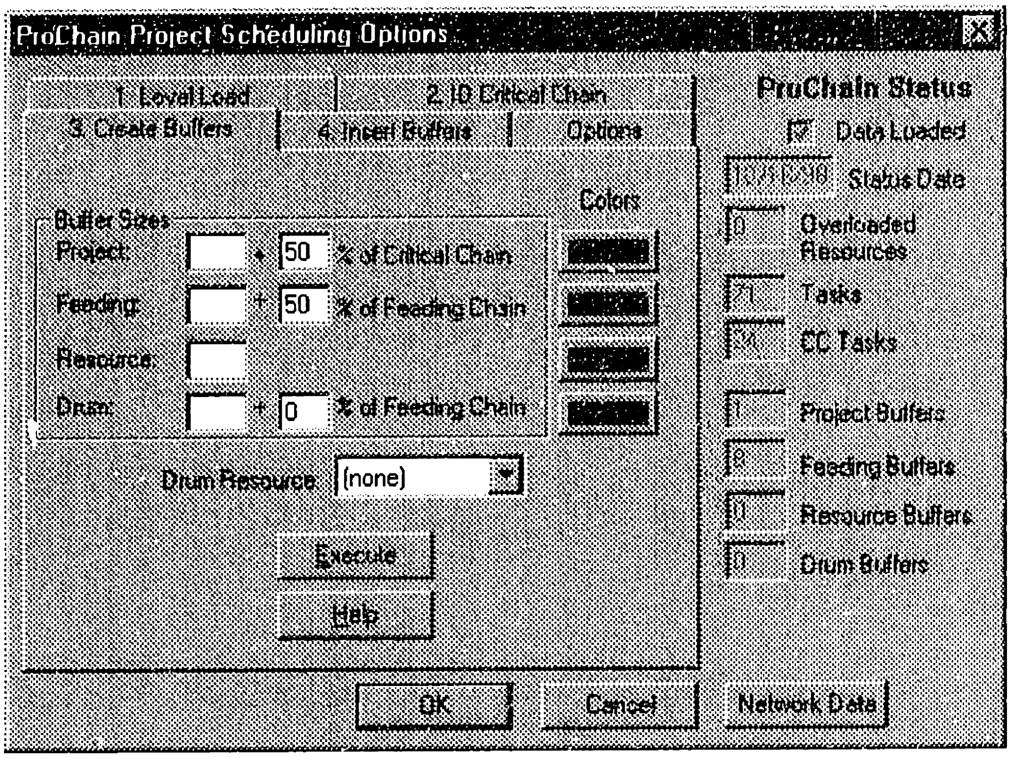
10/16/98	Status Date
0	Overloaded Resources
[illegible]	Tasks
21	CC Tasks
1	Project Buffers
8	Feeding Buffers
0	Resource Buffers
0	Drum Buffers

Figure 10: ProChain^{TM} **Scheduling Options (ProChain, 1998)**

Level loading is the first feature in the ProChain^{TM} Scheduling process. The ProChain^{TM} level loading feature uses the information entered in to Microsoft Project to load the project such a way that there is no planned multitasking and/or task dependency violations.

The second ProChain^{TM} stage is the identification of the Critical Chain. ProChain^{TM} analyzes the all of the task flows to determine which thread is the longest. The longest thread is identified as the Critical Chain.

Once the project is level loaded and the Critical Chain is identified, the project is ready for the third ProChain^{TM} stage of creating the buffers. ProChain^{TM} allow the operator to determine the size (% of chain) of the buffers. ProChain^{TM} defaults to the Goldratt recommended 50% of the linked chain. ProChain^{TM} , upon activation, creates the project buffer and all of the required feeding buffers. The last state in the ProChain^{TM} schedule creation process is to insert the buffers. Initiating the "Insert Buffer" feature instructs ProChain^{TM} to move tasks forward to allow for feeding buffers to be inserted. The project buffer is attached to the end of the schedule.



Buffer Sizes

Project:	+ 50	% of Critical Chain	■
Feeding:	+ 50	% of Feeding Chain	■
Resource:			■
Drum:	+ 0	% of Feeding Chain	■

Drum Resource: (none)

ProChain Status

☑ Data Loaded

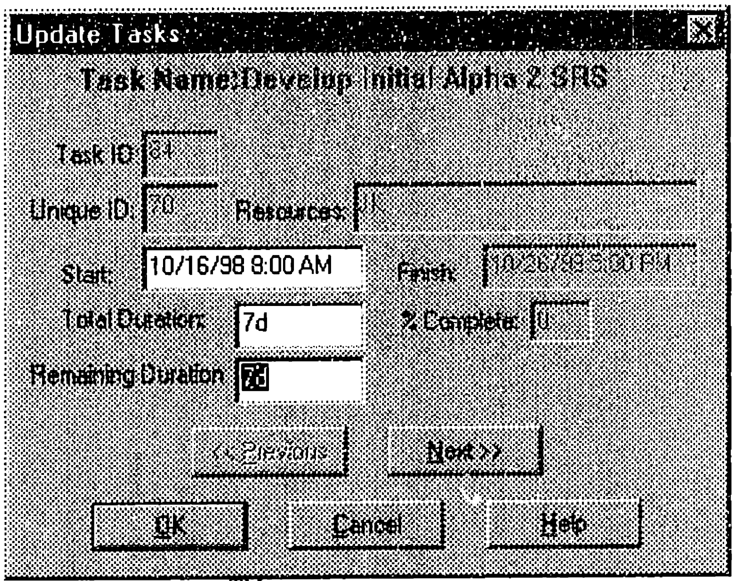
[illegible]	Status Date
0	Overloaded Resources
71	Tasks
34	CC Tasks
1	Project Buffers
8	Feeding Buffers
0	Resource Buffers
0	Drum Buffers

Figure 11: ProChain^{TM} Scheduling Options – Buffer Size (ProChain, 1998)

The last step in the process is to review the resultant schedule for opportunities, feasibility and correctness. ProChain^{TM} allows the operator to return to the appropriate step to modify the schedule.

1.3.2 Updating ProChain^{TM} Schedules.

ProChain^{TM} facilitates the process to update the project schedule. Initiation of the "Update Task" feature will cause ProChain^{TM} to present, one by one, all of the active tasks to the operator. The operator simply has to update the expected remaining duration for all of the active tasks. When all of the active task durations are updated, ProChain^{TM} automatically updates the schedule to reflect buffer consumption and updated start and end dates.


Update Tasks

Task Name:Develop Initial Alpha 2 SRS

Task ID: 24

Unique ID: 70

Resources: JK

Start: 10/16/98 8:00 AM

Finish: 10/26/98 5:00 PM

Total Duration: 7d

% Complete: 0

Remaining Duration: 7d

Figure 12: ProChain™ Task Update Screen (ProChain, 1998)

1.3.3 Tracking the Schedule with ProChain™

Tracking the schedule is accomplished by monitoring the ProChain™ Non-Resource Buffer Report. The Non-Resource buffer report provides information that is critical to tracking a Critical Chain program.

2 Staged Software Development

If you ask a software developer the status of something he's working on, his answer might be correct, but if it is, that's just a coincidence. (McCarthy, 1995)

In today's turbulent business environment, more and more companies need a development process that embraces change – not one that resists it. (Iansiti and MacCormack, 1997)

Staged delivery is not a panacea. But, on balance, the additional overhead it demands is a small price to pay for the significantly improved status visibility, quality visibility, flexibility, estimation accuracy, and risk reduction it provides. (McConnell, 1998).

2.1 Introduction

2.1.1 Summary of Staged Development

A Staged Software Development Process is a specific instance of a larger class of Incremental Product Development processes. The Staged Software Development Process defines the steps and the development order of a project. Staged development evolves the product in stages. Each stage represents a complete deliverable segment (feature) of the final product. In staged development the product can, theoretically, be released at the end of any stage.

Staged development has become popular with the emergence of modern software technologies. Technologies, such as Object Orientated Software development and Component based development, provide natural interfaces to cleave clusters of features into development stages.

An illustration of a Staged Software Development Process is shown below.

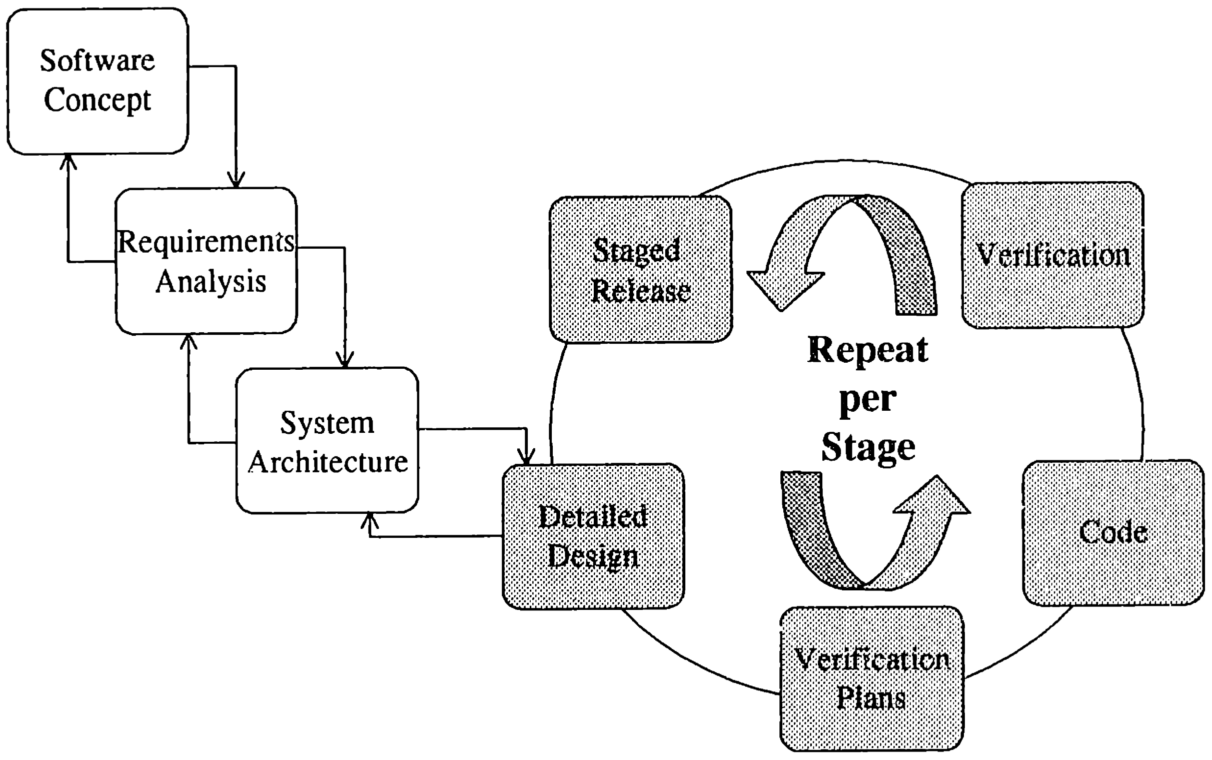


Figure 13: Staged Development Process – Adapted from McConnell (McConnell, 1996, 1998)

2.1.2 Phases of a Staged Development Cycle

Staged Software Development can be divided into three development phases. The first phase contains the concept generation and requirement analysis activities. Clear and well-understood concepts and requirements are imperative to determining the scope of the program. The challenge of the requirement analysis phase is to define the right level of detail. Up front requirements have to be detailed enough to allow proper scope and understanding of the complete program, but open enough to allow for creativity and innovation during the implementation process. The initial requirements must also allow considerable feature flexibility. This is essential in development environments in which new competitors and technologies appear overnight. (Iansiti and MacCormack, 1997). The initial requirement analysis process must be designed to allow the requirements to evolve with the staged progress while providing the system architects and programs planning people the information required to scope and plan the program.

The second phase in the process is the development of the system architecture. The purpose of the architecture is to provide a consistent framework into which the staged features can be integrated. Architectural consistency not only helps the product take on a singular form, but also helps to ensure system simplicity, component reuse and minimized development effort. Shortcomings in the initial system architecture will likely cause upstream rework when downstream features do not fit into the architecture.

The third phase in the process is the staging cycle. The staging cycle contains all of the activities that are required to synthesize and implement the features. The staging cycle is repeated many times in the development process. The deliverable at the end of each stage is verified software that can be reviewed (early stages) or released (later stages) to the customer. The staged deliveries provide the most reliable form of project status available. (McConnell, 1998)

2.1.3 Determining Stage Order

Implementation of a Staged Software Development Process requires careful stage planning. Some of the parameters used to determine the features that are clustered in each stage are; level of risk, functional dependencies, skill set requirements and customer feature rollout preferences. The following guidelines can be used to set priorities on the staged features:

1. High-risk features should be developed first.
2. Features that are important to the customer should be done early
3. Dependent features should be clustered into a stage.
4. Stages should be no longer than three months.

