Presentation
Paper
Bio
Return to Main Menu

P R E S E N T A T I O N

F3

Friday, February 16, 2001
9:30AM

BUFFERS & RISK: CRITICAL CHAIN PROJECT MANAGEMENT

Robert Muller
Cytokinetics, Inc.

International Conference On
Software Management & Applications of Software Measurement
February 12-16, 2001
San Diego, CA, USA

Critical Chain Project Management

Robert J. Muller

Cytokinetics, Inc.

+1 650 624 3080 rmuller@cytokinetics.com

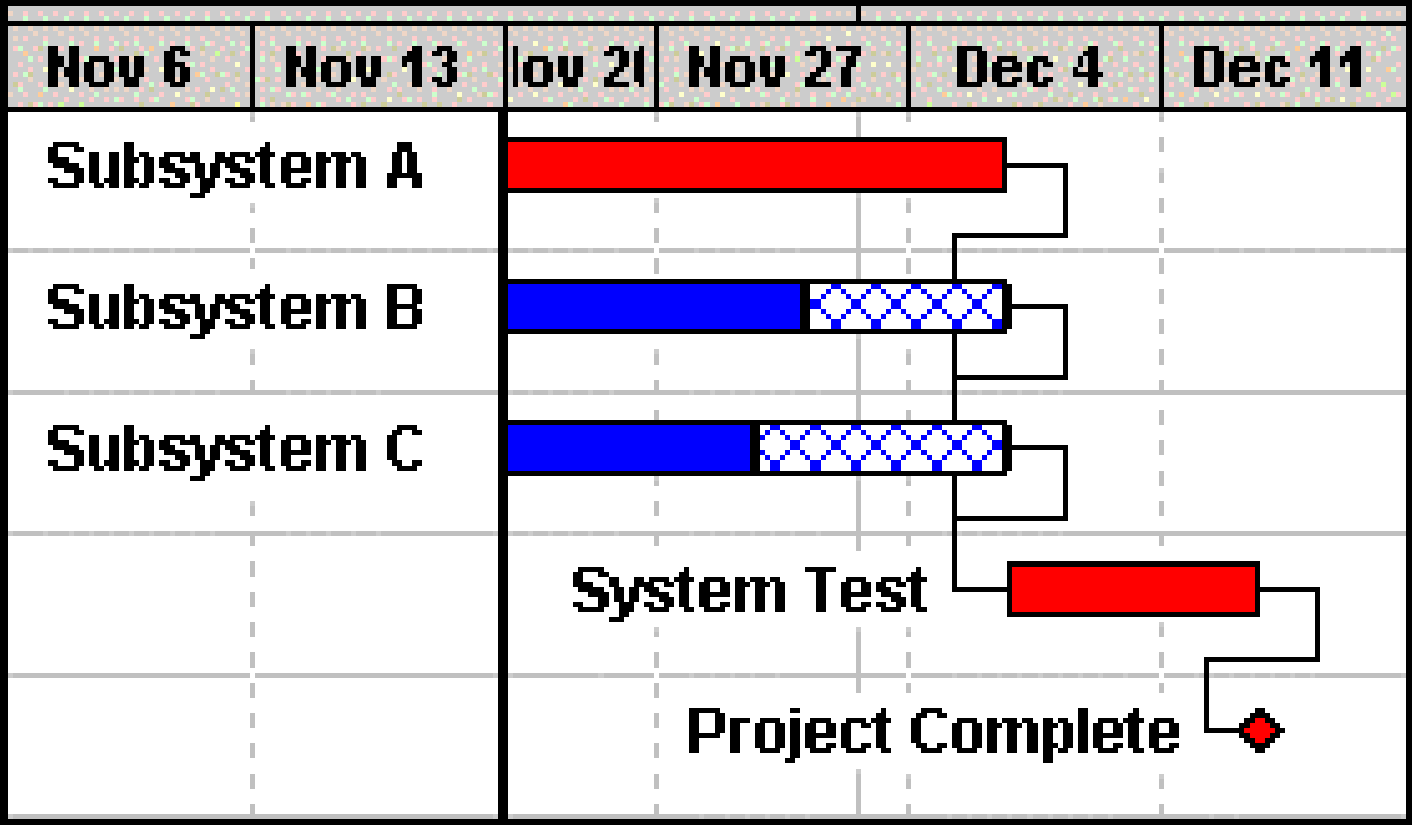
SPC and Project Management

- Critical Path
- Risk Analysis (Monte Carlo)
- Where is common-cause variation?

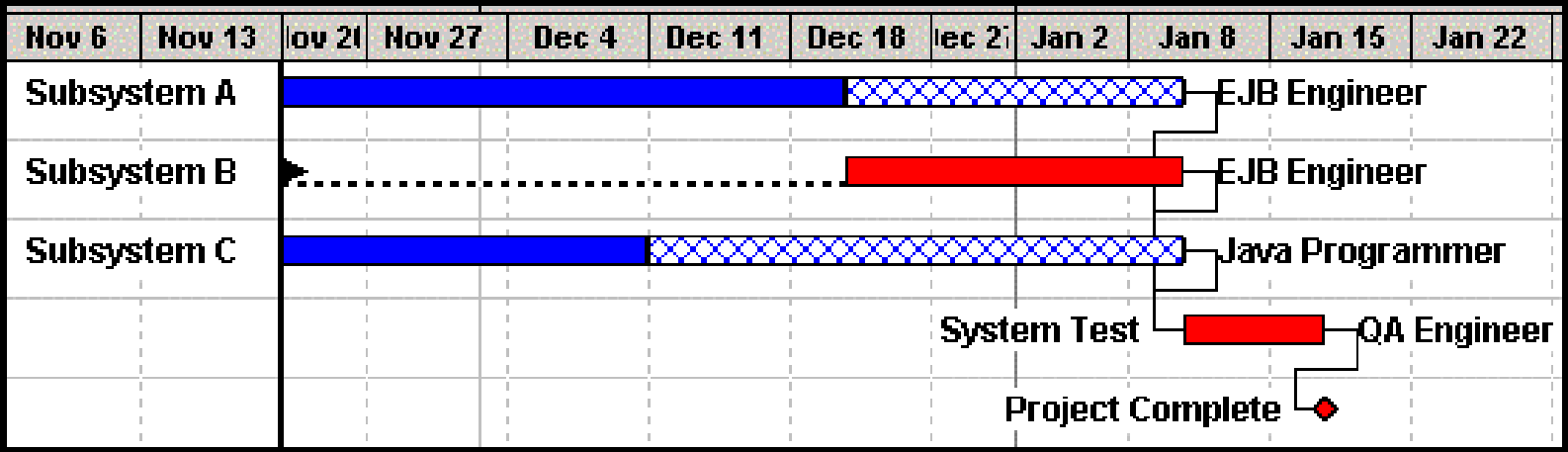
Process and Task

- Process: series of operations performed to create a product
- Project Task: a unit of work in a project that depends on other tasks
- Cost, schedule, and quality: the three horns of the project manager’s dilemma
- Management by task: controlling the project on a task-by-task basis

The Critical Path



The Leveled Critical Path



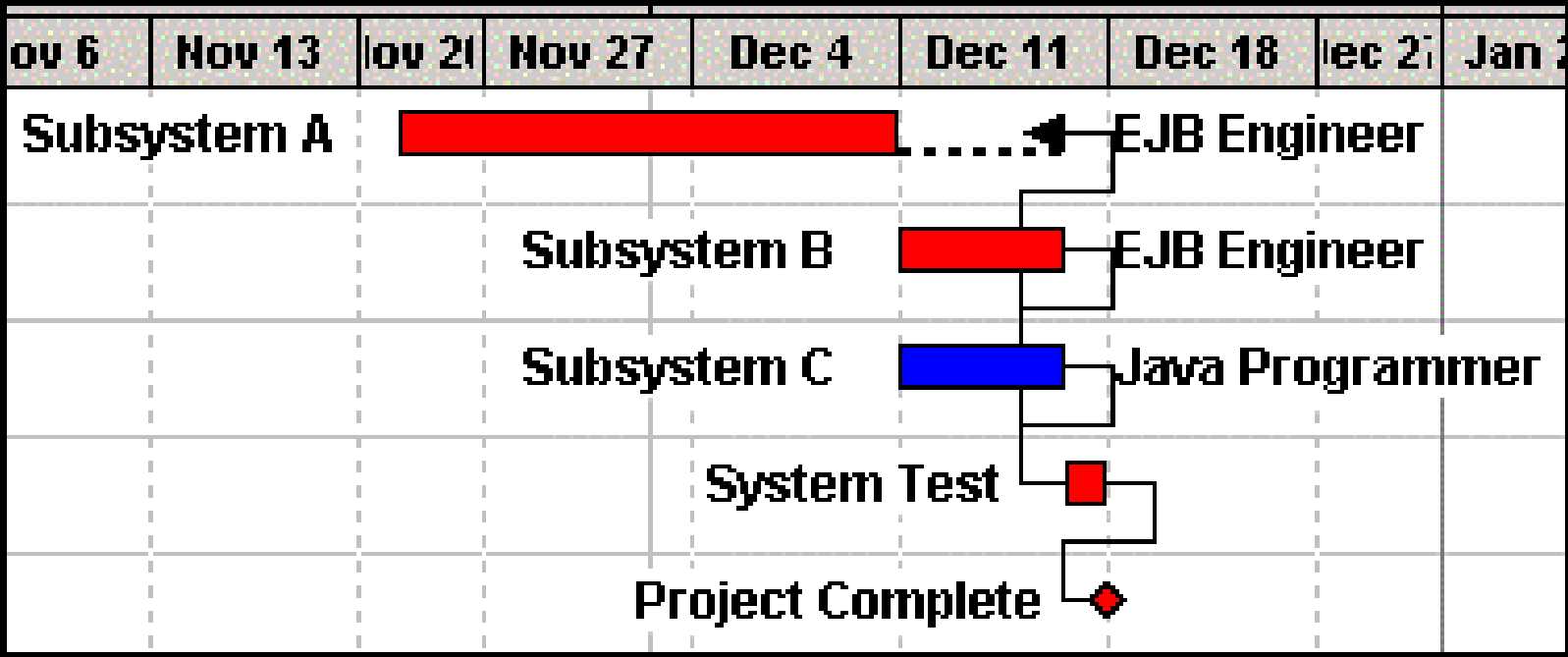
Variation

- Common causes of variation
  - Latent rather than explicit causes
  - Reduce variation by improving capability
- Special causes of variation
  - Identifiable and controllable
  - **RISK**
  - Reduce variation by avoiding or mitigating risk

