9

Network Planning Techniques

This chapter introduces project network planning techniques. The focus is on the Critical Path Method (CPM), as well as the Program Review and Evaluation Technique (PERT).

The *task* is the basic building block of a project. A task is a well defined, finite package of work that can be accomplished within a fraction of the project duration and can be assigned to an individual or a team. Sometimes tasks are also referred to as *jobs*. Tasks are typically organized and grouped in a Work Breakdown Structure (WBS). The WBS, however, is simply a tree structure and does not provide the logical dependencies between tasks.

When tasks are set in relation to each other, we obtain a *network* of tasks. This is very useful because the network can then be used for planning and monitoring the project. The two main network-based techniques in use today are the *Critical Path Method* (CPM) and the *Program Evaluation and Review Technique* (PERT).

9.1 Critical Path Method (CPM)

CPM represents a project (= a set of related tasks) as a network using graph theory. The technique:

- captures task durations
- captures the task logic (dependencies among tasks)

Tasks and Dependencies

Figure 9.1 shows the two basic relationships that tasks can have: *serial* or *parallel*.

In CPM tasks are represented as the nodes of the project network. Circles are typically used to denote tasks, and the inside of the circle contains a unique task identifier (task ID), followed by the *expected* task duration. For example A,5 identifies task 'A' which is expected to take 5 units of time. The time units are typically working days, but a finer or coarser time discretization can be used, depending on the situation.

Tasks are linked by uni-directional arrows if there is a logical dependency between tasks. In Fig. 9.1 (left) task 'B' depends on task 'A'. This means - according to this model - that task 'B'

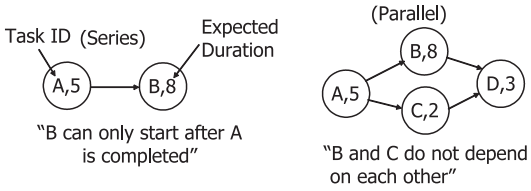

Fig. 9.1. Basic task dependencies in a project

can only start, once task 'A' has been completed. We call this a *serial* dependency, in other words: tasks 'A' and 'B' are in series.

The other situation is when two tasks are in *parallel*. This occurs when they have either the same preceding task, or they both feed into the same succeeding task, but do not directly depend on each other. In Fig. 9.1 (right) tasks 'B' and 'C' both depend on 'A' and feed into 'C', but they do not depend on each other directly. Therefore, they are said to be *parallel* tasks.

Dependency implies that a task produces an output or result which is needed as a prerequisite by a subsequent tasks. In the context of product or system development this task output is typically *information* such as market surveys, requirements, drawings, models, test results, etc... For construction and production projects task dependency typically means that physical parts and assemblies need to be completed before further construction can occur (e.g. pouring a foundation, vehicle underbody assembly, ...).

There are a number of assumptions underlying the CPM Method:

- A project consists of a collection of well defined tasks (jobs)
- A project ends when all jobs have been completed
- Jobs may be started and stopped independently of each other within a given sequence (no continuous-flow processes)
- Jobs are ordered in a 'technological sequence' that makes sense

It should be noted that one finds various ways in which to represent tasks (Fig. 9.2) in the literature. One should not be fooled by this, since the essence is the same. Traditionally, tasks are represented as *nodes* (circles or boxes) with arrows indicating the logical dependencies. Sometimes, however, tasks are represented as uni-directional arrows (arcs). In that case the nodes represent *states* of the project (see Fig. 9.2, right). This is referred to as the *Kelley-Walker* form of a project graph.

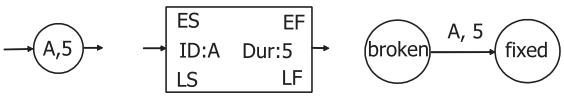

Fig. 9.2. Various ways of representing tasks in projects. left: tasks as nodes (circles), middle: tasks as nodes (rectangles) with task ID and expected duration and additional information, right: tasks as arrows (Kelley-Walker form)

Work Breakdown Structure (WBS)

We want to start assembling a project network graph. This requires a complete and well structured list of tasks. It is very important that this list be very carefully generated and that it adhere to some fundamental principles. The formal way of assembling a list of tasks is called a *Work Breakdown Structure* (WBS), which literally breaks the total work to be done in a project down into smaller pieces that can be more easily managed. Important points regarding WBS are:

- A WBS is used to create the task (job) list
- A WBS is a tree-decomposition of project tasks
- The smallest level tasks in a WBS are referred to as terminal elements
- A WBS is the key starting point for project planning
- A WBS is generally required to obtain U.S. Government contracts
- A WBS is typically part of a larger project Statement of Work (SOW)
- A WBS can be activity-oriented (tasks) or deliverable- oriented (results) or a hybrid of the two
- There are various ways to generate a WBS for a new project (e.g. sticky-notes method). If similar projects were done in the past one can usually modify an existing WBS and tailor it to the new project.
- Pritchard (1999) provides formal guidance on properly establishing a WBS

The simplest way of presenting a WBS is as an indented list (see example below).

