13 Managing projects

This chapter covers

- Work breakdown structure (WBS), or hierarchical composition of project tasks
- The program evaluation review technique (PERT), or project scheduling using time estimates
- The critical path method (CPM), a technique to identify the longest sequence of dependent project tasks
- Calculating the probability of on-time completion
- Project crashing, or accelerating project timelines

Project management is a critical yet often undervalued skill set in the business world. Although single-level contributors are typically selected for projects based on their specific hard skills, project managers are frequently chosen not for their proven management capabilities but rather for the absence of these technical skills. This paradoxical approach undermines the effectiveness of project management, as it overlooks the specialized skills truly necessary for leading projects successfully. Effective project management requires the use of several quantitative techniques, from estimating activity times to finding the project's critical path, calculating the probability of on-time completion, and implementing strategies to accelerate project timelines. *Without a solid understanding and application of statistics and quantitative methods, projects cannot be managed intelligently or effectively.*

Projects are typically defined by three core components: scope, schedule, and budget. It's not uncommon to emphasize any two of these components in planning and especially during execution. Nevertheless, they are intricately interconnected, and any imbalance among them can jeopardize the success of a project. Although our focus here is on building and managing a well-constructed project plan buttressed by the use of quantitative techniques, be aware that cost overruns and even compromises in scope are often direct results of project delays. Therefore, a well-managed schedule is not only necessary to ensure on-time completion but also pivotal in maintaining the project's scope and budget.

Projects often fail from the outset due to inadequate planning, which sets the stage for further derailment during execution due to poor management. Effective project management is not just about strong leadership and clear communication; it also demands hard skills. Poor planning frequently leads to unrealistic timelines, unaccounted-for activities, and unforeseen challenges, while also setting incorrect expectations with stakeholders, often resulting in a cycle of over-promising and under-delivering. These initial shortcomings are then exacerbated by ineffective project management, culminating in missed deadlines and eventually cost overruns and compromised project goals. Successful project management demands a comprehensive skill set that encompasses not only leadership and communication but also a robust understanding of quantitative methods to accurately forecast timelines, allocate resources, and mitigate risks.

The use of advanced project management techniques and other quantitative methods is essential for developing a project plan with reasonable time estimates, accounting for uncertainty, distinguishing between critical and noncritical tasks, applying statistical measures to calculate the probability of on-time completion, and implementing strategies to compress the project schedule without reducing scope. By mastering these techniques, you will acquire the essential skills and other tools needed to manage projects large and small with efficiency and efficacy.

One of the prioritized projects from chapter 8 will serve as our test bed: we will demonstrate how to apply best-in-class project management techniques and other quantitative methods, manually and programmatically, to plan and manage the project toward on-time completion. We will begin by creating a work breakdown structure, an essential step in organizing and planning a project. It's important to understand that a work breakdown structure (WBS) is not the same as a project plan; rather, it serves as a detailed framework that forms the basis for developing the project plan.

13.1 Creating a work breakdown structure

Back in chapter 8, we applied a linear programming technique called constrained optimization to prioritize a subset of capital projects from a backlog based on their value-to-effort ratio within given constraints. One of the projects that made the cut was building an automated reporting tool.

As the first step toward planning and managing the implementation of this project, we will build out a WBS. The WBS is a tabular or visual decomposition of a project into its component activities that records all the dependencies among these activities. The WBS is the first step toward developing a comprehensive project plan, providing a detailed blueprint that guides the entire project management process. By organizing tasks in a clear and manageable structure, the WBS lays the groundwork for subsequent project planning and management while ensuring that all project activities are accounted for from the outset.

The WBS is therefore not the project plan, but rather the foundation for developing the plan with further details. It lays out activities and the dependencies, if any, required to start those activities. A detailed project plan will include not only the data from the WBS but also estimated activity times and other attributes that, when analyzed, allow us to identify those tasks that collectively make up the project's critical path. All of this will be added after we first demonstrate how to generate a WBS.

An *activity*, commonly referred to as a *task* or *subtask*, is a distinct component of work that must be finished in full for the project to be 100% complete. Throughout the project planning process, metrics and other attributes are incrementally assigned to each activity, including but not limited to dependencies.

A *dependency*, meanwhile, refers to a relationship between activities where, typically, at least one activity cannot begin until one or more preceding activities have been completed. Dependencies ensure that tasks are executed in a logical sequence, preventing scheduling conflicts and maintaining the project's workflow. Properly identifying and managing these dependencies is essential for accurate project planning and timely completion.

Here is a breakdown of the project activities along with their dependencies. If you prefer to skip the details, you can proceed directly to table 13.1:

- A: Define Project Objectives and Scope
  - Description: Establish the overall goals and boundaries of the project.
  - Dependencies: None
- B: Assemble Project Team
  - Description: Form the team that will work on the project.
  - Dependencies: A
- C: Gather Requirements from Stakeholders
  - Description: Collect and document the requirements from all stakeholders.
  - Dependencies: A, B
- D: Data Collection and Integration
  - Description: Collect and integrate data from various sources.
  - Dependencies: C
- E: Data Cleaning and Preprocessing
  - Description: Clean and preprocess the collected data for analysis.
  - Dependencies: D

- F: Design Report Templates
  - Description: Create templates for the reports to be generated.
  - Dependencies: C
- G: Develop Data Processing Pipelines
  - Description: Develop pipelines to process the data.
  - Dependencies: E
- H: Implement Report Generation Logic
  - Description: Implement the logic needed to generate reports from the data.
  - Dependencies: F
- I: Develop User Interface for Report Access
  - Description: Create the user interface for accessing the reports.
  - Dependencies: F
- J: Integrate Data Pipelines with Report Generation
  - Description: Integrate the data processing pipelines with the report generation system.
  - Dependencies: G, H
- K: Conduct User Testing and Feedback
  - Description: Test the system with users and gather feedback.
  - Dependencies: I, J
- L: Finalize and Deploy Automated Reporting Tool
  - Description: Finalize the tool based on feedback and deploy it.
  - Dependencies: K

Although the layout and sequence of these tasks may suggest a waterfall methodology, where tasks are completed in a linear, sequential manner, it's important to note that these tasks can be adapted to fit an Agile framework as well. In an Agile approach, tasks can be broken down into smaller, iterative cycles, known as *sprints*, allowing for continuous feedback and adjustments throughout the project. Agile methodology emphasizes flexibility, collaboration, and customer feedback, which can be incorporated into the development of the automated reporting tool. Regardless of the chosen methodology, a WBS is essential and must clearly outline all dependencies to ensure smooth project execution. These activities and their dependencies are summarized in table 13.1.

Table 13.1 WBS for completing the automated reporting tool project. It includes an outline of the key activities required, along with brief descriptions and their respective dependencies. Each activity is identified by a unique code and details the necessary steps and preceding tasks needed for successful project completion.

Activity	Description	Dependencies
A	Define Project Objectives and Scope	None
B	Assemble Project Team	*A*
C	Gather Requirements from Stakeholders	*A, B*

Table 13.1 WBS for completing the automated reporting tool project. It includes an outline of the key activities required, along with brief descriptions and their respective dependencies. Each activity is identified by a unique code and details the necessary steps and preceding tasks needed for successful project completion. ***(continued)***

Activity	Description	Dependencies
D	Data Collection and Integration	*C*
E	Data Cleaning and Preprocessing	*D*
F	Design Report Templates	*C*
G	Develop Data Processing Pipelines	*E*
H	Implement Report Gathering Logic	*F*
I	Develop User Interface for Report Access	*F*
J	Integrate Data Pipelines with Report Generation	*G, H*
K	Conduct User Testing and Feedback	*I, J*
L	Finalize and Deploy Automated Reporting Tool	*K*

The WBS provides a comprehensive outline of the tasks required to deliver an automated reporting tool. Each activity is identified by a unique code, accompanied by a brief description and its respective dependencies. This structured approach is essential for effective project management, ensuring that all tasks are accounted for and executed in the correct sequence.

The WBS is constructed by listing all the necessary activities and specifying the dependencies for each task. For instance, the project begins with defining the project objectives and scope (Activity *A*), which sets the foundation for assembling the project team (Activity *B*). Following this, gathering requirements from stakeholders (Activity *C*) depends on the completion of both defining the project scope and assembling the team. This methodical approach ensures that all prerequisite tasks are completed before initiating subsequent activities, thereby minimizing potential delays and other issues.

