PERT, CPM, and Agile Project Management.

Robert C. Martin
5 October 2003

Some time in the late 60's my father brought home a book that he thought I'd be interested in. The title was *Introduction to Operations Research* by Frederick S. Hillier and Gerald J. Lieberman, Holden-Day, 1967. I was probably 15 or 16 years old.

The book languished on my shelves for perhaps ten years. Then, as a young software professional, I pulled the book down and ,thumbing through its pages, I noticed a chapter on PERT. I had heard of PERT as a method of project management, so I started to read and learn.

Since those days in the early 80's I have seen dozens of examples of PERT charts, and tools for drawing PERT charts. They always make me cringe. Invariably these charts and tools missed the fundamental principle of PERT that made it such a successful technique: *the management of probabilities*.

PERT (Program Evaluation and Review Technique) was developed in 1958 to help measure and control the progress of the Polaris Fleet Ballistic Missile program. The technique earned considerable respect for assisting in the management of thousands of different contractors and agencies, and is credited with helping to advance the completion date of the program by two years.

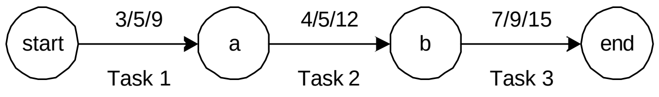
PERT: an overview.

PERT is a technique for estimating and planning a large project. One of its most powerful concepts is that project management is the management of *probabilities*. PERT makes use of simple statistical mathematics in order to come up with a probability distribution for the completion dates of the project milestones.

For example, in PERT tasks are estimated with three numbers: The best case, the nominal case, and the worst case. These three estimates are combined into an *expected duration*, and a *standard deviation*. Thus the duration of each task is presumed to be a random variable with a known distribution.

The math is very simple. Consider a task whose best/nominal/worst estimate is 3/5/9. The expected completion time (µ) is assumed to be (4*nominal + best + worst)/6, or in our case (4*5+3+9)/6 or about 5.33. The standard deviation (s) is assumed to be (worst - best)/6 or (9-3)/6 or 1.

Now consider a simple project consisting of three tasks. We represent this as a simple chart with circles and arrows. The circles denote events, and the arrows denote tasks.



If the first task begins on day zero, what day can we expect the third task to complete? The chart below shows the expected durations, and we can just add them up. So the expected duration of the project is 5.33 + 6 + 9.67 or 21 days.

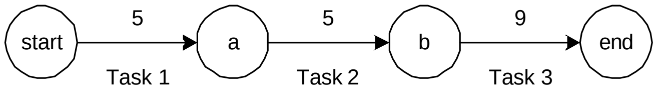
Task	µ	s
1	5.33	1
2	6	1.33
3	9.67	1.33

A more interesting question is the probability of making that date. A bit of simple reflection will convince you that if the estimates are correct then there is a 50-50 chance that the project will finish on time. There is just as much chance that it will be late as early. Clearly we cannot make a commitment based upon such poor odds, so what *can* we commit to?

The end date for the project is a random variable that has its own µ and s. We already know that µ for the project is 21 days. The s for the project can be calculated by summing s^2 for each task, and taking the square root of the result, or $(1^2 + 1.33^2 + 1.33^2)^{½} = 2.13$. Let's say we feel comfortable committing to a date that has a 90% chance of being met. A little probability math tells us that we get this certainty by adding about 1.3s about 3 days to the project. So we should commit to 24 days.

CPM

One of the more common project estimation schemes is Critical Path Method or CPM. Project managers who use CPM ignore the probabilities and use only nominal case estimates.



