Implementation of the Critical Chain Project Management Methodology in IBM's S/390 Software Development Environment

by

Theodore R. Maeurer

M.S., B.S., Computer Science
Rensselaer Polytechnic Institute, Troy (1989, 1988)

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
December, 1998
[February, 1999]

Signature of Author _____
System Design and Management Program
December, 1998

Certified by _____
Joyce M. Warmkessel
Lecturer, Aeronautics and Astronautics
Thesis Supervisor

Accepted by _____
Thomas L. Magnanti
Co-director, System Design and Management Program

This page intentionally left blank.

Implementation of the Critical Chain Project Management Methodology in IBM's S/390 Software Development Environment

by

Theodore R. Maeurer

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

Software Development projects have a long history of being notoriously difficult to manage. From early experiences with the IBM OS/360 Operating System over 20 years ago to more recent experiences with the IBM OS/390 Operating System, the Project Management challenges remain. This phenomenon exists despite the wide spread availability of well-developed Project Management techniques such as the Critical Path Method (CPM) and the Project Evaluation and Review Technique (PERT). The challenges also transcend the introduction of ever more powerful programming tools and techniques such as structured programming, high-level languages, source-level debuggers, and object-oriented programming.

As in many industries, a key challenge in the Software industry is the reliable delivery of products in an environment of ever decreasing product cycle times. Recent work by Eliyahu M. Goldratt suggests that the struggle with on-time delivery may well lie with the underlying Project Management techniques that have become so widely accepted. These techniques foster behavior patterns that are counter-productive to the shortening of product cycle times. They fail to focus the organization on the Project Management system at large and can encourage dysfunctional decision making [1]. Work in the field of System Dynamics has independently reached similar conclusions. The traditional Project Management techniques offer little to help the Project Manager cope with issues at the strategic level. Without strategic guidance, the Project Manager is left to make poor, informal judgements and may not make adequate allowances for factors that negatively impact project performance [2].

Goldratt offers a new, alternative project scheduling approach called Critical Chain as a mechanism for improving an organization's underlying Project Management structure. Critical Chain is based on principles developed a decade earlier in Goldratt's Theory of Constraints. The Theory of Constraints changed the way organizations think about Manufacturing processes. Likewise, Critical Chain requires that organizations reformulate their approach to managing development projects.

This thesis will study the successful results of applying Critical Chain on two actual Software Development projects in IBM's System 390 Division. Each of these projects achieved commitments on time. Critical Chain's contribution to these results will be discussed. The experiences gained along with potential pitfalls of Critical Chain will also be considered. In particular, the issues involved with applying this approach to a Software Development environment in which traditional methods are in widespread use will be emphasized. A discussion of the potential limitations of the Critical Chain approach will also be provided.

Thesis Supervisor: Joyce M. Warmkessel

Acknowledgements

I would like to thank those individuals in the IBM S/390 Software Development organization who participated in the case studies of this thesis, as well as those who provided their hearty support for this experiment in project scheduling. Without their enthusiastic support and participation, this thesis would not have been possible. My hat's off to your willingness to experiment.

I am especially grateful to my thesis advisor, Joyce Warmkessel, for her enthusiastic support for this work. Her thoughtful advice and ability to motivate was instrumental to my completion of this thesis. In the classroom, her ability to combine theory and practice was pivotal to more meaningful learning and helped provide the inspiration for this thesis.

I would also like to give special heartfelt thanks to my wife, Lora. Without her support, encouragement, and patience, my completion of this thesis as well as the System Design and Management program as a whole would not have been possible. I look forward to returning to a "normal" life and spending our future together.

Table of Contents

List of Figures

Chapter 1

Thesis Introduction

1.1 Motivation

The landscape of business today is characterized by constant price pressures and ever decreasing product cycle times. Many top companies in terms of sales growth and profitability obtain almost half of their revenues from products developed within the past five years. Competition is now based on not only cost, but time to market as well. In such an environment the need for effective Project Management is key to the financial success of the firm. Ineffective management may both jeopardize the ability of the firm to capture desired market share and burden it with a non-competitive cost structure. With more effective management of development projects, profits and stockholder value could be doubled [3] [4].

In the past, such competitive pressures have been addressed through diligent cost reduction programs. However, many firms have pushed this to the limit; additional cost reduction actions have the potential of damaging the firm's operational engine. There is little option but to better utilize existing resources for the generation of cash flows and profitability [5]. For this reason, the issue of effective Project Management has become of particular interest in recent years.

Despite this fervor for Project Management, the results are less than satisfying. Most projects do not meet their time to market and cost goals (Cooper, 1994). The problem is particularly acute in computer hardware and software firms where on-time delivery seems to be the exception to the rule. Furthermore, a broad base of Project Management experience does not appear to reduce the troubles. Even firms that are large and experienced with Project Management techniques such as PERT and CPM continue to have difficulties. Such techniques may be a poor fit for the task at hand; often, they are at their best when the

project involves repetitive tasks for which much history is available. However, few development projects today are afforded such luxury [4][5].

A basic reason for Project Management difficulties may be that Companies fail to see their processes and management policies as systems in the same sense as aircraft or data processing systems. As a result, organizations are often built through use of gut feel, committee, or even chance. The results are commensurate with technical systems built using the same approach. This problem is compounded by the fact that such managerial systems are often far more complex than Engineering systems [6]. Clearly a better means to improve our organizations is needed. The most successful corporations in the future will be those who can effectively learn. The ability to quickly channel learning into improved organizational capability may well be the last source of sustainable competitive advantage [7][8].

The Goldratt Theory of Constraints and the field of System Dynamics represent a growing body of work in which the study of organizations and their associated management systems is performed from a systems perspective in which underlying structures are analyzed. This mode of study utilizes approaches more traditionally associated with Engineering systems. However, by focusing on underlying structure, knowledge can be transferred across disciplines, such as Engineering and management, resulting in a more prolific learning environment.

Each of these approaches has revealed that the very mechanisms used to ensure project success may in fact be contributing to the inconsistent and often unacceptable results being experienced with their performance. The Theory of Constraints has shown that while typical Project Management approaches seek to ensure the timely completion of each step, the overall project may be at risk. For example, the benefits of scheduled protection time may be wasted due to the effects of such behavioral patterns as Parkinson's law (work expands to fill the available time) [1]. System Dynamics relates that poor,

informal strategic judgement is a key contributor to project failure. For example, estimates of productivity levels are often ill conceived, resulting in inaccurate schedules [2].

Through applying the Theory of Constraints to project planning, Goldratt offers an alternative activity scheduling approach called Critical Chain. The Critical Chain approach is based on the study of Project Management in the context of an overall system [9]. The potential benefits that are offered by this new approach, in conjunction with the increased importance of effective Project Management for future corporate success, served as motivation for this thesis.

1.2 Problem Statement

Evidence, provided by both work in the field of System Dynamics as well as that by Goldratt in the Theory of Constraints, suggests that disciplined use of traditional Project Management techniques can result in behaviors that are counterproductive to the goal of shorter product development cycles and reliable project completion dates [1][2]. If this is the case, do the alternatives offered by the Critical Chain methodology offer a more viable alternative? What are the issues involved with integrating Critical Chain methodologies into an organization in which traditional techniques are currently deployed? Software Development across the industry has been riddled with late product deliveries and cost overruns, despite the growing number of attempts to bring rigor to Software Engineering. The success of software projects is becoming more of a management issue than a technical one [15]. To what extent can Critical Chain provide a solution?

1.3 Goals and Objectives

This thesis will attempt to ascertain the viability of the Critical Chain approach for improving project performance in a Software Development context. This will be performed through trial of the Critical

Chain approach on two Software Development projects in IBM's S/390[1] Software Development organization. Specifically, the following will be done:

- The general problems of traditional project scheduling techniques will be discussed as well as the potential role of Critical Chain in alleviating these problems.
- Critical success factors to the application of project scheduling techniques will be discussed in the context of a Software Development organization.
- Issues involved with applying the Critical Chain approach in the S/390 Software Development organization will be examined.
- Two case studies will be conducted in which Critical Chain was applied to a Software Development project. The results of these case studies versus the claims of Critical Chain will be discussed.

1.4 Assumptions

The following assumptions are made which limit the scope of this thesis, but provide opportunities for future work:

- Project Management is a broad discipline. The case studies conducted in this thesis are focused on project scheduling issues. Clearly, there are project performance improvements that can be achieved through other means, such as improvements to the Software Development Process. However, it is the existing S/390 Software Development Process that will be considered within the scope of this thesis.
- Critical Chain is but one application of the Theory of Constraints for the improvement of project performance. The Theory of Constraints can be used as part of a larger overall continuous improvement program. This is beyond the scope of this thesis, but the interested reader is encouraged to consider this possibility.

[1] IBM and S/390 are registered trademarks of the International Business Machines Corporation.

- There are many factors that can contribute to the level of success of a project in addition to project scheduling methodologies. The case studies will consider to what extent additional factors may have contributed to the performance of the projects. However, this is not the primary objective of the case studies.

1.5 Overview of Remaining Chapters

- **Chapter 2** describes the Goldratt Theory of Constraints as well as the Critical Chain project scheduling approach.
- **Chapter 3** provides a synopsis of traditional scheduling techniques. These are techniques that are commonly accepted and in widespread use today.
- **Chapter 4** looks at the shortcomings of the traditional project scheduling methodologies and the resulting patterns of team behavior. The potential role of Critical Chain in eliminating these problems will be considered as well.
- **Chapter 5** discusses the S/390 Software Development environment including the commonly used project scheduling techniques.
- **Chapter 6** provides a preliminary look at the potential issues involved with applying Critical Chain in the S/390 Software Development organization. In particular, the factors contributing to the success and failure of applying traditional scheduling approaches within the organization will be studied. The emphasis here is to provide a solid foundation for the basis of the Critical Chain case studies.
- **Chapter 7** describes the case study experiences in applying Critical Chain to two Software Development projects. The key benefits of Critical Chain as well as potential limitations and pitfalls will be discussed. In particular, these case studies will be used to assess the 'claims' of Critical Chain.
- **Chapter 8** summarizes the overall findings of this thesis and lists some potential topics that merit further investigation.

Chapter 2

The Theory of Constraints and Critical Chain

2.1 Goldratt's Theory of Constraints

Goldratt describes systems as chains, or networks of chains. Within a chain there is one and only one weakest link at any given time. This weakest link is the constraint of the system. If that weakest link is strengthened, then the constraint of the system may move to another link. Still, there is one and only one weakest link in the system. In a Manufacturing environment, the links of the chain can be likened to the various machine stations.

In a Manufacturing environment, the Theory of Constraints shows that not more than one station will be operating at its full potential if the system is to be performing as well as possible. Running every machine at its maximum rate can result in increased work in process inventory depending on the location of the constraining machine in the production sequence. The result is increased lead time and wasted resources.

Thus, performance improvements of the system at large, in terms of its goals, cannot be achieved through local optima. The drive to achieve local optima has been coined by Goldratt as the cost accounting mentality. The cost accounting mentality strives for high efficiencies or utilization; it rewards busyness.

Goldratt defines global system performance in terms of *throughput*. Throughput is the rate at which the system generates money through sales of products and services. This definition is sufficient for profit seeking businesses, although other definitions are possible. Thus, the *goal* is increased throughput whether that means units sold or projects completed. This definition is sensitive to the fact that producing

to inventory is not an acceptable situation. In such a case, Marketing could be the constraint of the system. Goldratt prescribes a five-step process for global systemic performance improvement:

1. *Identify the System Constraint:* The system constraint is the weakest link. It is also known as the bottleneck.
2. *Exploit the Constraint:* In exploiting the constraint, we strive to achieve maximum performance from it. For example, in the case of a Manufacturing environment, ensure that the bottleneck machine does not remain idle over lunch.
3. *Subordinate everything to the constraint:* The rest of the system must perform at a rate that ensures that the constraint can maintain maximum performance. For example, in a Manufacturing environment, it makes little sense to run other machines at a rate faster than the bottleneck machine, since this builds up excess work in process inventory. However, protective buffers should be employed so that the bottleneck machine does not starve should a non-bottleneck machine become temporarily unavailable.
4. *Elevate the constraint:* Should the performance of the system still be unacceptable, the next step is to improve the performance (or elevate) the constraint. In a Manufacturing environment this could involve adding a machine, or replacing the bottleneck machine with one that has better performance.
5. *Go back to step 1:* Once the constraint has been elevated, it may no longer be the constraint of the system. The cycle must restart at step 1 to identify the new bottleneck.

The result of this cycle is an ongoing improvement process that focuses on achieving the greatest benefits from a surgical application of scarce resources. These improvements must be performed in a manner that minimizes inventory and operating expenses to ensure that throughput improvements do not come solely at the expense of these other elements. However, the primary focus should remain on solid throughput improvements as opposed to cost cutting; cost cutting has clear limits before it begins to degrade throughput.

Goldratt has described the application of the Theory of Constraints to a number of disciplines. His book, The Goal, describes the application of the Theory of Constraints to Manufacturing, while It's Not Luck describes the applications to Marketing. Critical Chain, describes its application to Project Management and will be the subject of this thesis. It is worth noting that Goldratt also describes a general problems solving approach that can be used to identify root problems and system constraints, which the interested reader can further pursue [10][11][12].

2.2 Goldratt's Critical Chain

Critical Chain represents the application of the Theory of Constraints to the Project Management discipline. A first step to analyzing the Project Management process is to determine the goal of a project-oriented organization. By and large, the goal of a project-oriented organization is to increase throughput where throughput can be defined as sales minus variable costs (such as materials). Throughput can be increased primarily through two mechanisms: increase the number of projects completed or increase the reliability of project completion dates. More reliable project completion dates can increase throughput, since price premiums may be afforded and/or additional market share captured when this capability exists.

Next, the constraint of the project must be identified (*step 1*). The Critical Path of the project network can *initially* be considered as the element of the project that determines the maximum performance in relation to the goal. The Critical Path is the bottleneck of the project in much the same way as a bottleneck machine in a Manufacturing environment. By definition, the Critical Path is the longest sequence of tasks that determines the shortest possible project completion time. If the performance of the Critical Path is improved, the performance of the overall system – the project – is improved. Note that the use of a Manufacturing analogy is not meant to imply that a project is equivalent to a Manufacturing line.

However, it does imply that there are structural similarities between these two systems and certain key pieces of learning can be transferred between them.

Traditional definitions of the Critical Path neglect the issue of resource contention; in essence, the Critical Path assumes infinite capacity in resources. This assumption of infinite resources may distract the Project Manager from the true constraint of the project. A schedule with resource contention de-conflicted may look quite different. It is here that the notion of a Critical Chain is introduced. The Critical Chain is the longest sequence of tasks that determine the shortest possible project completion time based on a schedule with resource contention de-conflicted. The Critical Chain is *the* constraint of the project schedule. This implies that not all tasks, resources, and elements of the project can be equally crucial at the same time. Projects in which 'fires' seem to be 'burning' far and wide very likely suffer from a lack of focus and understanding of the key pieces of work that are driving the project completion. Identifying and understanding the Critical Chain, and the associated planning and/or design work required to gain that understanding, is therefore an important contributor to project success.

Once the constraint has been identified, it must be exploited (*step 2*). This implies that the Critical Chain tasks are engaged with complete focus and minimal waste. To achieve this, the key contributors of waste must be identified. Goldratt identifies three key contributors that conspire together to cause waste:

1. The practice of spreading contingency across all the tasks in a projects.
2. Student syndrome and Parkinson's Law.
3. Multi-tasking.

Parkinson's Law postulates that as a worker discovers that a task is not as time consuming as originally thought, the worker will slow down; work expands to fill the available time. This effect can result in the accumulation of negative variance; when things go better than expected the project doesn't benefit, but

when things go worse than expected the project will be delayed. The spread of contingency across all tasks can make matters even worse. Contingency times that are intended to protect the project can be slowly lost, being unavailable when they're really needed. Student syndrome, which is the tendency of people to wait to the last minute, can exacerbate the problem even further. Finally, multi-tasking can delay the completion of Critical Chain tasks and thereby delay completion dates.

Goldratt's solution to these problems involves a basic 're-design' of the way people normally think about projects. Contingency should be removed from all task duration estimates; instead, average (or 50% probable) task duration should be used. When such a task duration is used, it should be expected that at least half of all tasks will be late. If people are to be comfortable providing these 50-50 dates, conformance to project milestones cannot be used as an employee or project evaluation criteria. The primary emphasis is for people working on Critical Chain tasks to work as hard as possible to achieve the scheduled average task duration; being late is ok if best efforts were made to avoid it. Those working on *non*-Critical Chain tasks work as hard as possible to ensure that Critical Chain tasks are not delayed as a result of their work and support those on Critical Chain tasks as much as possible. However, some project evaluation mechanism is invariably required. Project buffers and buffer management will serve this purpose.