Critical Chain Project Management

- Leveled resources and managing variation, not risk
- Remove unnecessary fudge time
- Add variation buffers
- No multitasking
- Late start scheduling
- Roadrunner behavior

The Critical Chain



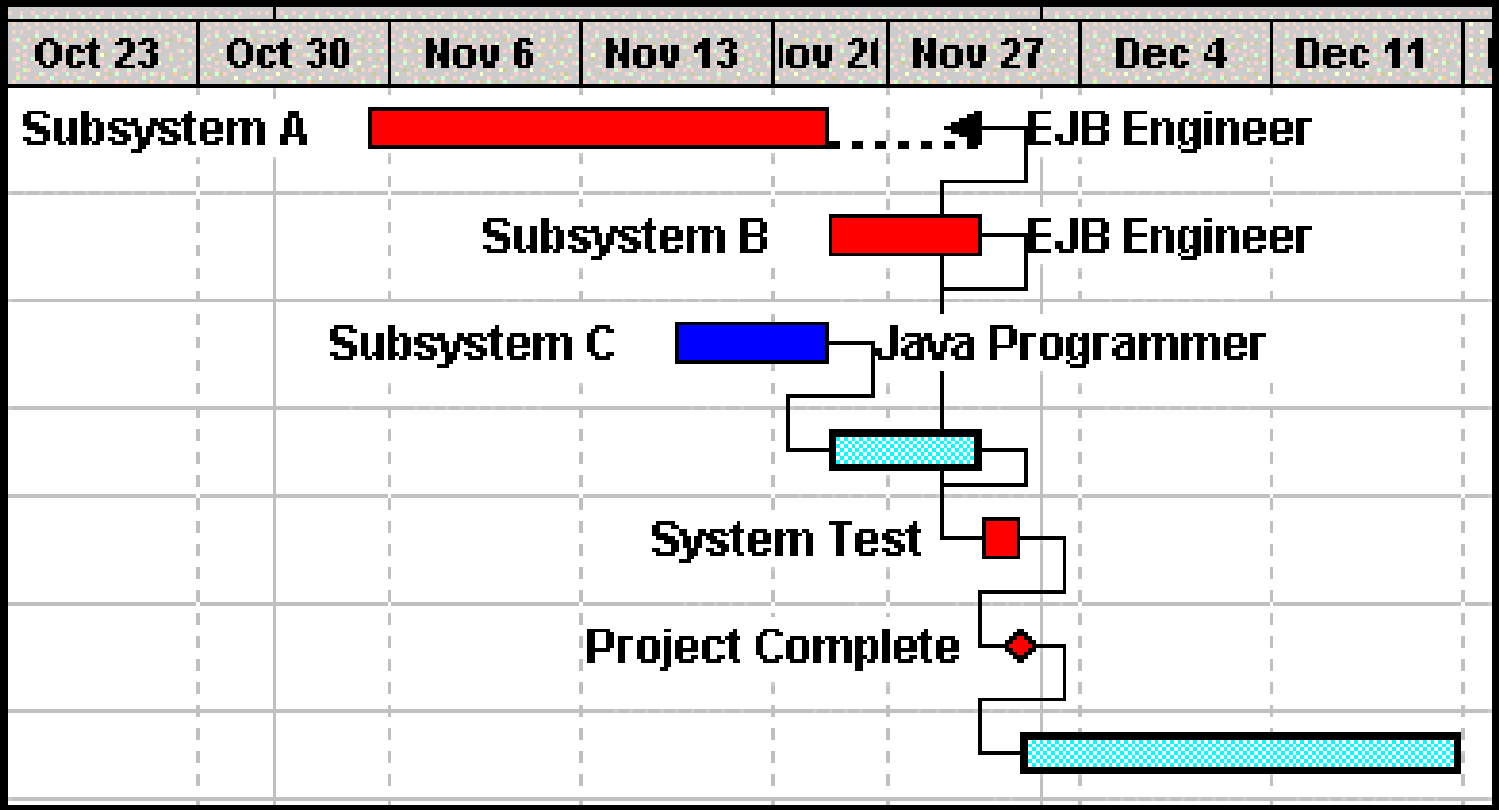
Constructing the Chain

- Identify and estimate effort for tasks and dependencies and a fixed-date project completion milestone.
- Assign resources based on skills.
- Level resources to availability.
- Revise project to fit needs.
- Compute critical chain as path with longest duration through the project.

Buffer the Tasks

- Project Buffer: Create a buffer for the project.
- Feeding Buffers: Create a buffer for each task that feeds into the critical chain.
- Re-level the resources to account for buffering changes to the plan.

The Buffered Project



Calculating Buffers

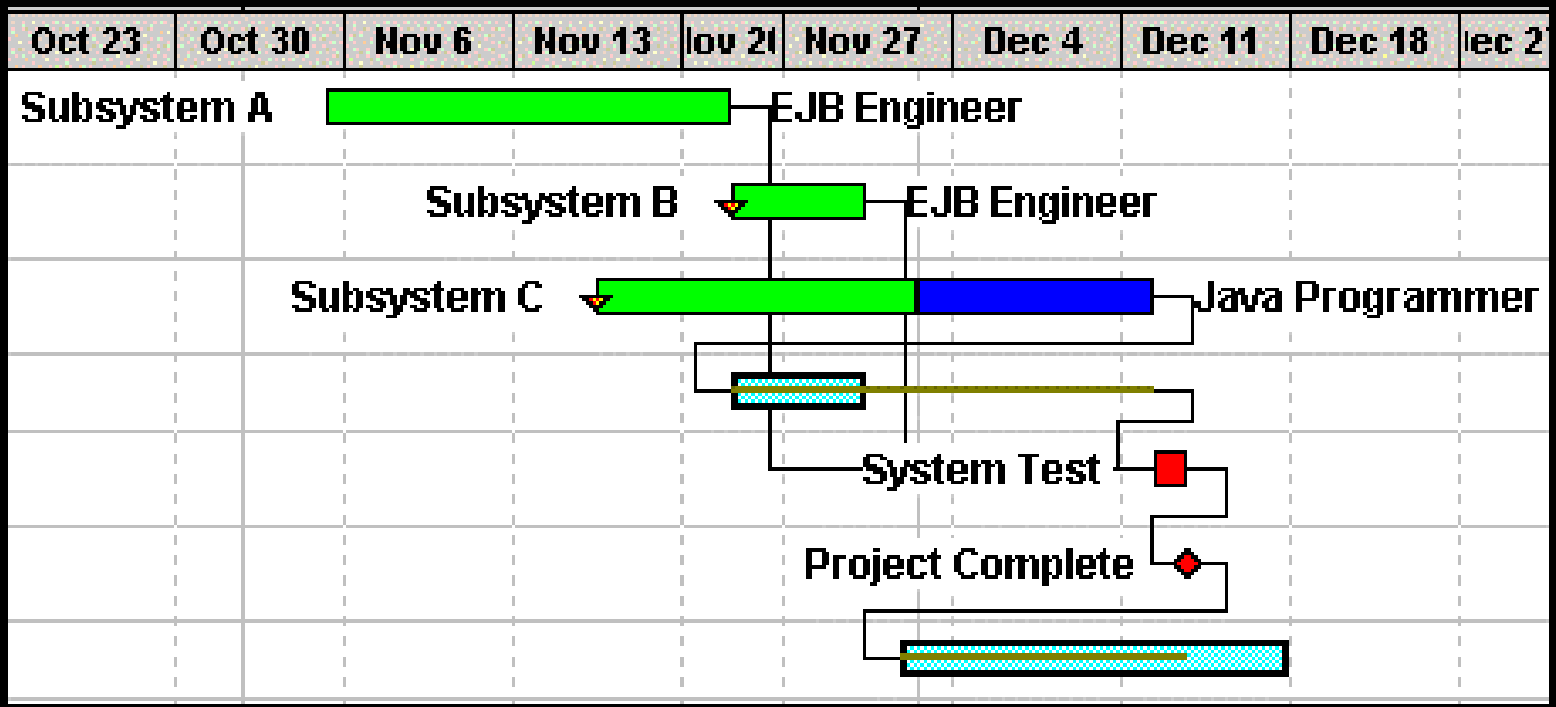
- **Rule-of-thumb method:**
  one-half the length of the chain the buffer terminates
- **Sum of Squares method:**
  sum the squares of the task lengths, then take the square root

Sum of Squares Example

A		B		Test	
50	**90**	**50**	**90**	**50**	**90**
12	20	4	12	1	3

SQRT (8 * 8 + 8 * 8 + 2 * 2) = 11.5

Tracking the Project



Multitasking

- Decreases focus
- Decreases productivity
- Increases variation
- Increases impact of special causes through additional dependency

Road Running

- Late start puts emphasis on priorities rather than ever-changing start date
- Road running moves responsibility for managing time to the resource instead of the project manager

Improving Capability

- Process capability is the ability of the process to meet a target with a certain level of variation
- You can reduce common cause variation by improving capability, not by managing risk