We can draw several key insights from the table. The sequential flow of the project is evident, starting with the initial planning and team assembly, moving through data collection and processing, and culminating in user testing and deployment. The Dependencies column highlights the interconnections between tasks, illustrating the importance of coordinating various project elements. For example, tasks like gathering requirements and designing report templates are foundational and affect several subsequent activities.

Additionally, the WBS emphasizes the critical phases of integration and testing. Tasks such as integrating data pipelines with report generation (Activity *J*) and implementing report generation logic (Activity *H*) are pivotal, relying on earlier stages of data processing and report design. The final stages of the project involve conducting user testing and gathering feedback (Activity *K*), followed by the finalization and deployment of the automated reporting tool (Activity *L*). This ensures that the tool meets user requirements and is ready for operational use.

In summary, the WBS serves as a detailed roadmap for the project, ensuring that each task is executed in the correct order and dependencies are managed effectively. This approach not only aids in planning and scheduling but also helps in monitoring the project's progress, ultimately contributing to the successful completion of the project.

But although the WBS provides a comprehensive roadmap of the tasks and their dependencies, it leaves an essential question unanswered: how long will the project take? This is a critical aspect of project planning that the WBS alone does not address. To gain a clearer understanding of the project timeline, the next step is to estimate the completion time for each task. Adding these time estimates will allow us to create a more detailed project plan, predict the overall project duration, and identify the critical path tasks that dictate the project's completion time.

13.2 Estimating activity times with PERT

Analogous estimating is a technique used in project management to estimate the duration of project tasks based on the duration of similar tasks from previous projects. This method uses historical data and expert judgment to provide a quick and relatively accurate estimate of activity times, particularly when detailed information is not available. Analogous estimating works best when similar tasks have been previously performed on similar projects and the actual activity times have been recorded. However, in the absence of such data, we turn to a quantitative technique like the program evaluation and review technique (PERT) to estimate activity times more precisely.

PERT is a statistical tool used in project management to estimate the duration of project activities. It involves calculating a weighted average of three different activity time estimates to derive an expected activity time. These estimates are as follows:

- *Optimistic time* (a)—The shortest time in which the task can be completed, assuming everything proceeds better than expected
- *Pessimistic time* (b)—The longest time the task might take, assuming everything goes wrong
- *Most likely time* (m)—The best estimate of the time required to complete the task, assuming everything proceeds as normal

The accuracy of PERT estimates depends heavily on the quality of the inputs. It is crucial that the assigned resources or subject matter experts provide their own estimates for completing their tasks. These estimates should be thoughtful and based on their expertise and experience. Ensuring that the input data is reliable will significantly enhance the reliability of the PERT outputs, leading to more effective project planning and management.

These estimates are inserted into the following formula to get the expected activity time(t):

$$t = \frac{a + 4m + b}{6}$$

So, if the first task in the automated reporting tool project is optimistically estimated to take 2 weeks, pessimistically estimated to take up to 4 weeks, and most likely to be completed in 3 weeks, the expected time for that activity is

$$t_{(A)} = \frac{2 + (4)(3) + 4}{6}$$

or

$$t_{(A)} = 3$$

Only the expected activity time should be assigned to each task; the optimistic, pessimistic, and most likely estimates are merely inputs to calculate t. Thus, an expected activity time of 3 weeks should be assigned to Activity A.

PERT assumes that estimates in activity times follow a beta distribution. The *beta distribution* is a continuous probability distribution commonly used to model the uncertainty in various fields, including project management. It is characterized by two shape parameters that determine the skewness and spread of the distribution, allowing it to accurately represent the variability in activity durations. This distribution is particularly useful in PERT because it accommodates the optimistic, most likely, and pessimistic time estimates, providing a more comprehensive and realistic estimation framework. Like all probability distributions, the beta probability distribution can best be understood when visualized: see figure 13.1.

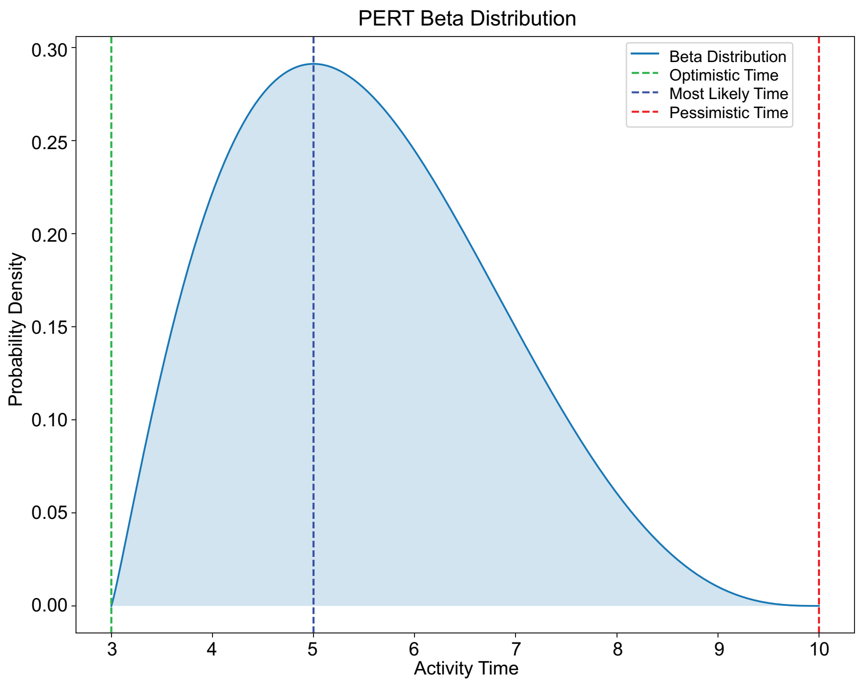


Figure 13.1 An illustration of the beta probability distribution assumed by PERT that displays the optimistic, pessimistic, and most likely activity times. The area under the curve represents the probability of different completion times, with the peak indicating the most likely duration.

Understanding the plot of the beta distribution helps illustrate how it captures the range and likelihood of different activity durations. It provides a clear line of sight into the probability of completing any task within the estimated times. This insight is crucial for project managers to make informed decisions, manage risks effectively, and ensure a realistic approach to scheduling and resource allocation.

The individual time estimates for every activity in the automated reporting tool WBS are provided in table 13.2. Several key points should be emphasized. First, the sum of the expected activity times does not represent the number of weeks it should take to complete the project. This is because some activities can be performed simultaneously by different resources, meaning the overall project timeline will likely be shorter than the sum of the individual expected times. Additionally, it is important to note that the most likely times and the expected times may not always align perfectly due to inherent variability in the estimates. This variance is significant as it influences the calculation of the probability of completing the project on time. Although planning primarily revolves around the expected times, understanding the variance is essential for accurately forecasting project completion and managing potential delays.

Table 13.2 Time estimates (in weeks) for automated reporting tool activities. Based on these estimates, the expected time to complete every activity sums to 39 weeks. However, because a subset of these activities can be performed in parallel, this does not represent the estimated project duration.

Activity	Optimistic, *a*	Most likely, *m*	Pessimistic, *b*	Expected time, *t*
A	2	3	4	3
B	1	2	3	2
C	2	3	4	3
D	3	5	7	5
E	2	3	4	3
F	2	4	6	4
G	2	4	6	4
H	2	3	4	3
I	2	3	4	3
J	2	4	6	4
K	2	3	4	3
L	1	2	3	2
				39

Despite these complexities, we now have sufficient data to identify the critical path. The *critical path* is crucial because it represents the longest sequence of tasks that must be completed on time for the project to be finished by the expected end date. Any delay in the tasks on the critical path will directly affect the overall project timeline, making it essential to monitor these activities closely. Identifying the critical path allows project managers to allocate resources efficiently and prioritize tasks to ensure that the project remains on track.

13.3 Finding the critical path

Finding the critical path is a vital step in project management, often achieved through the critical path method (CPM). The critical path represents the longest sequence of dependent tasks that determines the shortest possible duration to complete the project. Any delay in tasks on this path will directly delay the overall project timeline. Conversely, tasks not on the critical path may have some flexibility and can be delayed without affecting the project's end date. If there's a need to reduce the overall project timeline, it must be addressed by optimizing one or more tasks on the critical path. Understanding and managing the critical path ensures that project managers can effectively prioritize resources and mitigate potential delays.