Once the constraint (Critical Chain) has been elevated, the rest of the project must be subordinated to it (*step 3*). Work on non-critical tasks must not delay critical tasks, for this will delay the project. However, the use of 50% dates virtually ensures that chains of non-critical tasks will in fact delay critical tasks. In a Manufacturing environment, buffers of work in process (WIP) inventory are used to protect the bottleneck operation. Similarly, in a project, buffers will also be used to protect the Critical Chain of tasks from chains of non-critical tasks that feed into it. However, in the case of a project, the buffer will be one of time. This really is not so different from the Manufacturing analogy in that a WIP buffer can be expressed equivalently in terms of time.

The protective buffers will be inserted at the intersection of non-Critical Chains of tasks with the project Critical Chain. This is called a feeder buffer. Thus, the Critical Chain is protected from unforeseen difficulties that may be experienced on a non-critical task. Furthermore, should work on the Critical Chain proceed more quickly than expected, the feeder buffer will also allow a Critical Chain task to start early; there is a lower chance that it will be delayed by a predecessor task. A buffer is also placed at the end of the project to protect the end date; this improves the reliability of the overall end date.

The protective buffers are *not* slack time. They are an integral part of schedule given that task times are estimated using 50% dates. In addition, estimates of duration are never perfect. Without the protective buffers, the final project delivery commitment *cannot* reliably be achieved.

Various mathematical models can be used to estimate the required buffer sizes. However, given the factor of error in most estimates, such an accurate buffer calculation is probably not worth the effort. A rule of thumb is to double usual project productivity rates and define buffers to be 50% the duration of the chain of tasks that they are protecting. However, the specifics of the project at hand and prior experience should be considered as well.

If the project schedule that results from the preceding steps is unacceptable, the next step is to elevate the constraint (*step 4*). By elevating the constraint, its performance against the objective is improved. In a project environment, restructuring work, assigning additional resources, or making appropriate process improvements can achieve this. Once the constraint is elevated, the schedule planning must restart at step 1, since the constraint may have moved [1][9].

In subsequent chapters of this thesis, the benefits of this Critical Chain approach and its divergence from traditional Project Management policies will be considered further. An important consideration at this

point is that Critical Chain is only one application of the Theory of Constraints to the project environment. Even if Critical Chain is employed as the project scheduling mechanism, the Theory of Constraint philosophy can be still be employed as part of general continuous improvement program.

Chapter 3

Traditional Scheduling Techniques

3.1 Introduction

This chapter will provide a survey of traditional scheduling techniques. The use of the term 'traditional' in this context is meant to include those scheduling techniques that are commonly accepted or in widespread use. It is scheduling that is a major focus and distinguishing feature of Project Management. Its purpose is to ensure that project objectives are achieved within a desired time period. Project scheduling is distinct from the scheduling that takes place in a Manufacturing environment, such as a job shop, in that many of the tasks that comprise the project are unique.

The project scheduling process involves assigning time periods and resources to tasks in the work breakdown structure (WBS) within the bounds of various time, resource, technical, and task precedence constraints. The WBS is a decomposition of project work for the purposes of estimating time, cost, and resource requirements. The tasks of the WBS are related to one another in terms of sequencing or precedence requirements; the task sequencing structure forms a task network. This network serves as the basis for a number of scheduling techniques.

Before describing the various scheduling techniques, it is worth noting that scheduling is but one of the many tools of Project Management. The wide variety of scheduling software available today can create a false illusion that scheduling *is* Project Management. Other necessary elements of Project Management are defining objectives, high-level planning and organizing, allocating appropriate resources, tracking, reporting, and control.

Overemphasis on scheduling can lead to the creation of work breakdown structures and schedules that are more detailed than required; in fact, the level of detail may surpass what is actually known leading to a false sense of precision. In general, schedules should not be of a level of detail that exceeds one's ability to manage. Doing so may cause project scheduling to become more of burden than a tool [3][5][9][16][17].

With this potential pitfall in mind, the various traditional scheduling techniques will now be described.

3.2 Bar Chart Scheduling

For more than the first half of this century, Bar Charts were the only tool for scheduling projects. (Bar charts are also known as Gantt charts after their originator, Henry Gantt). A major benefit of Bar Charts is that they are simple and easy to understand. Each task is represented as a bar showing the amount of time consumed by a task. The following is a sample Bar Chart that schedules the painting of a room:

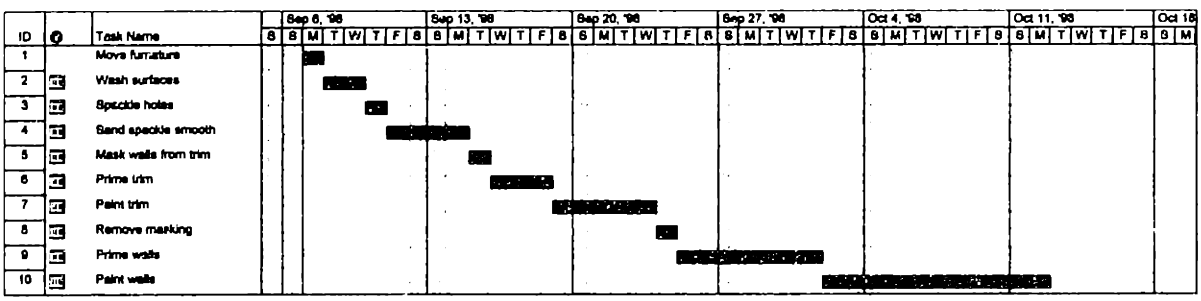


Figure 1: Sample bar chart

Note that the Bar Chart does not capture task precedence relationships. It is for this reason that the Bar Chart is *not* a network scheduling technique. This is also a significant limitation; a change in one task (such as a late finish) can be difficult to resolve in terms of the larger project schedule. As such, Bar Charts have limited power in terms of a scheduling tool.

Despite their limitations, Bar Charts remain popular as a graphical representation of network schedules. Variations of the Bar Chart have been developed to allow task precedence information to be conveyed. This is typically done by linking the various bars with arrows to represent the precedence relationship [3][16].

3.3 Network Techniques

3.3.1 The Project Network

Construction of the project network adds task predecessor and successor information to the project work breakdown structure. This requires additional information in the form of predecessors and successors of each task to be identified. This information is often conveyed graphically through activity on arrow or activity on node diagrams. (The key difference between the two being whether the task is associated with the arrow or the node.) The following is a sample activity on arrow diagram:

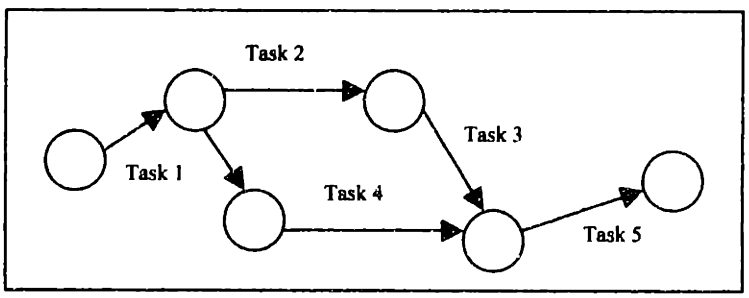


Figure 2: Activity on Arrow Diagram

With a network diagram and estimations of each task's duration in hand, a network scheduling technique can be used to determine the start and finish times for each task. Once such start and finish times are determined, the schedule is often presented in the form of an enhanced Gantt chart due to the intuitive nature of that format. [3][5][16].

3.3.2 Critical Path Method (CPM)

The primary goal of CPM is to determine the Critical Path of the task network. The Critical Path is the longest sequence of tasks that determine the shortest possible duration of the project schedule. The Critical Path is the path with the minimum slack time and whose tasks delay the project end date if they are individually delayed. In a pure CPM application, task duration is assumed to be deterministic and is often based on some likely (or normal) estimate with a contingency allowance added.

The earliest possible start time for tasks is determined by performing a forward pass through the network. Calculations begin with the start node and a project start time. Activities whose predecessor is the start node are assigned a task start time equal to the project start time. The early completion time of the task is determined by adding the task duration to the task start time. This task completion time becomes the task start time of successor tasks. These calculations ripple through the project network until the finish task is encountered. The project finish time is equal to the latest finish of predecessor tasks to the dummy finish task.

A backward pass on the network is then performed to determine the latest start and completion times for the tasks in the network. This calculation proceeds in a similar fashion as the forward pass, except that we begin with the finish node using the project finish time determined in the previous step. The calculations work their way backward through the project network until the start task is reached.

The Critical Path is determined by finding the path with the minimal slack time. The slack time for a single task or path of tasks through the network is the difference between the early and late start times. Alternatively, the Critical Path can be determined by finding the longest path in the network. These calculations need not be a burden for the Project Manager. Today's Project Management software can automatically determine task start and finish times, as well as the Critical Path.

Once the Critical Path schedule has been identified, the Project Manager can use it to manage the project. Staying on schedule serves as the goal. Particular attention is given to tasks on the Critical Path, since a delay in one of these could delay the entire project. Managers can also use the Critical Path schedule to understand the cost of speeding up certain parts of the project (known as crashing) [3][5][16][17].

3.3.3 Program Evaluation and Review Technique (PERT)

Unlike CPM, PERT recognizes that activity duration is often not deterministic and can be prone to uncertainties. PERT addresses the existence of uncertainties by using three time estimates for each task: most likely, optimistic, and pessimistic. The more uncertainty involved with a task, the greater the spread in these three estimates.

PERT assumes that the distribution of time required for an activity is expressed as a beta distribution. While the beta distribution is the most commonly used, it is also possible to use other distributions (normal, lognormal, etc.) When using the beta distribution, the mean duration of a task can be expressed as the mean of the three estimates with the mode four times as likely as the end points. The formula is as follows:

$$t_e = (a + 4m + b) / 6$$

where

a = the optimistic time estimate
m = is the most likely time estimate
b = the pessimistic time estimate
t_e = the expected time for the activity

PERT's formulas make a simplifying assumption that most task times will lie within plus or minus three standard deviations. There is no theoretical support for this. However, it does simplify the formula for variance, which is as follows:

$$s^2 = (b-a)^2/36$$

where

s^2 = variance of the task duration

PERT invokes the central limit theorem to dictate that the duration of the entire project be normally distributed. In the case of a project network, use of the central limit theorem requires that the duration of all tasks be independent. This may not always be a reasonable assumption. However, it further simplifies the calculations. The total expected project duration (T_e) is given by the sum of the duration of tasks on the Critical Path. Total project variance (S^2) is given by the sum of the variances of all tasks on the Critical Path. Thus a Critical Path must be calculated as in CPM, only expected task times t_e are used for the duration of tasks.

Given this use of the central limit theorem, the probability of meeting a particular deadline is given by the following formula:

$$z = (T_d - T_e)/S$$

where

T_d = the deadline
T_e = expected project duration
S = standard deviation of project duration

The probability of the project completing on or *before* the deadline is therefore given by the following formula:

$$P(T \leqslant T_d) = P(z \leqslant (T_d - T_e)/S)$$

Given this formula, the probability is determined by a lookup in a standard normal table. The application of PERT would typically follow a seven-step process where the above formulas are put to use:

1. Determine the list of project activities.
2. Determine the precedence of the activities.
3. Review the data to ensure consensus.
4. Create a PERT chart and use the expected duration of tasks (t_e).
5. Calculate the Critical Path using expected task duration (t_e). The length of the Critical Path will be given by T_e. An analysis of the Critical Path can proceed as in CPM.
6. Level the schedule, based on resource limitations. This step is also applicable to CPM.
7. Calculate the completion probabilities. Schedule duration versus risk can be assessed in this step. This step is not an available option in CPM.

The key distinguishing factor of PERT as compared to CPM is the use of probabilistic task duration. Aside from this, much of the analysis is the same. The additional ability to calculate the probability of project completion as of a certain time allows the Project Manager to assess how realistic the schedule is. In this sense, it represents an improvement over CPM. However, the additional overhead of obtaining three duration estimates per task may exceed the potential benefits for some projects [5][16][17].

3.3.4 Graphical Evaluation and Review Technique (GERT)

The Graphical Evaluation and Review Technique (GERT) extends CPM and PERT through the addition of probabilistic branching from nodes (or tasks) in the network. Arcs in the network may represent other variables such as cost in addition to time (which is a requirement of PERT and CPM). The structure of the GERT network allows looping back to earlier tasks. In addition, a variety of probability distribution functions may be chosen.

While GERT is difficult to use as a control tool, the GERT network allows a project simulation to be performed to allow the potential outcomes of the project to be explored. This additional capability over PERT requires additional information to be collected on the part of the Project Manager. The various

outcomes of a task and their associated probabilities must be considered. For example, if a task has a 75% probability of success and a 25% probability of failure, the resulting outcomes and associated task flows can be considered as part of the project planning.

GERT provides a more dynamic view of the project than does PERT. However, the additional data collection and computational requirements may not be justified for all projects. This barrier is rapidly diminishing, as more powerful simulation software becomes available [17].

Chapter 4

Critical Chain and the Shortcomings of Traditional Scheduling Techniques

4.1 Overview

Across a number of industries, the use of the traditional scheduling techniques (discussed in the previous chapter) has proven inadequate to ensure the timely completion of projects on a consistent basis. This is particularly true with respect to Engineering endeavors such as Software Development [4]. Some, but not all, of these difficulties can be blamed on the Project Management techniques themselves; clearly, other elements may be contributing factors, such as consistency and rigor of Project Management application, product and process architecture, and organizational issues. This chapter will discuss common limitations and problems with traditional scheduling methods and whether the Critical Chain approach can help.

4.2 Estimation Problems

It appears to be an unfortunate truth that human beings exhibit poor performance in terms of estimating ability. The confidence people exhibit in terms of their estimates of an unknown quantity is unrealistically high. This overconfidence leads to a high fraction of incorrect estimates. In terms of projects, estimates of task duration tend to be optimistically skewed. Regrettably, for software development, available estimation models are usually unable to estimate cost with even a modest level of precision [15][17][19]

Optimistically skewed estimates even occur in the 3 part estimates (worst, best, most-likely) employed in the PERT approach. To make matters worse, many projects are not of a repetitive nature and therefore lack the necessary history on which to base more reliable cost estimates. In addition, the notion of

random error also plays a role; task completion times are not necessarily deterministic, but may exhibit a probabilistic distribution. It is only this random element that PERT plays a role in addressing [5][17].

These factors have contributed to irrational schedules, which have proven to be a chief cause of missed deadlines. It is all too common for schedules to be created without full regard for project complexity or staff ability [18]. Thus, a project may be doomed to be late almost as soon it starts.

A mechanism suggested in traditional Project Management texts for addressing this problem involves techniques for improving the estimating ability of the team. These include systems for learning from past mistakes more effectively as well as systems for obtaining experiences necessary to make good judgements without necessarily living through actual product cycles [3][17]. These approaches clearly have merit in terms of their ability to reduce estimation error. However, these approaches do not completely solve the problem. There will always be inaccuracies in estimating in the context of an *Engineering* project environment.

Other mechanisms suggest increasing task times with a contingency allowance or placing a contingency task at the end of the project. Some of the more responsible Project Management texts note the danger inherent in this approach; the practice of adding contingency time to the schedule raises the danger or time being lost through Parkinson's Law [17].

PERT enthusiasts argue that the use of the 3 duration estimates can help alleviate the problem. Given the availability of these three duration estimates, the PERT equations can be used to determine probabilities of project completion for different end dates. Thus, the management team can reach a decision on a reasonable project end date, some contingency can be added if necessary, and the Project Manager should be held accountable for project performance [17].

The author questions the merit of this approach. In a project environment in which historically grounded estimates are unavailable, the generation of three estimates for use in the PERT approach is not likely to improve matters. If a single most-likely (or average) estimate is so elusive, the generation of three estimates seems even more unreasonable. It may even be dangerous, since it can add the illusion of accuracy. Thus, we appear to be left with no effective mechanism for dealing with this problem.

The Critical Chain method offers a systematic approach to dealing with these problems. It extends the suggested approaches noted within the context of traditional techniques with a new Project Management policy designed to improve overall project performance. It also challenges some of the basic assumptions of traditional Project Management.

Chief among these challenges is that of milestone-driven Project Management itself. In milestone-driven project scheduling, each task, or group of tasks, is tagged with a completion milestone. Workers feel pressured to achieve these milestones and a positive employee performance evaluation often depends on it. However, we have noted that cost estimates are often wrong, and in general things can go wrong (Murphy's Law). A worker is left with two choices: hope for the best, or pad time estimates with contingency. Each of these can be a losing proposition [1][9].

In a schedule that uses average completion times (as CPM suggests), it is reasonable to expect that some tasks will be late, since real life projects are non-deterministic. It may be difficult to make up for this, since the accumulation of negative variance in a milestone-driven schedule is often common. When things go better than expected the project often does not benefit; there is rarely an incentive to finish early. But when things go worse then expected the project usually pays the price. Thus, the use of average completion times and hoping for the best does not seem to be a robust solution [1][9].