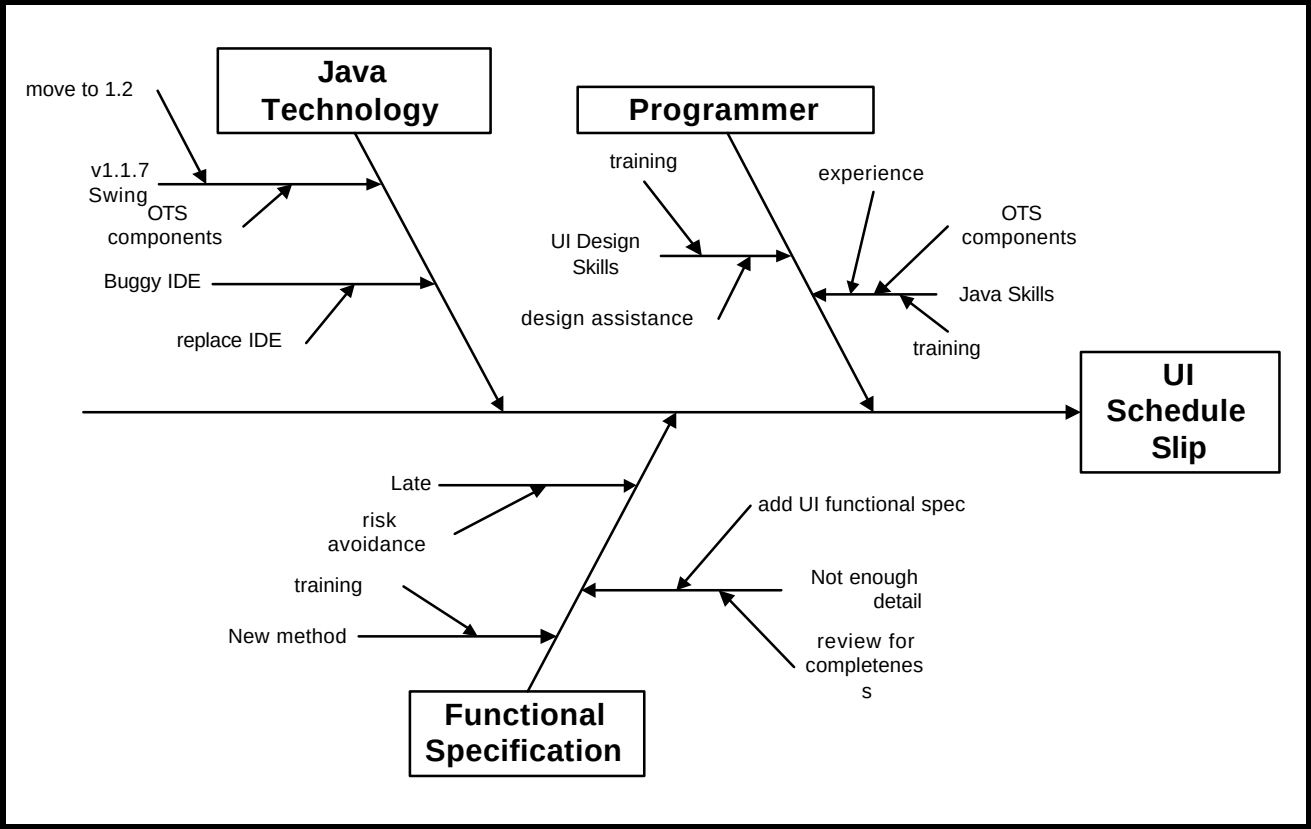
Risk as Special Cause

- Risk is the probability of a significant negative impact on the project.
- Special causes contribute the risk to the project.

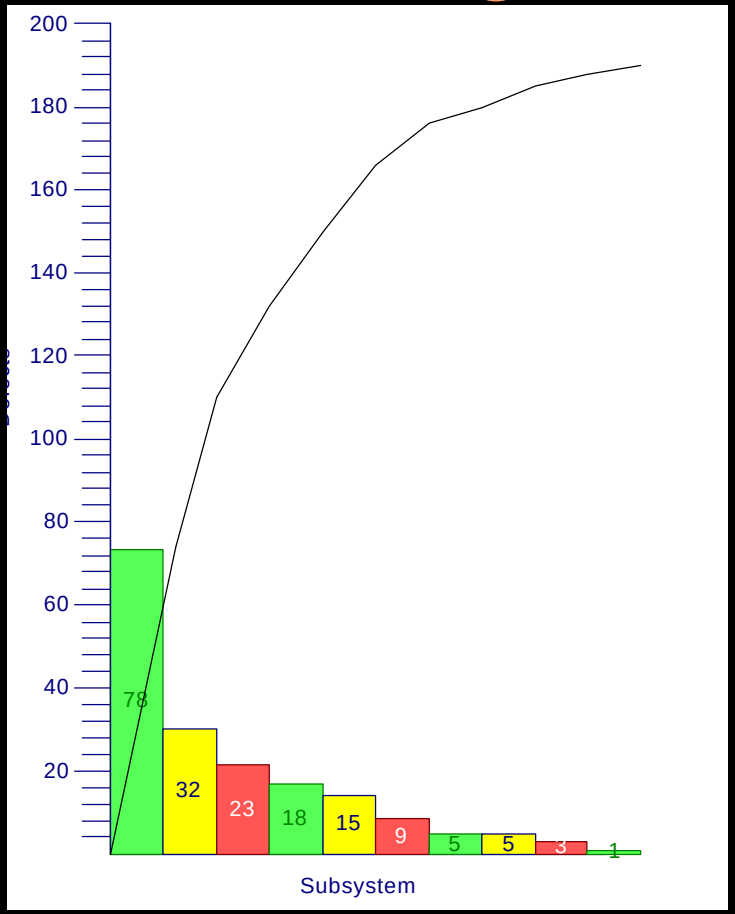
Managing Risk

- You manage special causes by
  - Avoiding them
  - Mitigating their effects
- You can use SPC tools to identify special causes.
- You should identify risk long before the task is ready to go.

