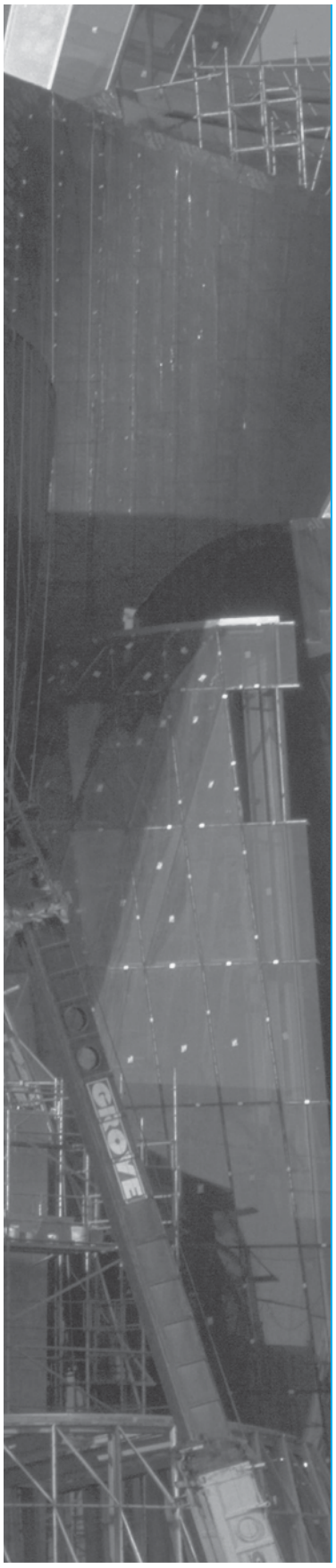

26

Process improvement

Objectives

The objective of this chapter is to introduce software process improvement as a way of increasing software quality and reducing development costs. When you have read the chapter, you will:

- understand the rationale for software process improvement as a means of improving both product quality and the efficiency and effectiveness of software processes;
- understand the principles of software process improvement and the cyclic process improvement process;
- know how the Goal-Question-Metric approach may be used to guide process measurement;
- have been introduced to the ideas of process capability and process maturity, and the general form of the SEI's CMMI model for process improvement.

Contents

Nowadays, there is a constant demand from industry for cheaper, better software, which has to be delivered to ever-tighter deadlines. Consequently, many software companies have turned to software process improvement as a way of enhancing the quality of their software, reducing costs, or accelerating their development processes. Process improvement means understanding existing processes and changing these processes to increase product quality and/or reduce costs and development time.

Two quite different approaches to process improvement and change are used:

1. The process maturity approach, which has focused on improving process and project management and introducing good software engineering practice into an organization. The level of process maturity reflects the extent to which good technical and management practice has been adopted in organizational software development processes. The primary goals of this approach are improved product quality and process predictability.
2. The agile approach, which has focused on iterative development and the reduction of overheads in the software process. The primary characteristics of agile methods are rapid delivery of functionality and responsiveness to changing customer requirements.

Adherents of each of these approaches are generally skeptical of the benefits of the other. The process maturity approach is rooted in plan-driven development and usually requires increased 'overhead', in the sense that activities are introduced that are not directly relevant to programming. Agile approaches focus on the code being developed and deliberately minimize formality and documentation.

I have discussed agile methods in Chapter 3 and elsewhere in the book, so I focus in this chapter on process management and maturity-based process improvement. This does not mean that I prefer this approach to agile methods. In fact, I am convinced that, for small to medium-sized projects, adopting agile practices is likely to be the most cost-effective process improvement strategy. However for large systems, critical systems, and systems involving developers in different companies, management issues are often the reasons why projects run into problems. For companies whose business is large, complex systems engineering, a maturity-focused approach to process improvement should be considered.

As I discussed in Chapter 24, the development process that was used to create a software system influences the quality of that system. Therefore, many people believe that improving the software development process will lead to better quality software. This notion of process improvement is the brainchild of American engineer W. E. Deming, who worked with Japanese industry after World War II to help improve quality. Japanese industry has been committed to continuous process improvement for many years, which has led to the acknowledged high quality of Japanese manufactured goods.

