

INTRODUCTION TO CRITICAL CHAIN PART II: MANAGING DURATION UNCERTAINTY BY MIKE MANNION AND SVEN EHRKE

FROM PROCESSES TO PROJECTS

When Dr. Eli Goldratt first described his theory of constraints in the 1985 book “The Goal” [Gol84], many people raised the question as to how the theory could be applied to project management. Goldratt responded to this question in 1997 with the book “Critical Chain” [Gol97].

As we mentioned in Part I, constraints can, in practice, take many forms. Goldratt recognised that in the case of projects, *the project constraint is the sequence of dependent tasks, which together dictate the project delivery date.* He termed this task sequence the *critical chain*.

But hold on, doesn't this sound a bit like the *critical path* from the Critical Path Method?

Before we jump to conclusions we need to recognise that fifty-year-old CPM was designed in an environment in which task durations were well known and highly stable. In part I we saw that statistical fluctuation plays a fundamental role in determining a process's throughput, so at the very least we should be wary about using CPM in projects where task durations are often highly uncertain.

CPM IGNORES “COMMON-CAUSE” VARIATION

In the context of project management, *variation* concerns the inherent uncertainty of task durations. When working with CPM, managers and staff frequently deal with variation (often unconsciously) by introducing so-called “slack” into the tasks which make up the project's schedule. The implicit rule is: *The bigger the uncertainty the bigger the slack.*

There are two fundamental oversights with this approach. The first is the fact that processes actually exhibit *two* types of variation. These are referred to as *special-cause* and *common-cause* variation respectively. This terminology was first coined by Dr. W. E. Deming [Dem00], who's now legendary writings on statistical control were to transform manufacturing, beginning with the Japanese in the 1950s.

Special-cause variation refers to the variation in parts of the process, which essentially makes those parts of the process unpredictable, and which consequently make the process as a whole unpredictable. By way of example, consider the “system test” in software development. Under normal circumstances, “system test” should deliver certain artefacts (test report, bug list etc.) within a given timeframe. However, if on some occasion the test environment fails for some reason (the tester falls ill, the hardware fails etc.), “system test” will become an instant bottleneck, and the software development process as a whole will fail to deliver as originally envisaged.

When considering measures to improve process performance, Deming stressed the importance of eliminating special-cause variation before anything else. This demands that steps are brought under a basic level of control, so that results are delivered predictably and within acceptable norms. Conventional risk-management is used to address the possible causes and consequences of special-cause variation.

However, even when each and every step operates within acceptable norms, the process as a whole will still exhibit a degree of variation. This is referred to as common-cause variation, and once special-cause variation has been eliminated, *it* becomes the key determinant of process predictability. Unfortunately, neither CPM nor traditional project management offers a recipe for dealing with common-cause variation. This means that even if we have successfully brought all the parts of our process under a control, the process as a whole will continue to be unpredictable, with consequences that will range from sub-standard quality to unmitigated disaster. Crucially, common-cause variation cannot be eliminated; indeed any attempt to do so will prove futile – even counter productive. However, we shall shortly see that it is possible to *manage* common-cause variation to great effect.

So distinguishing between variation types is crucial. As Deming wrote: “Confusion between common causes and special causes leads to frustration of everyone, and leads to greater variability and to higher costs, exactly contrary to what is needed”.

CPM's second crucial oversight is that even with consistently performing process steps, the variation exhibited by those steps will always be disproportionately great compared to the variation of the process as a whole. To understand why, we must consider the probability distribution of a typical activity, which makes up the schedule in an uncertain environment.

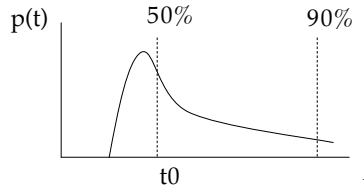


Figure 1: A typical probability distribution for an activity with uncertain duration

Activity durations generally exhibit what statisticians call a *beta probability distribution*. Figure 1 shows a probability graph, which is typical for an activity with uncertain duration. The x-axis shows time, the y-axis shows the probability of task completion at a given time. Note that the probability of completing the task in zero time is zero. Time t_0 is the *mean* or average time it takes to complete the task. By definition this occurs with a probability of 50%. Just to the left of the mean (the peak in the curve) is the most frequently occurring duration – the *mode* in stats terminology.

