PERT charts can help a manager identify bottlenecks in the schedule immediately—even mechanically: A node with many outgoing arcs is a sign of trouble, and the manager should try to reschedule activities to avoid it. Such a node should be rescheduled especially if it is on the critical path. One way to reschedule an activity is to break it up into smaller activities such that the arrows going into the original node are distributed among the new (smaller) nodes. Another schedule risk that can be spotted from a PERT chart is a node with many ingoing arcs. This type of node depends on too many activities and often acts as a synchronization point. While such a node may sometimes be necessary, heavy interdependencies should, in general, be avoided.

Table 8.6, from Boehm [1989], can be used as a checklist of common risks and their typical solutions in software engineering management. Examining the risks listed in column 1 of the table, we can see that they overlap the items that are used in software cost estimation models. If a factor has a high multiplier in cost estimation, it represents a risk that must be managed carefully.

Risk Items
1. Personnel [illegible]
2. Unrealistic [illegible]
3. Developing [illegible]
4. Developing [illegible]
5. Gold plating
6. Continuing [illegible]
7. Shortfalls [illegible]
8. Shortfalls [illegible]
9. Real-time [illegible]
10. Straining [illegible]

TABLE 8.6

Common [illegible] [1989], [illegible]

8.6 CAPABILITY MATURITY MODEL

The Capability Maturity Model (CMM) was developed by the Software Engineering Institute to help both those organizations which develop software to improve their software processes and those organizations which acquire software to assess the quality of their contractors. To fulfill the former goal, CMM provides a guide for software organizations in selecting process improvement strategies by determining the current level of their process maturity and identifying the key factors that would lead to improvement. The underlying assumption, as usual, is that, ultimately, better processes lead to improved quality in the product.

The concept of *maturity* is key in CMM. An immature organization is an organization in which processes are improvised during the course of a project. A process is a sequence of steps that focus on resolving unanticipated crises. Products are often delivered late and their quality is questionable. Conversely, a mature organization has an organizationwide standard approach to software processes that is known and accepted by all engineers. The organization is focused on continuous improvement both in the performance and in the quality of the product. Paulk et al. [1993] states,

> Software process maturity is the extent to which a specific process is explicitly defined, managed, measured, controlled, and effective. Maturity implies a potential for growth in capability and indicates both the richness of an organization's software process and the consistency with which it is applied in projects throughout the organization. As a software organization gains in software maturity, it institutionalizes its software process via policies, standards, and organizational structures. Institutionalization entails building an infrastructure and a corporate culture that supports the methods, practices, and procedures of the business, so that they endure after those who originally defined them have gone.

To capture growth in capability, CMM defines five maturity levels, shown in Figure 8.8, and suggests a number of key practices that help organizations to progress from one level to the next. We briefly review the five levels and discuss the key practices.

Level [illegible] ule. Their [illegible] developing [illegible] uals involved complex [illegible] because of the successful [illegible]

Level [illegible] exclusively [illegible] outputs of the are known [illegible]

Risk Items	Risk Management Techniques
1. Personnel shortfalls	- Staffing with top talent; job matching; teambuilding; key-personnel agreements; cross training; prescheduling key people
2. Unrealistic schedules and budgets	- Detailed multisource cost & schedule estimation; design to cost; incrmental development; software reuse; requirements scrubbing
3. Developing the wrong software functions	- Organization analysis; mission analysis; opsconcept formula-tion; user surveys; prototyping; early user's manuals
4. Developing the wrong user interface	- Prototyping; scenarios; task analysis; user characterization(functionality, style, workload)
5. Gold plating	- Requirments scrubbing; prototyping; cost benefit analysis; design to cost
6. Continuing stream of requirements	- High change threshold; information hiding; incremental development (defer changes to later increments)
7. Shortfalls in externally furnished components	- Benchmarking; inspections; reference checking; compatibility analysis
8. Shortfalls in externally performed tasks	- Refernce checking; pre-award audits; award-fee contracts; competitive design or prototyping; teambuilding
9. Real-time performance shortfalls	- Simulation; benchmarking; modeling; prototyping; instrumentation; tuning
10. Straining computer science capabilities	- Techinal analysis; cost-benefit analysis; prototyping; reference checking

TABLE 8.6

Common risks in software engineering management and techniques for managing them. (From Boehm [1989], reprinted by permission of the author.)

Level 1 is the *initial* level. Level-1 organizations work on a day-by-day schedule. Their processes are *ad hoc*, often even chaotic. If these organizations succeed in developing quality products, it is only because of the efforts and skills of the individuals involved. In most cases, however, that is not sufficient, especially if projects are complex or if the organization experiences a high turnover in its staffing. Moreover, because of the absence of consolidated knowledge and an organizationwide culture, successful projects can hardly be repeated in other, similar situations.