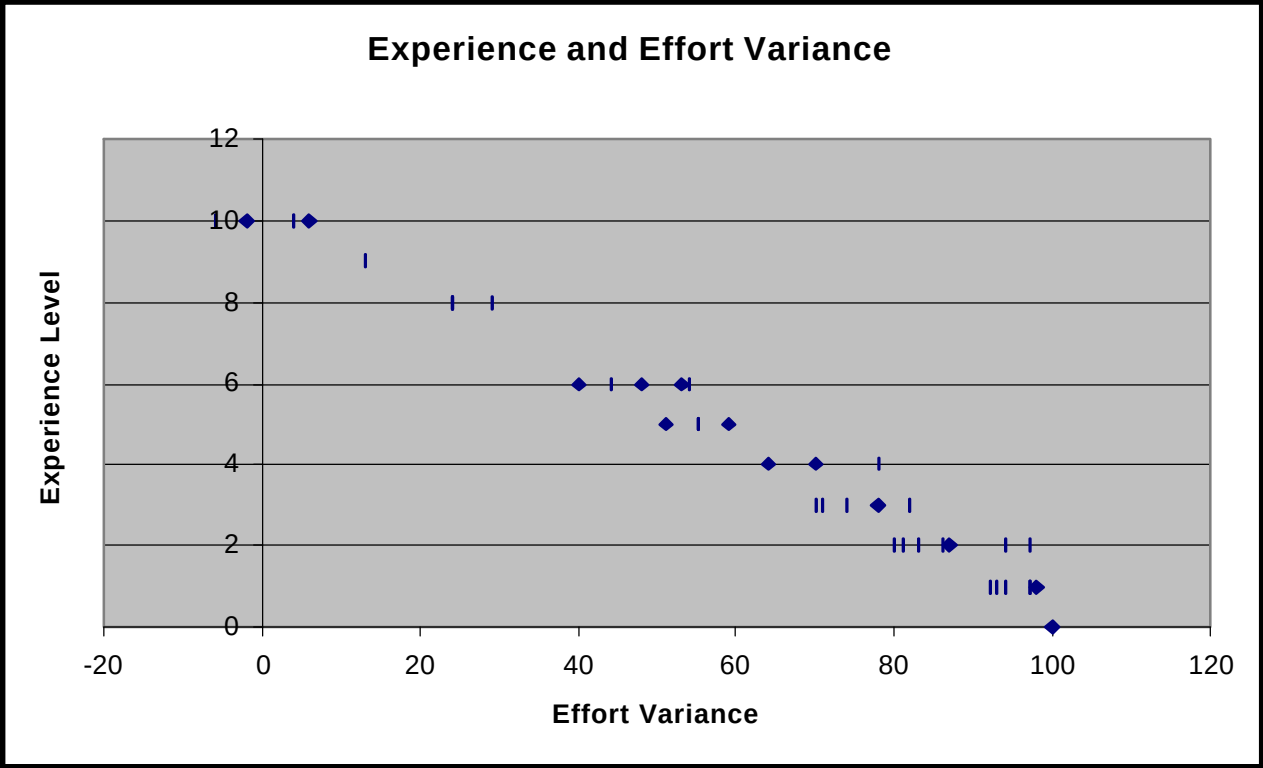
Cause and Effect Diagram



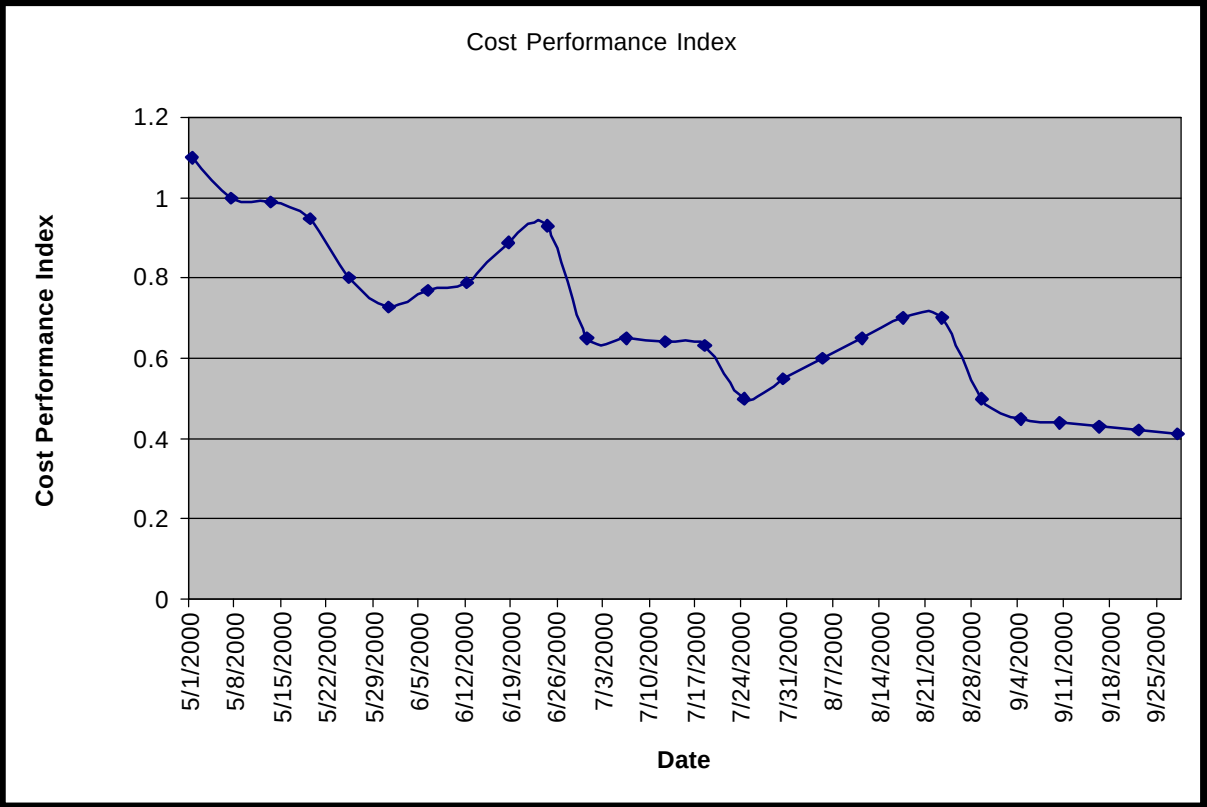
Pareto Diagram



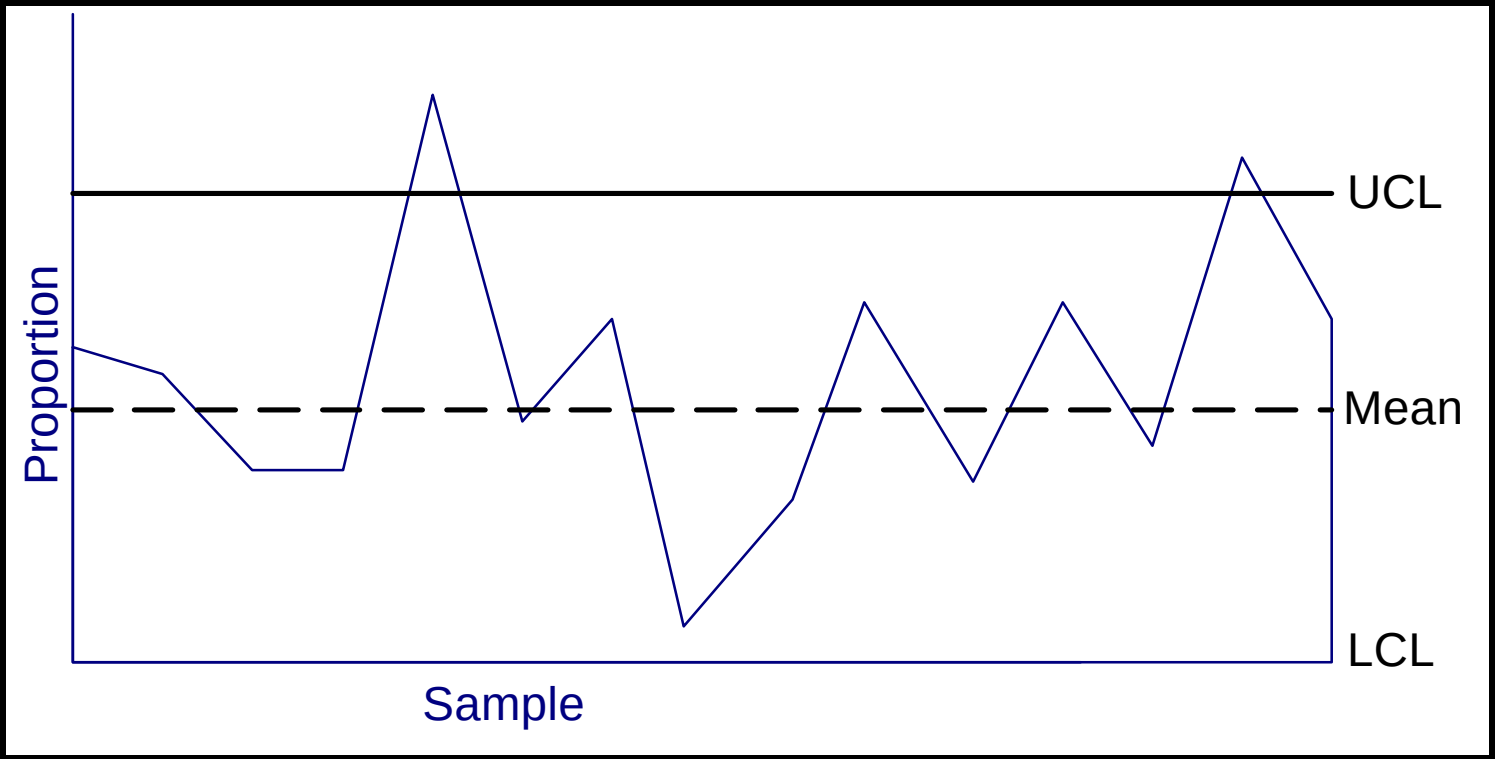
Scatter Plot



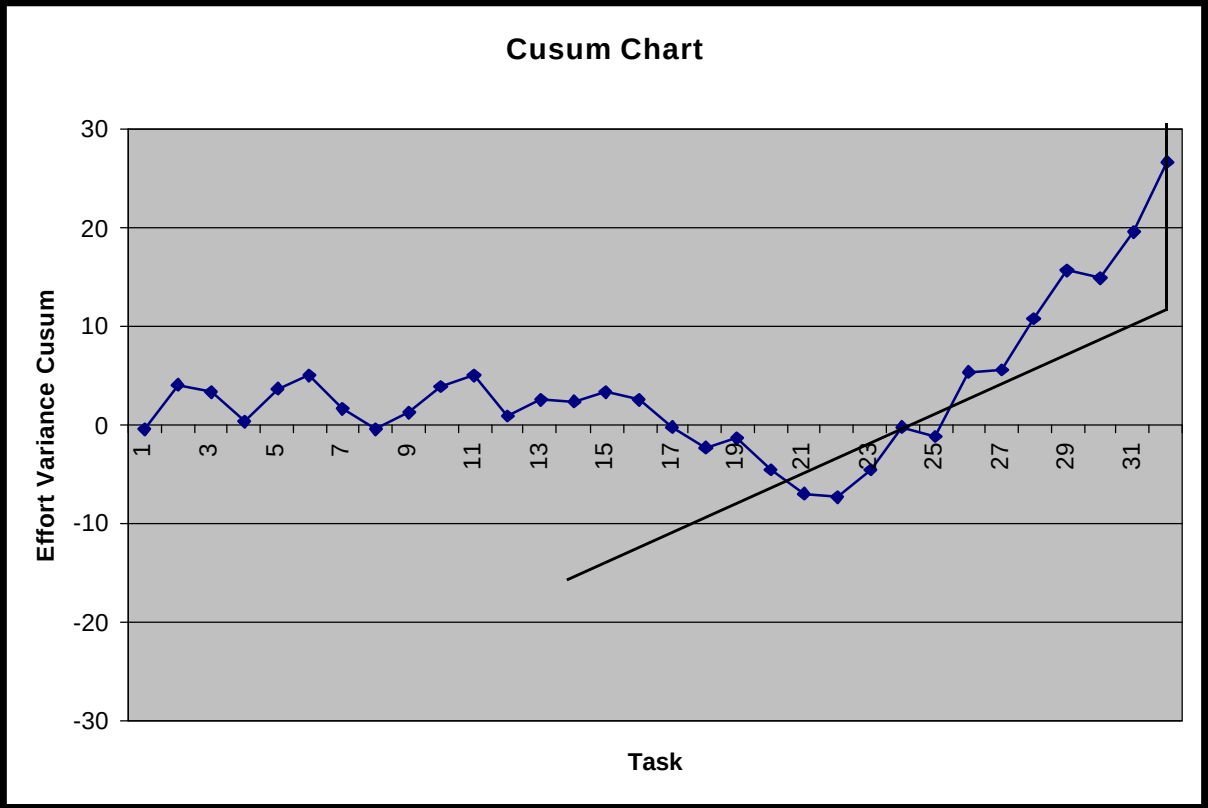
Run Chart


Cost Performance Index
Cost Performance Index
1.2
1
0.8
0.6
0.4
0.2
0
5/1/2000
5/8/2000
5/15/2000
5/22/2000
5/29/2000
6/5/2000
6/12/2000
6/19/2000
6/26/2000
7/3/2000
7/10/2000
7/17/2000
7/24/2000
7/31/2000
8/7/2000
8/14/2000
8/21/2000
8/28/2000
9/4/2000
9/11/2000
9/18/2000
9/25/2000
Date

Control Chart



Cusum Chart


Cusum Chart
Effort Variance Cusum
30
20
10
0
-10
-20
-30
1
3
5
7
9
11
13
15
17
19
21
23
25
27
29
31
Task

Questions?