Note also the “long tail”, which reflects the time required for the task to complete with a high degree of certainty. For many real-world tasks, 90% or 95% certainty corresponds to a duration, which is anything from three or more times longer than the average time – and longer still than the most frequently occurring time.

Figure 2 shows what happens when we string together a bunch of such activities, just as we do when constructing a project schedule.

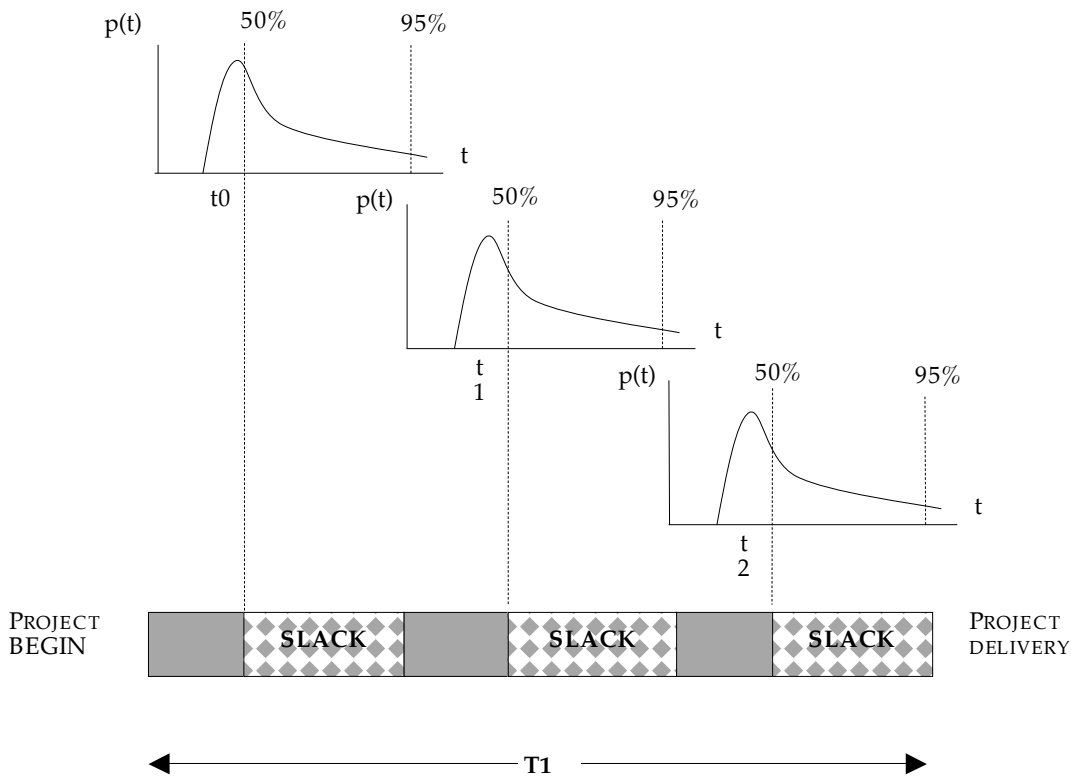


Figure 2: Achieving 90% certainty by adding slack to task estimates

Thanks to slack, there appears to be a reasonable chance that this schedule will deliver the project in time T_1 . However, anyone familiar with how insurance works should have a bit of a problem with this. How likely is it that the worst-case-scenario shows up on *all* of the activities? The answer should be *highly unlikely*. And the likelihood should shrink the more activities we add.

So contrast this with Figure 3, which shows the probability distribution and corresponding schedule of the *sum of the mean* activity durations. This setup exhibits a single aggregated *project buffer*.