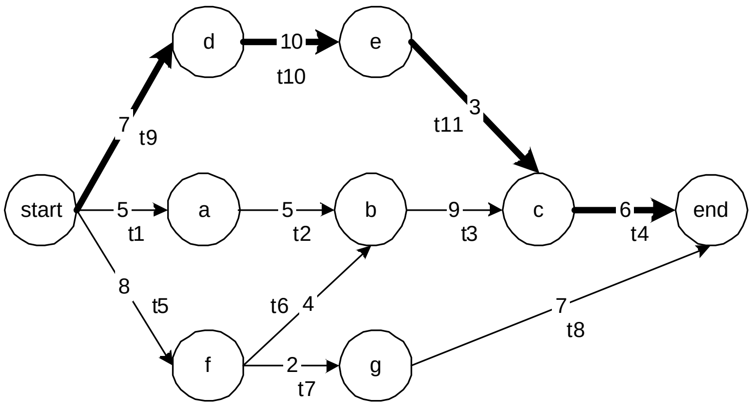
Following this approach would lead us to commit to finishing the project in 19 days.

CPM is the method supported by most project management tools. This has always puzzled me since it seems to me that knowing the probabilities is critically important to managing a project, and such tools could easily provide those calculations. Indeed, I'd much rather have a tool calculate the probabilities, than a tool that draws bubbles and arrows.

Dependencies

Both PERT and CPM represent the project with a diagram that shows the tasks and their dependencies. The simple diagrams above show three tasks that have sequential dependencies. The arrow from Task 1 to Task 2 indicates that the start of Task 2 depends on the end of Task 1. Task 2 cannot start until Task 1 finishes.

In larger projects these charts can get fairly complex.



The chart above shows a project with many tasks and dependencies. At the start of the project three tasks (t1, t9, and t5) begin concurrently. Other tasks start as these tasks end. Notice that t6 and t7 cannot begin until t5 completes. Notice also that t3 cannot start until t2 and t6 both complete.

The longest path through this network is known as the *Critical Path*, from which CPM gets its name. The critical path is shown in bold. This path through the project tells us that the *end* event is 26 days from the *start* event.

If we were using PERT, then the date of the end event would be a random variable based upon the µ and s of each of the tasks on the critical path. We would also be able to calculate the probabilities of other paths going critical.

The wrong choice.

Over the last three decades, software project managers have become enamored with the power of these charts to represent dependencies. I've often seen projects represented as huge diagrams showing all the various software tasks and their interdependencies. These are often broken down well below the subsystem level to tasks that are measured in man-days. While I agree that the ability to represent dependencies on a chart like this is *cool*; it is also fairly useless in a software project.

First of all, the number of concurrent tasks in a software project usually cannot exceed the number of people in the development team. If you have five developers, then typically no more than five programming tasks can be going in parallel. The sixth task cannot begin until one of those five completes. This represents a dependency that is usually not shown on a CPM chart. There are some automated tools that attempt to address this problem by automatically allocating tasks to available resources. However, I've never found that kind of allocation to be very useful or accurate in a software project.

A more serious problem is that most dependencies in a software project are avoidable. It may make some kind of logical sense that you have to finish writing the login servlet before you start the logout servlet; but in reality you could write and test them in any order. Analysts often assert that you must complete the data model before you can start writing queries, but in fact the queries and data model can evolve together, and the business rules can often precede both. Architects often insist that you must design and implement the infrastructure before you can begin writing the application itself; but again we've found that application and infrastructure can evolve together.

So, as cool as the dependency charts are, they almost always represent a set of arbitrary and irrelevant decisions in a software project. This may explain why these charts so frequently become useless as the project proceeds (one team I consulted for referred to it as the *laugh track*). The tasks just don't just don't get done the way the dependency chart says they should.

PERT was designed and used for *huge* projects involving thousands of contractors. Tasks were months long, truly parallel, and the dependencies were real. In this kind of environment it makes sense to create a dependency chart. It makes even more sense to use the best/nominal/worst estimation scheme and to calculate the probabilities. But for managing software projects at the man-day level the dependency charts make little sense at all.