Buffers and Risk

Critical Chain Project Management

Robert J. Muller

Cytokinetics, Inc.

Abstract

Critical Chain Project Management uses statistical process control to more clearly define the relationships between tasks and the project end date by using buffers to represent process variation in project tasks. This paper shows how integrating effective statistical process control with the use of function points and standard processes can dramatically improve your ability to plan and control projects. Using examples based on real-life experience, it provides a simple method for controlling variation and risk.

SPC and Project Planning

If you've managed many projects, you're probably well aware of the problems associated with schedule and cost risk. What you probably don't realize is that you're mostly not dealing with risk at all, but *variation*. Manufacturing process professionals have the jump on project managers here because they've been dealing with process variation since manufacturing began five thousand years ago.

For whatever reason, project managers are held to a standard few are likely to reach—the perfect project, under budget and on time. Project management theory and practice [PMBOK] assumes a deterministic approach to management that doesn't reflect the probabilistic reality we all face dealing with everyday events. A *critical path schedule* is a plan that takes into account the work to be done, the dependencies of one task on another, and the carefully estimated time and money it will take to do each task. You can use "risk analysis" in the form of Monte Carlo simulation to figure out how likely your estimates are. Once you know where your schedule and budget risks lie, you can increase the estimated time to eliminate the risk of failure. Sometimes this approach works, and sometimes it doesn't. In software project management, mostly it doesn't, at least in my experience.

In the last few years, a new way of looking at project management has been introduced—critical chain project management.

Process Variation in Project Tasks

A *process* is a series of operations you perform in the creation of a product. Statistical process control (SPC) is the application of mathematical statistics to controlling variation in processes in pursuit of continuous improvement [Bissell]. Processes have inputs and outputs, and the variation in the inputs and the process affects the quality of the outputs.

A *project task* is a special kind of process that takes place in the context of a project. The project manager breaks the overall project down into a series of tasks, units of work, connected by dependencies based on inputs and outputs.

Project managers must control three aspects of the project: cost, schedule, and quality.

A common way of controlling these variables is to manage them at the task level. That is, project managers look at how much a task costs, how long it takes, and how much value the outputs contribute to the project in order to control the ultimate result of the project.

Comparing this to manufacturing, a manufacturing supervisor would control his or her processes by controlling each piece of work. But that's not really possible or useful, as SPC points out. Individual pieces of work, like almost any system, have an inherent variation based on the capability of the process, ultimately due to entropy (the second law of thermodynamics). SPC defines process control as controlling variation and improving capability.

Moving from manufacturing back to the project, how does the concept of statistical process control apply to tasks? Tasks, like any dynamic system, have an inherent

variation with respect to their control variables of time, money, and quality. You cannot control this variation by sitting on each task individually because of the nature of variation: it varies. Trying to control each element in manufacturing leads very quickly to over-control, which in turn leads to low process capability, longer and costlier projects, and lower quality. In the case of projects, this translates to tasks that take too long, cost too much, and yield mediocre quality at best. Instead, if you use some basic tools from SPC, you can control your project at a higher level. Before getting into details, however, you need to understand a couple of different kinds of variation.

Special Causes versus Common Causes of Variation

Shewhart, Deming, Juran, and all the other gurus of SPC distinguish between two basic sources of variation in systems: special causes and common causes.

Special causes are factors that you can identify as direct causal agents of variation and that you can control by correcting or eliminating the cause. In manufacturing, special causes take the form of broken machines, untrained operators, and poor quality lots of input materials. In software projects, special causes are things like tool defects, immature technologies, lack of effective requirements elicitation, and so on. Special causes are the risk events that software project managers assess and mitigate or eliminate through risk management.

Common causes are latent or uncontrollable factors that perturb the system; thus, common causes are not controllable through correction or elimination of the cause. You either don't know the cause, or you can't control it. Common causes are relative to the state of the art of the process—the technology, the level of automation, or whatever affects process capability. Within the current capability, these common causes produce the level of variation you must accept. To improve, you must improve the capability of the system. In software projects, the technology level and choices you make, and the architectures within which you operate, impose direct limits on cost, time, and quality of the project tasks. Common causes are the fudge factors, project budget set-asides and reserves, and "good-enough" quality criteria that software project managers use even though they often feel badly about doing so.

Process Variation, Buffers, and Capability

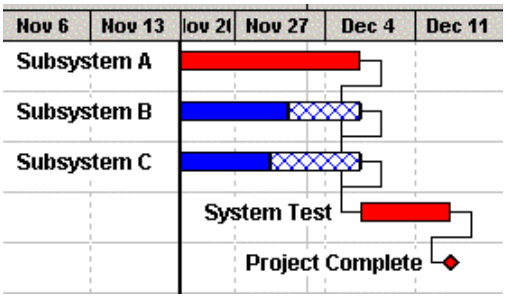
Critical chain project management [Goldratt, Leach] formalizes the SPC approach to controlling schedule and cost (but not quality) through the adoption of several control techniques:

- Calculating a critical chain (not a critical path) based on leveled resources (not just task dependencies) and managing variation relative to that chain
- Removing unnecessary "fudge" time and effort from task estimates
- Adding project, resource, and feeding buffers to projects to control common cause variation in schedule and budget
- Eliminating multitasking in projects
- Using late start task scheduling and "Roadrunner" responsibility management

Resource Leveling, Buffers, and the Critical Chain

After constructing the logical network of tasks and dependencies, critical path project management is done: you've identified the critical path through the tasks. Unfortunately for project managers, knowing the critical path doesn't help very much, for a simple reason: it doesn't take resource availability and skill into account.

Consider these tasks, typical of a software project:

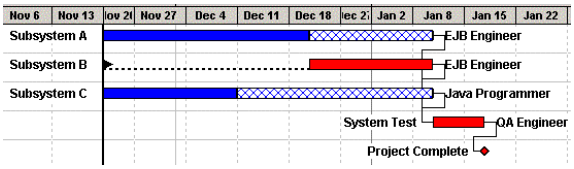


Subsystems A and B are Enterprise Javabean (EJB) systems that access a relational database. Subsystem C is an application that provides a user interface for A and B and that is developed in parallel with A and B. The system test integrates these systems and tests them against the requirements. The figure shows the critical path after initially estimating duration: Subsystem A, Subsystem C, and System Test. The conclusion: focus

your efforts on subsystem A and the system test and your project will complete on time.

Unfortunately, subsystems A and B require a special developer, one who understands EJBs and database programming. Subsystem C requires a Java user interface programmer. You have one EJB engineer and one new hire who wants to learn Java GUI programming, and the EJB engineer spends half of his time training other engineers in the company in the new technology. Also, marketing insists on a product launch date of December 15 to coincide with a major industry trade show at the North Pole.

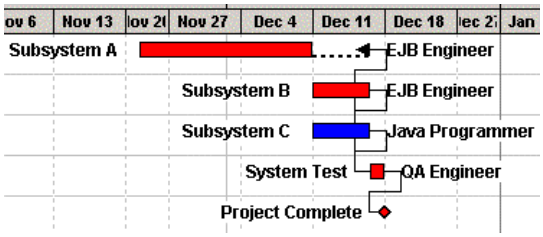
If you managed your schedule according to the critical path in this project, you're doomed from the start. Here is the project with "realistic" estimates and resource leveling (adjusting task starts based on the actual availability of resources given conflicting tasks):



As you can see, this project plan bears little relation to the standard critical path project plan. A and B have much longer durations because their resource is only available 3 hours a day. B must happen after A rather than in parallel because there is only one EJB programmer. The duration of C is longer because the novice programmer will need twice as much time to figure things out. The critical path is now B and System Test, not A and System Test, and notice there is no critical path at all for the month of December. This is where critical chain planning starts.

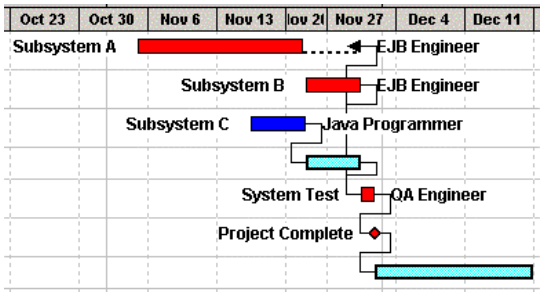
First, reduce the estimates for the tasks back to the industrial effort estimates, eliminating the fudge factors and the allowance for training. For example, the safe estimate of 20 days of full-time work for Subsystem A given the EJB programmer's schedule should really be the median estimate of 12 days. Level the tasks, delaying the start of subsystem B until A is done. Also, extend the duration of Subsystems A and B to accommodate the half-time availability of the EJB engineer, as the company can't afford to dispense with her training activities. Finally, start the tasks at the last possible start date that will allow finishing by the desired date.

The result is this schedule:



Now, mark the tasks along the path through the network that has the longest duration. That is the critical chain, as the chart above shows: Subsystems A and B and System Test.

Next, create a project buffer for the critical chain and a feeding buffer for the connection of non-chain tasks into the chain:



The project now starts at the beginning of November. The Project Complete milestone is on 11/30, and there is a project buffer from 11/30 to 12/15 of 11.5 days.

Note: Cost budgeting works just like schedule budgeting: add buffers (reserves) to the critical chain based on cost as opposed to duration and to tasks feeding into the cost critical chain after allocating resources. Similarly, manage value by measuring output and quality and buffering the critical chain that adds the most value to the project.

Constructing Buffers Statistically

The buffers you add to the project let you manage the common causes of variation in the tasks on the critical chain. Therefore, the best way of allocating time to these buffers is to base the size of the buffer on the variance of your processes.

The rule of thumb is to make each buffer one-half the length of the chain of tasks it terminates. For example, if your critical chain is 60 days, you should construct a 30-day project buffer.