Padding task times with contingency may also not solve the problem. Doing so, to the extent that the completion of individual tasks is reasonably assured, can result in overall schedules that are under aggressive and insensitive to time to market issues. In addition, the specter of Parkinson's Law makes the self-fulfilling schedule a distinct possibility. Thus, a "give and take" with the Project Manager will probably ensue with some middle-of-the-road contingency used in the end. The worker is still hoping for the best [1][9].

Given this, the use of milestones appears to be counter-productive. Milestones represent a cost mentality – a focus on localized efficiency. What we need to protect is the ultimate milestone – the project delivery date. It is for this reason that Critical Chain suggests the use of 50-50 times for task duration and the use of protective buffers – the project buffer and the feeder buffer. In conjunction with this, the basic manner in which the project is managed must be changed as well [1][9].

The status of the project can no longer be expressed as a collection of task-completion time-success indicators. Rather, it is expressed in terms of buffer consumption versus progress. In addition, employee progress is not assessed relative to scheduled task-completion date. Rather, it is assessed relative to that employee's latest estimate of time to completion [1][9]. Thus, estimates are continually reassessed and the level of focus becomes more global.

These modifications are designed to put protective buffers where they can help most. These buffers are not considered slack time; given the schedule uncertainties and the use of 50-50 dates they are a necessary part of the schedule. The use of buffers is also designed to protect against Parkinson's Law. By removing the stigma of milestones, employees should become comfortable with providing 50-50 dates [1][9].

The use of the protective buffers provides a margin of protection against the uncertainty and potential skew of cost estimates. Clearly, the protective ability of the buffer has limits and to strive for more accurate estimates is still of merit. The margin of error tolerated by the Critical Chain approach remains to be seen and will be further explored via case study.

4.3 Unexpected Changes

A common complaint with respect to project scheduling is that priorities change so often that it is a waste of time to create task networks; they do not remain valid long enough to justify the investment. Some Project Management texts provide a stern scolding and point out that such problems are generally a result in inadequate planning, which results in oversights [3]. They also suggest that often 'management does not have its act together, and the organization may be trying to tackle too much work' (Lewis, p. 54). However, change can be expected even under the best of conditions. Certain behavior patterns exhibited on a project can worsen the effect of such change by widening the window of opportunity for it to occur and by exacerbating its impact. Multi-tasking is a prime example of such a behavior pattern.

Multi-tasking introduces waste in the project, since task switching generally involves some amount of overhead. Often a task switch requires additional setup work to be performed. For example, a software tester may need to configure and initialize different execution environments depending on the task at hand. Task switches can also consume mental bandwidth. Workers need to mentally prepare themselves for the change and may also spend additional time to stay on top of the latest developments and news for multiple tasks, even if only a fractional amount of work is to be performed on each of those tasks [1][3][9].

Multi-tasking may be, at least in part, an outcome of the milestone mentality. Given the overall focus on task completion times in traditional CPM, the worker will be motivated to pad estimates with contingency time to improve the probability of meeting the schedule completion date. However, such padding can

often result in a schedule that appears to be under aggressive. To avoid the possibility of getting caught with insufficient work and the negative stigma associated with it, a worker or team may be motivated to take on additional tasks. The hope is that worse than average tasks or projects will balance out with better than average tasks or projects. However, doing this has the effect of obscuring priorities, making it quite difficult to develop a historical cost database, and feeding the motivation for overestimation [9].

The Critical Chain approach provides a framework for considering these matters. In addition to potential productivity losses, multi-tasking also has the undesirable effect of increasing lead times. For example, if task A is performed in parallel with task B, then the lead time of both tasks will be pushed as far as possible to the right. Had those tasks been performed sequentially, then one of them had the potential of finishing quite a bit earlier [1][9].

Given that all non-critical tasks on the project must be subordinated to those on the Critical Chain, multi-tasking that involves Critical Chain tasks must be avoided. If multi-tasking is performed on Critical Chain tasks, the entire lead time (or duration) of the project will be lengthened. Thus, the Theory of Constraints, upon which Critical Chain is based, provides a framework that allows the negative effects of multi-tasking to be easily understood and conveyed. As a result, the Critical Chain approach makes the generally recommendation that multi-tasking should be avoided whenever possible [1][9].

A reduction in multi-tasking and the potential shortening of project lead-time can reduce the window of opportunity during which changes can be introduced. For example, a shorter lead-time reduces the opportunity for a customer to change their mind or a manager to re-prioritize. A shorter lead-time may therefore result in fewer unexpected changes in general [1][9].

While it may be possible to reduce lead times through careful planning, the occurrence of an unexpected change is nevertheless a virtual certainty. There are a number of reasons for this. There is a possibility

that all technical parameters were not completely understood at the beginning of the project. As the project progresses, these issues may become better understood and result in necessary project changes. There is also the possibility that the customer may climb the curve of understanding along with the project team and may therefore find cause to modify requirements [17].

The Critical Chain approach is more tolerant of such changes. Given that a Critical Chain schedule is buffer-managed rather than milestone-managed, fewer schedule changes will be required as tasks meet or exceed their originally estimated start and completion times. Perhaps, more importantly, the Critical Chain buffer itself is designed to absorb such changes without jeopardizing the project completion date. Thus, the Critical Chain approach accepts uncertainty as a given and uses the strategic placement of buffers both as a tool for protection and a tool for measurement of progress [1][9].

4.4 Resource Allocation Problems

Traditional scheduling methods such as CPM and PERT do not consider resource availability and capacity when making scheduling decisions. Rather, the focus is on the time variable. Ignoring resource capacity can result in schedules that are unachievable, since there may be times during which resources are overbooked.

Some Project Management texts and scheduling software packages recognize this problem. The solution involves the application of a load-leveling algorithm. These algorithms analyze resource usage in the CPM or PERT schedule. Resources are assigned to tasks according to a priority rule. These priority rules are often adaptations of job shop scheduling algorithms. During periods of overbooking, tasks can be slowed or delayed until the required resources are available [17].

The Critical Chain approach also addresses this issue. In fact, this is the motivating factor behind the use of the name "Critical Chain." After overbooking of resources has been resolved through load-leveling, the

Critical Path may no longer be the constraint of the project. That is, the Critical Path may no longer represent the longest sequence of tasks that determines the shortest possible duration of the project. The new constraint is the Critical Chain.

In Critical Chain scheduling, all tasks are scheduled as late as possible (which is in contrast to the early-as-possible scheduling typically performed by Project Management software). Tasks are scheduled with zero forward slack time. They are not started earlier, because this is considered unnecessary. This philosophy is intended to decrease work in process (WIP), since tasks are not skewed to the left causing a bulk task start. This is consistent with the goal of using resources only when they are needed and reducing the potential amount of rework (having to do things over) should design problems be discovered. In addition, this approach has the added benefit of delaying investment for as long as possible [9]. Scheduling when needed and no sooner also seems likely to reduce the temptation of multi-tasking and can be a hedge against work silently expanding into the positive slack time. Both of these issues would lead to less efficient use of resources.

Given this scheduling decision, no task can be pushed into the future without impacting the non-buffered completion date. The set of tasks that has no room to be pushed into the past *or* into the future defines the project's Critical Chain. The Critical Chain is therefore the constraint of the project; it represents the chain of tasks that determine the shortest possible completion time for the project [9].

The author notes that the Critical Chain identified by Critical Chain specific Project Management software, such as ProChain, is quite similar to the critical tasks identified by resource leveling performed in traditional scheduling software such as CA SuperProject. In examples examined by the author, both packages identified what amounted to a "Critical Chain." Above this, it appears that the Critical Chain approach offers a better framework for considering these resource issues.

4.5 Dynamic Environment

Most scheduling approaches, with the possible exception of GERT, present a static view of the project. However, the development process of many high technology endeavors, such as software, tends to be more iterative in nature. In such cases, neither the design of the product nor the task network defined to achieve the creation of that project can be fully understood at the beginning. As effort is expended on the project, the team's understanding of the goal becomes clearer.

While such planned iteration is a fact of life in the development of certain products, clearly it is not an acceptable excuse for substandard design or planning. Effective use of iteration involves a process that reaches closure; that is, over time one would expect the scope of the iteration to narrow. In the development of software, one would expect to see reductions in the amount of change experienced over time in higher levels of a hierarchical (or layered) decomposition of the product architecture. Research in Design Structure Matrices seeks to reduce the complexities and introduce a level of control into the iteration process. Project tasks can be organized such that the nature of the iteration becomes more localized [20].

Even with the latest structuring techniques, some degree of controlled, planned iteration will remain. Network scheduling techniques are not well suited to handling such iteration. However, iterative blocks of tasks can be represented in such schedules; careful activity aggregation can help a network schedule better accommodate the iteration process. In addition, a documented task network can provide value even in such environments, since at a minimum it represents the current level of understanding of the team with respect to schedules and thereby serves as a communications mechanism. One can liken this to the use of design specification to capture the current level of understanding with respect to the product's capability and structure. When iteration yields insights that require changes to a product's design specifications, so too must the product's schedule specification's be updated.

Like its CPM and PERT ancestors, Critical Chain is a network scheduling technique. As such, it presents a static view of the project that is not entirely well suited to handling iteration. However, its basic fundamentals accept the fact that a degree of uncertainty will always exist in a project environment. Whereas many of the existing scheduling tools can accommodate *planned* iteration, Critical Chain provides some amount of protection against *unplanned* iteration. When the presence or level of iteration in a certain area is beyond that which was expected, the Critical Chain protective buffers provide a damper against the resulting effects. In this respect, the Critical Chain approach is vastly superior over CPM and PERT. Presumably as the organization gains a better understanding of its process iteration over time, a better tuning of explicit network tasks as well as appropriate buffer size can be performed.

4.6 Behavioral and Cultural Effects

An obvious concern involves whether organizational inertia or cultural biases will undermine the effort to instate more formal and rigorous Project Management practices. Resistance can be met on both ends of the spectrum. Managers may feel that more rigorous Project Management practice may reduce their flexibility. Those performing the work on tasks may feel a lack of obligation to the schedule if they did not participate in its construction [5]. These types of effects can potentially be stronger than the effects of using a new tool or scheduling technique.

Critical Chain is not immune from such problems. The resistance against Critical Chain may even be more substantial. Project Management as a discipline has become better developed in recent years with the availability of substantial training programs and professional certifications. Many of the mainstream teachings foster the establishment of a milestone culture, which is quite different from the approaches suggested in Critical Chain. The receptiveness to new approaches, such as Critical Chain, under such circumstances remains to be seen.

4.7 Additional Thoughts

The potential problems with traditional project scheduling mechanisms described in previous sections are long standing and well recognized in the general Project Management literature. The author was unable to find a single problem highlighted in Critical Chain that was not mentioned in the general Project Management literature at large.

What seems to be lacking in the general Project Management literature is a systematic, intuitive, and non-contradictory approach to addressing such problems. Critical Chain appears to provide some much-needed guidance in this area. The principles on which it is built are logically consistent and generally well proven.

Two questions emerge: Does this approach really work, and to what extent can it tolerate inaccuracies in planning and estimation. Subsequent chapters of this thesis will explore these questions. Case studies will explore the extent to which the use of buffers and buffer-management protect against uncertainty and whether the Critical Chain status-collection mechanism is a barrier to the self-fulfilling schedule.

Chapter 5

IBM's S/390 Software Development Environment

5.1 IBM S/390 Software Development Environment

The case studies in this thesis were conducted in IBM's S/390 Software Development Organization. Among the products that this organization has responsibility for is the OS/390[2] Operating System for the IBM S/390 Mainframe systems. These systems are direct descendants of the System 360 and associated OS/360 Operating System that originated over 20 years ago and the development experiences of which have been described by Fred Brooks in The Mythical Man Month [13].

Over the past eight years, this organization and its products have undergone a tremendous transformation. There has been a basic technology shift from a propriety higher cost bi-polar based system to a lower cost open CMOS based clustered system. As part of this transformation, the OS/390 Operating System has been re-written to support Parallel Sysplex Clustering in which up to 32 S/390 mainframes, each with the power of up to 1000 million instructions per second, operate as a cohesive unit. In addition, OS/390 now supports open technologies such as UNIX, the Distributed Computing Environment (DCE), and TCP/IP to name just a few. The OS/390 customer expects a product that can meet their needs in terms of growth, system availability, quality, and the availability of open technologies as well as the latest 'hot' applications.

The increasingly rapid rate of change in the computer industry (as in other industries), along with the increasing level of capabilities in the OS/390 Operation System, has increased the level of pressure on the Software Development Organization. The OS/390 system contains more than 70 integrated applications

and release cycle times have been reduced to 6 months from 18 to 24 months [14]. In order to embrace these challenges, the OS/390 Software Development organization has become flatter, matrix managed, and more project and customer focused.

Many of the observations by Brooks are equally relevant today if not more so given the demanding environment. Given the demands of the market in terms of product cycle time and cost, OS/390 Software Development has put increased emphasis on Integrated Product Development Processes as well as the Project Management discipline and its application. The Project Management approaches utilized are traditional techniques as taught by the Project Management Institute.

5.2 Scheduling Techniques Used in S/390

This section will describe some of the major activity scheduling techniques that are used in the S/390 Software Development organization. These techniques span the range from simple bar chart techniques, to lightweight milestone-driven techniques, and finally to more involved network techniques.

5.2.1 Bar Chart Techniques

Bar charts are by far the most prevalent scheduling technique used in the S/390 Software Development organization. Although this technique is simple both conceptually and in terms of implementation, it has proven quite effective for many years.

The scaling limitations inherent in bar charting techniques has been mitigated through integration of the technique with both process and product architecture. While this integration was perhaps not conscious, it has allowed application in a wide variety of projects in terms of size and number of people. A single monolithic schedule is not typically created for a project. Rather, the schedule is decomposed into more

[2] OS/390 is a registered trademark of the International Business Machines Corporation.

manageable chunks that allow the bar chart approach to be applied to a much wider range of projects. (Section 6.2 fully explores this issue.)

The application of this technique typically involves sub-dividing the project into a series of delivery segments called Drivers. These Drivers are milestones that represent the phased delivery of the product over time. The use of bar chart scheduling involves the sequencing of work according to precedence relationships for delivery into a target Driver. The following figure depicts a product delivery according to bar chart scheduling:

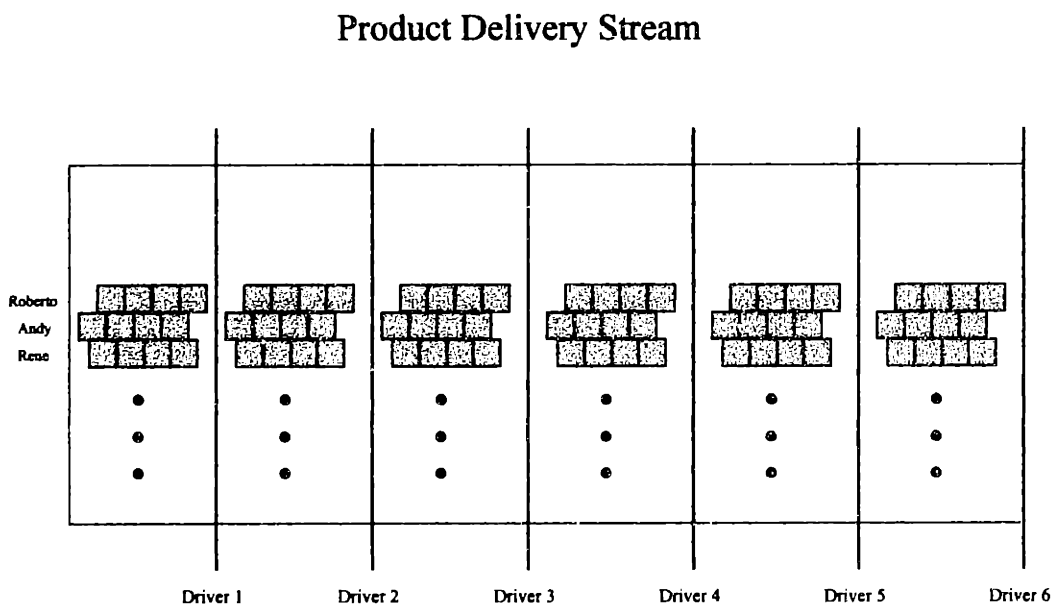


Figure 3: Bar Chart Scheduling in S/390

Most commonly, the bar chart schedule represents a resource perspective rather than a task perspective. That is, each horizontal line in the schedule represents the work for a specific person or team *(Roberto, Andy, and Rene* in the simple example of the previous figure). This allows for more careful management of people resources, the scarcest resource in a Software Development Environment. Should a person's work for a given Driver complete before Driver ship, work on the next Driver can begin.

The bar chart project schedule would typically be generated at the beginning of a project. An aggregate planning estimate for number of required people is performed for the project based on an estimated line of code cost, a standard productivity rate, and target completion date that includes contingency. From here, project tasks can be assigned to a Driver and the schedule adjusted to incorporate the specific people available. Should this exercise reveal a problem in terms of achieving the target date, the date either can be moved, additional people be added, or the associated level of risk be accepted.