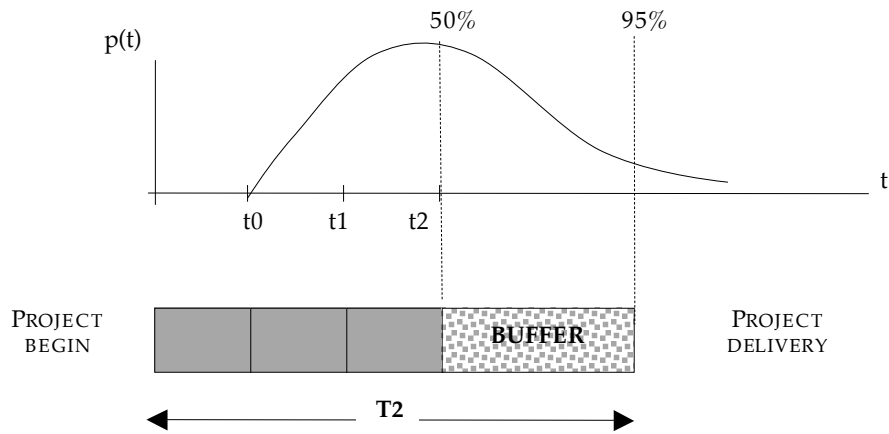


Figure 3: Achieving 90% certainty with an aggregated project buffer

Note that, whilst we have retained 95% certainty of timely project delivery, we also have a significantly shorter schedule i.e. $T_2 \ll T_1$. By the magic of statistics, as long as task variations are more or less independent of one another, the project buffer will be significantly less than the sum of the individual slacks. In other words, variation of a sequence of process steps is typically *considerably less* than sum of variations of individual process steps.

For all practical purposes, the project manager need not concern herself with the math. She simply needs to understand that, like the Theory of Constraints, this is another reality of process behaviour. It is hopefully also clear to the reader that buffer is *not* optional. It is a prerequisite for timely project delivery just as excess capacity in non-constraints is a prerequisite for maximising process throughput.

The good news is twofold. The schedule which includes aggregated project buffer is always shorter than the equivalent CPM schedule, which makes the offering to the customer potentially much more attractive. On the other hand the project manager can feel secure in the knowledge that a high degree of safety is built in.

ON BEHAVIOURAL REALITIES

To many people, all this talk about 90-95% certainty of timely project delivery will sound like desperately wishful thinking – not least to those in the software development industry. Nothing we have explained thus far really explains why certain types of projects *always* end up being delivered late and with great stress. There can be no doubt that project managers build some degree of slack into project schedules – consciously or otherwise. But where does it all go? It is simply a question of buffers being too small?

It is interesting to note at this point that many experienced project managers strongly object to the use of buffers at all. One reason for this is that in a cost-efficiency seeking organisation, the buffer appears to be a source of potential “savings” (ignoring the fact that eliminating the buffer destroys any chance of predictable delivery).

Another reason for eliminating buffer, managers claim, is that people will use up the buffer regardless of whether they need it. Indeed, there is a law, termed *Parkinson's Law*, which states exactly this. What is more, there is a heap of empirical evidence which supports these managers' objections!

If this weren't bad enough, there exists yet another reason why buffer is likely to be abused: The so-called *student syndrome*. Student syndrome occurs because a person's ability to focus on a task is more or less inversely proportional to the time left to complete it. Until the alarm bell starts ringing, the student always finds something else of “higher” priority to work on. Multitasking, which we shall examine in Part III, is an outstanding promoter of student syndrome.

Let us bear these behaviours in mind when we consider the following scenario – a daily occurrence in organisations the world over:

1. A project manager is instructed to put together a project schedule. She cannot determine the complexity of all the project's tasks herself, so she calls on her colleagues to assist her. After some analysis, each of them presents their time estimate of a particular task;
2. She adds up the estimates and – lo and behold – the schedule exceeds the required delivery date. Wanting to appear to always deliver tasks in a timely manner, her colleagues have, in the face of uncertainty, included generous slack in their estimates (remember the “long tail”?). It is not in their interest to state this explicitly;

3. The boss knows the customer will not accept the given schedule. In any case her experience tells her that her colleagues have a tendency to overestimate (although she can't quite explain why). Yet not knowing about aggregated project buffers, and that there is a way to create a shorter, still protected schedule, she "trims" down the estimates, setting delivery dates for individual tasks;
4. Although they have seen this game countless of times before, the workers still feel the need to put up a fight about this. If they don't, they believe, the boss is likely to present them with even tighter schedules in future;
5. With all project members essentially dissatisfied, a final schedule is drawn up.