Example of a simple WBS: Painting a Room
Source: www.wikipedia.com

1. Prepare materials
   - 1.1 Buy paint
   - 1.2 Buy brushes/rollers
   - 1.3 Buy wallpaper remover
2. 2. Prepare room
   - 2.1 Remove old wallpaper
   - 2.2 Remove detachable decorations
   - 2.3 Cover floor with old newspapers
   - 2.4 Cover electrical outlets/switches with tape
   - 2.5 Cover furniture with sheets
3. 3. Paint the room
4. 4. Clean up the room
   - 4.1 Dispose or store left over paint
   - 4.2 Clean brushes/rollers
   - 4.3 Dispose of old newspapers
   - 4.4 Remove covers

Another common representation of a WBS is as a tree structure. This is graphically appealing, and can also be used to highlight who is responsible for what part of the project when multiple partners are involved (Fig. 9.3).

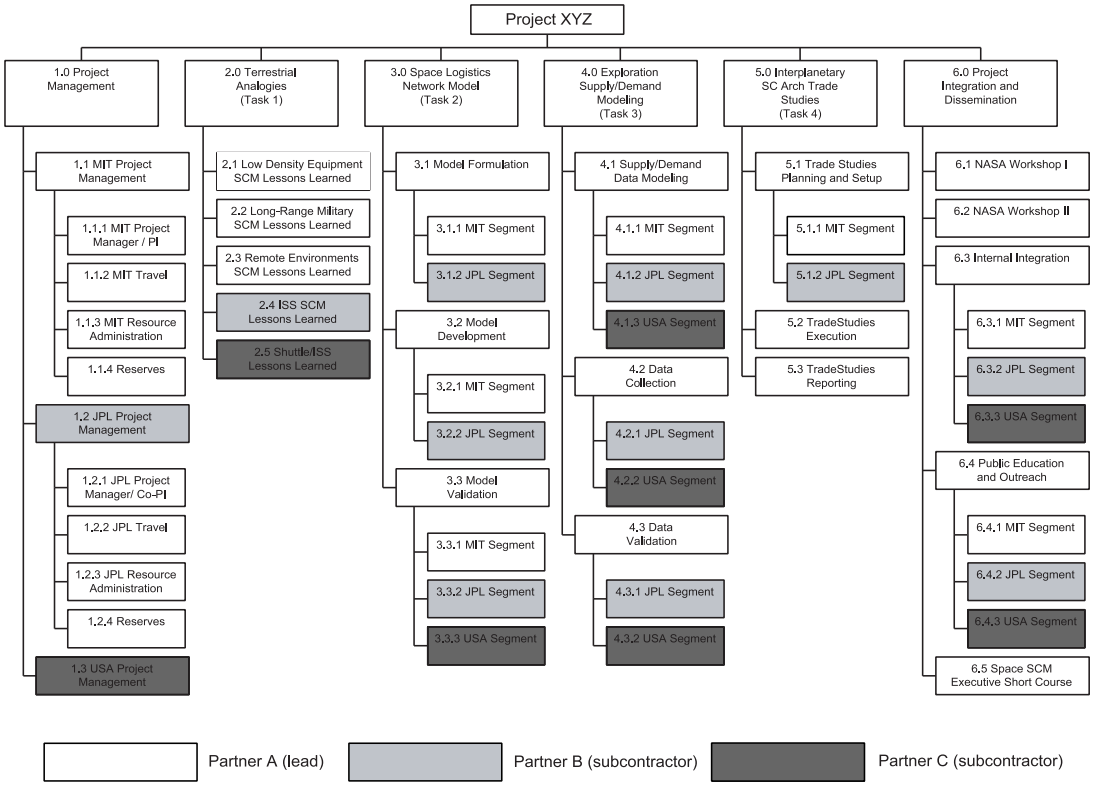

Fig. 9.3. Three-level WBS for an actual project. Different project partners are shown with different shading.

Discussion Point: Why is it difficult to come up with a good Work Breakdown Structure (WBS) for a project?

Coming up with a good WBS is not an easy task. This activity is difficult because of the following factors:

- Not all tasks are known ahead of time if we are dealing with a completely new product/system or problem
- It is difficult to create tasks that are roughly of equal size, because task durations are uncertain
- Contractors may be involved and their sub-tasks may not be known
- It is difficult to find the right level of detail (not too fine, not too coarse)

Some guidelines for establishment of a *good* WBS are as follows:

- No more than 100-200 terminal elements, if more are necessary $\Rightarrow$ use sub-projects
- Can be up to 3-4 levels deep
- Not more than 5-9 jobs at one level
  - Human cognitive bandwidth only 3 bits=$2^3 = 8 \approx 7 \pm 2$ (Miller 1956)

  - Short term memory for most people is 5-9 items
  - Poorer planning if too-fine grained, causes dilution of attention
  - The more tasks there are, the more intricate dependencies there will be to keep track of
- Jobs should be of similar size and complexity
- Manageable chunks ⇒ give a sense of progress
- Find the right level of graininess (not too fine, not too coarse)

Creating a Project Graph

Once a WBS for the project exists, there are a number of steps to generate a project graph that can be used for planning:

1. Create a task list (see Fig. 9.4) by itemizing all tasks in a table with an
   - Identifying symbol (tag, ID number)
   - Task description
   - Immediate prerequisite jobs
   - Expected task duration
2. Rearrange jobs in 'technological order'
   - No job appears in the list until all its predecessors have been listed
   - Iterations are *not allowed*. These are referred to as 'cycle errors' in CPM. We will return to this important point later.
   - Example: job 'A' precedes 'B' precedes 'C' precedes 'A' is not allowed
   - We will discuss iterations in depth in Chapter X.
3. Plot jobs on a large piece of paper or using a computer program[1]
   - Create a dummy task called 'Start' (task duration zero)
   - Each job is drawn on the graph as a circle or box, see Fig. 9.2
   - Connect all tasks without predecessor to 'Start' with a uni-directional arrow →
   - Proceed from left to right or top-down[2] and connect tasks to their immediate predecessors with uni-directional arrows →
   - Rearrange the tasks such that the number of crossing arrows is minimized
   - Create a dummy task called 'End' or 'Finish' (task duration zero)
   - Connect all jobs that don't have a successor to 'Finish'

[1] The most popular project management software is Microsoft Project, but others exist as well. See Section on Project Management tools.

[2] Unless you are from a culture where reading is done from right to left, but international standard is to make network graphs starting on the left side.

- 'Start' and 'Finish' are pseudo-jobs of length 0
- A finite number of 'arrow paths' from 'Start' to 'Finish' will be the result
- You might have to iterate a few times until the graph looks tidy
- If there are too many tasks to fit on one page, use dummy connector tasks to connect the graph over several pages
- Project management computer programs do this automatically, once a task list with precedence information has been built; nevertheless it is a good exercise to build a rough project graph by hand at least once in order to understand the procedure

Job #	Description	Immediate Predecessors	Time [min]
A	Start	-	0
B	Get materials for X	A	10
C	Get materials for Y	A	20
D	Turn X on lathe	B,C	30
E	Turn Y on lathe	B,C	20
F	Polish Y	E	40
G	Assemble X and Y	D,F	20
H	Finish	G	0

Fig. 9.4. Simple task list for fabrication and assembly of two parts X and Y

An example of a project graph, based on the simple example from Fig. 9.4 is shown in Fig. 9.5. Note that in this simple example there are a total of 4 unique paths. The total time of each path is the sum of job durations on the path. The four paths are:

- A → C → E → F → G → H ; total duration: 100
- A → C → D → G → H ; total duration: 70
- A → B → D → G → H ; total duration: 60
- A → B → E → F → G → H ; total duration: 90

In Fig. 9.5 we see that the **critical path (CP)** is the path that has the longest expected total duration: **ACEFGH = 100**. This is important information, because it is the critical path that determines the *total expected duration* of the project. Some important aspects of the critical path are:

- The CP is the 'bottleneck' of the project in terms of time
- Shortening or lengthening tasks on the critical path directly affects project finish time
- The duration of 'non-critical' tasks (those that are not on the CP) is irrelevant for the project finish time
- 'Crashing' (=compressing) all jobs is ineffective, it is more efficient to focus on the (small) subset of jobs that are on the CP

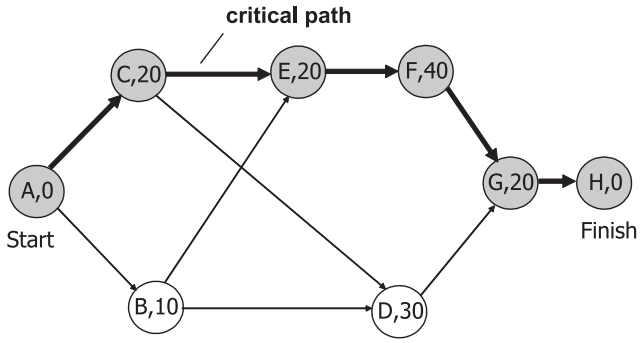

Fig. 9.5. Example project graph for fabrication and assembly of two parts X and Y. The **critical path** is shown using bolded arrows. Critical tasks are shaded gray.

- As projects get larger (10s, 100s, 1000s of tasks) the relative percentage of critical tasks tends to get smaller
- 'Crashing' (=compressing) tasks on the CP can shift the CP to a different set of tasks
- Shortening tasks ('crashing' them) is a technical and economical challenge
- Previously non-critical tasks can become critical
- Lengthening of non-critical tasks can also shift the critical path
- There can be multiple critical paths in the same project

Reflection Point: What factors affect task duration in a project? How can task durations be shortened in projects? What might be the consequences of such action?

Knowing the critical path of a project can be very useful for the following reasons:

- Allows the project manager to focus on key tasks
- Helps allocation of resources (funding, best people) to the critical tasks
- Helps to forecast the end date of the project better
- Helps focus the project team, can be used as a management tool at weekly staff meetings and for project reviews
- *Caution:* it can be dangerous to focus exclusively on the CP, because there may be near-critical tasks in a project that could easily become critical

Finding and Analyzing the Critical Path

So how does one determine the critical path?