In addition to techniques that limit the scaling problems associated with bar chart scheduling, the use of explicitness has also contributed to the success of this technique. A more explicit identification of tasks can reduce the level of contingency needed and yield a more reliable schedule. There are a number of activities that contribute to a successful product that are beyond the main act of developing and testing software, but nevertheless can consume non-trivial amounts of developer and tester time. These include, but are not limited to, contributions to publications, working with beta customers, prior release service commitments, and general education of customers, vendors, and support personnel.

Through experience, it is usually easy to have quite a good idea as to the timing and requirements of these additional activities. As such, they can easily be added to the bar chart schedule. A quick summation of all the tasks along any vertical line (or point in time) can provide an indication as to whether the total workload exceeds the available capacity.

While the approach has been used successfully, there are some inherent limitations. Perhaps most significant, the technique does not highlight the most critical elements of the project in the way that CPM or Critical Chain does. As long as things go according to plan, this doesn't matter too much. However, should things go awry, it can be difficult to see what specific elements should receive the most focus. The most extreme outcome of this being a situation in which almost everything seems equally stressed at the same time.

Furthermore, the lack of knowledge concerning the Critical Path or Critical Chain does not allow an easy assessment of the efficiency and/or reliability of the schedule. For example, should the resulting schedule be of a greater duration than desired, it can be difficult to determine what resource or task rearrangements may yield improvement. The Driver sub-divisions may obscure this issue even further; many precedence issues are resolved at the Driver boundary, thereby reducing the possibility for finer grained optimization. Lastly, the lack of carrying explicit precedence information may result in an inefficient overlapping of activities, especially after modifications that result from mid-course corrections.

5.2.2 "Wave" Scheduling

Another scheduling technique involves a simplification of bar chart scheduling, which will be called "wave" scheduling for the purposes of this thesis.. This approach is applicable to a project that involves the development of a completely new and unprecedented software product as opposed to an enhancement. It essentially accepts the fact that it will be very difficult, and perhaps wasteful, to generate an end to end schedule for the project up front. This approach will instead perform the scheduling in waves.

Central to this approach is a list of all known project activities. Many of these may be in a state in which their cost is not completely understood; some amount of work will need to be first performed on the project to gain the necessary level of understanding. An attempt to gain this understanding without exploratory work can waste valuable project time.

The list of known activities is scheduled to Drivers in waves. At the beginning of the project, activities are scheduled to Drivers 1, 2, and 3 with the most accurate information known for Driver 1. Once Driver 1 completes, activities are then scheduled to Drivers 2, 3, and 4. Thus, as additional information is gained, Drivers 2 and 3 may be adjusted. For example, additional activities may be identified, or

activities may be moved between Drivers according to a revised functional rollout. In general, when Driver *n-1* is complete, activities are scheduled to Driver *n, n+1*, and *n+2*.

Due to concerns with respect to Parkinson's Law, estimates of activity duration are made with little or no contingency. Thus, movement of activities between Drivers over time can be quite common. Often, the general trend is movement to the right. This may be caused by a combination of overly aggressive estimations or a failure to return positive variance to the project.

The verdict is still out with respect to this approach. On the positive side, this approach provides for a very aggressive project start with significant early progress. However, the weaknesses of the bar chart scheduling approach are amplified with "wave" scheduling. In addition, a fear is that this approach not only accepts uncertainty as a given, but may also encourage it. Given that there is a basic expectation that estimates of duration and the set of project tasks will be inaccurate and subject to change over time, a relaxing of design and planning emphasis may result. Thus, rather than the project attempting to control uncertainty, uncertainty may end up controlling the project.

5.2.3 CPM

There has been recent renewed interest in the CPM scheduling technique in the S/390 Software Development organization as a result of more formal Project Management training programs. The result of applying this approach has not been at the same level as that of bar chart scheduling. There appears to be a greater success in applying CPM in S/390 Engineering. While there may be some inherent difference in the nature of the work that may contribute to this disparity, there may also be some basic implementation issues involved as well.

When one considers the application of bar chart scheduling versus experiences with CPM, key differences emerge:

- *The issue of detail:* The CPM schedule tends to be far more detailed. In addition, rather than using a combination of simpler schedules as done with bar chart scheduling, the tendency is more towards a single monolithic schedule. The result can be a very large schedule even considering the software automation available. Such a schedule may not correspond to the level of detail at which the project is managed and become overly cumbersome to modify.
- *The issue of whom*: The element of who creates the schedule is also a factor. In bar chart scheduling, those doing the actual work typically create the individual schedule elements. In CPM, the schedule tends to be created by a single person on behalf of several teams due to the specialized Project Management software and skills involved. As a result, those doing the actual work may feel less obligated to ensure that schedules are accurate and remain accurate.
- *The issue of changing for the sake of changing*: On teams whose projects are small to medium in size (say 25 people or less), and whose teams have deep experience in the technology area, CPM may simply offer too little benefit as compared to bar chart scheduling when compared to the cost. What is the motivation for a team to switch, if good results have been achieved with bar chart scheduling?
- *The issue of familiarity*: General Project Management and CPM training has not consistently spread to those who would typically have been performing bar chart scheduling in the past, such as team leaders. Thus, CPM represents an additional level of burden, especially considering the technical challenges involved in the typical project.

If the factors that have contributed to the success of bar chart scheduling were also used in CPM, more benefits could possibly be achieved. It would seem a wise investment for the organization to explore this issue further. Despite the inherent limitations to CPM, the additional level of insight it offers may indeed prove valuable.

Chapter 6

Critical Chain Implementation Issues in S/390

6.1 Overview

Critical Chain on its own and applied in isolation has little chance of contributing to a successful project outcome. Rather, in order to provide potential benefits, the Critical Chain approach must be carefully integrated with existing organizational processes. Particular attention must be paid to those factors that have affected the success of other scheduling techniques in the past, as well as to those factors that may be a particularly significant detriment to the Critical Chain approach.

This chapter will explore the factors that have historically proven critical to successful application of scheduling techniques in the S/390 Software Development organization. In addition, new issues that must be considered with the introduction of Critical Chain will also be discussed.

6.2 Integration with Function-Form Decomposition of Product Architecture

Decomposition of Product Architecture along the function-form attribute involves "cleaving" the product according to the various functions performed and the associated physical manifestations of those functions. This allows the tasks associated with the development of a product to be arranged in a tree fashion with additional detail visible as one travels down the tree.

Typically, this approach to decomposition alternates between function and form. A high level function is first identified. Given this function and with a product concept in mind, a product form can be defined. For example, the function to provide an abstraction layer between hardware and applications can take the form of a general purpose Operating System product given the product concept of a data processing

system. Given the definition of concept and form, additional lower level functions can be defined. These lower level functions can be embodied in terms of form. The decomposition can proceed in this manner until the desired level of granularity is attained. *Figure 4: Decomposition of OS/390 along function and form attributes* provides an example of this decomposition for OS/390 alternating between function and form (Note that the decomposition is expanded for the Component Broker subsystem).

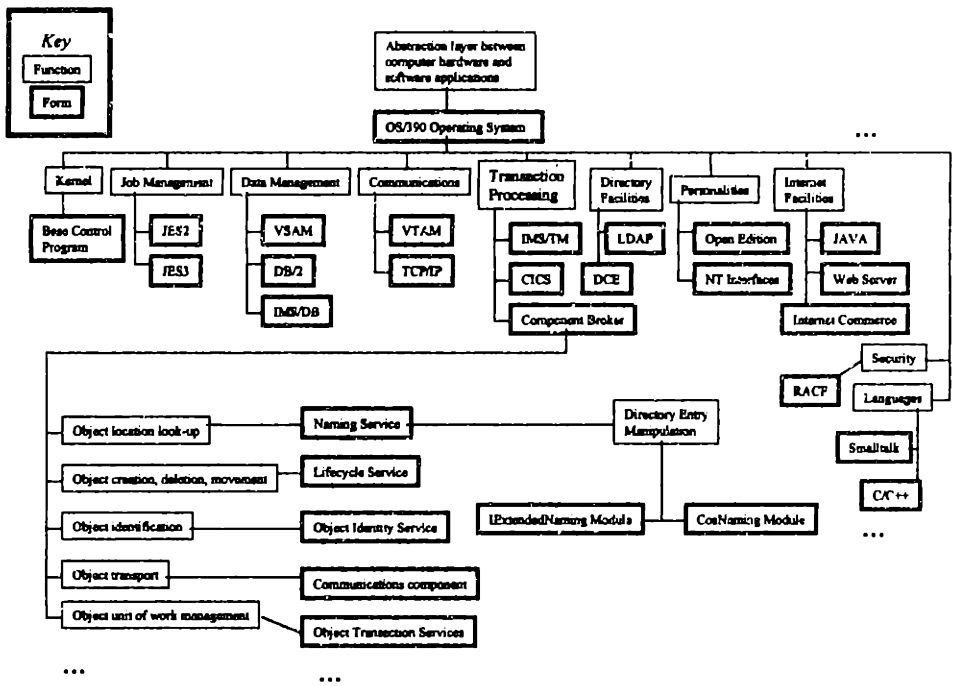


Figure 4: Decomposition of OS/390 along function and form attributes

The elements of *form* in the figure above identify the basic physical elements of the operating system. These *physical* elements of the Operating System can be decomposed along the form attribute as follows: Operating Systems → Subsystem → Component → Sub-component → Module → Subroutine. For example, the figure shows only the form elements of the decomposition.

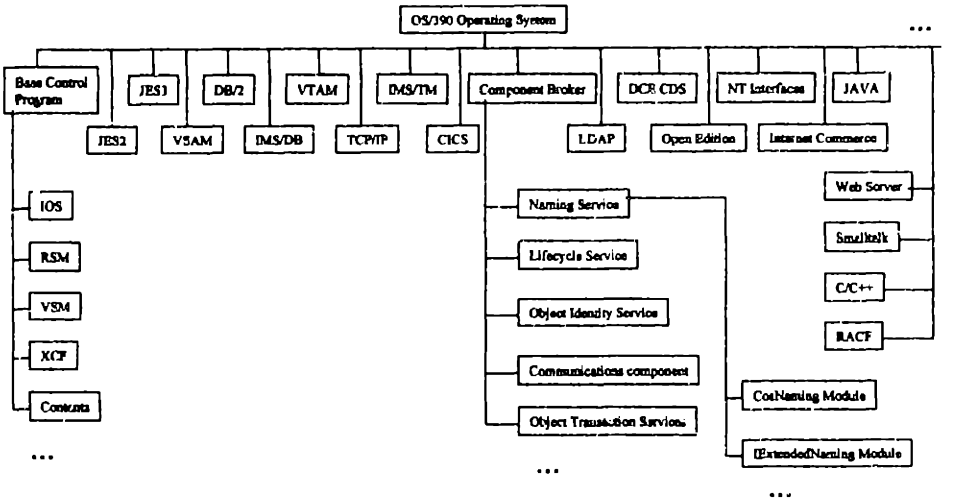


Figure 5: Decomposition of OS/390 along the form attribute

The organizational structure, in terms of team definition, is typically aligned according to product form, especially in the case of new product development. For example, there may be a team for each major component. These teams often have technical as well as planning responsibilities for their portion of the product. Thus, the teams doing the actual work can maintain the lowest level of scheduling detail. Those in broader scope Project Management roles can maintain a higher level of detail.

With this approach, the schedule can be decomposed along with the product architecture. Thus, a release Project Manager may deal with a relatively aggregate schedule maintained at an upper level of the product-schedule decomposition. Such a schedule might deal with important subsystem or component integration and test milestones. A particular component team may deal with a highly granular schedule that is maintained at lower level of the product-schedule decomposition. A schedule at this level might include important sub-component or module integration and test milestones. Through this approach, the complexity of the schedule can be controlled while providing a necessary level of detail to those who need it.

The integration of scheduling with the decomposition of product architecture along the form attribute allows simple bar chart scheduling approaches to achieve adequate scaling; the schedule is decomposed into smaller, more manageable chunks. An attempt to create a single detailed schedule during usage of CPM scheduling applications (as opposed to a hierarchy) contributed to the burdensome nature of the experiences with this approach.

This integration with product architecture decomposition is therefore a relevant success factor. Any application of Critical Chain in the organization should therefore take full advantage of this knowledge. The key factor here is maintaining schedules of manageable size using decomposition of product architecture along the form attribute as a method of simplifying individual schedules if necessary. This observation is extensible to other scheduling approaches besides Critical Chain, such as CPM.

6.3 Integration with Horizontal Decomposition of Product Architecture

Horizontal decomposition of product architecture involves cleaving the architecture across functional layers. For example, the TCP/IP stack is decomposed in this manner and consists of the following major layers: link layer, network layer, and end-to-end layer. Decomposition of the architecture in this manner is complimentary to function-form decomposition and allows the product to become more extensible. Various layers of the product can be used for multiple purposes, creating a platform upon which more complicated software can be built. The following figure represents the layering present in the S/390 I/O subsystem to further illustrate this point:

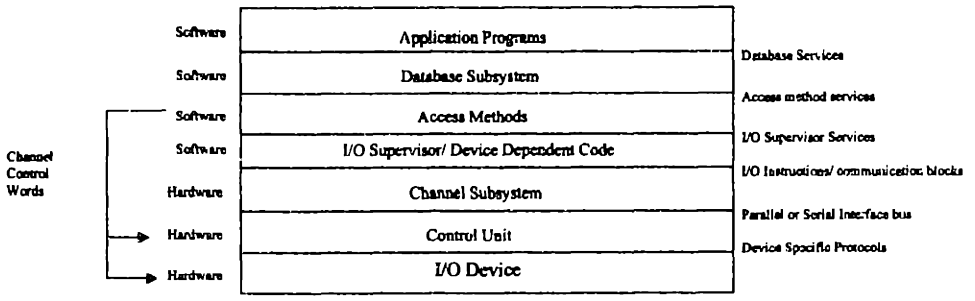


Figure 6: Horizontal decomposition of S/390 I/O Subsystem

The horizontal decomposition not only serves as a technically useful method of decomposing product architecture, it also serves as an excellent mechanism for ordering project work. The lowest levels of the horizontal decomposition would be delivered first, thus establishing a growing base of usable function. This horizontal decomposition fits well with the delivery pattern of S/390 software products.

The process for the implementation of a new Software product (or product enhancement) involves the creation of a delivery stream into which software function is delivered. At the end of the delivery stream, a specified level of the function for the product will have been delivered. This delivery stream is typically subdivided into discrete sections called Drivers. These Drivers typically occur on a monthly basis, but this can vary depending on the needs of the specific project. Typically, additional layers of function are transmitted (added) into the delivery stream with each passing Driver.

This division of the delivery stream into discrete segments has proved to be a simplifying factor that has made the extensive use of bar chart scheduling possible. Product functions that are associated with various tasks are targeted to specific Driver milestones. Thus, a bar chart schedule can be formulated that maps the delivery of the product in discrete phases (or layers) that correspond to discrete Drivers. This allows the precedence relationships between tasks to be hierarchically decomposed. A precedence relationship between Drivers exists at the highest level with a precedence relationship between tasks

existing within a Driver. This decomposition allows the simple bar charting technique to be used despite its lack of explicit provision for specification of task precedence relationships. Functions are assigned to Drivers in the order given by the horizontal decomposition and planning focuses on Driver content and ordering. The follow figure depicts the decomposition of precedence relationships:

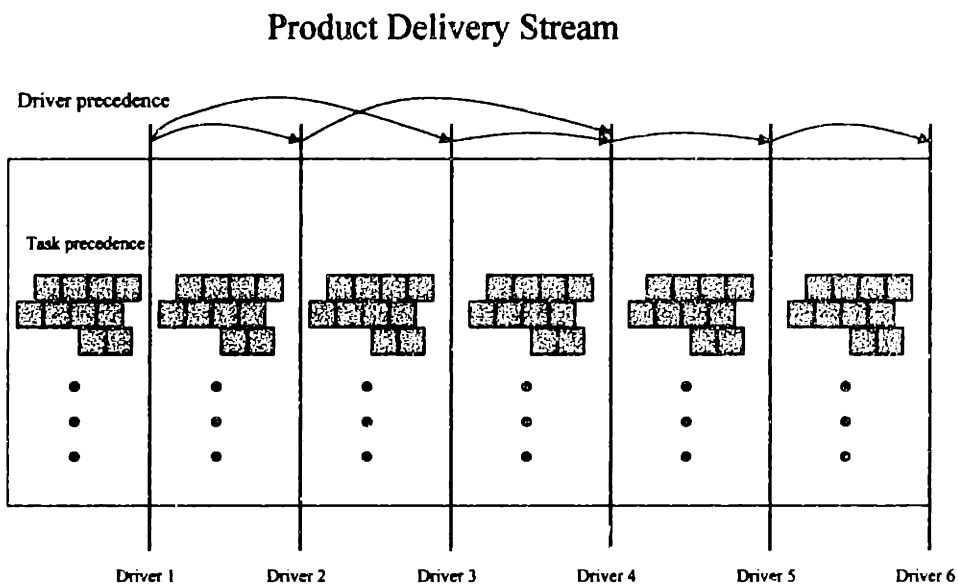


Figure 7: Decomposition of Precedence Relationships

This horizontal decomposition is another factor contributing to the success of prior project scheduling techniques such as bar chart schedule. This too should be embraced in the implementation of the Critical Chain methodology. Clearly the notion of phased and layered delivery of product function can easily be applied to the Critical Chain approach.

