Chapter 3

Making IT Right: Managing Goals, Time, and Costs

3.1 Before You Start: Assessing Value and Risks

Projects create new products, new services, or new capabilities. The first step of a sound management process is to understand whether the new products, services, or capabilities are worth our effort. The relevance of a project depends, in general, upon two main factors:

1. The **value** generated by the project
2. The **risks** associated with the project.

The meaning of *value* and *risk*, however, is not absolute and depends on the circumstances and on the project environment. For instance, a project developed for humanitarian reasons measures value in a different way from a project to launch a commercial product. Similar is the concept of risk. As mentioned by Maylor (2010), the first projects related to the Apollo mission, although they were considerably high risk, were critical to gain the know-how necessary to send a man to the moon. A typical scenario in the software industry is represented by a **make or buy** decision, namely, choosing between developing a new system or acquiring an existing one with features similar to those needed.

In the rest of this section, we look at factors and techniques to assess the value of a project. These can be used with different purposes:

1. To decide whether a project is worth pursuing
2. To select which project to start out of a portfolio of possible proposals
3. To choose the best project plan, given a project with different plans.

3.1.1 Project Value: Aspects to Consider

Three main factors determine the value generated by a project:

1. **Direct and indirect value**. As mentioned earlier, the value of a project does not refer necessarily and only to the revenues it generates directly and through its outputs. Considerations relative to the social and environmental impact, image and publicity, entering a new market, and know-how acquired are some of the considerations that could add or subtract value from a project.
2. **Sustainability**. Many IT projects start without an idea or a strategy to sustain their outputs. Thus, the outputs of a project might not live long after a project and its resources end. Taking into account the operational costs of a project's outputs and the way in which the project outputs will survive after a project ends is an important consideration to understand whether a project is worth doing.
3. **Alignment with the strategic objectives of the organization**. Ensuring that the project aligns with the goals of an organization is an essential point to consider before a project is worth starting. Alignment with the strategic objectives can determine the **priority** of a project. As pointed out in Maylor (2010), Toyota is a leader in defining priorities: projects are started only if they directly contribute to one of the strategic objectives of the company, namely, *quality*, *cost*, or *delivery performance*.

3.1.2 Project Risks: Aspects to Consider

Various factors determine the risk profile of a project. Among them are

- **Resource availability**. Projects require the availability of resources—human, financial, and technical—in specific time frames. Although it might be difficult to preempt the required resources in advance, a check on the project's needs is a good sanity check to verify whether a project is worth pursuing.
- **Timing**. Many projects have specific time windows for the delivery of their outputs. Deliver too early or too late and the outputs of the project might be useless. Consider, for instance, a project to build a rocket to reach another planet. The actual launch can occur only on a specific time frame, to take advantage of the relative position of planets. Deliver the rocket too early, and docking and maintenance might become an issue. Deliver too late and you might lose the opportunity to launch.
- **Technical difficulty or uncertainty**. The success of many projects relies on the actual capability of solving various technical challenges. Pointing out what

these challenges are, understanding the level of risk associated with such challenges, and possible corrective or alternative courses of action are important in determining the values and risks of a project.

- **Project environment and constraints**. Projects are influenced by various constraints, both internal and external. Various internal and external stakeholders will have an interest in positively or negatively influencing a project. Regulations and standards can severely limit what can be done on a project.

3.1.3 Techniques to Assess Value and Risks

Different techniques are available to assess the value and risk generated by a project. Some are based on financial considerations, while others are more qualitative. In the following, we present some of the most used techniques.

3.1.3.1 Financial Methods

3.1.3.1.1 Payback

The simplest financial evaluation is the **payback** method, which measures how long it will take to return a project's investment. When using the payback method, an estimation of the project expenses and incomes determines the profits and losses at the end of each project year. The payback is the year at which the project covers all expenses and starts earning. The shorter the payback, the better.

The payback favors projects that minimize financial exposure. One of the issues with payback is that it does not take into account total profit. This second issue is that it does not measure the *efficiency* with which the money invested in the project is paid back.

3.1.3.1.2 Return of Investment

To overcome some of the limitations of the payback method, another technique that is often employed is the **return of investment** (ROI), which measures how much we get back for each dollar invested. ROI is calculated from the **annual profit**, defined as the *average profit per year* and computed by dividing the profit by the duration of a project:

$$\text{Annual profit} = \frac{\text{incomes} - \text{expenses}}{\text{project duration}} \tag{3.1}$$

The ROI is then computed by dividing the *annual profit* by the total project expenses:

$$\text{ROI} = \frac{\text{annual profit}}{\text{expenses}} \tag{3.2}$$

3.1.3.1.3 Net Present Value and Internal Rate of Return

Payback and ROI do not take into account the effects of inflation, namely, the fact that an amount of money in the future has a lower value than the same amount available now. (In a sense, "better an egg today than a hen tomorrow"). Thus, for longer projects, payback and ROI tend to overestimate profits, which are usually gotten toward the end of a project.

The **net present value** technique (or NPR for short) overcomes this issue by taking into account the inflation rate. Thus, if a reliable estimation of the inflation rate can be provided, the value of future expenses and incomes for a project can be recomputed in terms of their actual value.

In particular, when using NPR, profit and losses are computed using the following formula:

$$\text{Value} = \frac{1}{(1+r)^i} * \text{amount} \tag{3.3}$$

where r is the inflation rate, *amount* is the net profit at year i, and *value* is the current value of *amount*. Note that the first project year is year 0.

An even more complex method is the **internal rate of return** (IRR for short), which determines the inflation rate which zeroes profits. The interested reader can consult Burke (2006), which contains a nice discussion about financial methods.

3.1.3.1.4 Applying Financial Methods: An Example

Consider two projects, called "Project A" and "Project B," for which we have estimated expenses and incomes as described in Table 3.1. In particular, the table shows that project A is not profitable in the first 2 years and then starts earning. Project B has a similar behavior, but both expenses and incomes are higher.

Suppose we have the resources to start only one of the two projects and we use a financial method to choose which project to activate.

If we use the payback method, we select Project A, since it has a shorter payback period. In fact, year 0 is forecast to end with losses, for Project A. So will year 1. At the end of year 2, however, the financial statement will show earnings of €10,000.

Table 3.1 Assessing Two Projects Using Financial Methods

	Project A	*Project A*
Year 0	−€20,000.00	−€30,000.00
Year 1	−€10,000.00	−€30,000.00
Year 2	€40,000.00	€50,000.00
Year 3		€100,000.00

Project B's payback is 4 years. Years 0 and 1 end with losses. In year 2, project B earns profits of €50,000, but these are not yet sufficient to cover the expenses of the first 2 years. In year 3, however, the project pays back the initial investment.

If we use the ROI method, we select Project B, since it has the highest ROI.

Project A, in fact, has an ROI of 11%, computed as follows:

$$\text{Annual profit} = \frac{€40{,}000 - (€20{,}000 + €10{,}000)}{3} = €3333 \tag{3.4}$$

$$\text{ROI} = \frac{€3333}{€300{,}000} = 11\% \tag{3.5}$$

The ROI of Project B, whose calculation we leave to the reader, is 38%.

3.1.3.2 Score Matrices

Financial methods help determine the financial viability of a project, but tell nothing about the project characteristics that are not measurable with profits and losses. Therefore, other methods have been proposed to assess a project. One of the simplest is the **score matrix**, which allows one to measure a project along several dimensions and assign it a value.

A **score matrix** is a list of project criteria, each of which is assigned a weight, which measures the importance the criteria have for us or for the organization we work for. The criteria highlight the desirable and undesirable aspects of a project; the weights of desirable features are positive numbers (e.g., from 1 to 5) and the weights of undesirable features are negative numbers (e.g., from −1 to −5).

When we evaluate a project using a score matrix, we measure how well the project satisfies each criterion we have identified, for instance, by assigning a number from 1 (very low) to 5 (very high). We then multiply the scores with the weights and sum all values. Projects scoring a higher value are more desirable than projects with a lower score. Projects can also be compared side by side; hence, the use of the term "matrix" in the name of the technique.

EXAMPLE 3.1

Table 3.2 shows an example of a score matrix used to evaluate three different projects.

The starting point is a list of criteria and weights, which we imagine have been selected by an evaluation committee. Note that the last criterion is negative and it has been assigned a high relevance. Thus, the selection process will tend to favor projects in which stakeholders are not difficult to manage.

The second step is the evaluation of how well each criterion is met by a project. Table 3.2, for instance, shows the value assigned to each project.

The third and final step is computing the scores, which are shown in the last row of the table. According to the data, "Project 1," the one with the highest score, is preferred over the others.

Table 3.2 A Score Matrix Example

			Project 1		*Project 2*		*Project 3*	
Factor	*Description*	*Weight*	*Value*	*Total*	*Value*	*Total*	*Value*	*Total*
Profit >30%	The project will yield a profit >30% if no exceptional events occur	3	4	12	4	12	4	12
Low-risk profile	The project does not present particular risks. That is, there is no risk with a very high impact	2	2	4	3	6	3	6
Schedule is not tight	Project delivery does not require activities to be performed in a very tight schedule	3	3	9	2	6	2	6
Manageable complexity	The complexity is manageable	2	1	2	1	2	1	2
Consistent with current business	The project is mainstream with the activities of the organization	1	1	1	1	1	1	1
Stakeholders	Stakeholders are difficult to manage	−4	2	−8	4	−16	3	−12

3.1.3.3 SWOT Analysis

The **strengths, weaknesses, opportunities, and threats** analysis technique is credited to Albert Humphrey, who used it to determine the competitive advantages of the Fortune 500 companies in the 1970s (Friesner, 2013). The technique can also be used to evaluate the feasibility of a project.

The SWOT analysis is usually performed on a two-by-two matrix, like that shown in Figure 3.1. The analysis proceeds by identifying the strengths, weaknesses, opportunities, and threats related to the project under analysis, and by listing them in the matrix shown in Figure 3.1. (Other formats, of course, are possible.) Once the elements of the SWOT analysis are identified, decision makers use the information to evaluate whether the opportunities are worth the effort and how strengths can overcome weaknesses and threats.

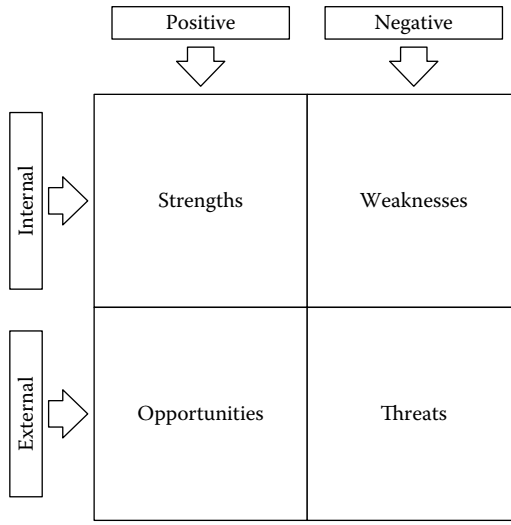

Figure 3.1 A SWOT matrix.

3.1.3.4 Stakeholder Analysis

Stakeholders can exert quite a lot of influence on a project and determine the success or failure of a project. Understanding how stakeholders can influence, positively or negatively, a project is good practice to assess the project's chances of success and to define a **stakeholder management policy**.

The stakeholder identification process is informal as it usually proceeds with a mental swipe of the project environment and the actors who might be directly or indirectly involved or affected by the project and its outputs. Once the stakeholders have been identified, the next step consists in understanding what kind of influence each stakeholder can exert in a project. This allows one to *cluster* the stakeholders and define specific policies for each cluster.

Various ways have been proposed to classify stakeholders. In Maylor (2010), stakeholders are classified in two dimensions:

1. The **power** they can exert in the project
2. The **interest** they have in the project.

This classification allows one to define different policies according to the positioning of the stakeholders in this two-dimensional space.

The extreme case is the one with high-influence and high-power stakeholders. In this case, careful analysis and specific treatment are necessary.

Other situations can use more generic strategies. For instance, it is good practice to keep high-influence and low-power stakeholders informed, while it is safer to keep low-influence and high-power stakeholders satisfied.

Finally, low-influence and low-power stakeholders require minimum effort.

For more complex stakeholder analyses, Yu et al. (2011) propose the **i*** model, which was developed for requirements engineering. The model is based on two concepts, **actors** and **goals**, and models dependencies between these entities. Using the notation, it is thus possible to identify, for each project goal, the stakeholders who have the most influence, whether this influence is positive or negative, and their motivations. The information can then be used to define adequate management strategies.

3.1.3.5 Assessing Sustainability

Evaluating the operational cost of the project outputs helps assess the long-term benefits of a project and what additional actions a project should include to ensure that its results will live after the project ends. This analysis is relevant in several high-risk or highly constrained projects, such as projects to start new companies, research projects, and projects to foster economic development. Often, in fact, these projects deliver benefits as long as steady financing is available; little or nothing survives when the project ends.

For products and services, the two important pieces of information that constitute the basis for a sustainability analysis are (1) the definition of the **business model**, a specification of how the operating costs will be paid for, and (2) when applicable, how profit will be generated. At a minimum, a good business model includes a **value proposition**, which clearly identifies the offering, that is, the product or service being offered and its value for the customers. Additionally, an analysis of the key partners, the key activities, and the key resources allows one to understand which resources are needed. Finally, a financial analysis of costs and revenues helps understand the profitability. This is composed of an analysis of the **cost structure**, which identifies costs and the most costly resources, and of a **revenue stream** analysis, which lists the sources of incomes, highlighting the most important sources.

The financial analysis might also include a **break-even** analysis, which identifies the break-even point, that is, the point at which a product generates neither losses nor gains. If the break-even point is not reached, a product will cause losses; any number of items sold above the break-even point will yield a gain.

The break-even point is computed by looking at three pieces information: the operational costs, the price of each item, and the number of items. The first determines the minimum profit that has to be reached. The second determines the profit in terms of items sold. The break-even point is the number of items in which profit equals the operational costs.

See The Business Model Generation (2013) for a freely available and very compact template, which also includes the identification of **distribution channels** and **customer segments**.

3.1.3.6 A Recap of Project Selection Techniques

Table 3.3 recaps the project selection techniques presented so far, highlighting the type of support each technique offers to evaluate the value and risk generating factors of a project.

Table 3.3 Recap of Project Selection Techniques

	Payback	*ROI*	*NPR*	*Goal Analysis*	*Score Matrix*	*SWOT*	*Stakeholder Analysis*
Direct value	Fully	Fully	Fully	Partially	Partially	Partially	
Indirect value	Partially	Partially	Partially		Partially		
Sustainability	Partially	Partially	Partially		Partially		
Alignment with strategic objectives				Fully	Partially	Fully	
Resource availability					Partially	Fully	
Timing					Partially	Fully	
Technical difficulty or uncertainty					Partially	Fully	
Environment and constraints				Fully	Partially	Fully	Partially

Some techniques provide better coverage than others to specific characteristics. This is qualitatively captured in the table using the terms "Fully," "Partially," or leaving a cell blank, when the technique does not offer support. Thus, for instance, the stakeholder analysis helps one to get a better understanding of the project environment. The technique, however, is very specific and does not offer support to measure other criteria.

3.1.4 The Project Feasibility Document

The **project feasibility document** closes the assessment phase by describing the main characteristics of a project and by analyzing, using the techniques presented above, the value and risks of a project. The document can be used to formally authorize the start of a project.

Thus, a feasibility document contains at least the following information:

- A **statement of work**, which describes what the project will accomplish
- The **business objectives** of the project and its outputs (value) and information about the **business model**, if relevant
- A summary of the **project budget**, which forecasts expenses and incomes
- A summary of the project **milestones**, which is a rough schedule of the project identifying the most important events
- An **analysis of the stakeholders**
- The project **risks**

- Possible **alternatives** to the project, such as a **make-or-buy** decision
- An **evaluation** of the project and alternatives, using the techniques described above.

3.2 Formalizing the Project Goals

If the analyses performed with the feasibility study are convincing enough, one of the first activities is fixing and having stakeholders agree on the project scope. This, in fact, constitutes the basis for any further management activity, namely, the definition of the work to be performed, schedule and budget, and the skills required by the team that will perform the work. Project goals, together with the schedule and the budget, are also the basis for contractual agreements with the clients.

Defining project scope is one of the most delicate activities in setting up a project, since

1. It ensures that the project includes all and only the work necessary to achieve the project goals.
2. It establishes a baseline of the work to be performed.
3. It defines a reference document for project acceptance.

The first point is obvious. Adding unnecessary work, in fact, would cause a burden that needs to be sustained either by the project team, which will work on features that are not really necessary, or by the client, who pays for unnecessary work, which, in any case, would negatively affect the project, adding risks and increasing costs.

Ensuring that a project will not include any useless work, however, can be more difficult than it looks. Among the reasons, we mention uncertainties about the product to be built. A customer, for instance, might be uncertain about whether a particular feature is needed or not and, in doubt, add it among the ones to be developed, just to play on the safe side.

In a more difficult scenario, an ambiguous description of the project goals causes the stakeholders to form slightly different opinions about the objectives to achieve. This lack of **integrity** in a project's vision will most likely cause additional work, glitches, and anomalies.

A clear project scope mitigates the risks illustrated above by ensuring that all project stakeholders form a clear view of the project.

The second and third items on the list should also be quite clear. The specification of the project goals determines the characteristics of the products to build. These, in turn, define the work to build the project outputs and the criteria to verify whether the goals have been met.

An additional consideration is that sometimes tight timing constraints tend to compress this activity, shortening the time the manager has to precisely define the goals. The assumption is that the goals are clear and agreed among the stakeholders, even if there is no document describing them in details. Therefore, the limited time

available in the project is better used in more productive activities. The risks of this approach, however, typically become clear too late.

The project scope is fixed in a **project scope document**. In particular, it is a good idea to include at least the following information in a project scope document:

- **Project goals and requirements**, which describe what we intend to achieve with the project and the main characteristics of the project and its outputs
- **Assumptions and constraints**, which describe the conditions which have to be met for the project to succeed
- **Project outputs and control points**, which describe the *outputs* of the project and, in some cases, a rough timing of their delivery
- **Project Roster**, which describes the *who*.

The first two items should be written in a way that also defines the **project acceptance criteria**, namely, the minimum conditions for the client to accept a project. Although my experience (together with that of many other project managers) is that client and stakeholders satisfaction is more important than the syntactic compliance with the acceptance criteria, making these explicit beforehand can contribute significantly to establishing a good relationship with the stakeholders and simplify project closing.

Another important piece of information that might be included in a scope document is the procedure to manage changes, namely, how request for changes will be dealt with. This is described in more detail in Chapter 4.

In the rest of the section, we look in more detail at the main content of the scope document.

3.2.1 Project Goals and Requirements

The **project goals and requirements** define what is inside and what is outside the scope of a project and the characteristics of the project outputs.

To elaborate a bit, two good practices help make the goals **SMART** and assign them priorities using the **MoSCoW** classification.

SMART stands for **simple**, **measurable**, **agreed upon**, **realistic**, and **time bound**. A goal is SMART when it has the qualities specified by the acrynom. Thus, a SMART goal is simple in its formulation, so that there are no ambiguities in its interpretation; it has measurable criteria to understand whether it has been achieved or not; it is agreed to by all the stakeholders; it can be reached with the resources available in the project; and finally, it has a date by which it has to be reached.

MoSCoW allows the project manager to assign a priority to the goals. The acronym stands for **must have** (features that are essential), **should have** (features that are important but not essential), **could have** (features that would be nice to have), and **won't have** (features that will not be included in the product).

Note that a classification with MoSCoW allows the manager to distinguish between **base scope**, which is the scope required to meet the business requirement

and **value-added scope**, which is discretionary but improves the economics of the overall project. This is similar to what Cameron (2005) and Tomczyk (2005) suggest with the identification of the critical success factors.

In many cases, it is also worthwhile to point out goals and work products that will **not** be delivered by the project, because they fall **outside the scope of the work**. The list, of course, should focus on elements typically included in similar projects or items some stakeholders might *assume* as included in the project scope. The goal is to reduce ambiguities and false expectations.

Consider, for instance, a project related to the development of a one-off software for a specific client. Although not explicitly mentioned in the project scope, some stakeholders might take for granted that user documentation and training will be part of the deal. The work necessary for such items, however, might be beyond the resources available to the team. Making clear that such items will not be included can be a way to better align expectations with actual delivery.

Note that, in general, project requirements differ in two ways from the software requirements we have introduced in Chapter 2. The first is that the project requirements are a superset of the software requirements. Many software development projects, in fact, will include activities that are indirectly related to the software being built, such as, for instance, training of resources, production of user manuals, hardware procurement, setup of an infrastructure to provide user support, and setup of the infrastructure to distribute the system. The second is that project requirements are often at a higher level of abstraction than the software requirements.

A final test that should be performed on the requirements is double checking that the **project goals are under the power and control of the project team**. If they are not under the control of the project team, in fact, they are, in the best scenario, under the control of some other project stakeholder or, in the worst scenario, completely out of the control of the project. In the first case, it is better to list the constraints and assumptions that make the goals achievable (see Section 3.2.2). In the second case, their achievement will depend on the good luck of the manager.

Once again, things are not so simple. Consider, for instance, a project related to the experimentation of a new technology. One could set the following measurable criteria as a project goal: "the system will be used by 20,000 people during the experimentation." However, unless people are coerced to use the system, there is no way for the project manager to **ensure** that the system **will** be used by 20,000 users. A more realistic wording could highlight that the "experimentation will be set so that at least 20,000 people will be offered the chance to try the system."

3.2.2 Project Assumptions and Constraints

Project **assumptions** and **constraints** define important hypotheses on which the manager bases the achievement of the project goals.

Assumptions are those conditions that are considered to be true, but might not in fact be. Assumptions are not under the control of the project manager, but they might be under the control of some project stakeholders. When this is the case, assumptions can be used to define the duties and obligations of project stakeholders.

Consider a case in which the installation of a new system requires the client to stop operating his or her business for a given period. Stopping operations is clearly not in the power of the project manager, who will list this operation as an essential assumption for project success.

Whether they are under the control of project stakeholders or not, assumptions should be properly addressed in the risk management plan, finding appropriate management strategies in case they cannot be satisfied or turn out to be false (see Section 4.2).

Constraints, by contrast, are known limitations, which shape and define the work we can do. They are used to explain why we set some goals and not others and why we structure the work in some way rather than another.

3.2.3 Project Outputs and Control Points

A **deliverable** is defined in Project Management Institute (2004) as **a unique, measurable, and verifiable work product**. Deliverables are the result of work performed in the project and, in many cases, they are also the prerequisites of project activities. For software development projects, examples of deliverables include a software system, a requirements document, and a user manual.

A very common classification distinguishes between **internal** and **external** deliverables. The former are functional to the implementation of the plan and are used only by the project team. As such, they can maintain a level of informality, which often simplifies their production and management. The latter are delivered to the customers. They often require additional work to ensure proper quality and formal procedures and, in some cases, a bit of ceremony, like, for example, delivery in hard copy through courier.

Deliverables might contain sensitive information. It is thus good practice to define the **dissemination level** of each deliverable. In common scenarios, circulation can be public, limited to selected stakeholders, to the project team, or only to selected project members. Such classification is an integral part of the project communication plan and is described in further detail in Section 5.3.

Milestones are defined by the Project Management Institute (2004) as **a significant event in the project**. Milestones are identified, at a minimum, by a label and a date, and they are typically used to highlight, significant control points, in the plan. Examples of milestones include a *mid-term project review*, or *phase transition milestones*, to identify the transition from one project phase to the next.