Deming (and others) introduced the idea of statistical quality control. This is based on measuring the number of product defects and relating these defects to the process. The aim is to reduce the number of product defects by analyzing and

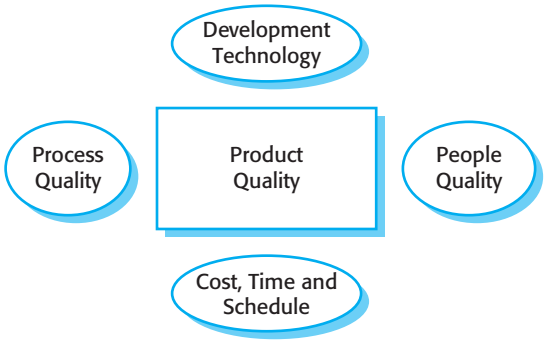

Figure 26.1 Factors affecting software product

modifying the process so that the chances of introducing defects are reduced and defect detection is improved. Once a lower defect count has been achieved, the process is standardized and a further improvement cycle then begins.

Humphrey (1988), in his seminal book on process management, argues that the same techniques can be applied to software engineering. He states:

> *"W. E. Deming, in his work with the Japanese industry after World War II, applied the concepts of statistical process control to industry. While there are important differences, these concepts are just as applicable to software as they are to automobiles, cameras, wristwatches and steel."*

Although there are clearly similarities, I do not agree with Humphrey that results from manufacturing engineering can be transferred easily to software engineering. Where manufacturing is involved, the process/product relationship is obvious. Manufacturing usually involves setting up automated tools and product checking processes. If someone makes a mistake in calibrating a machine, this will affect all of the products produced by that machine. Avoiding mistakes in setting up machines and introducing more effective checking processes clearly improves product quality.

This process quality/product quality relationship is less obvious when the product is intangible and dependent, to some extent, on intellectual processes that cannot be automated. Software quality is not influenced by its manufacturing process but by its design process, where people's skills and experience are significant. In some cases, the process used may be the most significant determinant of product quality. However, for innovative applications in particular, the people involved in the process have more influence on quality than the process used.

For software products, or any other intellectual products such as books or films where the quality of the product depends on its design, there are four important factors that affect product quality. These are shown in Figure 26.1.

The influence of each of these factors depends on the size and type of the project. For very large systems that include separate subsystems, developed by teams who may be working in different locations, the principal factor that affects product quality is the software process. The major problems with large projects are integration, project management, and communications. There is usually a mix of abilities

and experience in the team members and, because the development process usually takes place over a number of years, the development team is volatile. It may change completely over the lifetime of the project.

For small projects, however, where there are only a few team members, the quality of the development team is more important than the development process used. Hence, the agile manifesto proclaims the importance of people rather than process. If the team has a high level of ability and experience, the quality of the product is likely to be high, irrespective of the process used. If the team is inexperienced and unskilled, a good process may limit the damage but will not, in itself, lead to high-quality software.

Where teams are small, good development technology is particularly important. The small team cannot devote a lot of time to tedious administrative procedures. The team members spend most of their time designing and programming the system, so good tools significantly affect their productivity. For large projects, a basic level of development technology is essential for information management. Paradoxically, however, sophisticated software tools are less important in large projects. Team members spend a smaller proportion of their time in development activities and more time communicating and understanding other parts of the system. Development tools make no difference to this. However, Web 2.0 tools that support communications, such as wikis and blogs, can significantly improve communications between members of distributed teams.

Irrespective of people, process, or tool factors, if a project has an inadequate budget or is planned with an unrealistic delivery schedule, product quality will be affected. A good process requires resources for its effective implementation. If these resources are insufficient, the process cannot be really effective. If resources are inadequate, only excellent people can save a project. Even then, if the deficit is too great, the product quality will be degraded. If there is not enough time for development, the delivered software is likely to have reduced functionality or lower levels of reliability or performance.