2.2 Staged Development Vs Waterfall Development Model

The pure waterfall model is the most widely used product development model (McConnell, 1996). The pure waterfall model is a sequential development model that requires the preceding phase to be completed before the succeeding phase can be initiated. An illustration of the waterfall model is as follows:

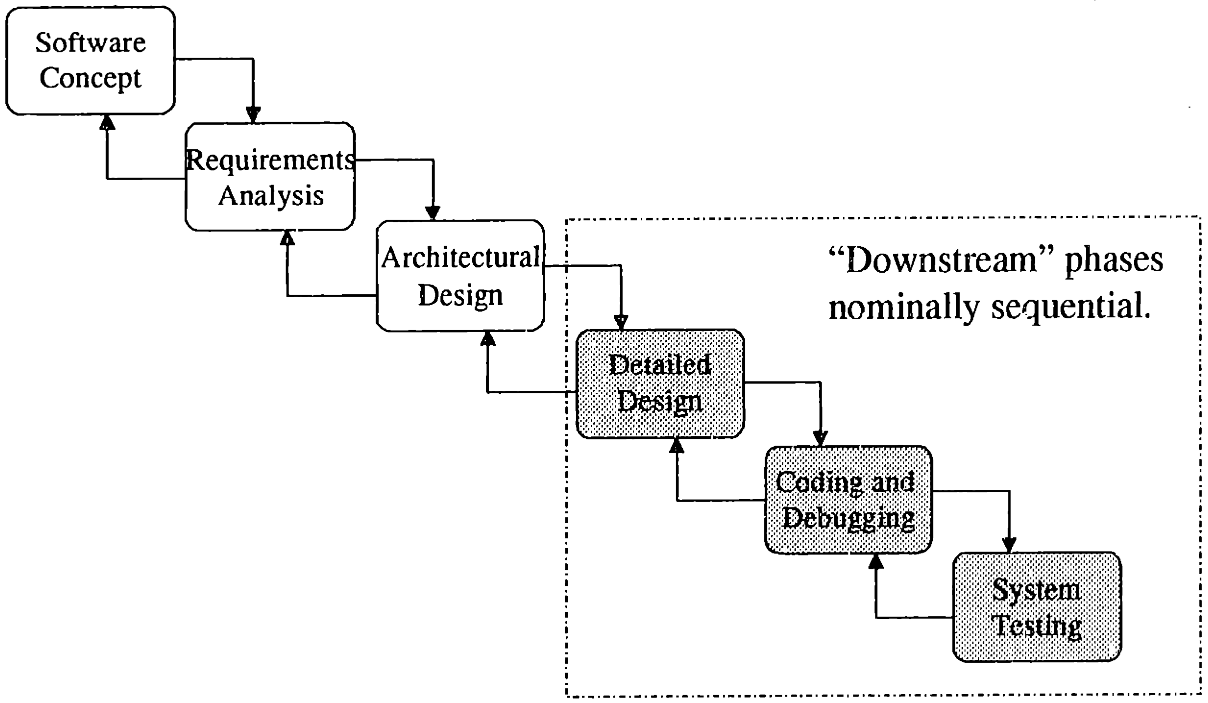


Figure 14: Waterfall Development Process – Adapted from McConnell (McConnell, 1996)

A major downfall of the waterfall model is the non-iterative nature of the process. In the waterfall model each phase is declared complete at the end of the phase. Once a phase has been declared complete it is difficult to return to that phase. The adverse effect of the non-iterative approach is amplified by the large time gap between defect creation and defect detection. In a serial process a defect inserted early in the design activity may remain undetected until the design is completed and implementation is started. In a staged development cycle, functional tasks are evolved in complete lifecycle stages. Iterating on the complete development cycle in small stages closes the time gap between defect insertion in upstream analysis task and defect detection in a downstream implementation task. The result is a shortened defect discovery time, which limits the overall level of rework. Undiscovered rework is the single most important source of project and schedule risk. (Cooper and Mullen, 1993). Therefore, staged development helps to expose the level of undiscovered rework earlier in the development cycle.

The waterfall model also assumes that the assumptions made at the start of the program hold true for the complete program. In the software development world, the pace of technology, frequently upgraded development tools and system capabilities, and the requirements flexibility require assumptions to be challenged throughout the program. The staged development process contains frequent break points, between stages, in which the team can take advantage of to challenge assumptions and take action on the lessons learned. Frequent cycling of the complete development process provides the development team with the opportunity to improve their productivity and development speed throughout the program.

Visibility of progress also differentiates the waterfall and staged development models. The only true means of tracking a software project is the measurement of working code (McConnell, 1998). In the waterfall model "working code" is not delivered until late in the development process. The delay in the production of "working code" makes it difficult to get an accurate read on the project status. In staged development, working code is developed and added to in every stage.

Staged development also provides a pipeline for increased customer involvement. The working code that is generated provides an efficient conduit for customer feedback. The primary conduit with the customer is in the staged software deliveries. Staged deliveries allow the customer the ability to see, touch and feel the system before the design is complete. In the waterfall model, the primary conduit with the customer is written material. The customer is forced to develop an image of the product from words and pictures until such time in which implementation is complete. At that point, when the design is complete, the customer can provide design feedback. The lack of customer feedback violates a well-known software development best practice; Customer involvement throughout the project is a critical to the project survival (McConnell, 1998).

2.2.1 Benefits to the Waterfall Model

Even with all of it's shortcomings, the waterfall model has stood the test of time and has been proven to work well for projects that (McConnell, 1996);

1. Are well understood but complex.
2. Have quality requirements that dominate cost and schedule.
3. Have a technically weak development staff.

Well understood but complex projects benefit from the orderly and structured process flow of the waterfall model. The end to end thoroughness of each stage minimizes the complexity effect, while the deep understanding of the project minimizes the early learning benefits of a staged development process.

The quality improvements are due to the waterfall model's inability to accept additional requirement changes mid-stream. The openness of the staged process exposes the risk of including high-risk requirement changes late in the project.

The structured waterfall process also benefits weak development teams. The process allows the team the time to strengthen their skills in each functional area before they have to move to the next phase. Staged development requires a strong team that is open to learning and improving at a rapid pace.

2.2.2 *Benefits of Staged Development:*

The benefits of staged development have been documented by Steve McConnell (McConnell, 1998) and in a white paper produced by Microsoft Corporation (Microsoft, 1998). The following list is a combination of the benefits listed by the heretofore-mentioned sources:

1. **Critical Functionality is Available Earlier:** Important features and functions are delivered first.
2. **Risk Is Reduced Early:** High-risk items are designed and tested first. Having a potentially deliverable product at the end of each stage reduces schedule risk.
3. **Problems Become Evident Early:** Completing the development lifecycle early draws out process issues. Early customer feedback, efficient program planning and risk reduction.
4. **Status-Reporting Overhead is Reduced:** Progress is measured in working code. Therefore, developers do not have to spend a lot of time on status reports.
5. **Makes More Options Available:** The process provides the option to release the software at any stage.
6. **Reduces the Possibility of Estimation Errors**: productivity of the team in the early stages can be used to estimate the productivity in the later stages.
7. **Flexibility and Efficiency**: detailed design decisions can be delayed until implementation time.
8. **Promotes Frequent and Honest Communication between the Team and the Customer:** Working software is the most honest means of communicating status, features and interactions.
9. **Sets Clear and Motivational Goals for All Team Members**: each stage provides clear attainable short-term goal. The development team can see and understand the goal.
10. **Rapid Learning**: Staged development allows the developers to learn the complete development cycle early in the process.

3 The Project

3.1 Introduction

The project that is being used as the subject of this research is the Network Management Sub-System (NMS 2000) for the Remote Elevator Monitoring (REM® 2000) system at Otis Elevator Company. The REM® 2000 system is a major sub-system of a Service Management System (SMS). Two other major sub-systems of SMS are the Service Dispatch System and the Maintenance Planning System.

3.2 Project Explanation

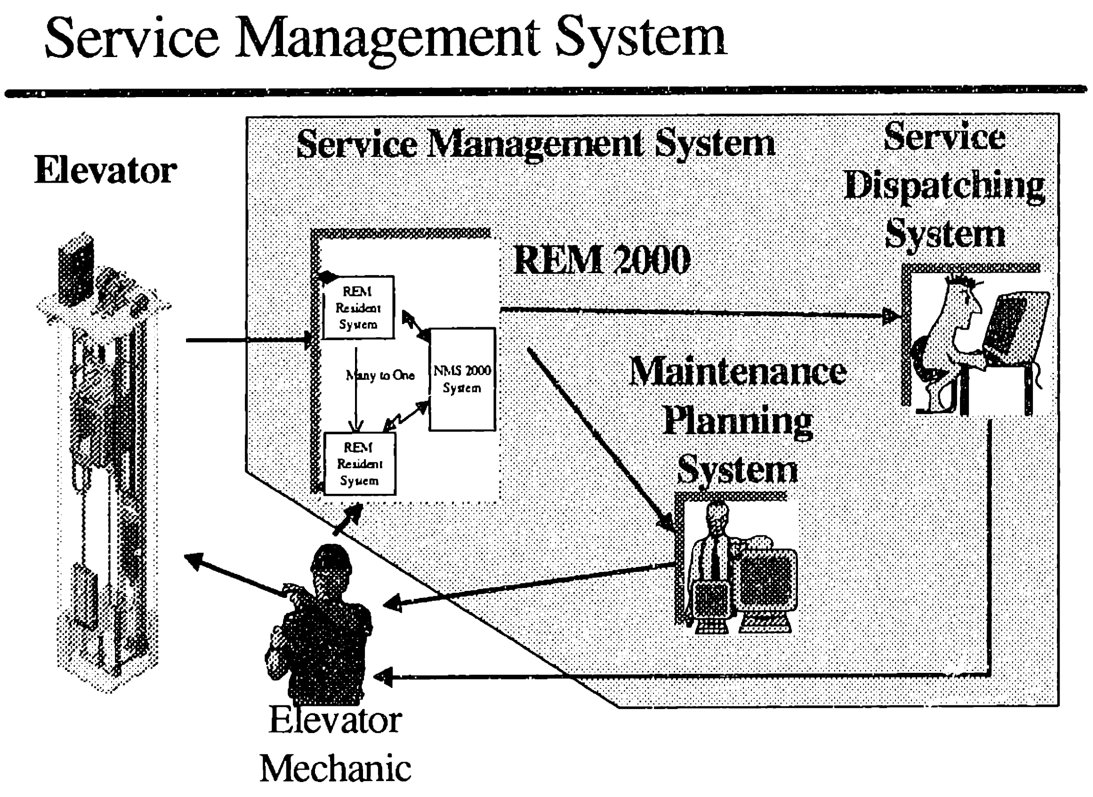

Service Management System
Service Management System
Elevator
Service Dispatching System
REM 2000
REM Resident System
Many to One
NMS 2000 System
REM Resident System
Maintenance Planning System
Elevator Mechanic

Figure 15: Service Management System

A primary purpose of the Service Management System is to provide information to elevator mechanics. The primary purpose of the REM® 2000 System is to provide information to the Service Dispatching System and the Maintenance Planning System. The REM® 2000 System provides information that is associated with elevator shutdowns and elevator performance.

Elevator shutdowns are system failures that result in an elevator out of service condition. By definition, an elevator shutdown requires a mechanic to be dispatched to site to correct the problem. Elevator shutdowns can occur with or without passengers in the elevator car. The REM® 2000 system is designed to detect elevator shutdowns and to indicate if passengers are in the car. Once a shutdown is detected, the REM® 2000 system communicates the shutdowns to the Service Management System. Shutdowns with passengers are allocated the highest priority and the system establishes a "voice link" with the trapped passenger in the elevator car. The Service Dispatching System is designed to forward the REM® 2000 dispatch messages to the Mechanic. The Mechanic reacts to the dispatch message by traveling to the site to perform corrective action.

A second primary function of the REM® 2000 system is to monitor, collect, store and analyze elevator performance information. The REM® system combines new data with historical data to formulate performance statistics. Door open time is an example of a measured performance parameter. The REM® system monitors the length of time from when the doors are commanded to open until the doors are fully open. A statistical trend of the door open times is calculated. An increase in the door open time indicates that the elevator door requires service. This "on-demand" service information is transmitted to the Maintenance Planning System. The Maintenance Planning System uses the information to schedule a mechanic to perform the required service.