Deliverables and milestones can be presented as textual lists (like we do in the example below). They are also very often inserted in the graphical representation of

plans (e.g., AON or Gantt chart), with the advantage of showing which activity or activities are responsible for the production of any specific deliverable.

According to the formality of the project development process that is being adopted, milestones can be used in different ways:

- To form (and present) a **high-level roadmap** of the project: milestones represent how the project unfolds in a series of significant events.
- As a **verification point** in the project: milestones identify control points in the project, which serve as a general "orientation" mechanism to steer the project in one direction or another.
- As a **gate** in the project: milestones clearly separate different phases of the project; if the goals of the milestone are not achieved, the transition to the next phase is blocked.

There is no mechanical technique to identify the milestones and deliverables of a project. They depend on the project type and their identification can be supported by adopting a development standard, by personal experience, or by discussing and negotiating them with the project stakeholders.

EXAMPLE 3.2

Table 3.4 shows the standards the European Union enforces for the specification of deliverables produced by the research projects it sponsors. In particular, the following information is associated with each deliverable:

- A unique identifier, typically an integer number
- The name of the deliverable
- The nature of the deliverable, which can be one of the following: a report (R), a prototype (P), a demonstrator (D), or other (O)
- The dissemination level, which, simplifying a bit on the EU rules, can be public (PU), restricted to the project team (RE), or restricted to the project stakeholders (CO)
- The delivery date, expressed in the number of months after the start of the project
- The partner (team member) responsible for the delivery of the project.

Similarly, Table 3.5 shows (a subset of) the milestones of a European Research project. Milestones have the following information:

Table 3.4 An Example of Deliverables

Del. ID	*Deliverable Name*	*Nature*	*Dissemination Level*	*Delivery Date (Project Month)*	*Responsible Parter*
1	Requirements	R	CO	0	FBK
2	Architecture UML diagram	R	CO	1	FBK
3	Software	P	PU	6	FBK
4	User manual	R	PU	12	FBK

Table 3.5 An Example of Milestones

Milestone Number	*Milestone Name*	*Date (Month from Start)*	*Means of Verification*
M0.1	Kick-off	1	Kick-off meeting done, meeting minutes available, project collaboration tools available.
M0.2	Experimental sites 1&2 up and running	12	Experimental sites 1 & 2 up and running.
M0.3	Midterm review	18	At least 80% of activities starting before M18 have started; at least 80% of activities ending before M18 have ended.
M0.4	Experimental sites 2&4 up and running	24	Experimental sites 3 & 4 up and running.

- A unique identifier
- A name
- A date, for instance, expressed in months from the start of the project
- A description of the purpose of the milestone
- Means of verification, which specify how it is possible to verify the achievement of the work associated with the milestone.

3.2.4 Project Roster

The **project roster** is the list of people participating in the project, together with their role and other information, such as the contact point. The project roster is a simple practice that allows the project manager to identify the project stakeholders and, in the process, simplify the definition of a project communication plan and favor team interaction.

3.3 Deciding the Work

Now that we have properly described the main goals and the boundaries of our project; we can start identifying the activities that we need to carry out in the project. This is very often accomplished with a **work breakdown structure** or WBS from now on. The notation, developed in the 1960s alongside the program evaluation and review technique (PERT), is defined in Project Management Institute (2004) as "a (deliverable-oriented) hierarchical decomposition of the work to be executed by the project team to accomplish projects objectives and create the required deliverable."

Today, the technique is widely adopted and many project management and process standards make its use compulsory. For instance, NASA (1994) and NASA

(2007) require a WBS to be built for each major program or project and suggest its use in any project, big or small, when it can be practically done so.

A WBS establishes the basis for

- Defining the work to be performed in a project
- Showing how various activities are related to the project objectives
- Establishing a framework for defining, assigning, and monitoring work and costs
- Identifying the organizational elements responsible for accomplishing the work.

See PERT Coordinating Group (1963) for a more detailed description.

In the rest of this section, we are going to describe the main construction techniques for WBSs and some rules of thumb to build and evaluate their quality and soundness.

3.3.1 Building a WBS

A WBS is a tree in which the root node represents a project or its main output. Each level of decomposition shows how a node of the tree can be structured and organized in more elementary work components.

WBSs come in two different notations. The first is graphical: the WBS is shown as a tree unfolding from top to bottom. The second is textual and similar to the table of contents of a book. The nodes of a WBS can be labeled according to their position in the tree: the top level is numbered "1," its children "1.1," . . ., "1.n," and so on for all the nodes.

EXAMPLE 3.3

Figure 3.2 shows a graphical representation of a WBS for the development of a software application for mobile phones.

The WBS is structured in four levels.

The first level of decomposition contains the main activities, including the procurement of tools we need for development (1.5) and writing the user manual (1.6). Tests (1.3.4) are organized in three different types of activities: unit tests (1.3.4.1), system tests (1.3.4.2), and integration tests (1.3.4.3).

From the example above, we can infer various important characteristics of a WBS:

1. The WBS does not specify the order in which the activities have to be executed, nor does it specify any dependency among activities. For instance, the procurement of development tools has to be completed before coding can start; the WBS, however, does not show this dependency.
2. The decomposition follows the **100%** and the **mutual exclusion rules**. That is, each level of decomposition includes all and only the items that are nec-

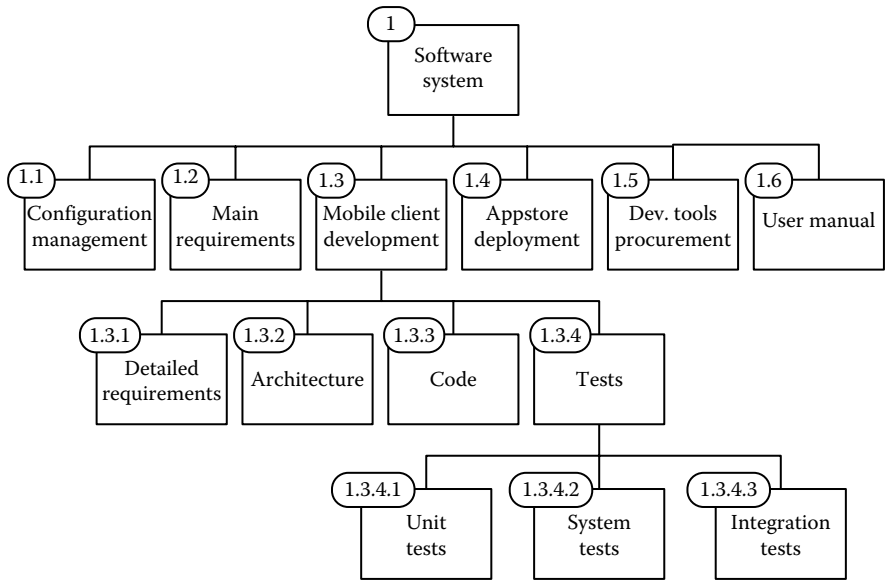

Figure 3.2 WBS example.

essary to develop the parent node. Moreover, there are no overlaps between nodes. Each node specifies work different from that of any other node. In this way, the WBS becomes a powerful tool to define what is in the scope of the project and to allocate responsibility for the development of each activity.

3. The WBS tree does not need to be balanced. In the diagram, for instance, some activities stop after the first level of decomposition, while others are refined up to the fourth level.
4. The WBS can contain support activities, such as "management" and "development tools procurement."

The level of decomposition and detail of a WBS depends upon its use. For instance, if the WBS is used as the basis for planning, its refinement process stops when we reach a level for which we can reliably estimate duration and effort.

According to NASA (1994), the leaves of the WBS must be such that the "completion of an element is both measurable and verifiable by persons (i.e., quality assurance persons) who are independent of those responsible for the element's completion." Thus, the advantage is that the WBS provides a solid basis for planning and monitoring and "no other structure (e.g., code of account, functional organization, budget and reporting, cost element) satisfactorily provides an equally solid basis for incremental project performance assessment."

3.3.2 WBS Decomposition Styles

Different decomposition styles allow one to build a WBS.

In a **product-oriented** WBS, the decomposition proceeds by identifying the items that must be developed to build deliverables. A product WBS thus establishes a one-to-one correspondence between project activities and project (sub)products. This simplifies accountability, since the responsibility for a group of activities in a project will correspond to the responsibility of delivering a specific system component. When using a product-oriented WBS, a good rule is to ensure that tightly connected components are not separated in the WBS, since such a decomposition style does not allow one to allocate responsibility for integration.

In a **process-oriented** WBS, the decomposition proceeds by taking into account the activities that are necessary to carry out the project. One advantage of the process-oriented decomposition is that it can include activities, such as management, which are not directly related to the development of a product, but which are still necessary in a project. Another advantage is that it is simpler to build a WBS by analogy, using a similar project as a reference. This can also be considered a weakness, since the WBS could result in being too generic and uninformative.

To get the best of both worlds, one can use a **hybrid** WBS. A hybrid WBS contains both process- and product-oriented nodes. This approach is the one suggested, for instance, by NASA (2007), where a part of the WBS is a specification of the components of a product and the remaining parts are the activities necessary to manage the project, integrate the components, and perform quality control.

Other types of WBS highlight the **organizational** or **geographical** aspects of work. For these WBSs, the first level of decomposition contains, respectively, the **organizational structure** or the **geographically distributed teams** responsible for the development of a group of activities. Starting with the second level, the WBS contains the work to be performed. These decomposition styles can be effective for highly cross-functional projects or for projects in which geographical distribution is significant.

As a specific case of organizational WBS, we mention that defined in NASA (1994) and the Department of Defense (2011), which distinguish between **Project Work Breakdown Structure** (PWBs) and **Contract Work Breakdown Structures** (CWBs). In this case, the top levels of the WBS contain a logical structuring of the project or program, while the lower levels represent the WBSs of the contractors responsible for the development of the project. Thus, for instance, NASA (1994) and the Department of Defense (2011) suggest defining PWBs organized at three levels:

1. Level 1 is the entire project or program.
2. Level 2 includes the projects or major elements to develop the project.
3. Level 3 includes the components necessary to develop the major elements.

Level 3 items constitute the statement of work for contractors and the first level of decomposition of the CWBs, whose development proceeds according to the standards of the different contractors, as shown in Figure 3.3.

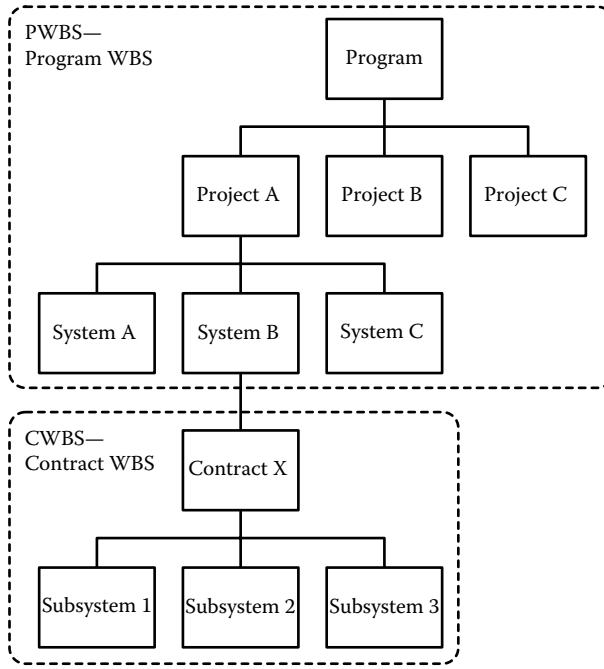

Figure 3.3 Example of PWBS and CWBS.

Remark

When defining the work to be performed, the term **work package** is often used. There are different usages and slightly different definitions of the term, so it might be worthwhile to have a look at them.

According to the Project Management Institute (2004), a work package is an element of the WBS at which **estimation for time and costs can be reliably provided**. In bigger projects, this can correspond to the leaves of the WBS, rather than the higher level of the WBS. Note, however, that the leaves of a WBS could be the top level element of finer-grained work breakdown structures, like we have seen for a contract WBS.

NASA (1994) adopts a different definition. A work package is the **unit of work required to complete a specific job**, such as a report, a test, a drawing, a piece of hardware, or a service, which is within the responsibility of one operating unit within an organization.

Finally, in common practice, the term *work package* is often used to denote the **first level of decomposition of a WBS**. This is, for instance, the notation used for research projects sponsored by the European Union and other funding agencies.

3.3.3 WBS Dictionary

A **WBS dictionary** helps to annotate each element of the WBS with more detailed and structured information, such as

- Title and item number, to connect the description to an element of the WBS
- Detailed description of the element, including, for instance, quantities, relevant associated work, and contractual items, where applicable.

Additional information can help manage a WBS dictionary over time. Some WBS dictionary templates thus require one to provide **references and links to other elements of the project**, such as references to the scope document, budget and reporting, contract reference, and **information about the history of the element**, such as, for instance, revision number, author, and authorization.

See CDC (2013) for a template of a WBS dictionary and Space Division—North American Rockwell (1971) for a good example of a WBS and a WBS dictionary.

3.3.4 WBS Construction Methodologies

WBSs can be built top-down, bottom-up, or by analogy.

In the **top-down** approach, the construction of the WBS proceeds from the top level down to the leaves. It is usually best suited for projects that are well known or whose structure is clear. One risk is overlooking activities (e.g., achieving a "90% decomposition").

In the **bottom-up** approach, the process proceeds in the opposite way. First, the leaves of the WBS are identified and then these are grouped in homogeneous items, thus giving structure to the WBS. It is best suited for projects that are new for a company or for the team responsible for their development and they work better with brainstorming sessions in which the whole team is involved. One technique that can be used for building the WBS elements is the so-called **post-it on a wall**. As each element of the WBS is identified, it is written on a post-it and posted on a wall. The post-it can then be physically grouped together to form the WBS structure. The main risks with bottom-up constructions are building WBSs that are too detailed and violate the mutual exclusion rule.

The third option, construction by **analogy**, starts from an existing WBS, which is adapted and customized for the project at hand. It can be very effective when an organization standardizes the structure of its projects.

3.4 Estimating

Yogi Berra is reported to have said that "it is difficult to make predictions, especially about the future." The sentence could not be more appropriate for estimations of the work to be performed in a project. When we consider software projects, some

of the characteristics of software, such as intangibility, flexibility, and complexity, make the estimation process even more complex.

In this section, we will look at some of the most common estimation techniques. The starting point is a discussion about what characterizes a project task. We then discuss the nature of estimations and continue with the presentation of the main techniques to estimate.

3.4.1 Effort, Duration, and Resources

Estimation is the process that determines the requirements to carry out an activity. These are expressed in terms of

- Duration, namely, how long an activity will last
- Effort, namely, the amount of work necessary to complete an activity
- Resources, necessary to complete an activity.

Duration is the amount of time an activity lasts. It is measured in calendar units, such as days, weeks, months, and years. Sometimes the string "calendar-" is prefixed to unambiguously specify the nature of the measure.

Effort is the amount of work required to perform a task, and is measured using man-hours, man-days, man-months, or man-years meaning, respectively, the amount of work expressed by one worker in an hour, a day, a month, or a year. Thus, for instance, an activity requiring 40 man-hours can be completed by a person working for 40 h.

The main **resource** needed for software development projects is **manpower**, which is expressed in terms of **units of work**, that is, the effort that can be produced per calendar period.

Manpower is further qualified by identifying the **type of resource** required to carry out the work, namely, by identifying which kind of competences and what kind of personnel is needed. For instance, the development of a simulator for a rocket system might require the work of an expert in aerodynamics.

Finally, **manpower** is determined by the **work calendar** and the **percentage of availability**.

The first determines the maximum number of units of work that can be expressed during a calendar unit. For the service industry, typical values for one resource (one person) are 8 man-hours per calendar-day and 40 man-hours per calendar-week, considering 2 days of rest per week. Other calendars are used. For instance, industries working in shifts have a one-to-one relationship between units of work and calendar time; this is achieved by having three people working 8 h each throughout the day.

The percentage of availability reduces the maximum presence of a resource and it is used when a resource is not available full-time. For instance, if a resource is working part-time for an organization or for a project, the percentage of availability might be 50%.

For many tasks, a simple relationship links effort, duration, and manpower:

$$D = \frac{E}{M} \tag{3.6}$$

where D is the duration of an activity, E is the effort required by the activity, and M is the manpower required to carry out an activity. Of the three variables, one is usually estimated, another chosen, and the third computed.

The equation holds for *reasonable* values of D, E, and M. In fact, as M increases, so does the burden of coordinating the work and exchanging information (Brooks, 1995). So, please, never plan to use 1000 people so that you can finish in half a day an activity that requires an effort of 500 man-days, like a famous Dilbert cartoon suggested.

Most activities are estimated either in duration or in effort, according to the nature of the work. For instance, the activity "writing a document" is best estimated by looking at the effort. In this case, for instance, we can estimate the effort, choose the manpower to allocate to the task, and use the equation above to compute the duration of the task.

The equation does not hold for any type of task, though. For instance, the activity "waiting for the foundations of a home to solidify" requires a fixed duration and is independent of manpower and effort.

Finally, note that some activities might also require one to specify other types of resources, namely, materials and equipment:

- **Material** is necessary for certain activities and is consumed while work progresses. The measurement unit for material depends on the kind of material used. For instance, in building a house, a given amount of concrete will be used to carry out the construction activities.
- **Equipment** includes the tools required for carrying out work. In an oil exploration project, for instance, certain activities will require drilling equipment. Equipment is measured by the number of units that are necessary to carry out a given activity. Equipment is not consumed by the execution of the activity; that is, after the activity has been completed, the equipment can be used for another purpose. The availability of equipment introduces constraints in a plan. Thus, the equipment typically specified in a plan includes tools that are available in limited quantities or that are costly to use (and might impact the project's budget). Consider, for instance, the development of a software system to control a robotic surgeon. Certain project activities might require access to a robotic arm. If this requirement is made explicit in the plan, it becomes possible to schedule activities so that no overlaps or conflicts arise in the usage of this limited resource.

3.4.2 The "Quick" Approach to Estimation

The simplest and probably most commonly used approaches to estimation are **expert judgment** and **analogy**. The project manager, possibly in

collaboration with the team or other experts, provides estimations using his/her experience or by looking at similar projects. The approach has the advantage of being very fast and simple. See the next section, however, for a discussion about some of the limitations.

These estimations can either proceed bottom-up or top-down.

In the bottom-up approach, the manager provides an estimation for each leaf of the WBS. If the estimations are effort-based, these can be easily propagated upward, thus determining the effort required for each node of the WBS. For instance, the effort required by a node A of the WBS whose children are $A_1, \ldots, A_n$ is the sum of the efforts of its children, namely, $\mathit{effort}(A_1) + \cdots + \mathit{effort}(A_n)$.

In the top-down approach, the process is reversed. The manager provides an estimation for the overall project. If the estimations are effort-driven, the effort of each activity of the WBS is then determined by distributing it to the lower levels. The propagation to the lower levels, however, is constrained by a relationship that is weaker than the one we defined above. If a node A of the WBS, for which we have estimated an effort of E_a, has children $A_1, \ldots, A_n$, then we can only say that $\mathit{effort}(A_1) + \cdots + \mathit{effort}(A_n) = E_A$, but we cannot tell exactly how much effort has to be allocated to each of $A_1, \ldots, A_n$, using the structure of the WBS only. Rules of thumb are often used: for instance, the total effort of a project is split into different activities in percentages that are similar to those measured in previous projects.

Whether a top-down or bottom-up approach is more appropriate depends on the project at hand. The rule of thumb is that top-down estimations will tend to underestimate the duration or effort (since they might abstract away details), while bottom-up estimations tend to overestimate the effort or the duration, because too much importance is given to details. In my experience, bottom-up estimations are simpler to come out with. It is, however, a subjective matter and your experience might be different.

If more than a person is involved in the estimation process, various techniques can be used to elicit information. We mention the Delphi method, which is presented in more detail in Section 5.3.3.3.

3.4.3 The Uncertainty of Estimations

(Software) project managers make many implicit and explicit assumptions when estimating the resources necessary for an activity and many estimations are based on a number of "ifs."

For instance, consider the problem of estimating how long it will take to complete a "requirement definition" activity, which produces a software requirements document. The reasoning could proceed as follows: *if* the final requirement document will be *about* 100 pages and *if* analysts can produce *about* 2.5 pages per hour, then the work will require *about* 40 man-hours. If one person will be able to dedicate *about* 80% of her time, then the actual duration will be *about* 50 h. The final estimation depends upon all these assumptions. For instance, if the estimation on

the number of pages to write is increased by 10% and the productivity is reduced to 2 pages per hour, the duration of the tasks increases to 68.75 h.

On top of the "random" guess of the project manager, work has some implicit variability, which depends on many factors. Weather, for instance, might interfere with the construction of a house. The productivity of workers changes the speed at which progress is achieved.

We can look at estimations as random variables characterized by a mean and a variance. Thus, precise scheduling techniques should or could be based on the rules of probabilistic reasoning. This is what certain techniques, like PERT and critical chain management, do. However, the requirement of coming out with *precise* numbers (e.g., "When does the project end? How much does it cost?") and the need to keep plans simple (e.g., "The requirements document will be delivered on April 1, 2013" and not "between April 1 and April 13, with a probability of 68%") favors an approach in which we choose a value and use it like it was a certain measure.

Thus, when we provide an estimation, we give our best guess of a task duration. An important implication, however, comes from considering the error between our estimation and the mean value of the actual duration (work) of a task. In fact, estimations (and estimators) can be **optimistic** if they are below the mean value or **pessimistic**, if they are above the mean value. In the first case, the actual plan is more likely to run late; in the second case, the plan will more likely over allocate resources (e.g., time and manpower). This is shown in Figure 3.4, where the area

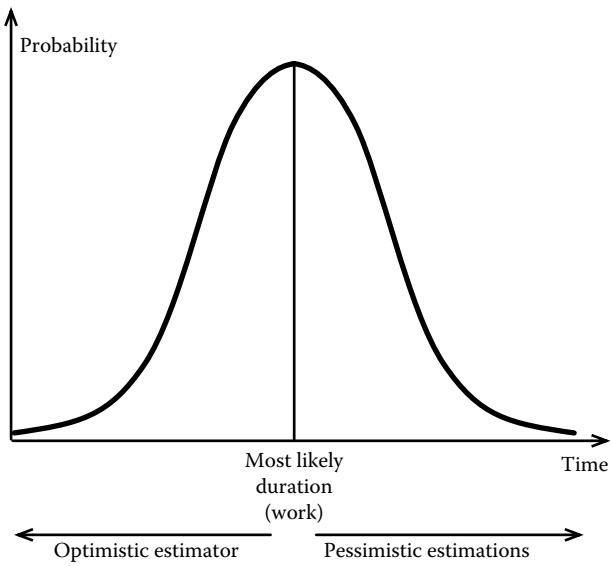

Figure 3.4 Estimations as random variables.

left of the mean corresponds to an optimistic estimation, while the area right of the mean includes the pessimistic estimations.

We tend to be optimistic in our estimations. For this reason, it is often the case that estimators *pad* their estimations to make sure they end above the most likely duration. If worst comes to worst, the reasoning goes, we will end earlier. Some rules of thumb go up to doubling the value of the estimation. Thus, in the example above, given our initial estimation of 40 h, we would schedule a plan in which the task takes twice as much time, namely, 80 h. This would easily accommodate the variations we mentioned above, but the consequence, in general, is that of building plans that are equally unrealistic.