Needless to say, this mode of operation does not result in the most trusting, collaborative working atmosphere. Everyone is gaming the system, trying (unconsciously) to protect him or herself from uncertainty. What is more, individually, everyone is behaving absolutely rationally. Collectively the result is a disaster. Workers not only have an incentive to present even longer task estimates in future, they have every incentive to use up the time they are given. *Nobody has any incentive to deliver a task result early.* To compound matters, to those that have seen their estimates slashed by the PM, the schedule lacks credibility. In the worst case, commitment to the project takes a downturn.

So once Parkinson's Law, student syndrome and worker disaffection inevitably kick in, project delay, overtime and/or project cancellation are virtually inevitable.

BREAKING THE BEHAVIOURAL DEADLOCK

So it appears that we have two alternatives:

1. Accept that these kinds of behaviours are a fact of life and ban all talk of buffers from the organisation;
2. Give people an incentive to deliver task results on completion of the task, as opposed to when the allotted time is used up or exceeded.

Let's write off the first option here and now. Duration uncertainty is a reality and the buffer an indispensable, efficient tool to protect projects against it.

To achieve the second alternative, the PM needs to create an environment which encourages new behaviour. The following outlines this:

- The PM and the workers work together to estimate *average* task durations. Average task durations are defined as *aggressive but achievable*. They are generally slightly higher than best-case, significantly lower than worst-case estimates. The PM records the slack-time of each task as worst-case minus best-case.
- The PM builds a schedule based on average durations, and then estimates the size of the project buffer using some function of the sum of the slack-time durations or using the 50%-rule described below. The project buffer will be significantly less than sum of all slack-times. The schedule is acceptable from the customer's perspective, feasible from the PM's perspective and credible from the worker's perspective.
- Workers are instructed to do their best to complete the task according to the average duration estimate. If the task is completed early, it must be handed over immediately (think *relay-race*). Equally importantly, if a worker completes a task later than the average duration, *he will not suffer any consequences*. Remember: By definition of expected or average duration we expect fifty percent of tasks to be delivered *before* and fifty percent *after* the expected time. Consistent timely delivery of tasks strongly suggests over-estimation.
- Workers and the PM operate as a team working towards a common goal, which is to meet or beat the *project delivery date*. In contrast, *task delivery dates* are considered irrelevant and the PM never emphasises them.

Cultures are not changed over night. It would be naïve to assume that an organisation can suddenly switch from being *task-date-driven and individual-focused* to being *project-date-driven and team-focused*. Yet this is what must (and can) be done if major gains are to be achieved – a fact proven again and again in thousands of organisations in recent years.

ADDITIONAL PROTECTION FOR THE CRITICAL CHAIN

Projects rarely consist of a single sequence of dependent tasks. Most of the time there are inputs coming from subprojects. When should work on a subproject be started?

CPM – and the project management tools that support it – promote the idea of starting such work *as soon as possible* (it is the default on newly created tasks in MS Project, for example). The idea is that by starting work as early as possible, we reduce the chance of negatively impacting critical path.

This approach is dissatisfactory for a number of reasons. Firstly, it ties up resources before they are really needed. These resources could be doing other things, like working on *other* projects. Secondly, from a financial perspective, it is clearly a disadvantage to have to commit capital to an undertaking any sooner than necessary. Finally, something regularly witnessed in software development, *requirements change over time*. So beginning work on a set of requirements too early increases the risk of having to redo the work later on. This all adds up to waste.

To protect the current project from delays occurring in subprojects, we use a so-called *feeding buffer*, such as shown in Figure 4.

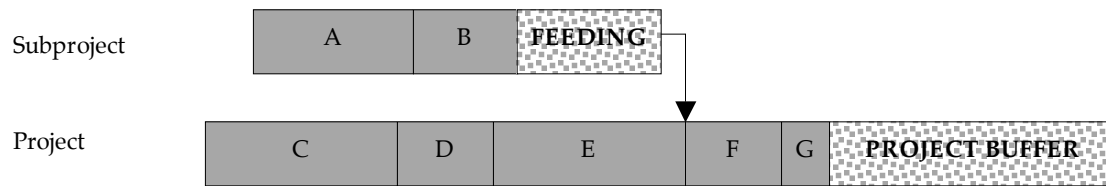


Figure 4 Project and Feeding Buffers in Action

Feeding buffers therefore answer the above question as follows: Work on subprojects should be started *as late as possible*, but their schedules are buffered to avoid endangering the critical chain. Once again, we witness a radical departure from traditional PM thinking.