The REM® 2000 system is divided into two major sub-systems: the REM® Resident Sub-system (RRS) and the Network Manager Sub-System (NMS). The RRS is an embedded microprocessor system that is installed at the elevator site. The RRS interfaces directly with the elevator to collect the required information. In the event of an elevator shutdown the RRS immediately alerts the Network Management System (NMS), via a dial-up modem connection, of the detected shutdown

REM 2000 System

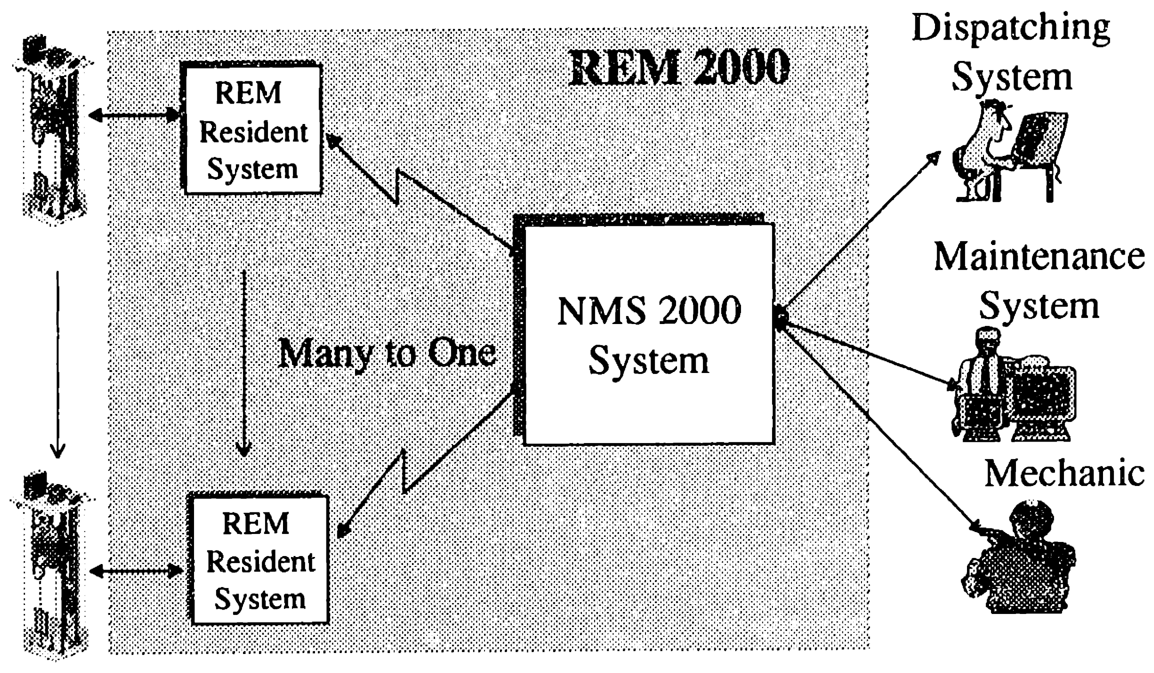


Figure 16: REM® 2000 System

The Network Manager System is the central repository for REM® data. All the data that are collected by the RRS system are periodically transmitted and stored in the NMS. The NMS adds value by performing real-time and post processing functions. Real-time processing is performed on shutdowns. On a shutdown, the NMS attaches fault isolation information to the shutdown message. The additional fault isolation information helps the mechanic locate and correct the failure promptly. The value in the post processing of historical data is to detect trends in the performance parameters. Detecting trends in the data aids efficient scheduling of elevator maintenance tasks.

The NMS is also responsible for providing the REM® 2000 interfaces to the outside world. The NMS user interface allows an operator to interrogate and analyze the REM® data. The external system interface allows external systems to communicate information to and from the REM® 2000 system. The Dispatching System and the Maintenance System are two examples of external systems that interface with the NMS. A high level architectural diagram of the NMS is shown below:

NMS 2000 System Architecture

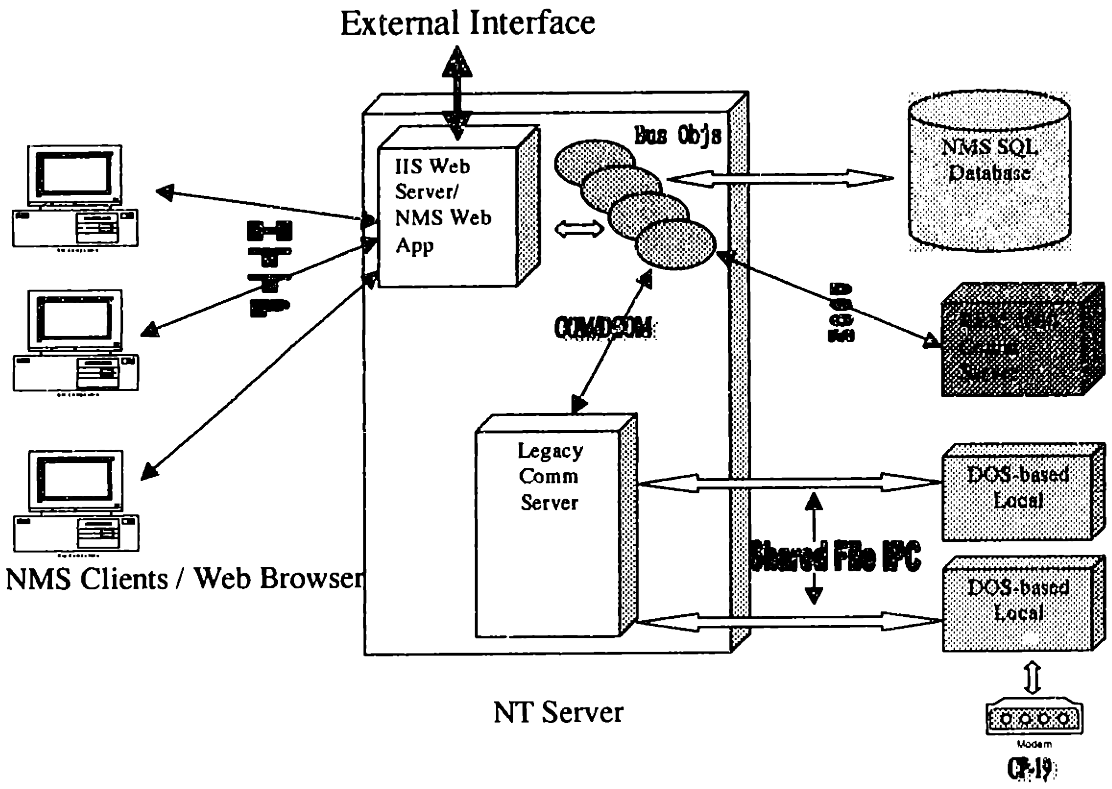


Figure 17: NMS 2000 System Architecture

From a software development technology perspective, the NMS 2000 system is a multi-tier-distributed Internet application. A graphical representation of the multiple tiers is as follows:

NMS 2000 Multi-tiered Architecture

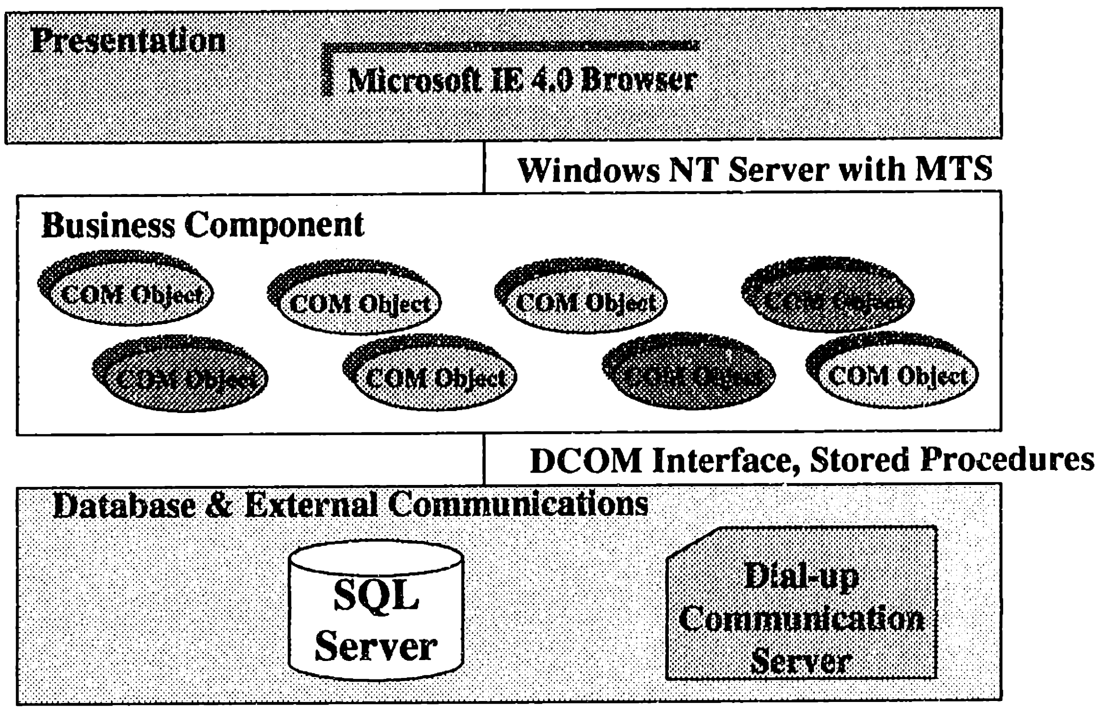

Presentation
Microsoft IE 4.0 Browser
Windows NT Server with MTS
Business Component
COM Object
DCOM Interface, Stored Procedures
Database & External Communications
SQL Server
Dial-up Communication Server

Figure 18: NMS 2000 Multi-tiered Architecture

In a multi-tier architecture the business components are implemented separately from the presentation and data components in order to enhance the scalability, reusability and robustness of the solution (Dolgicer, 1998). The key technologies in the Business Component tier are Microsoft Transaction Server (MTS) and Microsoft's Component Object Model (COM). MTS is a distributed runtime environment for COM objects. Max Dolgicer's description of MTS is as follows (Dolgicer, 1998):

> *"MTS provides a sophisticated infrastructure for activating and running objects across a network. MTS provides automatic transaction management, database connection pooling, process isolation, automatic thread-pooling, automatic object instance management, resource sharing, role-based security, transaction monitoring within a distributed application and much more. The services are necessarily for scaling server side components and supporting a substantial number of concurrent client requests. MTS performs all of these services automatically, without the need for application developers to write special code. A developer can therefore develop server-side components with a single client in mind."*

MTS is designed to facilitate component-based development. Component based development is really nothing more than modular programming with defined interfaces (Bear, 1998). Microsoft's Component Object Model (COM) defines the standard on how component based applications inter-operate. COM also defines how to build components that can be dynamically interchanged. The benefits of component based development are (Rogerson, 1997):

- *Application Customization.* Each component can be easily replace with a different component that meets the specific need.
- *Component Libraries*: Well-constructed components are reusable.
- *Distributed Components*: Components can be distributed anywhere in a high bandwidth network.

The NMS 2000 Business Components were developed using Visual Basic 6.0. The User Interface was developed using VB Script and Java Script (both from Microsoft). The client side User Interface is Microsoft's Internet Explorer 4.0. Microsoft's SQL Server 7.0 is the technology used in the database tier. Microsoft SQL is a relational database that is designed to support high volume transaction processing on Microsoft's NT server based networks. The Dial-up communication applications are internally developed executables developed in Java and C++.

4 Applying the Critical Chain and Staged Development

4.1 Introduction

The application of Critical Chain to a Staged Software Development project presented an array of challenges. The initial challenge was to scope the method of training and amount of training that would be required to introduce the development team to the concepts. The second challenge was to develop a process in which to select the features and functions that would be implemented in each stage. The third challenge was to develop the process to estimate and schedule the stages and a process to track the program was also developed.

Diversity in the team member backgrounds nourished the training challenge. Team membership consisted of eight direct employees, four newly hired software contractors, a software contractor with prior NMS experience and two technology specialists from Microsoft Corporation. The diversity in backgrounds made it difficult to gauge the level of training that would be required. The political environment created by recent company wide training initiatives also complicated the training issue.

In the past three years, many of the company employees were part of an aggressive initiative to train all development engineers on the "standard" product development processes. The goal of the initiative was to reduce the time to develop a product by fifty percent. The initiatives were developed to establish common frameworks and standards for all development projects. Many of the direct company employed NMS 2000 team members had been recently been trained on:

1. Product Development "Best Practices"
2. The Otis Software Development Process
3. The Systems Engineering Process
4. The Program Management Process

The company invested a lot of time and money to train the development staff on these four processes. At the start of the NMS 2000 project the only process that still had momentum was the Otis Software Development Process. The other initiatives had perished due to lack of support. The Otis Software Development process (OSP) is modeled after the process developed by the Carnegie Mellon Software Engineering Institute (http://www.sei.cmu.edu/). The Otis Software Development process provides guidelines on the following topics:

- Project Initiation
- Software Project Management
- Software Product Development
- Software Quality Assurance

Application of Critical Chain with a Staged Development process affected standard processes in most of the OSP segments.

On the positive side, the diversity in team membership helped to ensure that there was not team wide consensus on a single development process. The lack of an existing process eased the resistance to adoption of Critical Chain and Staged Development.

4.2 Training

The purpose of this section is to explain the training method that was employed to introduce the concepts. As previously mentioned, the training effort was constrained by the investment the company made in the just completed, relatively unsuccessful, process training initiative. In this difficult environment, the author felt it was best to incorporate Critical Chain and Staged Development training into the standard project review sessions. The author took on the complete responsibility to train all of the project internal stakeholders: the team, the REM® 2000 Program Director, peers and engineering senior management on the fundamentals of Critical Chain and Staged Development.

The Critical Chain and Staged Development concepts were first presented to the team members in a two-hour meeting. The goal of the initial presentation was to develop a team wide understanding of the fundamentals and to generate discussion. The Critical Chain training continued with a number of ad-hoc discussions between interested team members. The team members were concerned with their ability to accurately estimate the tasks and the Goldratt fifty percent task time reductions. The next formal training step was to apply the techniques in the first stage of the development cycle. The team was asked to develop the network and estimate to 50% task times. Each subsequent stage was then used to continue team training. The relatively short duration of the staged releases limited the risk of the informal training technique and allowed the team to learn the techniques rapidly. The training effort was eased by a few of the team members taking the initiative to read *The Critical Chain* (Goldratt, 1997) and *Software Project Survival Guide* (McConnell, 1998). The incremental training approach was consistent with the Staged Development approach.

The effort to training the REM® 2000 Program Director was minimal due to his previous study of the fundamentals presented in *The Goal* (Goldratt, 1992). The effort to train the Program Director was limited to his reading *The Critical Chain* (Goldratt, 1997) and a short follow-up discussion.