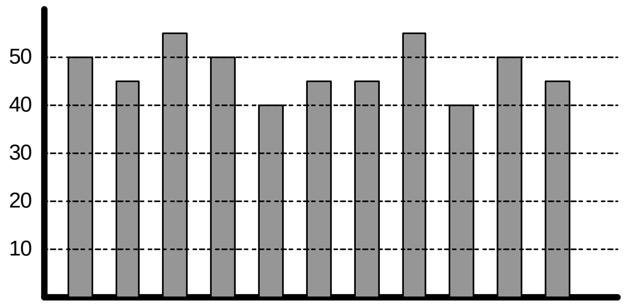
Managing Probabilities.

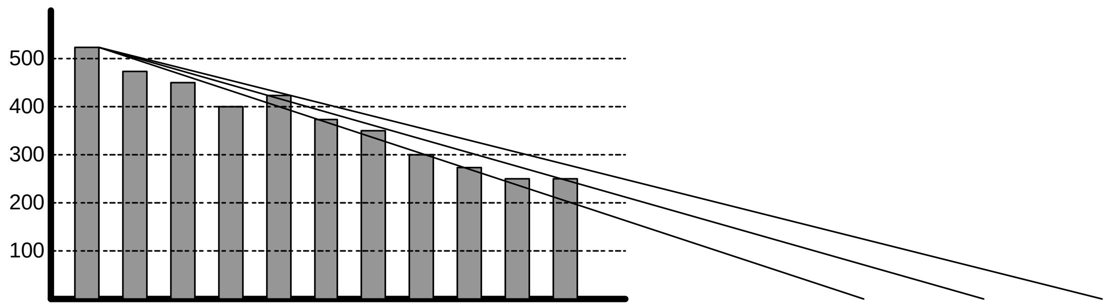
What *does* make sense in a software project is the management of probabilities. Not in the formal sense of PERT, but in the informal sense of Agile Project Management (APM).

In APM the project is broken down into tasks that are in the few man-day range. These tasks are estimated, but not in absolute units like man-days. Rather they are estimated in terms of each other. One of the tasks is arbitrarily assigned a number of points (usually three) and then all other tasks are given a point value in terms of how much harder or easier they are than that reference task.

The project is divided into iterations that are usually one or two weeks in length. At the beginning of each iteration stakeholders arrange the tasks in the order they'd like to see them implemented. The developers work down this list in the order specified. At the end of the iteration the team counts the number of points they completed. This number becomes their *velocity* for the iteration.

Two bar charts are kept on the wall. The first shows the velocity of the team for each iteration, and the second shows how many points remain in the project.





Notice that the first chart shows a random distribution of velocities around a mean of about 45 points per iteration. The second chart shows the effect of all this work on the project. Note, however that sometimes the bars on the second chart don't shrink by enough, and sometimes they even grow. This is because new features are being added to the project, and the developers are re-estimating certain tasks based on better knowledge.

The slope of the second graph is also a random distribution that predicts when the project will be done. Though it would be possible to calculate the probabilities, there isn't really much point. Given a nice straight ruler, the probabilities can be eyeballed with sufficient accuracy for the kinds of decisions that need to be made.

Given the charts above, a good software manager can make decisions about schedule, manpower, and scope and thereby manage the project to a desirable outcome.

Summary.

Project plans are afflicted by uncertainties that turn schedules into random variables. Fortunately we can measure that randomness and use it to determine the probabilities that certain tasks will be complete by certain dates. Once we know the probabilities, we can manage the project by adjusting scope, manpower, and schedule so that the probability of a desirable outcome remains high.

PERT is a scheme for managing probabilities for very large and complex projects, however it does not scale down to the level at which most software projects are planned. CPM has all these failings for software, and also ignores the randomness inherent in planning software systems.

Agile Project Management (APM) is well suited for the kinds of tasks that populate most software project plans. It also captures and characterizes the randomness and uncertainty that are inherently part of those projects. By using APM, managers have constant and accurate data with which to manage the project.