THE 50%-RULE: A SIMPLE APPROACH TO BUFFER SIZING

Underestimating the size of project and feeding buffers endangers the project's timely delivery, which in turn increases project risk. Overestimating the size of project and feeding buffers reduces project viability. Clearly, the buffer size needs to lie between zero and the most pessimistic estimate.

A simple but effective approach recommended by many critical chain practitioners is simply to use 50% of the sum of the slacks as the buffer size. This approach, shown in Figure 5, has proven itself workable in many projects. It has the advantage of being extremely simple, requiring no knowledge of statistics. An even simpler approach is to use 50% of the critical chain.

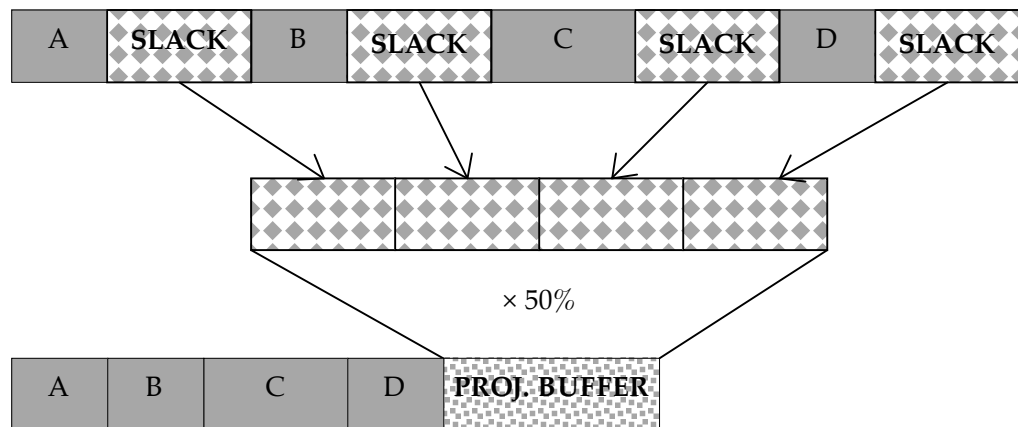


Figure 5 Goldratt's simple approach to buffer sizing

No matter which rule is chosen, it can also be applied to feeding buffers. Alternative and more sophisticated techniques are described in [Lea05]. The best rule of thumb for an organisation new to critical chain, however, is to *keep it simple*.

SUMMARY OF PART II

- In project management, the process's constraint is called a *critical chain*. It is the sequence of tasks, which together dictate the project's delivery date.
- Two types of variation have an impact on the duration of a project: Special-cause and common-cause variation. Special-cause variation occurs when the process behaves abnormally, such as when illness or breakdowns occur. Common-cause variation is the overall variation resulting from the interplay of elements in the process.

- It is essential to make an explicit distinction between special-cause and common-cause variation, because the techniques for managing them are necessarily fundamentally different. Special-cause variation is always dealt with first. Its causes and consequences are addressed using conventional risk-management. (We do not use buffers to deal with special-cause variation!)
- Common-cause variation is addressed by using an explicit project buffer. By using aggressive but achievable (average) task duration estimates in combination with the project buffer, we end up with a schedule that is shorter than a naively estimated (traditional) schedule, yet which is protected. Subprojects are prevented from endangering the main project's timely delivery by using a *feeding buffer*.
- Project teams must adopt *relay-race* behaviour, handing over work as soon as it is done. Team focus is on the project deadline. Task delivery dates are not emphasised.
- A simple approach to buffer sizing is the 50% rule.

For more information, please contact: Elke Gemperlé (Email: elkegemperle@cuttingedge.ch)

For book recommendations, references and links, check out <http://www.cuttingedge.ch/resources>

REFERENCES

[Dem00] W. E. Deming, *Out of the Crisis*, MIT Press, 2000

[Gol84] E. Goldratt, *The Goal*, The North River Press, 1984

[Gol97] E. Goldratt, *Critical Chain*, The North River Press, 1997

[Lea05] L. P. Leach, *Critical Chain Project Management, Second Edition*, Artech House, 2005