Members of the Engineering Senior Management team were introduced to the concepts in the first NMS 2000 quarterly project review. The first senior management review was held four months after the initial CC schedule was put into place. In that review the author scheduled a thirty-minute time slot to present a brief overview of the concepts. The senior management presentation was focused establishing the fact that buffers are not slack and progress will be extremely visible through the staged releases.

A summary of the team training effort is as follows:

- The majority of the team training was accomplished by the actual application of the techniques. The frequent pace of the stages allowed the team to learn the process incrementally. The project leader had previously studied the concepts and held a solid understanding of the concepts.
- Direct management was self-trained by reading the appropriate material and discussing the concepts with the project leader.
- Other stakeholders were familiarized with the concepts by inclusion of brief explanations of the methods in normal program review meetings.

4.3 Developing the Stage Plan

The initial step in a Staged Development process is the task of identifying the features and functions to include in each development stage. A guideline, that the NMS 2000 team decided to follow, was to put all of the components that could be associated with one feature in a stage. A feature can be viewed as subset of components that allows the user to perform a function. Each stage represents the complete development cycle for that particular feature. The staging design goal was to cleave the complete system into independent features that could theoretically be developed independently without affecting the development of other, previous and future, stages. The artifact at the end of each stage is working code that could be (if required) released at any time.

Development and analysis of the staging process identified natural boundaries in which to cleave the system. The boundaries were drawn on functional and application vectors. The functional vectors segregated the system into five core functions. The application vectors segregated the system into three distinct user applications of the NMS 2000 system. The intersection of the two vectors defined the stages.

The order of development for the stages was determined by blending three decision parameters. The first parameter was the customers wants. When asked, the customer was quick to identify the features (stages) that they would like to have developed first. The second parameter was risk (technology and requirements). High-risk items were scheduled as early as possible. The third factor used to determine order was the natural flow of development tasks. Viewed from a development perspective some features had to be developed before other features. Violation of the natural order created inefficiencies in the development process.

The definition of the stages and the order of the stages was captured and communicated in table format. The left-hand column of the table defined the core functions, the top line defined the use perspectives and the boxes identified the stages. The number in the boxes indicated the implementation order. Each box in the table represents a development stage.

	Application 1	Application 2	Application 3
Core Function 1	1	9	3
Core Function 2	2	10	4
Core Function 3	7	11	5
Core Function 4	8	12	6
Core Function 5	13	14	15

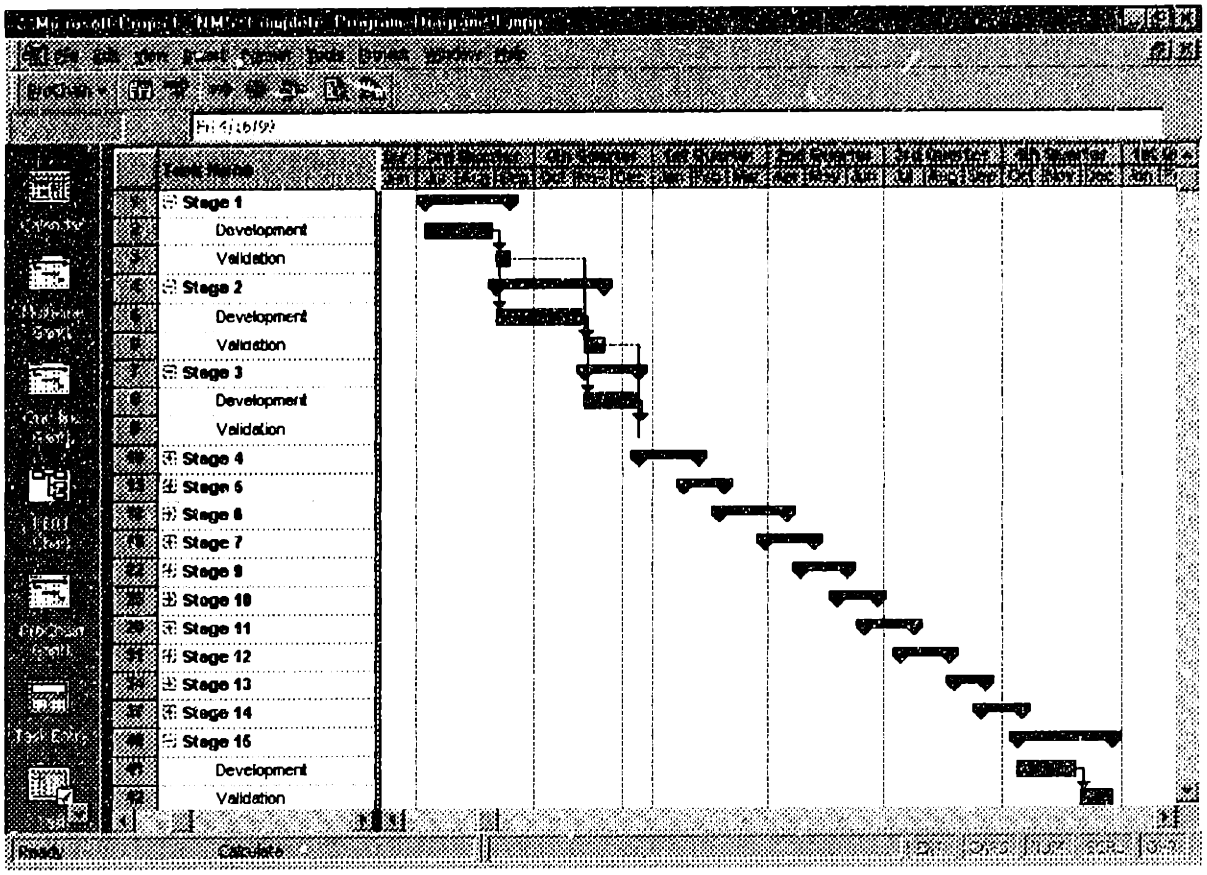
Stage Number and Order

Figure 19: Stage Development Order

4.4 Project Scheduling

Software estimation and scheduling is difficult (McConnell, 1996). McConnell indicates that accurate software estimation is a "process of gradual refinement". Understanding the difficulty drove the author to investigate an estimation process that allowed the team to quickly learn their productivity level without getting bogged down in stifling estimation details. The process developed was an iterative scheduling process. The iterative scheduling process, which was consistent with the staged development process, provided a quick estimate of the scope of the program, the first level of schedule detail on next stage and low level detail on the current stage.

The first step in the iterative scheduling process was to estimate, on a high level, the resources and time that would be required to complete each stage. The development time for each stage was entered into a non_Critical Chain Gantt Chart to determine the aggregate scope of the project. The complete project Gantt chart was purposely kept simple and easy to update. The author felt that these high-level estimates were the most efficient and accurate estimates that could be provided, at the start of the program with a reasonable amount of effort, for the complete program. This feeling was based on the industry wide estimation statistics and the developer low level of knowledge of the software technology that was being applied. As time marched on and the developers gained experience the complete program schedule estimates would became more and more accurate. A simplified example of the initial complete system schedule is as follows. All stages contained a development and validation phase. The "+" in the task name indicates which tasks have been rolled-up.



Task Name
Stage 1
Development
Validation
Stage 2
Development
Validation
Stage 3
Development
Validation
Stage 4
Stage 5
Stage 6
Stage 7
Stage 9
Stage 10
Stage 11
Stage 12
Stage 13
Stage 14
Stage 15
Development
Validation

Figure 20: Complete Project Schedule

At the completion of each development stage the planning/estimation process was re-initiated. Using what was just learned; the complete schedule was re-evaluated for validity. The next two staged were detailed out and a Critical Chain schedule was developed for the next stage. This process was repeated at the end of every stage. As the project progressed more and more productivity data was collected and factored into the future estimates. As time progressed the uncertainty in the estimates are expected to decrease. Steve McConnell has label this phenomena as the "Estimate Convergence Graph" (McConnell, 1996).

Estimate Convergence Graph

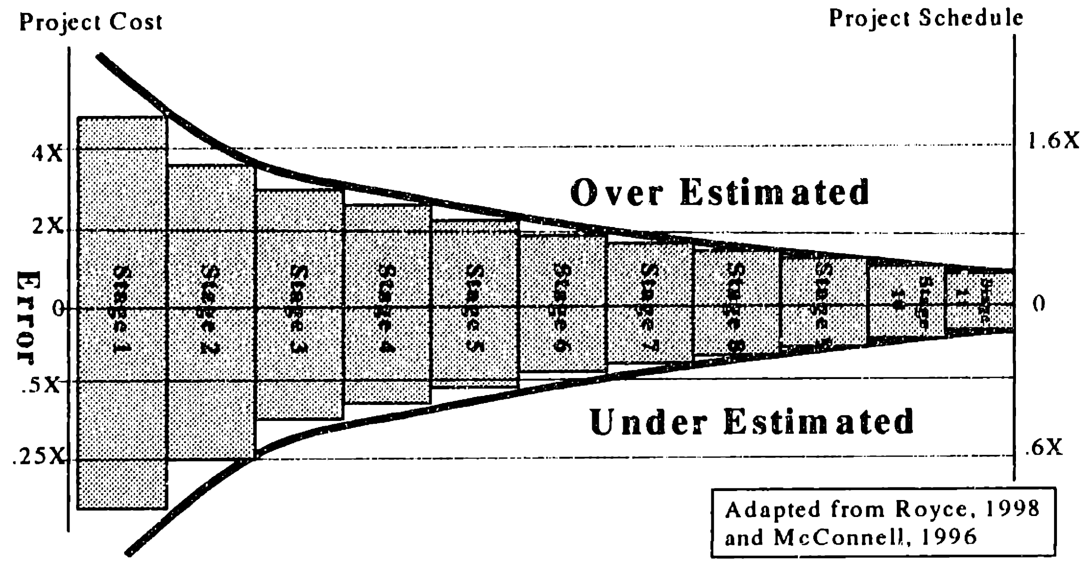


Figure 21: Estimate Convergence Graph

After the overall scope of the project was reevaluated the developers were asked to develop the first level of detailed work breakdown structures and task estimates for the next two staged releases. The synergy between the first level detailed estimates and the high-level estimates was used to validate the complete program high-level estimates. A more detailed Critical Chain schedule was only developed for the current stage. Developing a Critical Chain schedule for only the current stage helped to reduce the complexity and needless detailed re-estimation of the complete schedule as the program progressed.

The initial step in creating the Critical Chain schedule was an analysis of the detailed WBS (tasks), project network (dependencies) and task estimates (durations). The analysis included:

1. The process of eliminating unnecessary tasks
2. Adding of tasks that were forgotten
3. Establishing which tasks that had to be serial and which could be done in parallel
4. Establishing which resources had the skills to complete the task
5. Determining the 50% confidence level of the task estimates.

The resulting information from the analysis was entered into a Critical Chain schedule for that specific stage. Once the Critical Chain scheduled was developed, the team ignored the complete program schedule (until the stage was complete) and focused their energy on implementation the features and functions for that stage as quickly as possible. Each stage was tracked as an independent project.

A pictorial of the staged scheduling process that was implemented is as follows:

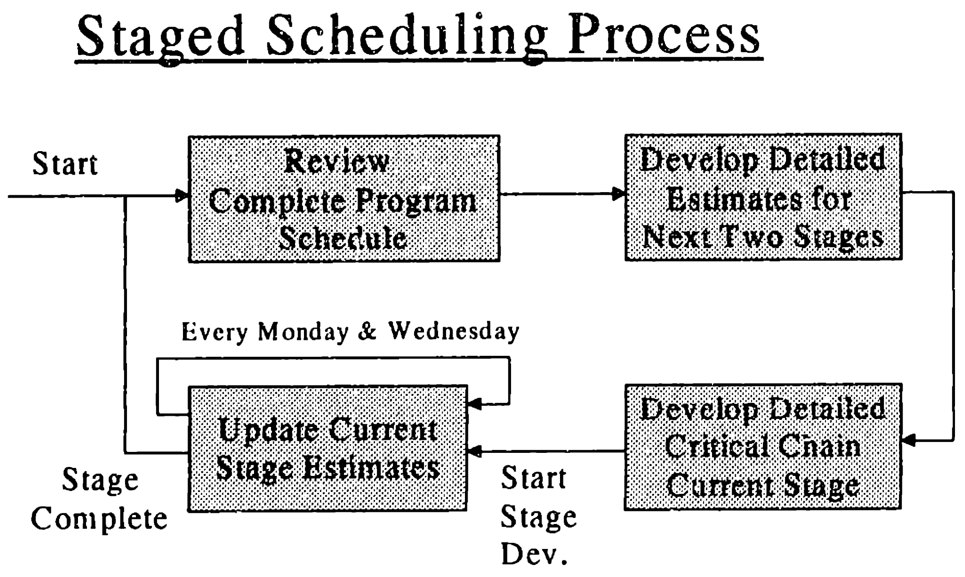

Staged Scheduling Process
Start
Review Complete Program Schedule
Develop Detailed Estimates for Next Two Stages
Every Monday & Wednesday
Update Current Stage Estimates
Develop Detailed Critical Chain Current Stage
Stage Complete
Start Stage Dev.

