Introduction to Critical Chain Project Management

Robert C. Newbold, ProChain Solutions, Inc.

Introduction

We are all aware that virtually every business sector has become more and more competitive in recent years. There is cutthroat competition both at home and abroad, and the need for improvement embraces virtually every aspect of business. The popularity of downsizing, rightsizing and re-engineering attests to the need for change. There can be no question that this need will grow even stronger into the next century. We need more than one-time fixes; there is a clear need for processes for ongoing improvement, processes that can enable major leaps in performance.

This paper explores such a process, which is derived from an improvement methodology called "Theory of Constraints" (TOC). TOC consists of a number of common-sense tools and processes. These tools allow us to focus efforts on those few areas, called "constraints," which restrict our ability to improve. These constraints are the leverage points towards which successful improvement efforts must be directed.

The TOC concepts are well-established in manufacturing; see for example (Noreen, Smith and Mackey 1995). Application of TOC to project management is relatively new, but initial results are very promising. Completion times have been dramatically shrunk for defense R&D contracts, aircraft repair, new product development and various types of construction.

The Goal

What is a process for ongoing improvement? Intuitively it sounds like a useful idea, but we need to be very clear about what we want. A "process" is a systematic series of actions. "Ongoing" means the process can be repeated over and over. We want a process that we can use more than once; otherwise we'll have to spend all our time looking for the next bandage and hoping it works.

What is an improvement? In order to know this, we first need to understand the goal of the organization we're talking about. For what reason was it created? We could define the goal of the U.S. military to be "defense readiness." It needs to be as prepared as possible to protect U.S. interests, primarily (let's say) as a response to aggression. The goal of a school district might be to educate children who can live up to their potential to contribute to society. The goal of a public company is better bottom-line results, now and in the future. People invest in it in order to make money.

In determining an organization's performance there are two important, fundamental measurements: what goes into the company to allow it to produce, and what it produces. Since we are measuring over time, these measurements must be rates. The rate of input is called "operating expense;" the rate of output is called "throughput" (Goldratt 1992, 60-61). Some examples:

Type of Organization	*Throughput*	*Operating Expense*
U. S. military	"Defense units"	Federal tax dollars
School district	"Potential achieved"	Local tax dollars
Public company	Money made (through sales)	Expenses

EXHIBIT 1. FUNDAMENTAL MEASUREMENTS

It is not easy to measure throughput of not-for-profit organizations precisely. However, many counter-productive actions and attitudes can be avoided just by a broad understanding of the organization's throughput, by asking "What are people supposed to be achieving?" We will simplify the discussion by confining ourselves to organizations whose goal is to make money, now and in the future.

In for-profit organizations, "throughput" is the rate at which the organization generates money through sales. Usually the costs that directly depend on an individual product (also known as truly variable costs), such as raw materials and contractor prices, are subtracted from sales in order to get a value for throughput (Goldratt 1990, 19-20). Operating expense is the rate at which money is spent generating throughput. The standard bottom-line measurement "net profit" can be defined as throughput minus operating expense (Goldratt 1990, 32).

In addition to net profit, there is another common bottom-line measurement: return on assets. Return on assets is net profit, divided by total asset value. (Goldratt 1990, 23) concentrates on inventory rather than assets in this calculation, probably because of the importance of physical inventory in manufacturing.

Typically the effect of increasing assets is difficult to assess. On the one hand more capital is tied up, and on the other hand those assets represent value that the company could (through sales) turn into cash. For purposes of this discussion we'll assume that reduction of assets is not a major avenue for improvement.

We're now in a position to be much more precise about the meaning of "improvement." We want throughput to go up and operating expense to go down. To make operating expense go down, we can reduce head count, if necessary by laying people off. This avenue has a lower limit for improvement. To make throughput go up, we can sell more, or sell at higher prices; the potential here is unlimited. Whether we choose the limited or unlimited direction, we must leverage our resources to improve. We must find those key areas that are most important to focus improvement efforts. There are several ways to do this.