There is also a subtler risk. As we acknowledge our implicit or explicit padding, we might be tempted to use estimations as a negotiation factor. If our estimations do not satisfy upper management (which is responsible for allocating resources) or our client, we might be tempted to just change the numbers so that they end up as the ones our stakeholders expect. Consider the example above: if 40 h of writing requirements are too long, why not reduce them to 30? After all, it is still relatively close to what we believe to be the most likely duration. The risks of such an attitude are clear, both in the short term and in the long term. In the short term, our plan becomes unreliable, since it is based on numbers chosen to please stakeholders, rather than to measure the work that has to be done. In the long term, it becomes difficult not only to make sense of our past plans, but also to use past experience and know-how to become more reliable in our estimations.

Various techniques have been proposed to tackle or mitigate the problems described above. **PERT** considers the random nature of estimations. **Algorithmic techniques** support the estimation process of the software system by relying on mathematical models that take stock of data collected over the years. **Agile methodologies**, by contrast, take a more radical approach by iterating the estimation process many times till convergence is reached or the project ends.

3.4.4 PERT

PERT was developed in the 1960s as a methodology to define and monitor projects (PERT Coordinating Group, 1963; Hamilton, 1964). The goal of the technique is to assess the probability of a plan to finish on a certain date, given the probabilistic estimations of the activities of composing the plan.

To use PERT, three values have to be assigned to each activity: an **optimistic value**, a **most likely value**, and a **pessimistic value**. (The values can be elicited using the techniques described above.) The most likely duration of an activity is then computed assuming a beta distribution. According to the distribution, the median duration of an activity is

$$t = \frac{(a + 4m + b)}{6} \tag{3.7}$$

where a is the optimistic value, b the pessimistic value, and m the most likely value. The variance of t is given by the formula

$$\sigma^2 = \frac{(b-a)^2}{6} \tag{3.8}$$

Standard statistical methods can then be used to sum all the activities of a plan and determine, in such a way, the duration of the overall plan, together with the confidence of the data.

3.4.5 Algorithmic Techniques

Algorithmic techniques determine the effort or the duration required for developing a software system, given some of its characteristics. In other words, given a set of measurable features of a system $x_1, \ldots, x_n$, an algorithmic technique defines a function $f(x_1, \ldots, x_n)$, such that

$$f(\overline{x_1}, \ldots, \overline{x_n}) \tag{3.9}$$

returns the *effort*, *duration*, and *team size* required to develop a system described by $\langle \overline{x_1}, \ldots, \overline{x_n} \rangle$. The function f is typically defined by analyzing and interpolating data of sample projects for which both the input values (the measurable characteristics) and the output characteristics (effort, duration, and manpower) are available.

The advantages of the algorithmic techniques are evident, since they potentially provide a reliable, repeatable, and objective way to estimate fundamental parameters of a software development plan. Delivery dates, project budget, and so on can be derived from the measures computed by these algorithms.

Algorithmic techniques also have some limitations. The first is given by the models themselves, since they are derived from a limited (although growing) number of sample projects.

The second is given by the inputs to the models, namely, the values characterizing the system to build. These are not always easy to assess and might require significant analysis work, often performed by people with specific training and certifications.

It must be pointed out, however, that the analysis process required to come out with estimations has a value in itself, if not in the values it produces. In fact, the application of the algorithmic technique helps one to get a better understanding of the system to build.

Two big families of algorithmic techniques exist:

1. **Function-based** estimations measure a system in terms of its functions. The most well-known technique in this family is probably the **function point** (FP) technique. A more recent addition includes the **object point** technique, to mention one.

2. **Size-based** estimations measure a system in terms of its physical size, as well as measures in lines of code. The most well-known technique is probably the **constructive cost modeling** (COCOMO) family of models.

Finally, some techniques mix both function- and size-based approaches. This is the case, for instance, of **web-objects**, a technique to estimate web applications.

Several studies have been published on software estimations. One famous historical reference is Boehm (1981). An analysis of the existing literature, based on 304 papers and pointing to many resources, can be found in Jørgensen and Shepperd (2007).

3.4.5.1 Function Points

The FP technique was proposed by Albrecht in the 1970s (see, e.g., Albrecht (1979)). It finds its original application in business systems, but it has since evolved to embrace a wide range of systems and applications. Today, it has a large user base and an organization, the **International Function Points User Group**, to support its diffusion, application, and evolution (International function point user group, 2013c).

FP estimations are based on 19 different characteristics of a system, five of which refer to the functional characteristics of a system and 14 of which refer to nonfunctional aspects. These characteristics can typically be provided once the requirements of a system have been defined. The technique can, thus, be applied only after a part of the development process has started.

The five functional measures are

1. **User inputs:** the number of user inputs, or elements in the system which require an input from the user.
2. **User outputs:** the number of user outputs, or elements in the system which produce an output.
3. **User inquiries:** the number of user inputs that generate a software response, such as word count, search result, or software status.
4. **Internal logical files:** the number of files created and used dynamically by the system.
5. **External interfaces:** the number of external files that connect with the software to an external system. For instance, if the software communicates with a device, it is counted as one external interface.

The data above are collected and classified according to a three-step complexity scale, which distinguishes among simple, average, and complex elements. Thus, for each of the five characteristics described above, we need to produce a three-dimensional vector: number of **simple** elements, number of **average** elements, and number of **complex** elements.

The technique defines a matrix of weights to take into account the impact that different elements have in determining the complexity of a project. Each pair *functional characteristic*, *complexity* has a weight. Thus, for instance, simple user inputs have a weight of 3, average user inputs a weight of 4, and complex user inputs a weight of 5.

The functional size of a system S, called by the method **unadjusted function points** (*UFP*), is the weighted sum of the characteristics of the system.

It is computed as follows:

$$\text{UFP} = \sum_{i=1}^{5} \begin{bmatrix} k_i^S & k_i^A & k_i^C \end{bmatrix} \cdot \begin{bmatrix} n_i^S \\ n_i^A \\ n_i^C \end{bmatrix} \tag{3.10}$$

where the index i runs over the five characteristics (user inputs, user outputs, ...); the vector $[n_i^S, n_i^A, n_i^C]$ is the number of simple (^{S}), average (^{A}), and complex (^{C}) elements of type i that we forecast in S. Finally, $[k_i^S, k_i^A, k_i^C]$ is the weight assigned by the method to simple (^{S}), average (^{A}), and complex (^{C}) elements of type i.

UFP provides an estimation of the functional size of the system. Systems with a higher *UFP* are more complex to develop than systems with a lower *UFP*. However, *UFP* does not take into account the complexity deriving from nonfunctional characteristics of a system. For instance, high reliability systems are difficult to implement, requiring more effort. The method therefore introduces 14 questions that list various nonfunctional features of a system, which can positively or negatively affect a project. The questions are summarized in Table 3.6.

Similar to the functional evaluation, managers answer the 14 questions by providing, for each question, a qualitative answer ranging from 0 (irrelevant) to 5 (very influential).

The answers are summed together and added to the magic number 65, thus yielding a value in the range 65–135, where 135 is obtained when the answer to all the questions is 5, since $65 + 5 * 14 = 135$. The value, divided by 100, is called the **value adjustment factor**, or VAF, and measures the additional (or reduced) effort that is necessary to take care of the implementation of nonfunctional requirements.

Formally,

$$\text{VAF} = \frac{65 + \sum_{i=1}^{14} C_i}{100} \tag{3.11}$$

VAF is then multiplied by *UFP* to yield the function points (*FP*):

$$\text{FP} = \text{VAF} * \text{UFP} \tag{3.12}$$

FP measures the complexity of the system to be developed.

Table 3.6 Nonfunctional Characteristics of a System (FP Method)

1. Does the system require reliable backup and recovery?
2. Are data communications required?
3. Are there distributed processing functions?
4. Is performance critical?
5. Will the system run in an existing, heavily utilized operational environment?
6. Does the system require online data entry?
7. Does the online data entry require the input transaction to be built over multiple screens or operations?
8. Are the master files updated online?
9. Are the inputs, outputs, files, or inquiries complex?
10. Is the internal processing complex?
11. Is the code to be designed reusable?
12. Are conversion and installation included in the design?
13. Is the system designed for multiple installations in different organizations?
14. Is the application designed to facilitate change and ease of use by the user?

Once we have the estimation in function points, we can use it to estimate the effort required for the implementation of the system. There are two main approaches to map function points into effort. The first transforms the function points into a size measure (for which effort estimations can then be provided).

The second, which is preferred by the people advocating the use of function points, uses productivity metrics, such as **man-months per function point**, to estimate the effort required to develop a system. The actual values of productivity metrics depend on many factors, among which are team experience, organization maturity, and application fields.

Some example values can be found in Longstreet (2008), where work increases exponentially with a system's size. It starts at 1.3 h/FP for a system of 50 FP, continuing with 12.1 h/FP for a system of about 7000 FP, and ending with 133.6 h/FP for a system of about 15,000 FPs. See Longstreet (2008) for the complete set of data.

Organizations willing to use the technique, however, should establish their own measurement programs to determine their productivity metrics.

3.4.5.2 COCOMO

Constructive cost model (COCOMO) is a family of estimation techniques first introduced by Barry Boehm in the 1980s and steadily improved over the years. The method was defined in the context of a broader analysis of software economics, which focused on improving the capacity of reasoning about software development costs and benefits, value delivered, and quality (Boehm, 1984; Boehm and Sullivan, 2000). The techniques are very detailed and make assumptions about the

development process that is used to build a system. Moreover, different models are used during the development process to increase the accuracy of the estimations.

In this book, we will present the basic model, called COCOMO81, abstracting away many details and just hint about the more recent formulation of the model, called COCOMO II and introduced in 2000.

The COCOMO models use a size measure as the starting point for estimations and define a simple relationship between size and effort and duration of a project, which is captured by the following formula:

$$\text{OUTPUT} = A \cdot (\text{SIZE})^B \cdot M \tag{3.13}$$

that is, the *OUTPUT* of the estimation (effort and duration) depends on a system's *SIZE* and three other elements A, M, and B. In general, A and B are organizational-dependent constants, while M depends on the project at hand. Note that A and M have a multiplicative effect, while B has an exponential effect over the *size*. Reference values for A and B are given by the model.

3.4.5.2.1 COCOMO81

COCOMO81 is the first COCOMO model. It was defined by Barry Boehm and his group using a carefully screened sample of 63 projects developed between 1964 and 1979 (Boehm, 1981). The model applies to the software development practices in use by then, among which the use of the waterfall development process is probably the most relevant assumption.

The method defines three different variants, which can be applied at different stages of the development process as the information about a project increases. The **basic model** can be used when little information is available; the **intermediate model** can be used when the requirements are defined, and the **advanced model** can be used when the architecture is sketched.

In more detail, COCOMO81 computes effort and development time as follows:

$$\text{PM} = A_{\text{PM}} \cdot (\text{KSLOC})^{B_{\text{PM}}} \cdot M \tag{3.14}$$

$$\text{TDEV} = A_{\text{TDEV}} \cdot (\text{PM})^{B_{\text{TDEV}}} \tag{3.15}$$

$$\text{TEAM} = \text{PM}/\text{TDEV} \tag{3.16}$$

where *PM* is the total effort, *TDEV* is the ideal duration of the project, *TEAM* is the team size, and *KSLOC* is the estimated number of thousands of lines code. In all three variants, A and B are constants, while M (which is not required in the basic model) is computed by looking at various project and product characteristics.

The actual values of A and B depend on the overall complexity of the project. COCOMO81, in particular, distinguishes between three types of projects, **organic**, **semidetached**, and **embedded**, according to the project characteristics listed in Table 3.7. For each type of project, the model provides values for A_{PM}, A_{TDEV}, B_{PM}, and B_{TDEV}, as shown in Table 3.8.

Table 3.7 COCOMO Development Modes

Project Type	*Main Characteristics*
Organic	Simple projects with clear requirements and about which the performing organizations have a thorough understanding and experience. No or few technical and development risks.
Semidetac head	More complex projects. The performing organization has considerable know-how in the application field and with tools to be used for development. Some technical or development risks.
Embedded	Complex system with high variability in requirements. The organization has moderate know-how in the area. Various technical and development risks.

Table 3.8 COCOMO Base Model

	APM	*BPM*	*ATDEV*	*BTDEV*
Organic	2.40	1.05	2.50	0.38
Semidetached	3.00	1.12	2.50	0.35
Embedded	3.60	1.20	2.50	0.32

Finally, note that the formula to determine the team size is the same as in Equation 3.6.

The intermediate model takes into account various project- and product-related characteristics, which can contribute positively or negatively to the overall effort and to the project schedule. This is reflected by computing M as the product of 15 different parameters obtained by answering 15 different questions with a value from 1 (very low impact) to 6 (extremely high impact). Different from the FP method, each evaluation corresponds to a numerical constant, with values ranging from a minimum of 0.75 to a maximum of 1.56.

Tables 3.9 and 3.10, for instance, show the values assigned to the parameter RELY (required software reliability) and the corresponding assignment criteria. If RELY is evaluated as very low, for instance, the corresponding value to be used for the computation of M is 0.75, yielding a reduction in the effort of 25%.

The parameters considered for the M factor can be organized in four different classes. Two of them describe the characteristics of the project outputs. They are

1. **Product attributes**, which model aspects related to the software to be developed and include aspects such as expected reliability, database size, and overall product complexity

Table 3.9 COCOMO RELY Parameter

	Very Low	*Low*	*Nominal*	*High*	*Very High*	*Extremely High*
Required Software Reliability (RELY)	0.75	0.88	1	1.15	1.4	–

Table 3.10 Explanation of the COCOMO RELY Parameter

Very low	The effect of a software failure is simply the inconvenience incumbent on the developers to fix the fault.
Low	The effect of a software failure is a low level, easily recoverable loss to users.
Nominal	The effect of a software failure is a moderate loss to users, but a situation for which one can recover without extreme penalty.
High	The effect of a software failure can be a major financial loss or a massive human inconvenience.
Very high	The effect of a software failure can be the loss of human life.
Extremely high	No rating—defaults to very high.

2. **Computer attributes**, which take into account aspects related to the platform that will be used to run the software and include aspects such as constraints related to performance and stability.

The remaining two classes describe project attributes. They are

1. **Personnel attributes**, which model the influence of the personnel involved in the project and include five parameters related to the capability and experience of the personnel
2. **Project attributes**, which model some aspects related to the project organization and include three parameters describing tool support and automation and schedule constraints.

3.4.5.2.2 COCOMO II

COCOMO II significantly revises and enhances COCOMO81 to take into account several new factors that intervened after the first definition of the model. Among them are new development processes, new development paradigms (e.g., object orientation), and new development techniques (e.g., code reuse). The model also takes advantage of an enlarged set of project data, which is based on 161 projects in place of the 63 used in the definition of COCOMO81 and an improved definition of the term "source lines of code." Finally, COCOMO II uses the spiral development process as its reference process.

COCOMO II introduces three main changes:

1. UFPs are used at an early stage of the development process to determine a system's size. The UFP are then transformed into lines of code using translation tables that map UFP into SLOCs. Some example values can be found in Center for Software Engineering (2000) and Quantitative Software Management (2013). This helps solve the "chicken–egg" problem with the original definition of the model. In fact, the accuracy of the model depends on the accuracy of the estimation of the system size, which however is known only when development ends.

2. The computation of lines of code is adjusted to take into account reused code and requirements volatility. The first is computed using a nonlinear model derived by analyzing about 3000 projects from NASA. The latter simply increases the count of SLOC by a percentage that represents the number of requirements that will change. This allows one to use the method with more modern programming practices.
3. The parameters are refined or updated. In detail, more the exponent is computed as the sum of five scale factors. The scale factor includes aspects related to the development process. They are assessed similar to the effort adjustment factors; the result is always between 0.91 and 1.226. Finally, the other parameters are updated to match analyses conducted on a larger set of data.

The Center for Software Engineering (2000) and Merlo-Schett et al. (2002) give more information about the application, while the University of Southern California (2013) and NPS (2013) make available an online calculator.

We conclude the section on COCOMO by mentioning that various extensions have been proposed to the model, among which are COQUALMO, to estimate software defects, COCOTS, to estimate integration of COTS component, and COSYSMO for system engineering.

3.4.5.3 Web Objects

Web object is a technique that mixes FP analysis and COCOMO models to estimate the effort and schedule of web application development. The main motivations for the definition of (yet another) estimation technique are some fundamental differences between desktop and web applications development.

In fact, the development of web applications tends to be driven by time, rather, that costs; it prefers more informal (and speedy) processes; it uses smaller teams (3–6 people), often composed by younger and less experienced personnel. According to Reifer, who defined and proposed the model, these motivations make the application of other techniques less effective (see, e.g., Reifer (2000), Ruhe et al. (2003)).

The method is organized in two phases, **Web objects** and **web modeling**. The first, similar to the FP estimation, is used to estimate a web application size. The application of the technique requires one to measure nine different characteristics of a system, classifying them as simple, average, or complex. Five of these nine characteristics are those we already saw in the FP estimation. Four new elements are specific to the web application domain and they include **multimedia files**, **scripts**, **links**, and **web building blocks**. Using a weighted count of the characteristics yields the number of **web objects**, that is, a measure of a system's size.

The second part of the method, **web modeling**, is structured similarly to the COCOMO method and it transforms the number of web objects into effort and schedule. The relationship between web objects and effort is given by the following formulas:

$$\text{Effort} = A \cdot \prod_{i=1}^{9} cd_i (\text{Size})^{P_1} \tag{3.17}$$

$$\text{Duration} = B \cdot (\text{Effort})^{P_2} \tag{3.18}$$

where A, B, P_1, and P_2 are constants (similar to COCOMO81), *Size* is the size in web objects, and cd_i are nine cost drivers similar to those defined by the COCOMO model.

3.4.5.4 Effort and Project Phases

FPs and COCOMO provide top-down estimations. As we have seen, to distribute the total effort and duration to lower elements of the WBS, we need to select an appropriate approach. COCOMO provides reference tables that break down effort and schedule for different software development processes. The breakdown is given in terms of percentages and ranges of percentages and can be found in Boehm (1981).

Thus, for instance, for a small project, the effort computed using the COCOMO81 model can be distributed as follows: 21% to *plan and requirements and product design*; 26% to *detailed design*; 42% to *coding* and *unit testing*; 16% to *integration* and *testing*.*

See Boehm et al. (2000) and Boehm (1984) for more information and Yang et al. (2008) for a critical study related to phase distribution in various projects.

3.5 Scheduling a Plan

WBS identifies the work that it is necessary to carry out, but it does not show any constraints between activities, nor does it specify anything about scheduling, that is, when each activity should start and how long it should last. This is exactly what we are going to do in this section.

Scheduling the plan is composed of the following steps:

1. **Identify dependencies among activities**. During this step, we highlight the dependencies in our project to understand the degrees of freedom we have in scheduling our project. Some activities will have no dependencies and we will be able to schedule them more freely. Others will depend on tasks to finish (or to start) before they can be started; for these, we will clearly have less options.

* Note that the sum of percentages is 106%; this is because the planning phase is outside of the scope of the COCOMO81 computation; analysis of effort and schedule distribution, however, allowed the planning phase to be estimated as an additional 6% on top of the values provided by the model.

2. **Identify the critical path of the plan.** The goal of this activity is to identify the most critical activities in the plan. These are the activities that, if delayed, will delay the plan.
3. **Allocate resources to tasks and level resources.** The goal of this activity is to allocate actual resources to the different activities. During this step, various additional constraints emerge, due to the availability of resources and the maximum amount of work that can be allocated to each resource. The process of dealing with such constraints is called **resource leveling**. The output is a plan that introduces additional constraints, called *soft constraints*, which ensure that the limitations related to resource availability are actually met.

Note that the order in which we listed the activities above is merely for presentation purposes. In practice, there is a lot of freedom in the way in which these steps are executed. In many cases, schedules are constructed by looking at the different concerns in parallel, trying different scenarios. In the current practice, the use of a modern Gantt charting tool integrates the steps above, promoting a process in which the schedule is built by looking at all these concerns in parallel.

3.5.1 Identify Dependencies among Activities

No one will start building a house from the roof. Thus, the first step to scheduling our plan is to identify the order in which the activities can be executed. This is done by identifying the dependencies among activities. In general, a **dependency** between two activities *A* and *B* defines some kind of constraint in the executability of *A* and *B* and imposes a partial ordering on the execution of the activities.

The dependencies can be characterized according to different dimensions, as illustrated in the following paragraphs.

3.5.1.1 Type of Dependencies

Four types of dependencies can be set between two activities, according to whether the constraints involve the start or the end of the two activities.

Two types of constraints are relatively common. These are

1. **Finish to start** (FS). An FS constraint between *A* and *B* expresses the fact that *B* can start only after *A* is finished. This is probably the most common constraint between activities. For instance, the activity "baking a cake" can start only when all the ingredients have been poured into the baking pot.
2. **Start to start** (SS). An SS constraint between *A* and *B* expresses the fact that *B* can start only when *A* starts. For instance, a "monitoring" activity can start only when the activity that is being monitored starts.

Two types of constraints are used and found less often. These are

1. **Finish to finish** (FF). An FF constraint between *A* and *B* describes a situation in which *B* can finish only when *A* finishes.
2. **Start to finish** (SF). An SF constraint between *A* and *B* describes a situation in which *B* can finish only when *A* starts.

Another classification distinguishes between hard and soft constraints. A **hard constraint** between two activities *A* and *B* models a dependency that is in the nature of the work to be performed. A hard constraint cannot be broken without violating the logic of the project. An exception is fast tracking, which optimizes a plan by breaking hard constraints, at the cost of a riskier project execution—see Section 3.6 for more details.

Vice versa, a **soft constraint** between two activities *A* and *B* can be set as a convenience to simplify project execution or to reduce risks in the project execution. A soft constraint can be broken without violating the logic of the project.

An example can be of help. In the preparation of a meal, a hard constraint exists between the preparation of the ingredients and cooking them. Cooking, in fact, cannot start if the ingredients have not been prepared. The constraint is in the logic of the activities and there is no way to break it, unless you decide to be extremely creative in your cooking.

In the same scenario, we could decide to impose a soft constraint between preparing the dessert and preparing the main course, that is, arbitrarily decide that we will start preparing the dessert and then move on to prepare the main course. No dependency between the two activities exists. In principle, we could even prepare both dishes in parallel, if we wanted to. Soft constraints can be broken, if required, by changing the hypotheses for which the links were introduced in the first place.

Remark

When people start using Gantt charting tools, sometimes they introduce dependencies between activities to schedule them in a specific order, even if there is no hard constraint between the activities.

The introduction of these soft constraints in a plan is a questionable practice, since it reduces the degrees of freedom one has in scheduling. Moreover, it can make rescheduling and evaluating alternatives a lot more complex when it becomes difficult to distinguish between the hard and the soft constraints.

The use of task priorities, resource leveling, and scheduling constraints are more effective means to achieve the same goal. See Section 3.5.3 for more details.

3.5.1.2 Lead and Lag Time

When defining dependencies between two activities, sometimes it is convenient to specify a time interval, positive or negative, that occurs between the activities. We speak of **lag time** if the time interval is positive. We use the term **lead time** if the interval is negative. Thus, for instance, if *A* and *B* are connected by an FS dependency with a lag time of 3 days, it means that *B* can start three days after *A* has finished. In the example above, if the FS dependency had a lead time of 3 days, *B* could start 3 days before *A* ends.