Figure 22: Staged Scheduling Process

The detailed Critical Chain Schedule that is developed for each stage is a reasonably complex schedule with over 100 dependent and independent tasks. A snapshot of that contains about 25% of a detailed schedule for one of the stages is as follows:

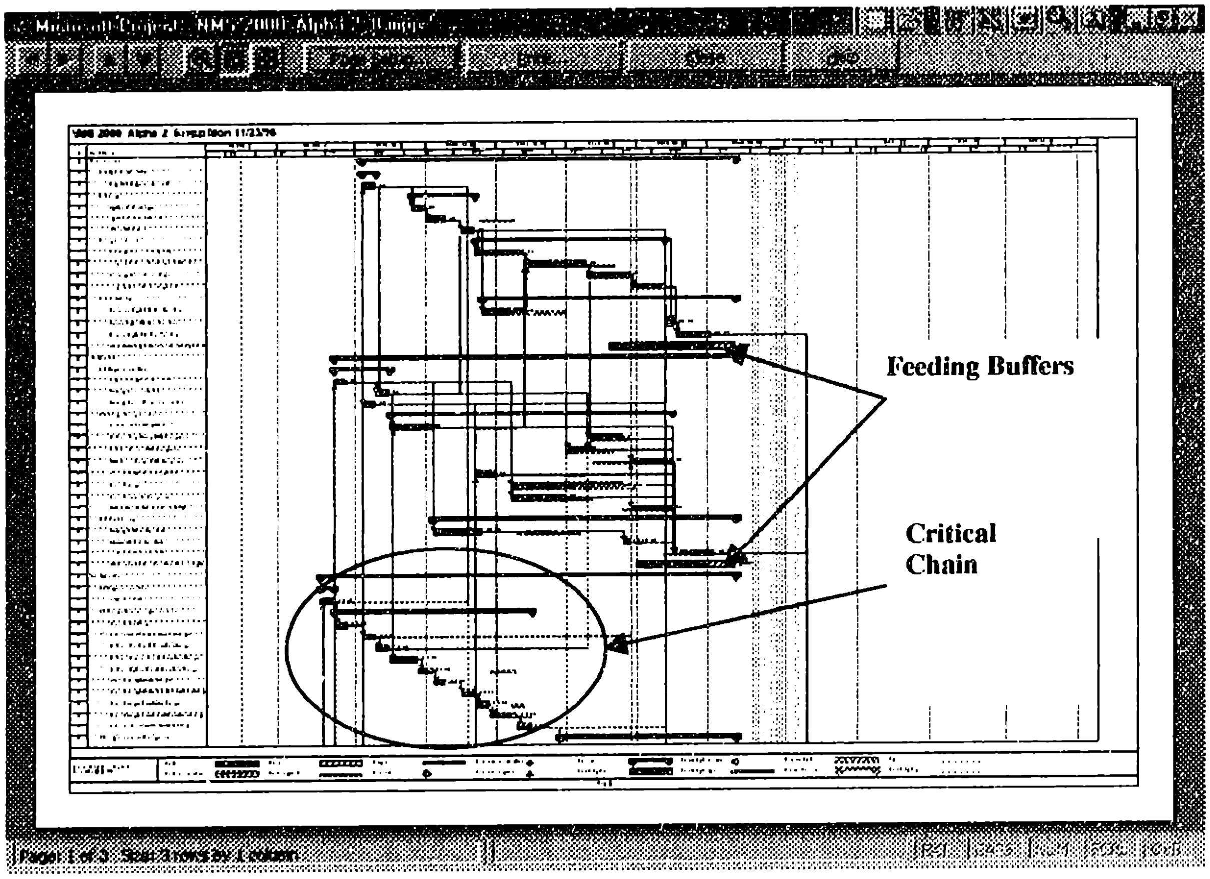


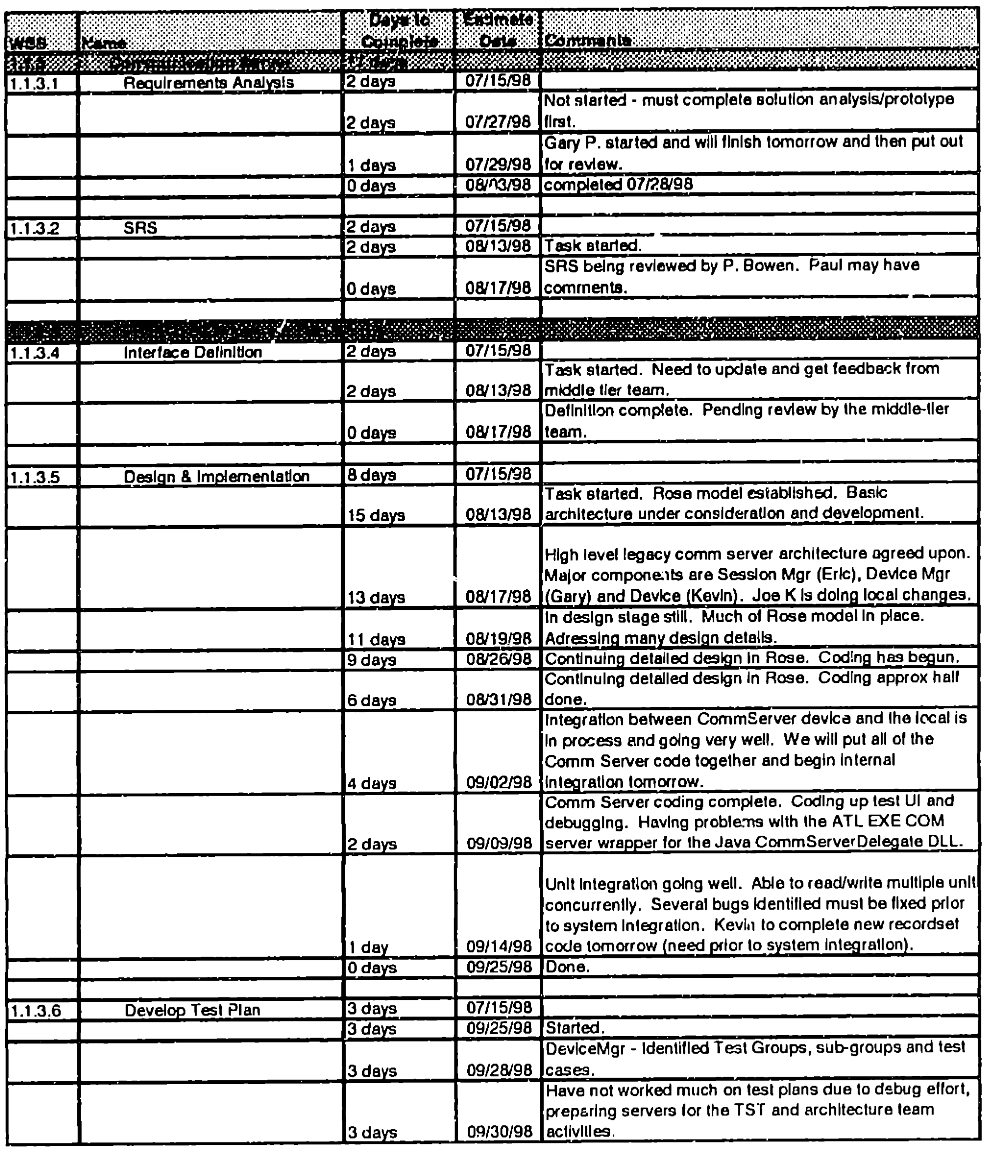
Figure 23: Detailed Critical Chain Schedule

4.5 Project Tracking

The project tracking process that was implemented was designed to allow an efficient method for the development team to communicate task status to program management and for program management to communicate program status back to the team and to other stakeholders. The program was tracked on two levels. The first tracking reference point was the complete program target end date. The second tracking reference point was the current stage end date.

As stated above, the validity of the program end date was only checked at the end of each stage. When a stage was completed, the developers used their knowledge from developing previous stages to re-estimate the remaining work in the program. The complete program was tracked at a high-level in a straight Microsoft Project schedule.

Tracking within a stage development cycle was a much tighter loop. The development team used a spreadsheet on a shared harddrive as their means to communicate task status to program management. The developers were required to update the spreadsheet with the "number of days to complete" their current task. The developers were also requested to provide comments explaining the updated estimate. The comments were critical to the post stage development process analysis. A sample of that spreadsheet is shown below:



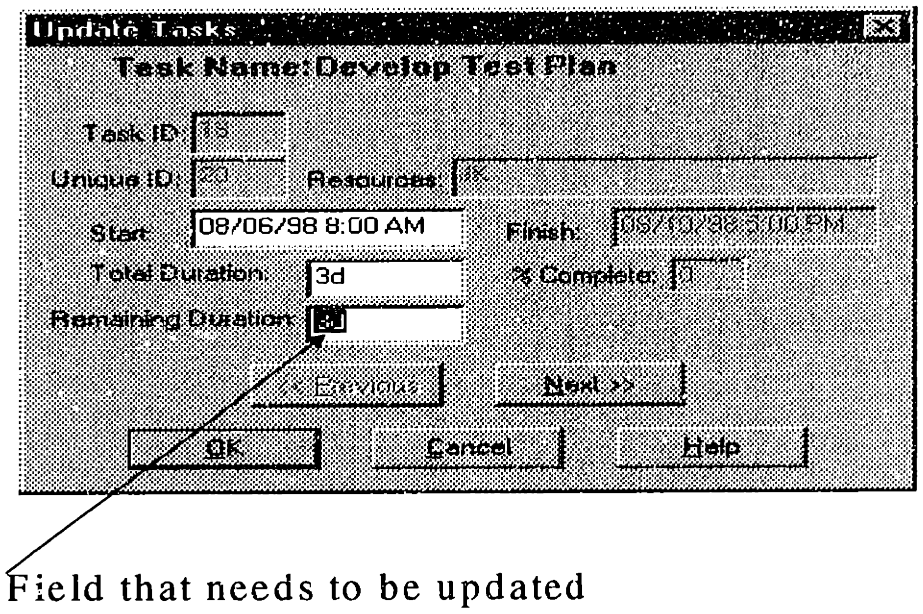
WBS	Name	Days to Complete	Estimate Date	Comments
1.1.3	[illegible]	[illegible]		
1.1.3.1	Requirements Analysis	2 days	07/15/98	
		2 days	07/27/98	Not started - must complete solution analysis/prototype first.
		1 days	07/29/98	Gary P. started and will finish tomorrow and then put out for review.
		0 days	08/03/98	completed 07/28/98
1.1.3.2	SRS	2 days	07/15/98	
		2 days	08/13/98	Task started.
		0 days	08/17/98	SRS being reviewed by P. Bowen. Paul may have comments.
[illegible]	[illegible]	[illegible]		
1.1.3.4	Interface Definition	2 days	07/15/98	
		2 days	08/13/98	Task started. Need to update and get feedback from middle tier team.
		0 days	08/17/98	Definition complete. Pending review by the middle-tier team.
1.1.3.5	Design & Implementation	8 days	07/15/98	
		15 days	08/13/98	Task started. Rose model established. Basic architecture under consideration and development.
		13 days	08/17/98	High level legacy comm server architecture agreed upon. Major components are Session Mgr (Eric), Device Mgr (Gary) and Device (Kevin). Joe K is doing local changes.
		11 days	08/19/98	In design stage still. Much of Rose model in place. Adressing many design details.
		9 days	08/26/98	Continuing detailed design in Rose. Coding has begun.
		6 days	08/31/98	Continuing detailed design in Rose. Coding approx half done.
		4 days	09/02/98	Integration between CommServer device and the local is in process and going very well. We will put all of the Comm Server code together and begin internal integration tomorrow.
		2 days	09/09/98	Comm Server coding complete. Coding up test UI and debugging. Having problems with the ATL EXE COM server wrapper for the Java CommServerDelegate DLL.
		1 day	09/14/98	Unit integration going well. Able to read/write multiple unit concurrently. Several bugs identified must be fixed prior to system integration. Kevin to complete new recordset code tomorrow (need prior to system integration).
		0 days	09/25/98	Done.
1.1.3.6	Develop Test Plan	3 days	07/15/98	
		3 days	09/25/98	Started.
		3 days	09/28/98	DeviceMgr - Identified Test Groups, sub-groups and test cases.
		3 days	09/30/98	Have not worked much on test plans due to debug effort, preparing servers for the TST and architecture team activities.

Figure 24: Task Update Sheet

At the start of the development stage the WBS number, task name, initial estimate and initial estimation durations are inserted into the spreadsheet. By the end of the day Monday and Wednesday, each developer was expected to update the spreadsheet with a new days to complete estimate (logging the estimation date) and comments on that task. The effort to update the spreadsheet required less than five minutes per update.

