Contents

Ch. 1 - Modeling Basics

Send this to a friend. Click here.

| contents | previous | next |

About This Chapter

We begin this chapter by discussing an organization's need for accuracy in its project plans and schedules. We also discuss the statistically valid way to model individual tasks and chains of tasks, for the purpose of building credible project plans and schedules. Finally, we discuss important changes in management policies, measurements, and practices, changes which are required to cause ongoing improvement in the quality of the organization's project plans and schedules, and in the ability of the organization to perform to those plans and schedules.

The Ongoing Purpose Of A Project Plan

While project plans and the accompanying schedules are most useful in managing projects of any complexity, virtually all organizations use their project plans not only as tools with which to manage their projects but also as tools with which to make delivery commitments to clients. This is particularly true of organizations that serve industrial clients, because such clients always have projects of their own, which require the outputs of the projects that the supplier organizations agree to perform. Therefore, a vitally important purpose of project plans and of effective scheduling methods is to enable organizations to estimate the completion dates of the corresponding projects. This brings us to the real purpose of a project plan.

The real purpose of a project plan is to act as a predictive model of a team of resources and the project work destined for that

team. It is this model that permits the management team of the organization to make credible, competitive commitments to the organization's clients. Without accurate project plans and the accompanying schedules, the organization and its clients are exposed to paralyzing levels of uncertainty and to extreme ignorance regarding the organization's ability to complete its projects as promised.

Some may argue that the greater purpose of a project plan is to enable the project manager and the resource managers to manage the work. Perhaps, some may say even that the plan serves to enable the resources to manage their own work. But what does it mean to manage a project, if not to monitor the real state of the work relative to a valid model of that work and to make whatever adjustments are suggested by the comparison? Without an accurate, credible, feasible project plan, there can be no proper management of a project. There can be only measures aimed constantly at pushing resources relentlessly toward the earliest possible completion, measures that rarely achieve their purpose. Like a driver who has lost his way and fears being terribly late for an important meeting, a management team without a credible project plan sees no alternative but to rush all aspects of the journey that is the project. Perhaps, this is the underlying explanation for the ubiquitous PUSH-THE-TEAM style of project management.

Therefore, having a valid and credible project plan is a most important need, not only at the start of a project but during all aspects of the project's execution. A valid project plan is a beacon that guides the project to a safe landing, despite the seemingly impenetrable fog created by variability and uncertainty. This observation brings us to the question at hand: how do we begin to create valid, credible project plans?

As we stated above, a project plan is nothing more than a model of a team of resources and its work. It is the nature of models that they capture only a minute fraction of all the features of the systems that are modeled. It is the goal of the builders of models that their models should capture at least the features that are important to the builders and users of the models. At times, no, frequently it is important to limit the set of modeled features to just those few that are absolutely necessary. This economy in creating models of real systems is driven not by cost

considerations but by the need, our need, to limit the complexity of our models, for we are largely incapable of grasping and managing highly complex systems.

To understand how we can build valid project plans, we begin with the smallest component of a project, the individual task. From individual tasks, we move to chains of tasks. But throughout this process we focus less on the tasks, themselves, and more on properly modeling the vast amounts of variability that accompany virtually all product development tasks. Please note that for the sake of this discussion we assume that the project manager and the project team members are capable of creating a proper logistical network of the project -- an assumption that is rarely valid, but a necessary one for this chapter. A process with which proper logistical networks can be created is discussed in a later chapter. For now, we address the valid way to model variable quantities, such as tasks.

Variability In Task Duration

There exists a huge body of knowledge that deals with valid, effective ways to estimate and manage variability in the performance of products. Shewhart was the first to address this subject. Significant contributions to that work have been made by countless others, including Deming, Crosby, Juran, and Taguchi. Further, the strides made by many Japanese manufacturers, to manage variability in product performance, are legendary; now even many American companies and in fact companies throughout the world have made great progress in their quest to manage the variability in product performance. Thanks to Taguchi, we even have a design strategy with which we can manage variability, since we find it impossible to eliminate its causes. That strategy is known as *Robust Design*.

However, all those efforts have been applied almost exclusively to the products, not to the overall projects with which those products are developed. Yet, the components (tasks) that make up a typical product development project exhibit variability that is orders of magnitude greater than that exhibited by the components of even the poorest quality product. While our ability to estimate and manage variability in *product* performance is nothing short of miraculous, by comparison, our ability to estimate and manage the variability in the performance of our

projects can be described as primitive and ineffectual.

Ask any developer to estimate the duration of even the simplest task, and you're very likely to get a range of duration in response, not a single estimate. I always wondered what miraculous, predictive abilities some managers possessed, when in meetings they stated flatly that a project (consisting of at least several hundred tasks) would be completed on such and such a date. Months afterward I always marveled at the excuses.

The duration of every task, no matter how small, no matter how repeatable the task seems, is a statistical quantity. This means that no two repetitions of the same task will have precisely the same duration. If we could perform the same task thousands of times, the duration of each repetition would be different from that of every other repetition. For example, consider manufacturing operations, where the variability in task duration is greatly diminished once a production line is in place and running smoothly. With equal certainty, the variability in the duration of the tasks that manufacturing engineers perform, when designing, installing, and adjusting the production line, is far greater. With utmost certainty, the variability in the duration of the tasks that developers perform, when developing new products, is greatest of all. Yet, we still need to represent such variable quantities, if we are to build valid, useful project plans. Hence, we face a second question: how do we represent a single task properly?

Let me be a bit blunt; permit me to simply tell you the answer, which you can verify easily with any introductory text on probability and statistics. The statistically valid way to represent any statistical quantity is with our best estimate of the expected value of the distribution associated with that quantity. Our best estimate of the expected value is provided by the average of any recorded measurements that we might possess.

Consider the figure, below. The figure shows a rather skewed distribution, which was generated with a numerical simulation. It's not difficult to understand why the shape of the distribution shown in the figure is characteristic of those that we could expect to observe, were we to record task duration, particularly in product development organizations. The distribution is essentially truncated at its left side. This feature captures an important aspect of product development tasks. All such tasks

require some minimum amount of work. Hence, no product development task can have a zero duration. In addition, the distribution exhibits a peak. This, too, makes sense. We cannot expect such a distribution to increase constantly. A constantly increasing distribution would indicate that the instances of longest task duration would outnumber significantly the instances of shorter task duration; this would not be consistent with reality. Therefore, we should expect to observe a peak corresponding to some range of task duration. Further, the distribution has a long tail to the right. This, too, is characteristic of product development activities. The tail indicates that at times a task can take far longer than expected. As any experienced project manager can confirm, this sort of shift happens, and when it does, it is unmistakable. Therefore, the distribution shown in the figure, below, is at least a reasonable approximation of a real but unknown and unknowable distribution of task duration.

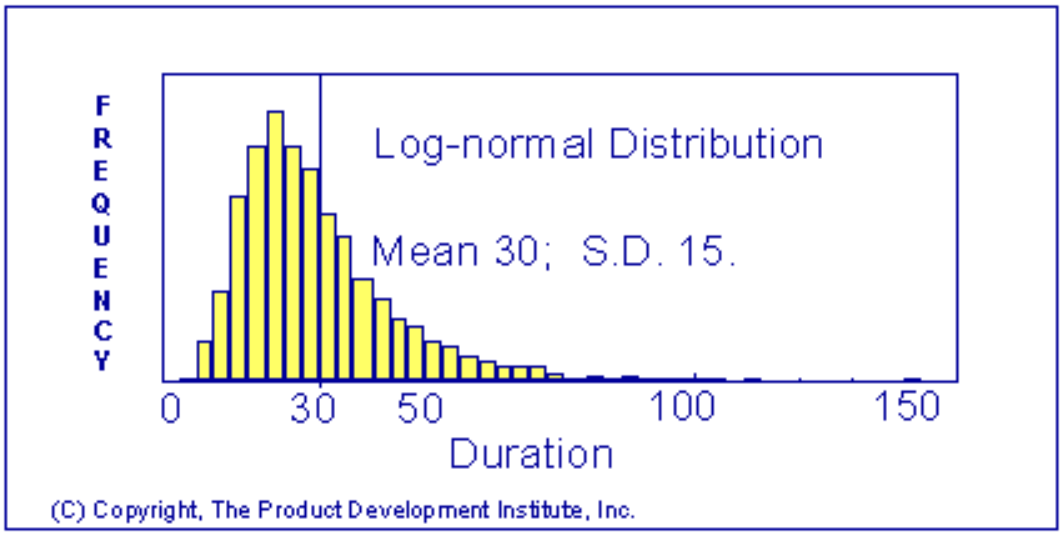


Figure 1.1: The statistically valid way to represent a single task is with the expected value of the distribution associated with the task. The expected value is best estimated with the average of whatever data we have about that distribution.

Why do I say unknown and unknowable? I say it, because we cannot know the actual distribution of any product development task, for a very pragmatic reason. When a developer performs any product development task, he/she performs that specific task for the very first time. There is almost zero repetition in product development. Each time that we undertake a product development project, we create purposely only the first copy of the product. We create only the prototype, and every prototype is different from every other prototype, by design. Consequently, we can never enjoy the luxury of repetition in product development, and we can never collect sufficient data with which to even begin to define a distribution.

For modeling purposes, i.e., for schedule creating purposes, we could choose to represent tasks, a sample distribution of which is shown in the figure (above), with the shortest possible estimates of duration. Clearly, this would yield a very short, aggressive, albeit infeasible project plan. Should we do so? Given the need that a project plan fulfills, we should not. Such a plan would not be an accurate representation of the project team's ability to complete the work of the project. The plan would not be a useful, predictive tool. It would be quite misleading, and it would expose the project (and the customer of the project) to an entirely unacceptable level of risk.

We could represent the task and all other tasks with much longer estimates of duration, say, estimates that gave the developers a 90% probability of successful completion within the estimated intervals. Should we do so? Could our customers tolerate the correspondingly long project plans and the greatly delayed commitments that such plans would cause us to make? Clearly, they could not. Therefore, we should not represent tasks with such heavily protected estimates of duration, despite the joy that such a practice might bring to our competitors.

An average estimate of task duration makes good sense, intuitively. Fortunately, the use of such average estimates of task duration is also quite correct, as any university professor of probability and statistics can confirm. Therefore, our first modeling rule is: the average estimate of duration is the statistically valid way to model a single task, within a project plan. But, if we can never collect the requisite volumes of data, how can we obtain those average estimates of task duration?

How can we obtain any credible estimates of task duration?

The best sources of such estimates of task duration are the very people who will be performing the tasks. They are the specialists, engineers, technicians, and developers who are the organization's working resources. They are the very same people without whose efforts a product development organization is little more than a pile of overhead. The developers with the greatest experience are also the ones whose estimates are most accurate. More to the point, the developers with the greatest experience are the ones who know how to perform the tasks as expediently as possible. Unfortunately, those highly experienced product developers are also the ones that many management teams have eliminated from their payrolls during the last ten years, in a misguided effort to shrink their organizations to profitability. Still, the developers who remain are the best sources of such estimates of task duration.

If we could eliminate all causes of variability in task duration, we would do so, because the absence of such variability would enable us to make uncannily precise predictions regarding the completion dates of our projects. We cannot eliminate the causes of variability. Yet, we still have to contend with our need to estimate project duration, and this requires that we estimate the duration of individual tasks. The estimates of average task duration provided by our organizations' working resources are the best that we can hope to acquire, for the purpose of building feasible project plans and making credible commitments to our clients. Now we face the next question. How do we model not individual tasks but chains of tasks?

The Expected Value Of A Sum

In the previous section we discussed the statistically correct way to represent a single task within a project plan; we found that our intuition was correct. However, since no real project ever consists of just one task, our ability to represent sequences of tasks in a valid manner is indispensable. But how can we represent sequences of tasks properly?

Again, I'm going to tell you the answer. I leave it to you to verify this second, statistically valid modeling rule: the expected value of a sum equals the sum of expected values, always! But, be

sure that you're dealing with a sum, i.e., a simple sequence of tasks. This rule is not valid when we are dealing with two or more sequences of tasks that take place in parallel and whose outputs are required for an assembly step. The impact of assembly points on project duration is addressed in a later chapter. You would be well advised to understand it thoroughly.

The next figure, below, illustrates our second modeling rule. That figure shows the distributions of completion times of two individual tasks and the distribution of the combined completion times of the two tasks. The tasks are assumed to have been performed sequentially. The distribution of each task has an expected value of 30 days. The expected value of the combined completion times of the two sequential tasks, as estimated with 1000 runs of a static Monte Carlo simulation, is 60 days; this is consistent with our second rule.

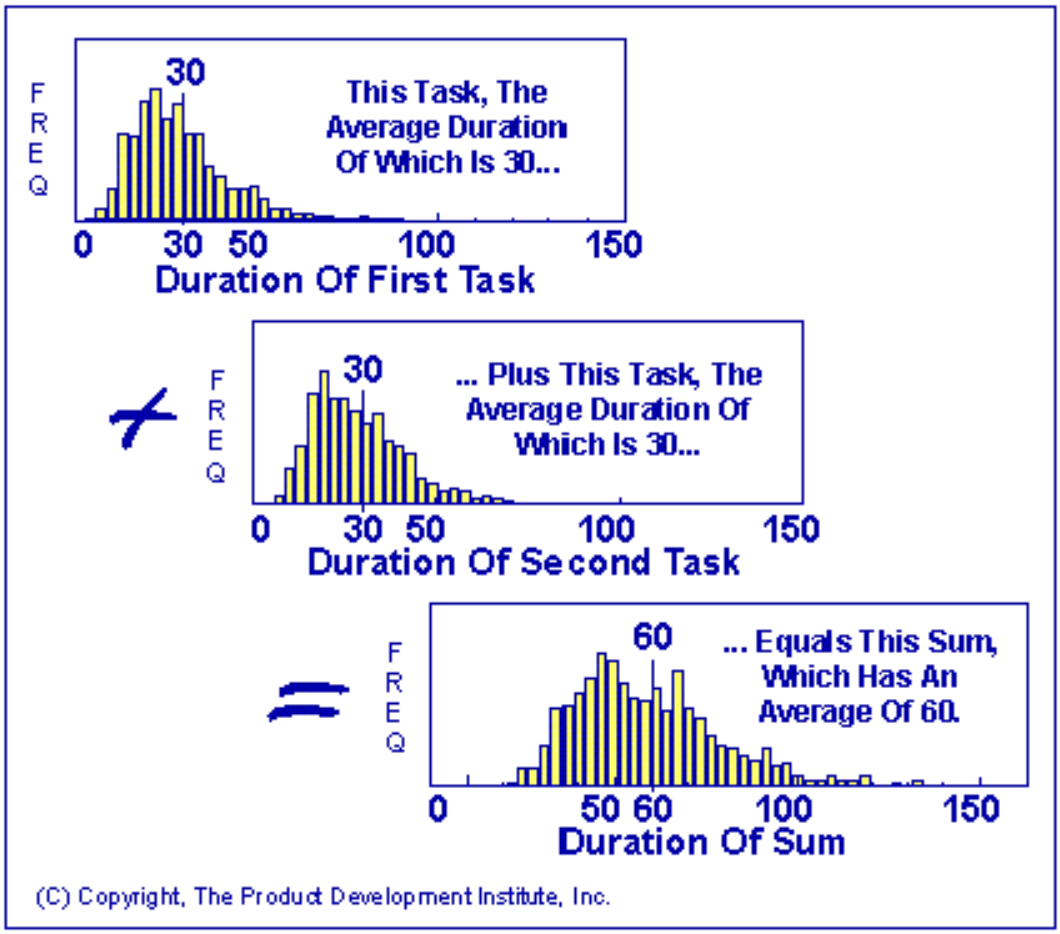


Figure 1.2: The statistically valid way to represent the overall duration of a sequence of tasks (a sum) is with the sum of expected values of the individual tasks in the sequence.

Therefore, we have two valid yet simple rules with which to begin to create credible, feasible project plans:

1) The average estimate of duration is the statistically valid way to model a single task, within a project plan.

2) The expected value of a sum equals the sum of expected values, always!

Now that we are armed with these two modeling rules, are we ready to tackle the task of modeling an entire project? Well, not quite! But we've taken an important first step by understanding our simple but valid rules. Still, there is much that we have not considered with respect to modeling individual tasks and sequences of tasks. For example, we have not discussed the assumptions that we've made regarding the behavior of developers. Nor have we discussed any process with which we can obtain constantly improving estimates of average task duration from developers. So, let's continue. We discuss first the assumptions that we've made about the behavior of developers.

Don't Bait And Switch

With our first rule, which states that we should model each task by representing it with an estimate of its average duration, we are building an important assumption into our model of the project. We are assuming that each task will be worked at a full level of effort. With our second rule, which states that the expected value of a sum equals the sum of expected values, we are building a second vital assumption into our model of the project. We are assuming that, as each developer completes his/her respective task, the developers who are slated to perform the successor tasks are able to begin their successor tasks as soon as practicable. By building these assumptions into a project plan, we accept an important responsibility. We accept responsibility for enabling the organization's developers to really work each task at a full level of effort and to really begin successor tasks as

soon as practicable.

Why is this important? It's important, because it would be disastrous for us to model the performance of a project team in one way and then to cause the team members to work in a less effective way. The project team would find it impossible to meet its due-date commitments. Further, if all our project plans were based upon that same assumption, and if all the developers of the organization were blocked from working each task in a focused manner, then many of the organization's commitments could not be met. The organization's on-time performance might plummet to 70% or even to 60%, despite the heroic efforts of many developers.

Whenever we create a model of any system, it is our responsibility to ensure that the system can live up to the assumptions that we build into the model. Otherwise, our model (our project plan) is nothing more than wishful thinking. Further, if we knowingly make commitments to paying clients on the basis of our grossly unrealistic project plans, then we are guilty of using an ancient confidence ploy, known as bait-and-switch. This is considered to be not only unethical but criminal in many nations.

"Sit There!"

"Don't Just Do Something. Sit there!" This was the advice that W. Edwards Deming often gave to the managers whom he trained. His topsy-turvy version of the cliche was designed to impress a point upon the fortunate managers: they should not tamper with the complex systems for which they were responsible, because frequent tampering would yield many unanticipated and undesirable results. Rather, the managers were expected to think about their systems, plan their actions carefully, give their systems time to respond to a change, and evaluate the response, before doing any further planning or taking any additional action.

Deming's advice was difficult for the managers to follow; it continues to be very difficult to follow, particularly for managers in product development organizations, where apparent inaction is often associated with unwillingness to do one's job or, worse, with incompetence. So, many managers in fact do do something, even when they should just "Sit there!"

The temptation to tamper with the organizational system is greatest when the plan for a new project is being created, because nearly all managers don't like the initial results of their project teams' planning efforts. The resulting project plans are nearly always much longer than the managers can tolerate. Hence, the managers are prone to a strong urge to do something, anything.