However, care must be taken with respect to Drivers in the context of a Critical Chain implementation. Drivers are the basic element of integration and are required for creation of the product. They represent points of 'fan-in' during which current in-progress work is packaged together in order to establish a base for future work. These Driver integration points can be effectively represented in a Critical Chain

schedule. The Critical Chain of the schedule would presumably weave its way though each Driver with the Drivers being protected from non-critical chains of tasks by feeder buffers. However, Drivers also represent significant project milestones and are typically fixed in time. Achieving such a project milestone with reasonable certainty implies that buffers should be established to protect the date. This would require that the project buffer be distributed among the Drivers. Thus, the project schedule would have buffer periods dispersed throughout. Doing this may effectively protect Driver dates, but may also introduce a level of risk greater than that which would be experienced with a single aggregated project buffer placed at the end of the project. In essence, an important project end date may be put at risk in order to protect less important intermediate dates, since the benefits of positive variance gained during early Drivers may be lost. This approach, taken to the extreme, would take things back to no project buffer with contingency added to each task. However, with careful management, such an approach could be made viable.

Another viable approach involves eliminating the restriction that Drivers be fixed in time. This can be achieved by either allowing them to float or having them occur at a sufficiently high frequency (perhaps weekly). As a result, Drivers would integrate when ready and the Critical Chain for the project would weave its way through the Drivers as described earlier. A more aggressive manifestation of this would be a continuous integration model. Here, each element of function would integrate when ready. The notion of a Driver as a major integration point would essentially disappear. This approach may exceed the capabilities of the processes and tooling available. In addition, a side effect of this approach could be increased schedule complexity, since precedence relationships would no longer be hierarchically decomposed and a much higher grained horizontal decomposition would be required.

The use of the horizontal decomposition as a simplification mechanism was a success factor for previous scheduling techniques. This combined with the potential limitations of tooling, makes the use of a Driver as an integration point a good starting point for a Critical Chain implementation in the S/390

organization. Allowing the Driver to float or occur at a sufficiently high frequency such that the Driver does not become a milestone that requires buffer protection also seems like a reasonable starting point, given the potential downside of doing otherwise.

6.4 Integration with Process Architecture

While fairly obvious, the importance of process architecture should nonetheless be stated. The product development process embodies the organization's knowledge in terms of the activities required to bring a product to fruition. In terms of schedule, it defines the set of activities that need to be performed with respect to the various elements of the product architecture. While the specific make-up of the process may vary from product to product, the existence of a well-defined product development process is crucial to project success. Clearly, this is a key element in controlling the level of risk and uncertainty on a project. An important aspect of creating a well-defined product development process also involves structuring the sequence of process phases such that planned iteration can be minimized. Design Structure Matrices (also mentioned in section 4.5) provide a means to achieve this.

6.5 Aggregation versus Granularity

In any given schedule, whether schedule decomposition is used or not, the issue of proper degree of activity aggregation must be considered. The issue of Software Development Process becomes relevant here. To implement any given product function, a number of process phases must be executed. In S/390, the typical development team would be responsible for seven of these phases: Product Level Design (PLD), Component Level Design (CLD), Module Level Design (MLD), Code, Unit Test, and Function Component Test (FCT).

If a given project consisting of m elements and a team of n people responsible for p process phases, then clearly a schedule consisting of $m*n*p$ activities can be generated. With this high level of granularity,

even a seemingly simple project can generate an amazingly cumbersome schedule. Care must be taken to avoid a schedule that is more detailed than the level that the team desires or is capable of managing.

A typical solution to this problem is to choose an appropriate layer of the product function-form decomposition (perhaps component) and an aggregation of process phases. Using component as the decomposition layer typically allows a single small team to manage its schedule.

Process phases are typically combined as follows: PLD-CLD, MLD-Code, Unit Test, and Function Component Test. The only special meaning here is that this grouping is often how teams prefer to think about their work. Another possible grouping is PLD, CLD-MLD-Code-Unit Test, and Function Component Test. This grouping has a benefit in that much of the iteration that occurs in iterative development occurs between CLD and Unit Test. Thus, a somewhat static network schedule can be used in an environment of planned iterative development without need for continual updates. A caveat here is that there is a risk of such aggregation hiding important information.

Level of schedule granularity is an important tool that influences the effectiveness of scheduling discipline. It represents another mechanism by which simple bar chart scheduling was extensible to a wide variety of project environments. This aggregation should often be combined with hierarchical schedule decomposition to ensure that detail is not simply aggregated away. This granularity issue must also be considered in a Critical Chain implementation.

6.6 The Human Element

Another critical contributor to the success of bar chart scheduling is related to the human element. Bar chart scheduling is easy to understand, everyone knows how to do it, everyone is experienced with it, and there are no special software tools required. No Project Management gurus are required to generate the schedule.

This alone could explain the relatively slow spread of CPM scheduling techniques. The special tooling and training required in conjunction with the apparent lack of value introduced by the technique can make it a tough sell. Perhaps CPM's only saving grace is the fact that it is a standard Project Management practice. It is here that Critical Chain may have the greatest barriers to break. Critical Chain is non-standard, unfamiliar, requires specialized software tools to implement effectively, and a single person often utilizes that tool.

In the context of the case studies of this thesis, the following will be done in an attempt to mitigate these factors:

- The participating team will receive education with respect to Critical Chain and the motivation to use it.
- The participating team will make the decision as to whether they will participate in the case study.
- An appropriate management environment will be established to alleviate the negative stigma associated with milestone management and encourage the generation of 50-50 dates.
- Schedules will not be imposed on the team. Rather, the team will have the most significant voice in identifying project tasks and most importantly their duration.

While this in no means completely solves these problems, the hope is that it will sufficiently alleviate them to allow a reasonable case study to take place.

6.7 Conclusions

The factors discussed in this chapter together make the point that the generation of reliable project schedules is only partly a result of the scheduling technique chosen. A foundation of solid product

architecture, process architecture, and organizational architecture must be in place before any matter of credible schedules can be discussed. This issue is depicted by the following figure:

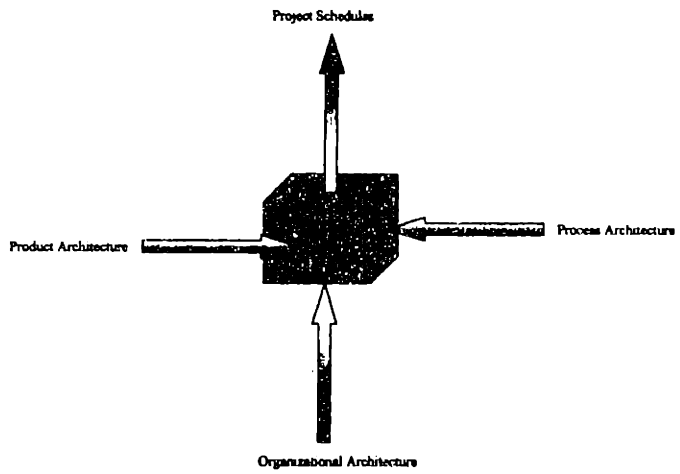


Figure 8: Foundations for Successful Project Scheduling

The more poorly defined the Architectural elements, the more unreliable the schedule. Thus, any study of a project-scheduling mechanism must be careful to highlight the factors influenced by the approach chosen versus underlying foundational issues. An additional opportunity for research involves how to resolve this issue against the realities of new technology development. Since architecture may not be well defined up front, what are the appropriate approaches and mechanisms that can be use to transition into a better-defined environment with more reliable schedules? What are the implications of this with respect to Integrated Product Development Processes and product delivery commitments?

Lastly, one must not neglect aspects of implementation specifics or those related to the human element. These issues also have the potential of influencing the potential case study in a manner more significant than the actual scheduling mechanism chosen. Again, care must be taken to separate factors that can be attributed to the scheduling technique from others.

Chapter 7

S/390 Critical Chain Case Studies

7.1 Background

Critical Chain project scheduling is a technique that many people find quite intuitive. The literature associated with it makes substantial claims as to its potential benefits. However, are these benefits real? Can Critical Chain be made to work in a Software Development environment? Can it be made to work within the context of existing organizational culture and a milestone driven management system? Can it specifically be made to work in the S/390 Software Development organization? Such questions can only be answered though case study.

Two Critical Chain case studies were conducted in the S/390 Software Development organization. Each of these case studies involved portions of IBM's new Component Broker product for the OS/390 Operating System. The first case study represented new product development involving work in the core of the Component Broker/390 product, specifically the Naming and LifeCycle services. The second case study involved enhancements to the security subsystem of the Operating System (known as RACF[3]) on behalf of the Component Broker/390 product. Special care was taken to ensure adequate emphasis was given to foundation factors that contribute to successful project planning as described in the previous chapter.

Each of the two case studies was conducted within the context of an overall larger milestone-driven Project Management system. The existing milestone driven system is one in which a considerable amount of effort has been invested and whose scope it quite broad. It is also a system that has been

[3] RACF is a trademark of the International Business Machines Corporation.

successfully used to manage large projects for quite some time. It is therefore unreasonable to expect that a Critical Chain case study will dismiss these systems entirely and start over. Rather, a Critical Chain case study must integrate with such systems. Should the case studies prove motivational to the organization, further consideration can be given as to whether Critical Chain should take a broader and more widely integrated role.

Specialized Critical Chain scheduling software was used in the implementation of the case studies. This software came in the form of ProChain, an add-on to Microsoft Project. An initial attempt was made to use CA-SuperProject, which has no explicit Critical Chain support. Buffers were represented by dummy tasks and buffer consumption was calculated manually. This approach proved too cumbersome for even the simplest of schedules and motivated the switch to the ProChain software.

7.2 Product Highlights

Component Broker is IBM's implementation of the Common Object Request Broker Architecture (CORBA). This product provides a distributed object transaction capability for the operating system. For OS/390, a goal is to combine the operating system's historical strengths in transaction processing with the openness, standardization, and ease of programming offered by CORBA.

A considerable benefit of Component Broker is that it will allow application programmers to focus on solving their business problems while reducing the time spent writing infrastructure. This infrastructure consists of both communication and object distribution capabilities. It also consists of the qualities of service of the execution environment. For example, application programmers will no longer need to worry about ensuring that their object requests are secure and transactional. Doing so requires significant expertise and experience to implement. Providing such infrastructure in an implementation neutral fashion via an Object Request Broker allows the application programmer to solve business problems with shorter lead times.

Another considerable benefit of Component Broker is a reduction in skill required on the part of application programmers who write programs for the OS/390 Operating System. In the past, such programmers were required to be skilled in the proprietary programming environment of transaction processing systems, such as CICS[4], which run on OS/390. With Component Broker the strengths of the programming environment offered by transaction processing systems such as CICS will be maintained, but the programming model of the new system will be CORBA based. Thus, programmers with knowledge of CORBA can write applications for OS/390, focussing on the business problem at hand, with Component Broker and the underlying operating system handling the details of instance management, security, locking, and transactional scope.

7.3 Nature of the Software Development Environment

The Component Broker project represented a highly aggressive undertaking. It affected not only the core components of the Operating System, but also the linkages between these components. In terms of the classifications given by Henderson and Clark for the categorization of innovations, Component Broker fits most closely to the *radical* classification with respect to the technological characteristics of the Operating System.

The actual programming environment was more dynamic than is typically the case for a single project. The programming languages used included C++, Java, S/390 Assembler, and PL/X (a proprietary language). In addition, programmers made use of the Interface Definition Language (IDL) specified by the CORBA architecture. This highly heterogeneous environment required substantial changes to the normal tooling and infrastructure, such as the development environment and the library system. The net result was an extremely challenging environment along a number dimensions.

Case Study 1 involved work that was in the core of the Component Broker product. As a result, it was exposed to a significant number of challenges both in terms of technology development and underlying infrastructure and tools. Case Study 2 involved changes to an existing subsystem of the Operating System. The environment here was considerably more stable.

7.4 Case Study 1: Naming and LifeCycle Services

7.4.1 Introduction

The Naming and LifeCycle Services are core components of the Component Broker/390 (CB/390) product. The Naming Service provides object lookup functions. It can essentially be considered a distributed directory service that associates human readable names with objects. The LifeCycle Service provides a means to find factories (objects that create objects) as well as create, destroy, move, and copy objects. Each of these two services is a key element of the product's programming model; that is, they provide key functions to the application programmer.

Just as these two object services are part of an overall larger system, so too is this Critical Chain case study conducted as part of an overall larger Project Management system. The overall Project Management system for the product was a milestone-driven system. Thus, mechanisms had to be developed to allow these two systems to function together.

The team consisted of six people at its peak and the scope of the case study spanned Module Level Design through Unit Test. The team members had a range of experience in the Industry from less than 1 year to over 20. The author's role on this team was that of team leader, designer, developer, and Critical Chain advisor.

---

[4] CICS is a register trademark of the International Business Machines Corporation.

Application of the Critical Chain approach encompassed the latter portion of Naming and the entirety of LifeCycle (this being the point at which the author learned of Critical Chain). The case study spanned a period of approximately 8 months. Total size of the Naming and LifeCycle frameworks was approximately 7,500 lines of code with an original estimate of 7,000 lines of code.

7.4.2 Getting Started

The first step of the case study involved soliciting interest in the Critical Chain methodology. To this end, the author discussed the approach and its potential merits with both the team involved as well as those involved with the overall management of the project. Most found Critical Chain to be an intuitive, common sense based approach.

The team made the decision to try the approach with the support of management based on the potential benefits it could offer. This aspect of the case study went more smoothly than expected. A potential factor that contributed to this involves the level of history and trust among members of the team. The team and overall management trusted that the author would not do anything to jeopardize the project at large or the careers of the individuals involved. The team was confident of its skills to get the job done irrespective of the activity scheduling mechanism chosen.

This level of trust at least partly helped alleviate concerns stemming from the pursuit of more aggressive schedules based on 50-50 dates in an environment of milestone-driven Project Management. The author made it clear that with the use of more aggressive 50-50 dates, movement of individual tasks can only be expected. In addition, team members would have a voice in the creation of these dates. As long as every effort was made to achieve the dates, team members would not be 'penalized' for missing a task end date. In addition, other team members would jump in to help out if that made sense for the particular situation. This helped to alleviate concerns, but the concern of how to resolve this against the milestone-driven Project Management of the larger project remained.

The team's initial, and perhaps ideological, approach to interfacing with the larger Project Management of the team was to educate them. Once they understood the Critical Chain approach, they would understand the nature of the intermediate delivery commitments provided to them. They were encouraged to make use of the project buffer versus progress made as their assessment mechanism. As will be discussed later, this approach proved to be unworkable.

The question of incentives was also discussed. Given that the un-buffered project completion date is an aggressive *goal* and the buffered date represents a *commitment*, what are the incentives to strive for the aggressive goal? The team realized that variability could strike at any time and therefore striving for the aggressive goal was a part of making sure the commitment was achieved. The team also realized that beating the commitment date could allow time for other things like education, vacation, or assisting other project areas.