The program manager reviewed the task update table every Tuesday and Thursday morning. Information from the task update table was entered into the staged ProChain^{TM} schedule. The information collected (days to complete) was selected to be consistent with ProChain^{TM} task update feature. The ProChain^{TM} update task feature required entry of the days to complete. A pictorial of the ProChain^{TM} task update screen is as follows:

ProChain Update Task Screen



Update Tasks

Task Name: Develop Test Plan

Task ID: 13

Unique ID: 20

Resources: [illegible]

Start: 08/06/98 8:00 AM

Finish: 08/10/98 5:00 PM

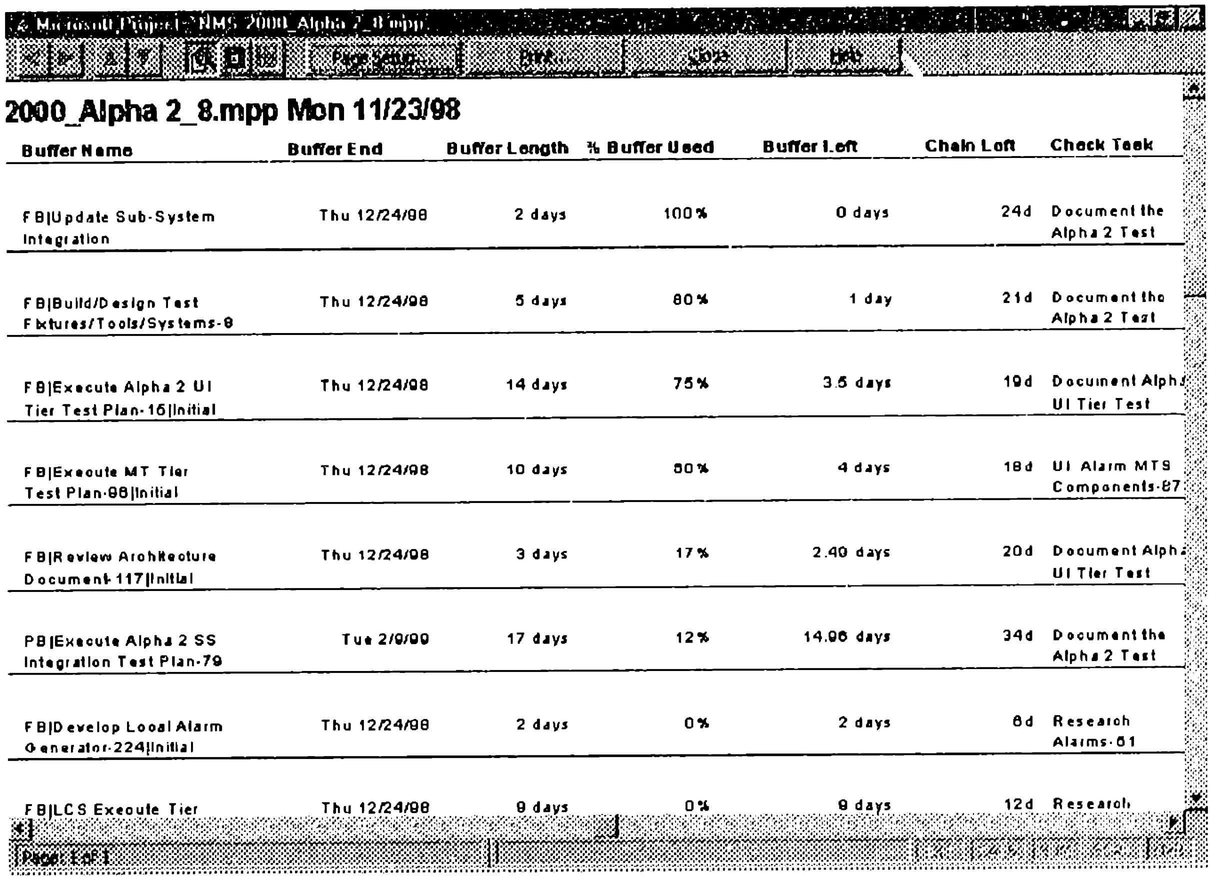
Total Duration: 3d

% Complete: 0

Remaining Duration: [illegible]

Figure 25: Task Update Screen

The program manager used the standard ProChain^{TM} Buffer and Project Task reports to communicate stage status to all of the program stakeholders. Samples of the reports that were used are as follows:



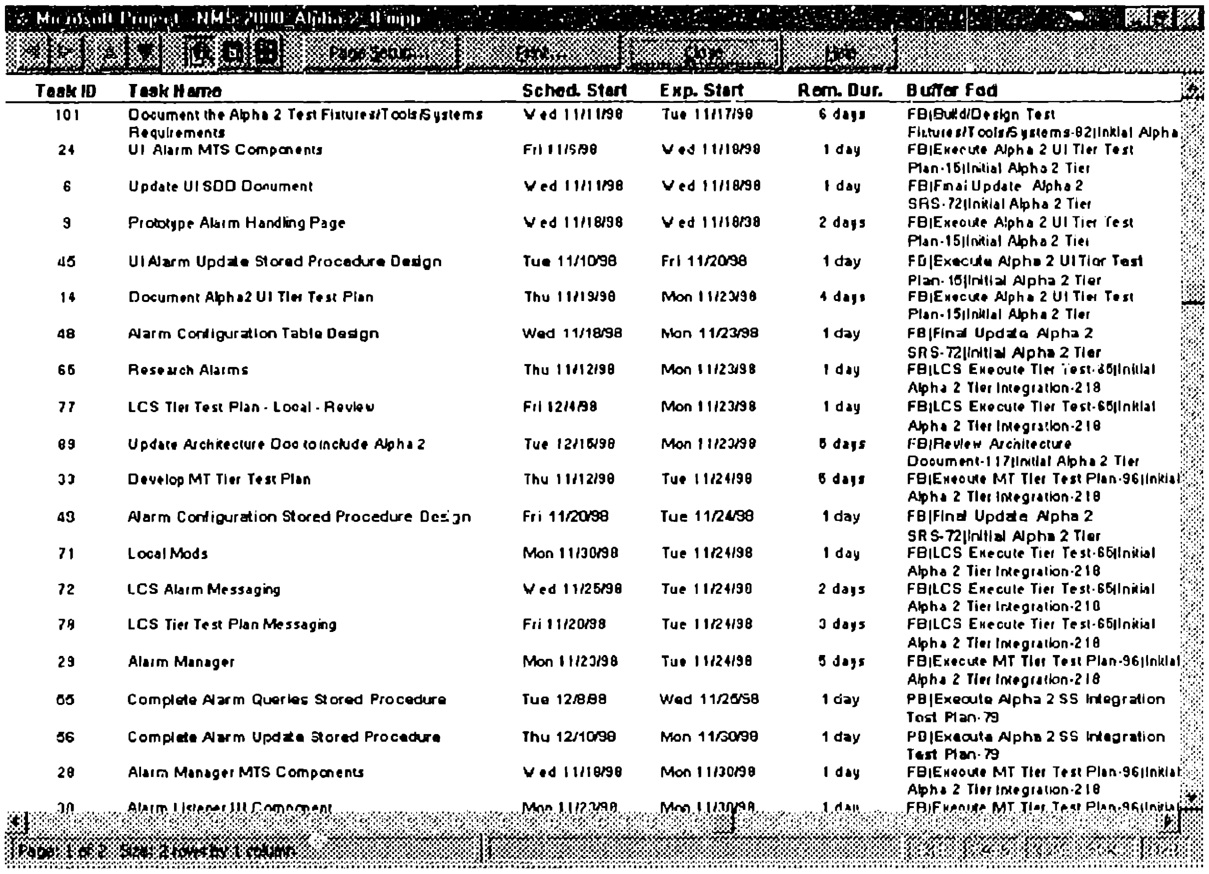
2000_Alpha 2_8.mpp Mon 11/23/98

Buffer Name	Buffer End	Buffer Length	% Buffer Used	Buffer Left	Chain Left	Check Task
FB|Update Sub-System Integration	Thu 12/24/98	2 days	100%	0 days	24d	Document the Alpha 2 Test
FB|Build/Design Test Fixtures/Tools/Systems-8	Thu 12/24/98	5 days	80%	1 day	21d	Document the Alpha 2 Test
FB|Execute Alpha 2 UI Tier Test Plan-16|Initial	Thu 12/24/98	14 days	75%	3.5 days	19d	Document Alpha UI Tier Test
FB|Execute MT Tier Test Plan-98|Initial	Thu 12/24/98	10 days	60%	4 days	18d	UI Alarm MTS Components-87
FB|Review Architecture Document-117|Initial	Thu 12/24/98	3 days	17%	2.49 days	20d	Document Alpha UI Tier Test
PB|Execute Alpha 2 SS Integration Test Plan-79	Tue 2/9/99	17 days	12%	14.96 days	34d	Document the Alpha 2 Test
FB|Develop Local Alarm Generator-224|Initial	Thu 12/24/98	2 days	0%	2 days	6d	Research Alarms-61
FB|LCS Execute Tier	Thu 12/24/98	9 days	0%	9 days	12d	Research

Figure 26: Stage 2 Buffer Report

The key field in the Buffer Report was the "% Buffer Used" field. This single field provided insight to how well the stage development was tracking.

The second standard report that was used was the Project Task Report. The project Task Report was used to provide up to date information on update expected task start and end times. A sample of the Project Task Report is shown below:



Microsoft Project - NMS 2000_Alpha 2_0.mpp

Task ID	Task Name	Sched. Start	Exp. Start	Rem. Dur.	Buffer Fed
101	Document the Alpha 2 Test Fixtures/Tools/Systems Requirements	Wed 11/11/98	Tue 11/17/98	6 days	FB|Build/Design Test Fixtures/Tools/Systems-82|Initial Alpha
24	UI Alarm MTS Components	Fri 11/6/98	Wed 11/18/98	1 day	FB|Execute Alpha 2 UI Tier Test Plan-15|Initial Alpha 2 Tier
6	Update UI SDD Document	Wed 11/11/98	Wed 11/18/98	1 day	FB|Final Update Alpha 2 SRS-72|Initial Alpha 2 Tier
9	Prototype Alarm Handling Page	Wed 11/18/98	Wed 11/18/98	2 days	FB|Execute Alpha 2 UI Tier Test Plan-15|Initial Alpha 2 Tier
45	UI Alarm Update Stored Procedure Design	Tue 11/10/98	Fri 11/20/98	1 day	FB|Execute Alpha 2 UI Tier Test Plan-15|Initial Alpha 2 Tier
14	Document Alpha2 UI Tier Test Plan	Thu 11/19/98	Mon 11/23/98	4 days	FB|Execute Alpha 2 UI Tier Test Plan-15|Initial Alpha 2 Tier
48	Alarm Configuration Table Design	Wed 11/18/98	Mon 11/23/98	1 day	FB|Final Update Alpha 2 SRS-72|Initial Alpha 2 Tier
66	Research Alarms	Thu 11/12/98	Mon 11/23/98	1 day	FB|LCS Execute Tier Test-65|Initial Alpha 2 Tier Integration-218
77	LCS Tier Test Plan - Local - Review	Fri 12/4/98	Mon 11/23/98	1 day	FB|LCS Execute Tier Test-65|Initial Alpha 2 Tier Integration-218
89	Update Architecture Doc to include Alpha 2	Tue 12/15/98	Mon 11/23/98	5 days	FB|Review Architecture Document-117|Initial Alpha 2 Tier
33	Develop MT Tier Test Plan	Thu 11/12/98	Tue 11/24/98	6 days	FB|Execute MT Tier Test Plan-96|Initial Alpha 2 Tier Integration-218
43	Alarm Configuration Stored Procedure Design	Fri 11/20/98	Tue 11/24/98	1 day	FB|Final Update Alpha 2 SRS-72|Initial Alpha 2 Tier
71	Local Mods	Mon 11/30/98	Tue 11/24/98	1 day	FB|LCS Execute Tier Test-65|Initial Alpha 2 Tier Integration-218
72	LCS Alarm Messaging	Wed 11/25/98	Tue 11/24/98	2 days	FB|LCS Execute Tier Test-65|Initial Alpha 2 Tier Integration-218
79	LCS Tier Test Plan Messaging	Fri 11/20/98	Tue 11/24/98	3 days	FB|LCS Execute Tier Test-65|Initial Alpha 2 Tier Integration-218
29	Alarm Manager	Mon 11/23/98	Tue 11/24/98	5 days	FB|Execute MT Tier Test Plan-96|Initial Alpha 2 Tier Integration-218
65	Complete Alarm Queries Stored Procedure	Tue 12/8/98	Wed 11/25/98	1 day	PB|Execute Alpha 2 SS Integration Test Plan-79
56	Complete Alarm Update Stored Procedure	Thu 12/10/98	Mon 11/30/98	1 day	PB|Execute Alpha 2 SS Integration Test Plan-79
28	Alarm Manager MTS Components	Wed 11/18/98	Mon 11/30/98	1 day	FB|Execute MT Tier Test Plan-96|Initial Alpha 2 Tier Integration-218
30	Alarm Listener UI Component	Mon 11/23/98	Mon 11/30/98	1 day	FB|Execute MT Tier Test Plan-96|Initial

Page: 1 of 2 Size: 2 rows by 1 column

Figure 27: Stage 2 Project Task Report

The communication of the complete schedule status was provided in terms of gain and slippage to the target end date.

5 The Results

The results of applying Critical Chain within a Staged Software development project support Frederick Brooks' proposition that there are "no silver bullets" in software development (Brooks, 1995). Critical Chain within a Staged Software development provided positive results and identified areas of opportunity.