Frequently, the tampering materializes as a dismissal of the project plans that the teams put forth. In some organizations, the phrase, "That's not acceptable. It's too long!" echoes in the ears of managers and developers with such uncanny regularity that, after a while, those people say it to themselves and make arbitrary cuts in their project plans, before they show the plans to upper management. The outcome of this most inappropriate tampering with an organizational system is an unending stream of virtually useless project plans, which, rather than serving as effective predictive tools, regularly mislead the organization's leadership team into making commitments that the organization cannot meet. Such plans begin the cycle of feature shedding, improper and incomplete testing, field breakdowns, rework of existing products, and delays that cause subsequent projects to suffer the same outcome.

Still, the need that upper management feels, to have project plans whose estimates of task duration do not include unnecessary protection (also known as padding), is quite real. It would be foolish for proponents of the Theory of Constraints to ignore this need. Hence, we have a difficult conflict to resolve. We also have an opportunity to design a breakthrough solution.

The conflict, which the organization's management team feels with each new project plan, is between the management team's desire to force the project team to cut estimates of task duration and the same management team's desire to accept the plan that the project team creates. To avoid failure, the management team must be able to make a competitive due-date commitment to the organization's customer. To make a competitive due-date commitment to the organization's customer, the management team must force the project team to cut estimates of task duration, thus eliminating unnecessary protection intervals. Simultaneously, to avoid failure, the management team must

retain the project team's commitment to the success of the project. To retain the project team's commitment to the success of the project, the management team must accept the project team's proposed plan, hence the conflict.

For a conflict of this nature, that is, a conflict that coincides with the frequent occurrence of an event like the start of a new project, the wrong time to resolve it is after the conflict exists. A better strategy is to resolve such a conflict in advance of its inception. In other words, it is more effective to prevent the conflict from ever existing. But how can we achieve this seemingly difficult objective? Here, we apply some of the learning provided by the late Genrich Altshuller, the founder of the Theory for the Resolution of Inventive Problems.

Altshuller's suggestion is to first conceive of the ideal situation and then to arrive at an inventive solution that causes the (organizational) system to shift appreciably in the direction of that ideal situation. So, what would be an ideal situation for a management team, as each new project plan were created? Here's one ideal situation: most project team members voluntarily and consistently arrive at accurate estimates of the average duration of their own tasks. Let's see if this ideal situation resolves the management team's conflict.

Recall. The conflict is between the management team's desire to force the project team to cut estimates of task duration and the same management team's desire to accept the plan that the project team creates. If most members of the project team voluntarily and consistently arrive at accurate estimates of the average duration of their tasks, does the management team really need to force the team members to cut their estimates of task duration? Well, no, that need evaporates, and so does the conflict. Is the management team able to make a competitive due-date commitment to the organization's customer? Yes, so long as the management team is sincere in its negotiations with the customer and not interested in tricking the customer with the deceitful bait-and-switch tactic. Does the management team retain the project team members' commitments to the success of the project? Certainly! So, our ideal solution appears to resolve the conflict. But, does our ideal solution affect the (management + project team + customer + stockholder) system in a positive way or in a negative way, and is our ideal solution sustainable?

Let's see.

Once our ideal solution is in place, the management team enjoys accurate project plans, which can be used as effective predictive tools throughout the life of the project. This is a positive outcome and one that is sustainable indefinitely. The project team members (eventually all the developers of the organization) acquire much more ownership of their projects. This, too, is a positive outcome and one that is sustainable indefinitely. The customers of the organization are happy to see the organization become a much more reliable supplier. This is good for the customers, and it is sustainable. The stockholders see bottom-line evidence that the organization's customers are becoming more loyal. Again, this is positive and very sustainable.

So, all is well, except for one minor detail. How do we cause most project team members to voluntarily and consistently arrive at accurate estimates of the average duration of their tasks? This is a question that to date has baffled most proponents and practitioners of the Theory of Constraints in project management. There is a way, an effective way that, again, is sustainable. We'll describe the mechanics of the solution first. Then, we'll explore the changes in policies, measurements, and practices, which are required to implement this solution in an organizational system.

The mechanics of this solution are rather simple. We cause each developer to record and analyze two pieces of data for each task that the developer performs. We ask each developer to record the initial estimate of duration of each task and the actual duration of each task.

The analysis of these is equally easy. Each developer simply estimates the percentage error between the initial estimate of duration and the actual duration of each task. Each developer, then, can apply his/her own correction factor. But, it is important that each developer track his/her own initial estimates of the duration and the corresponding actual intervals.

What have we achieved with this data collection request? We have ensured that each developer gets an accurate measurement of two things, the developer's own ability to estimate task duration and the actual task duration that the developer can achieve, given the current environment of policies, measurements, and practices. In other words, we have provided each developer with an accurate, rapid feedback loop. We have enabled each developer to adjust his/her estimates of duration so as to track accurately the personal performance that the organizational environment permits. This is a vital first step.

The second step is for the management team to improve the organization's environment of policies, measurements, and practices, so that the environment permits a higher level of developer performance. The following effect/cause/effect relationships are at work here. First, the developers see what they can achieve in the current environment, with their own measurements, and they adjust their estimates to match more closely the performance that they can achieve. Second, the management team improves the environment, so that a higher level of developer performance is caused. Third, the developers detect their higher levels of personal performance, with their ongoing measurements, and they adjust their estimates accordingly. Finally, the organization's project plans begin to reflect the higher level of performance, and they become shorter and more accurate.

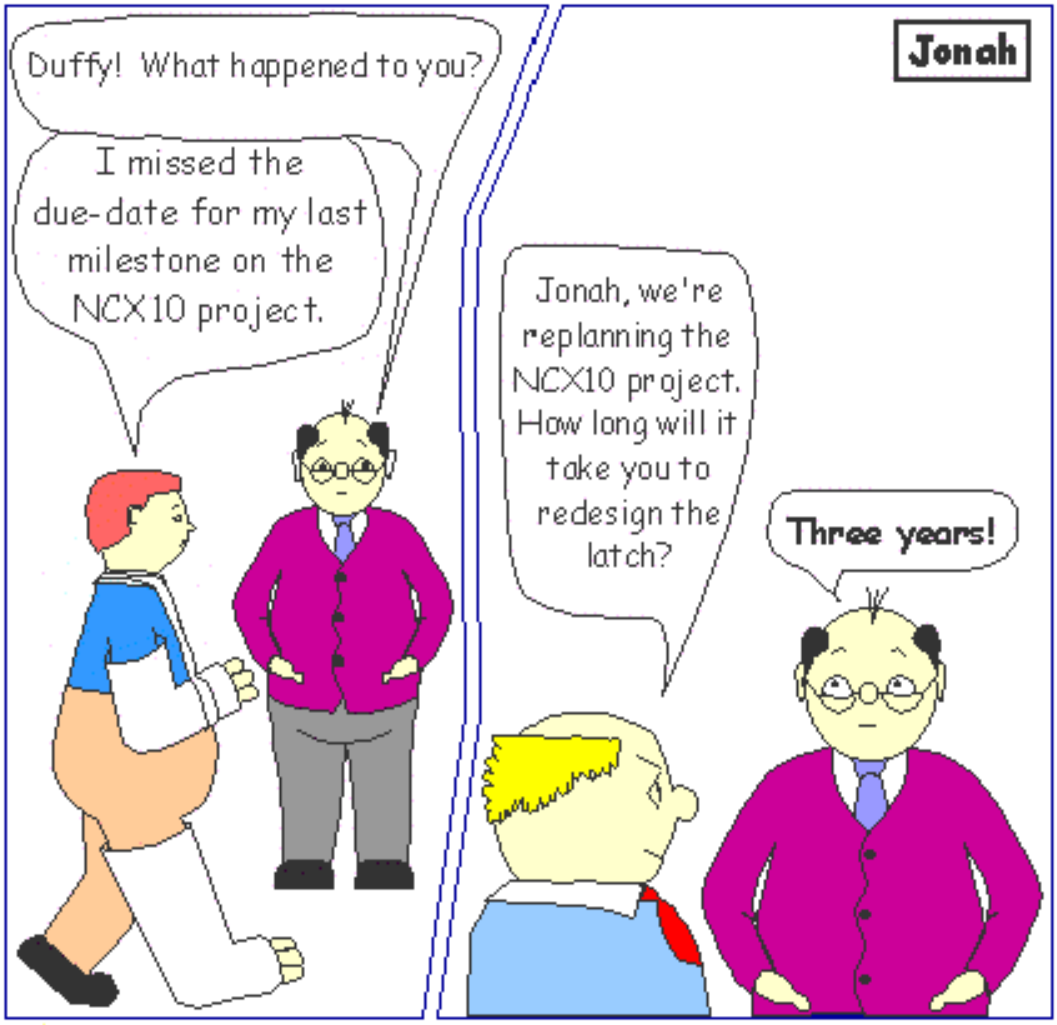
Of course, the obvious question now is: what policies, measurements, or practices must the management team change, to cause the improvement? The most important change is to help developers to achieve and maintain the proper focus, when working their tasks. Rather than encouraging and even requiring developers to share their capacity across several tasks simultaneously, the management team can and should identify the most effective sequence of tasks, for the project, and should ease the way for developers to work each of their tasks with a full level of effort. But there are other necessary conditions that the management team must achieve.

Developers are very aware of an organization's performance evaluation process. They are keenly aware of the precise measurement upon which their performance evaluation and salary treatment are based; nearly always they will strive to

optimize such a measurement. Goldratt's Critical Chain method does provide an effective set of (operational) measurements for an organization. But, for the sake of this discussion, it is much more important to know what not to use as a personal performance measurement.

Measurements Drive Behaviors

Consider an organization where the personal performance evaluation of developers is based largely upon the percentage of tasks that developers complete in accordance with their initial estimates of task duration. In other words, developers are measured on the basis of how often they meet their intermediate task due-dates. Do developers have any means of improving this measurement in ways that do not necessarily improve the organization's performance? Most certainly, they do. Were I a developer in such an organization, I would likely strive to improve my measurement by making sure that my initial estimates were sufficiently generous. This would be most certainly true, particularly if the paradigm of the managers were to assign multiple tasks to each developer and to cause frequent, severe interruptions.



Would I be behaving in a deceitful manner, by using generous estimates of task duration? No, I would not be. Rather, the use of the generous initial estimates of task duration would be required for my very survival within such an organization. Imagine what would happen to me, were I to use average estimates of task duration instead. The enforced multitasking would prevent me from working each task in a highly focused manner. Further, the frequent interruptions would block me completely from completing my tasks in a timely manner. Then, I would be held responsible for not meeting my initial, average estimates of task duration. With each performance review, I would be branded a poor performer, and I would suffer correspondingly poor salary treatment. Eventually, or not so eventually, I would be forced to leave the organization.

Now, recall the suggestion that we made earlier, to have developers track their own performance relative to their initial estimates of task duration. Is this at odds with the suggestion to not measure such performance? In fact, it would be completely

at odds with that suggestion, were it not for the fact that the performance data are collected and controlled entirely by the individual developers. If individual developers measure themselves, and if they are strongly discouraged or even forbidden from sharing their individual performance data with anyone, and if managers are entirely forbidden to even request their developers' data, then the performance data cannot be used in management's evaluation of the performance of individuals. Consequently, there exists no need for developers to optimize their performance measurements; individuals are freed to strive for accuracy in their own performance measurements, rather than feeling compelled to improve their measurements by any means possible.

By putting the organization's resource managers in charge of enabling developers to work in a more focused manner, we cause the resource managers to strive to improve real performance. By providing developers with an unbiased measurement of their ability to estimate task duration, we give developers a control mechanism with which they can continue to estimate task duration accurately, even in the presence of ongoing improvements in their real performance. But, we still have to deal with one negative possibility. Some resource managers might feel compelled to influence the developers' initial estimates of task duration. In fact, were resource managers to actually do this, they would be able to cause shorter project plans to be constructed. These shorter project plans, of course, would be inaccurate, and they would be useless as predictive tools. Therefore, we must prevent such tampering.

We can prevent the tampering in two ways. First, we can issue and enforce a policy, via the project management department. The policy is that only developers, whose estimates of duration are unbiased, may provide the task duration estimates with which the project plans are constructed. Second, the performance of the resource managers can be linked to the organization's on-time performance. Thus, if resource managers attempt to push for inaccurately short project plans, then they jeopardize their own performance measurement as they jeopardize the organization's on-time performance.

This concludes the first chapter. We have discussed two simple but statistically valid rules with which to model individual tasks

and chains (sums) of tasks. Specifically, we have discussed the rule that the best way to represent a task is with an estimate of its average duration, and we have discussed the rule that the expected duration of a sequence of tasks equals the sum of the expected values of duration of the individual tasks in the sequence. We have also discussed the need to build project plans that accurately represent the organization's ability to perform the corresponding projects. In addition, we have discussed several policies that promote, both, ongoing improvement in the accuracy of the organization's project plans and ongoing improvement in the organization's ability to deliver the corresponding projects rapidly and in a timely manner.

| contents | previous | next |

Ch. 2 - Managing Variability, Part 1

Send this to a friend. Click here.

| contents | previous | next |

About This chapter

In the previous chapter we discussed the statistically valid way to model individual tasks and chains of tasks, for the purpose of building project plans and schedules that can serve as usefully predictive tools. We also noted that the variability in the duration of product development tasks is extensive. 100% overruns of the initial estimates of task duration are not uncommon. In this chapter we discuss how to manage that variability effectively, so that we might shield our customers and our organization's bottom line from its impact, to a degree.

Variability And Chains Of Tasks

The variability in the duration of nearly all product development tasks is huge. Despite a developer's sincere effort to estimate the duration of a task precisely, much can happen during the course of the task, indeed, during the course of the project, that can easily double or even triple the actual duration of the estimated task. Even experienced developers are exposed to variability of this magnitude. In addition, a project consists of sequences of events, chains of tasks that determine the actual duration of the project and compound the variability in that duration. Therefore, understanding the impact that a chain of tasks has on variability is necessary, if we are to manage variability effectively.

The figure, below, illustrates how variability in task duration increases, as the number of sequential tasks increases. The first segment of the figure, (a), shows a numerical experiment with but

a single task. The task is modeled with a log-normal distribution, which has a mean of 10 days and a standard deviation of 5 days. The chains of tasks shown in the rest of the figure are modeled with the same distribution. The experiment was repeated 1000 times. Note that the distribution is skewed; the mean does not correspond with the peak of the distribution.

The second segment of the figure, (b), shows the results of the same numerical experiment with two tasks in the sequence. The mean of the resulting distribution is 20. This is entirely consistent with the second rule that we discussed in Chapter 1. The resulting distribution, too, is skewed. But it is skewed less than that of the individual tasks. Note also that the variability of the duration of the two-task chain is considerably greater than that of the single task.

The third segment of the figure, (c), shows the results of the same numerical experiment with four tasks in the chain. Note, again, that the mean of the overall distribution (40 days) coincides with the end of the chain of tasks. Note too that the distribution is more symmetrical than the earlier distributions and that the variability is much greater than that of the individual tasks.

The fourth segment of the figure, (d), shows the results of the same numerical experiment with eight tasks in the chain. Again, the mean of the overall distribution (80 days) coincides exactly with the end of the chain of tasks. In addition, the overall distribution is now nearly symmetrical, and its variability is rather large in comparison to that of the individual tasks.

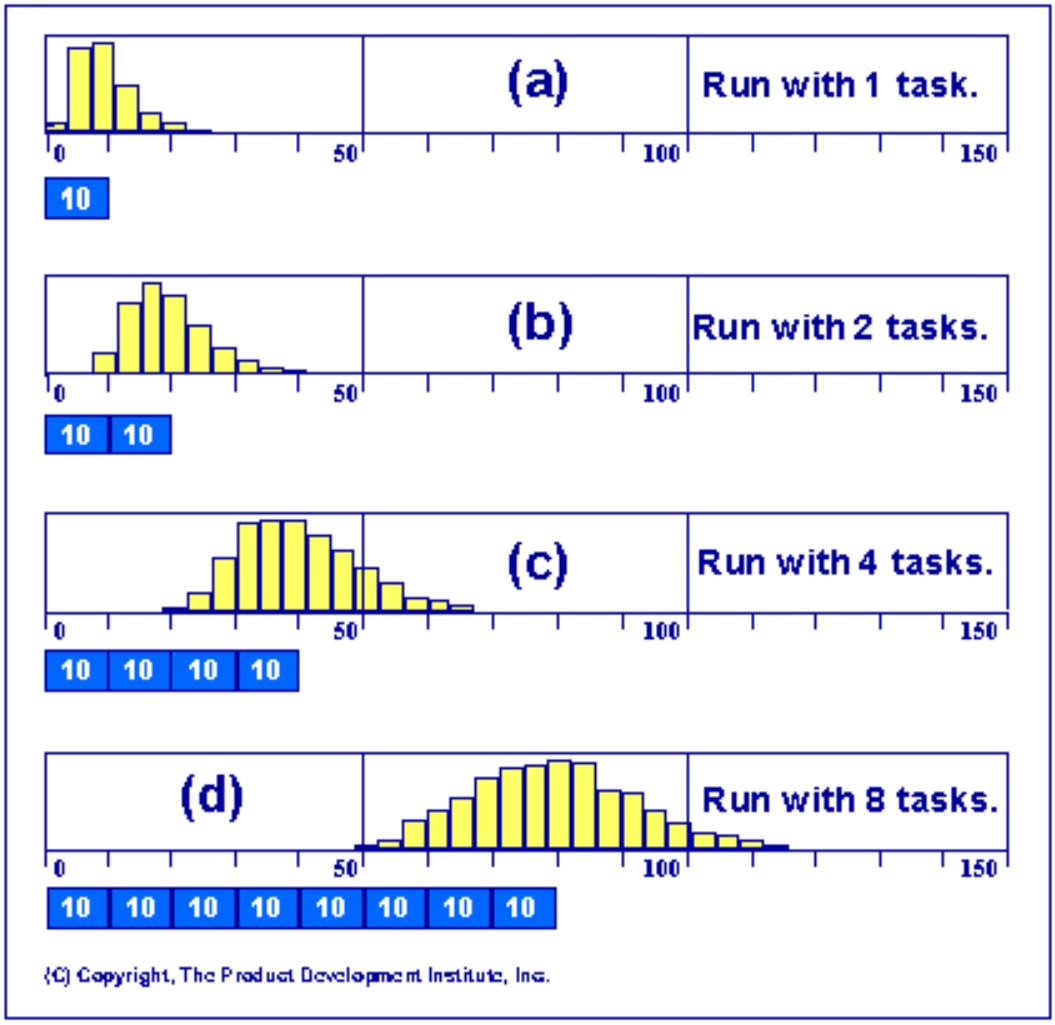


Figure 2.1: The Central Limit Theorem guarantees that, in the limit, the distribution of completion times of the sum approaches the normal distribution. Numerical experiments demonstrate that if a "sum" of tasks includes a sufficient number of terms (as few as 8 or 10), then the distribution of the overall completion times of the sum closely approximates a normal distribution.