In Fig. 9.5 this was easy because the project was small and we were able to fully enumerate all paths and find the longest one, the CP. In large projects there may be hundreds or thousands

of paths and it would be unwieldy to evaluate all of them (by hand). Therefore, a critical path algorithm was developed in order to find the CP, without having to evaluate all paths. This is also useful because - at the same time - additional information about the tasks may be developed in the context of the overall project.

First, we have to introduce some conventions regarding task *times*:

- Start time: S
- For each job: Earliest Start: ES. This is the earliest start time of a job if all its predecessors start at their ES and take the expected duration to complete
- Job duration: t
- Earliest Finish: EF = ES + t

The earliest Finish Time of the entire project (F) is the EF of the last task, the 'End' or 'Finish' dummy task. The workings of the algorithm are shown in Fig. 9.6. We will eventually switch to a more compact task representation using rectangular boxes as shown earlier in Fig. 9.2 (middle).

The **CP Algorithm** (part 1) goes through the following steps:

1. Mark the value of S to left and right of Start (0)
2. Consider any new unmarked job, *all of whose predecessors have been marked*. Mark to the left of the new job the largest number to the right of its immediate predecessors ⇒ ES
3. Add to ES the job time t and mark result to the right ⇒ EF
4. Stop when Finish has been reached

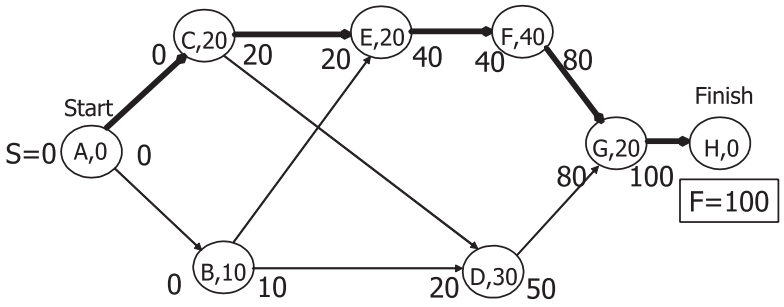

Fig. 9.6. Simple example: Critical Path algorithm to determine Early Start (ES) and Early Finish (EF) times of project tasks

This, however, does not yet tell us where the CP is. Next, we define latest start and latest finish times for each task:

- Set target finish time for project: T ≡ F
- Usually the target is a specific calendar date, e.g. October 1, 2011
- Late Finish (LF) is the latest time a job can be finished, without delaying the project beyond its target time T

- Late Start: LS = LF-t
- When is the latest date the project can be started? Answer: It is the LS time of the 'Start' task

The **CP Algorithm** (part 2) then goes through the following steps:

1. Work from the end of the project: T
2. Mark value of T to left and right of Finish
3. Consider any new unmarked job, all of whose successors have been marked - mark to the right the *smallest* LS time marked to the left of any of its immediate successors
4. Subtract from this number, LF, the job time t and mark result to the left of the job ⇒ LS
5. Continue upstream until Start has been reached, then stop

The results of this are shown in Fig. 9.7.

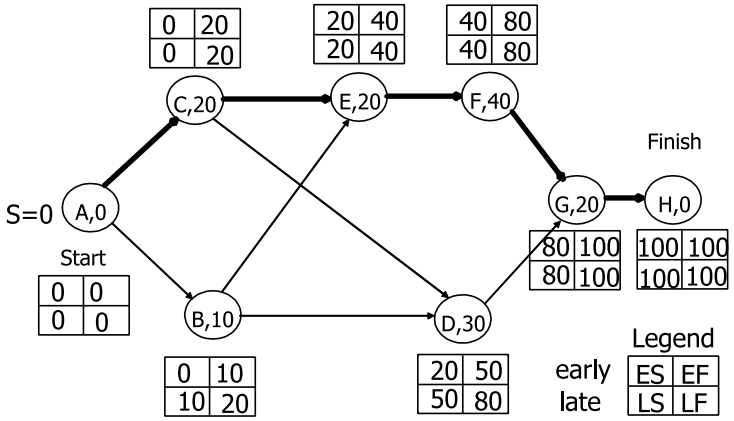

Fig. 9.7. Critical Path algorithm (2nd part) to determine Late Start (LS) and Late Finish (LF) times of project tasks. The legend shows that the upper numbers are the ES and EF, the lower numbers are the LS and LF, respectively.

Examining Fig. 9.7 we notice that for some tasks the Early Start (ES) and Late Start (LS) times are the same, i.e. ES=LS. These tasks have **no slack**. Slack is a very important concept in project management.

Slack

The main points regarding slack are:

- For some tasks it is true that ES=LS, in that case there is no slack. All such tasks are **critical**, i.e. they are members of a critical path. Specifically, this means that once the last predecessor task is completed, zero-slack tasks need to be started immediately in order not to delay the total project completion
- The Total Slack of a task is TS=LS-ES (late start minus early start)

- The *total slack* of a task is to be interpreted as the *maximum amount of time a task may be delayed beyond its early start without delaying project completion*
- Slack time is precious, it represents managerial freedom. Don't squander it unnecessarily, e.g. slack can be used for work load smoothing in a project or between multiple projects
- When T=F then all critical tasks have TS=0
- There is at least one path from Start ⇒ Finish with critical jobs only
- When T>F, then all critical jobs have TS=T-F

In Fig. 9.8 the simple project graph is shown with the total slack (TS) indicated for each task. We see that, indeed, all tasks on the CP have zero total slack, TS=0. Non-critical tasks (the lower branch) have TS>0.