To find the critical path, we first need to determine several key metrics based on the dependencies from the WBS and the expected activity times we just derived. These metrics are crucial for calculating the critical path and ensuring that the project is completed on time:

- *Earliest start time (ES)*—The earliest point in time when a task can begin, considering the completion of all preceding tasks
- *Earliest finish time (EF)*—The earliest point in time when a task can be completed, calculated by adding the expected activity time to the earliest start time
- *Latest start time (LS)*—The latest point in time a task can begin without delaying the project's overall completion
- *Latest finish time (LF)*—The latest point in time a task can be completed without delaying the project's overall completion

With tasks, dependencies, and expected activity times now clearly defined, we have established a comprehensive project plan. To identify the critical path, we will perform forward and backward passes through this plan, applying the dependencies and expected activity times to determine the earliest and latest start and finish times for each task. This process will ultimately reveal the critical path, highlighting the sequence of tasks that directly affect the project's overall timeline.

13.3.1 Earliest times

The earliest times in project scheduling are crucial for determining when activities can start and finish to ensure the project stays on track. The earliest finish time for any activity is calculated by adding the expected activity time to the earliest start time. For most activities, the earliest start time is the same as the earliest finish time of any immediate predecessors. However, if an activity depends on the completion of more than one prior activity, its earliest start time is determined by the latest of the earliest finish times of those preceding activities. This ensures that all necessary prerequisite activities are completed before a dependent activity begins.

The forward pass in project scheduling is the process of determining the earliest possible start and finish times for each activity within a project. This approach ensures that tasks are scheduled as soon as possible, which is essential for identifying the critical path and overall project timeline. In the automated reporting tool project plan, the forward pass begins at time zero. Activity *A*, which has no immediate predecessors, therefore has an earliest start time of 0 weeks. Given its expected duration of 3 weeks, the earliest finish time for Activity *A* is the sum of the earliest start time and its activity duration, resulting in 3 weeks.

Activity *B* can begin as soon as Activity *A* is completed. Therefore, the earliest start time for Activity *B* is the same as the earliest finish time of Activity *A*, which is 3 weeks. Considering that Activity *B* has an expected activity time of 2 weeks, its earliest finish time is calculated to be 5 weeks from the project start date. This process continues for all subsequent activities, ensuring that each task begins as soon as its predecessors are completed, thereby optimizing the project schedule. Through the forward pass, we can effectively map out the timeline of the project, ensuring that each activity is allocated the earliest possible start and finish times, which is critical for timely project completion.

The network diagram in figure 13.2 provides a visual representation of the automated reporting tool project plan. Each activity is labeled with its unique identifier. Lines and arrows between activities indicate dependencies, signifying that one or more activities must be completed before subsequent activities can begin. The sequence of digits beneath each activity represents the earliest start time, expected activity time, and earliest finish time, respectively.

This analysis indicates a project duration of 29 weeks, which is determined by the longest path in the network. However, to accurately identify the critical path and ensure timely project completion, a backward pass through the network is necessary to determine the latest start and finish times for each activity. Only when we have both the earliest and latest times can we then identify the project's critical path.

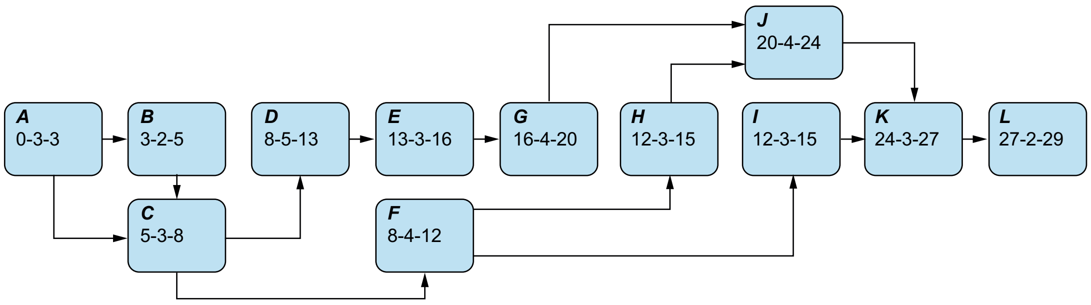


Figure 13.2 A visual representation of the automated reporting tool project plan, where each activity is represented by its unique identifier. Lines and arrows between activities represent dependencies. The sequence of digits immediately beneath each activity represents, in order, the activity's earliest start time, expected duration, and earliest finish time. The earliest times are derived by using a forward pass through the network. Based on this analysis, the project timeline is 29 weeks, which is the length of the longest path in the network.

13.3.2 Latest times

Determining the latest times for project activities is essential for understanding the flexibility within the project schedule. The latest finish time for any activity is the latest time it can be completed without delaying the overall project. This is calculated through a backward pass, starting from the project's finish and moving backward through the network. Similarly, the latest start time for an activity is the latest time it can begin without delaying the project, calculated by subtracting the activity duration from the latest finish time.

With respect to our project plan for the automated reporting tool, the process begins by setting the latest finish time for the final activity, which in this case is 29 weeks. For the final activity, Activity *L*, the latest finish time is 29 weeks, and the latest start time is calculated by subtracting the expected activity time (2 weeks) from the latest finish time, resulting in 27 weeks. This means Activity *L* must start by week 27 to finish by week 29.

Next, we move to the preceding activity, Activity *K*. The latest finish time for Activity *K* is set to the latest start time of Activity *L*, which is 27 weeks. The latest start time for Activity *K* is then calculated by subtracting its expected duration (3 weeks) from its latest finish time, resulting in 24 weeks. Therefore, Activity *K* must start by week 24 and finish by week 27 without delaying the project.

The backward pass continues through the network similarly for all activities. For tasks with multiple succeeding activities, the latest finish time is the minimum of the latest start times of all immediate successors. By calculating the latest start and finish times for each activity, we can determine the amount of slack time available. Activities with zero slack time are on the critical path, meaning any delay in these activities will directly affect the project's completion date. The network diagram version of the automated reporting tool project plan is updated to include a second three-digit sequence for each activity that represents the latest start time, the expected duration, and the latest finish time (see figure 13.3).

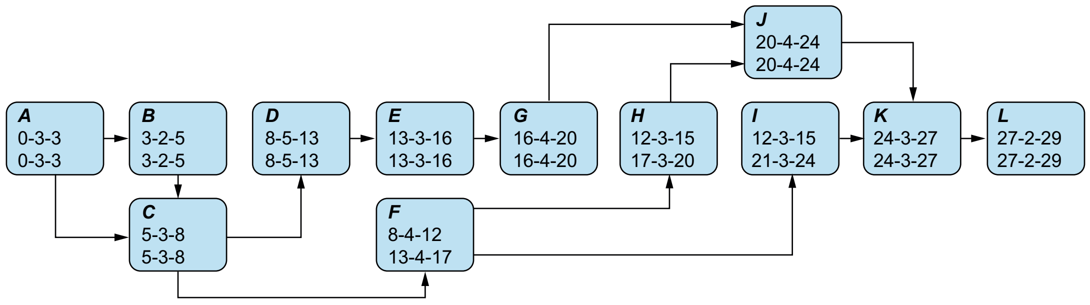


Figure 13.3 A second visual representation of the automated reporting tool project plan. The new three-digit sequence for each activity represents, in order, the latest start time, the expected duration, and the latest finish time. A quick and easy way of distinguishing critical tasks from noncritical tasks is to compare the three-digit sequences: critical tasks have similar three-digit sequences, whereas noncritical tasks do not.

Now that the earliest and latest start and finish times for each activity in the project plan are defined, calculating the slack time—that is, the amount of time that an activity can be delayed without causing a delay to subsequent activities or the overall project completion date—becomes a matter of simple arithmetic. This will allow us to determine the flexibility we have with scheduling each task without affecting the overall project timeline.

13.3.3 Slack

Slack is the amount of time an activity can be delayed without causing a delay to subsequent activities or the project's overall timeline. It is calculated as the difference between the latest and earliest start times or, alternatively, as the difference between the latest and earliest finish times (either method will return the same result). Activities with no slack are on the critical path, whereas those with some slack are not. In other words, activities with no difference between their earliest and latest start or finish times have zero slack and are therefore classified as critical path activities, meaning any delay in these will directly affect the project's schedule. On the other hand, activities that have a discrepancy between their earliest and latest times have some slack, thereby consigning them to a noncritical path. This slack allows these tasks to be delayed—up to a certain extent—without affecting the overall project timeline.

The results, both direct and derived, from the critical path method are presented in table 13.3. The earliest start and finish times were determined through a forward pass analysis of the project network, whereas the latest start and finish times were obtained via a backward pass analysis. Slack time for each activity was calculated by taking the difference between the latest and earliest start times (but could have been derived by instead taking the difference between the latest and earliest finish times). Activities with zero slack are classified as critical path tasks, meaning any delay in these activities will directly affect the project's overall timeline. Conversely, tasks with any amount of slack are not on the critical path.