This hereafter text is a summary of the results of the first three stages of the NMS 2000 development program. The first three stages represented a total of 9 months calendar time. The timing and pace of the first three development stages was as follows:

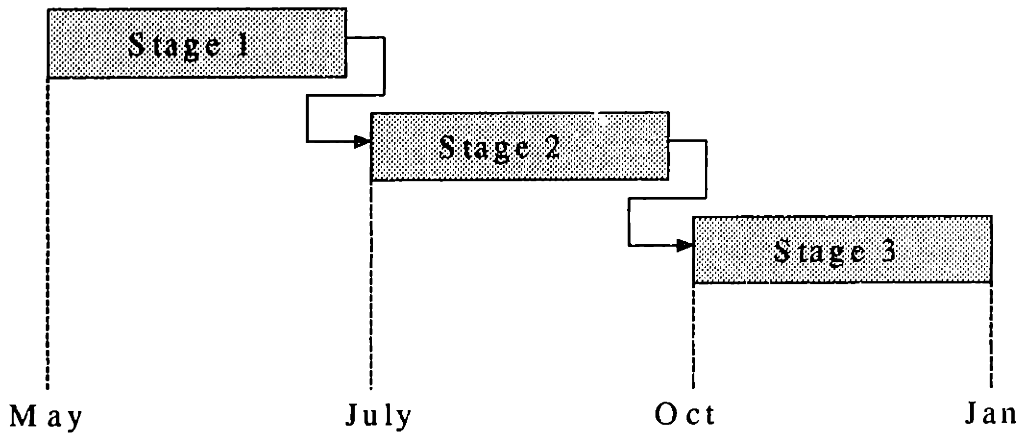


Figure 28: NMS 2000 Development Pace

Stage 1 development represented the transition from the architectural and requirements analysis phase to the development phase. Critical Chain and Staged Development was introduced to at the start of stage 1. The introduction was complicated by the fact that over 70% of the development team had no direct experience with developing software in the target development environment. Because of this inexperience the stage 1 estimates were understood to contain a large amount of uncertainty. The development goal for stage 1 was to gain familiarity with the complete staged development cycle and the software technology that was being applied. It was decided that the functional requirements of stage 1 would be adjusted to ensure that the team completed the development cycle in the estimated time (including the buffer). The function selected for stage 1 was selected to meet the stage 1 goal. Stage 1 was estimated to take 7 weeks to complete and the results are as follows:

- Completed stage 1 development three days after the project buffer end.
- Requirements had to be significantly compromised to meet the schedule.
- Large variations in task time, due to technological difficulties, made it difficult to track task times on a Critical Chain schedule.
- Technological progress in stage 1 was significant.

From a development perspective stage 1 was declared a success. The large amount of learning and the technological hurdles that were solved in stage 1. Forcing the implementation of the complete development cycle allowed the team to discover unknown technology issues early. The early discovery of these issues proved to be valuable in completing the system architecture and development plans.

From a schedule perspective, stage 1 was a failure. The initial scope of stage 1 had to be significantly reduced to meet the planned schedule. Unfamiliarity with the technology was the main reason for the schedule delay. The early consumption of the feeding buffers put most of the development task on the Critical Chain. The creation of multiple Critical Chains directly exposed the project buffer to delays in a number of tasks.

The technical difficulties in completing stage 1 consumed energy that was slated to do the initial design analysis for stage 2. The result was that the stage 2 design process was hurried and estimates were provided without a complete understanding of the design. Stage 2 was estimated (with the project buffer) to be a 10 week project. Stage 2 took 13 weeks to complete. The main reasons for the schedule delay were:

- A significant portion of the database schema had to be changed in the middle of development.
- Many required tasks were not identified and therefore not estimated at the start of stage 2.
- Team was still marching up the technology learning curve.

The positive effects of Staged Development were visible at the end of stage 2. The team had generated working code that could be reviewed by lead users. The second pass through the development cycle also made the team much more comfortable with Critical Chain and the Stage Development Process. The end of stage 2 also produces signs that the "Cone of Uncertainty" in task estimation was starting to narrow. Near the end of stage 2 the development team started to provide accurate "estimates to complete" entries in the task update sheets.

Leveraging on a lesson learned in stage 2; the team did a much-improved job at requirement analysis and technology prototyping before the official initiation of the stage 3 development cycle. Also, a team productivity history was used to estimate stage 3.

5.1 What Worked

The advantages of using the Critical Chain method in the NMS 2000 Staged Software Development process can be cleaved into advantages derived from Staged Development and advantages derived from Critical Chain.

5.1.1 Staged Development Process

5.1.1.1 Early Learning

One of the most significant benefits from Staged Development was the forced early learning of the complete development cycle. Zigzagging between the development process and detailed requirements analysis/planning process provided the team with a means to briskly learn and apply the new knowledge. The team benefited from gaining technological knowledge in early stages and then applying that knowledge in the planning and development of the subsequent stages. In a waterfall model detailed technical learning does not occur until all of the project planning has been completed. This serial effect limits the level of knowledge that can be inserted into the development process.

5.1.1.2 Staged Scheduling

The staged scheduling cycle proved to be an effective method to limit the time and resources required to schedule a program. As the team completed a stage, and technological and productivity knowledge was gained, the team was in a much improved position to provide estimates. The staged scheduling process allowed the team to leverage the learning that contributes to the natural effect of the Estimate Convergence Curve.

In a non-staged scheduling process the team is asked to provide detailed estimates at the start of the project. These estimates, by definition, contain a great deal of uncertainty. The process of trying to provide accurate estimates when so much is unknown results in a delay to the start of the project with limited benefit.

5.1.1.3 Early "Working Code"

The generation of working code at the end of each stage provided notable benefits to the project. The leading benefit was the ability to honestly measure progress. Working code could be evaluated, validated and reviewed by the project stakeholders. Early evaluation of the software provided the information required to isolate perceived work complete (work complete plus undiscovered rework) from actual work complete. The working code artifact provided a stable metric to truly measure the level of work complete.

5.1.1.4 Clear and Concise Short Term Goals

Each stage in the project was viewed as a mini-project. The mini project had defined goals, requirements and a detailed schedule. These mini projects provided the development team very real and attainable short-term goals to strive for.

5.1.2 Critical Chain

5.1.2.1 Program Status Method

One of the major benefits of Critical Chain was the simple and consistent method of providing program status. In a non-Critical Chain program schedule it is always difficult to determine the "real" effect of any task duration slippage. The Critical Chain buffer report provided an easy and simple method to report project status. The buffer method addresses a perennial shortfall of traditional program management methods. Traditional methods do not provide an obvious early indication that a project is slipping. The Critical Chain buffers provided sufficient early warning signals. The early warning signals provided enough time to allow management, and the team, to react and attempt to recover.

The Task Update Sheets that were developed provided a standard and efficient means for the team to provide status input. The frequent updating of "days to complete" on the Task Update Sheets provided timely and accurate feedback on the actual status of a task. The days to complete estimate metric proved to be an easy metric to estimate and communicate.

The task update sheets also provided valuable information when conducting the between stage postmortems. The task sheets captured the actual task duration estimates overtime and comments from the developers. Information in the task update sheets was captured in Microsoft Excel. Capturing the information in Excel allowed the team to easily plot information on specific tasks. For example, a plot of task duration estimates overtime highlights the pace at which the developer(s) converged on completing a task. The plot shown below highlights the following observations: 1) the original estimate of 8 days was increased to 15 days as soon as the task was started, 2) the "Completion Convergence" slope (days to complete / calendar days) flattened out as the task progressed and 3) the actual duration of the task was from 8/12/98 to 9/23/98 (29 working days). Information that is captured and highlighted in the Task Update Sheets is critical to ensuring rapid estimate convergence.

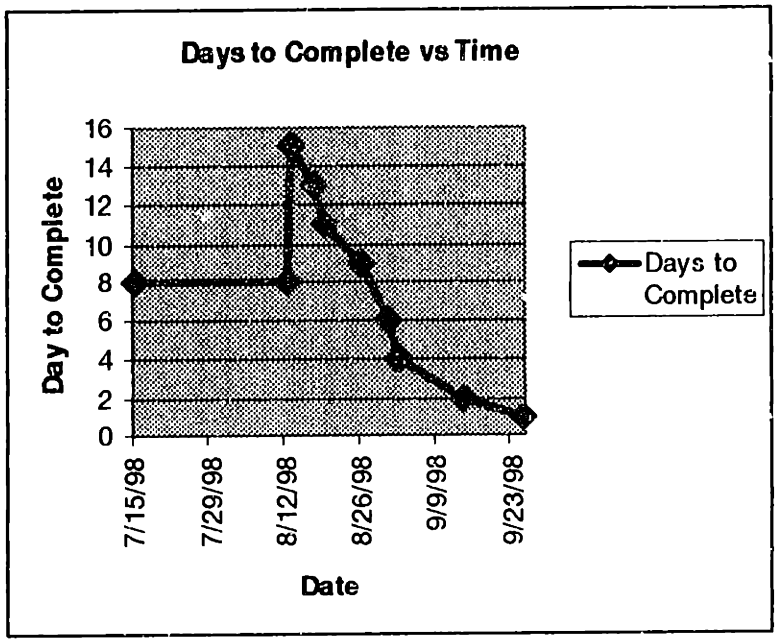


Figure 29: Days to Complete vs. Time

5.1.2.2 Development Team Focus

Critical Chain provided a method for focusing the development team on the tasks that paced the development project. Understanding the Critical Chain "bottleneck" concept encouraged the team to become more schedule aware. The Critical Chain method provided the framework for the team to ask: "Who is on the Critical Chain?" and "What can we do to make that task go faster?"

Initially the team did not grasp the "a day lost on the Critical Chain is a day lost to the project" concept and team members on the chain were still receiving low priority interrupts from other team members. As time progressed the team started to internalize the concept and started to focus on helping the person on the chain to get their task complete. A slight paradigm shift occurred when some of the developers realized that there were some benefits to being on the critical chain. Being on the chain meant: 1) you did not have to attend non-critical meetings 2) you did not have to do low priority paperwork (somebody else did it) and 3) the rest of the team was focused on helping you complete your task.

5.1.2.3 Gold-plating

Gold-plating is a significant factor software development project. Most software development tasks have a 2-day, a 2-week or a 2-month solution. The development of detailed schedules for each stage release encouraged the development to gravitate toward the 2-day solutions.

Establishing and communicating the importance of the Critical Chain also helped to minimize "gold-plating". A developer on the Critical Chain was aware that taking time to add "nice to have features" would delay the project schedule. This awareness encouraged the developers to complete their task as soon as possible.

5.1.2.4 Stabilizing the Critical Chain

Stabilizing the Critical Chain with feeding buffers proved to be a valuable concept. In all stages, except one, the feeding buffers provided the necessary buffer to keep the Critical Chain from switching from one path to the next. This stability helped to eliminate confusion created when the critical path switches form one series of tasks to another. The Critical Chain that was established and announced at the start of a stage remained the Critical Chain throughout the stage development. This stability allowed the developers to clearly understand the schedule leverage points.

5.2 What Did Not Work

The areas of the process that did not work can be aggregated into the following areas:

- 50% Task Times for all tasks
- Rescheduling
- Planning Resource Availability
- Rework Planning
- Starting Tasks As Late As Possible
- Strict Task Flows

5.2.1 Critical Chain

5.2.1.1 50% Task Duration

The Goldratt suggested (Goldratt, 1997) method for arriving at the 50% task duration time did not work. The recommendation was to take the normal (80% confidence level) task estimates and cut it in half. The recommended method was unsuccessfully applied in the stage 1 planning cycle.

The task duration estimates in stage 1 of the NMS 2000 program were known to contain large amounts of uncertainty. The high level of uncertainty was unavoidable in stage 1 because of: 1) application of new technology 2) the formation of a new team, 3) the teams unfamiliarity with the project requirements and 4) the general "Estimate Convergence" factor. These factors caused the optimistic development team to severely underestimate the task at hand.

Globally applying the Critical Chain 50 % task duration rule to underestimate tasks compounded the effect of the initial unfeasible estimates. The effects of cutting severely underestimated tasks in half were:

1) Decreased developer buy-in and motivation, the task duration was clearly no longer feasible.
2) Schedule was completely off plan, could not track to baseline plan
3) The size and the placement of the buffers were no longer valid.

As an example, a task in stage 1 was originally estimated to be a 10-day task. The 50% cut brought it down to a 5-day task. Five days into the task, the developer realized that the task was under scoped and re-estimated the task for 15 additional days to complete (20 total). The 20-day total time was almost three times the 50% task time plus the buffer allocated to that task. This one event consumed over half of the total stage project buffer.

In hindsight, considering the high level of uncertainty, cutting the 10-day task by less than 50 % (say 30%) and creating a larger buffer (say 100% of 80% estimate) would have improved the quality of the schedule. The 30% cut would generated a task time of 7 days and the 100% buffer allocation would have added 10 days to the project buffer. Therefore, the total buffered task time allocated for this task would have been 17 days. The 17 days proved to be a much better estimate of the actual 20-day task time.

5.2.1.2 Modifying the Schedule

The difficulty to add and subtract tasks from a baseline Critical Chain schedule proved to be prohibitive. Discoveries in the stage 2 development forced the need to add tasks to the Critical Chain schedule. In a Critical Chain schedule the buffer, which is an accumulation of time cut from each task on the chain, is not encapsulated within the task. Therefore it's more difficult and time consuming to add, remove and move (on/off of the Critical Chain) tasks. Every time a task is added, removed or moved the Critical Chain and the associated buffers need to be checked and in most cases recalculated. One advantage to keeping the padding in each task is that the task time is encapsulated and can be added, removed or moved within the schedule.

5.2.1.3 Resource Planning

The attempt to schedule each of the development stages as separate project created resource-allocation difficulties. Each staged Critical Chain schedule included the complete life cycle of steps to complete the stage. The last step being validation testing. The testing cycle consumed a significant portion of the scheduled calendar time and was generally not performed by the developer. The Critical Chain method made it easy to determine when the stage would be complete, but made it difficult to determine when resources within the schedule would be available to start preliminary development on the next stage.