A typical usage of lag time is with SS constraints, when some progress in the first activity is necessary to start the second one. For instance, in a roadwork project, an SS constraint with a lead time of one or two weeks could be set between *digging* and *laying pipes*. The lag time, in fact, is to allow the *digging* to progress sufficiently to actually make the *laying pipes* activity doable.

Similar to the identification of constraints, it is good practice to introduce lead and lag times between activities only if strictly imposed by the logic of the plan. Introducing arbitrary constraints reduces the degrees of freedom and it could make scheduling a lot more complex than needed.

3.5.1.3 Network Graphs

The dependencies among activities in a project can be represented using a table. Each row contains an activity, identified by a unique number. The dependencies are shown as a list of identifiers, possibly followed by the type of dependency, if different from FS, and the lead and lag times, if present.

Table 3.11 shows an example in which activity C depends on activity A with an FS dependency; C also depends on activity B with an SS dependency. Similarly, activity E depends on activity C with an FS dependency and on activity D with an FS dependency with a lag of three days (+3d).

A far more intuitive notation uses a graph, called **network diagram**, in which all the constraints between activities can be shown visually. Two kinds of network diagrams exist. Network diagrams that represent activities on the nodes are called **activity on node** (AON) diagrams. Network diagrams with activities on the edges are called **activity on arrow** (AOA) diagrams. Both notations were introduced in the 1950s. Of the two, the AON notation usually yields a more natural representation

Table 3.11 A Plan Specification

ID	*Activity*	*Dependency*
1	A	
2	B	
3	C	1, 2SS
4	D	
5	E	3FS + 3d, 4

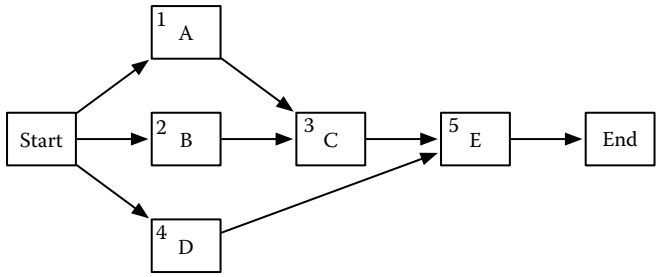

Figure 3.5 AON representation of the plan of Table 3.11.

of a plan, while the AOA notation, initially developed for PERT, is less used, now. Project Management Institute (2004) uses the term **precedence diagram** for AON and **arrow diagram** for AOA.

Figure 3.5 shows the AON representation of the plan presented in Table 3.11. Each activity is represented by a rectangle and arrows represent dependencies among activities. Two special activities Start and End represent, respectively, the start and the end of the project. A popular extension of the AON notation enriches the description of the nodes with duration, start date, and end date, as shown in Figure 3.6. The notation will be used for the computation of the critical path of a plan.

Figure 3.7 shows the AOA representation of the same plan. As can be seen from the figure, activities are shown on arrows and circles are used to represent dependencies. AOA diagrams might require the introduction of dummy activities to represent the dependencies correctly. Consider, for instance a plan with four activities, P, Q, R, and S, in which Q depends upon P and S depends upon R and P. The dependency between S and P requires the insertion of a dummy activity, represented with a dotted line in Figure 3.8.

3.5.2 Identify the Critical Path

In complex plans, the start or the end date of certain activities can be chosen or moved without affecting the overall schedule of the plan, that is, without affecting

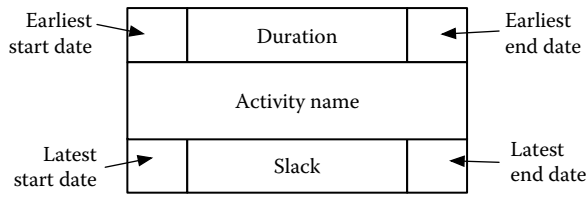

Figure 3.6 Adding information to a node.

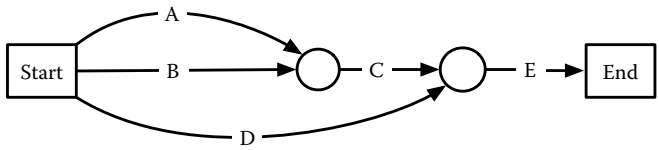

Figure 3.7 AOA representation of the plan of Table 3.11.

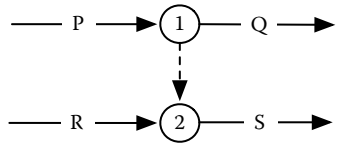

Figure 3.8 AOA dummy activity example.

the end date of the project. Consider the example shown in Figure 3.9, where activities A and B run in parallel, but A lasts longer than B. Note that Figure 3.9 extends the AON notation adding time and using the length of the boxes to represent the duration of the activities. Clearly, B, the shorter activity, can start up to four time units later than A (or, during project execution, even delay its finish date), without affecting any subsequent activity in the plan, as long as the delayed start or the extra duration does not move its end date after that of A. In fact, if it did, we would need to move the start of C, to respect the FS dependency between the two activities.

We call **free float** or **slack** the amount of time an activity can be delayed without affecting subsequent activities. In the example above, the slack of B is four time units, since we can delay its start up to four units without delaying the start of any of its successors (in the example, without delaying C). We call **total float** or **total slack** the amount of time an activity can be delayed without affecting the end of a project. In the example, A has a slack of 0 time units, but a total slack of 1 time unit, since C has a slack that can absorb a delay of A of 1 time unit, without moving E.

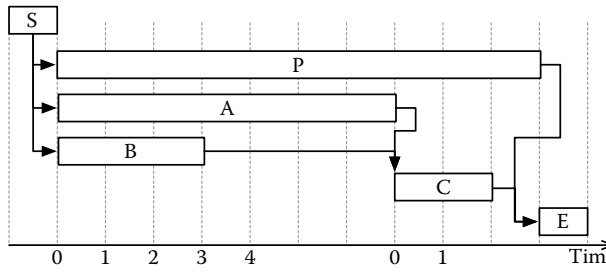

Figure 3.9 Computing the critical path.

The path in which all activities have zero total slack is called the **critical path of the plan**. Any delay in any activity in the critical path will cause a delay to the end date of the project, since the delay cannot be absorbed by any other activity in the path. Understanding what is the critical path of the plan allows a project manager to focus on those activities that are the most important to keep the project on schedule. Note that **all plans have a critical path**.

The critical path method is a technique developed in the 1960s that computes the critical path. It was developed to control the schedule of projects related to the development of the Polaris missile system. Today, the critical path computation is a basic feature of any decent Gantt charting tool. Understanding how the computation is performed, however, is interesting and useful. In particular, the method uses the AON representation of a plan and is performed by determining the earliest and the latest dates at which each activity can start (or end) without affecting the overall schedule.

The computation of the earliest and latest dates is performed in two passes:

1. A **forward pass** determines the **earliest start** and **earliest end** of each activity in the plan. The earliest start (end) date of an activity *A* is the earliest date at which we can start (finish) an activity, without breaking any dependency on the plan and without moving any other activity in the plan. Intuitively, it measures how soon we can start an activity.
2. A **backward pass** determines the **latest start** and **latest end** dates of each activity in the plan. The latest start (end) data of an activity *A* is the latest date at which we can start (finish) an activity, without delaying or moving any other activity in the plan. Intuitively, it measures how late we can start an activity without affecting the overall schedule.

The earliest and latest starts determine how much an activity can slide back and forth in a plan and are the basis for the computation of an activity's slack. The slack is in fact computed as the difference between the latest and earliest start of an activity or, equivalently, between the latest and earliest end. An activity whose latest start date (and, respectively, latest end date) is equal to its earliest start date (and, respectively, earliest end date) has a slack of zero.

The **forward pass** starts from the *start node* of the project and proceeds according to the following rules:

1. The start node of the plan is assigned the *earliest start* and *earliest end* date of zero.
2. For any other activity in the plan, the *earliest start* date is set to the highest *earliest end date* of its predecessors. The *earliest end date* is computed by adding to the *earliest start* the duration of the activity.

The calculation is best performed by analyzing the plan from left to right or, more precisely from the start node, then moving to its successors, till we reach the last

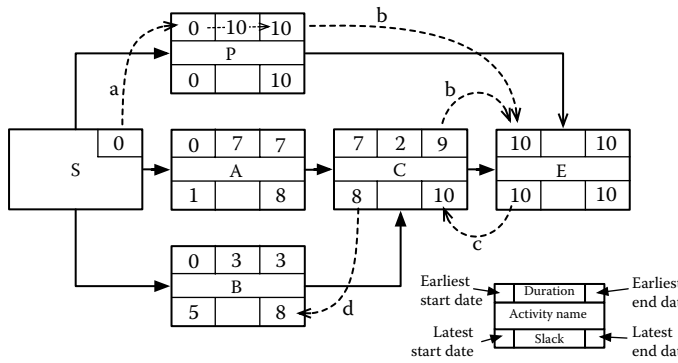

Figure 3.10 An example of critical path computation.

activity of the project.* Hence the name. Note that the *earliest end* of the plan is the *earliest end* of the last activity in the plan, that is, the *earliest end* of the end node.

When we finish the forward pass, we can start the **backward pass**. This starts from the *end node* of the plan and proceeds as follows:

- The *latest end* date of the end node is set to its *earliest end* date; the *latest start* date is set to the *latest end*.
- For any other activity in the plan, the *latest end* date is set to the lowest *latest start* of its successors, while the *latest start* is computed by subtracting the activity's duration from the *latest end*.

The computation of the backward pass is performed analyzing the plan from right to left or, more precisely, beginning from the end node, then moving to its predecessors, till we reach the start node of the project.† Hence the name.

Once we have computed the earliest and latest values of all the activities in the plan, we can determine the slack of each activity as explained before, namely, by subtracting the earliest start from the earliest end. The **critical path** of the plan is the one in which all activities have zero slack.

An example might help clarify the process. Consider the plan in Figure 3.9, for which we present the AON notation and the computation of the critical path in Figure 3.10. Note that each node uses the notation introduced in Figure 3.6, according to which the top boxes contain, respectively, the earliest start date, the duration, and the earliest end date; the bottom boxes contain, respectively, the latest start date, the slack, and the latest end date.

The handwritten numbers are the results of the computations to find the critical path, while the dotted lines show the process.

* Technically: breadth-first visit of the AON starting from the start node of the project.
† Technically: breadth-first visit of the AON, starting from the end node.

We begin from the start node, S: we assign an earliest start date of 0 to each activity depending exclusively on the start node. The earliest end dates are then computed adding the duration to the earliest start date. See, for instance, the computation for P—dotted line marked "a" in Figure 3.10. If an activity has more than one predecessor, we take the highest earliest end date. Thus, in the example, the earliest start date of the end node (marked E in the diagram) is 10. See the dotted lines marked "b" in the diagram. We then repeat the process till we compute the earliest start and end dates of each activity.

For the end node, the earliest end, latest start, and latest end are all set to the earliest start.

We can now start the backward pass, propagating the latest dates from the end nodes to the start. In our plan, for instance, the latest end date of activity C is 10, namely, the latest start date of the only successor of C. See the dotted line marked "c" in Figure 3.10. The latest start date of an activity is given by subtracting the duration from its latest end date. This explains why the latest start date of C is 8, that is, the result of subtracting 2, the duration of C, from its latest end date, which is 10.

Once we complete the backward pass, we are now ready to compute the slack of the different activities. Thus, for instance, the slack of A is 1, while the slack of B is 5. There is only one path having all activities with zero slack. This is the path made by just one activity, namely, P. So the critical path of the plan is composed by P. Note that this is a peculiar case; in general, more than one activity will be in the critical path. We now know that any delay to P will cause the entire project to deliver late. During the execution phase, particular attention has to be dedicated to the activity.

3.5.3 Allocate and Level Resources

So far, we determined the logic of the plan in terms of the activities necessary to achieve the project's goals, their dependencies, and the resources needed to carry out each activity. In Section 3.4, we have also seen the relationship between duration, effort, and resources. In many projects, the availability of resources is one of the most critical constraints; activities, in fact, can be carried out only when the required resources are available and at the speed determined by their availability. In this section, therefore, we look in more detail at the process of allocating resources to the plan. This will allow us to determine the duration of the project.

There are basically three constraints we need to satisfy:

1. The allocation has to comply with the estimation of the activities. If an activity requires an effort of 404 man-hours, we need to allocate sufficient resources or time to cover the effort.
2. The allocation needs to comply with the availability of resources. If a resource is available 4 h per day, we cannot create a plan in which he or she is supposed to work above that limit.
3. The constraints of the tasks must be satisfied.

This is taken care of by a three-step process composed of the following steps:

1. Qualify the resources needed for each task.
2. Verify the resources available.
3. Allocate the resources satisfying the constraints.

3.5.3.1 Qualifying the Resources Needed for a Task

As mentioned earlier, tasks require manpower, equipment, or material. Before a resource allocation can start, of course, these needs require to be made explicit.

In more detail:

- For **manpower**, we need to specify the effort necessary to carry out the activity, possibly organized per type of resource, if this is necessary.
- For **equipment**, the number of units required by each activity. For instance, the final testing activity of a software to control a robot could require the availability of two robots for 10 days.
- For **material**, the **quantity** necessary to carry out an activity. Software development plans rarely require the specification of material.

A simplifying assumption that is often made when planning is that the need for a specific resource in a task is uniformly distributed. For example, if a task requires 40 h of a designer over a period of 2 weeks, we assume a constant need of 20 h per week. Although strategies exist to deal with specific cases (e.g., a resource is available every second week; an activity requires an effort that ramps up at the beginning of the activities and slowly decreases at the end of the activity), the extra effort necessary to model such cases is usually not worth the advantages we can get.

Remark

Most entry-level planning tools allow a project manager to specify resource needs in terms of the total effort required in an activity, without distinguishing among the competences or types of resources needed.

In such situations, it is the responsibility of the project manager to allocate the actual resources in a way that is compatible with the requirements of the plan.

3.5.3.2 Specifying Resource Availability

For material and equipment, availability is expressed with quantities and units. For instance, 100 km of optic fiber; four excavators.

For manpower, resource availability is expressed as the total units of work we can allocate to our project. As we have seen, these data can be expressed, person by person, in terms of the percentage of availability. Thus, for instance, if we have one resource available at 80%, it means that he or she can allocate 32 work-hours per week.

A complex plan might abstract away the availability of individual employees. In this case, availability of resources is expressed in the form of a percentage greater than 100%. Thus, an availability of 800% means that we have the equivalent of eight people; it could be eight people full time or, maybe, six full time and four at 50%. The term **full-time equivalent** (FTE) is used to specify the availability of a person working full-time. Thus, for instance, three FTEs correspond to the availability of three people full-time or 300%.

Another aspect, to consider for resource availability are holidays and other leaves. A detailed plan will take into account such data, by specifying the nonworking days person by person. Higher-level plans typically take into account leaves by lowering the amount of work that can be performed by each resource. Thus, for instance, the maximum effort which can be expressed in a calendar-week could be set to 4.8 man-days, to take holidays into account.

Finally, note that the maximum effort available for a project is often a *theoretical* value. The actual effort a resource will be able to allocate to a task during a typical working day is much lower. In fact, we need to consider all other activities (phone calls, meetings, interruptions, breaks) occurring in a typical working day and taking time from the total availability. According to Wysocki (2011), the actual availability of resources is between 50% and 80% of the theoretical value.

EXAMPLE 3.4

In a project, we can count on the following resources:

- Dominique, a designer, who will work full-time.
- Rick, another designer, who is involved part-time, at 50%.
- Elva, an analyst, who works part-time on the project, at 50%.
- Giannetta, another analyst, who works part-time on the project, at 50%.

Given the data above, we can say that

- Dominique will be available 40 h per week.
- Rick will be available 20 h per week.

Concerning the availability of an analyst, we have two resources at 50% or, equivalently, 1 FTE.

Remark

For small/medium projects, mentally transforming percentages in actual days at the office is a good way to picture the actual involvement of a resource. For instance, a resource at 20% of his time will be able to work on the project 1 day per week.

From the previous example, it should also become clear that different types of involvements are not equally effective, due to considerations similar to those we already considered in Section 3.4. In particular, the lower the percentage of involvement, the higher the incidence of the time required to get into the task (for instance, catching up with work performed by the rest of the team).

3.5.3.3 Allocating Resources to a Plan

The third step consists in assigning resources to tasks. For manpower, this is done by allocating a percentage of a resource to the tasks.

Given an allocation of resources to a plan, a resource usage profile can be determined. The **resource usage profile** is a graph (or bar-chart) that depicts the number of hours of a resource dedicated to a given project. This is determined by summing up the hours dedicated to each activity to which the resource is dedicated in a given period.

Thus, for instance, if a resource full-time (40 h per week) in January works on two tasks, the first at 50% and the second at 25%, the resource profile in January will show 75% or 30 h per week; 20 h derive from the first task and another 10 come from the involvement in the second task.

The allocation of resources to a plan has two main effects: the first is that determines the duration of activities that have been estimated using effort. This results from the application of Equation 3.6. The second is that it introduces soft constraints in the plan. These are due to the fact that we cannot overallocate resources, that is, use resources over their maximum availability. Thus, activities that in principle could run in parallel will be sequenced if constraints over the availability of resources prevent us from doing so.

Resource leveling is the process of introducing soft constraints in a plan to ensure that no resource is overallocated. Some tools have resource leveling algorithms; others require the manager to do the job.

A resource leveling algorithm typically requires one to specify additional information for each task in the plan.

This includes:

- The **priority** of tasks, typically expressed in the form of a number. The priority determines which activities have to be scheduled earlier in case of conflicts with resource allocation. Typically, higher-priority activities are scheduled earlier.
- The **scheduling constraint** of tasks, which imposes limitations on the possible start and end dates. These are
  - **As soon as possible**, if the task has to be started at the earliest possible date, given the fact that any other constraint is satisfied. This is typically the default for planning tools and corresponds to an aggressive approach, in which we try and get done with the project as early as possible.
  - **As late as possible**, if the task has to be started at the latest possible date, given any other constraint is satisfied. This corresponds to a cautious approach, in which activities, and more important expenditures, are delayed till the very last moment.
  - **Must start on**, if the task has to start on a specific date.
  - **Must finish on**, if the task has to end on a specific date.

- **Start no earlier than**, if the task cannot start earlier than a specific date, but it is perfectly fine if it starts later than the set date.
- **Start no later than**, if the task has to start no later than a given date, but it is perfectly fine if it starts earlier than the set date.
- **End no earlier than**, if the task cannot finish earlier than a specific date, but it is perfectly fine if it ENDS later than the set date
- **Finish no later than**, if the task cannot finish later than a given date, but it is perfectly fine if it ends before the set date.

A resource leveling algorithm schedules activities ensuring that all constraints (dependencies among activities), the properties set by the project manager (priorities and scheduling constraints), resource availability, and priorities are satisfied. If the algorithm succeeds, all activities are laid out so that they satisfy the constraints.

A resource leveling algorithm can also fail. This is the case when the constraints are too tight. Consider the case of a project that must finish on a date but does not have enough resources to meet the deadline. In such cases, one or more constraints have to be relaxed. The most common strategy is adding more resources to shorten some activities. However, this does not necessarily make a project faster if coordination becomes too much of a problem, as was highlighted in Brooks (1995). Other techniques include renegotiating the project scope or compressing the schedule, as we will see in Section 3.6.

3.5.4 The Gantt Chart

The Gantt chart is a very popular notation that can be used to present schedules. Henry Gantt first introduced the notation in 1917 to control shipbuilding works.* The notation we use today is an extension of the original work, which also allows one to represent the WBS, dependencies among activities, the critical path, and various other information about tasks. Many Gantt charting tools exist, and the activities we describe in this section are often carried out interactively using these tools.

The (modern) Gantt chart notation (called a logical network in Burke (2006)) is shown in Figure 3.11. It is organized in two main parts. The left-hand side of the figure contains the list of activities, together with the start and end dates of each activity. The list can present the activities using an outline structure to highlight the hierarchical nature of the plan. The right-hand side of the chart shows the calendar time and the activities.

In particular, on the right-hand side

- Activities are laid on the calendar as rectangles. The positioning and size of the rectangle shows the start date, duration, and end date of the activity: the

* The interested reader can find the description of the original work in Clark and Gantt (1923).

	Task Name	Start	Finish
1	System development	21/05/2012	19/06/2012
2	Sprint 1	21/05/2012	25/05/2012
3	Proto 1 (client + server)	25/05/2012	25/05/2012
4	Comments	28/05/2012	01/06/2012
5	Comments on proto 1	01/06/2012	01/06/2012
6	Sprint 2	28/05/2012	04/06/2012
7	Proto 2 (client + server)	04/06/2012	04/06/2012
8	Comments	05/06/2012	08/06/2012
9	Comments on proto 2	08/06/2012	08/06/2012
10	Sprint 3	05/06/2012	08/06/2012
11	Proto 3	08/06/2012	08/06/2012
12	Sprint 4 and deployment	11/06/2012	19/06/2012
13	System	19/06/2012	19/06/2012

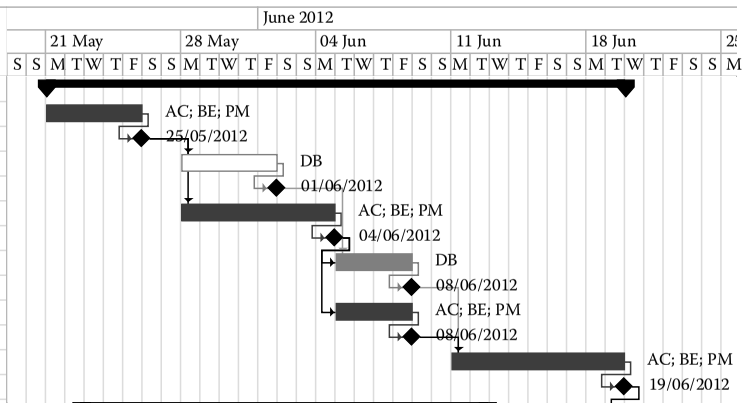

Figure 3.11 An example of a Gantt chart.

left side of the activity corresponds to the start date; similarly for the right side of the activity. Activities are labeled with the initials of the people allocated to the activity.*
- Deliverables are presented as diamonds. They have zero duration and are labeled with the date before which they are to be produced.
- The dependencies among activities are marked with arrows starting from an activity (deliverable) and ending at another activity (deliverable). The starting and ending places of the arrow determine the constraint type. Thus, for instance, an arrow starting from the end (finish) of an activity and pointing to the start of another activity represents an FS constraint. Similarly for other kind of constraints.
- The black lines with triangles at the ends represent work packages, grouping sets of activities.

3.6 Optimizing a Plan

In many situations, the project schedule ends up by being too long to respect the constraints set by the stakeholders, by the project goals, or by the environment. This can cause a bit of frustration to the project manager, since, as soon as he or she comes out with a realistic plan, this has to be revised and changed!

In the following, we analyze the most common techniques to compress the schedule of a plan so that it meets the customer's needs.

3.6.1 Renegotiating Goals and Deadlines

If all the project goals cannot be achieved in the required time frame, renegotiating the project scope and other project constraints can yield a satisfactory solution.