Table 13.3 CPM results for the automated reporting tool project. Activities in any project have slack or no slack, which is derived by subtracting the earliest start from the latest start or, alternatively, by subtracting the earliest finish from the latest finish (it doesn't matter). Activities with no slack are on the critical path, whereas activities with slack are not.

Activity	Earliest start (ES)	Earliest finish (EF)	Latest start (LS)	Latest finish (LF)	Slack (LS – ES)	Critical path?
A	0	3	0	3	0	Yes
B	3	5	3	5	0	Yes
C	5	8	5	8	0	Yes
D	8	13	8	13	0	Yes
E	13	16	13	16	0	Yes
F	8	12	13	17	5	No

Table 13.3 CPM results for the automated reporting tool project. Activities in any project have slack or no slack, which is derived by subtracting the earliest start from the latest start or, alternatively, by subtracting the earliest finish from the latest finish (it doesn't matter). Activities with no slack are on the critical path, whereas activities with slack are not. ***(continued)***

Activity	Earliest start (ES)	Earliest finish (EF)	Latest start (LS)	Latest finish (LF)	Slack (LS – ES)	Critical path?
G	16	20	16	20	0	Yes
H	12	15	17	20	5	No
I	12	15	21	24	9	No
J	20	24	20	24	0	Yes
K	24	27	24	27	0	Yes
L	27	29	27	29	0	Yes

This activity analysis reveals two key takeaways, with the first being far more consequential than the second. Our first takeaway is that a substantial number of the project's activities are on the critical path, meaning any delay in these tasks will directly affect the overall timeline. In typical projects, there are usually multiple parallel paths of activities, with only one of these paths being critical. Consequently, the ratio of critical to noncritical activities is generally much smaller than what we observe in the automated reporting tool project. Regardless, project management should be laser-focused on critical path tasks, even at the expense of those activities that have slack; day-to-day monitoring, risk mitigation, and even project crashing strategies should be directed at critical path activities to be impactful. By concentrating on one or more critical path tasks, project managers can effectively expedite the project without unnecessary efforts on noncritical activities.

The second takeaway is that the noncritical activities have considerable, rather than moderate, slack. This slack provides flexibility, allowing these tasks to be delayed without affecting the project's end date. Additionally, the significant slack means there is no immediate risk of these noncritical activities becoming critical. Understanding this distinction helps in planning and resource allocation, ensuring that project management efforts are directed where they are most needed to maintain the project schedule.

Identifying the critical path manually, however, especially from a project plan containing dozens or even hundreds of activities, can be impractical and prone to errors. As project complexity increases, the interdependencies and scheduling nuances become too intricate to handle without automation. Therefore, finding the critical path is best done programmatically. Using Python, we can perform forward and backward passes through a virtual project network, derive the earliest start and finish times for every activity, and subsequently classify tasks as critical or noncritical. Manual methods can't scale with project complexity, so we need to understand how to apply programmatic solutions to project management techniques like the CPM.

13.3.4 Finding the critical path programmatically

Having manually demonstrated the CPM, we will now automate it using a mix of common Python libraries and user-defined functions. By first creating a pandas data frame with the required data—activities, their dependencies, and their expected durations derived from optimistic, pessimistic, and most likely estimates—we can replicate the process of calculating earliest and latest start and finish times, thereby determining the slack for each activity and identifying the critical path. This automation streamlines the process, allowing us to efficiently manage complex project schedules and ensure accurate results, even for large and intricate project networks.

Be aware that finding the critical path programmatically, although significantly more efficient than a manual approach, still requires a fair amount of effort, especially for large and complex projects with numerous dependencies and constraints. The process involves careful data preparation, defining logical sequences, and ensuring accurate calculations for early and late start and finish times. However, the consequences of *not* identifying the critical path far outweigh the effort required to determine it. Without a clear understanding of the project's most time-sensitive activities, delays can easily cascade, increasing costs, extending timelines, and potentially forcing reductions in project scope. By automating critical path analysis, project managers can proactively mitigate these risks, allocate resources more effectively, and ensure that high-priority tasks remain on track to meet project deadlines.

CREATING THE DATA FRAME

Our first snippet of code creates a pandas data frame called `plan` that contains the following four attributes:

- `Activity`—A unique code or identifier assigned to each activity or task
- `Description`—A short description of each activity
- `Dependencies`–The preceding activities, if any, that must be completed before the current activity can begin
- `t`–The expected time, in weeks, for each activity, representing a weighted average of the optimistic, pessimistic, and most likely time estimates

The data frame is then displayed by passing `plan` to the `print()` method:

```
>>> import pandas as pd
>>> plan = pd.DataFrame({
>>>     'Activity': ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',
>>>                  'K', 'L'],
>>>     'Description': [
>>>         'Define Project Objectives and Scope',
>>>         'Assemble Project Team',
>>>         'Gather Requirements from Stakeholders',
>>>         'Data Collection and Integration',
>>>         'Data Cleaning and Preprocessing',
>>>         'Design Report Templates',
>>>         'Develop Data Processing Pipelines',
>>>         'Implement Report Generation Logic',
>>>         'Develop User Interface for Report Access',
>>>         'Integrate Data Pipelines with Report Generation',
>>>         'Conduct User Testing and Feedback',
>>>         'Finalize and Deploy Automated Reporting Tool'
>>>     ],
>>>     'Dependencies': [None, ['A'], ['A', 'B'], ['C'], ['D'], ['C'],
                         ['E'], ['F'], ['F'], ['G', 'H'], ['I', 'J'],
                         ['K']],
        't': [3, 2, 3, 5, 3, 4, 4, 3, 3, 4, 3, 2]
})

print(plan)
  Activity                                      Description Dependencies  t
0        A              Define Project Objectives and Scope         None  3
1        B                            Assemble Project Team          [A]  2
2        C             Gather Requirements from Stakeholders       [A, B]  3
3        D                   Data Collection and Integration          [C]  5
4        E                   Data Cleaning and Preprocessing          [D]  3
5        F                          Design Report Templates          [C]  4
6        G                 Develop Data Processing Pipelines          [E]  4
7        H                  Implement Report Generation Logic          [F]  3
8        I           Develop User Interface for Report Access          [F]  3
9        J  Integrate Data Pipelines with Report Generation       [G, H]  4
10       K                 Conduct User Testing and Feedback       [I, J]  3
11       L     Finalize and Deploy Automated Reporting Tool          [K]  2
```

The `plan` data frame is our foundation. Now that it's established, we can proceed with automating the CPM.

WRITING ERROR-HANDLING FUNCTIONS

Error-handling functions are essential for managing and responding to errors that occur during the execution of a program. These functions ensure that the program can gracefully handle unexpected issues, provide informative feedback to the user, and terminate the program if necessary to prevent further errors or data corruption. In the context of our project, we have defined several error-handling functions to proactively address potential problems with the input data. The following code snippets illustrate how we intend to manage any subsequent errors related to the `Activity`, `Dependencies`, and `t` columns in our `plan` data frame:

```
>>> def errorActivityMsg():
>>>     print('Error in input file : Activity')
>>>     sys.exit(1)

>>> def errorDependenciesMsg():
>>>     print('Error in input file : Dependencies')
>>>     sys.exit(1)

>>> def errortMsg():
>>>     print('Error in input file : t')
>>>     sys.exit(1)
```

For instance, the `errorActivityMsg()` function prints an error message related to the `Activity` variable and terminates the program using `sys.exit(1)`, ensuring that the error is promptly handled and the program stops execution to prevent further issues.

RETRIEVING ACTIVITY INDICES FOR TASK IDENTIFICATION

The `getTaskCode()` function is designed to retrieve the row index of a specific activity code within a pandas data frame. This function is essential for efficiently locating and referencing tasks based on their unique identifiers, which is crucial for project management tasks such as critical path analysis. By checking whether the activity code exists in the `Activity` column and returning the corresponding row index, the function ensures accurate task identification. If the code does not exist, the function calls `errorActivityMsg()` to handle the error gracefully. This mechanism enhances the robustness and reliability of our project management automation:

```
>>> def getTaskCode(mydata, code):
>>>     x = 0
>>>     flag = 0
>>>     for i in mydata['Activity']:
>>>         if i == code:
>>>             flag = 1
>>>             break
>>>         x += 1
>>>     if flag == 1:
>>>         return x
>>>     else:
>>>         errorActivityMsg()
```

Initializes a user-defined function designed to retrieve the row index of a specific activity code

Sets the initial index counter to 0

Sets the initial flag to 0, indicating the code hasn't yet been found

Starts a loop to process each element in the Activity column

Checks whether the current activity matches the given code

If a match is found, the flag is set to 1.