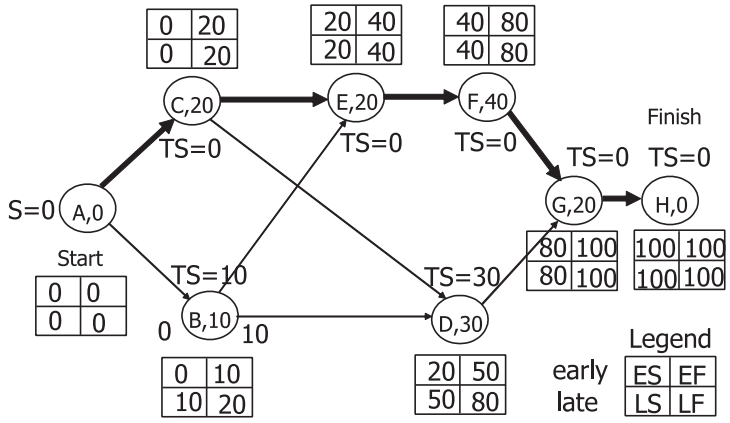

Fig. 9.8. Project CPM graph showing critical tasks with total slack TS=0

Fig. 9.9 shows graphically the different times that are associated with a single task.

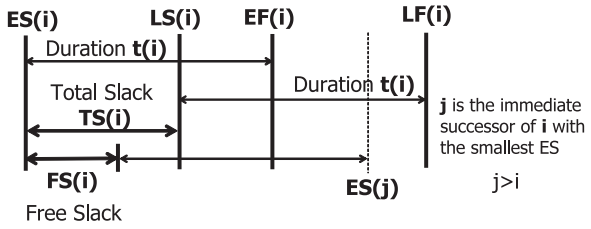

Fig. 9.9. Times associated with task (i) in the Critical Path Method

The mostleft time of task i is the early star time ES(i). Adding to that the expected (nominal) task duration gets us to EF(i), the early finish time, EF(i). If the task is critical the early and late start are the same, ES(i)=LS(i). If however, there is non-zero total slack TS(i), then LS(i)=ES(i)+TS(i). Therefore it is also true that LF(i)=LS(i)+t(i)=ES(i)+TS(i)+t(i).

This brings up a second type of slack, which is a bit more subtle. Each of the successor tasks of i will have an early start date ES. Let j be the immediate successor of task i which has the smallest (earliest) ES(j). If ES(j)<LF(i), then using up all of the total slack of task i will not affect the project end date, but it will affect the early start date of task j. In other words by consuming all of the total slack in one task one possibly affects the slack of subsequent tasks. This, of course needs to be carefully tracked, particularly if the tasks have been assigned to different individuals or teams.

To facilitate tracking this, a second type of slack, the *free slack* FS(i) is introduced. Free slack is defined as the maximum amount of time a task may be delayed without affecting the Early Start ES(j) dates of immediate successor tasks. If j is the immediate successor task of task i that has the earliest start date ES(j) and ES(j)<LF(i), then FS(i)=TS(i)-(LF(i)-ES(j)). The role of slack in projects and strategic ways of managing it will be discussed in the chapter on the critical chain method.

Remember: Free Slack (FS) is the amount a job can be delayed without delaying the Early Start (ES) of any other job. Free slack of a task is always smaller or equal than its total slack FS≤TS.

Assessment of the Critical Path Method

The CPM Method is therefore very useful for:

- Estimating the expected end date, F, of a project
- Finding the critical path, CP, and all the tasks on it
- Computing the total slack, TS, and free slack, FS, for non-critical tasks, and using this information as a managerial resource
- Identifying tasks and paths that are near-critical
- Updating the project end date, critical path and slacks as the project progresses and individual tasks are completed

Despite its usefulness, there are a number of perils in using the CPM method. Some typical errors and issues are as follows:

- Estimated job times are wrong
- Predecessor relationships may contain cycles → 'cycle error'
- List of prerequisites contains more than the immediate predecessors, e.g. A→B, B→C and A,B→C
- Overlooked some predecessor relationships
- Some predecessor relationships may be listed that are spurious and perhaps most importantly ...
- Some tasks/jobs may be missing !

In order to cope with these difficulties one usually has to start with a first CPM project plan and then gradually refine and improve it through several iterations. Important points are:

- Job Times

  - Given rough time estimates construct a CPM chart
  - Re-estimate times for the CP and those tasks with very small TS
  - Iterate until the critical path is stable
  - Focus attention on the critical subset of tasks
- Predecessor Relationships
  - Check algorithmically for cycle errors and predecessor errors
  - Cancel all, except immediate predecessor relationships
- Wrong or Missing Facts
  - Cannot be detected by computers!