Level 2 is the *repeatable* level. At this level, the process does not depend exclusively on individuals, but also on the organization. The required inputs and outputs of the process are clearly identified, the constraints (schedule and budget) are known, and the resources needed to accomplish the goal are carefully evalu-

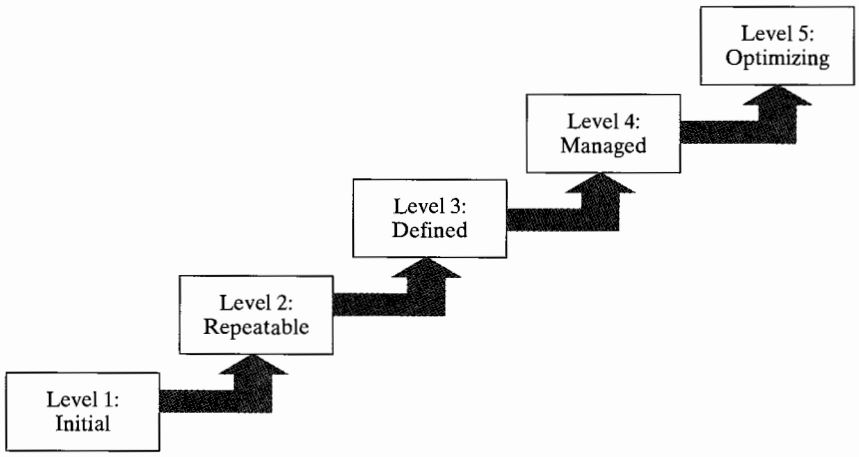

FIGURE 8.8

The five levels of CMM and their relationships.

ated and managed. The main focus of a level-2 organization is on management activities. As a consequence of this effort, the process is repeatable, in much the same way as a piece of program with the same inputs and outputs can be executed several times. Each project follows a defined process, but there are no organizationwide, standard processes.

Level 3 is the *defined* level. At this level, the management and engineering activities that lead to a repeatable process are fully documented, standardized, and integrated. The result is an organizationwide standard process that everyone in the organization knows and follows. Each project may then tailor the standard to develop its own defined process, which accounts for the unique characteristics of the project. Because of the importance of the standard process, a specific group in the organization is often responsible for all the process activities.

Level 4 is the *managed* level. At this level, the organization sets quantitative goals for both the process and the product. Project and process data are collected and analyzed; then they are used to perform corrective actions and improvements while the project is running.

Level 5 is the *optimizing* level. At this level, quantitative feedback is incorporated into the process to produce continuous improvement in the organization's processes in general, not just in the currently running process. New technologies and methods that can introduce innovation into the process are carefully evaluated and transferred throughout the organization. Defect analysis is taken seriously. Defects are analyzed carefully in an attempt to uncover their causes, so that steps can be taken to prevent them from recurring.

According to CMM, each maturity level is characterized by some *key process areas* (KPAs). A KPA defines the key practices on which an organization should focus to elevate its process to that maturity level. KPAs are summarized in Table 8.7.

TABLE 8.7

Key process
of CMM

The
areas in
the next
management
a requirement

8.7 CONCLUDING

This chapter
based on
planning,
tant aspects
While
ment of a
many
engineer
monitor,
only two
fully
neers on
While
have also

CMM level	Key process areas
Initial	None
Repeatable	requirements management Software project planning Software project tracking and oversight Software subcontract management Software quality assurance Software configuration management
Defined	Organization process focus Organization process definition Training program Intergrated software management Software product engineering Intergroup coordination Peer reviews
Managed	Software quality management Quantitive process management
Optimizing	Process change management Technology change management Defect prevention

TABLE 8.7

Key process areas for the five levels of CMM.

The intuition behind KPAs is that unless an organization has mastered all the areas in one level of maturity, it is not able to perform at, or even attempt to reach, the next level of maturity. In the table, we can see the importance of configuration management, required at the repeatable level. Peer reviews, such as inspections, are a requirement at the defined level.

8.7 CONCLUDING REMARKS

This chapter has presented a concise overview of software engineering management, based on the standard management practice of dividing the management function into planning, organizing, staffing, directing, and controlling. We have discussed the important aspects of software that require special management attention.

While we have talked only about management in the large—that is, management of a group of engineers who must cooperate to produce a common product—many techniques we have discussed can and should be used by the individual engineer as well—that is, in the small. Indeed, each engineer must carefully plan, monitor, and execute the plan for his or her own work. Staffing and directing are the only two management functions that do not apply in the small. In particular, a carefully derived work breakdown structure and PERT chart can help individual engineers on nontrivial activities.

While we have discussed the difficulties in measuring software productivity, we have also stated the importance of defining and collecting such metrics. A software