Ends the loop, assuming a code is found

Increments the index counter for each subsequent iteration

Checks whether the flag is set, indicating the code was found

Returns the index of the found activity

Calls the error-handling function if the code is not found in the Activity column

Next, we will demonstrate how to automate the forward and backward passes through the project network to calculate the earliest and latest start and finish times for each activity.

AUTOMATING THE FORWARD PASS IN PROJECT SCHEDULING

The following code implements the forward pass algorithm, a crucial step in project scheduling for calculating the earliest start and earliest finish times for each task. This function iterates through the project's activities, checking their dependencies to determine when each task can begin and end. If a task has no dependencies, it can start at time zero. For tasks with dependencies, the function calculates the earliest possible start time based on the latest finish time of all preceding tasks. This ensures that each task's scheduling considers the completion of its dependent tasks. The resulting earliest start and earliest finish times are then appended to the `plan` data frame, enabling a comprehensive view of the project timeline:

```
>>> import numpy as np
>>> def forwardPass(mydata):
>>>     ntask = mydata.shape[0]
```

Imports the numpy library, which is often required for numerical computing

Initializes a user-defined function designed to calculate the earliest start and earliest finish times

Determines the number of rows (or tasks) and stores the results in ntask

```
>>>     ES = np.zeros(ntask, dtype = np.int32)
>>>     EF = np.zeros(ntask, dtype = np.int32)
>>>     temp = []
>>>     for i in range(ntask):
>>>         if not mydata['Dependencies'][i]:
>>>             ES[i] = 0
>>>             try:
>>>                 EF[i] = \
>>>                     ES[i] + mydata['t'][i]
>>>             except:
>>>                 errortMsg()
>>>         else:
>>>             for j in mydata['Dependencies'] \
>>>                     [i]:
>>>                 index = \
>>>                     getTaskCode(mydata, j)
>>>                 if index == i:
>>>                     errorDependenciesMsg()
>>>                 else:
>>>                     temp.append(EF[index])
>>>             ES[i] = max(temp)
>>>             try:
>>>                 EF[i] = \
>>>                     ES[i] + mydata['t'][i]
>>>             except:
>>>                 errortMsg()
>>>         temp = []
>>>     mydata['ES'] = ES
>>>     mydata['EF'] = EF
>>>     return mydata
```

Initializes an array to store the earliest start times as 32-bit integers

Initializes an array to store the earliest finish times as 32-bit integers

Initializes an empty list called temp to temporarily hold the finish time of dependencies

Starts a loop to process each task

Checks if the current task has no dependencies

Sets the earliest start time to 0 for tasks without dependencies

Calculates the earliest start time for tasks with dependencies by finding the maximum earliest finish time of those dependencies

Handles errors if t is invalid and calls errortMsg()

Initiates a separate set of procedures where or when there are dependencies

Starts a loop to process each dependency

Gets the index of each dependency

Checks for circular dependencies (that is, tasks dependent on themselves), handles errors if so, and then calls errorDependenciesMsg()

Appends the earliest finish time of the dependencies to temp

Sets the earliest start time to the maximum finish time of all dependencies

Calculates the earliest finish time as the sum of the earliest start time and t

Handles errors if t is invalid and calls errortMsg()

Resets temp for the next task

Appends the earliest start times as a new column to the data frame

Appends the earliest finish times as a new column to the data frame

Outputs the data frame so that it can be used or further processed by subsequent functions

With the forward pass completed, we have successfully automated the calculation of the earliest start and earliest finish times for each activity. Next, we will automate the backward pass algorithm to determine the latest start and latest finish times, further refining our project schedule and identifying potential slack for each activity.

Automating the backward pass in project scheduling

The following code implements the backward pass algorithm, which is essential for calculating the latest start and latest finish times for each task in a project schedule. This function iterates through the project's activities in reverse order, determining the latest times each task can start and finish without delaying the project. By considering the dependencies and successors of each task, the algorithm ensures that all tasks are scheduled efficiently, identifying potential slack and further refining the project timeline. The calculated latest start and latest finish times are then appended to the data frame, thereby providing a complete and automated view of the project's scheduling details:

```
>>> def backwardPass(mydata):
>>>     ntask = mydata.shape[0]
>>>     LS = np.zeros(ntask, dtype = np.int32)
>>>     LF = np.zeros(ntask, dtype = np.int32)
>>>     Successors = np.empty(ntask,
>>>                           dtype = object)
>>>     for i in range(ntask - 1, -1, -1):
>>>         if mydata['Dependencies'][i]:
>>>             for j in mydata['Dependencies'] \
>>>                     [i]:
>>>                 index = \
>>>                     getTaskCode(mydata, j)
>>>                 if Successors[index] \
>>>                         is not None:
>>>                     Successors[index] += \
>>>                         mydata['Activity'][i]
>>>                 else:
>>>                     Successors[index] = \
>>>                         mydata['Activity'][i]
>>>     mydata['Successors'] = Successors
>>>     max_EF = np.max(mydata['EF'])
>>>     for i in range(ntask - 1, -1, -1):
>>>         if mydata['Successors'][i] is None:
>>>             LF[i] = max_EF
>>>             LS[i] = LF[i] - mydata['t'][i]
>>>         else:
>>>             for j in mydata['Successors'][i]:
>>>                index = getTaskCode(mydata, j)
>>>                 temp.append(LS[index])
>>>             LF[i] = min(temp)
>>>             LS[i] = LF[i] - mydata['t'][i]
>>>             temp = []
>>>     mydata['LS'] = LS
>>>     mydata['LF'] = LF
>>>     return mydata
```

Initializes a user-defined function designed to calculate the latest start and latest finish times

Determines the number of rows (or tasks) and stores the results in ntask

Initializes an array to store the latest start times as 32-bit integers

Initializes an array to store the latest finish times as 32-bit integers

Initializes an empty array called Successors to store the successors of each task

Starts an outer loop to process each task in reverse order

Checks whether the current task has no dependencies

Starts an inner loop by iterating through each dependency of the current task

Retrieves the index for each dependency

Checks whether the successor list for the dependencies is already initialized

Appends the current task to the successors list of dependencies

Initializes the successor list for dependencies with the current tasks if it is not already initialized

Adds a new column to the data frame called Successors that contains values from the Successors array

Calculates the highest value of the earliest finish times from all tasks

Loops through each task, starting from the last one to the first

Checks whether the current task has no successors

Assigns the maximum earliest finish as the latest finish time for tasks with no successors

Computes the latest start time for tasks with no successors

Appends the latest start time of the successor to temp

Sets the latest finish time to the minimum value in temp

Calculates the latest start time as the difference between the latest finish time and t

Resets temp for the next task

Appends the data frame with the calculated latest start times

Appends the data frame with the calculated latest finish times

Outputs the data frame so that it can used or further processed by subsequent functions

With the backward pass automated, we now have both the earliest and latest start and finish times for each activity. By taking the difference between the earliest start and latest start times or, alternatively, the difference between the earliest finish and latest finish times, we can calculate the slack for each activity. This allows us to easily determine which tasks are critical, as they have no slack, and which tasks are noncritical, having some degree of flexibility. This comprehensive analysis ensures efficient project scheduling and management, enabling us to identify and focus on key activities that directly affect the project timeline.

Automating slack in project scheduling

The following code implements the slack calculation for each activity in the project schedule. Slack represents the amount of time an activity can be delayed without affecting the project's overall timeline. This function iterates through all the tasks, computes the slack by taking the difference between the latest start and earliest start times, and determines whether each task is critical. Tasks with zero slack are marked as critical, indicating they directly affect the project's timeline. The updated data frame includes the calculated slack and critical path information, with just the necessary columns. This comprehensive analysis helps in identifying tasks that need close monitoring to ensure timely project completion:

```
>>> def slack(mydata):
>>>     ntask = mydata.shape[0]
>>>     Slack = np.zeros(ntask, dtype = np.int32)
>>>     Critical = np.full(ntask, dtype = object)
>>>     for i in range(ntask):
>>>         Slack[i] =
>>>             mydata['LS'][i] - mydata['ES'][i]
>>>         if Slack[i] == 0:
>>>             Critical[i] = 'Yes'
>>>        else:
>>>            Critical[i] = ‘No’
>>>     mydata['Slack'] = Slack
>>>     mydata['Critical'] = Critical
>>>     mydata = mydata.reindex(
>>>         columns = ['Activity', 'ES', 'EF', 'LS', 'LF',
>>>                    'Slack', 'Critical'])

>>>     return mydata
```

Initializes a user-defined function designed to calculate the slack time for each task and derive whether or not each task is critical

Determines the number of rows (or tasks) and stores the results in ntask

Initializes an array of zeros with a length equal to the number of tasks, using 32-bit integers to store the slack times for each task

Initializes an array with a length equal to the number of tasks, defaulting each element to No to indicate that tasks are not critical

Starts a loop to process each task

Iteratively calculates the slack time for each task by subtracting the earliest start time from the latest start time and stores the results in an array called Slack

Iteratively checks the calculated slack time for each activity. If 0, marks the task as critical and otherwise as not critical.