A slightly more sophisticated approach requires some additional calculation. First, you need to estimate two durations for each task: the low-risk duration and the 50% duration. The low risk duration is the one you would commonly use in a regular critical path project, and is usually something like 90% probable. The 50% duration is the duration that has 50% chance of occurring. Take the difference between the two durations. This value is proportional to the standard deviation of the task. Assuming that the proportion is the same for all tasks, you then take the square root of the sum of squares of the values and use that for your buffer size.

In our continuing example, Subsystems A and B and system test are the three critical-chain tasks. This table shows the 90% and 50% estimates in days for each task:

A		B		Test	
50	**90**	**50**	**90**	**50**	**90**
12	20	4	12	1	3

Note that these estimates take into account the resources assigned to the tasks; it is essential to work with estimates that reflect the availability and skill of the resources. (We'll ignore C and our poor Java novice for now, but we'll return to her later.)

The differences are 8, 8, and 2, respectively. The square root of the sum of squares is 11.5:

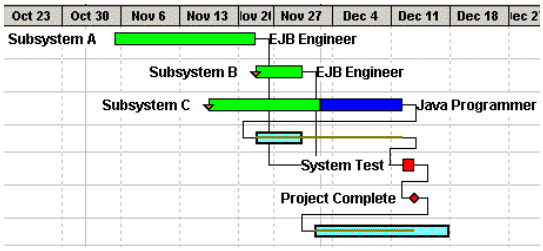
SQRT (8 * 8 + 8 * 8 + 2 * 2)

So the project buffer is 11.5 days. Using the rule of thumb, the buffer would be one-half of the 50% estimates: (12+4+1)/2 = 8.5 days.

Obviously, the better your estimates and their probabilities, the better your buffer sizing will be. Improving your estimation capabilities will improve your ability to reduce task and buffer estimates and deliver. As you gather data, you can develop estimating equations that will give you real probabilities based on frequency analysis and/or Bayesian posteriors that you can use to provide accurate medium- and high-risk estimates.

Again, these buffers represent the common cause variation in your project. They do not represent special causes; that requires risk analysis and management.

When you track the project, as you increase the actual duration of tasks and delay tasks past their late start dates, the chains push into the buffers.



When a buffer is one-third consumed, you need to start assessing risks for the task; when a buffer is two-thirds consumed, you should start implementing your risk mitigation plan. The special causes are winning at this point.

Multitasking, Late Starts, and Road-running

Multitasking in the project context occurs when a resource works on two tasks at once, switching between tasks as needed. Most project managers faced with resource limitations level their projects assuming people will be working on more than one thing at a time.

Critical chain project management prohibits multitasking for two reasons. First, multitasking leads to a lack of focus and reduces productivity. Think of the overhead of multitasking programs and their context switching. Projects are real-time environments that simply cannot afford context-switching overheads. More to the point, overlapping tasks increases the variation in the tasks, often leading to much longer task times and certainly leading to much longer projects than might otherwise be the case.

I add a third reason for not multitasking: good project design. Multitasking renders two tasks interdependent, coupling them based on the fact that one person is working on them simultaneously. Problems in one task then affect the other through the medium of the assigned resource. Good design uncouples the tasks by eliminating such multitasking. This makes risk analysis much easier.

Critical chain project management also schedules the start of a task as late as possible. Critical path project management usually schedules tasks to start as early as possible on the assumption that earlier is better than later because you get done earlier. What happens is just the reverse: because early-start tasks have slack at the end, the Peter Principle kicks in to allow the resource to think that the duration is not critical and therefore they can go easy until the task runs up against the critical path.

Instead, critical chain project management removes any notion of post-task slack time, buffers the tasks for

control, and emphasizes personal responsibility for resource time management: road-running. The Roadrunner mentality looks at the set of assigned tasks and immediately starts charging ahead on the most critical. In projects I've managed, many tasks get done long before they were scheduled to start. That leaves time to manage the buffer-eating tasks, often leaving key resources free that you can move onto a task without fear of incurring Brooks' Law and making the late task later.

Improving Process Capability

Before moving on to the specifics of constructing a critical chain project, it is worth spending a little time on continuous improvement.

As with any application of SPC, the process assumes that you first eliminate special causes, then manage your process to get it under statistical control. This strategy has two implications for software process capability.

First, you must do risk analysis, or you won't be aware of your special causes and your buffers won't be effective.

Second, once you've eliminated the special causes, you're now left with your common causes. This may seem obvious, but there's a problem. It may be that your process capability is simply insufficient to get you where you need to go. Project management can manage variation, but it can't reduce it beyond the common-cause variability. If this isn't good enough for your projects, you have a capability problem. Your process can't deliver the goods with the variation you need to succeed.

For example, if you work through the estimates and construct your buffers, and your start date comes out as sometime last year or your budget put a man on the moon in 1968, you are facing two choices: deliver a lot less value, or improve your capability to reduce the estimated effort or cost. Whether you use the Capability Maturity Model, Weinberg's Laws, or any other method [Muller], improving capability is required to reduce common cause variation. There are some specific techniques that will improve common-cause variation discussed in the remainder of the paper, along with the risk management techniques necessary to dealing with special-cause variation.

Processes and Projects

Improving process capability has many different aspects [Muller]. Building a critical chain plan benefits the most from those improvement techniques relating to process formalization and repeatability. Software technologists will immediately recognize this as reuse, the ability to use components again to reduce cost and improve quality.

This section gives you some basic tools that you can use to produce effective Critical Chain project plans: reusable processes, work breakdown structures, and project boundary determination.

Reusing Standard Processes: the WBS

Processes are not in themselves reusable, because they by definition happen only once. You can, however, base a task or a set of tasks on a process model, a template that identifies and standardizes tasks and workflow dependencies. The process model is what people usually refer to as a process document or a standard operating procedure (SOP). These models are reusable in the same sense as software components and provide much the same benefits: lower costs and better quality. Having written processes is one of the key steps in achieving the ability to repeat your performance in a project.

It is possible to use process models to better structure your projects for critical chain planning [Muller]. By reusing standardized tasks and workflows, you enable the better collection of cost, effort, and quality statistics. You can then use these statistics to better estimate future projects.

As well, you can use process models to build your project plan through the Work Breakdown Structure (WBS). A WBS is a list of tasks, often relating the tasks to a cost accounting or schedule management system. You need a WBS of some kind to begin the critical chain planning process; process models give you a clear and easy way to produce the WBS.

First, you need to determine what kind of process life cycle you want to use: waterfall, modified waterfall, spiral, iterative, hacking, or whatever suits your organization and production needs.

Next, you need to develop standard process models for the steps of the life cycle, specifying the processes with their inputs and outputs and the workflow dependencies between those processes.

Now you're ready to build a project plan. Each project has a set of deliverables. A deliverable is a complete, working system that the project delivers to its stakeholders. You can't do better than to plan your

project around the deliverables, since that's what the project is all about.

For each deliverable, you figure out what process models to apply to build the product. Given the process models linked together, you can generate the tasks required to produce the deliverable along with the workflow dependencies between them. You can then add in special tasks required for specific deliverables, adapting the process model to the specific product. This is the point at which you do your risk analysis, as the next section discusses, and add in tasks that mitigate risk, such as inspections, tests, and reviews.

Knowing Where to Stop Defining Tasks

Critical chain project management works best when you have a level of detail that exhibits reasonable statistical properties. Unless you're sending people to Mars, your project shouldn't have more than a few dozen activities. The WBS provides a starting point, but if you want to manage effectively, you have to have tasks that represent a reasonable variation.

You should treat meetings and other very short tasks as zero-effort milestones, excluding them from task sizing and variance calculations. If the meeting represents a real milestone, include it in your project management software, but don't include it in buffer calculations. That leaves the tasks with a relatively large amount of time.

To size tasks in a project, you need two numbers: the average accuracy of effort estimation and the average estimated effort per task for the project. You should collect statistics on your planned versus actual effort to obtain the first number. A rule-of-thumb estimate is ±10%. Some do better, some do worse; the worst situation is not knowing the real number when you've been doing it for years because you haven't kept track of how well you've done.

The total effort estimate for the project is the sum of all the effort estimates for the tasks in your WBS. You can then divide the total effort by the number of tasks in your WBS to get the average effort per task. Apply your range estimate to this number by subtracting; if the result is less than zero, your average task size is too small. You can compute the desired number of tasks by reversing the equation, dividing your target average effort by the total effort estimate. Combine smaller tasks into larger ones until you are at or below this target number of tasks.

Alternatively, you can break up your project into multiple projects. Because the project is the place where you manage the project buffer, by breaking a project into pieces you are effectively breaking up your variation tracking into pieces. The only constraint is that a project must deliver a complete deliverable, so it's not totally arbitrary where you draw the boundaries of the project:

- You want the project to deliver at least one major deliverable to the stakeholders.
- You want to have enough tasks to provide random behavior, usually at least 20-30.
- You want to have no more tasks than you can comfortably manage (7 ± 2 concurrent tasks is a good cognitive limit).
- You want to have tasks at a level of detail that does not exceed your estimation accuracy, as discussed above.

How to Generate a Critical Chain from a WBS

Now that you've defined your tasks, you can figure out the critical chain and the buffers.

1. Enter the tasks and the dependencies into your project management software, creating a schedule task with start and end dates for each WBS task. The dependencies between schedule tasks should converge on a single milestone task, the end-of-the-project milestone. Assign a fixed date to this milestone (usually the must-have date from marketing or your other stakeholders). This fixed date becomes the basis for your resource leveling and task scheduling. Set the tasks to start as late as possible (ALAP to the cognoscenti).
2. Assign resources to schedule tasks from your resource pool.
3. Level your resources backward from the project finish date. Make sure that no resource is working on more than one task at a time (including tasks in other projects, see the later section on multiple projects for more information).
4. If the resulting start date is in the past, you'll need to do some revisions to the WBS to reduce the amount of effort in the project. Bear in mind that you'll be adding a buffer, probably

somewhere between ¼ and 1/3 of your total project time, so add that in to get the real start date estimate.