What does all this mean? It means that any project that consists of the same number of similar tasks is certain to exhibit the same, extensive amount of variability. Of course, virtually all real projects are much longer than these simple examples. Consequently, the variability in the duration of real projects is even greater. So, should we throw our hands in the air and declare defeat, or should we make some attempt to manage all this variability? In fact, what does it mean, to manage variability?

Managing variability means limiting the adverse effects that variability creates. It's clear that we cannot eliminate the causes of variability. Therefore, our only remaining option is to limit its

impact, but on what, or on whom? What are we really protecting?

We are protecting our customers! Our most important reason for managing the variability in the duration of our projects is to protect our customers from its adverse effects. This, in turn, protects our organization's reputation as a reliable supplier and gives our customers yet another reason to spend their money with us, rather than spending it with our competitors.

One method of shielding our customers from the effects of all this variability might be to ensure the timely completion of every individual task. We might try to do this by protecting every task in the project. In fact, the widely accepted method of tracking progress relative to a schedule of milestones is an example of this approach.

However, there's a severe penalty to be paid, for using this method. The protection required to protect all those tasks adds up to quite a long interval. This is illustrated in the next figure, which shows the same chain of eight tasks, with each task protected to approximately a 90% probability of successful completion within the indicated interval.

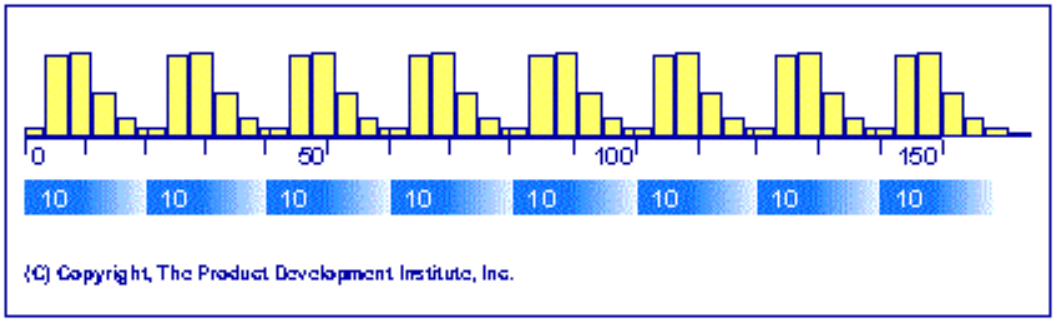


Figure 2.2: The milestone tracking approach to managing variability frequently leads to bloated schedules and non competitive proposals.

It's very doubtful that any organization could be competitive in today's business environment, if the organization's managers attempted to manage variability in this manner. Most managers know this, of course. This is why they struggle with a conflict, between being able to present a competitive proposal to a customer and protecting that same customer from the adverse effects of the inevitable variability in project duration.

There's another problem with the milestone method of project management. It causes us to model individual tasks in a statistically incorrect manner. Recall. In Chapter 1 we observed that the statistically valid method for modeling individual tasks was with estimates of the average task duration. The milestone method, clearly, is at odds with our statistically, mathematically valid approach.

We need a better alternative for managing variability in project duration, because choosing to not manage it can be very damaging. Consider the next figure. Imagine that the chain of tasks shown in the figure is the only chain in the project. Therefore, it's the longest chain. Imagine, too, that the duration of the various tasks has been estimated using average estimates, as our first modeling rule suggests.

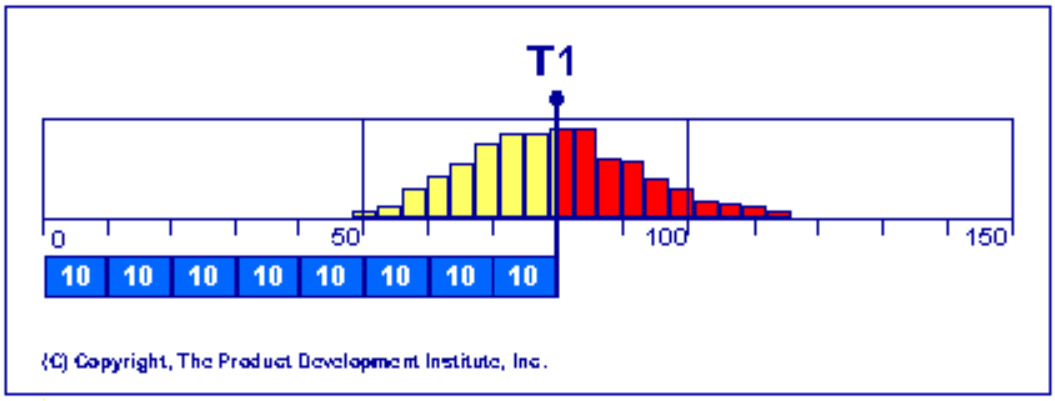


Figure 2.3: There is approximately a 50% probability that the chain of tasks will not be completed on or before time T1. Will your customers tolerate this sort of on-time performance for very long?

The accompanying distribution illustrates that there's a 50% percent probability that the project can be completed by time T1. Inevitably, this means that there is also a 50% probability that the project will be completed later than time T1. Should we make a commitment, on behalf of the organization, to deliver the project on or before time T1? If we were to make such a commitment, then essentially we would be choosing to not manage the variability in the duration of the project. Yet, this is precisely how most project commitments are made.

The managers making such commitments nearly always look for the date of the last scheduled workday of a newly planned project, and they promise to deliver the completed project on that day. By doing so, such management teams expose their customers to an incredible amount unnecessary risk; their organization steadily earns a reputation as an unreliable supplier, since at least half its projects are delivered late or with significantly reduced scope.

The choice to not manage variability is not a viable, long-term option. The impact of such a choice on customers, on the organization's reputation, and ultimately on the organization's profitability is devastating. Under the best possible circumstances, such a business can claim only that it is no worse than its competitors. This is hardly a claim that sets the business apart from its competitors, in the minds of customers. We must protect customers from the tremendous amount of variability in the duration of our projects, and we need not do so by creating bloated project plans and the resulting non competitive schedules, as the milestone tracking method causes us to do.

The logical, effective, and statistically valid way to manage variability is to protect the delivery to the customer from all the variability in the project, with a PROJECT BUFFER. This is illustrated in the next figure.

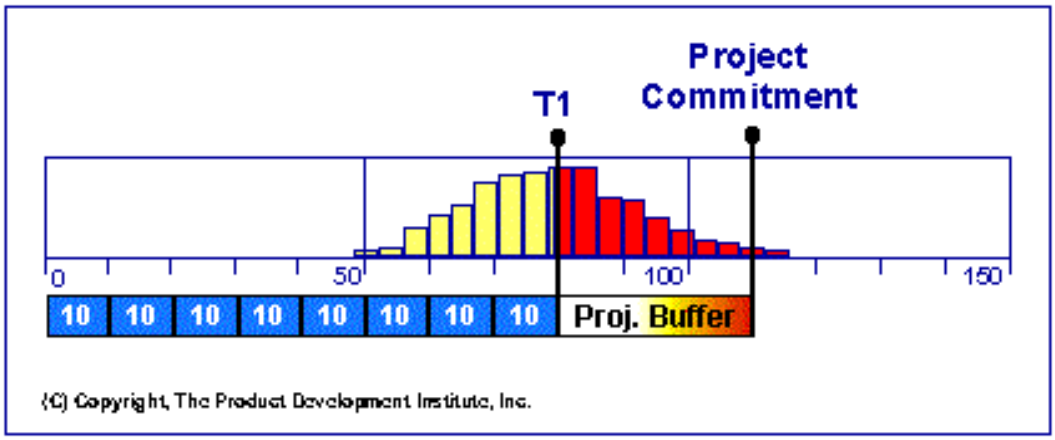


Figure 2.4: Project Buffers protect customers, from all the causes of variability that a management team cannot possibly eliminate. If we remove the project buffers from our project plans, then we are exposing our customers to significant risk unnecessarily.

Project Buffers shield customers from the many sources of variability in the duration of our projects. With a project buffer, a management team can easily make a delivery commitment, on behalf of the business, and can do so with a 90% probability of success, or greater.

Further, making such a high-reliability delivery commitment can be accomplished without the bloated project plans and non competitive schedules that the milestone tracking method creates. However, great care must be taken to ensure that at all times project plans reflect the organization's real ability to perform its projects. The assumptions that we discussed in Chapter 1, regarding the management changes required to let developers work in a focused manner and also required to let successor tasks begin as soon as practicable, must be made valid before this method can be used safely. Management's failure to make the required changes well in advance of the use of this method will certainly cause the organization to take on commitments that it cannot possibly keep.

There's an even greater benefit that a project buffer provides. With the milestone tracking method, it is extremely difficult, if not impossible, to recover any of the protection time that inevitably is embedded in the individual estimates of task duration. Not so with a project buffer! In the very likely event that the project is completed before the project buffer is entirely consumed, the buffer can be recovered easily, and the project can be delivered

ahead of schedule.

In addition, by monitoring the amount of buffer remaining, as well as the total number of workdays required to complete the project's longest chain, the management team gets an ongoing indication of the risk to which the project's customer is exposed. In a very real sense, the project buffer acts as a risk transducer for the management team. This most valuable measurement function is the core of a real-time feedback system for the entire project team as well as for the management team. It is also the greatest benefit that a project buffer provides. It can and does act as an early warning system, shining a spotlight on trouble areas well before it's too late for the management team and the project team to take corrective measures.

In this chapter we've taken the first step toward managing effectively the inevitable variability in project duration. We've seen that aggregating protection, with the project buffer, not only gives us a more effective means of shielding customers from the adverse effects of variability but also provides a real-time measurement of the risk to which customers are exposed. Of course, we can use this risk measurement to effect, by taking appropriate action when the measurement suggests that such action is necessary.

In the next chapter we will see how, by neglecting to manage variability as we attempt to exploit Concurrent Engineering Methods, we often put in place the foundation for the failure of our own projects well before those projects are begun.

| contents | previous | next |

PDI

Ch. 3 - Managing Variability, Part 2

Send this to a friend. Click here.

| contents | previous | next |

About This Chapter

In the previous chapters we learned how to properly model individual tasks and chains of tasks. We also learned how to counter the effects of variability in task duration and in the duration of chains of tasks. We saw the very real need to aggregate protection for the longest chain of tasks, in the form of a project buffer.

However, no project consists of a single chain of tasks. All real projects consist of multiple chains, only one of which can be the longest. All other chains, of course, happen in parallel with that longest chain. Therefore, it is in our interest to explore the effects of these parallel chains on the variability in the overall duration of a project. To ignore these would be to stick our heads in the proverbial sand.

A Very Simple Project

Figure 3.1, below, shows a very simple project network. The network consists of one long chain, and two feeding chains. The last task of the long chain, task number 8, is an assembly task. It's start requires the output generated by the first seven tasks of the long chain and also the outputs generated by the two feeding tasks. The absence of any of these outputs precludes the start of the assembly task.

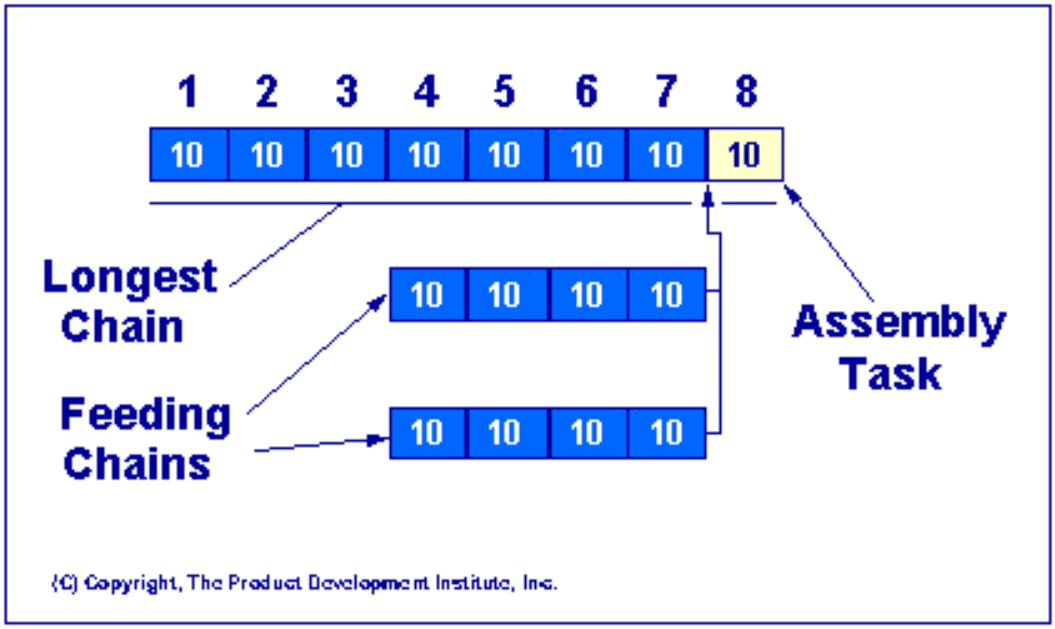


Figure 3.1: Task 8 is an assembly task. It cannot be started until the first seven tasks of the long chain and the two feeding chains are completed. Real projects often exhibit a number of similar dependencies.

Is this a real project? No, not by a long shot! It is far too simple, to be a real project. However, while it is simple, it is also realistic in its structure, and its simplicity serves us well, by letting us examine the impact of feeding chains on the distribution of completion times. If that impact is significant with this simple project network, then we can rest assured that the impact of a greater number feeding chains, as we can expect to see in real project networks, is even more pronounced.

As we did in the first two chapters, we use this simple project network in a static Monte Carlo analysis, i.e., we use it in a statistical simulation. A partial set of the results of the simulation is shown in Figure 3.2, below.

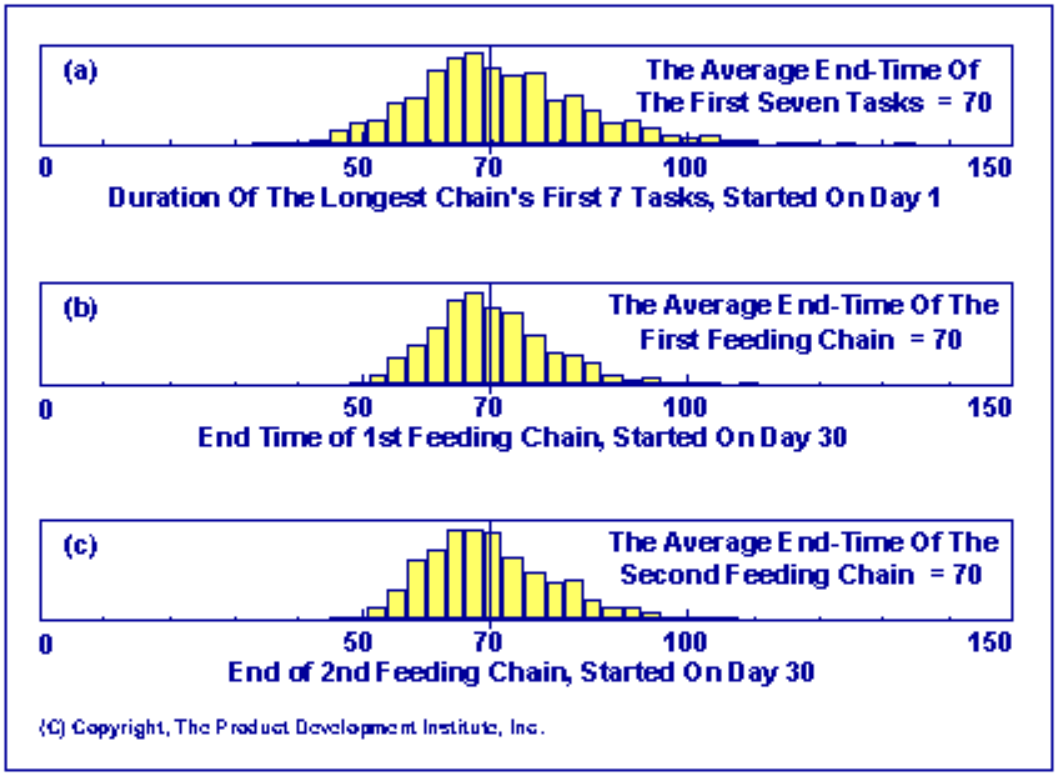


Figure 3.2: Distributions of completion times of each of the chains of tasks that feed the assembly task.

Figure 3.2 shows the distributions of completion times for each of the chains of tasks that feed the assembly task. Figure 3.2-a shows the distribution of completion times for the first seven tasks of the longest chain. That chain is assumed to have started on day 1 of the project. As we might expect, due to our rule about the average of a sum, the average completion time is 70.

Figures 3.2-b and 3.2-c show the distributions of completion times for each of the two shorter chains. These are assumed to have started on day 30, as scheduled. Since each of the shorter chains has an expected duration of 40, these exhibit an average completion time of 70. This brings us to the question at hand.

The Effect Of Parallel Chains

If the chain consisting of the first seven tasks of the longest chain finishes on day 70, on average, and if each of the two feeding chains also finishes on day 70, on average, then what might we expect as the average start time of the assembly task? To help you answer this question for yourself, I've provided the exercise

shown in Figure 3.3, below.

Before you can perform the exercise, you'll have to print the figure and cut out the small triangle. The triangle is a spinner, which, when used in conjunction with the circle, becomes a random number generator. Click on the figure to have it show up in a separate window, from which you can print it more conveniently.