7.4.3 Planning

The planning for the development schedule was based upon a well-defined technical design and a well-defined process. Thus total cost of the project could be well estimated and the identification of tasks could be made with some confidence. In addition, Drivers occurred on a weekly basis. This allowed the team to effectively integrate code and fixes without need for buffering drivers. As a result, a single aggregate project buffer placed at the end of the Unit Test phase could be used. The initial project schedule, including Critical Chain buffers, is shown in the following figure:

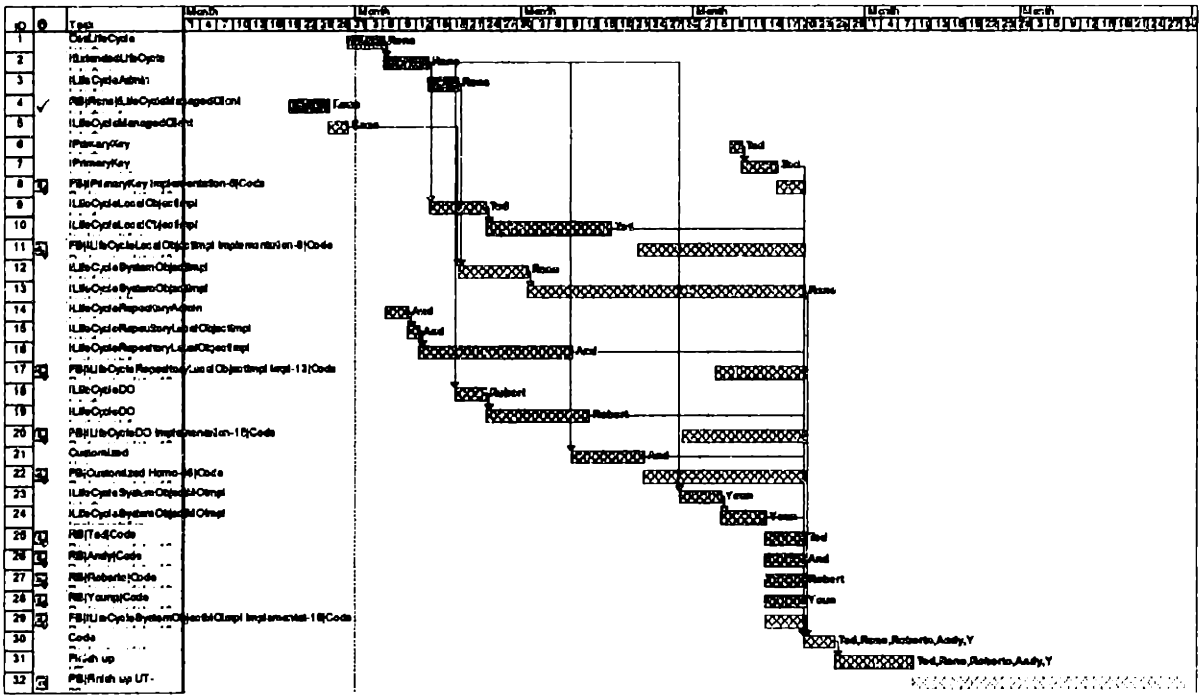


Figure 9: Initial Case 1 Project Schedule

The above schedule was created through use of the ProChain software. This process begins with the creation of a task network in much the same way as would be done for a normal CPM based Microsoft Project schedule. However, the duration of tasks is given by 50-50 times. At the end of this step, the schedule consists of the work-items, their interdependencies, as well as who is responsible for them. In terms of the schedule above, this would consist of all tasks with the exception those with names beginning with *RB, FB,* and *PB* (these are added later by the ProChain Software: RB represents a resource buffer, FB represents a feeder buffer, and PB represents a project buffer).

The next step in the use of the software involves performing load leveling via a ProChain option. This step de-conflicts the schedule for resource contention. For example, in the schedule above, load-leveling would ensure that tasks 1 and 5 as well as 14 and 21 do not occur simultaneously even though there is no precedence relationship between them. Those tasks cannot occur simultaneously, because the same person is working on each 100% of the time.

The next step is to identify the project critical chain via a another ProChain option. In the schedule above, the Critical Chain consists of tasks 5, 1, 2, 3, 12, 13, 30, and 31. The remaining tasks are non-critical and feed into the Critical Chain. For example, tasks 9 and 10 represent a chain that feeds into the Critical Chain.

The last step is to activate the ProChain option to add the protective buffers. In the above schedule, this adds the tasks whose names begin with *RB*, *FB*, and *PB*. The *RB* tasks represent resource buffers and are given by tasks 4 and 25-28 in the above schedule. These represent a warning (or alarm) to an individual that they soon will be working on a Critical Chain task. They do not add any duration to the overall schedule. For the most part, neither of the case studies of this thesis derived any benefit from these buffers. There was a sufficient awareness such that they added no value. However, for highly complex schedules, they could offer benefits.

The *FB* tasks represent Feeder Buffers and are given by tasks 8, 11, 17, 20, 22, and 29. These buffers protect the Critical Chain of tasks from disruptions that occur in non-critical tasks. Task 11 protects the Critical Chain at task 30 from problems occurring in tasks 9 and 10. In addition, should some Critical Chain task that is a predecessor to task 30 finish early, the buffer represented at task 11 helps ensure that task 30 has at least a possibility of starting early. Without it, task 30 would be delayed by task 10.

The size of the Feeder Buffers used was simply 50% the duration of the Chain of tasks it is protecting. So, the duration of the Feeder Buffer given by task 11 is 50% the duration of tasks 9 and 10. (Note: it may not appear so on the schedule above due to scheduled vacation times.) It should be noted that the addition of feeder buffers might extend the duration of the schedule. There may not be room to add them without spreading out the Critical Chain tasks a bit. However, ProChain has an option that allows the Feeder Buffers to be truncated in order to prevent the Critical Chain tasks from being spread out. This option

was selected for this thesis. Without selecting this option, the schedule can began to appear somewhat 'fat'. While this choice had little impact on this case study, the impact was more significant on the second study. Truncating Feeder Buffers increases the level of risk in the schedule. It increases the probability that a non-critical task will delay the critical chain and thereby cause Project Buffer to be consumed. In essence, it forces the Project Buffer to provide a level of protection beyond which it was designed for. This risk was deemed acceptable in the context of this case study.

Finally, the *PB* task denotes the Project Buffer. The project buffer protects the ultimate end date of the project from difficulties that might occur on the Critical Chain of the project. In the schedule above, the Project Buffer is denoted by task 32. The size of the project buffer was 50% the duration of the Critical Chain. Note that the ProChain software allows percent size of the buffer to be specified. However, 50% is the recommended rule of thumb.

In the construction of the schedule, the team was careful not to create an over abundance of detail. In this case, the granularity of the schedule was limited to software module, and Module Level Design and Code tasks were aggregated together. This provided a sufficient level of control to avoid the need for low-level hierarchically decomposed schedules. As a generally rule of thumb, the team found that a schedule requiring more than six or so pages to be represented in a printout proved overly cumbersome to manage given the fluidity of the environment. This finding was also true of the second case study.

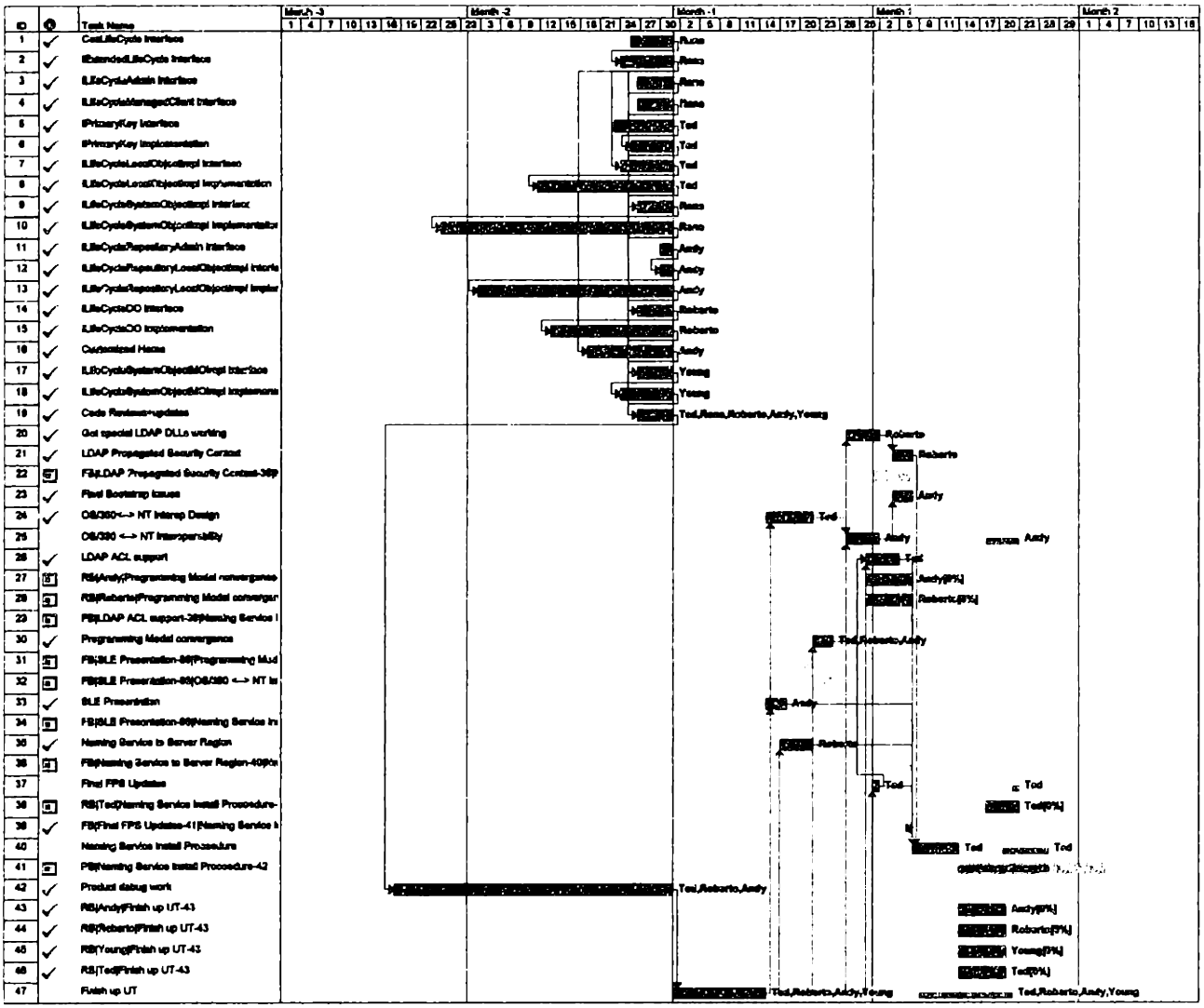
Limiting schedule granularity to the desired level of product architectural detail (module), in conjunction to the limited number of process phases involved in the case study, provided for a very manageable schedule in terms of its size. While some tasks exceed the 2-week maximum task duration suggested in work breakdown structure guidelines, the team felt that it was more important to have a manageable schedule and that the higher level of granularity would not impede its ability to manage the project.

This initial project schedule included activities for the LifeCycle Service only and was created during the Unit Test phase of Naming. By the time work began of activities defined in this schedule, the unit test of Naming was not complete due to difficulties experienced with the underlying infrastructure of the product. The decision was made to proceed with the development of Lifecycle and later conduct a joint Unit Test of both Naming and LifeCycle.

7.4.4 Case Study Progression

The team was able to either meet or come very close to the initial 50-50 schedule through the code phase of LifeCycle. In general, the team was able to employ a focused approach to accomplishing work. The team ensured that those working on Critical Chain tasks were protected from distraction. As work progressed, additional work items were defined (mostly with respect to the Naming Service) as a result of latent requirements imposed by other elements of the product. These additional activities were deferred to maintain the current level of focus of the team.

Thus, the coding phase of LifeCycle completed with the project buffer largely in tact. At this point, the team created a new schedule that included the remaining test activities for both the Naming and LifeCycle services as well as the set of work-items that were deferred earlier. This new schedule, complete with protective buffers, was created totally within the confines of the original project buffer and so the project end date did not change. The following figure shows an example schedule where work has been scheduled into the project buffer. The task aligned to the left are completed tasks (task 1-19, 42). The remaining tasks represent new work plus protective buffers. Note that the original Project Buffer is no longer visible. In essence, a new schedule with the remaining work has been created. And this schedule just so happens to fit into what was the original Project Buffer (with the exception of task 42- discussed shortly).



Work on many of the remaining activities progressed nicely. However, testing remained a stumbling block. The Naming and LifeCycle services served as a comprehensive first test case for the overall product. Thus, the team spent large amounts of time in the role of overall product tester, as opposed to testers of their own component. The significance of this fact made the team realize that the project buffer was intended to protect against variability in tasks defined within its own schedule as opposed to chains of tasks that enter the schedule from the outside.

This caused the team to add an overall product test task to the schedule, to take time spent testing other areas of the product into account. This is represented by task 42 in the schedule above. This resulted in a

significant one-month plus slip to the original buffered end date (work scheduled into the original project buffer moved to the right). However, by this time, the overall project milestones had been adjusted to the right. Naming and LifeCycle work completed by the revised development end milestone and was one of the few teams to be considered 'on-time'. The quality of the work performed by the team appears to be equal to or somewhat better than that experienced in other components of the project.

7.4.5 Case Study Results

Overall, the Case 1 team felt that the experience with Critical Chain was positive. Critical Chain provides a more intuitive way to organize the project work as compared to other scheduling techniques, such as bar chart scheduling. This was largely a result of the explicit consideration of uncertainty in the schedule, a clear understanding of the project constraints, and a larger degree of robustness against schedule updates. The basic conflict between a desire to be aggressive *and* to provide reliable end dates was eliminated.

7.4.5.1 Benefits of Critical Chain

It was the use of strategic project buffers that allowed the team to protect the project against uncertainty without sacrificing the level of aggressiveness that was desired. The un-buffered project end date was a target that the team aimed for and the buffered end date was the commitment provided to those external to the team. In general, this structure increased the level of urgency of the team. The team took the un-buffered dates quite seriously and initiated actions, including over time and "crashing", to maintain these dates. Thus, when some buffer was consumed, it was only as a last resort.

The team had a good understanding of the level of buffer consumption and its meaning. Thus, the level of buffer consumption was assessed relative to progress. If the percent buffer consumed did not exceed the percent Critical Chain tasks complete, the team was generally unconcerned. Even so, the team often sought to recover buffer anyway. In at least one case, a person on the Critical Chain worked to finish

early to deprive those working on feeding task chains of some part of their buffer. This was clearly a different dynamic than that experienced under milestone-driven approaches.

Even when project buffer was consumed, this proved to be motivating rather than demoralizing. This is in contrast to the psychology that occurs when a Critical Path task slips under CPM and thereby pushes out the project end date. The team was motivated to attempt to recover buffer whenever possible. Thus, they were consciously attempting to reclaim positive variance. This is in contrast to experiences with milestone-driven approaches in which the motivation is more to finish each individual task within the allotted time. In milestone-driven approaches, there is rarely an incentive to finish a specific task earlier other than for the sake of a public display of heroism.

This motivation to recapture positive variance was very much interrelated with an understanding on the part of all team members as to the constraints of the project. Thus, the set of tasks on the Critical Chain was well understood by everyone on the team. This understanding of the Critical Chain and its relationship with other tasks provided a framework that allowed the team to evaluate and adjust its own performance.

Thus, when project buffer was consumed, it was obvious where effort needed to be expended to recover time – on the Critical Chain. As a result, it never appeared that all project tasks were of equal priority. This allowed the team to make more intelligent use of resources. For example, if the Critical Chain was falling behind and a team member working on feeder tasks was doing better than expected, that team member would volunteer to help out with the Critical Chain tasks. In addition, the understanding of project constraints led to a more productive use of overtime. For example, use of overtime to finish a feeding chain early does not make sense unless the Critical Chain tasks are going to finish early as well. Otherwise, it becomes a "hurry up and wait" scenario. Such overtime was better spent crashing the Critical Chain if it was determined that such crashing would be beneficial.

The continual emphases on aggressiveness caused the team to explore the efficient use of resources more than it might have otherwise. For example, it was realized that tightly integrating the Function-Component Test (FCT) team with the development team could yield benefits (the FCT work was not managed within the scope of the Critical Chain schedule). These benefits came in the form of test cases and test scenarios that the development team could make use of earlier than normal. This allowed the development team to save time in unit test while getting the FCT team to climb the learning curve early. This technique was not particularly new or unique, but represented an example of a more global perspective taken by the team.

The Critical Chain status collection approach also contributed to the level of aggressive while ensuring the ongoing accuracy of the schedule. The initial "knee-jerk" reaction to a status request was, "How much more time do I have?" However, Critical Chain requires the person doing the work to give an assessment of how much *more* time they believe is needed. The team grew accustomed to this subtle distinction and the result was a more continual assessment of the schedule with a reinforcement of the need to keep work from expanding into the time available. When there were problems developing, the team was therefore able to become aware of it earlier.

The Critical Chain approach also exhibited a good robustness to change. The availability of the project buffer made it unnecessary to update the schedule every time a task was either early or late, or when additional tasks were identified. Small, "must do right away tasks" were handled as noise. Rather than re-generating the schedule, the occurrence of such tasks was allowed to consume buffer. Larger tasks were deferred and later scheduled into the project buffer to maintain the focus of the team. Start and end time changes for tasks simply moved the start and end dates of subsequent tasks in or out. In addition, the team worked on tasks out of precedence order at times, but did not need to update the schedule to reflect

this. As work on the tasks progressed, the scheduling software provided accurate calculations of buffer size.

7.4.5.2 Limitations of Critical Chain

Despite these positive contributions of Critical Chain to the project, some considerable limitations were encountered as well. These limitations were centered on the application of Critical Chain in a milestone-driven environment. The distinction between 50-50 task times and the buffered project end date proved particularly difficult to manage. The initial approach of educating those who obtain project status proved to be unworkable. There were two facets to this issue.