Appends the calculated slack times to the data frame as a new column called Slack

Appends the critical status of each task to the data frame as a new column called Critical

Reorders and subsets the columns in the data frame

Outputs the data frame so that it can used or further processed by subsequent functions

With the columns subsetted and reordered to include only the necessary information, our data frame now provides a clear and comprehensive view of the project schedule, featuring automated calculations for the earliest and latest start and finish times, slack, and critical status for each task. This streamlined format enables efficient project management by highlighting critical tasks that directly affect the project timeline and identifying noncritical tasks with available slack. By automating these detailed analyses, project managers can prioritize resources and focus on key activities to ensure timely project completion, all while minimizing the risk of human error.

FINALIZING AND DISPLAYING THE AUTOMATED PROJECT SCHEDULE

The first snippet of code implements a wrapper function, `computeCPM()`, that integrates all the necessary steps for automating the CPM calculations. A wrapper function is designed to call multiple other functions, streamlining the process into a single call. Here, `compute CPM()` sequentially invokes the `forwardPass()`, `backwardPass()`, and `slack()` functions to compute the earliest and latest start and finish times, as well as the slack for each task. This comprehensive approach ensures that all aspects of the project schedule are calculated and updated in the data frame, providing a complete and automated analysis of the project's critical path:

```
>>> def computeCPM(mydata):
>>>     mydata = forwardPass(mydata)
>>>     mydata = backwardPass(mydata)
>>>     mydata = slack(mydata)
>>>     return mydata
```

Another user-defined function, `printPlan()`, displays the project schedule and slack times in a tabular format. By setting pandas to display all columns, the function ensures that none of the important information is truncated. It prints a header to describe the key metrics and then outputs the data frame containing the calculated values for each task. This visual representation helps project managers quickly understand the project's schedule and identify critical tasks that need close monitoring:

```
>>> def printPlan(mydata):
>>>     pd.set_option('display.max_columns', None)
>>>     print('Automated Reporting Tool: Schedule and Slack Times')
    >>> print('*' * 50)
    >>> print('ES = Earliest Start; EF = Earliest Finish;\nLS = Latest
    >>>     Start; LF = Latest Finish;\nSlack = LS - ES')
    >>> print('*' * 50)
    >>> print(mydata)
    >>> print('*' * 50)
```

The next line executes the `computeCPM()` wrapper function on the `plan` data frame, which of course encompasses the `forwardPass()`, `backwardPass()`, and `slack()` calculation functions. This single line automates the entire process of computing the earliest and latest start and finish times, as well as the slack for each task, updating the `plan` data frame with these calculated values. It effectively transforms the initial project data into a comprehensive schedule with critical path analysis, ready for further review or display:

```
>>> plan = computeCPM(plan)
```

And the final line of code calls the `printPlan()` function to display an automated version of table 13.3. This function outputs the complete project schedule, including the earliest and latest start and finish times, slack, and critical status for each task. It provides a clear and detailed visual representation of the project timeline, helping project managers quickly understand the schedule and identify critical tasks that require close monitoring:

```
>>> printPlan(plan)
Automated Reporting Tool: Schedule and Slack Times
**************************************************
ES = Earliest Start; EF = Earliest Finish;
LS = Latest Start; LF = Latest Finish;
Slack = LS - ES
**************************************************
   Activity  ES  EF  LS  LF  Slack Critical
0         A   0   3   0   3      0      Yes
1         B   3   5   3   5      0      Yes
2         C   5   8   5   8      0      Yes
3         D   8  13   8  13      0      Yes
4         E  13  16  13  16      0      Yes
5         F   8  12  13  17      5       No
6         G  16  20  16  20      0      Yes
7         H  12  15  17  20      5       No
8         I  12  15  21  24      9       No
9         J  20  24  20  24      0      Yes
10        K  24  27  24  27      0      Yes
11        L  27  29  27  29      0      Yes
**************************************************
```

With the process now fully automated, project managers can efficiently finalize and display their project schedules. As long as you organize your project plan—specifically, a WBS with the estimated duration for each activity—into a pandas data frame that matches the provided format, you can repurpose this code for your own projects. This method scales seamlessly, allowing you to handle projects of any size and complexity and automatically determine the critical path, ensuring a more effective and streamlined project management process.

13.4 Estimating the probability of project completion

Stakeholders have been apprised that, based on critical path analysis, the automated reporting tool project is expected to take 29 weeks. However, it is crucial for the project to be completed within 30 weeks to ensure that resources can be reassigned to other essential duties. Projects often need to meet specific deadlines for a variety of reasons: to avoid contractual fines, to capitalize on market opportunities by getting a product to market promptly, to align with other project schedules, or to fulfill regulatory requirements.

For example, a delayed product launch could mean missing a critical sales window, resulting in significant revenue loss. In construction projects, delays can lead to penalties or increased costs due to prolonged equipment rentals. In the tech industry, missing a market window can allow competitors to gain a foothold, affecting long-term business viability.

The project manager is concerned because of the variance in many of the critical path activities. If the pessimistic estimates for these activities come to fruition, the likelihood of completing the project in 30 weeks, let alone 29 weeks, could be in jeopardy. This uncertainty necessitates a thorough risk analysis to estimate the probability of project completion within the desired timeframe, taking into account the potential for delays and the effect of these variances on the overall schedule.

The variance for any activity is calculated by inserting the optimistic (a) and pessimistic (b) time estimates into the following formula:

$$\text{Variance} = \left(\frac{b-a}{6}\right)^2$$

So, if Activity A is optimistically estimated to take 2 weeks and pessimistically estimated to take 4 weeks, the variance for that activity is

$$\text{Variance}_{(A)} = \left(\frac{4-2}{6}\right)^2$$

or

$$\text{Variance}_{(A)} = \frac{1}{9}$$

The variance for an activity represents the degree of variability or uncertainty in the activity's time estimates. A higher variance indicates greater unpredictability in the duration, whereas a lower variance suggests more consistency and reliability in the time estimates. Table 13.4 shows the variances for every activity in the automated reporting tool project.

Table 13.4 Variances in time estimates for automated reporting tool activities, with critical path activities highlighted in bold. The project variance is derived, in part, by summing the variances of the critical path activities.

Activity	Optimistic, a	Pessimistic, b	Variance, ((b – a)/6)2
A	**2**	**4**	**1/9**
B	**1**	**3**	**1/9**
C	**2**	**4**	**1/9**
D	**3**	**7**	**4/9**
E	**2**	**4**	**1/9**

Table 13.4 Variances in time estimates for automated reporting tool activities, with critical path activities highlighted in bold. The project variance is derived, in part, by summing the variances of the critical path activities. *(continued)*

Activity	Optimistic, a	Pessimistic, b	Variance, ((b – a)/6)2
F	2	6	4/9
G	**2**	**6**	**4/9**
H	2	4	1/9
I	2	4	1/9
J	**2**	**6**	**4/9**
K	**2**	**4**	**1/9**
L	**1**	**3**	**1/9**

Just as the project duration is derived by summing the expected times for every activity on the critical path, the project variance is derived by summing the variances from only the critical path activities, assuming the activity times are statistically independent. Hence, the project variance is

$$\text{Project Variance} = \frac{1}{9} + \frac{1}{9} + \frac{1}{9} + \frac{4}{9} + \frac{1}{9} + \frac{4}{9} + \frac{4}{9} + \frac{1}{9} + \frac{1}{9} = \frac{18}{9} = 2 \text{ weeks}$$

The standard deviation is equal to the square root of the variance:

$$\text{Project Standard Deviation} = \sigma = \sqrt{2}$$

This is approximately 1.41 weeks.