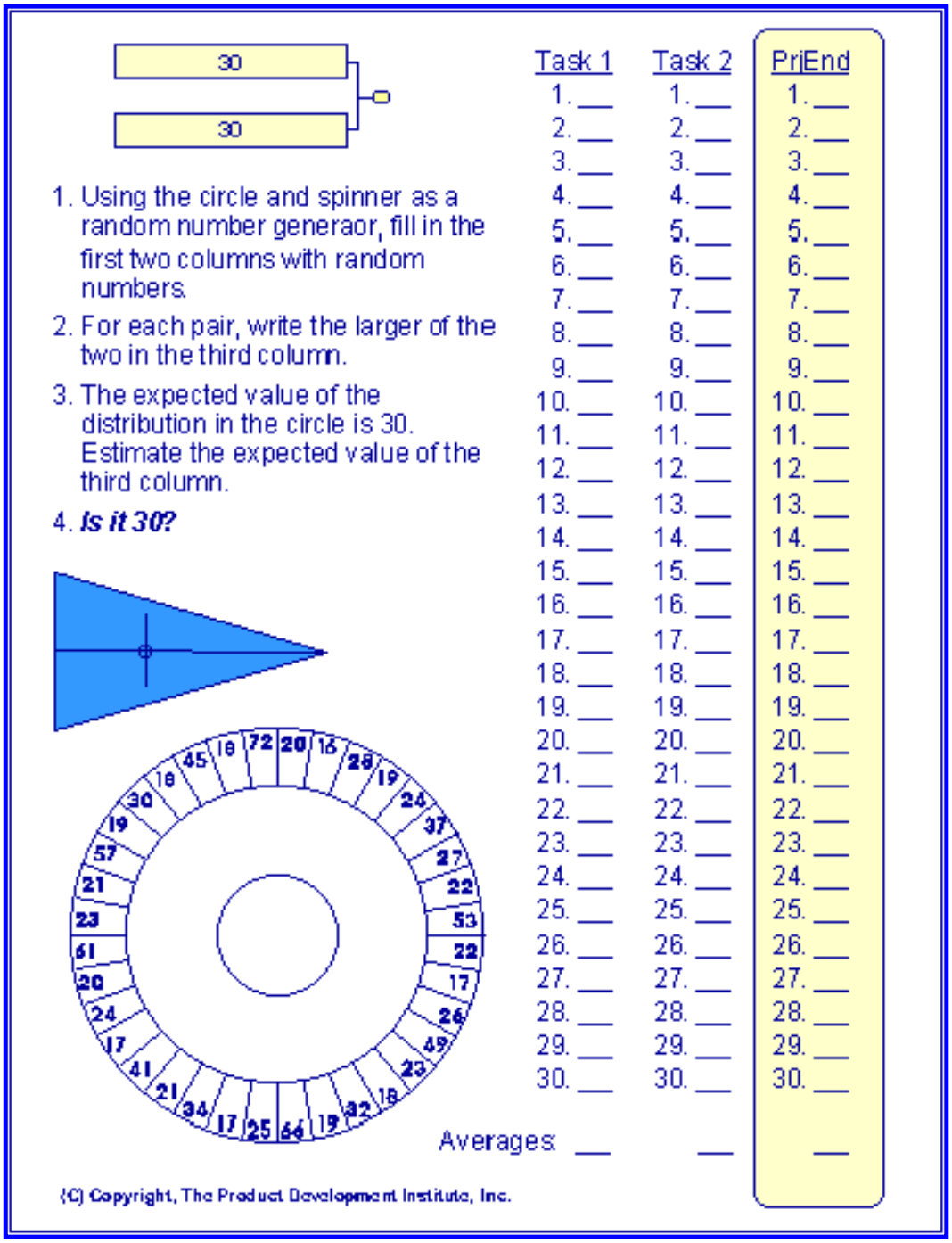


Figure 3.3: This exercise demonstrates the impact of parallel chains on the distribution of completion times of a project.

With the exercise depicted in Figure 3.3, you will be performing a static Monte Carlo analysis of the simple project. The project consists of two tasks only. But, the two tasks are in parallel. This means that both tasks must be completed, before the tiny project can be considered completed.

The distribution with which each task is modeled has been mapped into the circle. That distribution has an average of 30. This is the expected value with which each task is represented. Note, too, that the largest number represented in the circle is 72. Thus, either task could take as long as 72 days during the course of your simulation.

To perform the exercise, cut out the small triangle and insert a pin through its centroid - the centroid has been marked with a small circle. It may help to glue the triangle to an index card, before using it as a spinner. Once your spinner is ready, simply center it over the circle and give it a flick of the finger. Make sure that it spins at least a couple of times, to ensure an unbiased simulation.

Using your spinner and circle, generate 30 random numbers for each task. Record your random numbers in the appropriate column for each task. By generating 30 random numbers for each task, you will be simulating the completion of the project 30 times.

Once the first two columns are filled, record the larger of each ordered pair in the corresponding spot of the third column. This larger duration represents the actual project duration for the corresponding simulation. Is this valid? It certainly is. Recall. The two tasks are done in parallel. The project cannot be considered completed until both tasks are completed. Therefore, the project always takes as long as the longer task.

If you were to record not 30 random numbers for each task but, say, 1000 random numbers, then you would be able to estimate the expected duration of each task with precision. But, this is not necessary. The distribution with which each task is modeled has

an expected value of 30 days. Consider this as given information (which you can certainly verify independently, if you choose to do so). To estimate the effect of the parallel tasks, on the duration of this little project, simply calculate the average of the third column.

Is the average completion time of this small project 30 days?

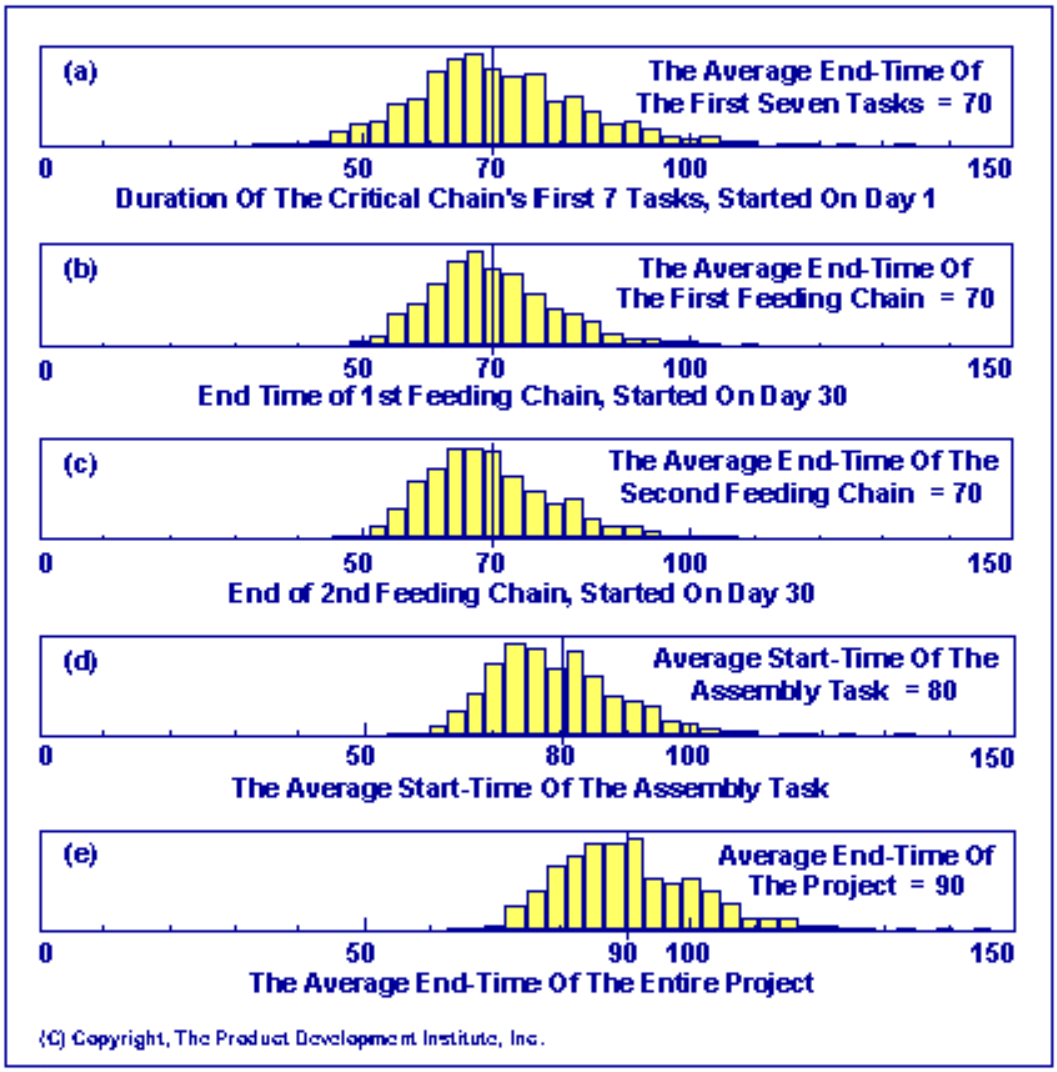


Figure 3.4: Distributions of completion times for (a) the first seven tasks of the long chain, (b) the distribution of completion times for the first feeding chain, (c) the distribution of completion times for the second feeding chain, (d) the distribution of start times of the assembly task, and (e) the distribution of completion times of the entire project.

If you've done the exercise, then you can understand Figure 3.4, above, more readily. Figure 3.4 shows the complete set of results from the analysis of the original, small project network. Sections (a), (b), and (c) are identical to those of Figure 3.2. Section (d) of Figure 3.4, however, is somewhat interesting. That section shows the distribution of the average start times of the assembly task. Notice that the average start time of the assembly task is not 70, even though each of the chains of tasks that feed the assembly task does have an average completion time of 70.

The discrepancy is caused by the dependency that the assembly task creates among the three chains of tasks. This dependency acts as a high-pass filter. The two earlier completions are always wasted, in the sense that the assembly task cannot take advantage of one or two early completions. The assembly task can begin only when the outputs of all three chains of tasks are ready. Consequently, the start of the assembly task always coincides with the latest completion of the tasks from which it gets its inputs. The average start time of the assembly task is not day 70 but day 80. The average duration of the project, in turn, is 90 days. This is somewhat later than we might have concluded from a visual inspection of the project plan. This impact is also a bit counterintuitive.

The next figure, 3.5, shows the full impact of the parallel chains on the distribution of completion times of the entire project. Time T1, shown in that figure, corresponds to the apparent end of the project. This is the time for which nearly all project commitments are made in industry today.

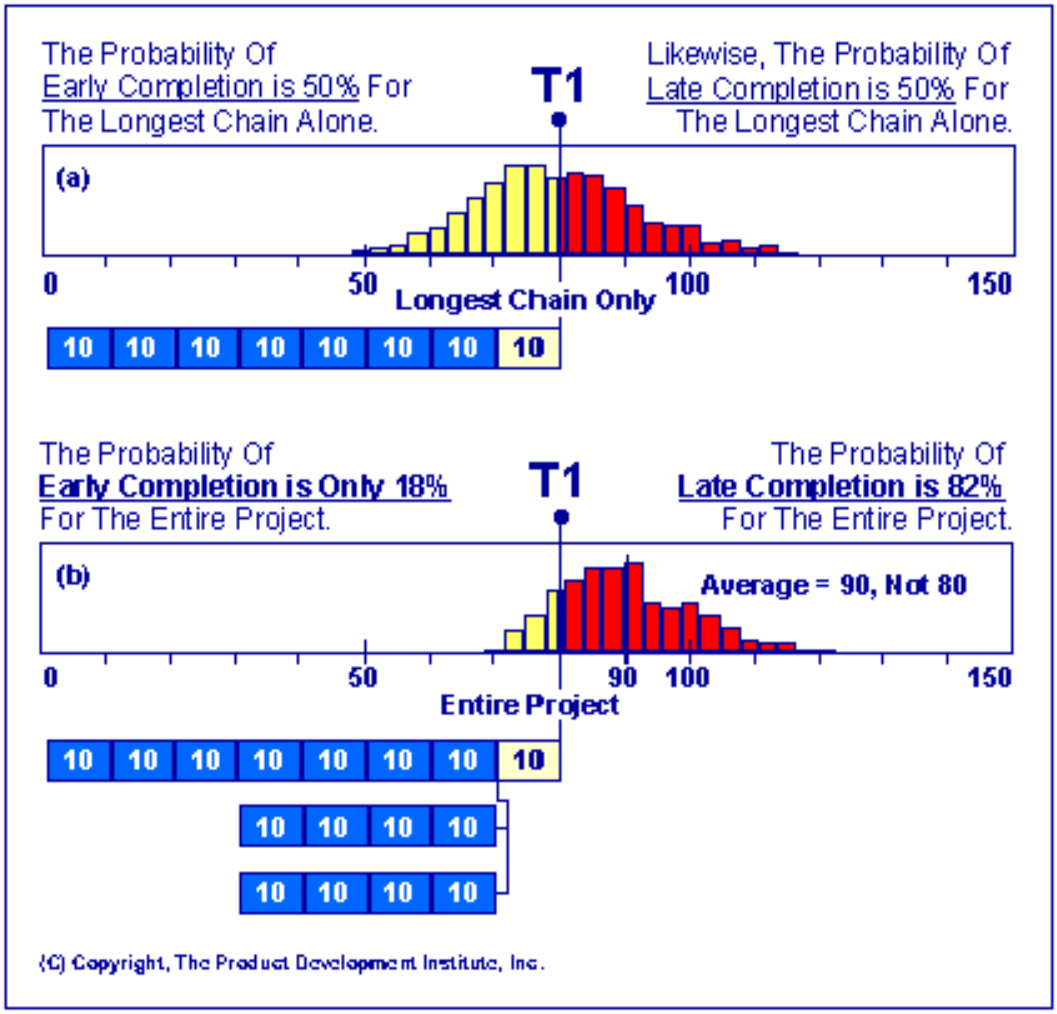


Figure 3.5: Probability of early completion, shown in yellow, for (a) the longest chain alone and (b) the full network, including the two parallel feeding chains.

A comparison of Figures 3.5-a and 3.5-b is rather revealing. Figure 3.5-a shows that the probability of completing the long chain alone, on or before the apparent end of the project, is approximately 50%. However, the presence of the two parallel chains and the dependency caused by the assembly task reduce this probability from 50% to only 18%. In other words, if such a project were promised to a customer for a time that corresponded to the apparent end of the project, then the odds would be 4-to-1 against the timely delivery of the project. The on-time performance that such odds would permit could hardly be considered stellar.

Now, consider that these results were obtained with a very simple project network. Our real projects have not just two parallel feeding chains but as many as a dozen parallel feeding

chains, all usually coming together at some assembly task near the end of the project. In fact, we go out of our way to create parallel feeding chains, because doing so lets us create shorter project networks. We've even elevated this to the status of "best practice." We call it Concurrent Engineering.

By making a commitment to deliver a project on or before the date that corresponds to the apparent end of the project, we put in place most certainly the causes of our own failure. The underlying mathematical cause of this unfortunate set of circumstances is illustrated in Figure 3.6, below.

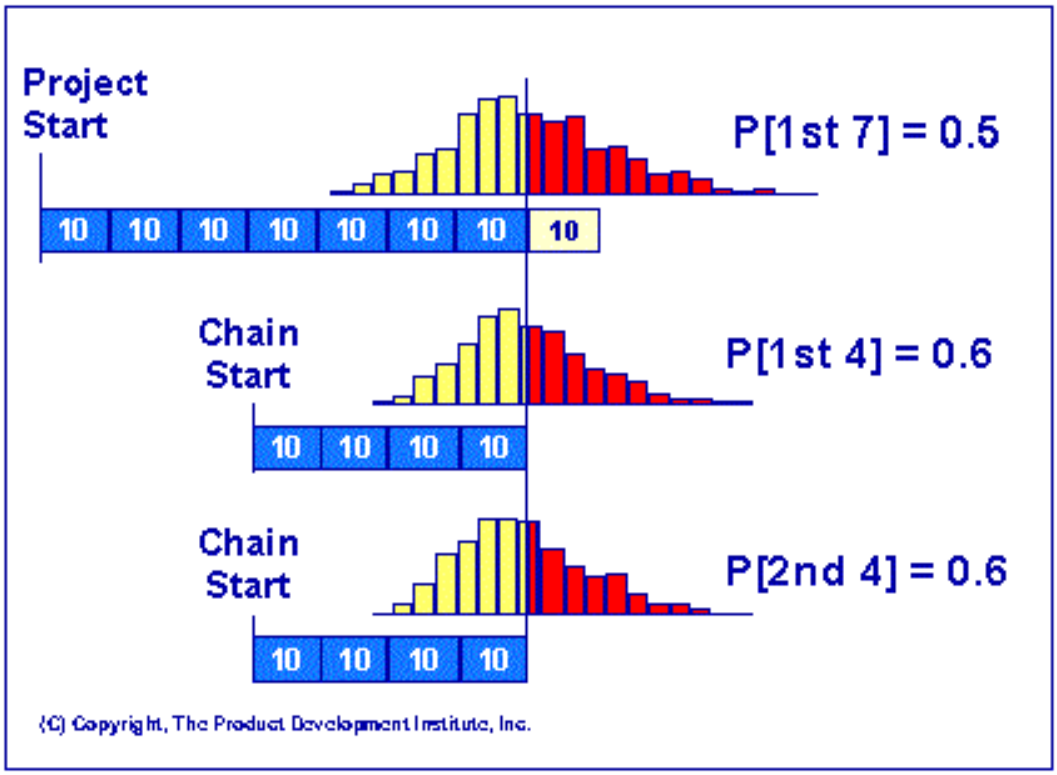


Figure 3.6: Probability that each of the three chains, individually, will be completed on or before day 70 (yellow). The probability that each chain will be completed after day 70 is shown in red.

Figure 3.6 shows the probability of successful completion for each of the three chains of tasks that provide inputs to the assembly task. For the first chain, consisting of seven tasks, the probability of successful completion by day 70 is approximately 0.5, or 50%. The probability of successful completion for each of the two shorter feeding chains is approximately 0.6, or 60%. These estimates are obtained by observation, from the accompanying histograms. However, the probability that all three chains of tasks will be completed on or before day 70 is given by the product of all three, i.e., the probability that the assembly task can begin on or before day 70 is given by the product of (0.5 * 0.6 * 0.6), which equals 0.18, or only 18%.

So, is Concurrent Engineering a bad thing? No! It's not a bad thing. Concurrent Engineering is a good thing. But, by itself it is not sufficient. We have to do much more than create short project plans with Concurrent Engineering practices. We also have to take vital steps to manage the variability in the duration of those projects. The Theory of Constraints and Goldratt's Critical Chain Method give us the means with which to do precisely this.

An effective solution to this probabilistic dilemma is to pull early the two shorter feeding chains. This, in fact, is what common sense would have most experienced project managers do. Unfortunately, experience gives no appropriate guidelines regarding how much earlier the two chains should be scheduled. TOC does.

Intuition alone would have us begin the two shorter feeding chains as early as possible. In many instances, this would not cause difficulties, other than perhaps increasing opportunities for rework. However, in a multi-project organization, the policy of beginning as early as possible all tasks that have no predecessors, for all scheduled projects, would immediately choke the organization with a mountain of work in process. This policy would cause far greater problems than it would solve.

The TOC Approach

The TOC approach, instead, is to pull the two feeding chains early as much as necessary, and no earlier. Thus, the longer chain of the project becomes rather insulated from variability in

each of the two feeding chains. The interval by which each chain is pulled early is called a Feeding Buffer. This is illustrated in Figure 3.7, below.