Leverage Points

Let's look at a hypothetical company that sells projects to develop custom hardware. This company has no problem with sales; the demand will meet the supply for the foreseeable future. Furthermore, the times and resources required to complete each project are very similar. The generic flow is shown in Exhibit 2:

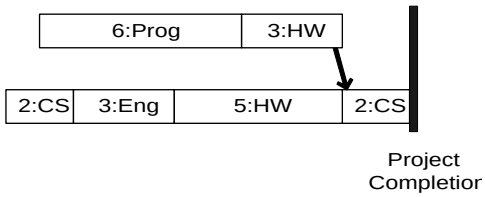

EXHIBIT 2. INITIAL PROJECT PLAN

Each rectangle represents a task. The horizontal lengths of the rectangles are proportional to the expected task durations. Inside each rectangle is printed the number of weeks the task is expected to take, and the type of person doing the work. The required resources are a customer services representative (CS), an engineer (Eng), a hardware technician (HW), and a computer programmer (Prog). Each rectangle which immediately follows another is dependent on the prior task. The arrow between 3:HW and 2:CS also represents a dependency. This means that both 3:HW and 5:HW must finish before the last 2:CS can start. The vertical bar to the right represents project completion.

How many of these projects could be completed in a year, assuming that the organization has one person of each type, and that each person has fifty productive weeks in a year? We can at least answer this theoretically, by noting that the hardware technician is going to be the most busy. Since eight weeks of her time are required for each project, about six projects can be completed per year. How can the company produce more? There are a few choices:

- Make sure the technician is focused on her work, and that she has as few distractions as possible. Every minute she is productive translates to more projects completed. Every project completed increases the bottom line.
- Make sure that everyone else produces in order to keep her busy. The programmer or engineer can't take a vacation unless there is sufficient work to keep the technician busy in the meantime.
- Hire another technician.

In fact, we have just gone through the following five-step improvement process, derived from (Goldratt 1990, 59-62):

1. **Identify the system's leverage points.** We noted that lack of hardware technician's time prevented us from making more projects, and hence more money.
2. **Exploit the leverage points.** That is, squeeze as much as possible from the technician. This might be as simple as freeing the technician from distractions or communicating her importance.
3. **Subordinate everything else to the above decisions.** Make sure everyone else is working to keep the "leverage point" busy. It only makes sense for others to produce enough to keep the technician busy, and no more. It only makes sense for the sales people to sell as much as the technician can produce.
4. **Elevate the leverage points.** That is, spend money to eliminate them. An example might be hiring another technician.

What happens if a new technician is hired? How many projects could be completed in a year? Assuming the new technician comes up to speed quickly, the new leverage point will in software department. At six weeks

per project, we now expect to be able to produce about eight projects per year. Further improvement focused on the technician will be at best useless. This suggests a fifth step:

5. **Go back to step 1; don't let inertia become a constraint.**

It looks simple and logical. Why aren't we doing it? Here we address two key reasons: local performance measurements and uncertainty. Local performance measurements are formal and informal means used to evaluate individuals. Uncertainty is embodied in Murphy's Law: if anything can go wrong, it will. The one thing we can be sure of regarding our project schedules is that they will never be followed precisely.

Local Performance Measurements

How can local performance measurements hurt the application of the five-step process? Let's use the company of Exhibit 2. What "improvements" would typical management look for? They would probably note that several people aren't working all the time. In fact, the customer service rep is spending half his time on coffee breaks. Maybe the engineer and programmer could help out with customer service. With the purchase of an automated phone system, the customer service person could be let go, and money would be saved. The bottom line has improved, and therefore the company seems better off.

The reality is that the "keep busy" mentality is a measurement. The message is "work or be laid off." It may not even be a formal measurement, but chances are everyone knows about it. If they don't, they will learn very quickly after the first layoff. What is the response to this measurement? One must keep busy. There are then two choices for someone who has insufficient work to do: slow down, or accept more work.

Slowing down is a manifestation of Parkinson's Law (Parkinson 1957, 2): the work expands to fill the time available for its completion. People become less productive. People procrastinate.

Accepting more work sounds promising. On the other hand, if people besides the technician continue to accept enough work to keep busy, uncompleted work will build up in the system. For a discussion of some of the results of this buildup of work, including increased lead times and reduced quality, see (Goldratt and Fox 1986, 32-53). For now, it's sufficient to note that excess work adds to the overall confusion. The more papers on your desk, the less likely you are to find the ones you need. The combination of Parkinson's Law and the buildup of uncompleted work means that management can't really predict when tasks will be done, or how much time is available to do more. It therefore becomes very difficult to predict how long a given project, or even a given task, will take.