Uncertain Task Durations

One of the main differences between reality and the assumptions of the CPM method concerns the expected *task durations*. CPM assumes that the task durations are fixed and known. In reality task durations will be variable.

Discussion Point: What are some of the factors that influence task durations in projects? Why are task durations variable?

All task durations in projects are variable due to a number of factor. This is true for 'thinking' tasks, such as in system design, but also for 'doing' tasks as in manufacturing.

A simple experiment in a class room setting is a powerful demonstration of this point. A class is asked to fold a paper airplane. A template is given (see MIT paper airplane, Appendix X) to them, and they are asked not to hurry but to carefully fold the airplane along the pre-printed fold lines. A particular configuration is chosen, e.g. **A1-B2-C1-D3**. All participants record their time of completion, and these are subsequently shown in a histogram, see Fig. 9.10.

The task durations that are used in CPM are typically the expected (mean) times, based on judgment or previous project experience. But as we can see from Fig. 9.10 the can be large variations, the distribution of task durations is not necessarily symmetric and there can be a long tail. There will always be circumstances and laggards that can make task durations be much longer, even factors of 2-5 times longer, than the expected value.

It would be helpful to account for variability in task durations, but retain the strengths of the CPM method.

9.2 Program Evaluation and Review Technique (PERT)

.

The PERT method is similar to CPM, except for some important differences:

- Difference how task duration is treated

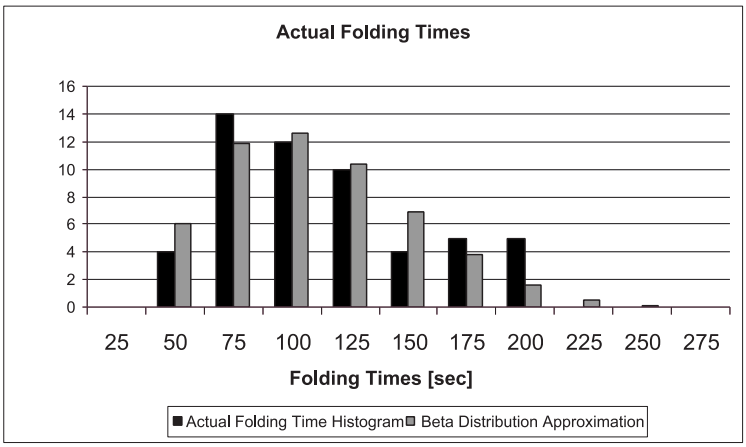

Fig. 9.10. Variability of task duration for folding a simple paper airplane, sample size: 54. The dark bars show actual recorded times. The lighter gray bars show a beta-distribution approximation with $\alpha = 2.272$ and $\beta = 5.256$. The expected (mean) task duration is TE=99.2 (sec), the task duration variance is TV=506.25 (sec), the task duration standard deviation is STDVE=$\sqrt{TV} = 22.5(sec)$.

- CPM assumes time estimates are *deterministic*
  - Obtain task duration from previous projects
  - Suitable for construction-type projects
- PERT treats durations as probabilistic
  - PERT = CPM + probabilistic task times
  - Better for R&D type projects
  - Limited previous data to estimate time durations
  - Captures schedule (and implicitly some cost)

In PERT, task time durations are treated as uncertain. For PERT, three task durations are estimated for each task: **A,M,B**:

- **A** - optimistic time estimate
  - minimum time in which the task could be completed
  - everything has to go right
- **M** - most likely task duration A
  - task duration under normal working conditions
  - most frequent task duration based on past experience
- **B** - pessimistic time estimate
  - time required under particularly bad circumstances
  - most difficult to estimate, includes unexpected delays

– should be exceeded no more than 1% of the time

A visualization of these times is shown in Fig. 9.11 for a hypothetical task. The underlying assumption is that task times are distributed according to a Beta-distribution.

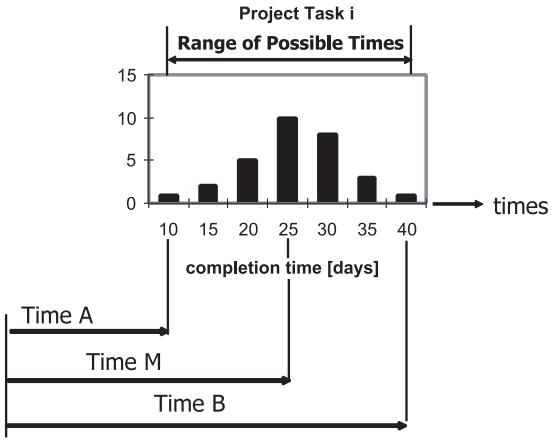

Fig. 9.11. Uncertain task durations in the PERT method: A=optimistic, M=most likely, B=pessimistic

Beta Distribution

The Beta distribution is better suited to approximating the probability distribution function (pdf) of task durations than the normal distribution because all values are contained in the interval $t \in [A, B]$. As the classes get finer (see Fig. 9.10) we arrive at a β-distribution. Indeed, the β-distribution does a good job of approximating the distribution of task times from the student paper airplane folding experiment.