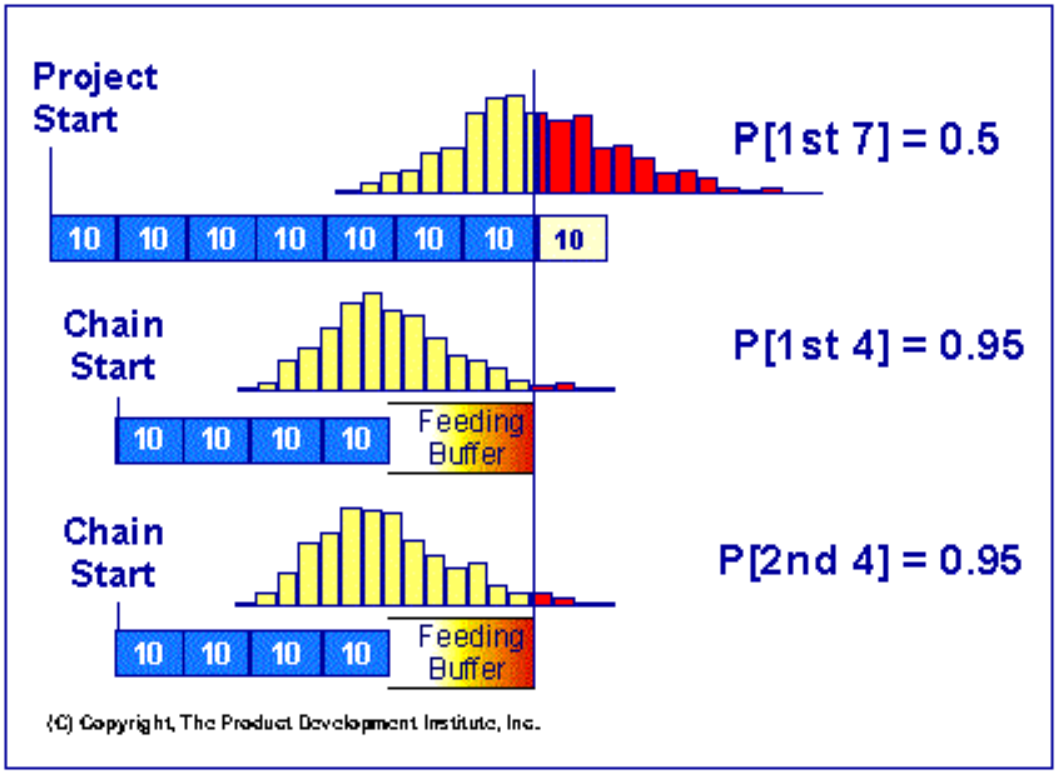


Figure 3.7: The impact of feeding buffers, on the probability that the output of each feeding chain will be available on or before the start of the assembly task. The probability of early completion is shown in yellow. The red section represents the probability of late completion.

From a probabilistic perspective, a feeding buffer increases significantly the probability that the output of the corresponding feeding chain is made available in a timely manner, for the successor task that requires it. Thus, for our little project plan, the probability that the assembly task can begin on or before day 70 is (0.5 * 0.95 * 0.95), which equals 0.45, or 45%. This is significantly greater than the 18% that the absence of feeding buffers yielded. However, is this really any different from the attempts of any experienced project manager to manage the slack in a project plan?

Indeed, the use of feeding buffers is decidedly different from attempts to manage slack. Slack occurs randomly within project plans. Feeding buffers (and project buffers) are calculated [the details of buffer sizing are discussed in a later chapter]. The size

of a buffer is a function of the number of tasks in the feeding chain and of the degree of uncertainty in the estimates of duration of those tasks. This is a drastic departure from simply managing the randomly occurring slack. The use of feeding buffers and project buffers is also far more effective, as Figure 3.8 illustrates.

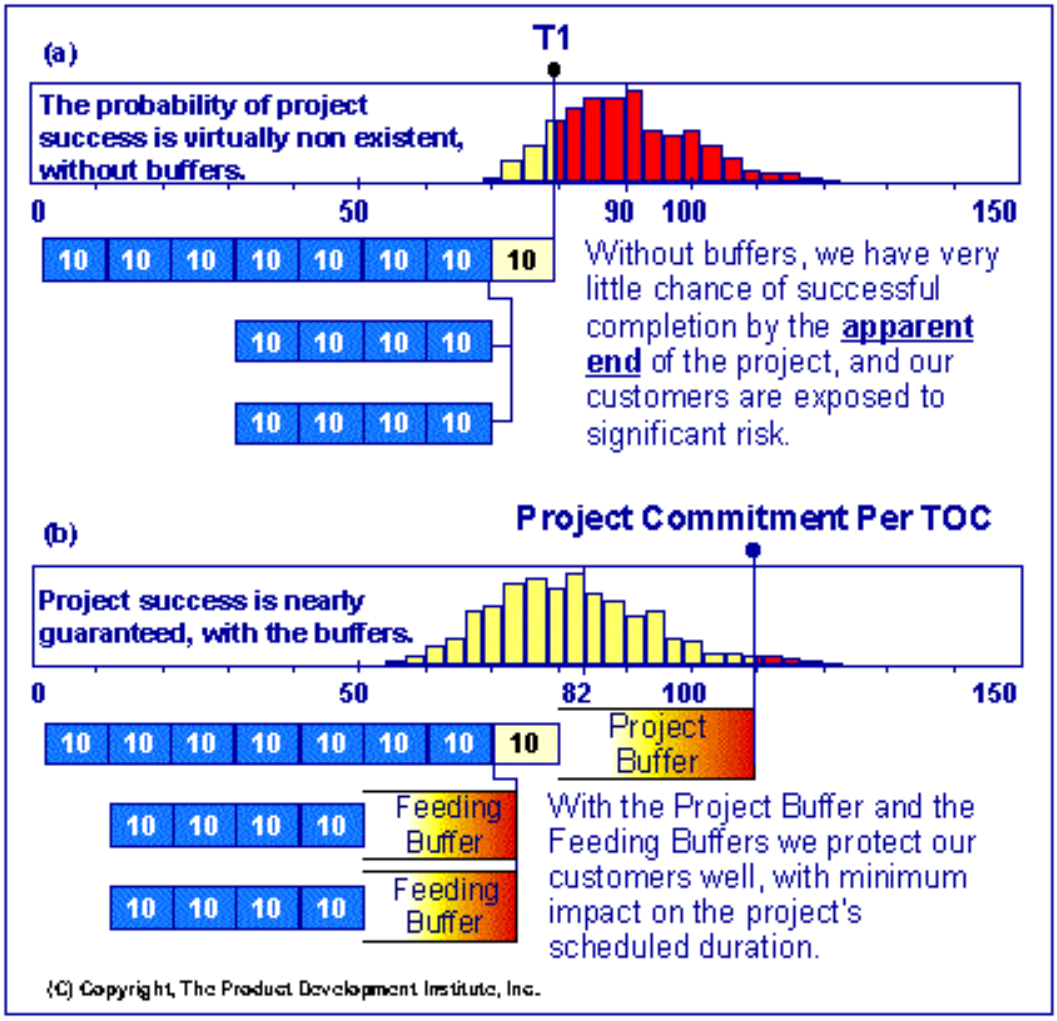


Figure 3.8: Comparison of results for (a) the example project plan with no buffers and (b) the same project plan properly buffered. The yellow area of each histogram indicates the probability of early completion. The red area of each histogram represents the probability of late completion. It is considered good TOC practice to propose commitment dates that correspond to the end of the project buffer.

Figure 3.8-a shows the probability of completing our little project on or before the apparent end of the project (T1), in the absence of an effective strategy for managing variability. The histogram shown in Figure 3.8-a represents the best that an organization could hope to achieve, were the projects of the organization every bit as simplistic as our little test project. Of course, real projects wouldn't fare as well, due to the larger number of parallel chains.

Figure 3.8-b shows how much better the on-time performance of an organization could be, were the organization's projects planned and managed per the Critical Chain Method. Of course, the commitments that the organization would make to its customers would be slightly later than the apparent end of each project. However, these later commitments would be rather credible and feasible. Would this be preferred by customers? Have your customers decide. Ask them if they'd prefer to have you lie to them or give them credible, feasible due-dates when making project delivery commitments to them.

Vulnerable Buffers

OK! I can just about see you squirming in your seat right now. I must be out of my mind, if I really think that you'll be able to put a project buffer in your next project plan, right? Without a doubt, you're living the nightmare in your mind's eye right now. You put the project buffer in your project plan, and your direct manager rips it out of your plan by the roots.

Well, this can happen. Any manager who won't tolerate project buffers in the organization's project plans is simply demonstrating complete ignorance of variability and of the methods with which to manage it. Such managers exist because many college curricula, technical or otherwise, fail to address probability and statistics. If the manager to whom you report is among the set of improperly educated managers, then I would advise you to either take appropriate steps to educate that individual or yield to his/her greater authority, while you look for a job in a more enlightened organization. After all, there's no point in becoming a martyr, is there?

In this chapter we've learned how to design a project plan in such a way as to limit the effects of the many causes of variability in

project duration. Variability simply exists. We can choose to manage it, or we can choose to ignore it. Ignoring variability won' t make it go away, and will leave us and our customers exposed to its full impact.

In the next chapter we will discuss buffer sizing. Identifying the Critical Chain of a project will be addressed in a subsequent chapter.

| contents | previous | next |

Ch. 4 - Sizing Buffers

Send this to a friend. Click here.

| contents | previous | next |

About This Chapter

In the first three chapters we discussed the need to manage variability; we saw how variability and chains of tasks interacted; and we concluded that, to achieve project plans that better represented reality and still let us protect our customers from the many inevitable sources of variability, we needed to include feeding buffers and project buffers in our project plans and schedules.

In this chapter we discuss two ways in which to determine buffer sizes. The method that you choose depends more on your environment than on anything else. Each of the two methods has advantages in some environments and disadvantages in others. Therefore, it's best that you understand both methods well, so that you can choose the one that is best suited for you and for your environment.

The Cut And Paste Method

The first method is described best as the cut and paste method. It's name hints at how it works. This method is illustrated in Figure 4.1, below.

The first step of the cut and paste method is to collect the usual, safe estimates of task duration from the individual contributors of a project team. This is illustrated in Figure 4.1-a. The second step, as Figure 4.1-b suggests, is to cut the safety from each estimate of task duration. The remaining

estimates of task duration are then used to schedule the work of the project. This is illustrated by Figure 4.3-c. Finally, half of all the removed safety is used as the project buffer. Feeding buffers, of course, get half of all the safety removed from their corresponding feeding chains.

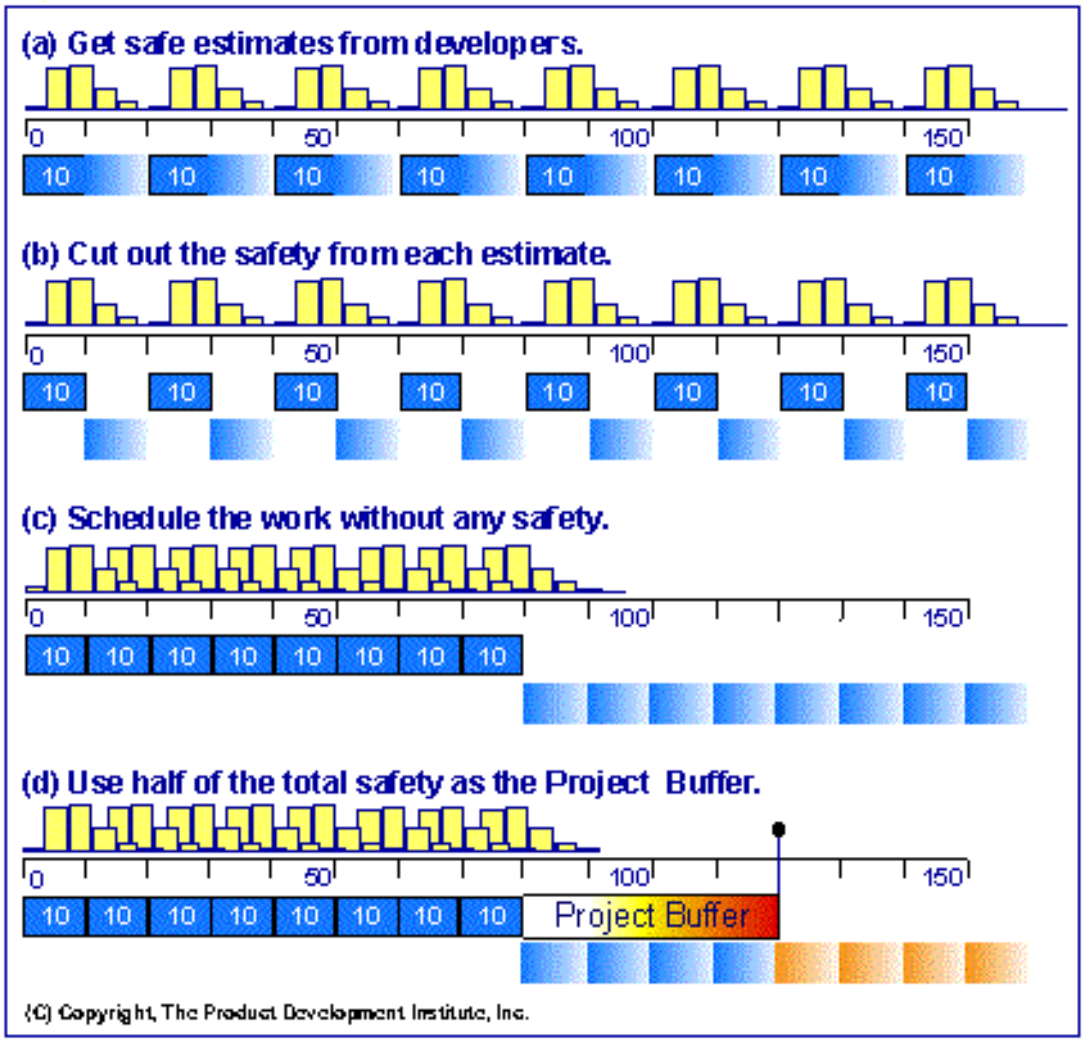


Figure 4.1: The Cut-And-Paste method for calculating project buffer sizes and feeding buffer sizes is illustrated in (a) through (d). The obvious advantage of this method is its simplicity.

The obvious advantage of the cut and paste method is its simplicity. If you work in an industry where high school graduates are a rare find, and if such rare finds are the individuals who will be calculating your projects' buffer sizes, then by all means use the cut and paste method. It's simple enough, and much of the time it's also good enough. The use of anything more sophisticated very probably will only frighten your people anyway, blocking them from buying into your proposed change process.

However, if you work in a product development environment, then the cut and paste method comes with a few problems. Consider the tremendous pressure that managers in product development organizations feel, from competitors and from customers, to create the shortest possible project plans and very aggressive schedules to accompany those plans. Consider the ongoing conflict that this pressure generates for the managers, between giving their developers sufficient time to do the work and making sure that no one includes any unnecessary protection intervals in the project plan. Consider that very often very many managers resolve this conflict in favor of having a shorter, more aggressive-looking project plan and, consequently, cut the task duration estimates of their developers. Finally, consider that nearly all experienced developers have a long history of watching their supervisors slash their estimates of task duration. Now, imagine the inevitable effects of trying to implement a management system that appears to sanction the practice of cutting task duration estimates.

What will be those effects? They will be an overwhelming degree of cynicism and a nearly complete absence of buy-in, from the very people who actually perform the work of your organization's projects. These effects are very likely to derail your efforts to implement this or any other management system.

There's another disadvantage. The cut and paste method is a linear algorithm. The size of the buffer that it calculates increases linearly with the length of the chain with which it is associated. For example, a twelve-month project could end

up with a six-month project buffer. A two-year project could end up with a year-long project buffer. In many cases, this would be an unnecessarily large amount of protection, which could lead to uncompetitive proposals and the loss of business opportunities.

In addition, due to the linear model upon which the cut and paste method is based, short chains of tasks tend to get dangerously short buffers. For example, a chain consisting of only two tasks, each with an average duration of 10 days and a safe duration of 20 days, would get a buffer of only 10 days. In the limit, a chain consisting of just one task would get a buffer equal to only half of the protection that the corresponding developer considered necessary.

In some environments, such as those where project managers with formal education are few, these undesirable effects could be considered necessary evils. However, in product development environments most managers have not only college degrees but graduate degrees in technical areas. Such highly trained individuals are not frightened by a simple equation of two. Hence, product development organizations need not tolerate the deficiencies of the cut and paste method. For these reasons, I do not recommend the cut and paste method for product development organizations. An alternate method is required.

The Root-Square-Error Method

The alternate buffer sizing method proposed here is known as the Root Square Error (RSE) method. The RSE method is actually quite common among mechanical engineers, who use it to estimate the clearances required to accommodate tolerance stack-ups. It is also used by electrical engineers, to estimate the variability in circuit performance, which is caused by variability in the components of the circuits. It is, in other words, a method that most developers and managers will recognize readily. They will embrace it just as readily, because it will make sense to them.

As illustrated in Figure 4.2, below, the RSE method requires that we obtain two estimates of task duration from developers.

The first estimate should be a safe estimate, i.e., an estimate with which the developer feels comfortable making a commitment. This should include enough safety to protect [the developer] against all the likely sources of delays. The second estimate should be one that includes no such protection. In addition, the developer should assume that the task will be worked at a full level of effort, with no interruptions imposed by external factors. This latter estimate will serve as the expected value discussed in Chapter 1.

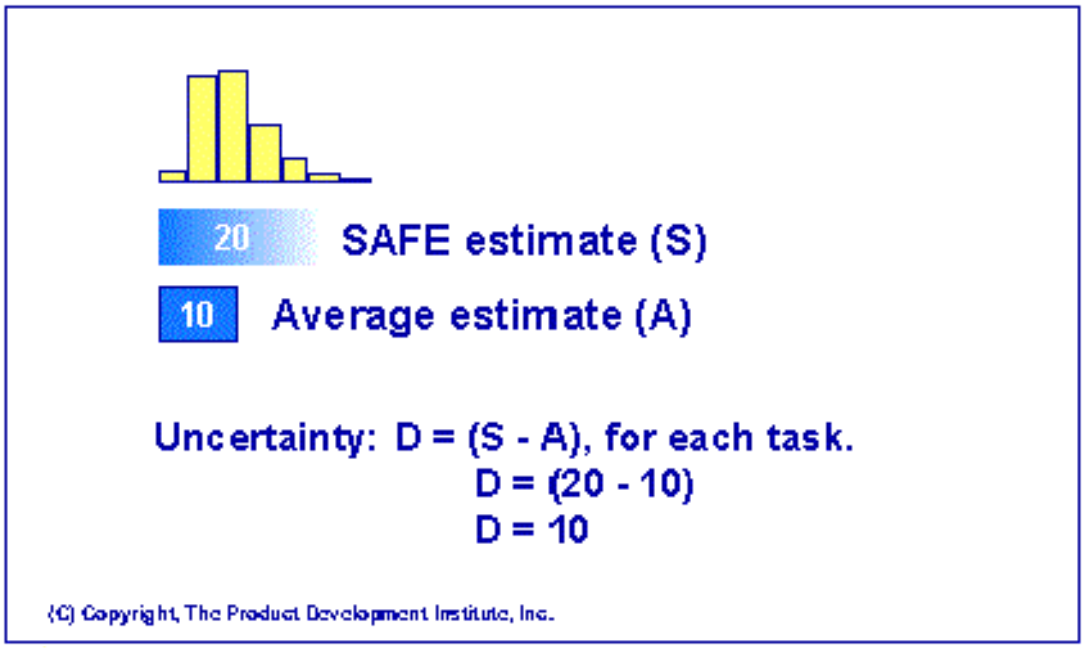


Figure 4.2: The Root Square Error (RSE) method for calculating buffer size takes its inputs directly from the developers who are expected to perform the corresponding tasks.