All too often, the real cause of software quality problems is not poor management, inadequate processes, or poor quality training. Rather, it is the fact that organizations must compete to survive. To gain a contract, a company may underestimate the effort required or promise rapid delivery of a system. In an attempt to meet these commitments, an unrealistic development schedule may be agreed upon. Consequently, the quality of the software is adversely affected.

26.1 The process improvement process

In Chapter 2, I introduced the general idea of a software process as a sequence of activities that, when executed, lead to the production of a software system. I described generic processes, such as the waterfall model and reuse-based development, and I discussed the most important process activities. These generic processes are instantiated within an organization to create the particular process that they use to develop software.

Software processes can be observed in all organizations, from one-person companies to large multinationals. These processes are of different types depending on the

Process characteristic	Key issues
Understandability	To what extent is the process explicitly defined and how easy is it to understand the process definition?
Standardization	To what extent is the process based on a standard generic process? This may be important for some customers who require conformance with a set of defined process standards. To what extent is the same process used in all parts of a company?
Visibility	Do the process activities culminate in clear results, so that the progress of the process is externally visible?
Measurability	Does the process include data collection or other activities that allow process or product characteristics to be measured?
Supportability	To what extent can software tools be used to support the process activities?
Acceptability	Is the defined process acceptable to and usable by the engineers responsible for producing the software product?
Reliability	Is the process designed in such a way that process errors are avoided or trapped before they result in product errors?
Robustness	Can the process continue in spite of unexpected problems?
Maintainability	Can the process evolve to reflect changing organizational requirements or identified process improvements?
Rapidity	How fast can the process of delivering a system from a given specification be completed?

Figure 26.2 Process attributes

degree of formality of the process, the types of products developed, the size of the organization, and so on. There is no such thing as an 'ideal' or 'standard' software process that is applicable in all organizations or for all software products of a particular type. Each company has to develop its own process depending on its size, the background and skills of its staff, the type of software being developed, customer and market requirements, and the company culture.

Process improvement, therefore, does not simply mean adopting particular methods or tools or using a published, generic process. Although organizations that develop the same type of software clearly have much in common, there are always local organizational factors, procedures, and standards that influence the process. You will rarely be successful in introducing process improvements if you simply attempt to change the process to one that is used elsewhere. You must always consider the local environment and culture and how this may be affected by process change proposals.

You also have to consider what aspects of the process that you want to improve. Your goal might be to improve software quality and so you may wish to introduce new process activities that change the way software is developed and tested. You may be interested in improving some attribute of the process itself and you have to decide which process attributes are the most important to your company. Examples of process attributes that may be targets for improvement are shown in Figure 26.2.

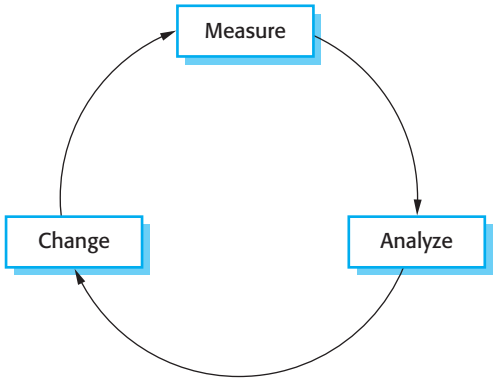

Figure 26.3 The process improvement cycle

These attributes are obviously related, sometimes positively and sometimes negatively. Therefore, a process that scores highly on the visibility attribute will probably also be understandable. A process observer can infer the existence of activities from the outputs produced. On the other hand, process visibility may be inversely related to rapidity. Making a process visible requires the people involved to produce information about the process itself. This may slow down software production because of the time it takes to produce these documents.

It is not possible to make process improvements that optimize all process attributes simultaneously. For example, if your aim is to have a rapid development process, then you may have to reduce the process visibility. If you wish to make a process more maintainable, then you may have to adopt procedures and tools that reflect broader organizational practice and that are used in different parts of the company. This may reduce the local acceptability of the process. Engineers may have introduced local procedures and non-standard tools to support their way of working. As these are effective, they may not wish to give them up in favor of a standardized process.