The mathematical form of the β-distribution pdf is:

$$P(x) = \frac{(1-x)^{\beta-1} x^{\alpha-1}}{B(\alpha,\beta)} = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} (1-x)^{\beta-1} x^{\alpha-1} \tag{9.1}$$

whereby

$$x \in [0,1] \tag{9.2}$$

The Beta function is known as:

$$B(p,q) = \frac{\Gamma(p)\Gamma(q)}{\Gamma(p+q)} = \frac{(p-1)!\,(q-1)!}{(p+q-1)!} \tag{9.3}$$

Figure 9.12 shows the shape of the beta distribution for a number of combinations of the parameters α and β. It is through these parameters that one can approximate the shape of the task

duration distribution, by optimizing the two parameters to best match data from past projects and experience.

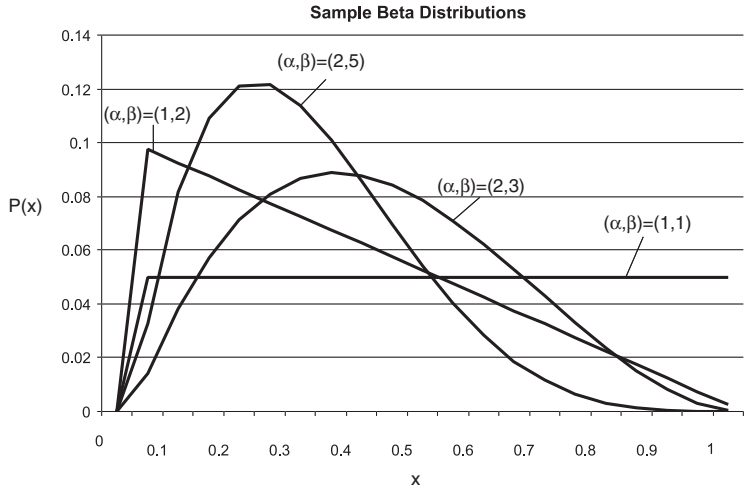

Fig. 9.12. Sample beta distributions for different parameters α and β

However, such data may not always be available. Using the β-approximation assumption we can estimate the mean expected time (TE) and the time variance (TV) of a task based only on an estimate of A, M and B:

- Mean expected task duration (TE): $TE = \frac{A+4M+B}{6}$
- Time variance (TV): $TV = \sigma_t^2 = \left(\frac{B-A}{6}\right)^2$
- Early Finish (EF) and Late Finish (LF) are computed as for CPM with TE
- Set T=F for the end of the project
- Example: A=3 weeks, M=5 weeks, B=7 weeks $\Rightarrow$ then TE=5 weeks

Now, however PERT explicitly acknowledges that both the EF and LF times of a task are probabilistic (Fig. 9.13).

The expected slack is then

$$SL(i) = E[LF(i)] - E[EF(i)] \tag{9.4}$$

The variance of the slack times of task i is then

$$\sigma^2\left[SL(i)\right] = \sigma^2\left[EF(i)\right] + \sigma^2\left[LF(i)\right] \tag{9.5}$$

whereby the variance of the Early start ES(i) of task i is the sum of all variances of the longest path from start (task 0) to task i.

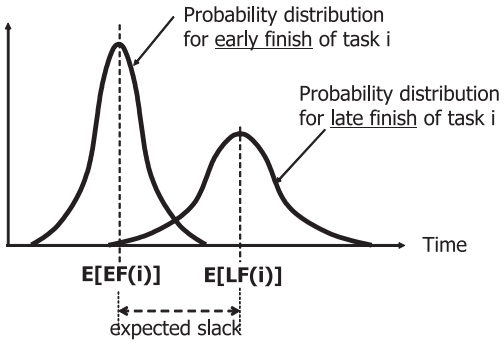

Fig. 9.13. Probabilistic slack in PERT

$$\texttt{Start Times:} \quad \sigma^2\left[ES(i)\right] = \sum_{j=0}^{i} \sigma_{t_j}^2 \tag{9.6}$$

The variance of the early finish of task i is the sum of all variances of the longest path from Finish (task n) upstream to task i.

$$\texttt{Finish Times:} \quad \sigma^2\left[EF(i)\right] = \sum_{j=n}^{i} \sigma_{t_j}^2 \tag{9.7}$$

Thus, the variance of the slack of a task, see Eq. 9.5 assumes that the slack is Gaussian (normally) distributed. This is a reasonable assumption, because, even though individual task durations may be β-distributed, the variance of the sum of a large number of tasks may again be nearly normally distributed due to the Central Limit Theorem.

Probability of Meeting a Target Finish Date

Ultimately, what PERT can deliver is a probability of meeting a certain project Target Finish date (T). This is the same as asking the question: What is the probability of no slack on the final task? The final task is the dummy task 'Finish' (task n) we introduced earlier.