Assuming project completion times follow a normal probability distribution, we can therefore divide the difference between the due date and the project duration by the standard deviation to get a z-score (see chapter 3):

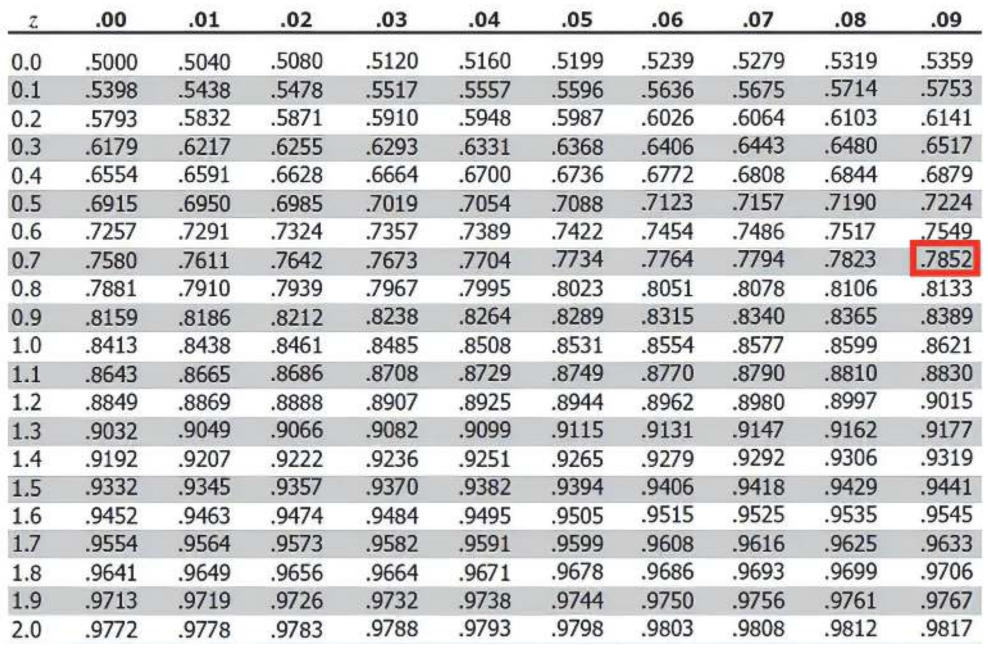
$$z = \frac{\text{due date} - \text{project duration}}{\sigma}$$

or

$$z = \frac{30 - 29}{1.41}$$

This equals 0.709, where z is the number of standard deviations the due date is above the project duration.

If the project duration represents the mean of a normal probability distribution, there is an equal 50/50 chance of completing the project on time. Therefore, if the due date is beyond the mean, the probability of completing the project within 30 weeks is greater than 50%. The exact probability can be determined by looking up the z-score in a z-score table (see figure 13.4).



z	.00	.01	.02	.03	.04	.05	.06	.07	.08	.09
0.0	.5000	.5040	.5080	.5120	.5160	.5199	.5239	.5279	.5319	.5359
0.1	.5398	.5438	.5478	.5517	.5557	.5596	.5636	.5675	.5714	.5753
0.2	.5793	.5832	.5871	.5910	.5948	.5987	.6026	.6064	.6103	.6141
0.3	.6179	.6217	.6255	.6293	.6331	.6368	.6406	.6443	.6480	.6517
0.4	.6554	.6591	.6628	.6664	.6700	.6736	.6772	.6808	.6844	.6879
0.5	.6915	.6950	.6985	.7019	.7054	.7088	.7123	.7157	.7190	.7224
0.6	.7257	.7291	.7324	.7357	.7389	.7422	.7454	.7486	.7517	.7549
0.7	.7580	.7611	.7642	.7673	.7704	.7734	.7764	.7794	.7823	.7852
0.8	.7881	.7910	.7939	.7967	.7995	.8023	.8051	.8078	.8106	.8133
0.9	.8159	.8186	.8212	.8238	.8264	.8289	.8315	.8340	.8365	.8389
1.0	.8413	.8438	.8461	.8485	.8508	.8531	.8554	.8577	.8599	.8621
1.1	.8643	.8665	.8686	.8708	.8729	.8749	.8770	.8790	.8810	.8830
1.2	.8849	.8869	.8888	.8907	.8925	.8944	.8962	.8980	.8997	.9015
1.3	.9032	.9049	.9066	.9082	.9099	.9115	.9131	.9147	.9162	.9177
1.4	.9192	.9207	.9222	.9236	.9251	.9265	.9279	.9292	.9306	.9319
1.5	.9332	.9345	.9357	.9370	.9382	.9394	.9406	.9418	.9429	.9441
1.6	.9452	.9463	.9474	.9484	.9495	.9505	.9515	.9525	.9535	.9545
1.7	.9554	.9564	.9573	.9582	.9591	.9599	.9608	.9616	.9625	.9633
1.8	.9641	.9649	.9656	.9664	.9671	.9678	.9686	.9693	.9699	.9706
1.9	.9713	.9719	.9726	.9732	.9738	.9744	.9750	.9756	.9761	.9767
2.0	.9772	.9778	.9783	.9788	.9793	.9798	.9803	.9808	.9812	.9817

Figure 13.4 The top of a typical z-score table. The probability is found where the integer and remaining fractional parts of the value intersect.

The probability is found where the first two digits of the z-score, located in the far-left column of the table, intersect with the next two digits of the z-score, located along the top row. For example, a z-score of 0.709 corresponds to a probability of 0.7852, or 78.52%, which is found at the intersection of 0.7 and 0.09. Therefore, there is a better than 78% chance of completing the project within 30 weeks or less.

We can use `scipy.stats` to compute the cumulative probability directly from the z-score without the use of a z-score table:

```
>>> import scipy.stats as stats
>>> due_date = 30
>>> mean = 29
>>> standard_deviation = 1.41
>>> z_score = (due_date - mean) / standard_deviation

>>> probability = stats.norm.cdf(z_score)

>>> print(f'z-score: {z_score:.3f}')
    z-score: 0.709
```

```
>>> print(f'Probability of completing the project within '
>>>       f'{due_date} weeks: {probability:.2%}')
    Probability of completing the project within 30 weeks: 76.09%
```

The discrepancy between the probability obtained from the z-table (78.52%) and the probability calculated using Python's `scipy.stats` library (76.09%) can be attributed to differences in precision and rounding methods. Z-tables typically provide rounded values and sometimes use interpolation to offer probabilities between listed points. On the other hand, `scipy.stats.norm.cdf()` computes the probability using precise floating-point arithmetic, which can result in slightly different values. Despite this minor difference, the overall effect on project completion probability is relatively immaterial. Both methods yield a close approximation, and the key takeaway remains that there is a high probability of completing the project within the required timeframe. For most practical purposes, either method provides a sufficiently accurate estimate to inform project management decisions.

To better understand the probability of project completion, it's helpful to visualize a normal probability distribution. Figure 13.5 illustrates this concept, where the due

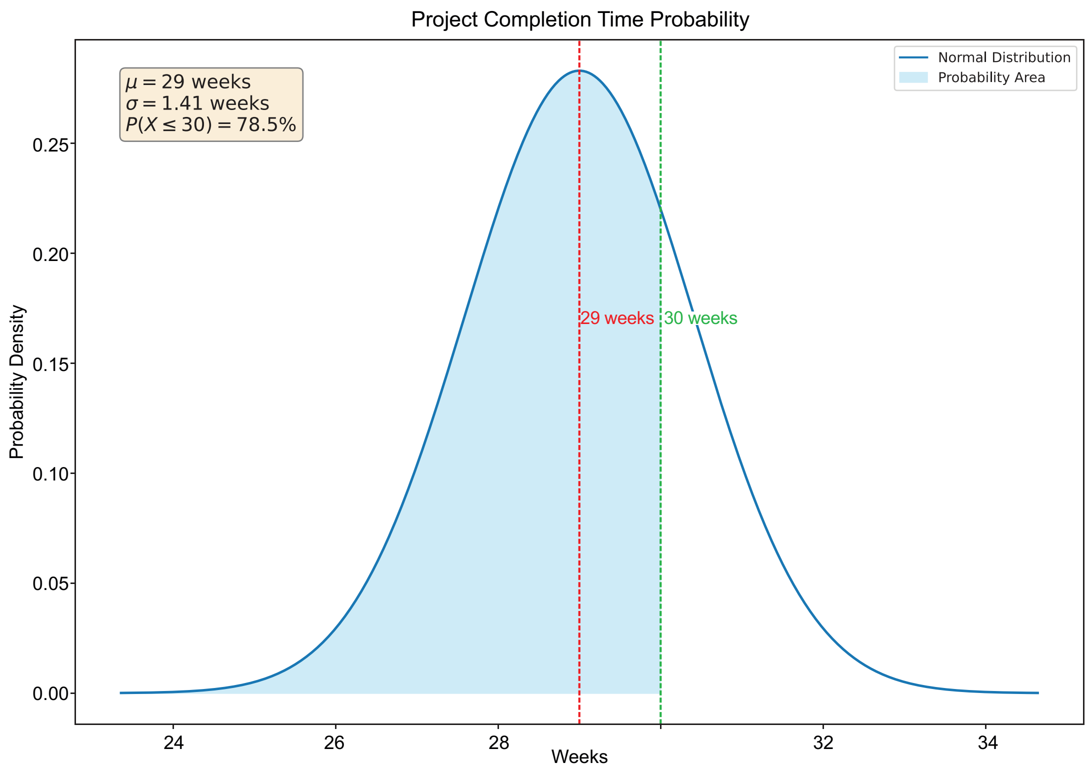


Figure 13.5 Normal probability distribution of project completion time, where the mean duration is 29 weeks and the standard deviation is 1.41 weeks. The due date of 30 weeks is marked, showing a 78.5% probability (using the z-table results) of completing the project within this timeframe.