5. Identify the chain of tasks and dependencies with the longest duration. This is the critical chain.
6. Create a buffer for the critical chain, inserting it as a task between the last critical chain task and the end-of-project milestone. Size the buffer using the approaches from the prior section on buffers.
7. For each task that connects to a task on the critical chain, create a feeding buffer. Size the buffer by adding up the tasks on the feeding chain. These feeding buffers tell you when a non-critical chain task is going to affect the critical chain.
8. Level the resources for the project again, as the buffer scheduling may have rearranged task schedules and created conflicts.
9. Update the project as the project progresses, delaying tasks or increasing duration to reflect reality. As you do this, track the amount of buffer consumption to manage and control the project.

If you use project management software that fully supports critical chain project management, the software will do most of the work for you. I use Scitor Corporation's Project Scheduler 8. It automatically computes the buffers and shows buffer consumption with special graphics. The resource leveling in most systems is not very good, though; you may find it better to do this manually, another good reason to keep the number of tasks to a reasonable level.

Multiple Projects versus Multiple Processes

The only fly in the ointment of this perfect project management world is the fact that almost all software shops engage in more than one project at a time. Introducing multiple projects with resource sharing makes it more difficult to level a project and to compute buffers. It also increases the pressure to use multitasking for key resources.

The difference between a project and a process is minor: a project always delivers a deliverable and is separately managed, while a process may deliver only an intermediate result and may be part of a larger effort. Nevertheless, for most purposes other than project management, projects and processes pretty much look like the same thing. A project is really a kind of process.

The last section described a decision process to use to size your project to your capability. Once you've done that, you are confronted with the need to manage resources across multiple projects.

I've found two approaches you can take to such management: cross-project leveling and drumbeat management.

Cross-project leveling is the practice of leveling all of the projects together as one big project. Most project management software lets you do this. After leveling, you then compute buffers project-by-project, then level again across all projects.

The problem with cross-project leveling is that it doesn't scale very well. If you are managing hundreds of tasks in dozens of projects that share resources, you will find it very difficult to do this kind of leveling at the global level. The obvious solution is to not share resources, or at least to minimize such sharing. Effective staffing techniques that optimize projects around available resources and provide new ones as required improve both your software production capability and your project management capability.

Real-world situations always arise, however, to make short work of your best intentions. In these circumstances, you can use the practical technique of drumbeat management to share resources. This is similar to concurrency management techniques in software development using a locking approach.

Leach intimates that the phrase comes from the old Roman practice of using a drumbeat to organize the proper workings of a galley [Leach]. Since those involved had little choice in the matter, I suspect the implications are not all that congenial, particularly with the other aspects of the directive forces. Leaving historical metaphor aside, you should focus on the resources that drive sharing across the projects. In software projects, these are usually the most valuable team members: architects, gurus, and other repositories of countless years of experience and knowledge.

When sharing between projects is more or less restricted to a few resources, you can focus leveling on those resources and ignore the others—hence, the drumbeat. The resources that share their time between projects become the driving force for scheduling, budgeting, and creation of value. Others revolve around them. You don't want your drumbeat resources multitasking under any circumstances; instead, you

make sure that projects organize their time around the availability of these resources.

As in the Roman galleys, however, it's wise to understand the limitations of your drumbeat resources when confronted with the unexpected: risk.

Managing Project Risk

Project risk is the potential for an event that has a negative impact on the project mission. Compare this definition with variation, the random values around a central tendency due to common causes. Risk has to do with identifiable or special causes of variation. Before you can manage your project variation with buffers, you must first manage your project special causes of variation with risk management.

Special Causes as Risk Events

The process expert, dealing with the ordinary manufacturing process situation, first looks to see what crazy things are going on. In manufacturing, if you stand around for awhile looking at how everybody does their job, you can quickly see a dozen things that are "obvious" about the work patterns that, unaccountably, no one actually doing the job had seen.

The slightly more experienced process expert, before presenting his conclusions to the boss, asks the workers a simple question: "Why are you doing that?" The answers are usually illuminating and humbling. The expert has missed something basic about the manufacturing process, the workers have developed massive workarounds for material inadequacies, or (sometimes) you find something that nobody's thought about.

For some reason, in software the last situation seems to emerge more often than in manufacturing. Why that's true, I don't know—nevertheless, it's a staple of software process reengineering. If you review the process, you'll easily find dozens of things that nobody's thought about. Maybe that's why software patent law is such a lucrative arena right now.

In software processes, risk plays a big role. Many aspects of the process lifecycle—requirements and design documentation, testing, reviews, and fudge factors, for example—are all ways of dealing with risk.

Translating this into an SPC worldview, software process reengineering is all about managing special causes—risk assessment and risk management. This paper makes the distinction between special and common causes because it is important for software project managers to understand the difference. That difference leads to very different styles of management. When you're managing the inherent variation in your process, you are managing common cause variation. When you're managing risk, you're managing special cause variation. SPC tells us that you cannot manage common cause variation effectively until you eliminate special causes. What that means is that, if you're not doing risk analysis, you aren't going to be able to manage your process using SPC techniques such as buffers.

For example, you can set up your project with estimates and buffers and a critical chain, and then watch in horror as that new technological solution you've bought turns into a nightmare of wasted time, bugs, and workarounds. You can hire resources and put them to work on tasks only to discover that they have forgotten every scrap of computer science they learned in school five years before because they've been working for e-commerce shops for several years without a thought for design or requirements engineering. You can spend a year developing a system only to discover that the hardware you're intending to run it on is too underpowered for the huge number of transactions that descend on the software.

In our previous example, we had a novice Java programmer untrained in using the Java user interface framework. A task that might normally take 5 days could take this novice as long as 20 days, transforming the task into a critical chain task and making it impossible to meet your deadline. This is not common variation; it's a special cause. You deal with it by training the programmer before the task needs to start or by replacing that programmer with a more experienced one, should that be an option. Most of these options disappear if you didn't identify the special cause long before the task began.

There are, of course, solutions for all of these problems. The underlying problem is that you failed to realize that there was a problem early enough to do anything about it. That's what leads to special causes of variation—not paying attention and not thinking about risk.

Assessing Risk as a Special Cause

To assess risk in software projects, you develop through brainstorming a set of risks that may affect various parts of your project. For each risk, you assess the probability of the risk event occurring and the

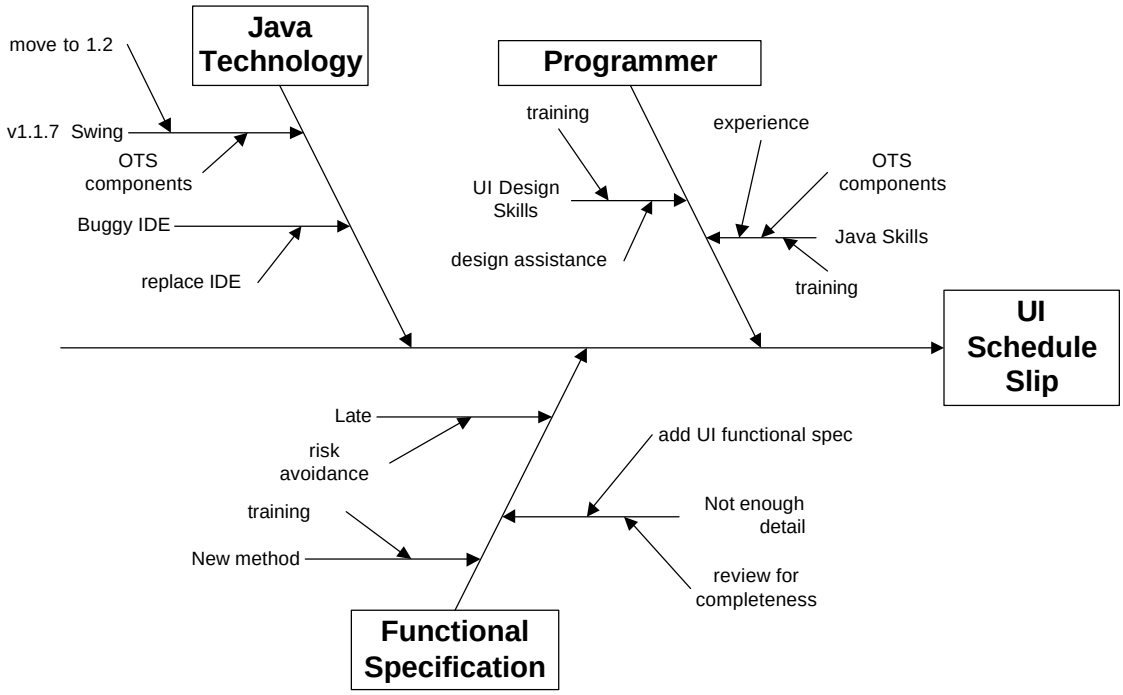
degree of impact the event will have on your project [Jones, Roetzheim].

Recasting this process as dealing with special causes in an SPC framework is easy. Once you identify the processes or systems that you are going to measure and control, you brainstorm possible special causes of variation for each. You then assess the probability and impact of the event and decide on that basis whether to eliminate the special cause or ignore it.

you are evaluating. Some techniques let you determine risk before it happens; others focus on analyzing historical data to identify risk that has happened and is preventable in future projects.

Cause-and-Effect Diagrams

The fishbone diagram gives you a tool to organize your thinking about causal chains.