- The target date is not met when SL(n)<0, i.e. negative slack occurs on the final task
- To estimate the probability of this occuring, we convert slack to a normalized random variable z:

$$z = \frac{EF - LF}{\sqrt{\sigma^2(EF) + \sigma^2(LF)}} = \frac{-SL}{\sigma\left(SL\right)}$$

- For each value of z one can look up the probability that SL=0 from a table of the normal (Gaussian) distribution function
- For tasks on the critical path, the probability that SL=0 is always 50

Many projects have fixed target completion dates, T:

- Interplanetary mission launch windows 3-4 days
- Contractual delivery dates involving financial incentives or penalties
- Timed product releases (e.g. holiday season)
- Finish construction projects before winter starts

So, we need to analyze the expected Finish time, F, of a project, relative to its target finish time, T. Fig. 9.14 shows the expected early finish time of a project $E\,[EF(n)] = F$. This can be assumed to be normally distributed for the aforementioned reason. The figure shows the probability that the project will be finish by time T as the area (integral) under this distribution from the left up to T. If T is the same as $E\,[EF(n)] = F$, then the probability of making this target completion date is 50%. If T is to the left it is less than 50 %, if T is to the right (i.e. later) the probability is greater than 50 %.

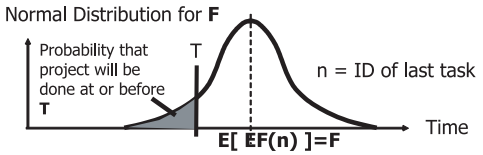

Fig. 9.14. Probability of finishing a project by a target date T in PERT

Next, we compute the random normal variable z as

$$z = \frac{T - E[F]}{\sigma(F)} \tag{9.8}$$

We can look up the probability in a standard normal probability table (see Appendix B)[3].

Example: A company wants to have a prototype at a car show on T=May 3rd. The expected Early Finish date F for the project from the PERT plan is: May 12 with a standard deviation σ=20. Then z= (5/2-5/12)/20=-8/20=-0.4. Note that there are 8 working days between 5/2 and 5/12. For z=-0.4 we obtain a probability of 35% that we can meet the target date.

The example above shows what additional information PERT gives above and beyond CPM. This is especially valuable if we consider 'crashing' (=accelerating) some tasks in the project:

- If we theoretically were building 100 prototypes, $\tilde{3}5$ of them would make it on time
- This is an important decision basis for management
- How could we speed up the project?
- What is the cost of speedup?
- Is there a net savings resulting from reduction in overall project time?

[3] Also available at: http://www.math2.org/math/stat/distributions/z-dist.htm

Project Time and Cost

Clearly, there is a relationship between the cost of a project and its duration, as well as the duration of individual tasks. If tasks are carried out very slowly ('stretched') they will be expensive due to the cost of the 'standing army'. If the task is carried out very fast we might have to pay penalties for overtime, express deliveries and so forth. There is generally an optimal task and project duration that will minimize overall project costs:

- We can compute project costs if the cost of each job is included in the task data, e.g. see Fig. 9.4.
- We can potentially shorten jobs by adding personnel (this doesn't always work, see Brook's law in Chapter X).
- Speedup carries a price tag. Typically we distinguish between: 'normal time' and 'crash time'
- We can assign some critical jobs to their 'crash time'
- Direct costs will increase as we 'crash' critical tasks
- Indirect (fixed, overhead) costs will decrease as the overall project duration decreases. This is due to the 'standing army phenomenon'
- We can attempt to minimize total project costs by minimizing the sum of fixed and direct costs

A typical cost pattern as a function of project time is shown in Fig. 9.15. There is an optimal project duration that minimizes total cost.

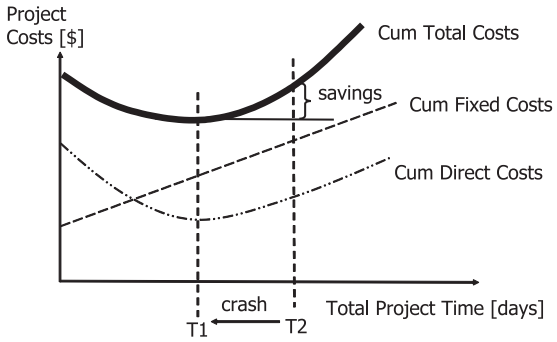

Fig. 9.15. Shortening total project length can reduce total cost

9.3 Summary

Both CPM and PERT are in widespread use today. The methods are clearly effective and important for project planning. However, they are only abstractions of reality and do not reflect all the real factors that drive project duration, cost and ultimately success. Some of the advantages and disadvantages of CPM are discussed in Table 9.1.

Summary comments:

Table 9.1. Advantages and Disadvantages of CPM

Advantages	**Disadvantages**
Focuses attention on a subset of critical tasks	Doesn't capture task iterations, in fact
Helps determine the effect of shortening/lengthening tasks	Prohibits iterations = 'cycle error'
Can evaluate cost of a 'crash' program	Treats task durations as deterministic

- CPM is useful, despite criticism, to identify the critical path
- Focuses attention on a subset of the project tasks
- Slack (TS and FS) is precious. We can apply this flexibility to smooth resource/schedules
- PERT treats task times as probabilistic
  - Individual task durations are β-distributed
  - Sums of multiple tasks are normal z-distributed
- Selective 'crashing' of critical tasks can reduce total project cost
- CPM and PERT do not allow task iterations