date is 30 weeks, the expected project duration is 29 weeks, and the project standard deviation is 1.41 weeks. This plot shows the distribution of possible project completion times, and the shaded area represents the probability of completing the project within 30 weeks. The vertical dashed lines indicate the mean and the due date, providing a clear visual representation of how the project timeline aligns with the expected duration and the due date. By examining this plot, we can see that there is a significant probability of completing the project on or before the 30-week deadline, quantified by the shaded area under the curve up to the due date.

Although the automated reporting tool project stands a good chance of being completed within 30 weeks or less, there may still be opportunities to compress the schedule, possibly at an additional cost, without compromising on scope. Implementing schedule compression techniques, such as fast-tracking or crashing, could further increase the probability of meeting the 30-week deadline and provide added assurance to stakeholders.

13.5 Crashing the project

Crashing a project involves accelerating the overall project timeline by reducing the duration of one or more critical path activities. This typically incurs additional project expenses, as accelerating the schedule often requires increased labor costs. It's a fundamental principle of project management that altering one parameter, such as the schedule, inevitably affects other components, like cost or scope. Despite these potential increases in project expenses, exploring options for crashing the project can provide stakeholders with valuable choices. They can decide whether to invest in accelerating the timeline to increase the probability of completing the project within 30 weeks or even less, or to maintain the current schedule, considering that a 76% to 78% chance of finishing on or before the due date is already a favorable likelihood.

To maximize the cost-effectiveness of project crashing, follow these four steps:

1. Change out the expected time for each activity with the most likely time, and recalculate the critical path. For the automated reporting tool project, recalculating the critical path is unnecessary because the expected and most likely times for each activity are similar. However, this is not typically the case. When there is a difference between these times, simply replace the expected times with the most likely times in the pandas data frame and re-run the Python code that automates the critical path processes to determine the new critical path. Otherwise, focus on those activities that lie on the (new or previous) critical path, as these directly affect the project completion date. Crashing noncritical activities will increase the project cost and not reduce the overall project duration.
2. Calculate the weekly crash cost (or other unit of time) for every activity using the following formula:

$$\text{Weekly Crash Cost} = \frac{\text{Crash Cost} - \text{Normal Cost}}{\text{Normal Time} - \text{Crash Time}}$$

This calculation helps to determine the additional cost incurred for each unit of time saved by accelerating an activity, thereby allowing for the prioritization of which tasks to crash based on both cost and time efficiency. It assumes that crash costs are linear.

3. Select one or more critical path activities with the lowest weekly crash costs. Crash these activities to the maximum extent possible or as needed to meet the deadline or significantly improve the probability of finishing before the due date.
4. Ensure that crashing does not alter the existing critical path or create a new critical path. Carefully monitor the project schedule after applying crashing techniques to confirm that the intended activities are shortened without inadvertently shifting the critical path or adding new critical activities, which could negate the benefits of crashing and introduce new scheduling risks. For the automated reporting tool project, this is not a concern due to the significant slack variances between critical and noncritical activities.

The most likely times and crash times per activity, along with their corresponding costs, are then summarized in table 13.5.

Table 13.5 Most likely and crash estimates for the automated reporting tool project in weeks and dollars. The most effective way of accelerating a project timeline is to crash one or more critical path activities with the lowest weekly crash costs.

Activity	Time (weeks)		Cost ($)		Weekly crash cost ($)	Critical path?
	Most likely	Crash	Most likely	Crash		
A	3	2	5,000	6,000	1,000	Yes
B	2	1	4,000	5,000	1,000	Yes
C	3	2	5,000	6,000	1,000	Yes
D	5	4	9,000	11,000	2,000	Yes
E	3	2	5,000	6,000	1,000	Yes
F	4	3	8,000	9,000	1,000	No
G	4	2	8,000	10,000	1,000	Yes
H	3	2	5,000	7,000	2,000	No
I	3	2	5,000	6,000	1,000	No
J	4	2	7,000	10,000	1,500	Yes
K	3	1	5,000	7,000	1,000	Yes
L	2	1	4,000	5,000	1,000	Yes

The most effective way to accelerate the timeline of the automated reporting tool project is to focus on crashing the critical path activities with the lowest weekly crash costs. Based on the data, the best (but not only) candidates for crashing include Activities *A, B, C, E, G, K,* and *L.* These activities not only lie on the critical path but also have relatively low additional weekly costs when crashed. These activities each have a crash cost of $1,000 per week, making them cost-effective choices for reducing the project duration. Additionally, Activities *D* and *J,* despite having slightly higher weekly crash costs as compared to the other critical activities, still present viable options due to their significant effect on the overall project timeline. By selectively crashing these activities, the project manager can optimize resource allocation, minimize additional costs, and significantly improve the likelihood of completing the project within the desired timeframe.

More complex projects, however, require more advanced crashing methods. To crash the automated reporting tool project (or any project) most effectively, we can set up a constrained optimization problem (see chapter 8). The objective function in this case would be to minimize the total crashing costs, whereas the constraints would include the allowable crash time per activity, the project deadline, and the dependencies between activities.

We have demonstrated how the use of quantitative methods can make a significant difference between unsuccessful and successful project management—and unsuccessful and successful projects. These techniques provide the precision and clarity needed to effectively plan, execute, and control projects. In the next and final chapter, we will focus on establishing quality control measures and visualizing them.

Summary

- Managing projects effectively requires a solid foundation in quantitative skills. These skills enable project managers to analyze data, forecast outcomes, and make informed decisions based on numerical evidence. Proficiency in areas such as scheduling, risk assessment, and resource allocation is critical. These competencies help in structuring plans that are realistic and achievable, ensuring project objectives are met efficiently.
- A work breakdown structure is pivotal in project planning and management. It systematically breaks down a project into manageable sections and tasks, making it easier to organize, schedule, and delegate. The WBS serves as the foundation for the project planning process, helping to identify all necessary activities, allocate resources appropriately, and establish a timeline that supports the project's objectives.
- The program evaluation and review technique is utilized especially in scenarios where there's a lack of analogous estimating data. PERT focuses on calculating time estimates for project activities by considering optimistic, pessimistic, and most likely durations. This approach accommodates uncertainty and variability in project planning, allowing for more robust and resilient scheduling.
- The critical path method identifies the longest sequence of dependent activities and calculates the shortest possible project duration. Automating CPM enhances accuracy in identifying the critical path, reduces manual errors, and

speeds up the reevaluation of the project schedule in response to changes. This automation supports dynamic project management and allows real-time decision-making to keep the project on track.

- Calculating the probability of on-time project completion involves using statistical methods to assess the likelihood of meeting project deadlines. By computing the z-score for the project duration and using it to estimate the cumulative probability, project managers can quantify the chances of completing the project by a specified due date. This method provides a clear, numerical understanding of schedule risks, enabling better planning and decision-making to ensure timely project delivery.
- Project crashing is a technique used to shorten the duration of a project by decreasing the time for one or more critical path activities. This is often achieved at additional costs, typically by assigning overtime to current project team members or by adding resources. The decision to crash a project is usually made when the benefits of completing the project sooner outweigh the additional expenses and other factors.
- Applying quantitative skills to project management requires significant analytical effort and careful planning, even with automation. However, the consequences of inadequate project management—delays, budget overruns, and compromised scope—are often far greater. Poor planning is a leading cause of project failure, and using quantitative methods helps mitigate risks, optimize schedules, and improve overall project outcomes.