The simplest renegotiation we can try is on the **delivery date**. If the customer does not have a strong constraint on the delivery date, acknowledging the actual work that has to be done and moving the project delivery date to a more reasonable deadline is a simple and elegant solution. The actual feasibility varies. Sometimes deadlines are set arbitrarily by the customer. In these situations, using the plan to demonstrate that the deadline cannot be achieved can convince the customer to come to more reasonable terms. In other situations, deadlines are set earlier than necessary, as a padding to protect other projects that might depend on our results. In these cases, understanding the actual margins and the real risks of delivering late can help both sides decide on the most appropriate strategy. In the remaining cases, the deadline cannot be moved. In this situation, we need to use another technique.

The second kind of renegotiation is on the **project goals**. Not all goals, in fact, are equally important. We have seen in Section 3.2 how we can assign a priority to

* This is the default adopted by many tools, but it is by no means a standard.

different goals. Selecting with the client the most important goals reduces the work we need to do, moving the delivery to an earlier date.

If both approaches are not feasible, we need to change the logic of the plan. This is what we discuss in the next few sections.

3.6.2 Phase the Project

Organizing the project in phases allows one to organize work so that the most important goals are achieved earlier. If only some of the project goals must be achieved for a given deadline, phasing the project might help meet the requirement.

For software development projects, this can be an effective strategy, since software development accommodates relatively easily an incremental construction. Using this approach, the first phases will release an initial version of the system with basic features. The system will then be refined in subsequent project iterations. An additional advantage of this approach is that the user is given a working solution to use: this allows both users and the development team to better understand what functions are important and, consequently, how to prioritize project development.

3.6.3 Project Crashing

Project crashing is a technique that works on the project schedule trying to find an optimal balance between time and costs. Project crashing works on the assumption that shortening a project yields savings and that the duration of tasks can be reduced by assigning more resources to them (labor, material, equipment). However, since an increment in resources causes an increase in project costs, which could be nonlinear with the decrease in duration, an optimal balance needs to be found between how much a project is shortened and how much the costs are increased.

When using project crashing, the

- **Crash costs** indicate the savings obtained by crashing the project.
- **Crash time** indicates the time used to shorten the project.

Project crashing can be an effective method to optimize costs. The reader, however, should be aware that the technique might be difficult to apply effectively. Consider, for instance, the additional risks introduced by crashing a project. They could, in principle, cause additional rework and delays to a project that we tried to shorten instead.

EXAMPLE 3.5

Consider a project that is late and scheduled to end 4 months later than the delivery date agreed with the client. Each month of delay costs us €20K, as per the contractual agreement with the client. Thus, with the current plan, we will lose €80K. To try and recover the situation, Cathy, the project manager, has analyzed the costs to shorten the project. According to her data, reducing the duration by 1 month is relatively easy and cheap, since we can use internal personnel. However, any further

Table 3.12 Crashing costs

Crashing	*Overrun*	*Crashing Costs*
4 months	0 month	€120,000
3 months	1 month	€90,000
2 months	2 months	€60,000
1 month	3 months	€10,000
0 month	4 months	€0

shortening will require us to hire expensive consultants. The actual estimations of the costs are shown in Table 3.12.

Project crashing can be used to decide what is the optimal crashing time. This can be done by computing the crashing costs per month, which include the expenses to crash the project (data of Table 3.12) and the penalty we pay for delivering late. The data are shown in Table 3.13, where we report the costs we incur for delivering late (column "Overrun Costs"), those we incur for crashing the project ("Crashing Costs"), and the total costs, given by the sum of the previous two values.

We can now determine the optimal crashing cost, which is given by the minimum value in the "Total Costs" column. This is shown in Figure 3.12, from which we can easily see that the optimal crashing time is 1 month. In this situation, we will lose €70K, saving €10K with respect to the situation in which we do not crash the project. Any other arrangement will result in incurring higher costs.

If costs are the main or the only parameter for choosing how much the project has to be crashed, then the answer is 1 month, that is, we reduce the duration of the project by 1 month.

3.6.4 Fast Tracking

Fast tracking tries to minimize the project duration by breaking the logic of the plan. That is, some of the hard constraints in the plan are removed so that activities that would otherwise be sequential can partially overlap.

Figure 3.13 shows an example where some activities that depended on a deliverable are actually started earlier, by breaking the dependency. This allows one to end the project earlier and achieve the project constraints.

Table 3.13 Crashing Example

Crashing	*Overrun*	*Crashing Costs*	*Overrun Costs*	*Total Costs*
4 months	0 month	€120,000	€0	€120,000
3 months	1 month	€90,000	€20,000	€110,000
2 months	2 months	€60,000	€40,000	€100,000
1 month	3 months	€10,000	€60,000	€70,000
0 month	4 months	€0	€60,000	€80,000

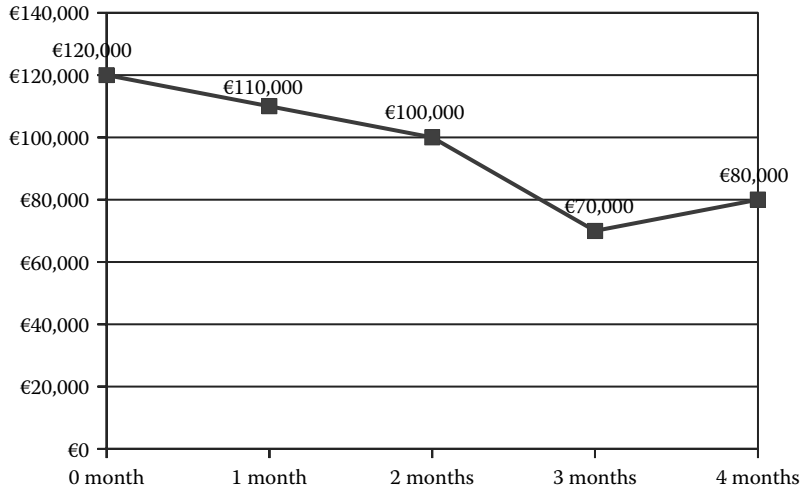

Figure 3.12 Crashing the project: Total costs over time.

Fast tracking is not odd as it might seem at first glance, if we think about the incremental nature of the work performed on this task. Work progresses in a nonlinear fashion. Thus, in many practical situations, little work and progress will remain close to the end of an activity.* Dependent activities, can thus, start on the partial results that are achieved as predecessor activity progresses, resulting in a more compact schedule.

The main risk with fast tracking is that rework might be necessary. Consider the case of writing the specification of a function and the subsequent activities related

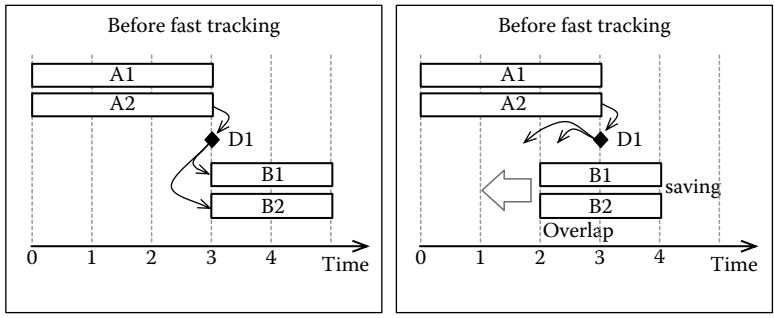

Figure 3.13 An example of fast tracking.

* See Section 3.6.5 for another way of looking at this issue.

to implement it. If we fast track, we start design and implementation before the specification is fully described. If important information is added after we start the implementation, we might end up implementing the wrong functions and having to redo implementation work.

Deciding what chains of activities are best suited for fast tracking is a tricky issue that depends on the tasks at hand and the project manager and team judgment. Some rules of thumb include looking at activities that will produce (stable) intermediate results and activities whose deliverables can be broken into independent pieces of work. For instance, a requirements writing activity could be fast tracked if the requirements can be organized in different and independent sections and the implementation of software can be started as each section is produced. If there are functional interdependencies, a little rework might be necessary.

Another item to consider is how rework could affect other components of the plan. One risk is propagating delays; another is producing outputs of low quality; a third is increasing the project costs.

3.6.5 Critical Chain Management

Critical chain project management is a technique developed at the end of the 1990s that has been successfully applied in many real projects. The technique is fairly complex and the presentation we give here is a rather significant simplification of the overall process. See Wysocki (2011) for a very nice introduction to the technique and Goldratt (1997) and Stratton (2009) for additional references.

Critical chain management starts from the assumption that estimations are random variables, as we have discussed in Section 3.4.3. Thus, if our best guess for the duration of an activity is n days, what we are really saying is that we *expect* the activity will take n days to complete. However, the activity might take longer or finish earlier. If we assume the probability distribution of the duration to be symmetric, half of the time the actual duration will be shorter than expected and half of the time will be longer.

If we take a cautious estimation, that is, an estimation that is above our best guess, *most of the time* the actual duration will be equal to or less than our cautious estimation. For instance, if we estimate the duration to be $n + \sigma_n$, where σ_n is the variance of n, the probability of the duration being lower than $n + \sigma_n$ is about 84% for normal distributions.

If we consider a sequence of activities, the guesses add up. If we consider the estimation of each activity to be independent, however, probability theory tells us that the variance of the sum of the durations is lower than the sum of the variances. In other words, the cautious estimation of the chain of activities is lower than the sum of the cautious estimations of each activity in the chain.

This is shown in Figure 3.14, where we have a sequence of two activities, A and B, for which we have provided two cautious estimations. That is, the duration we have chosen is above the mean value, as shown in the upmost diagram, by having

chosen a duration right of the mean, resulting in some "padding." We can also separate, for each activity, the best guesses from cautious estimations, moving them to the end of the chain, as shown in the center part of Figure 3.14. However, if we consider the estimation of the sum of the activities, its probability distribution will have a lower variance. Even with a cautious estimation, therefore, the duration of A + B will be lower than the sum of the cautious estimations of A and B.

This is what critical chain management does: it uses best guesses to estimate each activity (rather cautious estimations) and adds cautious estimations at the ends of the chains, rather than at the end of each activity. This results in two savings. First of all, for long chains, we can expect some activities to last more and some to last less than expected: delay might be compensated for by early deliveries. The second is that the padding added at the ends of the chains is lower than that computed for each activity. When using critical chain management (CCM), therefore, we consider the chain of activities, which are estimated at their best guesses, and **contingency buffers**, which contain the *padding* (cautious estimation), as shown in the third diagram of Figure 3.14.

There are other principles that make CCM effective; one of them is that resources are allocated *greedily*, so that we can exploit any saving deriving from an activity finishing earlier than expected.

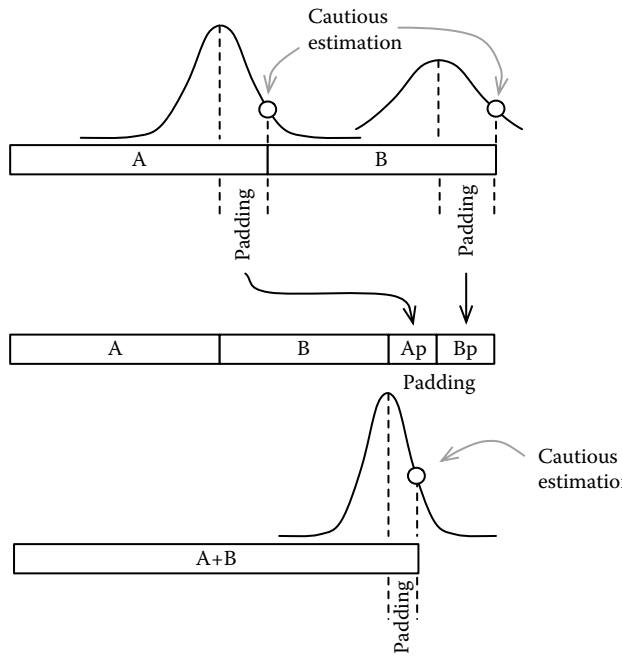

Figure 3.14 Critical chain management.

Concerning plan monitoring and management of delays, CCM assesses the execution of chains at one-third, two-thirds, and three-thirds (at the end) of their execution. In more detail, if delays were distributed uniformly during the execution of the chain, we should expect each third of the chain to delay the chain, at most, by one-third of the contingency buffer. Any worse and the chain might end late; any better and we can exploit early deliveries. Another essential element of critical chain management, therefore, is to place the contingency buffers so that they can *absorb* delays without compromising the plan. For this reason, the technique uses three types of buffers, namely, **project buffers** that protect the plan, **feeding buffers** that protect the chain of activities, and **resource buffers** that protect the plan from delays in resource availability. See Wysocki (2011) and Goldratt (1997) for more details.

3.7 Budgeting and Accounting

As pointed out in the introduction, the project manager is tasked with keeping under control three of the main aspects of a project: quality, time, and cost. This section is an introduction to the main techniques to determine the cost and price of a project and managing a project's cost over the duration of the project. Whenever possible, we instantiate the concepts to the software development domain.

3.7.1 Project Costs

Project costs are the expenses that an organization will incur into carrying out a project.

The items contributing to the expenses can be divided into in **direct costs** and **indirect costs**.

Direct costs are the expenses directly related to carrying out a project. These include

- **Personnel costs**, that is, the costs of the personnel involved in the project. This is computed from the effort and rates. Daily (or hourly) rates, for profiles or individuals, are typically determined by the performing organization and are computed by considering all the items that contribute to the cost of a resource (e.g., salary, tax, retirement funds, and fringe benefits). In some cases and countries, this can result in the cost being twice the gross salary.
- **Materials and supply**, that is, the costs of the material necessary to produce the project outputs, such as, for instance, the construction material necessary to build a house. For software development, this cost is usually very low.
- **Hardware and software**, that is, the costs of specific hardware and software necessary to carry out a project.
- **Travel**, **meeting**, and **events**, that is, the costs necessary to meet with customers and other stakeholders.

- **Consultants and subcontracting**, that is, the costs related to work that is subcontracted.
- **Other costs**, that is, all those expenses that do not fit nicely in the other categories, such as, for instance, books, training, and renting equipment.

Indirect costs include the expenses necessary to run the facility and make work actually doable. Indirect costs are also called **overhead** and include

- **General overheads**, that is, the costs necessary to run the infrastructure supporting the production team (e.g., office space rents, heating, administrative staff, consumables, and networking).
- **Project overheads**, that is, the costs necessary to run the project-specific infrastructure. Project overheads apply to large projects or to specific situations in which the accounting is performed at this level of detail. In general, all indirect costs are accounted for as general overheads and distributed uniformly among all projects that an organization is involved in.

There are three main aspects related to the management of indirect costs. The first is to determine what expenses contribute to their computation. This is usually done once and for all by analyzing the recurring expenses due to supporting operations and work.

The second is to forecast the indirect costs. This is done on a yearly basis and requires an organization to assess the fixed and variable costs, such as, for instance, costs of heating, rent, and electricity. Historical data are the basis for these kinds of estimations.

The third is to define a policy to distribute indirect costs to the different projects of an organization. A fair approach allocates indirect costs proportionally to a project's size since, in principle, larger projects will use more services and cause higher indirect costs. This can be done in different ways. One adds a flat rate to the personnel costs to take care of overhead; in this case, the overheads are computed as a percentage of the project effort. Another adds a rate proportional to the cost of each resource; the computation is like in the previous case, but resources with higher salaries will contribute with a higher overhead. A third technique computes a percentage of the overall budget of the project.

Finally, it has to be remarked that overheads can be a significant cost item of a project, in some situations even doubling the rates to be used for the personnel involved in a project.

3.7.2 Cost Element Structures

Budget should count each expense only once and no double accounting of the same costs should take place. For this reason, organizations use a **cost element structure** (CES), that is, a hierarchical structure that defines precisely what are the cost items to take into account in each project.

Table 3.14 Budget Example

	Unit Cost	*Overhead*	*Effort/Units*	*Total*	*Comment*
Personnel					
Resource A	€50	€30	100	€8000	
Resource B	€40	€30	100	€7000	
Total personnel costs				€15,000	
Hardware and software					
Hardware	€300		2	€600	Two tablets for testing the Application
Software	€80		1	€80	Library for graphs
Total hardware and software				€680	
Other costs					
Travel	€1000		5	€5000	
Meetings	€200		3	€600	
Training				€0	
Total other costs				€5000	
Total				€12,280	

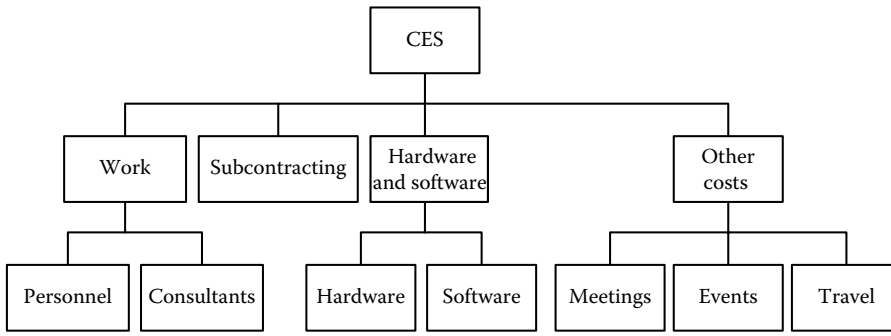

Figure 3.15 CES example.

Similar to a WBS, a CES can be presented as a tree. The advantages of a CES include no double accounting taking place. Moreover, it helps to present the budget in a standardized way and allows to aggregate and present financial data at different levels of detail.

For instance, the costs we presented in Table 3.14 could be organized as shown in the CES of Figure 3.15.

3.7.3 Determining the Project Costs

Given a CES, the rates of personnel, overhead costs, and a project plan (with the estimation of effort and other information, such as travels, etc.), the determination of the project costs can proceed by adding all the expenses foreseen in the project.

For the mathematically inclined, on the hypothesis that overheads are computed as a percentage on personnel costs

$$\text{Project budget} = \sum_{j=1}^{m} \text{Hours}_j * (\text{Cost}_j + \text{Overhead}_j) + \sum_{i=1}^{n} C_i \qquad (3.19)$$

The first part of the formula determines the personnel costs. For each of the m resources involved in the project, in fact, we determine the cost by multiplying the effort by the costs and overhead associated with the resource.

The second part of the formula includes the other foreseen expenditures.

This computation is typically performed with a spreadsheet or using a Gantt charting tool, in which case it is also possible to compute a detailed cash flow. Table 3.14 shows an example of project costs computed and presented with a spreadsheet. Whether the computation is performed on hourly costs rather than daily or monthly tariffs depends on the project size, with larger projects privileging longer periods, also to take into account the higher variability of the estimations.

For a large project, work packages are a good starting point to compute the budget. That is, the manager computes the budget for each work package and then aggregates the data. This process allows one to allocate each project expenditure in a two-dimensional matrix, made of the CES and of the WBS, as shown in Figure 3.16.* As pointed out in the Department of Defense (2011), the intersections of elements of the CES and of the WBS are the cost elements that need to be traced during project execution.

3.7.4 Managing Project Costs

A **project budget** is a view of the predicted cash flow (incomes and expenditures) of a project.

Its main goals are

1. To ensure that the money is available when it needs to be spent.
2. To monitor project expenditures so that the project remains within the budget, or appropriate actions can be taken when this is not the case.

The cash flow is built by determining, for each reporting period, the foreseen incomes and expenses. The reporting period depends on the project size and

* Note that we could include a third dimension, made of the organizational structure of the company involved, if the project costs have to be allocated to different departments.

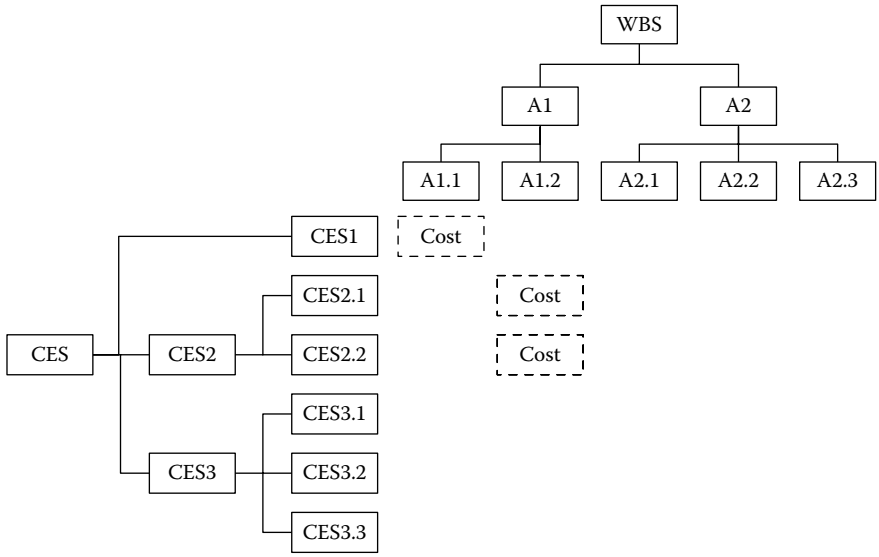

Figure 3.16 Cost accounting elements (CES and WBS).

on company policies. The allocation of incomes and expenses over time depends instead on the project plan and on contractual agreements with subcontractors, which might require an advance payment or might happily be paid after delivery. This information allows the project manager to determine the amount brought forward and the financial needs of a project, as illustrated in Table 3.15.

In the table:

- The first group collects the expenses over time, organized according to the CES.
- The second group records the expected incomes, as determined by the contractual agreement.

Table 3.15 Budget Structure

	Q1	*Q2*	*Q3*	*Q4*	*Total*
Expenses					
Expense 1	€10,000	€30,000	€50,000	€10,000	€100,000
Expense 2	€20,000	€40,000	€60,000		€120,000
Total expenses	€30,000	€70,000	€110,000	€10,000	€220,000
Incomes					
Payment	€50,000			€200,000	€250,000
Total incomes	€50,000	€0	€0	€200,000	€250,000
Balance	€20,000	−€70,000	−€110,000	€190,000	€30,000
Financial need		−€50,000	−€180,000		

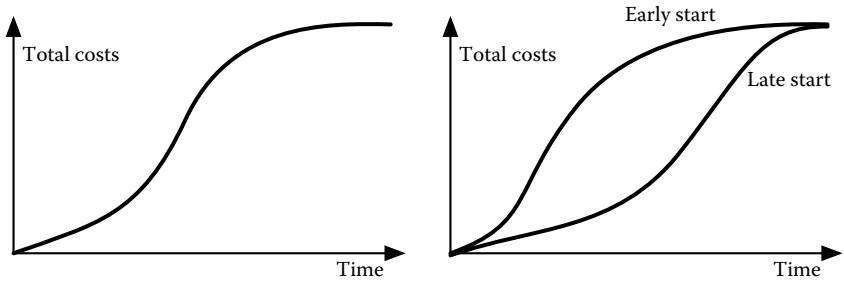

Figure 3.17 Expenditure profiles for a project.

- The third group determines the financial needs. In particular, the row "balance" is the net balance at the end of the period, computed as the difference between incomes and expenses. The row "financial need" indicates how much money the project needs to borrow to carry out activities. It is the result of adding the balance of the current period to the credits and debits accumulated.

Note that if we plot the accumulated expenses of a project over time, we typically get an "S"-shaped curve, with the expenditures rapidly rising when the project is in full swing and being relatively small at the beginning and end of a project. This is shown in the left part of Figure 3.17.