While those assessing project status had a good working understanding of the Critical Chain approach, this status information was often propagated to those who did not. These individuals assumed that completion dates were of high confidence. This was the proper assumption for them to make in the context of the usual Software development environment. However, it conflicted with the continual movement of task start and end dates experienced when using Critical Chain. The team did not feel that the time required to special case Critical Chain was justified given that the Critical Chain application was only one of several pieces of the overall project. Thus, the team resorted to the perhaps unfortunate maintenance of two sets of books. However, the team had developed a good enough track record by this point that this second set of books could be maintained at a relatively high aggregate level.

The second facet of problems caused by the distinction between 50-50 dates and the buffered project completion dates was one of perception. The case 1 team tracked to 50-50 dates for the early portion of the development effort. It was only later during the testing phase that buffer consumption became a more serious issue. However, the adherence to the 50-50 dates made the team's performance look better than it was in reality. This motivated a desire on the part of the overall product's Project Management to re-

deploy resources to other teams who did not seem as well off. The result of this would have been a rapid destruction of project buffer that was needed to protect upcoming tasks and unknown additional tasks. Fortunately, this redeployment did not occur. However, if it had, the benefits of Critical Chain would have been quickly equalized.

A final lesson in the application of Critical Chain on this project deals with the set of tasks that the strategic buffers protect. These buffers are calculated in terms of tasks that are endogenous to the schedule. Thus, protection against uncertainty (i.e. rework) generated by exogenous tasks will not be provided. This effect had a significant and persistent effect on this case study. It was the single most significant factor that led to the consumption of project buffer and eventually led to the one-time schedule correction aimed at specifically identifying the work that the team was performing in this area.

7.4.5.3 Net Results

On the whole, Critical Chain enhanced the performance of this team. The impact of exogenous tasks would have likely been worse had Critical Chain not been used, since the lead times associated with uncovering the associated problems would have in all probability been longer. The overall success of this team was in large part based on good design and process definition as well as superb team members. One cannot achieve realistic schedules in absence of these factors. Given these foundational conditions, Critical Chain helped the team more effectively manage its time and resources and helped the team achieve an additional level of performance that may not have been possible otherwise.

7.5 Case Study 2: RACF Product Enhancement

7.5.1 Introduction

RACF (Resource Access Control Facility) is the security subsystem of the OS/390 Operating System and provides such functions as authentication and authorization facilities for a variety of operating system

resources. Additional functions include the storage of digital certificates and the association of these certificates with OS/390 user identities. This case study involved an enhancement to RACF required by the Component Broker/390 product.

Like the first case study, this one was also part of an overall larger milestone based Project Management system. However, in this case, the influences of other teams on the project performance were not as strong. As mentioned earlier, the work of this team was more of a modification that could be carried out independently of other teams including other parts of the Component Broker team. Even so, appropriate actions needed to be taken to ensure that Critical Chain fit harmoniously within the scope of the overall Project Management system of the organization.

The team consisted of approximately a dozen people and the scope of the case study included the process phases from Component Level Design through Unit Test spanning a period of approximately 8 months. Thus, this effort generated a larger array of activities than the first case study. The initial estimate of the development effort was 8,000 lines of code with the actual size being around 7,000 lines of code. The author's role in this case study was solely that of Critical Chain advisor. The author provided guidance with the application of Critical Chain and served as focal point for usage of the ProChain project scheduling software.

7.5.2 Getting Started

This case study also began with soliciting interest in the Critical Chain methodology. This stage was a bit more involved than the previous case study, since it involved more than the author's immediate team. Various presentations and tutorials on the Critical Chain methodology were given to Functional Management, Project Management, as well as the prospective team. Project Management was supportive so long as the use of Critical Chain did not jeopardize the milestones of the overall release of which

RACF project is part. Like the former case study, the team made the final decision with respect to their participation.

The RACF team is well established and owns a well-regarded product. Their experience with this product is comprehensive and long standing. The implications of 50-50 dates on the movement of interim dates was not as significant an issue for them; management had always supported the team in terms of required schedule adjustments. Nevertheless, management reaffirmed its support of the team in the context of the Critical Chain trial.

The team's concerns were focused in two areas. One concern related to quality; if Critical Chain implies more aggressive dates, will product quality be sacrificed in an attempt to achieve these dates. A second concern involved the basic uniqueness of the Critical Chain approach. Given that the team prided itself on working as aggressively as possible, how would this approach differentiate itself from prior use of bar chart scheduling. This being especially relevant, since Software Engineers tend to be optimists, if anything, in terms of their schedule estimates. The use of buffer management and its potential benefits was one obviously difference, but the notion of extracting additional productivity out of the team was controversial. These questions remained to be answered by the case study.

The meshing of high-level Project Management processes with the team's application of Critical Chain was not an issue in this case study. The higher-level Project Management processes involved were at the level of OS/390 release of which RACF is part. The team was essentially left to manage its own interim dates with the OS/390 release chiefly concerned with gross milestones such as development and function test end. As a result, any movement of interim dates would be visible only to the team.

While the entire team received education on Critical Chain, it was the team leaders who would be most involved with it. After the initial education session they worked with the author in the establishing the

details of the Critical Chain implementation. This somewhat isolated the team members from any potential distinctions of the Critical Chain approach other than the more aggressive task times. This avoided giving them something new to worry about, but may have also weakened the benefits of Critical Chain.

7.5.3 Planning

The planning phase was perhaps the most revealing aspect of this case study. In certain respects, it represented a combination of the team's normal planning process with the Critical Chain planning process. Handling the planning in this manner resulted in some benefits, but the implications for the future are not as clear. The benefits stemmed from the team being able to initially integrate some of the Critical Chain concepts into the bar chart scheduling methodology. Thus, the team did not experience productivity losses that may have been introduced by having to learn new scheduling tools.

The team initially put together a schedule using its bar chart scheduling approach. However, the 50-50 task times were used and the element of a project buffer was considered. The 50-50 dates were created by using a 30% more aggressive productivity rate than usual and then performing a sanity check on the results to ensure that it was a realistic 50-50 date.

The next step involved use of the ProChain project scheduling software. The team's bar chart schedule was converted to a standard Microsoft Project schedule and then the ProChain Critical Chain features were used. This activity served to challenge the basic assumptions of the bar chart schedule. The initial Critical Chain schedule was wildly longer than the bar chart schedule. The bar chart schedule considered only 50-50 dates and a project buffer. The Critical Chain schedule considered the additional element of Feeder Buffers. In a number of cases, the buffer for a feeding chain of tasks was longer than the time available; the time between the earliest possible finish for the chain and the critical chain successor task was of insufficient size to contain the buffer. In such case the software by default spreads out the Critical

Chain to make room for the feeder buffers thus extending the schedule. The resulting schedule gave the impression of under-aggressiveness with significant periods of time during which team members were idle. The team did not feel that this was acceptable and the author agreed. The solution was to tell ProChain not to extend the schedule to fit Feeder Buffer; in this case, ProChain will truncate the Feeder Buffers to make them fit. While this increased the level of risk on the project by insufficiently protecting Critical Chain tasks from non-critical tasks, this level of risk was deemed acceptable. It was felt that the Project Buffer was sufficiently generous to accommodate any disruptions that might occur due to an insufficient Feeder Buffer.

Even after Feeder Buffers were truncated, the Critical Chain schedule was still sufficiently longer than the bar chart schedule. The problem uncovered was one of multi-tasking. The Critical Chain schedule was de-conflicting multi-tasking that was implicit and not as visible in the bar chart schedule. In response to this, each multi-tasking scenario was carefully reviewed. In some cases, tasks were reordered or assigned to different people to avoid the multi-tasking altogether. In other cases it was decided that the multi-tasking was required and the level of risk associated with it acceptable. The resulting Critical Chain schedule then met a duration that was close to what was initially envisioned with a task structure that was somewhat different from the initial bar chart schedule. In general the team felt that this step helped them confirm and validate their bar chart schedule. It highlighted areas of potential risk and allowed modifications to be made to mitigate that risk when it was unacceptable.

The aforementioned planning actually occurred twice: once in the first month of the project and again in the fourth month of the project. The fourth month change was a result of higher-level priority shifts. The following figure depicts the final schedule:

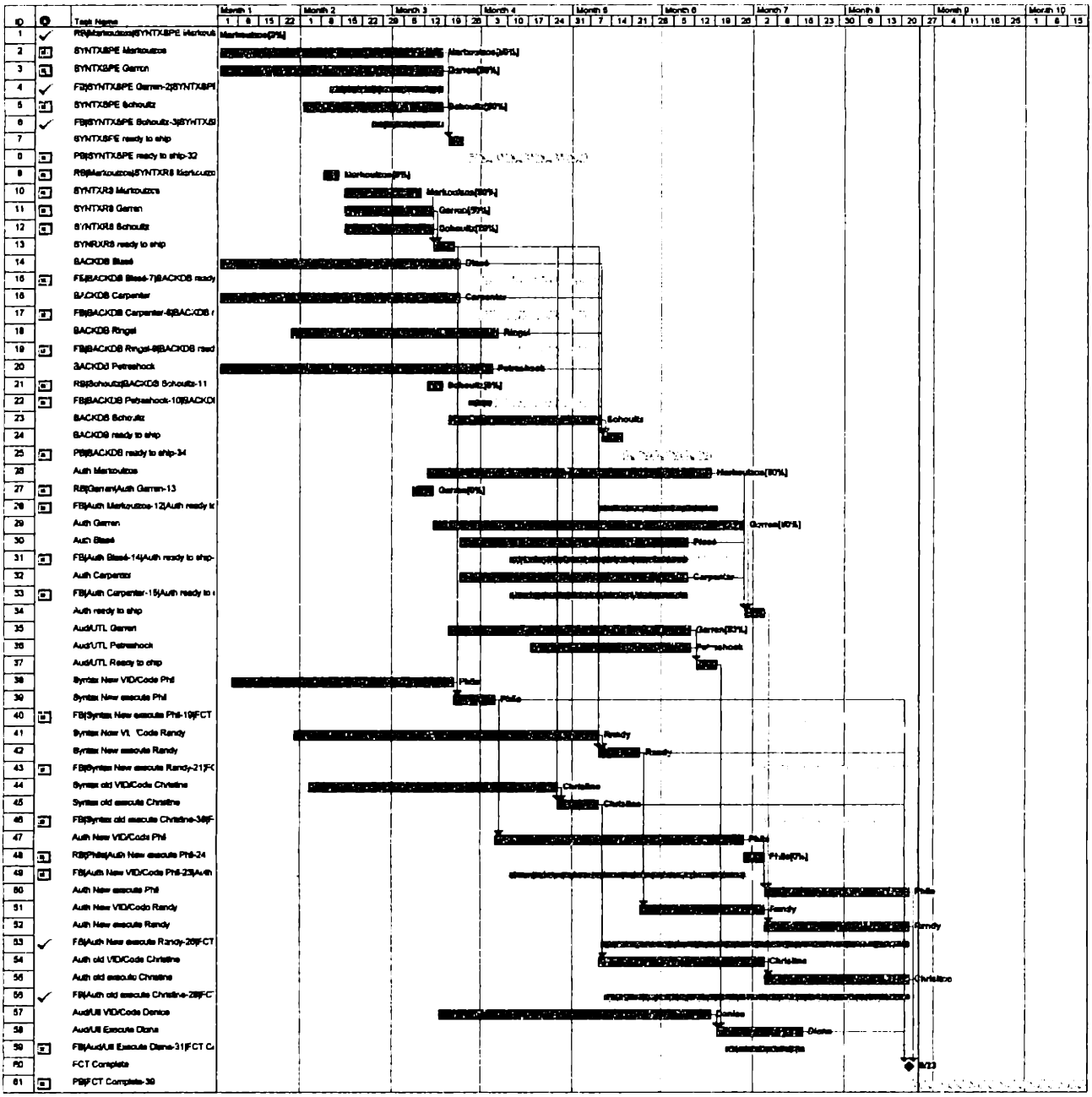


Figure 10: Case Study 2 Project Schedule

A major distinguishing feature of this schedule is that it consists of three distinct projects that comprised the team's deliverables. Tasks 7, 24, and 34 represent the Drivers into which the projects integrated. The need for 3 deliverables is at least partially a result of other organizations outside the subject team having responsibility for completion of some process activities as well as some work-items being mostly independent. Tasks 8, 25, and 61 represent the three Project Buffers (task names begin with *PB*). This

schedule represented a simple excursion into multi-project schedule. Simple, because the multiple projects could be realistically managed within a single Critical Chain file.

The Feeder Buffers are denoted by tasks whose names begin with *FB*. In the schedule above, the Feeder Buffers are represented by tasks 4, 6, 9, 15, 17, 19, 21, 28, 31, 33, 40, 43, 46, 49, 53, 56, and 59. Recall that ProChain was instructed to truncate the Feeder Buffers rather than extended the schedule. This is clearly visible in a number of the Feeder Buffers. In several places, the Feeder Buffers are seen in parallel with tasks they are protecting. The solid bar through the Feeder Buffer represents the amount of buffer consumed right at the start due to truncation. For example, Feeder Buffers 53 and 56 are entirely consumed right from the start; there is no real Feeder Buffer in this case. This is a potentially dangerous situation, since the project essentially has multiple Critical Chains! This can be dangerous in the case where things don't go as planned. The ability of the team to focus can be eroded since it may become unclear as to which chains are in the worse shape. In addition, delays imposed on the primary Critical Chain by the secondary Critical Chain can jeopardize the project end date. The Project Buffer will be consumed by the secondary Critical Chain (the feeding chain) and will therefore not be available to protect the tasks for which it was intended. Nevertheless, the team decided that this level of risk was acceptable. Had this risk been determined to be unacceptable and lengthening schedule duration not an option, further restructuring of the schedule would have been required. This may have included re-sequencing work or assigning different people to tasks.

In the creation of this schedule, a great deal of care was taken to ensure that its complexity was at the lowest possible level while still providing value, since this was a key success factor visible in the use of other scheduling techniques. This particular schedule involved five basic architectural elements (key functions) and seven process phases (CLD, MLD, Code, Unit Test, Function Test Variation Definition, Function Test Case Creation, and Function Test Execution). This alone could yield 35 activities. To

decompose the schedule further along the attribute of product or organizational architecture could quickly yield an unmanageable number of tasks.

After experimenting with some alternatives, the decision was made to maintain the decomposition of product architecture at the sub-component level. Process phases would be combined in a manner that allowed the sub-teams to have some latitude in terms of the realities of iterative development. For example, one could schedule MLD (Module Level Design) and Code as two distinct tasks, but this is not how people work. The MLD and Code phases tend to be highly iterative with progress in one leading to progress in the other. Thus, there is very little value in making these tasks separate activities. While they could have been scheduled as two parallel tasks, the resulting schedule complexity did not justify the value. The resulting process aggregation combined CLD (Component Level Design) through Unit Test, Variation Definition and Test case Creation, and left Variation Execution separate. Thus, there were three major process blocks. (The Development Phases typically consist of Component Level Design, Module Level Design, Code, and Unit Test. Test phases typically consist of Variation Definition, Testcase creation, and Variation execution).

The individual sub-teams were left with the option of creating a level of schedule decomposition below the project master schedule. A sub-team working on the aggregate CLD through Unit Test activities of a given architectural element could create a separate schedule for this work if they found it necessary or valuable. In this way, conscious use of product schedule decomposition, activity aggregation, and schedule decomposition helped maintain a manageable master schedule. It was believed that additional detail at the master schedule level simply added detail as opposed to accuracy.

A decision was also made to dis-aggregate resources. That is, every person's work on a given architectural element would be carried as a separate task on the schedule. In bar chart scheduling, this "resource view" helped the team efficiently schedule work to team members. It was carried over to the

Critical Chain schedule because it represented an element that the team found useful when thinking about the work at hand.

The team work assignments as well as task duration was performed with the input of the individual team members. While the team leaders took an initial pass of task duration, the individual team members had the opportunity to review these estimates and provide additional input. This was important to ensure that the team was not made to feel that the schedule was imposed upon them. In addition, it helped make achieving the goals set forth by the schedule more collectively owned by the team.

Like the first case study, the issue of Driver granularity was easily handled. While this team did not have a weekly Driver schedule, a mechanism to integrate code between Drivers was readily available. Thus, this Driver implementation was compatible with the application of Critical Chain.

7.5.4 Case Study Progression

Once the project schedule was defined, the entire team had the opportunity to review it. The identity of the Critical Chain was discussed and the relevant status collection differences of the Critical Chain approach were highlighted. The difference being a requirement to state more than whether things look good with respect to achieving the current task's scheduled end date. Instead, every person should indicate how may more days are required to complete the task. The subtle distinction here is that more than a cursory response is required; some extra analysis must be performed to provide such information.

As the project progressed, the team was consistently able to achieve the task 50-50 dates. Thus, the effort was completed with the protective buffers in tact – an early finish. Partially responsible for this performance was the fact that things went better than expected. The project sizing was 7,000 rather than the expected 8,000 lines of code. In addition, four extra people were available for the project – two for

development and two for function component test. Mathematically, these factors more than outweighed the additional aggressiveness represented by the 50-50 task duration times used.