Once the two estimates are at hand, for each task in a chain, the difference between the two estimates is treated as the magnitude of the uncertainty in the shorter estimate. The difference (denoted as **D** in Figures 4.2 and 4.3) is, in reality, the corresponding developer's subjective measure of his/her own discomfort with the shorter estimate. For each chain of tasks, the **D** values are used to calculate the most likely error in the duration of the entire chain. The calculation is illustrated in Figure 4.3, below.

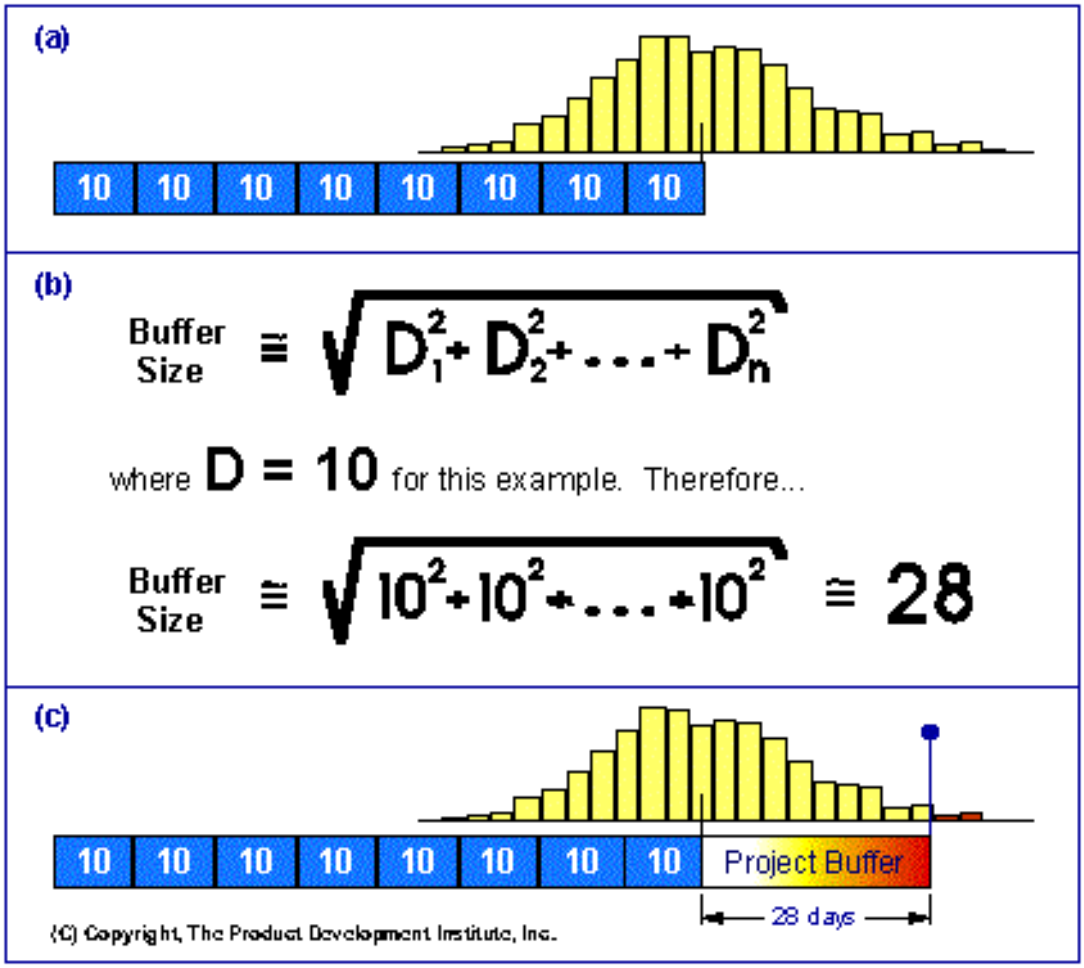


Figure 4.3: The buffer size required to protect against a chain of tasks is estimated as the square root of the sum of the squares of the uncertainty (D) in the individual tasks.

Is this a perfect method? No! It is not perfect, as those readers who are well versed in probability and statistics know. The method, in fact, makes the assumption of independence between tasks. This assumption, at times, is a stretch. Still, I'm not aware of a better method that shares this one's simplicity at this time. We could use a Monte Carlo simulation. But that's hardly a simple approach.

However, for all its imperfections, the RSE method does have some distinct advantages, particularly for product development organizations. First among these advantages is that the method does not require managers to arbitrarily cut the task duration estimates of their developers. Thus, the method preserves the vital buy-in of the developers. Second, the RSE method accepts, no, requires inputs from developers, thus

their degree of buy-in for this management method is enhanced in addition to being preserved. Third, as Figure 4.3 illustrates, the RSE method provides the requisite protection; it does so without causing us to use unnecessarily large buffers or dangerously small buffers.

In this chapter we've discussed two methods for calculating buffer sizes. They are the cut and paste method and the Root Square Error (RSE) method. In my opinion, the RSE method is the method of choice for product development organization. However, there are many instances where the cut and paste method is, to put it simply, necessary. Rather than following my suggestion or anybody's suggestion blindly, you would be well advised to think about your organization and its needs, when deciding which method to adopt.

In the next chapter we discuss a simple way in which to manually identify the Critical Chain of a project. We do this, so that we can understand where to position the feeding buffers.

| contents | previous | next |

PDI

Ch. 5 - Identifying The Critical Chain

Send this to a friend. Click here.

| contents | previous | next |

About This Chapter

In the previous chapter we discussed two methods for calculating buffer sizes. Each method is equally applicable to feeding buffers and to project buffers. But, a question remains unanswered. Where do the buffers belong? This is the subject of this chapter. We also discuss how to identify a project's Critical Chain and how the Critical Chain differs from the more conventional Critical Path.

Where Do The Buffers Go?

The answer to this question has two parts. One part addresses project buffers, and the other part addresses feeding buffers. First, let's discuss the project buffers.

As we said in an earlier chapter, the purpose of a project buffer is to protect the customer from the countless sources of variability in project duration, which the management team cannot eliminate. Specifically, every external customer who anticipates a deliverable from a project team must be protected with a project buffer. Usually, a project team has only one deliverable to a single external customer. When this is the case, the project plan requires only one properly sized project buffer. However, at times a project team is tasked with providing not one but several deliverables to external customers (or to the same external customer). Each such deliverable, then, must be accompanied by a properly sized project buffer, because, to omit a project buffer is to expose the

corresponding customer to a damagingly late delivery.

Now, let's talk about the feeding buffers. Recall. The strategy of the Critical Chain Method is to identify the longest chain of events within a project and to prevent that chain from becoming longer unnecessarily. The feeding buffers play a pivotal protection role for the Critical Chain, by ensuring that the outputs of feeding chains are ready and waiting for the Critical Chain tasks that require them. Therefore, before we can determine where to insert feeding buffers, we must first identify (or select) the project's Critical Chain.

In virtually all cases, you will not identify the Critical Chain of a project manually. You will simply provide the proper inputs to a software package, such as **ProChain**, and the software will identify the Critical Chain for you. In fact, the software will also determine where feeding buffers and project buffers are required; it will size those automatically; it will insert them into the project plan; and it will update them throughout the course of the project. However, while automation is useful and good, the use of automation without knowledge of what is really automated can be devastating. It is necessary to understand that which the software is doing for us, if we are to use such software tools effectively and without undue risk.

Identifying The Critical Chain

To illustrate at least one process by which the Critical Chain of a project can be identified, consider the simple project network shown in Figure 5.1, below. Each of the boxes in that network represents a single task. The colors represent unique skills. For the sake of discussion, let's assume that our project team includes only one individual with each skill. Therefore, we have on our project team only one developer with the Green skill, only one with the Blue skill, etc, and these individuals are not interchangeable.

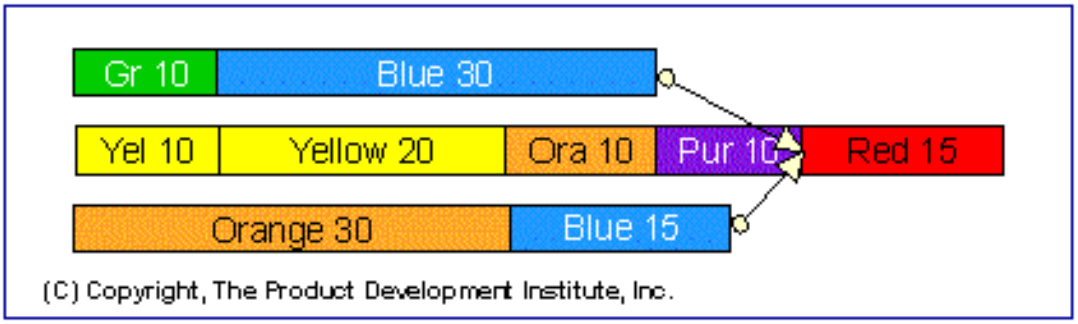


Figure 5.1: The process of finding the Critical Chain of a project begins with a conventional project network that has had all unnecessary protection intervals removed from the estimates of task duration.

Further, the numbers shown in the boxes represent the estimates of expected duration. Therefore, the Green-10 box indicates that the developer with the Green skill provided an estimate of average duration of 10 days, for his/her task. Similarly, the Blue-30 box indicates that the developer with the Blue skill provided an estimate of average duration of 30 days, for the corresponding task. In other words, the estimates of duration denoted by the numbers do not include any undue protection intervals.

How does this network differ from a conventional, Critical Path network? So far, it differs only in that the estimates of task duration include no protection intervals. In all other respects, the network shown in Figure 5.1 is identical to the Critical Path network for the project. But, this is only the first step. We're not done yet.

The second step, shown in Figure 5.2, below, requires that we push all tasks as late as possible, short of violating precedence dependencies. Why do we do this? We do so for three reasons. The important reason is that by pushing tasks late we minimize work in process throughout the organization. A second reason is that with this step we minimize the opportunities for rework. Instead of beginning work as soon as possible, we begin it when developers are more likely to have better information about their work. The third reason is a somewhat pragmatic one. Pushing the tasks as late as possible makes it easier to identify the Critical Chain. At least,

it does for me.

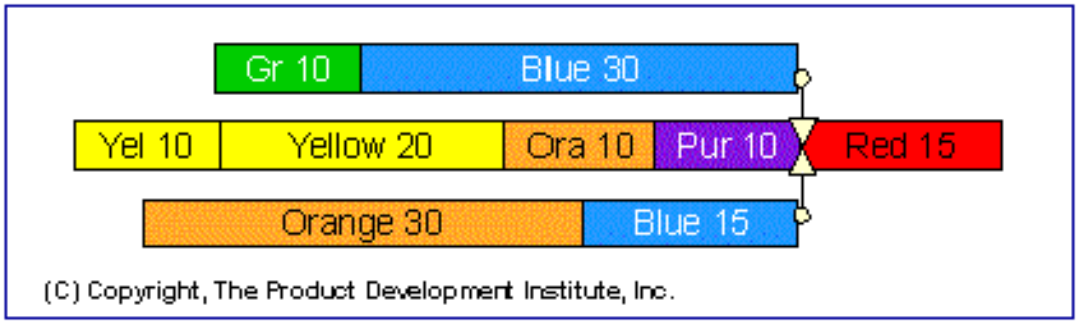


Figure 5.2: The second step in finding the Critical Chain of a project is to push all the tasks as late as possible, short of violating precedence dependencies. This step decreases work in process and opportunities for rework.

Already I can see you squirming in your seat. What? We're going to delay tasks that could be started sooner? Relax! We're not done yet. Soon we'll insert the feeding buffers, and these will let us determine the proper time to begin each chain of tasks in the project. The practice of beginning work as soon as possible is killing product development organizations, by creating countless opportunities for multitasking. The TOC method is not to begin work as soon as possible. Nor is it to begin work as late as possible. The TOC method is to begin work when it is necessary to begin it, and the buffers tells us when that is. Think of it as Synchronous Product Development. We strive to synchronize tasks, so that the required inputs are made available to the tasks that need them almost on a just in time basis. So, don't despair. Good sense rules, in a TOC organization.

The third step toward identifying the Critical Chain is to eliminate the instances of resource contention. No matter how good the project plan looks, if it calls for two or more copies of the same person at any time, the plan is not feasible. It is a pipe dream, which violates the most basic purpose of a project plan. Recall. The purpose of a project plan is to act as a predictive tool. So, we must eliminate the many instances of resource contention. This is illustrated in Figure 5.3, below.

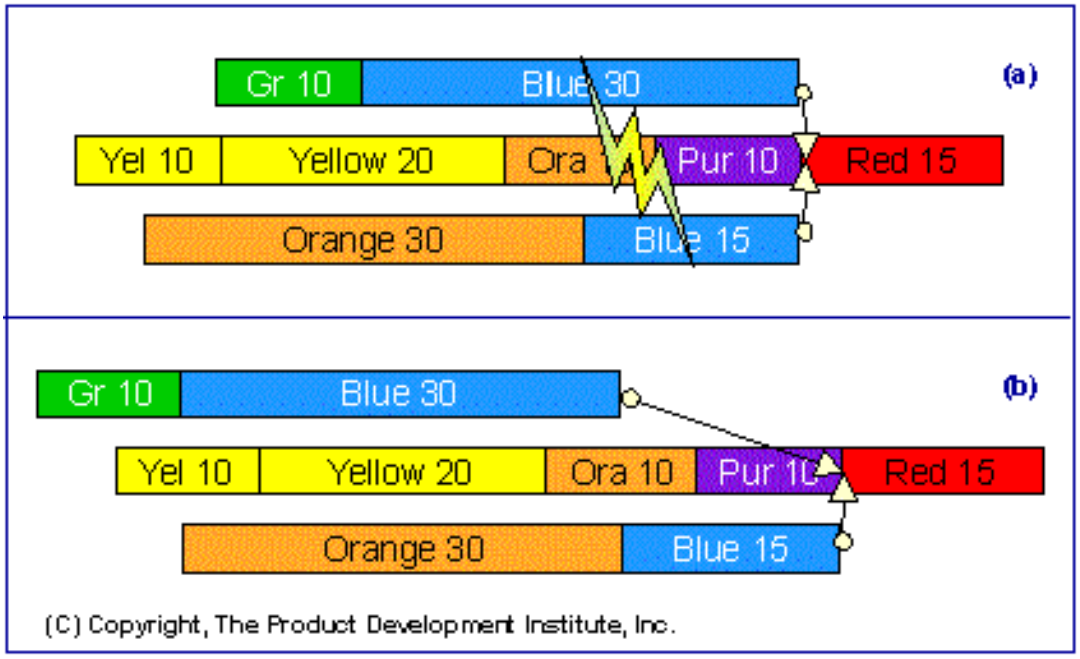


Figure 5.3: Resolving instances of resource contention, i.e., eliminating the overbooking of resources, is a crucial step toward creating project plans that represent reality. Virtually all project management software packages include canned routines for this planning step.

One practical approach to eliminating resource contention manually is to begin at the very end of the project network and to scan the network in reverse, looking for overbooked resources as we do so. This process highlights the fact that the Blue resource is overbooked in our little network (Figure 5.3-a). We now need to make a decision. We can eliminate the overbooking by pulling early the lower leg of the network (Orange-30 and Blue-15), or we can pull early the top leg (Green-10 and Blue-30). By inspection we can tell that the latter choice causes a lesser increase in the planned duration of the network. That's probably the better choice, the outcome of which is shown in Figure 5.3-b.

Is this the best process for a real project network? No, it isn't, and the outcome of using this simple process with a real network may not be the optimum. But, the reality is that I (and probably you) don't know and can't know the best possible outcome. We can know this. For all the effort required to identify the absolutely best network, we would get an outcome that 9 times out of 10 would be statistically indistinguishable

from a network that we might achieve manually. So, why bother?

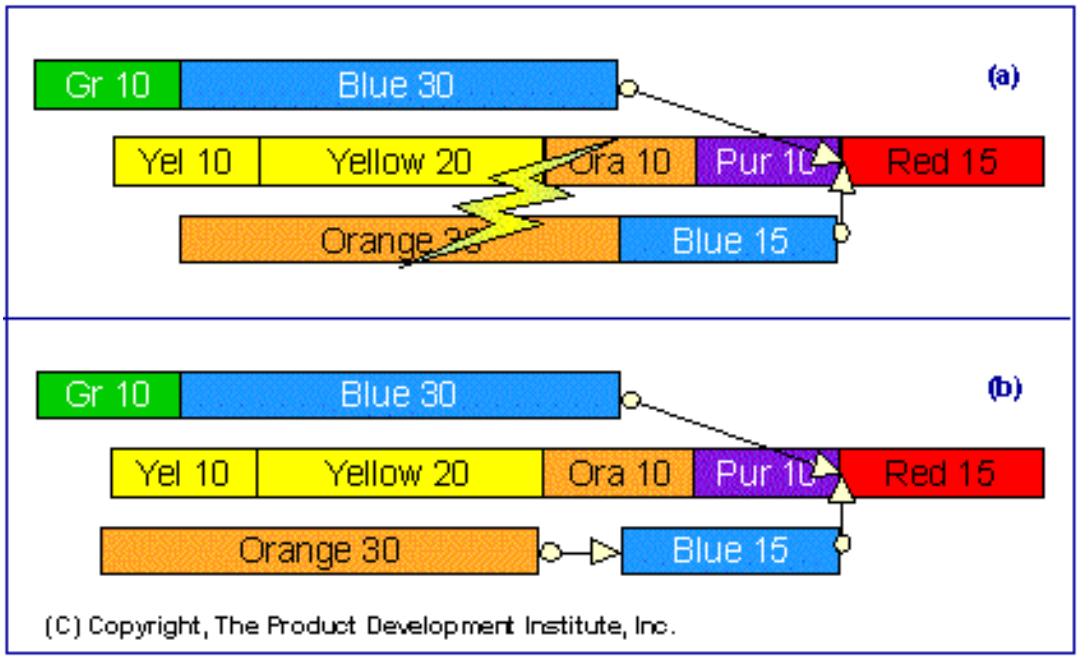


Figure 5.4: For this simple example, the overbooking of the Orange resource is eliminated by pulling the Orange-30 task sufficiently early.

Are we done? Not yet! The Orange resource is also overbooked. If we apply the same decision-making process to this second instance of resource contention, we get the outcome shown in Figure 5.4-b, above. Now we've eliminated the obvious instances of resource contention in our little project network, until the project is underway.

Why do I say, "until the project is underway?" I say it because variability in task duration is certain to create additional instances of resource contention very quickly. When that happens, we can spend a tremendous amount of effort constantly re-planning and rescheduling, or we can put in place an effective prioritization scheme, which lets us overcome the effects of statistical noise in task duration. For now, let's just agree that constant rescheduling is not a desirable pastime for project managers.