There's another local performance measurement that causes problems: the necessity of keeping individual commitment dates. When people commit to finishing a task by a certain date, they are likely to be held to this date. At first this seems both sensible and inevitable. But consider what typically happens in order that people can make their personal commitments. Suppose someone has a task that should take, on average, five weeks of work. If they estimate five weeks, they'll be late at least half the time, even if no other tasks come up. That is unacceptable. In order to have a good chance of finishing on time, they'll probably estimate at least ten. They'll provide for the worst case. Of course, they're only busy half the time, so they'll have to take some other actions — either slow down, or accept more work.

Now consider how these local measurements affect the bidding process. The project bid must be competitive. Time to complete is usually a significant factor. In order to be competitive, task times are often factored down, sometimes arbitrarily. In the process project performance criteria may be compromised as well. The chances of completing on time, and the chances of satisfying the customer, suffer.

Let's look at how this affects project schedules. A possible critical path schedule for the project in Exhibit 2 is shown in Exhibit 3.

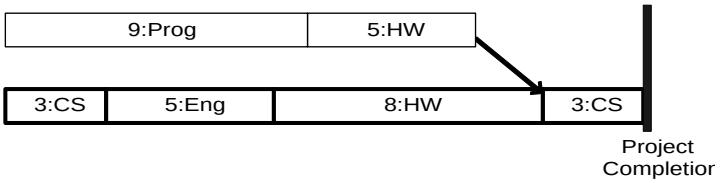

EXHIBIT 3. CRITICAL PATH SCHEDULE

The tasks with bold borders are on the critical path. Note that the individual tasks have been padded to protect their completion times. Some of that padding may have been reduced to make a competitive bid. The

overall project duration is nineteen weeks. The task durations, and hence the project completion date, are rather arbitrary.

An astute project manager might note that there is a conflict for the hardware technician's time. They might decide to resolve the conflict by creating an artificial dependency between the two technician's tasks, as follows:

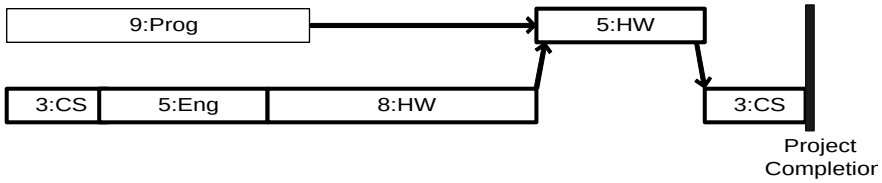

EXHIBIT 4. CRITICAL PATH SCHEDULE (MODIFIED)

This plan is probably more realistic. But now the project duration is twenty-four weeks, which is twice the duration of the critical path in Exhibit 2. Furthermore, the last task has only one week of protection; this may still be insufficient to protect the project due date from problems.

This scheduling approach also raises many questions. Suppose everyone gave estimated average completion times for tasks. Suppose we're not worried about keeping everyone busy. It seems that at this point we're more exposed than ever to the effects of Murphy's Law. How can we protect project completion dates against inevitable fluctuations? If we want to implement the five-step improvement process, if we want to establish some predictability, we will need a new approach to project planning. We need a technique that provides significantly more protection to the project commitment dates, without adding more slack time than traditional methods.

The Critical Chain: Breaking Murphy's Law

Our new technique is called "critical chain" scheduling, and is discussed in (Goldratt 1997) and (Pittman 1994). We start with the initial project layout, Exhibit 2, and completely ignore uncertainty. Exhibit 2 is not really a feasible schedule, because it has two tasks contending for the hardware technician's time. As our first scheduling step, let's resolve the resource contention.