Since the expenditures are related to the project activities, and various activities in a plan have a slack, we can determine, for each project, two different cost profiles, one in which all activities have an early start and the other in which all activities have a late start. This individuates a **banana-shaped** region in the graph, as shown in the right part of Figure 3.17. If the project is not delayed, the actual expenditure profile will be in the "banana" region. See Burke (2006) for more details.

Determining financial needs over time establishes a simple reference framework to monitor and control project expenditures. We will see in Section 3.9 a more sophisticated technique that allows progress and costs to be controlled in an integrated way. Here, it is sufficient to remark that project and organizational constraints determine what item of the CES the project manager should monitor, what detail of accounting is necessary, and what margins of maneuver the project manager has in authorizing expenditures. For instance, personnel costs are often directly managed by the administrative offices of the performing organization. Although they concur in determining the costs of a project, it is not the responsibility of the project manager to monitor the expenditure and ensure that salaries are paid.

3.8 Project Execution

Project execution is where work takes place and deliverables are actually produced. That is, during this phase, all the activities described in Chapter 2 are actually performed.

There are three main management activities to be taken care of during this phase. These are

1. Kicking activities off
2. Collecting the output of activities
3. Collecting information about the project health.

3.8.1 Kicking Activities Off

The goal of this activity is to formalize the start of one or more project tasks with a meeting or some other communication. There are three good reasons for doing so. The first is to ensure that there is shared vision on the work that has to be performed. The second is to ensure that the project team has the necessary resources to carry out the work. The third is to make official an actual start date. This has a symbolic value, which helps everyone to get into the right mindset and actually get started with the work.

The number and type of tasks for which a kick-off is necessary, the amount of time required to prepare a kick-off, and the formality of the kick-off activity depend on the project at hand. At a minimum a **kick-off** meeting should be held to start a project. Large projects might also foresee a kick-off meeting to start each work package. Holding a kick-off meeting to start some risky or critical activity in the project is also a good idea.

Formal meetings are not always good. Small projects or projects with experienced and well-oiled teams require less formality. In these situations, a stand-up meeting or just a chat at the coffee machine could be sufficient.

See Section 5.3.3.2.1 for more information on how to structure a kick-off meeting.

3.8.2 Collect the Output of Activities

Closing activities is the second important management practice during project execution. A proper closure, in fact, ensures that activities are promptly ended when the work is completed, rather than dragging around. Moreover, it becomes possible to assess the lessons learned and to understand how to improve in the next phase. Finally, a proper closure ensures that the project outputs are properly collected.

Concerning the means and tools, we can apply considerations similar to the ones we made for kick-offs. The main *tool* is a meeting, in which the team presents the results, the lessons learned are discussed, and the project outputs are stored.

3.8.3 Collect Information about the Project Status

The goal of this activity is to assess the project status. It can be performed on a regular basis or on a need basis, like, for instance, when a critical event occurs.

Systematic collection of quantitative data about the status of activities and work can be used to monitor progress, costs, and time and thus to evaluate whether the project is running late or costing more than budgeted. This is covered in Section 3.9.

Similar to the previous case, quantitative data about the number of defects, change requests, and risks that occurred can provide information about the status of the project and that of its outputs. This is covered in more detail in Section 4.3.

Discussions and status meetings with stakeholders can provide qualitative information about team morale, progress, and other information about the project.

3.8.4 The Project Routine in Agile Methods

The agile routine is a good example of a systematic application of the practices we have described above.

Agile methodologies are based on a strict sequence of fixed-length development activities. Each development frame is called a **sprint**. The project routine for agile teams is the following:

- At the beginning of the sprint, hold a meeting highlighting the sprint goals.
- During the sprint, on a daily basis, hold a 15 min stand-up meeting to highlight the main achievements, main obstacles, and commitments for the day.
- At the end of the sprint, release a **potentially shippable product** and demo it to the team and the customer.

3.9 Project Monitoring and Control

If we were perfect planners in a completely predictable world, being a project manager would probably be rather boring. Our plans would be a perfect representation of the future and the goal of monitoring and control would be that of observing how our project develops and progresses according to the plans we set. In a completely predictable world, in fact, many other activities would be rather boring.

In practice and, maybe, also fortunately, we are not perfect planners and the world is unpredictable. We need to plan not only to tame uncertainty but also to monitor and replan to take the appropriate corrective actions when the gap between our plans and reality becomes too wide.

Monitoring and control is a structured process that helps us

1. Understand whether our projections have been confirmed by the actual execution of the plan. This is achieved by comparing our plan (called **baseline** plan) with that derived by mapping how work is **actually** progressing.
2. Understand whether any deviation has occurred and their impact in determining the future trajectory of the project.
3. Understand what actions we can or need to take to bring the project back to the nominal situation, if a deviation has occurred.

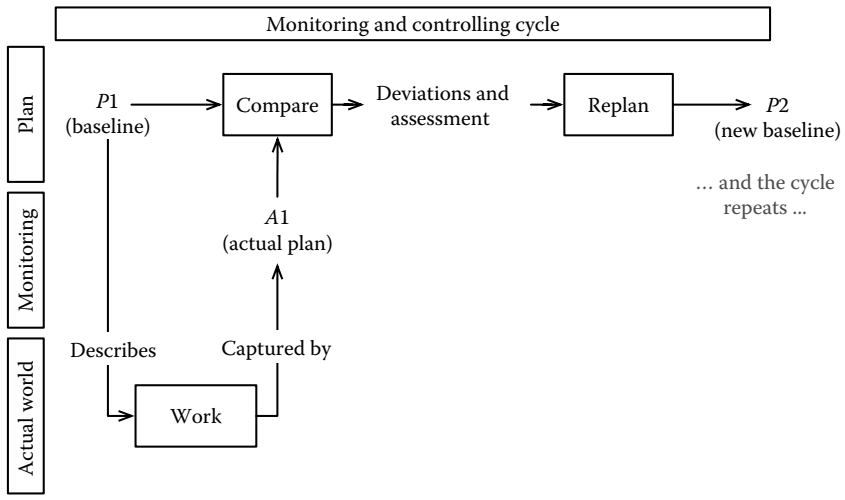

Figure 3.18 Monitoring and control cycle.

Thus, an effective monitoring and control process requires to periodically collect data, which we can compare with our plans. This generates a sequence of plans, as illustrated in Figure 3.18. The first **baseline** plan, $P1$, is generated before the start of the project. As the project progresses, data are collected in order to understand the actual progress; this can be done by building a Gantt chart, called **actual plan**, representing how the actual work has progressed ($A1$ in Figure 3.18). The baseline plan and actual progress are compared, deviations analyzed and taken care of and, if necessary, a new plan set ($P2$ in Figure 3.18). The new plan finally becomes the baseline for the next monitoring cycle.

The rest of this section is dedicated to defining what data are usually collected and what techniques are used to assess current status and make the new projections. We start by describing some simple techniques to monitor progress and time; we then continue with some basics about monitoring costs. Section 3.9 is dedicated to presenting earned value analysis (EVA), a technique that allows one to take an integrated view at the project progress, by measuring progress, time, and costs together.

We conclude the section describing the approach adopted by agile methodologies and with some information about software metrics and their role in measuring progress.

3.9.1 Bookkeeping Your Plan: Actual Start and End Dates

The simplest form of monitoring that can be performed on a plan consists in keeping track of the **actual start** and **actual end** dates of each activity in the plan. Note

that variations in the start or end date of a task can propagate to other activities in the plan, if the delay (in the start or end date) is bigger than the total slack of the activity.

Many Gantt charting tools allow project managers to compare a given baseline of a plan with the actual data. Each activity is represented by two bars; the lower bar is laid down using the planned data (i.e., the data of the baseline plan), while the upper bar shows the actual data (i.e., the data derived from the last monitoring performed). An optional bar is used to show the progress performed in an activity, in terms of the effort actually spent.

An example is shown in Figure 3.19, taken from an actual project. In the upper part, we can observe two activities, A1 and A2, connected together through a deliverable. Activity A1 has been delayed. This is shown in Figure 3.19 by the upper bar, which is longer than the lower bar. The delay in the end date of A1 propagates to A2, since the plan had no slack to accommodate for any delay in A1. Thus, activity A2 starts later than planned (upper bar shifted to the right with respect to the lower bar). Activity A2, however, ends earlier than expected, compensating the delay caused by A1. This is shown in Figure 3.19 by the upper bar of A2, which ends before the lower bar.

In A3, the second case, the activity starts as planned and lasts less than expected: both bars start on the same date and the upper bar is shorter than the lower bar.

Activity A4 is only partially completed: this is shown by the fact that the upper bar is only partially filled. Many tools show the percentage of completion to the right of the bar. Thus, for instance, we can see from the diagram that A4 is 40% complete. An aspect that is less obvious is that A4 was not in the original plan when the baseline was set. A4, in fact, has only one bar and no baseline bar is shown.

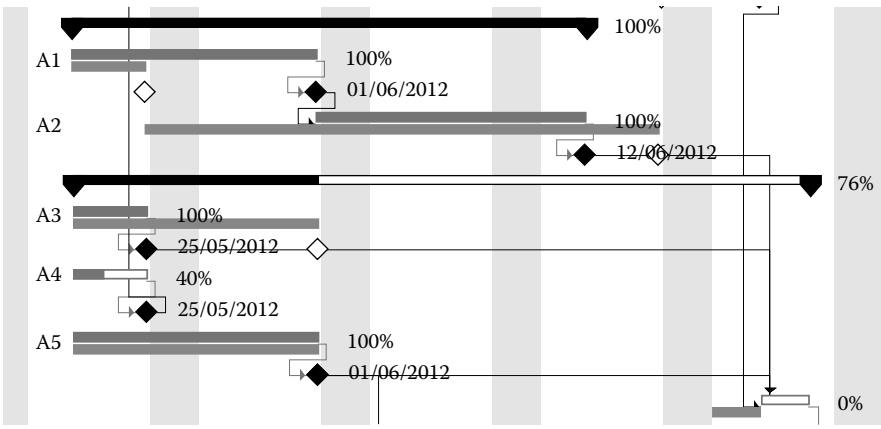

Figure 3.19 Gantt's monitoring.

3.9.2 Monitoring Time and Work

A slightly more complex form of monitoring measures the progress to understand whether a project is on time, early, or late. The process is bottom-up, with data collected from each activity that determines the general project status.

A simple estimation process is based on the collection of data about the *actual* work performed for each task. On the hypotheses that the estimations are accurate, the actual progress depends on the work, and that work is evenly distributed on a task, in fact, simple computations of percentages allow us to determine the progress we expected at the date, the current progress, and the estimation to the end.

In more detail, if we have a task A for which we have estimated an effort of $w_{planned}$ to complete the task and we currently have spent an effort of w_{actual} on the task, we can compute the **percentage of work completed**, p_{actual} as

$$p_{\text{actual}} = \frac{w_{\text{actual}}}{w_{\text{planned}}} \tag{3.20}$$

For instance, if we estimated task A to require an effort of 40 h and we have already worked 20 h on A, we can estimate the task to be 50% complete, since $20/40 = 50\%$.

Note that if we determine the amount of work we *should have* produced at the monitoring date on a task, we can compare it with the actual progress and determine whether we are early or late. In fact, if the actual progress is greater than the expected progress at the monitoring date, we are early; we are late otherwise. This can be done by looking at a task's duration and planned start date.

In fact, if work is evenly distributed, we can determine how much progress we expect per calendar unit and, consequently, how much progress we should have produced at the monitoring date. For instance, if activity A is scheduled to last 5 days, we can expect to produce 20% of the work per calendar day. If 3 days have elapsed from its planned start, the progress we expect to have produced is 60%. More formally, the expected percentage of work $p_{planned}$ is

$$p_{\text{planned}} = \frac{t_{\text{now}} - t_{\text{start}}}{t_{\text{end}} - t_{\text{start}}} \tag{3.21}$$

where t_{now} is the monitoring date, t_{start} is the start date of the task, and the denominator contains the duration of the activity, namely, $t_{end} - t_{start}$.

The data computed above can be used to determine whether each activity is early or late. We can do a bit more and also compute the new estimations to the end. The ratio $p_{actual}/p_{planned}$ gives us the *efficiency* with which we are producing effort in a task. Thus, for instance, if the ratio is less than 1, we are inefficient in the project execution, while if the ratio is greater than 1 we are producing more effort than we planned.

Given the efficiency and the hypotheses above, we can thus revise our plan and determine the new *estimated duration* of the activity and the new *estimated end date*

of each task. The *estimated duration*, in particular, is the planned duration of the task divided by the efficiency. The *estimated end date* is given by adding the estimated duration of the activity to the actual start date of the task.

In formulas

$$t_{\text{estimated end}} = \frac{t_{\text{end}} - t_{\text{start}}}{p_{\text{actual}}/p_{\text{planned}}} + t_{\text{actual start}} \tag{3.22}$$

where $t_{end} - t_{start}$ is the planned duration, $p_{actual}/p_{planned}$ is the efficiency with which we are producing work, and $t_{actual\ start}$ is the actual start date of the task.

The computation above allows one to assess the project status. Many Gantt charting tools perform the computations above, sometimes by having the user specify directly the "percentage of work complete" or "percentage of duration complete." For instance, Aksel (2008) discusses the implementation of these computation in MS Project.

It has to be remarked, however, that the hypotheses made at the beginning of the section do not always hold. In particular, it could be the case that our initial estimation of the work necessary to complete an activity is wrong. In this case, we first need to revise our estimation and then proceed with the computations described above.

The problem in the second hypothesis, namely, that progress corresponds to the work spent, is dealt with similarly to the previous case: we either revise our estimations of work to more accurately reflect the work needed or, if we have it, we find a way to measure the percentage of technical progress we expected at the monitoring date.

If the third hypothesis is false, we need to revise the computations to take into account the actual workload we expected in each activity. This, however, makes computations a lot more complex and one should evaluate whether it really makes sense to still apply the method.

Finally, before applying this method, it is always a good idea to assess the benefits and costs. For instance, in some cases, a less accurate and more informal monitoring could be equally good in terms of information and a lot more efficient in terms of the work required.

3.9.3 Monitoring Costs

Budget and expenditure monitoring can be performed using the same approach we defined above. The percentage of time elapsed determines a projection on the expense we should have performed. This is compared with the money actually spent on each item of the project's CES, in order to determine whether we are on budget, overspending, or underspending.

The process requires one to maintain an updated ledger of the project expenditures that we can use to assess the money actually spent. Since accurate financial

bookkeeping is required by law, the data collection process should be simpler than in the previous case.

Similar to the previous case, however, there can be some noise in the estimations we produce and in interpreting the financial data. These can be more or less evident, according to a project's size and duration and the magnitude of expenses to be performed in the project.

The first source of noise is due to the fact that many payments happen in lump sums at specific points in time during the project. Thus, expenditure does not progress linearly with time but rather with discontinuities. As mentioned above, the relevance and importance of such discontinuities vary with a project's size and, of course, with the size of payments budgeted in the project. Consider, for instance, a project that budgets hardware for deploying a solution to the client. Any monitoring on the corresponding CES item will either show 0% or 100%, according to whether the hardware has been bought or not. (Certain planning tools allow one to specify the spending profile, in order to more closely define expenditure over time and thus simplify budget monitoring.)

The second source of noise is due to delays in payments. Subcontractors often receive delayed payment (in Italy up to 3 months) after the actual delivery of tasks. In such cases, the amount shown in the ledger containing the actual expenses will be different from the money actually available. Simple accounting practices, such as the usage of a "liability" account, address the issue.

A third (and minor) cause is the way in which various expenses get classified in the CES. Although, in principle, errors or choice in the way in which some expenses are classified should never occur, it happens sometimes in practice.

3.9.4 An Integrated Approach: Earned Value Analysis

The techniques we have seen above provide a partial view on the status of the project. If we monitor progress like we describe in Section 3.9.2, we can tell little about the costs we are incurring to achieve the technical progress. Conversely, the technique for monitoring the budget described in the previous section tells us nothing about the technical progress we achieved. Even more complex is understanding the efficiency with which we are achieving progress.

EVA is a technique that addresses the problems mentioned above by representing progress, costs, and schedule in the same measurement unit. This allows one to compare them and thus understand a project's status and derive trends and projections.

The technique was defined in the 1960s and developed subsequently over a period of 20 years. Today, it is an important methodology that is widely adopted. See Christensen (2013) for a comprehensive bibliography, which includes various historical references. Over time, technique and terminology have been standardized. In this section, we use both the historical definitions and the new standard terminology.

The concept behind EVA is relatively simple: progress and schedule are mapped in terms of money and compared with the actual expenditure measured in a project. The analysis of the absolute values, that is, where progress, schedule, and costs stand at the monitoring date, informs us about the current status of a project. The ratios among the values tell us about the efficiency of our project, which we can use to make projections to the end.

The brilliant idea of EVA is how technical progress can be measured in terms of money. This becomes obvious, however, as we measure (technical) progress in terms of the work to produce it, to which we can assign a cost. We will refine and make the definition more precise in a couple of paragraphs.

The concepts we need to introduce are

- **Planned value**, that is, an analysis of the planned progress over time. It is also known as **budgeted costs of work scheduled** (BCWS).
- **Actual costs**, that is, the actual expenditure we incurred in the project. It is also known as **actual costs of work performed** (ACWP).
- **Earned value**, that is, an assessment of the value (technical progress) we produced so far. It is also known as **budgeted costs of work performed** (BCWP).

Let us see how we compute each of these values and then how these are put into use.

3.9.4.1 Planned Value

According to the Project Management Institute (2004), "**planned value** (PV) is the authorized budget assigned to work to be accomplished for an activity or WBS component. Total planned value for the project is also known as budget at completion (BAC)."

PV can be tabulated or plotted over time as shown in Figure 3.20, defining the expenditure profile we expect from a project. The computation of PV proceeds as follows:

- We choose a reporting period (e.g., monthly and quarterly).
- We draw the Gantt chart of the project.
- We compute the costs of each activity in our project. This is the PV of the activity.
- We divide the costs of each activity according to the reporting period. That is, if an activity *A* has a duration of 3 months, starting from month 4 of the project and at a cost of 3000 USD, we will allocate 1000 USD at month 4, 1000 USD at month 5, and 1000 USD at month 6.
- We sum all the amounts per reporting period.

Since the PV shows the values of the costs of our plan, it provides a cost baseline of our project. If the project behaves exactly as we planned, the expenses we incur will follow exactly the profile defined by the PV.

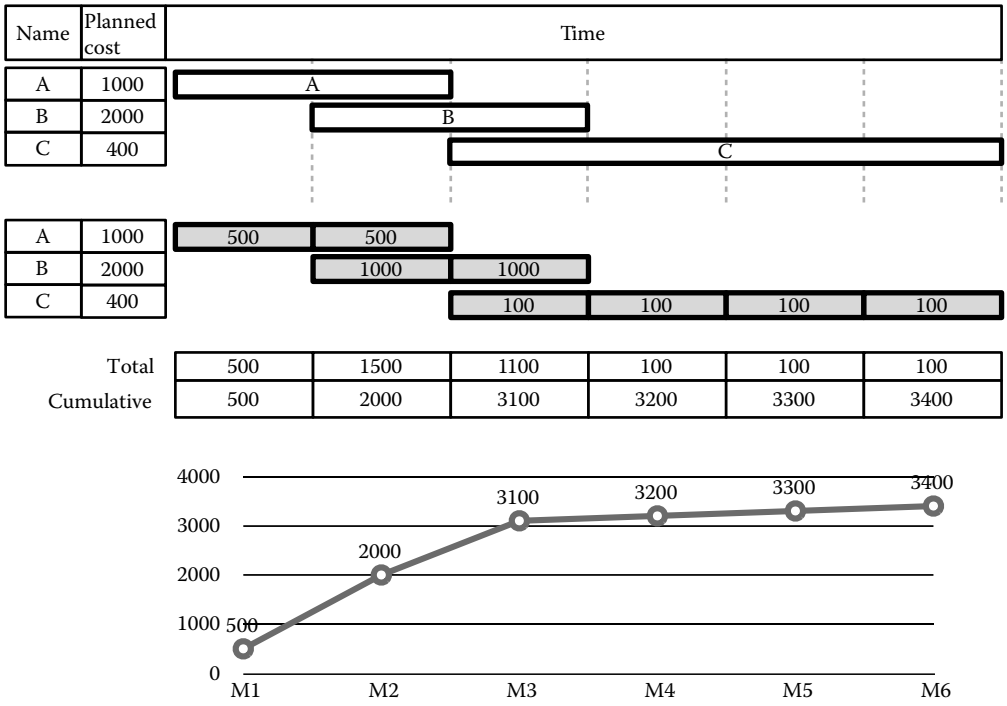

Figure 3.20 Computation of planned value.

3.9.4.2 Actual Costs

The **actual costs** (ACs) record the actual expenditures we incurred as the plan develops. ACs differ from PV for two reasons:

1. Some activities might have actual start or end dates different from those scheduled.
2. The actual effort and costs necessary to carry out an activity might be different from what was planned.

ACs are plotted similar to the PV, using, as input, the actual plan.

Comparing the PV and the ACs allows the project manager to understand how the actual plan is doing with respect to the plan defined at the project start. For instance, at any given time, we can tell whether we spent less or more than initially planned.

However, this information by itself is not sufficient, since we have no idea about the actual progress we achieved. That is, having spent more than planned could be a very good sign if the technical progress is also above the expectations, since the excessive expenditure could be a sign that our project is ahead of schedule.

In other words, in order to draw conclusions on the project status, we need to evaluate the technical progress. This is achieved with the computation of the earned value.

3.9.4.3 Earned Value

Earned value (EV) is the way in which we measure the technical progress of a project. There are two key concepts behind its definition.

The first is that we measure technical progress in terms of money. This allows one to plot EV using the same measurement unit of PV and ACs.

The second is that the value we assign to technical progress in an activity is exactly the money we *budgeted* for the activity, namely, its PV. Thus, if an activity has a PV of €3000, its EV will be €3000 when the activity is completed.

The EV of a plan is the sum of the EVs of its activities. Thus, when we complete a project, its EV will be the same as its PV. Similar to what happens with the ACs, **in the ideal plan**, the EV is an exact replica of the PV. However, during the actual execution of plans, we will have to take into account deviations from the ideal case.

In computing the EV, we are left with determining two other aspects: the first is when we accumulate EV and the second is how much we accumulate it over time.

The answer to the first question is easy: EV is accounted as the actual work to produce it. Thus, the EV of an activity that has not yet started is 0, while the EV of a completed activity is equal to its PV.

The problem is what happens in between, namely, activities that are started but not yet finished. We could have EV progress linearly with the duration of an activity. Thus, an activity at 40% of its duration could have earned 40% of its PV. The method, however, takes a simpler approach. A percentage of the planned value is accounted when an activity starts; the remaining part is accounted for when the activity ends. One simple allocation rule assigns 50% of the EV when an activity is started and the remaining 50% when the activity is completed. Another rule used very often assigns 20% when the activity starts and the remaining 80% when the activity is completed.

There are two advantages. The first is that the computation is a lot simpler. The second is that the computation is robust with respect to changes in the duration of activities started but not ended at the time of monitoring. Thus, the EV we compute at a given date does not have to be revised when we perform a second monitoring at a later date.

3.9.4.4 Assessing a Plan Health Using Earned Value Analysis

Now that we have these three values, we can assess easily the project status, since we have a way to instantly compare actual costs and actual progress (EV) with respect to our plan.