Now that we've eliminated at least the initial instances of resource contention from our project plan, we are ready to identify the Critical Chain of the project. Let's begin by defining

the Critical Chain. The Critical Chain of a project is that sequence of dependent events that prevents the project from being planned with a shorter estimate of overall duration. Keep in mind that this definition takes into account resource dependencies as well as precedence dependencies.

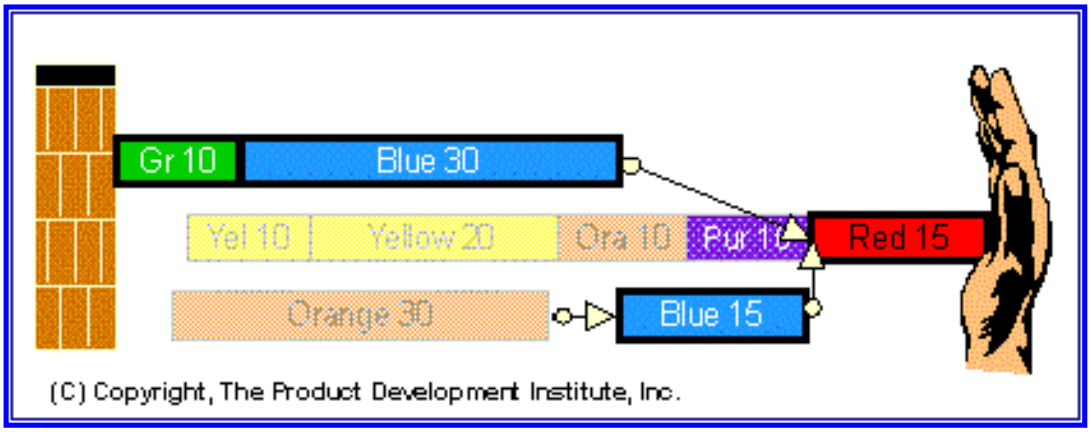


Figure 5.5: The Critical Chain of a project is that sequence of dependent events that prevents the project from being planned with a shorter estimate of overall duration.

Click here to see Figure 5.5 animated.

To identify the Critical Chain of our little project, imagine that your hand is pushing the network into an impenetrable wall, as illustrated in Figure 5.5, above. The Critical Chain is the sequence of tasks that prevents your hand from moving closer to that wall. Clearly, the Orange-30 task is not a Critical Chain task. It has slack on either side of it. The chain of tasks consisting of Yellow-10, Yellow-20, Orange-10 and Purple-10 also has slack before it. It is not preventing the network from being compressed by your hand either. The Critical Chain, therefore, consists of the tasks denoted by Green-10, Blue-30, Blue-15, and Red-15. Note, also, that the Blue-30/Blue-15 sequence is caused by a resource dependency. Given the resources that we have available for this project, those four tasks make up the longest chain in the project. That is the Critical Chain.

How does the Critical Chain differ from the conventional Critical Path? Take a look at Figure 5.6, below. Figure 5.6-a shows the Critical Chain tasks highlighted with thick borders. Note

that the Critical Chain jumps from path to path, due to the resource dependencies. Figure 5.6-b shows the Critical Path tasks highlighted with thick borders.

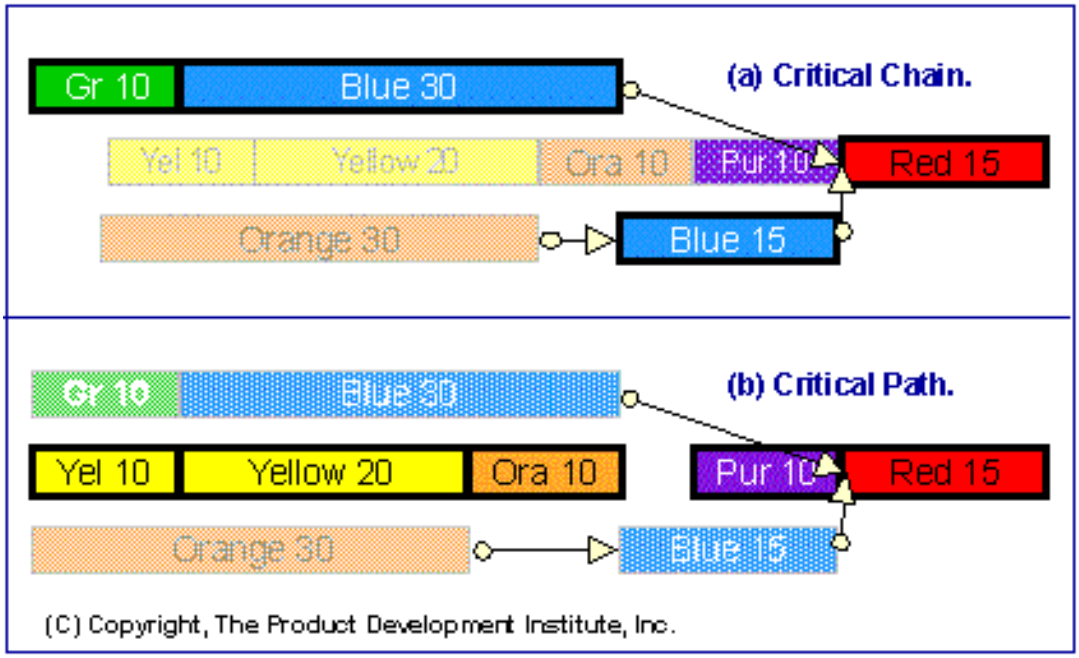


Figure 5.6: The Critical Chain of our little project, (a), takes into account resource dependencies as well as precedence dependencies. The traditional Critical Path, (b), ignores resource dependencies.

On the surface, this distinction between Critical Chain and Critical Path seems minor. In many respects, it is minor. Still, the Critical Chain offers a few interesting advantages.

One advantage is that the Critical Chain highlights where additional resources can cause the project to be completed in a shorter interval. Given the goal of completing the project in the shortest possible interval, the Critical Chain is the constraint that prevents the one-project system from making greater progress toward that goal. In other words, it is the limiting factor. If we make improvements to that limiting factor, we cause an immediate improvement to the system that is the project team and the project. The Critical Path does not make this readily apparent.

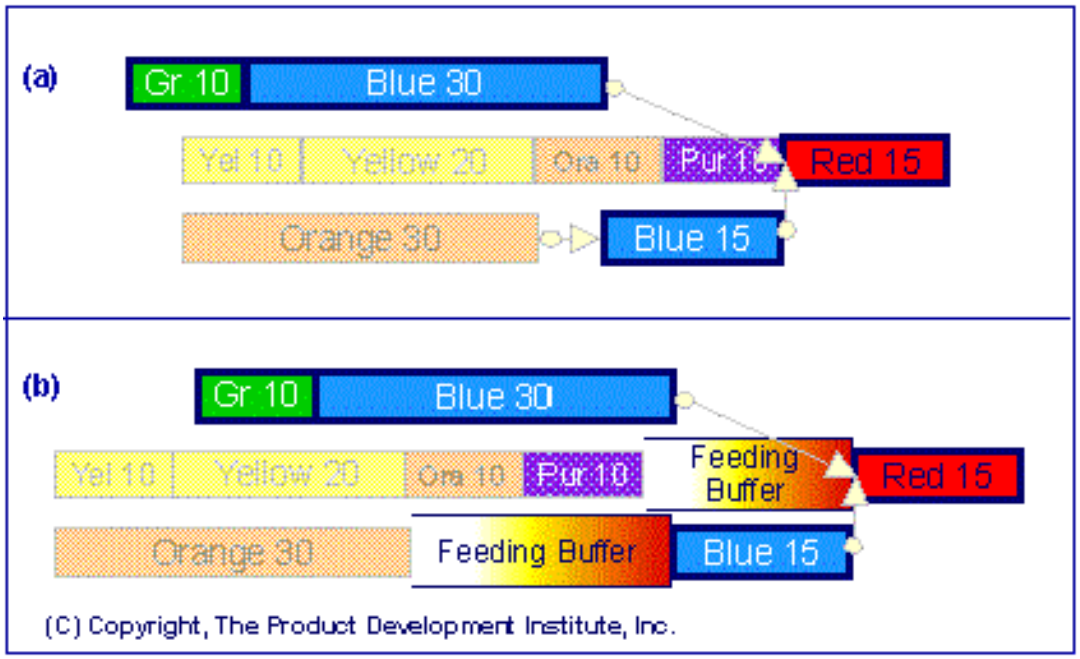


Figure 5.7: Feeding buffers protect Critical Chain tasks from variability in the duration of the chains of tasks that provide required inputs. We insert a feeding buffer wherever a non critical task feeds a Critical Chain task.

A second advantage is that the Critical Chain shows us the strategically effective points for inserting protection, in the form of the feeding buffers. Once we've identified the project network's longest chain, protecting it with feeding buffers becomes easier to do. Feeding buffers are inserted wherever a non Critical Chain task feeds a Critical Chain task. This is illustrated in Figure 5.7-a, above.

One advantage of the feeding buffers, indeed, of the Critical Chain Method, is that the need to re-plan project is greatly reduced. My own experience indicates that perhaps one project in 20 (or at worst one project in 10) experiences sufficient variability to warrant a new plan. This is a tremendous change from the monthly or even weekly re-planning sessions that project managers in many organizations undertake.

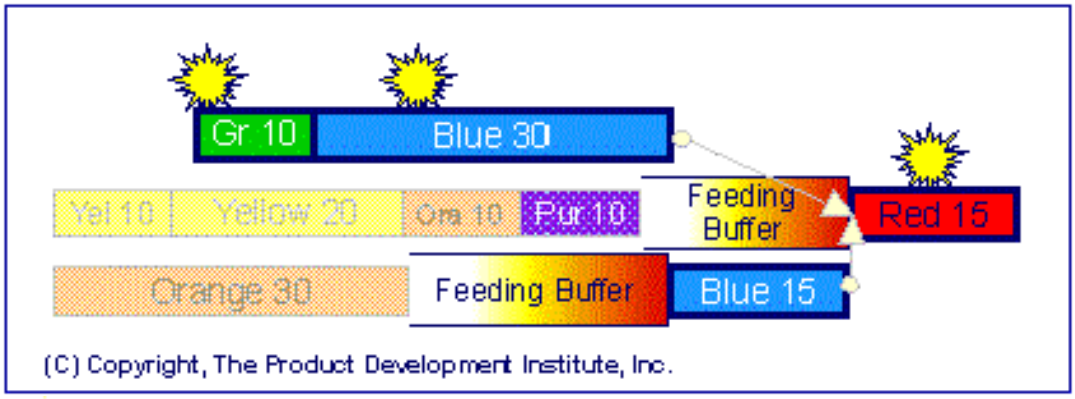


Figure 5.8: The advance booking of key resource (starbursts) on the Critical Chain of a project also prevents the project's Critical Chain from becoming longer unnecessarily.

Protecting the project from the untimely availability of key resources also becomes much easier. Once the Critical Chain of the project is underway, we can ensure that key resource on the Critical Chain are reserved sufficiently early. This not only prevents the Critical Chain and the project from becoming longer unnecessarily but also lets us take advantage of the early finishes of some Critical Chain tasks. The starbursts in Figure 5.8, above, show which Critical Chain resources need to be reserved in advance. The useful guideline is that every new instance of a resource on the Critical Chain requires an advance booking or, at least, a wake-up call. However, this guideline applies to one-project systems only. A less stringent measure is sufficient for multi-project systems.

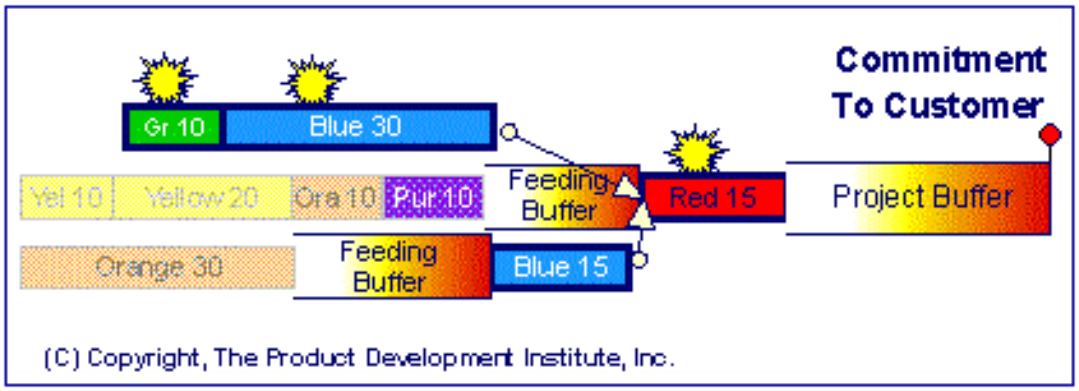


Figure 5.9: The project buffer, which protects the customer, is the final touch.

Finally, Figure 5.9 shows the Critical Chain plan for our little project. The plan exhibits the following useful features:

1. It is feasible with existing resources.
2. It is our most accurate representation of the ability of our one-project system to deliver the project.
3. It protects the customer from the many sources of variability, which the management team cannot eliminate.
4. It includes no undue protection.
5. Its feeding buffers protect the Critical Chain, thus minimizing the impact of variability in task duration and the accompanying need to reschedule.
6. It identifies those Critical Chain resources whose untimely availability might damage the project.
7. It enhances the team's ability to deliver the project early, instead of delivering it on or after the due-date.
8. It greatly enhances the project manager's ability to track and manage the project.

But Wait A Minute!

OK! Have you noticed it yet? Figure 5.9 shows both feeding chains starting sooner than the project's Critical Chain. Doesn't this mean that now the feeding chains are critical and the Critical Chain is no longer critical?

Well, no, it doesn't mean that. The Critical Chain is simply the chain that is longest, before we insert all the appropriate buffers. Consequently, it is the chain of tasks upon which we need to focus the project team, i.e., it is the chain of tasks to which the members of the project team should give priority, whenever they have to choose between two or more tasks.

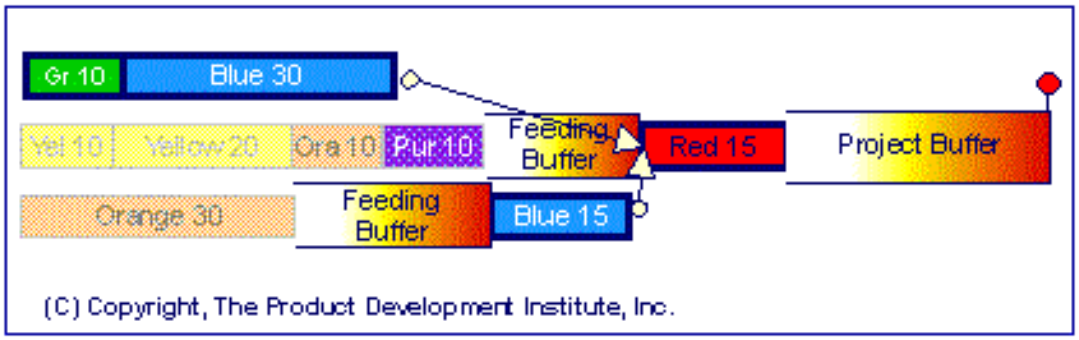


Figure 5.10: At times, a feeding buffer requires that a feeding chain be started earlier than the Critical Chain. Pulling the first Critical Chain task early, so that it starts at the same time as the earliest feeding chain, usually causes no difficulty.

Can we begin the Critical Chain at the same time as the earliest feeding chain? Should we? Sure! So long as by doing so we don't create opportunities for rework and wasted effort, there's no real reason to delay the start of the Critical Chain, when we have a dedicated project team available. Remember, this entire tutorial focuses on the one-project system, which presumes a dedicated project team. This is why Figure 5.10, above, shows the Critical Chain of our little network beginning at the same time as the two feeding chains. If your organization performs multiple projects with the same resources, then the one-project Critical Chain model doesn't apply. If that's the case, then you need to consider Synchronous Product Development, also known as the TOC Multi-Project Management Method. Readers interested in the Multi-Project Management Method should contact PDI by calling (973) 581-1885.

In this chapter we've learned how to identify the Critical Chain of a project. We've also discussed how the Critical Chain differs from the Critical Path and how to determine where to insert feeding buffers. Finally, we've discussed some of the significant benefits afforded by the Critical Chain Method.

In the next chapter we discuss an even more important use for the project buffer and the feeding buffers: buffer management.

| contents | previous | next |

Ch. 6 - Buffer Management

Send this to a friend. Click here.

| contents | previous | next |

About This Chapter

In the previous chapter we learned how to identify the Critical Chain and where to insert properly sized buffers. In this chapter we address the minor and major purposes of the buffers.

The minor purpose is the obvious one. The buffers provide aggregated protection against statistical variation in task duration. For example, the project buffer provides all the protection for the Critical Chain tasks. Thus, if any Critical chain task should run appreciably longer than planned, the increase in duration would be absorbed by the project buffer. Conversely, if a Critical Chain task should run shorter than planned initially, then the extra time would be added to the project buffer automatically. This automatic accumulation of unneeded protection time in the project buffer is one of the benefits of aggregating protection. But, as we said in the beginning of this paragraph, this is the minor purpose of the buffers.

The major purpose is less obvious but significantly more important. In addition to providing protection against statistical variation, the buffers act as transducers that provide vital operational measurements. This brings us to the definition of an operational measurement and the difference between an operational measurement and a performance measurement. Let's use an analogy with a measurement that is already familiar to you.

Imagine that you're driving on a country road. You see that you are approaching a curve. The yellow caution sign alerts you that the suggested speed limit for the curve is 40 miles per hour. Unless you have a suicide wish, you are very likely to look at your speedometer, which displays your current speed, and on the basis of the current speedometer reading you decide what to do in the next two seconds. That speedometer reading is an operational measurement.

Now, pretend that your joy ride is over, and you are calculating the average speed for your trip. The average speed is a performance measurement. The average fuel consumption rate also is a performance measurement.

Are performance measurements useful? Sure they are! But that's not the question that we should be asking. The real question is this: are performance measurements useful while you're operating your system? Imagine again that you're trying to negotiate that rapidly approaching curve. You look at your speedometer, and instead of displaying instantaneous speed your speedometer displays your average speed for the previous ten miles. Would you find that information very useful? Very probably, you would not! To operate your system safely, you can't rely on performance measurements. You need operational instruments, not unlike your car's speedometer, tachometer, fuel gauge, oil pressure gauge, and coolant temperature gauge.

The buffers are a project manager's operational instruments. Consider that we size and insert buffers, into the project plan, for the purpose of providing protection for the project's customer and for key events throughout the project. Doesn't it make sense to update the buffers and to use them to estimate the degree of risk to which each protected event is exposed? For example, wouldn't it be useful to monitor the status of the project buffer continually? By knowing the size of the remaining project buffer and the remaining critical chain length, at any time throughout the course of the project, we can estimate the level of risk to which the customer is exposed. This is the sort of operationally useful information that the buffers provide. Click on Figure 6.1, to see an animation that

illustrates this point.