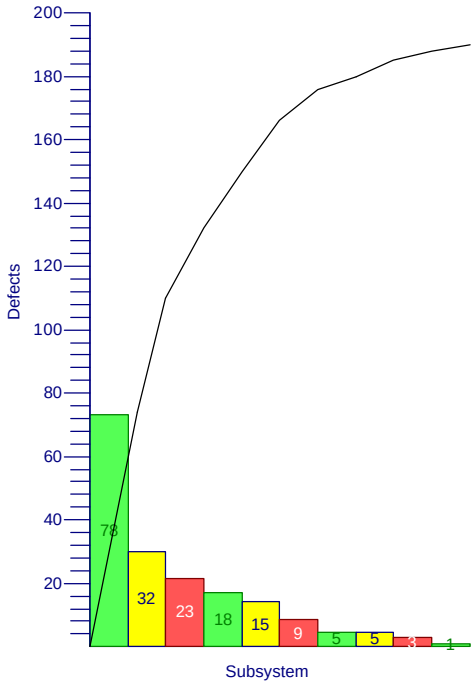
Ignoring a special cause transforms it into a common cause. By ignoring the specific risk, you account for the variation as part of your normal control efforts. These causes are not interchangeable, however. If a risk event is going to affect your project enough to cause control problems, ignoring it just makes your project fail. You should ignore only risks that have very low probabilities or minimal impact on the systems you're measuring. You should also keep track of events that happen in your projects and feed the risk information back into your risk assessments as time goes on.

Fortunately, SPC brings some tools to the party that you can use to help decide whether a risk is worth managing: Pareto diagrams, run charts, scatter plots, cause-and-effect (fishbone) diagrams, and control and cusum charts. What tools you use depend largely on the amount of data you have and the nature of the system

The diagram shows how a series of factors cause some event. For risk assessment, the event is the risk event you're assessing—the schedule slip for the user interface, for example. That's the box to the right of the line. The diagonal lines coming out of the backbone show the independent causes of the event, and the lines coming out of those causes show additional causal links or possibly ways to mitigate or avoid the cause. By the time you've finished drawing this diagram in a brainstorming session, you will have a pretty good idea of how an event could happen and how to control it.

Pareto Diagrams

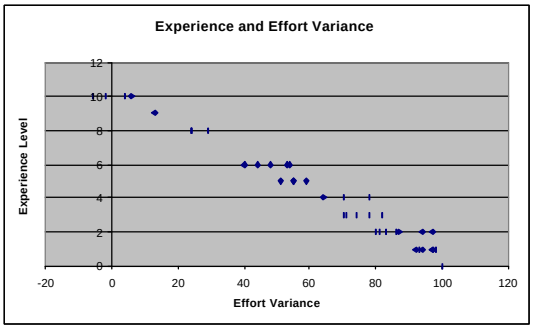
The Pareto Principle states (loosely) that 20% of the systems cause 80% of the problems. The Pareto diagram gives you a way to prioritize systems through measurement.



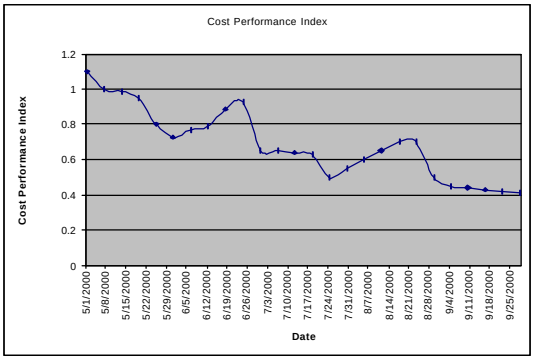
This diagram shows the classic Pareto diagram that compares different software subsystems in terms of number of defects. The first subsystem has twice as many defects as any other, and the cumulative curve shows the percentage of defects is around 40%. That translates into a clear indication of a high risk of failure due to some special cause, such as inordinate complexity, poor design, or inadequate unit and integration testing. The second system is big enough that you should probably look at it as well.

Scatter Plots and Run Charts

Sometimes the simplest techniques yield the best results. Just plotting your measures against each other (scatter plots) or over time (run charts) can give you a clear indication of special causes.



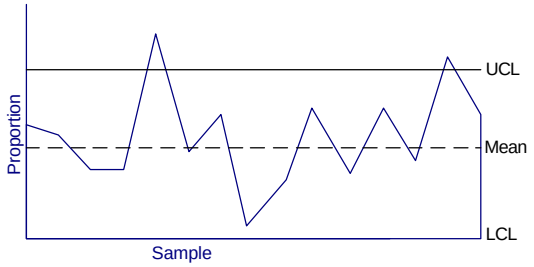
This scatter plot shows a straightforward graph of effort overrun versus technology experience for each task. As experience decreases, effort overruns grow. If there were no relationship (a random scattering of points), there would be no special cause indicated here.



This run chart shows the cost performance index (baseline cost of work performed divided by actual cost of work performed, an earned-value metric) over calendar time in a project. You can clearly see that there are events occurring in this project that are resulting in large changes in cost efficiency at specific times. Looking into those efficiency valleys should give you an idea about the special causes of the budget overruns. You can use a moving average to smooth the curve in a run chart if the too-frequent peaks and valleys obscure the changes over time.

Control Charts and Cusum Charts

The classic method of determining whether a process is under control is to use a control chart. These charts show the upper and lower control limits calculated from the normal probability distribution for the process metric you're using.



This example shows a control chart for defects as the proportion of number of defects in Java classes to total classes produced in a given week. Sampling the classes and doing a thorough code walkthrough, for example, could produce this measure.

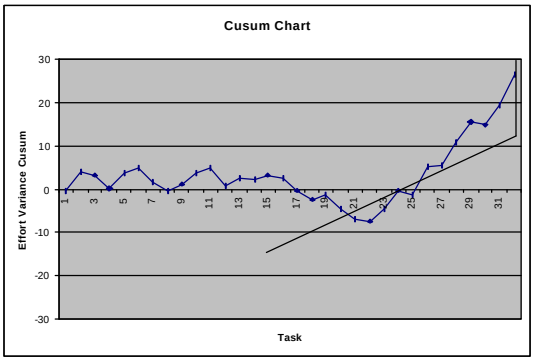
This chart is an example of a *u control chart;* there are several different kinds of control charts, varying by the kind of measure (continuous verus attribute), the kind of sampling (fixed-size versus variable-size samples), and the nature of the control variable. The u chart, for example, is a chart for proportions of varying-size collections, with the proportion relating one kind of object (a defect) to another kind of object (a Java class). The upper control limit (UCL) is statistically derived from the mean and standard deviation of the samples. The lower control limit is 0 in this case, the minimum value for the proportion. There are two signals of special causes where the sample exceeds the upper control limit.

Each type of chart has a slightly different variation on the statistical techniques for computing the control limits [Bissell, Burr, Wetherill].

There are several guidelines for identifying special causes from control charts [Bissell, Burr]:

- Any point outside a control limit indicates a potential special cause.
- A run of 7 points in succession above or below the central line shows a potential special cause.
- An unusual pattern (a regular cycle, for example) or a trend (from low to high over time, for example) shows a potential special cause.
- The proportion of points inside the middle third of the area within the control limits being less than 2/3 indicates a special cause.

Another statistical chart you can use to identify special causes of variation is the cusum chart. Cusum is short for cumulative sum; the chart shows the cumulative sum of differences from a target value over time. Cusum charts are useful for determining short-term changes in your processes.



This chart shows the cusum of effort variances over time. The V mask drawn around the points at the last task observation shows the cusum control limits. The narrow part is 5 standard errors around the target and the wide part is 10 standard errors. You look for two things in these charts: direction changes in the slope of the lines through the points and points outside the control limits. A slope change means that there was a change in the process, potentially a special cause; a point outside the control limits indicates a short-term variation due to a special cause.

Conclusion

Critical chain project management gives you a sophisticated method for combining statistical process control with project management. It lets you clearly distinguish between common and special causes of variation and provides buffering techniques for managing common variation in project costs, schedules, and quality.

Using feeding, resource, and project buffers, you can much more readily see where your project is under control and where it isn't. Using SPC techniques, you can identify and avoid or mitigate special causes of variation as risks, cleanly separating the management of the usual from that of the unusual.

References

The project management tool used to produce the pictures was **Project Scheduler 8** from Scitor Corporation (www.scitor.com). PS8 fully supports critical chain project management methods.

[Bissell] Derek Bissell. **Statistical Methods for SPC and TQM.** London: Chapman & Hall, 1994.

[Burr] Adrian Burr and Mal Owen. **Statistical Methods for Software Quality: Using Metrics for Process Improvement.** London: International Thompson Computer Press, 1996.

[Goldratt] Elihu M. Goldratt. **Critical Chain.** Great Barrington, MA: North River Press, 1997.

[Jones] Capers Jones. **Assessment and Control of Software Risks.** New York: Yourdon Press, 1994.

[Leach] Lawrence P. Leach. **Critical Chain Project Management.** Boston: Artech House, 2000.

[Muller] Robert J. Muller. **Productive Objects: An Applied Software Management Framework.** San Francisco: Morgan Kaufmann, 1998.

[Pearl] Judea Pearl, **Causality: Models, Reasoning, and Inference.** Cambridge, UK: Cambridge University Press, 2000.

[PMBOK] Project Management Institute, Standards Committee. **A Guide to the Project Management Body of Knowledge.** Project Management Institute, 1996.

[Roetzheim] William H. Roetzheim. **Structured Computer Project Managenment.** Englewood Cliffs, NJ: Prentice-Hall, 1988.

[Wetherill] G. Barrie Wetherill and Don W. Brown. **Statistical Process Control: Theory and Practice.** London: Chapman & Hall, 1991.

Robert J. Muller

Robert J. Muller is Manager of Software Development at Cytokinetics, Inc. He has managed object-oriented and database systems projects for eighteen years. A longtime consultant and teacher, he is the author of the Morgan Kaufmann books *Productive Objects and Database Design for Smarties*, and of the Oracle Press book *The Oracle Developer Starter Kit*. Dr. Muller currently specializes in applying object-oriented design methods to designing databases and project management.