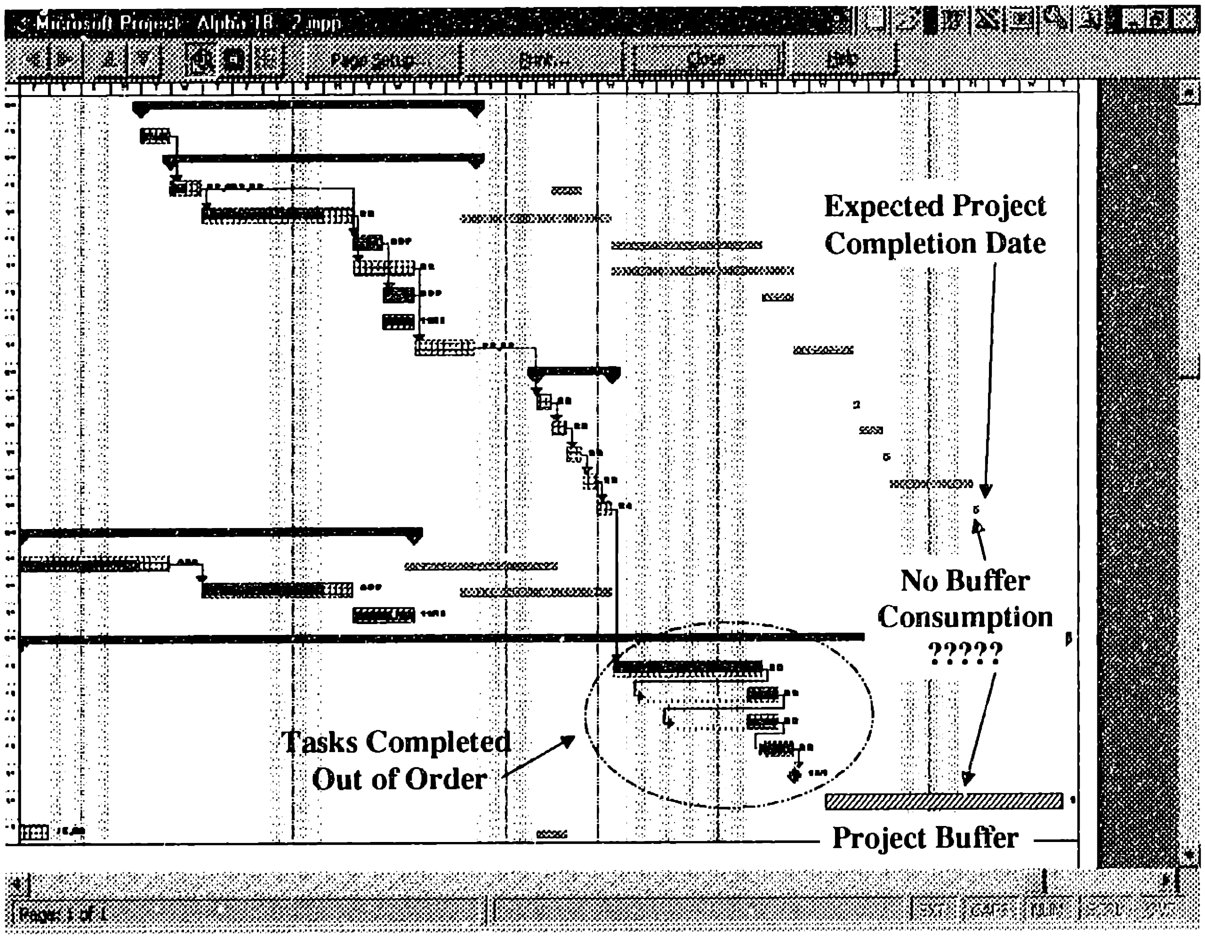
5.2.1.4 Starting Task As Late As Possible

The work in process (WIP) minimization theories of Critical Chain conflicted with scheduling employees on a dedicated team. Every person on the NMS 2000 team was 100% dedicated to the project. All team members were assigned roles and responsibilities that were consistent to their specific skill set. Therefore, resources were available to start tasks as early as possible. The Critical Chain methodology starts tasks as late as possible. The theoretical result, of starting as late as possible, is that dedicated resources would sit idle until the latest start time. Reality is that the resource started the task as soon as they are available and continued to work it until the Critical Chain due date. The ProChain^{TM} software was consistent with the Critical Chain rule and scheduled all tasks to start as late as possible. This situation, in which developers were starting task before they were scheduled, made it difficult to log the actual task duration of that specific task and to identify when developers were available to do other tasks.

5.2.1.5 Tracking Out-of-Order Task

Modern software object/component technology allows for considerable task flow freedom. The technology establishes robust external interfaces between components, which minimizes the inter-component development dependencies. In many cases there was a one-to-one correlation between software components and development tasks. Theoretically, if the interfaces are well defined, tasks can be developed in any order. The developers understood this situation. The result was that the developers, as they learned more and more about their set of tasks, adjusted the order in which the task were completed.

The Critical Chain methodology and ProChain^{TM} expects task to be done in a specific order. In some cases, the reordering of tasks caused ProChain^{TM} to become confused and incorrectly calculate the buffer consumption. An example of this situation is shown below.



Microsoft Project - Alpha 18 2.mpp
Page Setup...
Print...
Close
Help
Expected Project Completion Date
No Buffer Consumption ?????
Tasks Completed Out of Order
Project Buffer
Page: 1 of 1

Figure 30: Tasks Out of Order – Buffer Effect

The snapshot of a Microsoft Project Screen shows tasks that were completed out of order, tasks that have slipped and zero buffer consumption. Manual calculations indicated that the project buffer should have been more than 60% consumed at this point in time. This error (bug) caused project management, in some cases, to not update the status of out of order tasks until they were scheduled to be complete. This situation, which occurred infrequently, decreased the robustness and accuracy of the buffers.

5.2.2 Staged Development

5.2.2.1 Rework

The tight staged development process that was used made it difficult to schedule rework as it was discovered. In general, rework was discovered after the development teams were allocated to the next development stage. Most of the rework was discovered during design reviews, validation testing and Lead Users usability testing.

The initial process that was adopted to handle rework was to wait until the current stage development is complete and then plan the re-work in the next stage development cycle. This process extends the time between rework discovery and rework correction. This rework correction lag slows the process of discovering additional undiscovered rework. Undiscovered rework results in inaccurate project status reports.

6 Conclusion

"How does a project get to be a year late?........One day at a time!" (Brooks, 1995)

"The easiest way to shorten development cycles is to minimize the work required to develop the product." (Smith, 1998)

" The importance of the critical path is not its ability to be calculated by project management software and portrayed on a schedule network. Instead, its real value stems directly from its definition: any activity on the critical path that slips by one day will directly cause the project end date to slip by one day"
(Smith, 1998)

6.1 Conclusion

One of the greatest dangers in software development is the ease at which a development task can become lost in the process. The lack of tangible artifacts makes it difficult to measure the level of completeness or the necessity of a specific task. It is common for a developer to estimate that they are on schedule all the way up to the schedule completion date and then, and only then, realize that they are not going to hit the scheduled date. It is also common for a developer to implement features and functions that are not critical to the successful completion of the project.

Applying Critical Chain in a Stage Development program helps to mitigate the risk of tasks getting lost in the process. Critical Chain provides an easy method of scheduling and tracking development progress. Having an easy and efficient process to schedule and track the program helps to make the team "schedule aware" from the start of the project. Schedule awareness is required to establish the direct connection between hitting early task completion dates and the success of the complete program. A day lost early in the program is equal to a day lost at the end a development program.

Setting aggressive task dates, which are protected by the project buffer, is an effective means of attacking pitfalls in the scheduling process. The aggressive dates serve as an efficient antidote to Parkinson's Law (Newbold, 1998). Parkinson's Law states that work expands to fill the time available. The time available for an aggressive task date does not allow for significant task expansion. Aggressive dates also attack the common human characteristics know as the Student Syndrome. Setting aggressive dates force developers to start now if they want to finish on schedule.

Stage development also aids in expressing lost tasks. The artifact at the end of each development stage is working code. Working code is a physical artifact that can be reviewed, tested and delivered (prototype) to the user community. The amount of working code is the most accurate yardsticks in which to measure the amount of work completed. Releasing working code at the end of every stage closes the time gap between the development phase and the validation phase. Early discovery of rework is a critical factor in understanding the level of work competed.

Research indicates that "rework is the single most important source of project cost and schedule crises" (Copper, 1993). Staged development works to surface the level of rework as soon as possible. The key parameters in the rework cycle are the quality of work and the rework discovery time. Stage development address the quality of work by allowing for rapid learning and address the rework discovery time issues by producing artifacts that can be review early in the process. Undiscovered rework is often incorrectly perceived to be work completed.

At a higher level, the development plans that resulted from the Critical Chain and Staged Development analysis turned out to be a robust plan. The plan allowed the team to visualize the tasks required to complete the program and to understand when a task was "out of plan". The identification and management of out of plan tasks (feature creep) is critical to the success of a project. The staged plan made it very easy to identify the out of plan work and to access the schedule impact to the requested feature creep.

6.2 Areas of Further Research and Development

6.2.1 Estimating Aggressive Dates

The principle of developing a schedule paced by aggressive dates that are protected by buffers is solid. An effective method to estimate the aggressive dates (task duration) has not been developed. The 50% date rule that is recommended by Goldratt (Goldratt, 1997) is an over simplification of a complex issue. The Goldratt 50% rule ignores the level of uncertainty in each task. Consider two task estimates: Estimate 1 – 10 man months with a possible range of 9 – 12 mm and estimate 2 – 10 man months with a possible range of 6 – 18 mm. In estimate 1, the aggressive duration is 9 man months and a buffer of 3 man months is more than sufficient. In estimate 2, the aggressive duration is 4 man months and a buffer of at least 12 man months should be allotted. The simplified 50% would have schedule both task as 5 months tasks with 5 months of buffer.

A simple and effective method for including the level of uncertainty into the aggressive duration will increase the developers buy-in, the robustness and predictability of the Critical Chain schedule.

6.2.2 Scheduling of Object/Component Based Development Projects

One of the benefits of Object Orientated Software Development is it reduces the level of coupling between design tasks in a project. De-coupling the tasks reduces the task flow constrains that exist in a traditional project schedule. Object Orientated design methodology allows the software developers to "stub out" the external interfaces to other components without completing the code that is hidden inside of the object. Once the external interfaces are stubbed out the order in which objects are developed is flexible. This flexibility allows for the development team to adjust the order in which the objects are developed.

Critical Chain is not congruous to projects that adjust their task flow frequently. Tasks that are done out of order, especially task paths are done out of order, make it more difficult to quickly determine the status of the project. In Critical Chain, the feeding buffer of a series of task is consumed if a task is not done when it is scheduled even though downstream tasks have been completed.

Traditional project planning/scheduling must be made to allow development team to realize the full potential of Object Orientated Software design.

7 References

Bear, Tony " The Culture of Components" *Application Development Trends*, September 1998, Volume 5, Number 9, pp. 28-32.

Brooks, Frederick, *The Mythical Man-Month*, Addison Wesley Longman, Inc., Reading, Massachusetts, 1995.

Cooper, Kenneth and Mullen, Thomas "Swords and Plowshares: The Rework Cycles of Defense and Commercial Software Development Projects" *American Programmer*, May 1993

Cusumano, Michael, "How Microsoft Makes Large Teams Work Like Small Teams", *Sloan Management Review*, Fall 1997

Cusumano, Michael and Selby, Richard, "How Microsoft Builds Software", *Communications Of The ACM*, June 1997, Vol 40. No. 6

Cusumano, Michael and Selby, Richard, *Microsoft Secrets*, The Free Press, New York, NY, 1995

Dolgicer, Max "MTS: Fast Train Coming?" *Application Development Trends*, August 1998, Volume 5, Number 8, pp. 24-32.

Goldratt, Eliyahu *The Goal*. The North River Press, Great Barrington, MA. 1992

Goldratt, Eliyahu *Critical Chain*. The North River Press, Great Barrington, MA. 1997

Ianaiti, Marco and MacCormack, Alan "Developing Products on Internet Time", *Harvard Business Review*, September-October 1997

Lynch, Bill "Critical Chain Project Management", Creative Technology Labs, http://www.prochain.com/pmifinal.htm, 1998

McCarthy, Jim *Dynamics of Software Development*. Microsoft Press, Redmond, Washington, 1995

McConnell, Steve *Rapid Development*. Microsoft Press, Redmond, Washington, 1996

McConnell, Steve *Software Project: Survival Guide*. Microsoft Press, Redmond, Washington, 1998

McConnell, Steve *Code Complete*. Microsoft Press, Redmond, Washington, 1993

Microsoft, "Microsoft Solutions Framework", http://www.microsoft.com/solutionsframework, 1998

Newbold, Robert *Project Management in the Fast Lane: Applying the Theory of Constraints.* St. Lucie Press/APICS series on constraints, 1998

ProChain™ Project Scheduling, *User's Guide Version 2.0 : Critical Chain Concepts.* Creative Technology Labs, LLC. Lake Ridge, VA, 1998

Rogerson, Dale *Inside COM, Microsoft's Component Object Model.* Microsoft Press, Redmond, Washington. 1997

Royce, Walker. *Software Project management: A Unified Framework.* Addison-Wesley, Reading, Massachusetts, 1998

Smith, Preston. *Developing Products in Half the Time.* John Wiley & Sons, Inc, New York, 1998

Standish Group "Chaos" http://www.standishgroup.com/chaos.html