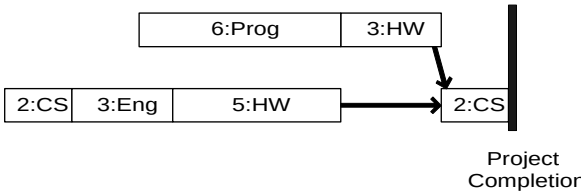

EXHIBIT 5. RESOURCE LEVELING

This schedule would be perfect if there were no uncertainty. So next we need to protect against the uncertainty. We need to protect the commitment date, because the commitment date is directly tied to throughput. In order to protect it, we must decide which tasks are responsible for the current project duration. If delayed, those tasks would make the project longer. Those tasks should therefore be considered most important. They should be protected. That set of tasks is called the "critical chain." The critical chain tasks are shown with bold outlines in Exhibit 6:

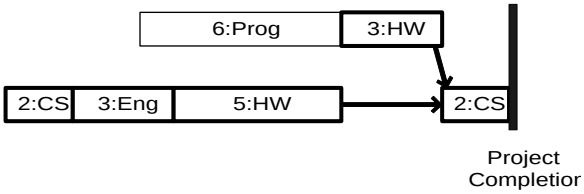

EXHIBIT 6. THE CRITICAL CHAIN

It's easy to see that a delay of any of the bold tasks would delay the project. Note that this is different from the traditional critical path in two ways: resource contention is taken into account, and tasks are placed at their late start times. Because of the resource contention the critical chain, unlike the critical path, can hop from one path to another.

Having identified the key tasks, how can we best protect the customer? We are dealing with a fixed number of resources, so the only feasible way of adding protection is by adding time. Traditionally we protect the

schedule by padding individual tasks or “spreading the slack” throughout the schedule. Using the critical chain approach we don’t protect individual tasks; we protect the project completion. We do this by means of lumps of protection, scheduled blocks of time, called buffers. The buffers look like slack and feel like slack, but they are not slack. They are necessary components of the schedule.

We need to identify the key places to put these buffers. First, since the critical chain determines the project duration, we need to protect the critical chain itself. If work is not ready for the critical chain tasks to start, the critical chain will be delayed, thus likely delaying project completion. This means we must have some protection every time a non-critical chain task feeds the critical chain. This type of protection is called a “feeding buffer”.

With the feeding buffer we have protected the critical chain from fluctuations. We haven’t yet protected the commitment date from fluctuations on the critical chain. This is done by means of a “project buffer” placed after the last-scheduled task. Exhibit 7 shows the fully-buffered schedule:

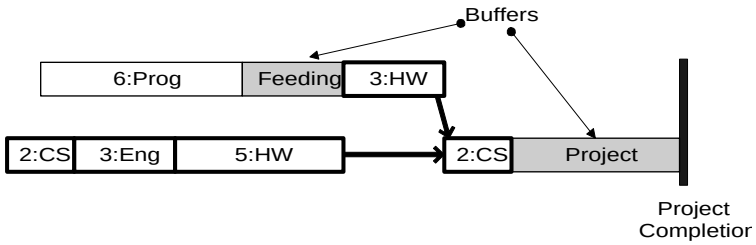

EXHIBIT 7. THE BUFFERED SCHEDULE

The feeding buffer protects the critical chain task 3:HW from uncertainty in the task 6:Prog. The project completion date is also protected by a project buffer of five weeks. This means that every task has at least five weeks of protection. By specifying average task durations and by removing the necessity to keep everyone busy, we have drastically reduced the need for people to take on multiple tasks to keep busy. This, in turn, helps reduce the normal chaos associated with fighting fires across multiple projects. If you compare this schedule with Exhibit 4, you’ll see that while the overall project duration is seven weeks shorter, we have actually gained significantly in reliability by pooling our slack into buffers.

Imagine for a moment that the critical chain tasks in Exhibit 7 are parts of an automobile, and that the uncertainty causes them to vibrate unpredictably. The buffers act as shock absorbers, so that the vibrations don’t affect the passenger, who also happens to be the customer. If we think about Exhibit 7 in this context, we’ll realize that the vibrations go both ways, left and right. Some tasks complete early, some complete late.

The critical chain approach helps projects to complete more quickly by encouraging tasks to start early. Typically, opportunities for starting tasks early are lost or ignored, because people don’t know which are high priority tasks and therefore need to be started early. When things go well, these “positive” schedule disruptions can’t accumulate. Frequent rescheduling ensures that late tasks or “negative” disruptions do accumulate, because they must be taken into account in the revised schedules. If we know which tasks are most important, i.e. the critical chain tasks, we know which tasks we would like to start early. Furthermore, the buffers allow this to happen. Suppose in Exhibit 7 the task 5:HW completes in four weeks. Unless 6:Prog is very late, the feeding buffer ensures that 3:HW can start a week early, thus speeding project along.