Figure 3.21 shows some of the values that allow a project manager to assess the health status of a project. Consider the following items:

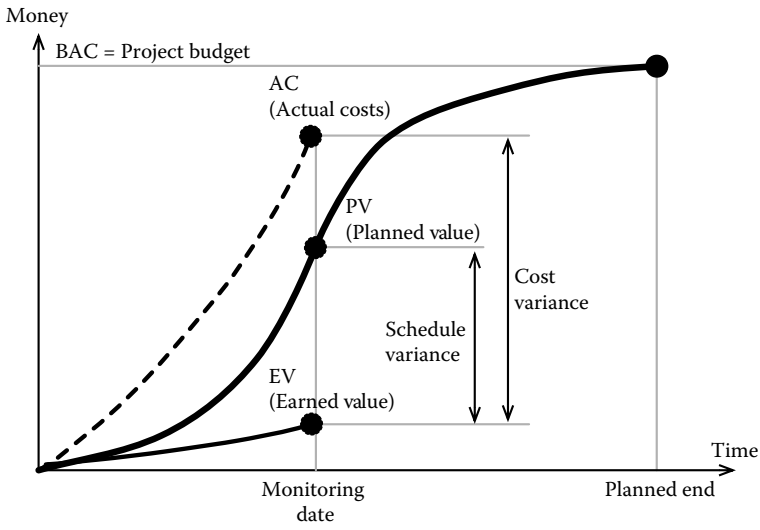

Figure 3.21 Earned value analysis.

- **Comparing PV and EV**: The difference between PV and the EV at the monitoring date tells us whether we are late or early: if EV is above PV, then we are early (we realized more technical progress than expected); the opposite will be true if EV is less than PV. The difference between PV and EV is called **schedule variance**.
- **Comparing EV and ACs**: The difference between ACs and EV tells us whether we are underbudget and overbudget. This concept might be slightly less intuitive than the previous example and deserves a bit more explanation. Consider the case of an ideal plan: PV, EV, and ACs will overlap perfectly. Now, if we are spending a bit more than expected to achieve the planned technical progress, ACs will be a bit higher than EV and PV. Similarly, if we are underachieving, EV will be less than PV and ACs. The difference between EV and ACs is called **cost variance**.

Two other measures are often taken into account to compute the efficiency. In particular, the following two measures are often used.

The **cost performance index** (or *CPI*) measures the efficiency with which we are *earning value*. It is computed as follows:

$$\mathrm{CPI}_t = \frac{\mathrm{EV}_t}{\mathrm{AC}_t} \tag{3.23}$$

where CPI_t is the cost performance index at time t, and EV_t and AC_t are, respectively, the EV and ACs at the same instance of time t.

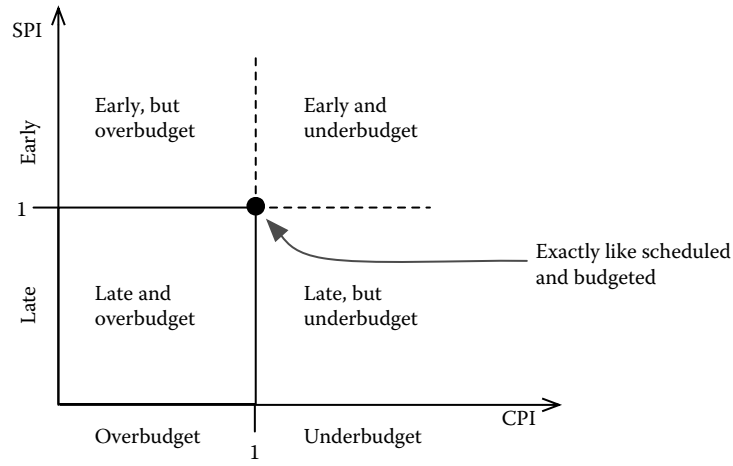

Figure 3.22 CPI and SPI tracking.

Note that $CPI > 1$ if we are achieving technical progress more efficiently than we are spending (we are earning more than one unit of EV for every unit of money we spend). If the trend is maintained, the project will be underbudget. Conversely, if $CPI < 1$, we are inefficient and the project will run overbudget if no corrective action is taken.

The **schedule performance index** (or *SPI*) measures the speed at which we are achieving technical progress. It is defined as follows:

$$\mathrm{SPI}_t = \frac{\mathrm{EV}_t}{\mathrm{PV}_t} \tag{3.24}$$

where SPI_t is the schedule performance index at time t, EV_t and PV_t are, respectively, the EV and PV at the same instance of time t.

Note that $SPI > 1$ if we are achieving technical progress more efficiently than we planned (we are earning more than one unit of EV for every unit of money we planned to spend).

CPI and SPI can be plotted over time on a two-dimensional space, which allows one to understand whether the project is over- or underbudget and behind or ahead of schedule. This is shown in Figure 3.22.

3.9.4.5 Some Considerations about Earned Value Analysis

EVA is an effective approach to provide an integrated view on some measurements that characterize project progress. It also has some limitations. The first is that EVA does not consider the quality of outputs as it focuses only on two of the three main dimensions characterizing a project, namely, cost and schedule. The second is that

the technique is best suited for larger projects, where the effort of the computation is paid back by the synthesis it produces.

The technique has been standardized (Eletronics and Department, 1998) and many resources are available. We mention the ubiquitous PMBOK, NASA, and Office-of-Management-U.S.-Department-of-Energy, from which various tutorial, publications, resources and guidelines can be downloaded.

3.9.5 Monitoring Progress, the Agile Way

All the planning techniques we have seen so far start from an estimation of the effort. During project monitoring, they measure the effort currently spent in an attempt to understand the schedule performance. As we have seen in the previous section, however, the connection between planned effort, actual effort, and technical progress is feeble. Given an actual effort, in fact, it is difficult to understand the technical progress achieved and the work necessary to finish the project. More precise approaches like EVA require a structured data collection approach and quite some work.

For these and other reasons, the agile methodologies take a different approach to measuring progress. The two fundamental differences are that the method focuses on the work left (which is always known) and on an estimation of the *velocity* required to finish the activity. Let us see the method in more detail by taking, as reference, the Scrum development methodology.

One of the fundamental differences of agile methodologies is that all project activities are organized in sprints that have a fixed time frame, that is, a project is a sequence of sprints of the same length.

During planning, rather than an estimation of the effort, the agile team produces an estimation of the size of the work to be done. This is measured in an abstract unit, called **points**. Thus, a given sprint could have allocated 45 points to develop. The value is determined by allocating points to each user story being developed and then by adding all the user stories whose implementation is allocated to the sprint. (See Section 2.1 for the definition of user story.)

Given the fact that a sprint has a predetermined fixed length, the estimation of the points to be developed (or **burned**, using the Scrum terminology) can be used to compute the **velocity** at which the points have to be burned during the sprint, that is, how many points we need to burn in order to deliver on time. These two dimensions define an *ideal* burndown.

At the start of the sprint, the *theoretical* burndown curve is set, summing all the points that need to be burned down. During the sprint, work progresses and the *actual* burndown curve is updated by subtracting the points of the user stories completed and adding those required by any rework needed. At any point in time now, by comparing the ideal and actual burndown, we can get the following information:

- **Remaining points**, that is, how many points we have still to burn down. This is shown by the current value of the actual burndown. Since the time slot is fixed, this value can be used to determine the (new) velocity needed to complete the burndown or, more realistically, what user stories the team will not be able to deliver in the time slot of the sprint.
- **Comparison with the ideal burndown**. By comparing the actual and the ideal burndown, we have a very fast way to understand whether we are early or late. If the actual burndown is below the ideal burndown, we are proceeding at a speed higher than planned; conversely, if the actual burndown is higher than the ideal burndown, we are late.

Figure 3.23 shows an example of a burndown chart. The diagram shows the ideal (line with circles) and the actual (line with triangles) burndown of a sprint lasting 10 days. As can be seen from the graph, the sprint started late and then proceeded at a higher speed than planned, bringing the sprint ahead of schedule.

This approach has two advantages over the approaches we have presented so far. The first is its *simplicity*. The second, subtler, is *focus*: the method focuses on the work left to be done, rather than on the work actually performed, and on the technical progress still to be achieved.

Some critiques have also been moved to the technique. The main problem is probably related to the abstract nature of the *points*, which makes estimation, especially for teams with little experience with agile methodologies, rather difficult. (One could argue that the same problem might arise with providing reliable estimations of the effort needed to complete a user story.) There is also a problem of consistency in the way in which points are interpreted by the different members of a team: four points for a person could be "equivalent" to nine points of another.

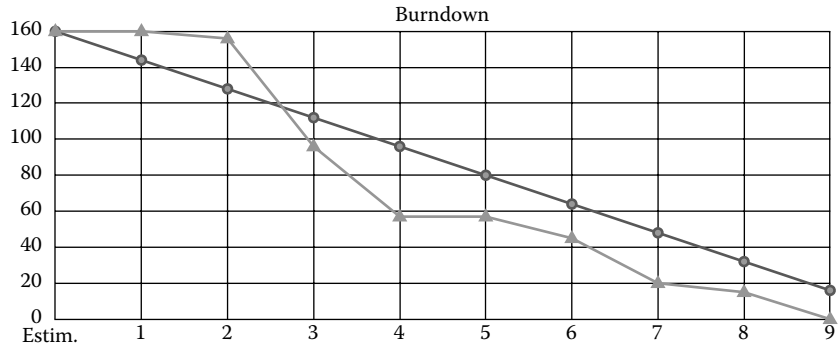

Figure 3.23 An example of a Scrum burndown chart.

3.9.6 Agile-Earned Value Analysis

The burndown chart is very simple to use and to assess the progress toward a goal. However, there are some situations in which it is important to understand how progress relates to the other dimensions that characterize a project, namely, costs and planned work. This is relevant, for instance, when Scrum is applied in contexts where people do not work full-time and with a fixed number of hours on a project, as might be the case in many situations. Moreover, the application of earned value management is suggested or compulsory for certain kind of projects. In these situations, adapting earned value management to an agile context provides a way to get the best of both worlds.

Different approaches have been suggested to apply earned value management to agile projects. Here we follow the one proposed in Rusk (2009), which works on the assumption that the product backlog (that is, the list of all the user stories to develop) must be known and estimated in advance and that the number of sprints has been decided. The hypotheses are not too strong and, as we will see, they can be further relaxed. The reader can look at Sulaiman (2007) and Rawsthorne (2010) to manage more complex scenarios.

The technique proposed in Rusk (2009) plots all data as percentages with respect to completion. The first line that can be plotted is PV, which is called the **gray line** by the method, which also emphasizes the usage of nontechnical terms to simplify the use of the technique. On the assumption of constant production speed, which is quite natural, when we use an agile methodology, the gray line is a straight line passing from the origin (no story developed) to the point having as x-value the date of delivery and as y-value 100% (everything delivered by the planned release date of the last sprint).

It is now possible to draw the other two lines, namely, ACs or **red line** and EV or **green line**, using the terminology of Rusk (2009). The first is the amount of the budget spent at the time of computation with respect to the total project budget. In a situation in which efficiency is as planned, ACs will match PV. The second is computed as the percentage of the story points actually delivered with respect to the total number of story points to develop. Similar to ACs, if the production is efficient as planned, the green curve (EV) will closely match the gray line (PV). An important aspect to highlight is the fact that value is earned when delivered and the points of a story are accrued when the functions implementing the story are actually released. In other words, a rule 0–100% is applied. The analysis will thus be accurate if the production is constant or the reporting period has a granularity that is low enough to abstract the time required for releasing a user story. If a user story requires 5 days to release, analyzing the status with EV on a daily basis will not yield any useful information till the last day: the EV, in fact, will stay the same till the function is released.

The analysis then proceeds as explained in Section 3.9.4.4. An additional advantage of agile EV over the standard analysis is that the projections are simpler, since PV is linear rather than S-shaped.

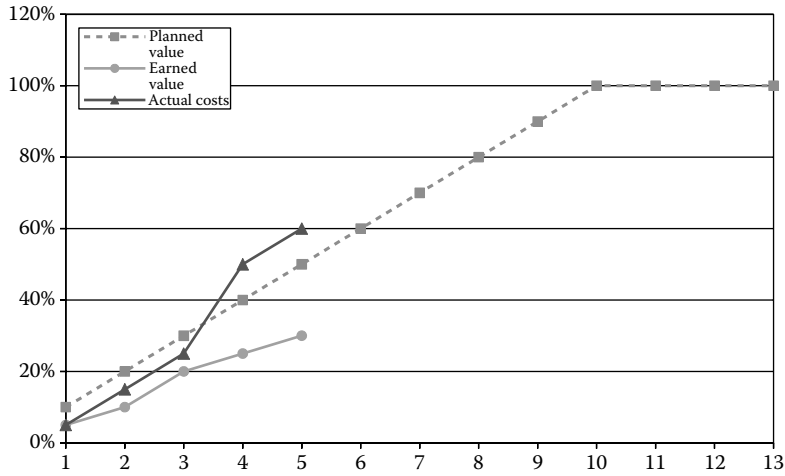

Figure 3.24 An example of agile-earned value analysis.

Figure 3.24 shows an example of an analysis, where the project is behind schedule, since the EV line is below PV, and over costs, since ACs are above PV and EV.

The initial hypothesis of an immutable backlog is quite limiting in an agile context, which embraces change. The technique, however, can be extended to accommodate changing requirements by creating a family of plots. Rusk, in particular, distinguishes between two types of changes. The first type of changes derives from an improved comprehension of the product to develop. Thus, the product backlog will increase in size as we get a better know-how on the work required to

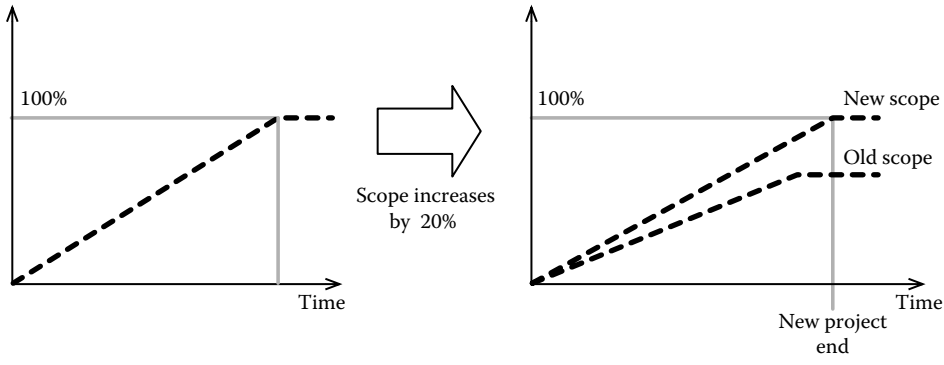

Figure 3.25 Accounting for changes in scope.

fully implement a user story. These changes, however, are residual and can be treated similarly to a systematic error in the estimations.

Different is the case of changes in scope, that is, situations in which the user changes the user stories to be implemented. Changes in scope might also trigger changes in the delivery dates. In this situation, we need to replot our gray line. The process is as follows. A new version of the backlog is the new baseline used to plot the current PV; the old PV, the ACs, and the EV can be recomputed and plotted against the new baseline. This is illustrated in Figure 3.25, where the left-hand side shows the initial status and the right-hand side the status after an increase in scope: both the new and the old PVs are shown. The new PV reaches 100%, while the old PV is at 83% (equivalent to 100%/120%).

3.10 Project Closing

All projects come to an end. According to Meredith and Mantel (2002) and Richman (2012), there are four ways in which a project comes to an end:

- **Termination by integration** and **termination by addition**. These are two successful cases in which the project outputs are integrated in an existing organization or generate a new business or a new business line.
- **Termination by starvation**. This is a case in which a project ends because resources run out.
- **Termination by extinction**. This is a case in which a project is terminated by management because it fails to meet user objectives, it has been superseded by technical advances, or it is not profitable anymore.

Project closing is the last phase of a project, when the project outputs are handed over to the stakeholders, contractual agreements properly taken care of, and project records elicited and stored for future reference. Project closing is also probably one of the most neglected phases of a project with many projects ending up spending 90% of their time with the remaining 10% of work.

There are various motivations for projects not being closed properly. Projects terminating unsuccessfully certainly do not motivate teams or project managers to invest further resources (technical, financial, and, why not, also emotional) on a proper closure. Many successful projects, however, are also not properly terminated. This is due to the following causes:

- Decreasing interest by the project team, as they might be concerned with their next assignment.
- The cost of performing closing activities, which are routine and often require little or no creativity. Consider the case of writing the installation instructions of a software system.
- Underestimation of how much implicit knowledge there is and how fast know-how of this implicit knowledge can get lost. Again, consider the case

of the installation procedure of a complex software: at the end of the project, all information is available and very clear in the project team. As time moves away, it might become more complex to recall by heart all the steps that were required.
- Reluctance, by the project sponsor, to release resources for various reasons, among which is the fear of losing the competences needed to fully exploit the project deliverables, and so on.

For this reason, a proper management process should be enforced to ensure that a proper project closure takes places (or, if not, that the decision is an explicit management decision, rather than the result of inaction).

Project closing can be organized in the following steps (Wysocki, 2011; Richman, 2012; NASA, 2007):

- Getting client acceptance
- Installing project deliverables
- Archiving old deliverables
- Documenting the project
- Performing a financial closure
- Performing postimplementation audit
- Releasing staff.

3.10.1 Getting Client Acceptance

The project is successfully completed when the customer and the project manager agree that the work performed is satisfactory. In software development projects, getting client acceptance might require some effort on the part of the project manager, since the client might be interested in keeping the project team allocated to the project.

According to Wysocki (2011), there are two ways in which acceptance is achieved. The first is a **ceremonial acceptance**, when there is no formal procedure or formal record for accepting project deliverables. Various scenarios are possible, such as a gentleman's agreement between the customer and the project manager or, simply, just reaching project deadlines.

The second is a **formal acceptance**, where there is a formal procedure for accepting project deliverables. In software development, such a procedure nearly always includes a system testing phase, in which tests are executed on the software system being handed over.

3.10.2 Installing Project Deliverables

During this phase, the outputs of the projects are installed. See Section 2.5 for the detailed list of activities to be performed here, in the case of software projects.

3.10.3 Archiving Old Deliverables

If applicable, any deliverable made obsolete by the project needs to be properly disposed of.

In software development projects, decommissioning typically requires one to archive a version of the old deliverables. It is in fact a relatively cheap operation that can bring a lot of advantages, should the new deliverables not work as expected.

3.10.4 Documenting the Project

The goal of this activity is to ensure that the documentation of the project is up to date.

It is a time-consuming activity, which is done on deliverables that might soon be archived. Still, maintaining a proper document record is not only essential in certain domains (like for the development of safety-critical applications), but also essential should a request about a project or its outputs come some time after the project closes. Consider, for instance, a request to fix a bug discovered months after the project end.

Maintaining a proper document record allows us to learn and improve, since we can use the project data and experience as a basis for our next projects. See also Section 3.10.6.

3.10.5 Performing a Financial Closure

The goal of this activity is to ensure that all expenses are paid, all credits cashed, and any remaining budget properly released.

Financial reports are generated during this phase.

3.10.6 Postimplementation Audit

The goal of a postimplementation audit, also called **postmortem**, is a critical analysis of the project in order to learn and improve and to avoid repeating the same mistakes.

According to Collier et al. (1996), a sound postmortem process requires the following steps:

1. **Conduct a project survey**, with the goal of eliciting from the project team the main issues and strengths of the project. This allows one to focus the rest of the process on the important items.
2. **Collect objective information**, with the goal of taking quantitative measures about the project. The metrics suggested by Collier et al. (1996) are shown in Table 3.16.
3. **Hold a debriefing meeting**, during which team members are given the opportunity to provide frank feedback on the project. While potentially

Table 3.16 An Example of Postmortem Quantitative Metrics for Software Development

Cost Metrics	*Schedule Metrics*	*Quality Metrics*
Planned effort and estimated SLOC	Original schedule	
Actual effort and actual SLOC	Final schedule	
History of changes to requirements and code	History of schedule slippage events	Errors at each stage

being very useful, **to be effective they require an open and constructive attitude both from the management and the team members**. If management shows a defensive attitude, in fact, the meeting will most likely yield no useful output, since the team members will not be encouraged to provide frank feedback. Conversely, if the team does not maintain a constructive attitude, the meeting risks become a dumping session in which resentful team members will monopolize time with no useful information. For this reason, it is often better to have a moderator/facilitator.

4. **Conduct a project history day**, which has the goal of understanding the root causes of problems identified at the previous steps. Each meeting is held with a selected number of participants and focuses on one specific problem among those identified at the previous steps. The meeting starts from a review of the project history, which allows participants to identify when the event under investigation started and, subsequently, what caused it.
5. **Publish the results**, during which the management team summarizes the findings of the postmortem and makes them available to the project team and relevant stakeholders in the organization. The content is structured with the following information:
   a. **Project description**: information about the project, to give context
   b. **The good**: what worked well
   c. **The bad**: the three worst factors that impeded the team's meeting its goals
   d. **The ugly**: a prescription for improvement.

A proper recording of the postmortem activities can ensure that the work carried out will also find usage in the medium/long term and become an organizational asset.

For small projects, a simpler procedure can be adopted, such as that described in Dingsøyr et al. (2005), where postmortem activities take place in a half-a-day meeting, organized with

- A brainstorming session, during which issues are elicited using the KJ method, which is based on post-it on a wall method that we saw while building WBS.
- A structuring session, during which the issues are clustered.

- An analysis session, during which the root causes are analyzed.
- A reporting session, where a report, containing main problems, main successes, and root causes, and the post-it used in the meetings are put together.

See Dingsøyr et al. (2005) for more practical information on structuring a postmortem and Birk et al. (2002) for a discussion of the advantages of conducting a postmortem.

3.10.7 Staff-Releasing

The transition from a closing project to new activities can be a disruptive experience from the project staff. It is an important management activity to ensure that this transition is the smoothest possible.

Two important aspects are:

1. Ensuring that proper recognition is assigned to the experience and the results obtained in the project. This is to ensure that working on a project does not turn out to be a disadvantage to the career of individuals.
2. Ensuring that proper tasks are assigned to the team members (e.g., by warning in advance the functional or unit managers about the availability of staff).

A final important aspect pointed out by Wysocki (2011) is celebrating success. Successful projects require teamwork and are a bonding experience. Showing gratitude for the work and effort your team put in a project is a good practice. In the words of Wysocki (2011), "my loud and continual message to senior management is this: Don't pass up an opportunity to show the team your appreciation."

3.11 An Example

In this section, we put into practice various notions illustrated in this chapter, by simulating the process that starts with a customer request and ends with a first plan of the activities to be performed. To do so, we imagine having been contacted by the marketing director of a group of theaters, who wants to be able to sell tickets through the Internet.

The system has to be operational at least one month before the season begins and should have the following functions:

1. View the list of shows of the upcoming season.
2. Register to the platform as owner of a seasonal ticket.
3. Register to the platform as an occasional viewer (one with no seasonal ticket).
4. Renew one's seasonal ticket, possibly choosing a new seat.

5. Buy tickets for a specific show, between one and three weeks before the show begins.
6. Access and use the system through the Internet or via smartphones.

We are asked to set up a project so that we can get started with development.

3.11.1 Initiating

The very first step is to give the project a name. We decide to name ours **Theater 3001**, as a homage to Arthur C. Clarke's *2001: A Space Odyssey*.