7.5.5 Case Study Results

As a result of achieving task 50-50 dates, buffer management was not an issue. Unlike the Case 1 team, this team did not have the opportunity to think of project status in terms of buffer consumption versus progress. In addition, continual analysis of the Critical Chain of tasks and feeding chains of tasks was not required. Once the initial planning was complete, the project could be managed in the usual fashion. The normal milestone management process was applied to the Critical Chain schedule.

The mechanism by which the project was planned and managed was essentially more of a hybrid approach as compared to a normal Critical Chain application. It combined elements of Critical Chain with elements of the organization's normal project scheduling approach.

The approach took from Critical Chain the notion of aggressive 50-50 dates and the application of strategic project buffers. These two aspects are perhaps the most important elements of Critical Chain. A more usual mechanism to increasing aggressiveness involves utilizing more aggressive task duration alone, without application of protective buffer. The more aggressive project end date becomes the committed date. Sometimes the team is able to achieve the more aggressive schedule; like in this case study, things sometimes go better than expected. However, more often, the project will not achieve this more aggressive date due to the elements of uncertainty and the general tendency for things to be more understated then not. This highlights the role of the protective buffers as not a secret "slush fund", but a required element of a reliable schedule.

The hybrid approach also took elements of the bar chart scheduling methodology from the organization's usual Project Management approach. This not only included the key elements of creating a manageable

and scaleable schedule, but also the general project tracking mechanisms. This allowed the team to be somewhat isolated by differences introduced by the Critical Chain approach.

This case study *did* make use of Critical Chain scheduling software in the *planning* process. However, the team could forgo use of this software in the future and instead focus on further development of the hybrid approach using bar chart scheduling as the means by which the schedule is generated. However, this path could be treacherous for the same reasons that bar chart scheduling can be treacherous; bar chart scheduling does not lend itself well to identifying the constraint of the project. This combined with aggressive schedules may not be a wise direction. If things do not go well and buffer is consumed, one needs a very clear picture of where to focus attention. Project scheduling software can provided much needed assistance in this area. In addition, should buffer management be required, the project scheduling software can take the drudgery out of buffer maintenance.

The use of Critical Chain in this case study was an important part of helping the team create a schedule that was consistent and achievable, especially with respect to the tendencies to multi-task. However, the team felt that the Critical Chain approach did not change much with respect to the way they do their work on a day-to-day basis. This is understandable given the hybrid approach that was used and the fact that 50-50 dates were consistently met. One may question the merit of using the Critical Chain approach in an environment such as that in which this team operated.

Uncertainty was not as great an issue as that experienced in case study 1. The case 2 team was making an enhancement to a product with which they had great experience. They had an established Project Management methodology and software development process. As a result, product designs tended to be fairly comprehensive and task duration estimates quite reliable. For an environment in which uncertainty has been apparently so well controlled, bar chart scheduling or CPM may be quite adequate.

Perhaps a worthy challenge in such a case would be to increase the level of aggressiveness in the schedule to the point at which the teams ability to meet the scheduled task times begins to falter. Although this team felt that its approach was fairly aggressive to begin with, they indicated that the Critical Chain approach encouraged them to 'stretch.' Even with the unexpected positive factors on this project they conceded that they might not have finished quite as early had the aggressive schedules not been established up front.

In the future, the team is not certain that it will continue to use Critical Chain. It felt that the overhead of dealing with the approach did not justify the benefits, although the tools available seemed better than those available for pure CPM. However, further thought and discussion is in order to determine whether this conclusion is a statement regarding the application of the Theory of Constraints to Project Management, Critical Chain as a project scheduling methodology, or the specific Critical Chain scheduling software. This is an important distinction to be made and may allow the team to benefit from elements of the Theory of Constraints without necessarily being forced to make use of a particular software package.

In the end, the team's performance resulted in completion of the project quite early. This did not become a 'hurry up an wait' scenario. The team was able to get an early start on its next set of work. While this case study is insufficient to gain a thorough understanding of the impact of Critical Chain on product quality, the belief with respect to this case study is that it had no negative effects.

Chapter 8

Conclusions

8.1 Thesis Conclusions

The experiences of the case studies confirm Critical Chain's role in reducing the impact of Parkinson's Law. In each case, Critical Chain was a contributing factor to productivity improvement. There appears to be two factors that contribute to this result: increased aggressiveness and constraint awareness.

It has been long known that increasing schedule pressure, up to a point, can be instrumental in achieving a level of performance improvement. This heuristic has long been used on projects to squeeze an additional level of productivity out of the team. Critical Chain is merely verbalizing this heuristic by formally soliciting 50/50 task times. However, Critical Chain takes a step beyond this simple heuristic and considers the elements of uncertainty as well. It explicitly addresses this uncertainty in the form of protective buffers. The presence of such buffers is instrumental in encouraging the team to increase the level of aggressiveness in an effective manner; they represent a meaningful, global *metric*. The Case 2 team was able to derive benefits from this factor. However, the potential productivity enhancement of Critical Chain goes beyond aggressive schedules.

A key element of productivity improvement involved raising every team member's level of focus beyond the individual task to the project as a whole. To this end, constraint awareness and its implications are crucial. The identification of the Critical Chain and its role as the determinant of schedule performance serves this purpose. The role of the project buffer in assessing the performance achieved relative to the Critical Chain makes the knowledge of the constraint more powerful. The psychology changes from ensuring timely completion of specific tasks to managing the global performance of the project as indicated by the various protective buffers. The Case 1 team was able to derive benefits from this factor

that were more synergistic than those obtained by a pure increase in task time aggressiveness. Knowledge of the constraint, and its role in directing the focus of the team to generate a level of global improvement, is a powerful capability.

Another contribution of Critical Chain, which was confirmed by the case studies, is the notion of a realistic schedule. The result of increasing task duration aggressiveness in isolation can be pure folly. There is a basic reality of probabilistic task completion times that must be considered. As schedules are compressed, this becomes an increasingly important factor due to the associated increase in risk. The PERT techniques are a direct attempt to address this issue. However, Critical Chain, with its notion of the project buffer, provides a more systematic approach to managing that uncertainty while maximizing team productivity. PERT offers no clear guidelines on where specifically to allocate time in order to reduce the effects of uncertainty. Perhaps more importantly, the Critical Chain approach seems to foster a greater emphasis on teamwork and stresses global as opposed to local performance.

Critical Chain fosters improvements to the quality of the schedule as a result of its somewhat different status collection mechanism. The Case 1 team derived benefits from this. Status collection required an assessment by the person *doing* the work as to *how much longer a task requires*. This is in contrast to other status collection mechanisms that focus more on progress relative to a milestone. This subtle distinction had two benefits. First, it required an active assessment as opposed to a cursory response on the part of the team member. In addition, it emphasizes that allowing work to fill up the available time is completely unacceptable. This had the important dual benefit of continual schedule assessment while also reducing the impact of Parkinson's Law.

However, the reader must be cautious in interpreting these benefits. No project scheduling mechanism holds the secret keys to project success in terms of achieving delivery commitments. Other factors contribute at least as much to achieving such goals, and these factors were clearly evident in the case

studies. Each of the case studies involved a product architecture and design that were well defined. This provided a necessary basis for defining the project schedule.

Without a well-defined product architecture and associated design, the notion of a realistic schedule is unthinkable. Without an understanding of what constitutes the project one can't possibly provide an estimate of project duration with any degree of certainty. Yet, all too often this is exactly what is done. As evidenced by Case 1, Critical Chain can provide some much needed damping of this effect through use of the project buffer as a 'landing' error for newly discovered work. However, a critical mass of understanding is required even for this. In projects where this level of definition is simply not possible, new product development processes may be in order that allow a project to begin in an 'experimental' state and transition into a state of more formal product development with more rigorous commitments.

Also present in both case studies was a well defined process architecture and organizational architecture. The process by which successful products are developed was well defined. In addition, there was a good association of organizational structure with both product and process. These conditions provided a solid foundation upon which a project schedule could be defined, independent of the scheduling methodology used.

In addition, both case studies were careful to draw on previous experience with respect to maintaining the manageability of the schedule. Key to this was careful consideration of hierarchical schedule decomposition, product architecture decomposition, and activity aggregation. This aspect became more important in Case 2, with the wider array of activities, but is extensible and allows project scheduling methodologies to scale to vastly more complex projects.

Another important aspect of the implementation was how to handle schedule changes, especially with respect to iterative development. Case 1 defined the notion of short and long wave changes. Short wave

changes were defined as those that tend to be somewhat minor and more frequent. This can be the result of iteration between adjacent process phases or other external disruptions. These changes were handled as noise; no schedule updates were made. Currently scheduled tasks were disrupted and buffer was consumed as a result.

Long wave changes were handled differently. These tended to be a result of broader process iteration or more significant external disruptions. These changes were deferred. While this may not always be possible, deferral was the initial decision. The goal of deferral was to avoid disruption and distraction of the team as well as the productivity losses associated with multi-tasking. These changes were scheduled into the project buffer. While there is potential here of impacting the project end date, there is also great potential for early warning of this possibility.

Even without the foundational factors, which contribute to reducing complexity to the level at which a reliable schedule can be built, Critical Chain may still provide value. Its influence over team behavior may contribute to greater focus and effectiveness. However, without the foundation factors, schedule reliability may still not be acceptable. Thus, Critical Chain may offer a level of improvement, but not a complete solution to the problem of unreliable project completion dates.

Another factor, which may have contributed to the benefits of Critical Chain in the case studies, is related to the nature of Software Development itself. The team composition is of reasonably homogeneous people. There isn't quite the variation in skills that may exists on a consumer electronics team, for example, where teams may consists of Mechanical Engineers, Electrical Engineers, Industrial Engineers, Software Engineers etc. As a result, it can be much easier to recover time on the Critical Chain through help from other team members. While there may be some learning curves and some skill incompatibilities, the barriers are not as insurmountable as that which can be imagined for other teams.

This may allow Critical Chain to be much more successfully applied in environments where skills are more homogeneous.

The reader must also recognize the limitations of Critical Chain when applied within the scope of a larger milestone-driven system. In such a situation, a basic question is what goal should the Critical Chain buffer protect? Ideally it should be a date that has meaning to the customer or larger market. However, when Critical Chain is applied under the umbrella of a larger system, this may not be possible. Both case studies are examples of such a situation. An important question is whether protection of more intermediate milestones makes sense.

The concern with respect to protecting the intermediate milestone is whether it will contribute to achieving the end goal, or whether any benefits are lost. While not the ideal situation, clearly improving the reliability of the schedule leading to an intermediate milestone can improve the ability for the entire product or release to achieve its commitments. Both case studies in this thesis demonstrated this; an on time or early finish allowed team members to get started on the next project or provide assistance to other teams.

Despite the benefits of protecting an interim milestone, care must be taken not to take this argument too far. Clearly, the protection of minute project milestones can dilute the effectiveness of Critical Chain. Taken to the extreme, this would take the project back to an environment in which contingency is placed in every task. Significant intermediate milestones must be chosen. For example, Case 1 chose Development End for a project that was part of a larger overall development effort. Case 2 chose Function Test End for a project that was a portion of a larger overall Operating System release.

This thesis also raised the issue of the benefit of Critical Chain in an environment in which uncertainty is not that great an issue, such as in Case 2. If a team is able to create reliable schedules, what good is a

technique that helps protect against the ravages of uncertainty? The answer to this is related to whether there is a need to push the 'envelope.' Critical Chain can help the team feel more comfortable with setting more aggressive goals. This, in conjunction with Critical Chain's potential contributions to improved team dynamics, can help improve the chances of the team achieving those goals.

When assessing the value of the Critical Chain approach, a level of clarity needs to be maintained regarding the scope of the decision. It is all too easy to confuse a particular piece of software with a general product scheduling methodology, and to confuse project scheduling with Project Management in general. The importance of this clarity became evident in both case studies. Clearly, when deciding whether this approach is applicable for future work, concepts must be separated from methodology and implementation.

The decision to use this approach in the future is also dependent on identification of how fundamentally different it really is from extended traditional techniques being used elsewhere in IBM. For example, an experienced Project Manager found Critical Chain quite intuitive and felt that certain elements of it are already in use. So what's new here? The load-leveling feature of the CA-SuperProject software calculates what is essentially a project Critical Chain. Some organizations have combined this with the notion of a protective buffer at the end of the project. An appropriate question is whether the use of feeder buffers and the practice of formal buffer management can offer anything above this. The results of this thesis suggest that perhaps it can. There is a basic change to the team psychology introduced that may indeed offer additional benefits in the recapture of positive variance. However, a carefully designed hybrid approach may be an appropriate means to achieve the majority of benefit with minimal conflict and cost.

8.2 Recommendations for Future Work

This thesis revealed two possibilities for additional work in the area of Critical Chain and reliable project completion dates. This would involve further exploratory work involving Critical Chain itself, as well as study of alternative Integrated Product Development Processes.

Further study on Critical Chain in the context of a Software Development environment involves its potential benefits and quality impact under a variety of project estimation scenarios. While the case studies suggest that Critical Chain can help a team more effectively utilize time (even when initial cost estimates are far afield), the quality implications of this scenario are less clear. This mode of study would involve the creation of a System Dynamics model to simulate the effects of Critical Chain under a wider variety of scenarios than are possible through case studies.

New product technology development introduces challenges in terms of the requirements for reliable schedules. In such cases, exploratory work may be needed to reveal more information as to the set of activities that must be performed and their associated cost. As a result, new product development processes may be in order. Such processes could involve a prototyping phase followed by a transition into more rigorous product development. This mode of study would explore this subject further and provide a suggested product development process along with requirements for its successful application.

Bibliography

[1] Goldratt, Eliyahu M., Critical Chain, The North River Press, Great Barrington, 1997.

[2] Rodrigues, Alexandre, and Bowers, John, "System dynamics in Project Management: a comparative analysis with traditional methods", *System Dynamics Review*, Vol. 12, No. 2., pp. 121-139, (Summer 1996).

[3] Lewis. James P., Fundamentals of Project Management, American Management Association, Boston, 1995.

[4] Cooper, Kenneth G., "The $2,000 Hour: How Managers Influence Project Performance Through the Rework Cycle", *Project Management Journal*, Vol. XXV, No. 1., (March 1994).

[5] Kerzner, Harold, Project Management: A Systems Approach to Planning Scheduling and Controlling – Fourth Edition, Van Nostrand Reinhold, New York, 1992.

[6] Forrester, Jay W., "System Dynamics and the Lessons of 35 Years", Excerpt from The Systemic Basis of Policy Making in the 1990s, 1991.

[7] Senge, Peter M., The Fifth Discipline, Currency Doubleday, New York, 1990.

[8] ADI Case Study, Harvard Business Review.

[9] Newbold, Robert C., Project Management in the Fast Lane: Applying the Theory of Constraints, St. Lucie Press, Boston, 1998.

[10] Goldratt, Eliyahu M., The Goal, North River Press, Great Barrington, 1992.

[11] Goldratt, Eliyahu M., It's Not Luck, The North River Press, Great Barrington, 1994.

[12] Dettmer, H. William, Goldratt's Theory of Constraints: A Systems Approach to Continuous Improvement, ASQ Quality Press, Milwaukee, 1997.

[13] Brooks, Fredrick P. Jr., The Mythical Man-Month, Addison-Wesley, Reading, 1995.

[14] Vijayan, Jaikumar, "Users falling behind IBM's upgrade cycle for OS/390", *ComputerWorld*, Vol. 32., No. 34, p 25, (August 24, 1998).

[15] Abdel-Hamid, Tarek and Madnick, Stuart E., Software Project Dynamics: An Integrated Approach, Prentice Hall, Englewood Cliffs, 1991.

[16] Badiru, Adedeji B., and Pulu, P. Simin, Comprehensive Project Management: Integrating Optimization Models, Management Principles, and Computers, Prentice Hall, Englewood Cliffs, 1995.

[17] Meredith, Jack R., and Mantel, Samuel J. JR., Project Management: A Managerial Approach, John Wiley & Sons, New York, 1995.

[18] Inwood, Clifford, "New Managers and Missed Deadlines", Computing Canada, Vol. 21, No. 19, p 28, (September 13, 1995).

[19] Sterman, John D., Draft copy of Business Dynamics: Systems Thinking and Modeling for a Complex World, Version 1.0, May 1998.

[20] Eppenger, Steven D., and Gebala, David A., "Methods For Analyzing Design Procedures", *ASME Design Theory and Methodology*, Vol. 31, 1991.

[21] Crawley, Ed, System Architecture class notes.

[22] Henderson, Rebecca M., Clark, Kim B., "Architectural Innovation: The Reconfiguration of Existing Product Technologies and the Failure of Established Firms", *Administrative Science Quarterly*, March, 1990.