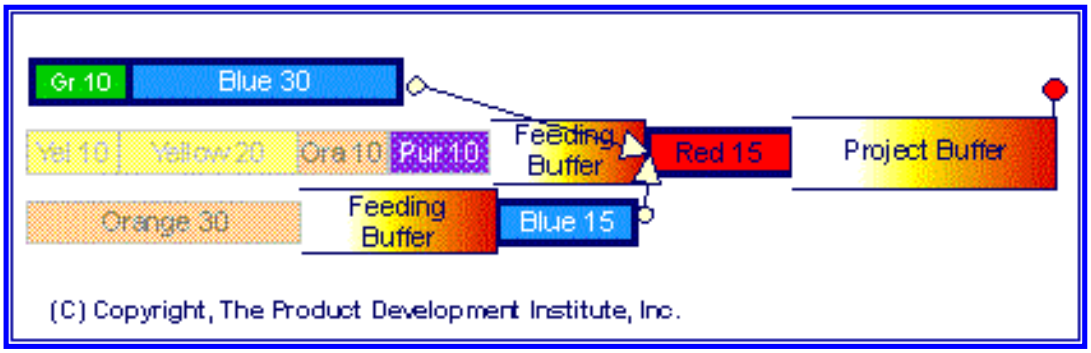


Figure 6.1: It is said that a picture is worth a thousand words. If this is true, then an animation is worth ten thousand words. Click on the figure, to see how to use the buffers.

The feeding buffers, of course, are used in the same manner. In the event that a feeding buffer appears to be in jeopardy, the project manager needs first to evaluate the situation. Most times, no action is required, because the remaining length of the corresponding feeding chain is usually quite short by the time that the buffer is consumed appreciably. This is an indication that the feeding buffer's consumption is the result of common cause variability in task duration.

Occasionally, the evaluation reveals that a special cause is at work. Such special causes of variability in task duration are not likely to go away on their own. Then, some action is required.

However, be it a feeding buffer or be it the project buffer, when a buffer sends a signal that something unusual is happening, that signal comes significantly earlier than most project managers are accustomed. This early warning capability alone, which usually comes as a surprise to project managers and to resource managers alike, makes the Critical Chain Method extremely valuable.

Feeding buffers and project buffers are to a management team as the gauges on an aircraft's instrument panel are to a pilot. It is possible to complete a project within budget and ahead of schedule, without buffers and without the concept of a Critical

Chain, just as it is possible to fly an aircraft without proper instruments. But the flying is much safer, if the craft has working instruments.

Beyond Critical Chain

In these six chapters we've learned of Dr. E. M. Goldratt's Critical Chain Method. We've seen how the Critical Chain Method gives us the means with which to "immunize" a project (and the customer of the project) from common cause variability and special cause variability in task duration.

However, there exist situations where a slightly more detailed model can serve us more effectively. For example, many project networks exhibit diamond-shaped structures with parallel legs of nearly equal length. Such diamond-shaped networks often arise in organizations that need to compete for development projects; quoted lead time is always a key success factor in the proposals with which such organizations compete. The ability to model such networks with a bit more fidelity to reality, therefore, becomes somewhat useful.

In addition, for some organizations scaling is a huge issue. Consider an auto maker whose projects are really collections of projects that have to be integrated and carefully coordinated. Consider the development of a weapon system, which consists of numerous subsystems. Often, many of the subsystems are developed by subcontractors, and they are integrated by a prime contractor. The task of coordinating the development of a number of large projects for the success of a procurement program has not been addressed by the Theory of Constraints yet. In fact, this task is less a project management task than it is one of supply chain management in product development. These issues and appropriately useful adaptations of Goldratt's Critical Chain Method will be discussed in future tutorials.

Finally, the Critical Chain Method is a one-project management system. It makes the assumption that resources are dedicated to the project at hand. While this assumption is valid for many projects, there are many more projects for which the assumption simply does not apply. Specifically, the

assumption of dedicated resources does not apply to the projects performed by most product development organizations, which are described best as multi-project organizations. For such organizations we need not a single-project management solution but a multi-project management solution.

Now, a multi-project management solution exists. The solution is the **TOC Multi-Project Management Method**. However, like the implementation of any new management system, the change process required to implement the TOC Multi-Project Management Method is filled with pitfalls. Implementing it successfully requires not only knowledge of the new management process but considerable experience in guiding organizations through the required change. **The Product Development Institute brings exactly this experience to your business.**

| contents **|** previous **|** next **|**

PDI

back to list of articles

The TOC Multi-Project Management Method

Introduction

The TOC Multi-Project Management Method overcomes a devastating policy constraint. That constraint is the policy of letting forces external to the organizational system determine the start dates of new projects. Such forces include high-level executives (who own the system but are actually external to it), customers (who make decisions that further their own goals rather than the goals of our organizations), and marketing consultants (whose estimates of market windows often appear to be based on ... well, let's not get into that).

Think of it, we are completely responsible for a highly complex piece of expensive machinery, which we call a product development organization, and we let others tell us how to run it. Specifically, we hand over the controls to people who want to further their own goals, not the goal of the stockholders. Would the owners of an expensive wafer fab let customers run the manufacturing system? Would a hospital's administrators let patients determine the operating room schedules? So, why do we give so much control to people who know little or nothing about the operations of our complex, expensive, product development systems? It doesn't make sense. Yet, we do it, and by so doing, we create a push system. That is, we create a system that has work pushed into it by external forces.

The consequences of running our product development organizations as push-systems are best illustrated by a commonplace occurrence. Have you ever pushed a lawnmower into tall grass too fast? If you have, then you know the outcome of pushing too fast. The engine begins to slow down, and finally it stalls.

However, usually you don't let the engine stall. Instead, you slow down or even stop pushing completely. You let the engine pick up speed, and then you begin pushing again. But, you push only so fast as the lawnmower lets you push.

Your response is the result of the control subsystem that is part of the human-machine system (you and the lawnmower are the system - your brain is the control subsystem). As you push your lawnmower, you listen to the frequency of the engine noise, and you associate engine speed with the frequency of the engine noise. As the engine begins stalling, you hear a lower frequency. That's your error signal. Your brain, then, issues a control signal in response to the lower frequency. It says, "*Stop pushing, until the engine picks up speed again.*"

The whole system works beautifully. The grass is cut as fast as the system is capable of cutting it, because the system (you plus your lawnmower) pulls work into itself at an optimum rate.

Unfortunately, there is no such control system in nearly all product development organizations. No one is listening to the speed of the product development engine. It's as if management were wearing ear muffs. Consequently, no one issues the necessary control signals, saying, "*Stop pushing,*" until it's too late. Thus, the product development engine never has the opportunity to pick up speed. Instead, it lurches and sputters its way to the marketplace at a slow crawl.

The slow crawl is due to a phenomenon called unfavorable multitasking. **Unfavorable multitasking is the result of the widespread practice of assigning resources to multiple projects simultaneously, without giving those resources the means for setting day-to-day priorities appropriately.** Consider the following example.

Imagine that you are a worker in a typical product development organization. You probably have responsibility for achieving a number of intermediate milestones on several projects. Let's say, three projects, A, B, and C. Let's say, also, that you've just completed a milestone for project A, and you've done so two weeks ahead of schedule. Further, you are the resource scheduled to continue the work

for project A. Since you've just completed an important milestone two weeks early for project A, would it be fair to assume that at least for a short time the pressure from project A is somewhat diminished? Very probably, this would be a fair assumption. However, you still have upcoming milestones for projects B and C. If you don't meet those, you'll feel more than a mild increase in pressure from those projects. So, how likely are you to continue working project A? Of course, you're not likely to do that at all. If you're like most people, i.e., logical, you'll focus on one of the other projects. Consequently, project A is virtually blocked from seeing the benefit of your early finish. Why? Project A is blocked, because your day-to-day priorities are being set not on the basis of what is best for the organization but on the basis of what is required for you to survive within the organization.

We'll discuss the TOC approach to setting individual priorities later.

The TOC Multi-Project Method

Goldratt's Multi-Project Management Method consists of five steps that together overcome the policy constraint just discussed. The five steps are:

1) Prioritize the organization's projects.

2) Plan individual projects via critical chain.

3) Stagger the projects.

4) Measure and report the buffers.

5) Manage the buffers.

Let's discuss these five aspects of the Multi-Project Management Method.

1) Prioritize The Projects

Some people call this portfolio management. Whatever we choose to call it, it is a very necessary aspect of the TOC operational model. With

this step, we recognize that we cannot do everything at once. No organization has the capacity to perform simultaneously all the projects that its marketing department identifies for it. Consequently, the projects must be prioritized.

This prioritization, of course, is a leadership task. Only the leadership of an organization has the complete view of the organization and its markets. Therefore, only the leadership of the organization is able to determine the optimum (read that as most profitable) sequence in which to bring the projects into the organization. Unfortunately, this step often is left to middle managers or, worse, to individual project managers, with disastrous consequences.

What are the consequences of not prioritizing an organization's projects? First, consider that the prioritization happens always. It's not a matter of prioritizing or not prioritizing. It's only a matter of who does it and on what basis he/she does it. Ultimately, the work of every employee is prioritized, because people are single-process systems. People can do only one creative thing at a time. Sure! Some of us can walk and chew gum at the same time. But when it comes to performing the intellectually creative work required for new-product development, people can perform only one task at a time. It's a fact of life. Consequently, employees ultimately do prioritize their work, in a variety of ways.

The most common prioritization method is FIFO, i.e., First In First Out: "Whichever task shows up in my in-box first, I do first." Unfortunately, this method of prioritization is almost never the best one for the bottom line. Nor is another prevalent prioritization method, which we might call the volume control method: "I do first the task that turns down the volume of the manager who is screaming at me the loudest." This may be a truly necessary thing for an individual's survival in the organizational system. But, the resulting prioritization is almost never consistent with the goal of maximizing the bottom line.

Therefore, if the leadership of the organization does not take the trouble to prioritize the projects of the organization appropriately, and if the leadership of

the organization does not communicate that prioritization throughout the organization, then the prioritization happens most often in a way that is grossly sub-optimal for the bottom line. The message is clear. Prioritize the projects that are to be performed by your organization!

2) Plan Individual Projects Via Critical Chain

In most product development organizations, today, there exist numerous forces that drive people to protect their individual task commitments. Such forces may be the result of measurements, such as the measurement of the performance of individuals relative to a set of intermediate milestone due-dates. Whatever the cause, the effect is that individuals protect their estimates of task duration. Dare we use the "p" word? They pad their estimates of task duration, so as to avoid negative consequences for themselves.

Unfortunately, a direct outcome of embedding such protection time within individual estimates of task duration is that the estimate of the overall duration of a project grows beyond the limits acceptable to management, customers, and the bottom line. Therefore, management usually responds with what we might call back-pressure. Typically, this means that management mandates cuts in all estimates of task duration, usually in a rather arbitrary manner. The battle that results between management and staff, of course, rages on.

The TOC approach to protection, against the inevitable empirical evidence in support of Murphy's Law, is to concentrate protection only in those areas of a project's network where the protection does the most good. There are two such areas. The first, and perhaps the more important, is at the end of the longest chain of tasks (the Critical Chain). At the end of a project's Critical Chain, and before the due-date for the project, the TOC approach has us insert the Project Buffer.

The second type of protection is called the Feeding Buffer. A Feeding Buffer is placed between every Critical Chain task and any non Critical Chain task that feeds the Critical Chain task. The job of the Feeding Buffers is to protect the starts of those Critical Chain tasks that require inputs from non Critical Chain tasks. By protecting the starts of the

Critical Chain tasks from the untimely availability of the required inputs, with the Feeding Buffers, we prevent the project's longest chain of tasks from becoming longer unnecessarily.

This strategy makes great sense, particularly from a probabilistic point of view. By concentrating the protection at the end of the project's longest chain of tasks, we provide the requisite amount of overall protection with the shortest estimate of overall project interval, due to what some call the law of aggregates. We concentrate each project's protection, rather than spreading it across all tasks. The result is a system of buffers that protects the projects from what the Master W. Edwards Deming would have called the common causes of variability in task duration.

But, that's not all. In addition to providing protection against the common causes of variability, with the buffers we can also detect the existence of special causes of variability in task duration. We do this by continually calculating how much of a buffer remains (and how much is consumed), as the project progresses. The key is the degree to which buffers are consumed. Moderate consumption of a buffer is no cause for alarm. It is attributable to the common causes of variability in task duration. Severe consumption of a buffer, however, indicates that something is at work which is not likely to go away by itself.

The Critical Chain approach of concentrated protection gives us a dual benefit. First, we protect the project appropriately and with a minimum impact on the estimate of overall project duration. Second, we gain a system with which to monitor risk effectively throughout the course of the project. As we'll see later, the system of buffers becomes an important ingredient in the ongoing success of the TOC Multi-Project Management Method.

3) Stagger The Projects

This step is most critical. Staggering the organization's projects is tantamount to pulling back on the lawnmower, so as to let the engine

achieve and maintain a more effective speed. But, we face a difficult question. How can we stagger the projects of the organization enough to let the organization work faster and yet not so much as to postpone earnings unnecessarily? This is where Goldratt's exceptional systems thinking really shines.

Goldratt's solution is to stagger the projects according to the availability of one resource, called the drum resource. The idea is first to select one resource that is relatively heavily loaded and required by most of the projects. Then, the work-schedule for this drum resource is used as the mechanism with which to limit the rate at which projects are allowed to enter the system. Projects are worked by each drum resource sequentially. Therefore, the drum resource is never overloaded. Further, since the drum resource is itself heavily loaded (relative to other resources), all other resources are, themselves, also not overloaded. But this is only part of the solution.

Another part is the Capacity Buffer. This is a window of time during which no work is scheduled for the drum resource. Capacity Buffers are placed between the drum resource's work on succeeding projects. It is useful to think of Capacity Buffers as insurance. They help us to ensure that everybody, on average, has enough time to do all the work of all the projects of the organization. This makes capacity buffers invaluable in preventing the damaging, indiscriminate multitasking. In addition, capacity buffers let us set aside time on everybody's schedule, for the overhead work of the organization.

With the selection of the drum resource, the effective use of the drum resource's schedule, and the effective use of Capacity Buffers, we can stagger the projects of the organization as much as we need to, without postponing earnings unnecessarily.

Wait. Let's think about this last statement. We're saying that we are purposely staggering the projects of the organization. Yet, we claim, we are not postponing earnings, which implies that our projects aren't delayed by the capacity buffers. Does this make sense?

Indeed, it does make sense. The schedule is only a representation of reality. It is not reality. Nor does the schedule drive reality. Once a project is launched, in a TOC system, the work of the project happens as soon as the members of the project team are able to do it. Once the project is started, team members are instructed and expected to begin their work as soon as all the required inputs are available. They finish their tasks and deliver their outputs to waiting team members as soon as those outputs are good enough. By design, the whole system works like a massive relay race. Once the race begins, the baton is passed from team member to team member as soon as possible, not on scheduled dates. Therefore, the amount of stagger that we put in the schedule of the drum resource has no impact on the real performance of the organizational system, unless that stagger is so small as to permit team members to become overloaded.

4) Measure The Buffers

The three steps that we've discussed so far do not deal directly with the organization's reality. For example, the first step discusses prioritization. This impacts the overall schedule for the organization, which is a model of reality. It is not reality. The second step discusses Critical Chain scheduling. The third step talks about staggering the projects. Again, these deal not with reality but with the model of reality that we call a schedule. All three steps are necessary, to be sure. But, unless we take appropriate steps to ensure that reality matches the assumptions built into our model, the organization will see no benefit from this management method. None!

Achieving and maintaining proper focus throughout the organization is vital to making reality match our model. To do this, we can exploit the tendency of most people to optimize whatever measurement is applied to them explicitly or implicitly. If we report the size of the buffer that remains and the estimated duration of the remaining work in a chain of tasks, to the people performing those tasks, then we provide those people with a direct measurement of the performance of their part of the organizational system. Most of the time, they will respond by trying to

optimize that measurement, i.e., they will respond by trying to safeguard the buffer. Consequently, a timely, unbiased buffer report becomes an invaluable tool for maintaining focus throughout the organization.

5) Manage The Buffers

While a timely, unbiased buffer report plays an important role in maintaining proper focus throughout the organization, it plays an even more significant role in setting priorities correctly.

Recall the example, earlier, where you were given responsibility for work on three projects simultaneously? We concluded that under most circumstances your need to survive within such an organization would preclude projects from seeing the benefits of your early task completions, because your day-to-day priorities would be determined on the basis of your survival needs, not on the basis of the organization's needs.

The buffer report completely reverses this damaging situation, because it gives everybody in the organization (employees and managers, alike) exactly the information that they all need, to set day-to-day work priorities. This is the first line of buffer management. By interpreting the buffer report properly, everybody in the organization knows what is truly urgent and what is not. Recall. Once the system of buffers is in place, every task becomes a link in a chain of tasks, which feeds one of the buffers. Every task can be identified immediately as, either, a Critical Chain task or a non Critical Chain task. Why is this important? Read on.

Let's say that you're that employee again, with simultaneous responsibilities on three projects. You have before you tasks from all three projects, and you need to determine which one of the three is the most urgent. All you need do is look at the buffers associated with the three tasks. A task that is associated with a project buffer is a Critical Chain task. It always has priority over tasks that are associated with feeding buffers. If two or more tasks are all associated with similar

buffers, then the task whose buffer is in greater jeopardy is clearly the most urgent of the set. With the system of buffers and with a timely, unbiased, comprehensive buffer report, you and every other engineer and manager become able to set your day-to-day priorities consistently in a manner that safeguards all the projects of the organization.

Does this mean that no one's work is ever interrupted? Of course not! But, this system does ensure that the damaging, unfavorable, indiscriminate interruptions are eliminated. The system even **enables** interruptions, when they are clearly not damaging. In fact, this system actually relies on interruptions, when they favorable to the projects of the organization, a condition that is easily detected by a brief look at the buffers.

The system of buffers is helpful in keeping everybody working on that which is truly urgent, day-to-day. This focus, coupled with clear instructions that the outputs of tasks are to be transferred to waiting resources immediately upon their availability, is what generates most of the real performance improvement promised by the TOC Multi-Project Management Method. It is what causes reality to match the TOC model closely.

We also said earlier that the buffers provide detection of what W. Edwards Deming would have called the special causes of variability in task duration. Indeed, they do; such special causes usually don't go away by themselves. When such special causes of variability in task duration are encountered, the management team must take appropriate action. Management action under such circumstances is what makes the difference between partial buffer management and full buffer management. It is also what can enable an organization to drive due-date performance well beyond 90%.

Further, the system of buffers and the organization's global buffer report, together, give the management team unprecedented flexibility in the assignment of resources. If the buffer report indicates that one project is in serious trouble, the same buffer report also shows where

the right resources can be borrowed, without jeopardizing a customer commitment. The least expensive course of action - it costs nothing to reassign existing resources temporarily - also becomes the safest course of action for the management team. There it is again, TOC's cost control benefit!

Do you want to learn more? Click here!

IF YOU CARE, THEN SHARE.