There is a useful refinement that can be added to the schedule. Consider what will happen if the Engineer is working on another project before the 3:Eng task starts, and that other work is delayed. That delay can delay the entire critical chain of this project, and potentially use up some of the project buffer. To avoid that, we can schedule a “wake-up call” (also known as a “resource buffer”) some time before resources are due to start their critical chain tasks. The resources are told in advance when they will be needed to perform these key tasks. This lets them know that they need to be ready for high-priority work, and adds further reliability to the critical chain schedule.

In creating a critical chain schedule, we have in fact carried out the five-step improvement process. We identified the leverage point by resolving resource contention and identifying the critical chain. We will exploit the leverage point by focusing on critical chain tasks. We have allowed everyone else to subordinate to the leverage point by inserting buffers. We know how to elevate the leverage point or “crash the project”, if we so desire: we can increase the resources available to work on critical chain tasks. Of course, this may create a new critical chain; so we must then go back to step 1.

An important question remains. If we don't worry about late tasks, how do we monitor project status? The answer is simple: we monitor how much of the buffers have been used up, compared with how much work remains on the path feeding it. For example, suppose delays have pushed completion of the final project task into the project buffer, so that only 30% of the buffer remains. If the project is 90% complete, we're probably in good shape. If it's only 50% complete, there may be a serious problem.

Conclusions

We can expect a number of benefits from the critical chain process for individual projects. Completion dates are more reliable due to the addition of buffers to the schedules. Project times-to-complete are reduced by pooling the slack into buffers. Costs typically go down as lead times go down. Lower lead times also minimize the opportunity for customers to change specifications, which is a common cause of uncertainty in projects. Because people are not rigidly held to task start and finish times, they can feel comfortable taking the time to address quality problems without fear of missing their completion dates. This reduces rework, a common and severe problem with defects discovered late in a project (Boehm 1983, 40), and helps ensure a higher-quality result.

In a multiple-project environment, there are additional benefits. If people are not measured by how busy they are or by precisely when their tasks complete, the incentive to slow down or accept multiple tasks is reduced. They are then free to follow an important rule: when you have work, finish it as quickly as possible. It is then much easier to estimate resource availability. It becomes possible to identify "constraint" resources that are leverage points for the organization to produce more projects; it even becomes possible to select such resources as strategic leverage points around which the business can be managed.

The TOC improvement tools come with a warning: there is no single individual who can implement these concepts. In an individual project, the entire project team needs to understand what is needed. In a company, the entire organization must be involved. A successful implementation requires going from a cost-oriented approach that requires attention everywhere, to a throughput-oriented approach in which everyone must work together and focus on key leverage points.

References

Boehm, Barry W. 1981. *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice Hall.
Goldratt, Dr. Eliyahu M. and Fox, Robert E. 1986. *The Race*. Croton-on-Hudson, NY: North River Press.
Goldratt, Dr. Eliyahu M. 1990. *The Haystack Syndrome*. Croton-on-Hudson, NY: North River Press.
Goldratt, Dr. Eliyahu M. 1992. *The Goal, Second Revised Edition*. Croton-on-Hudson, NY: North River Press.
Goldratt, Dr. Eliyahu M. 1997. *Critical Chain*. Great Barrington, MA: North River Press.
Noreen, Eric; Smith, Debra; and Mackey, James. 1995. *The Theory of Constraints and Its Implications for Management Accounting*. Great Barrington, MA: North River Press.
Pittman, Paul Howard. 1994. *Project Management: A More Effective Methodology for the Planning and Control of Projects*. Ann Arbor, MI: University Microfilms International.

For further information, please contact:

Focus 5 Systems Ltd
London, UK
Phone: +44 (0) 20 8959 7403
Fax: +44 (0) 20 8959 7449
Email: ProChain@Focus5.co.uk
Web: www.Focus5.co.uk

ProChain is distributed & supported in Europe by Focus 5 Systems Ltd