The first steps to get started include

1. Writing a scope document, which outlines the stakeholders, goals, budget, timing, deliverables, constraints, and risks. The scope document can be used as a basis for a contractual agreement and to ask for authorization to proceed from the management.
2. Identifying the requirements of the team that will be responsible for the development. The request could also comprehend the selection of a project manager, if this is going to be different from the person tasked with writing the scope document (unusual, but not impossible).
3. Obtaining an authorization to proceed from the client and from the performing organization.

As it is often the case in practice, the example starts with rough and incomplete information. One of the tasks in the initiating phase, in fact, is that of progressively refining and improving, so that we can come out with reliable estimations.

Some of the information we are missing from the specification given above, for instance, includes

1. The *goals of the project*, namely, what is in scope and what is not. For instance, are we being asked to deliver a product (a software system) or a service (develop a software system and operate it for our client)? This has an impact on different areas, such as
    a. *Project timing*, since additional work might be necessary for identifying a suitable hosting platform (a relatively simple task), setting the platform up (slightly more difficult), and preparing our organization to manage the service (definitely a more complex task).
    b. *Project pricing*. Although *operations* are outside the scope of the project (i.e., *outside the scope of any project*, for the definition of *project*), they might influence the project pricing. In fact, if we offer the solution as a service, subscriptions will be a source of revenue when we deliver the system. This could change the pricing schema we are willing to apply; for instance, we might be willing to charge a bit less for the project and return on investment by offering the service to different clients.

2. *Technical details about the required solution*. Various details about the requirements need to be refined. Consider, for instance, aspects related to the complexity of data the application will manage: the number of theaters we need to support, how many types of seasonal tickets there are, whether we need to integrate our solution with other existing systems, such as, for instance, an accounting system.
3. *Priorities*. Different functions have different priorities for the client. Making them explicit could help us understand what development process is best suited for the project at hand, what functions will have to be developed first, what goals are necessary to succeed. Remember the MoSCoW and SMART acronyms!

3.11.2 Building a Plan

On the hypothesis that we have collected from the client all the information required to have a clear picture of the project goals, we are now ready to build a plan. The plan could be organized as shown in Figure 3.26.

The plan is structured in four work packages:

1. **WP1. Inception**, where the system is specified
2. **WP2. Construction**, where the system is built

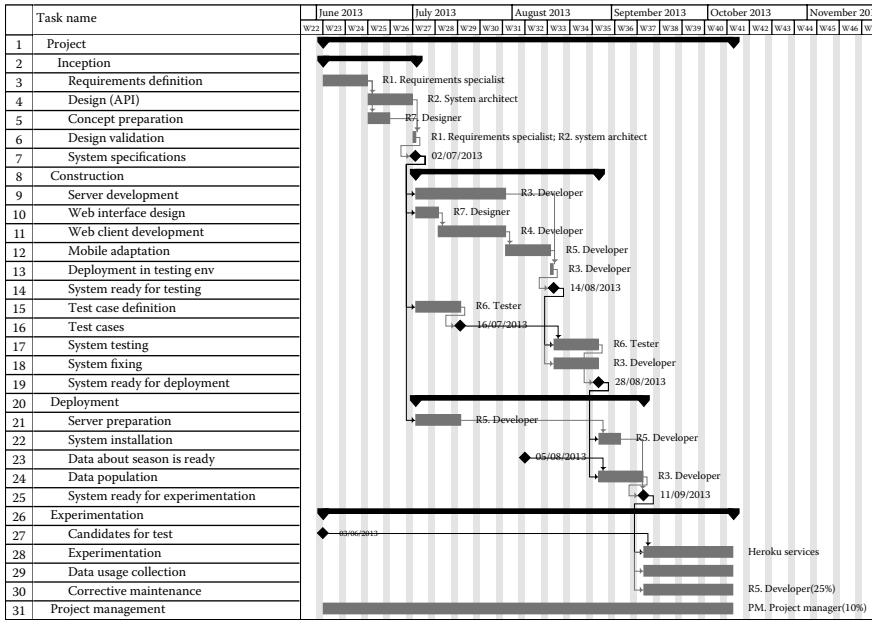

Figure 3.26 The Grantt chart of the Theater 3001 Project metrics for software development.

3. **WP3. Deployment**, which prepares the environment for a first experimentation of the system
4. **WP4. Experimentation**, where we test the system with a small group of selected users to verify the usability and interest in the application.

As can be seen, the plan is roughly scheduled as a waterfall, with work packages performed in sequence, with the exception of WP2 and WP3, which run mostly in parallel.

Two deliverables (ID 23 and 27) are required from the client. The first is the data that we need to populate the system (e.g., the shows that will be shown in the next season; the ticket prices). The second is the list of candidates for WP4, the experimentation. Note that both have a considerable slack, without providing any significant advantage to the project. A better allocation uses an "as late as possible" scheduling; this ensures that the information is collected when needed. Moreover, it minimizes the risks related to selecting candidates for the experimentation too early and then redoing part of the work to replace people who have changed their minds in between.

Another consideration is related to the effort and calendar time we have estimated for each activity. With an algorithmic technique, for instance, we could come out with an estimation of the whole plan, which we could then break down and allocate to the different activities, using the approach presented in Section 3.4.5.4. A different strategy uses reasoning by analogy or expert judgment; in this situation, we either resort to previous analogous projects or to the experience we (or our peers) have accumulated over the years.

Three milestones could be set in the project, corresponding to the main project events. The first could be set at the end of June (specifications ready), the second in the first week of September (system ready for experimentation), and the third at the end of September (system experimented). (Note that the milestones are not shown in the diagram above.)

A first analysis of the plan allows us to understand which activities are more critical. Various activities have some slack. All activities related to system development, however, are on the critical path of our plan: any delay in activities related to the development will delay our project. Thus, system development activities are those that will require more attention on our side, to ensure that the plan is not delayed.

In the plan, we have been a bit optimistic about the experimentation, for which we have not foreseen any support activity, such as, for instance, corrective maintenance that might be necessary if problems are found in the application.

Another activity that does not appear in the Gantt chart is a project management task. We decided to allocate to project management about 5% of the total project effort, since the project is not too complex. Note that the percentage for more complex projects is higher and sometimes estimated between 10% and 20% of the total effort. Whether a specific *project management* activity has to be added to the Gantt chart is a matter of personal choice. One advantage is that the management needs

of a project are made explicit. Another is that the computation of the project costs can be performed using the tool.

Also note that at this stage of planning, the timing indicated by the Gantt chart is indicative as the actual availability of resources has not yet been specified. Planning tools make the assumption that the duration is equal to the effort before any resource is assigned. The actual plan could thus stretch or squeeze, according to the number of resources we will actually be able to allocate.

3.11.3 Creating a Budget for the Project

The first step to compute the budget is to define the type of expenses we will be incurring. More precisely, we need to select a CES and estimate each element of our CES. Since each organization typically has a standardized CES, the difficult part of budgeting is really that of coming out with reliable estimations of our expenditures.

In our project, the following costs will have to be sustained:

- **Personnel costs.** Cost of the personnel responsible for the development of the solution. We imagine having access to a pool of six resources, namely, R1,. . .,R6. Table 3.17 shows their costs.
- **Costs related to hiring a designer for the Graphical User Interface (GUI)** (R7). Note that the cost of the designer is per use and paid for upfront. This is shown by the last column of Table 3.17 that reports *Start*.
- **Costs related to hosting the application for the experimentation**, marked as WS in Table 3.17.

We do not envisage any other cost for this project: no people will travel, and no cost is necessary for meetings, no costs are foreseen for consumables or special equipment.

The budget can be computed using the estimations of the effort and on the cost of the resources, yielding a total budget of €41,000 (of which €37,000 related to producing the system and €4000 to managing the project; the management has

Table 3.17 Cost of Resources We Can Use for the Theater 3001 Project

Name	*Type*	*Maximum Availability (%)*	*Cost (€/h)*	*Overtime (€/h)*	*Cost per Use*	*Accrue*
R1. Requirements	Work	100.00	40	60		Prorated
R2. System Architect	Work	100.00	40	60		Prorated
R3. Developer	Work	100.00	30	60		Prorated
R4. Developer	Work	100.00	30	60		Prorated
R5. Developer	Work	100.00	30	60		Prorated
R6. Tester	Work	100.00	50	100		Prorated
R7. Designer	Work	100.00			€500.00	Start
PM. Project Manager	Work	100.00	60	100		Prorated
WS. Web Services			2			Prorated

Table 3.18 Budget for the Theater 3001 Project

Task Name	*Resource Name*	*Work (Man-Days)*	*Cost*
Theater Project		167	€41,000.00
Project		158	€37,000.00
Inception		27	€7540.00
Requirements Definition	R1. Requirements Specialist	10	€3200.00
Design (API)	R2. System Architect	10	€3200.00
Concept Preparation	R7. Designer	5	€500.00
Design Validation	R1. Requirements Specialist R2. System Architect	2	€640.00
System Specifications		0	
Construction		81	€21,940.00
Server Development	R3. Developer	20	€4800.00
Web Interface Design	R7. Designer	5	€500.00
Web Client Development	R4. Developer	15	€3600.00
Mobile Adaptation	R5. Developer	10	€2400.00
Deployment in Testing Env	R3. Developer	1	€240.00
System Ready for Testing		0	
Test Case Definition	R6. Tester	10	€4000.00
Test Cases		0	
System Testing	R6. Tester	10	€4000.00
System Fixing	R3. Developer	10	€2400.00
System Ready for Deployment		0	
Deployment		25	€6000.00
Server Preparation	R5. Developer	10	€2400.00
System Installation	R5. Developer	5	€1200.00
Data about Season Is Ready		0	
Data population	R3. Developer	10	€2400.00
System Ready for Experimentation		0	
Experimentation		25	€1520.00
Candidates for Test		0	
Experimentation	Heroku Services	20	€320.00
Data Usage Collection		0	
Corrective Maintenance	R5. Developer[25%]	5	€1200.00
Project Management	PM. Project Manager	9	€4000.00

been estimated at about 5% of the overall project effort). The analytical data are shown in Table 3.18.

3.11.4 Changing the Plan to Meet External Deadlines

A common situation is one in which the plan we have built is too long, when compared with the client's constraints. That is, the project is four calendar months, but the client needs the system in 3 months. So we need to shorten the plan somehow.

As we have seen, the options we can use include

1. Changing the project approach
2. Reducing or changing the project scope
3. Allocating resources more efficiently
4. Fast tracking.

Let us see each option in more detail, after reminding that shortening a plan typically increases the risk profile of a project. Thus, a full analysis always needs to consider the possibility that the project is not feasible, given the constraints.

3.11.4.1 Changing the Project Approach

It might be the case that not all functions are equally useful for the client and that the waterfall is not the best process for the project at hand.

For instance, on the hypothesis that "buying a seasonal ticket" is more important than the other functions, we could revise the project and adopt an incremental approach, splitting the project into two cycles. The first cycle could be dedicated to implementing the most important functions; the implementation of the other functions could be postponed to the second stage. This option might reduce the estimated effort and duration by the amount necessary to achieve the project deadline. As an additional bonus, the contract (or payments) could be broken into two distinct parts (one for each cycle), with the option for the client (or supplier) to abandon the project, if the results achieved at the end of the first iteration are not satisfactory.

An alternative is to replan the project using critical chain management. In this scenario, a revision of the project durations and the allocation of feeding and project buffers might end up in a project that can meet the client's deadline.

3.11.4.2 Reducing or Changing the Project Scope

Changes in scope are also possible. For instance, eliminating the experimentation from the plan would spare us one calendar month, moving the delivery date to September 9, which might be good enough for the client, although not as good as the desired deadline. In addition to canceling the experimentation, giving up the implementation of the web client would allow us to squeeze the plan by another 10 days, thus meeting the client deadline, on the hypothesis that the web client is not important for the client. Note, however, that the resulting plan has significant risks, among which:

- The project has no buffer and many important activities are on the critical path. Any small delay could end up being catastrophic for the client, since it would most likely move the delivery date after the expected deadline.
- Canceling the experimentation might lower the quality of the end-product, which we will not be able to test on the field.

3.11.4.3 Allocating Resources More Efficiently

A quick analysis of the plan will show that developers are underutilized and that allocating more developers to the activities might shorten the plan a bit.

Consider, for instance, activities 9, 10, and 11, two of which are on the critical path and, therefore, if shortened, will cause the plan to shorten (till a new critical path kicks in). If we allocated all developers to these activities, activity 9 would be shortened to about 7 days, and activities 10 and 11 to about 8 days, together. This could shorten the plan a bit.

Trying the same approach for other activities has less effect: for instance, allocating more developers to activity 18 does not yield any saving, since it is not on the critical path; the overall duration of that portion of the plan depends on activity 17, which cannot be shortened.

3.11.4.4 Fast Tracking the Plan

Another possibility is to break some dependencies from the plan. We need to analyze the plan in detail, understanding what dependencies are "weaker" than others.

Some opportunities in the plan include the following:

- The "data population" activity, which requires one to populate a database with the data about the upcoming theater season, does not require a fully functional system. As long as the database structure is stable, the work can be performed with little or no risk of rework. Thus, we could break the SF dependency between activity 19 and activity 24 and save 1 week.
- Activity 12, "Mobile adaptation," consists in adapting the web interface for mobile clients. Also, in this case, we could break the dependency and run the activity in parallel with activity 11, either by developing a completely independent interface for the two worlds, or by using an approach in which activity 11 feeds activity 12 as soon as any portion of the web interface is ready (rather than waiting for the whole interface to be ready). This allows us to squeeze the plan by another 10 days, by introducing some risks related to rework.
- Similar to the previous case, we could interleave development and testing. The possibility of rework in this case, however, is rather high. An excellent synchronization is also required between the development and testing teams, increasing the complexity of the activities, stress, and the probability of delivering late.

3.12 Questions and Topics for Discussion

1. Consider the introduction of a market information system in a developing country. A market information system collects and sends information to subscribers about prices of vegetables in local markets, so that farmers can decide

when and where to sell their produce. The information is sent through SMSs. Farmers in developing countries, however, live on a tight budget. What could be the sustainability model of such a solution?

2. Perform the stakeholder analysis of a project to build a motorway connecting two cities.
3. Imagine we want to pilot a software system to track personal finances with a restricted set of users. Try and imagine some SMART goals for the project.
4. Build a WBS for a business reengineering project to automate the enrollment of students in courses. Define also the WBS dictionary.
5. What are the main advantages and disadvantages of algorithmic estimation techniques?
6. What could be the value of mixing different techniques when evaluating the effort required for the development of a software system?
7. What is the impact on the schedule of doubling the effort, according to the COCOMO model?
8. Define a CES for a software development project. Do the same for a house construction project. Look for CES on the Internet, if necessary, and compare the differences.
9. What are the main advantages of earned value analysis? What are the limitations?
10. What are the possible ways in which a project terminates?
11. Consider a ticket reservation system for Greyhound buses. Try and replay the Theater 3001 example, outlining the goals, a plan, and a budget of this new project.

References

Aksel, J. E., 2008. Defining White paper, Celeris Systems. Last retrieved June 21, 2013.

Albrecht, A. J., 1979. Measuring application development productivity. In *Proc. IBM Application Development Symposium*, pp. 83–92. IBM Press.

Birk, A., T. Dingsoyr, and T. Stalhane, 2002, May/June. Postmortem: Never leave a project without it. *Software, IEEE 19*(3), 43–45.

Boehm, B. W., 1981. *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice Hall.

Boehm, B. W., 1984, January. Software engineering economics. *IEEE Transactions on Software Engineering SE-10*(1), 4–21.

Boehm, B. W., C. Abts, A. W. Brown, S. Chulani, B. K. Clark, E. Horowitz, R. Madachy, D. J. Reifer, and B. Steece, 2000. *Software Cost Estimation with COCOMO II*. Prentice Hall, Englewood Cliffs, NJ, USA.

Boehm, B. W. and K. J. Sullivan, 2000, June. Software economics: A roadmap. In *Conference on the Future of Software Engineering at the International Conference on Software Engineering*, Limerick, Ireland, pp. 319–343. Technical report available at http://www.cs.virginia.edu/people/faculty/pdfs/p319-boehm.pdf. Last retrieved November 15, 2013.

Brooks, F. P. J., 1995. *The Mythical Man Month* (Anniversary ed.). Addison-Wesley: Boston, MA, USA.

Burke, R., 2006. *Project Management, Planning and Control Techniques* (4th ed.). John Wiley & Sons, New York, NY, USA.

The Business Model Generation, 2013. The business model canvas. Available at http://www.businessmodelgeneration.com/downloads/business_model_canvas_poster.pdf. Last retrieved June 14, 2013.

Cameron, W. S., 2005, March. Lessons learned again and again and again. *Ask Magazine* (Issue 12). Available at http://askmagazine.nasa.gov/issues/12/features/ask12_features_lessonslearned.html. Last retrieved April 3, 2013.

CDC, 2013. Work breakdown structure dictionary. Available at http://www2.cdc.gov/cdcup/library/templates/CDC_UP_WBS_Dictionary_Template.doc. Last retrieved June 15, 2013.

Center for Software Engineering, 2000. Cocomo II model definition manual. Available at http://csse.usc.edu/csse/research/COCOMOII/cocomo2000.0/CII_modelman2000.0.pdf.

Christensen, D. S., 2013. Earned value bibliography. Available at http://www.suu.edu/faculty/christensend/ev-bib.html. Last retrieved April 2, 2013.

Clark, W. and H. L. Gantt, 1923. *The Gantt Chart—A Working Tool of Management* (Second printing ed.). The Ronald Press Company, New York, USA.

COCOMO 81, 2013e, April. Available at http://csse.usc.edu/csse/research/COCOMOII/cocomo81.htm.

Collier, B., T. DeMarco, and P. Fearey, 1996, July. A defined process for project post mortem review. *Software, IEEE 13*(4), 65–72.

Department of Defense, 2011. Work breakdown structure for defense materiel items. Technical Report MIL-HDBK-881C, Department of Defense Standard. Available at http://www.everyspec.com/MIL-STD/MIL-STD-0800-0899/MIL-STD-881C_32553/. Last retrieved November 15, 2013.

Dingsøyr, T., T. Stålhane, and N. B. Moe, 2005, August. A practical guide to lightweight post mortem reviews. Available at http://www.uio.no/studier/emner/matnat/ifi/INF3120/h05/studentarbeider/Prosjektoppgave/PMA_practical_guide.pdf. Last retrieved January 5, 2013.

Government Eletronics and Information Technology Association Engineering Department, 1998, May. Earned value management systems. Technical Report ANSI/EIA-748-1998, Electronic Industries Alliance.

Friesner, T., 2013. History of SWOT analysis. Available at http://www.marketingteacher.com/swot/history-of-swot.html#. Last retrieved January 5, 2013.

Goldratt, E. M., 1997. *Critical Chain*. The North River Press, Great Barrington, MA, USA.

Hamilton, L. R., 1964, June. Study of methods for evaluation of the pert/cost management system. Technical Report ED-TDR-64-92, MITRE Corporation.

International function point user group, 2013c, Available at http://www.ifpug.org. Last retrieved April 9, 2013.

Jørgensen, M. and M. Shepperd, 2007, January. A systematic review of software development cost estimation studies. *IEEE Transactions on Software Engineering 33*(1) p. 33–53.

Longstreet, D., 2008. Estimating data. Available at http://www.softwaremetrics.com/Articles/estimatingdata.htm. Last retrieved April 9, 2013.

Maylor, H., 2010. *Project Management* (4th ed.). Harlow, England: Pearson.

Meredith, J. R. and S. J. Mantel, 2002. *Project Management: A Managerial Approach*. New York, NY: John Wiley & Sons, Inc.

Merlo-Schett, N., M. Glinz, and A. Mukhija, 2002. COCOMO (constructive cost model). Working notes of Seminars in Software Engineering, Available at https://files.ifi.uzh.ch/rerg/arvo/courses/seminar_ws02/reports/Seminar_4.pdf.

NASA, 1994, May. *Work Breakdown Structure Reference Guide*. Program/Project management series. NASA.

NASA, 2004. Software assurance standard. NASA TECHNICAL STANDARD NASA-STD-8739.8 w/Change 1, NASA.

NASA, 2007, December. Systems engineering handbook. Technical Report NASA/SP-2007-6105 Rev1, NASA.

NASA, 2012. Earned value management (EVM). Available at http://evm.nasa.gov/tutorial.html. Last retrieved October 13, 2012.

NPS, 2013. COCOMO II calculator. Available at http://csse.usc.edu/tools/COCOMOII.php. Last retrieved April 20, 2013.

Office of Management, U.S. Department of Energy, 2012. Earned value management. Available at http://energy.gov/management/office-management/operational-management/project-management/earned-value-management. Last retrieved October 13, 2012.

PERT Coordinating Group, 1963. *PERT Guide for Management Use/PERT Coordinating Group*. Number NASA-TM-101864. U.S. Government Printers, Washington, D.C.

Project Management Institute, 2004. *A Guide to the Project Management Body of Knowledge (PMBOK Guides)* (4th ed.). Project Management Institute, Newtown Square, Pennsylvania 19073-3299 USA.

Quantitative Software Management, 2013. Function point language table. Available at http://www.qsm.com/resources/function-point-languages-table. Last retrieved April 20, 2013.

Rawsthorne, D., 2010. Monitoring scrum projects with agileevm and earned business value (EBV) metrics. Available at http://danube.com/system/files/CollabNet_WP_AgileEVM_and_Earned_Business_Value_Metrics_032510.pdf. Last retrieved June 3, 2013.

Reifer, D., 2000. Web development: Estimating quick-to-market software. *Software, IEEE 17*(6), 57–64.

Richman, L., 2012. *Improving Your Project Management Skills*. American Management Association (AMACOM).

Ruhe, M., R. Jeffery, and I. Wieczorek, 2003. Using web objects for estimating software development effort for web applications. In *Software Metrics Symposium, 2003. Proceedings. Ninth International*, pp. 30–37, Sydney, Australia.

Rusk, J., 2009. Earned value for agile development. Available at http://www.agilekiwi.com/EarnedValueForAgileProjects.pdf. Last retrieved June 3, 2013.

Space Division—North American Rockwell, 1971, June. Space shuttle program—space shuttle phase c/d baseline volume 3: Work breakdown structure dictionary. Technical Report N76-71555, Space Division—North American Rockwell. Last retrieved June 15, 2013.

Stratton, R., 2009. Critical chain project management theory and practice. In *POMS 20th Annual Conference*. Last retrieved June 16, 2013.

Sulaiman, T., 2007, October. Agileevm: Measuring cost efficiency across the product lifecycle. Available at http://www.infoq.com/articles/agile-evm. Last retrieved June 3, 2013.

Tomczyk, C. A., 2005. *Project Manager's Spotlight on Planning*. Harbor Light Press, San Francisco, CA, USA.

University of Southern California, 2013. COCOMO® 81 intermediate model implementation. Available at http://sunset.usc.edu/research/COCOMOII/cocomo81_pgm/cocomo81.html.

Wysocki, R. K., 2011, October. *Effective Project Management: Traditional, Agile, Extreme* (6, illustrated ed.). John Wiley & Sons, New York, NY, USA.

Yang, Y., M. He, M. Li, Q. Wang, and B. Boehm, 2008. Phase distribution of software development effort. In *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '08, New York, NY, pp. 61–69. ACM.

Yu, E., P. Giorgini, N. Maiden, and J. Mylopoulos, 2011. *Social Modeling for Requirements Engineering*. The MIT Press, Cambridge, MA, USA.