The process of process improvement is a cyclical process, as shown in Figure 26.3. It involves three subprocesses:

1. *Process measurement* Attributes of the current project or the product are measured. The aim is to improve the measures according to the goals of the organization involved in process improvement. This forms a baseline that helps you decide if process improvements have been effective.
2. *Process analysis* The current process is assessed, and process weaknesses and bottlenecks are identified. Process models (sometimes called process maps) that describe the process may be developed during this stage. The analysis may be focused by considering process characteristics such as rapidity and robustness.
3. *Process change* Process changes are proposed to address some of the identified process weaknesses. These are introduced and the cycle resumes to collect data about the effectiveness of the changes.

Without concrete data on a process or the software developed using that process, it is impossible to assess the value of process improvement. However, companies starting the process improvement process are unlikely to have process data available as an improvement baseline. Therefore, as part of the first cycle of changes, you may have to introduce process activities to collect data about the software process and to measure software product characteristics.

Process improvement is a long-term activity, so each of the stages in the improvement process may last several months. It is also a continuous activity as, whatever new processes are introduced, the business environment will change and the new processes will themselves have to evolve to take these changes into account.

26.2 Process measurement

Process measurements are quantitative data about the software process, such as the time taken to perform some process activity. For example, you may measure the time required to develop program test cases. Humphrey (1989), in his book on process improvement, argues that the measurement of process and product attributes is essential for process improvement. He also suggests that measurement has an important role to play in small-scale personal process improvement (Humphrey, 1995), where individuals try to become more productive.

Process measurements can be used to assess whether or not the efficiency of a process has been improved. For example, the effort and time devoted to testing can be monitored. Effective improvements to the testing process should reduce the effort and/or testing time. However, process measurements on their own cannot be used to determine if product quality has improved. Product quality data (see Chapter 24) must also be collected and related to the process activities.

Three types of process metrics can be collected:

1. *The time taken for a particular process to be completed* This can be the total time devoted to the process, calendar time, the time spent on the process by particular engineers, and so on.
2. *The resources required for a particular process* Resources might include total effort in person-days, travel costs, or computer resources.
3. *The number of occurrences of a particular event* Examples of events that might be monitored include the number of defects discovered during code inspection, the number of requirements changes requested, and the average number of lines of code modified in response to a requirements change.

The first two types of measurement can be used to discover if process changes have improved the efficiency of a process. Say there are fixed points in a software

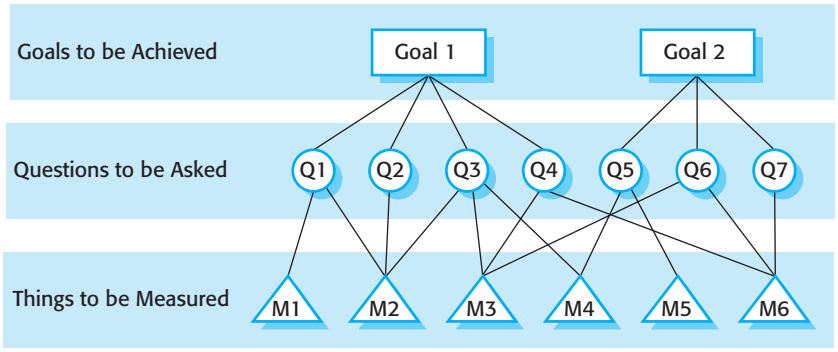

Figure 26.4 The GQM paradigm

development process, such as the acceptance of requirements, the completion of architectural design or the completion of test data generation. You may be able to measure the time and effort required to move from one of these fixed points to another. After changes have been introduced, measurements of system attributes can show if the process changes have been successful in reducing the time or effort required.

Measurements of the number of events that occur have a more direct bearing on software quality. For example, increasing the number of defects discovered by changing the program inspection process will probably be reflected in improved product quality. However, this has to be confirmed by subsequent product measurements.

A fundamental difficulty in process measurement is knowing what information about the process should be collected to support process improvement. Basili and Rombach (1988) proposed what they call the GQM (Goal-Question-Metric) paradigm, which has become widely used in software and process measurement. Basili and Green (1993) describe how this approach has been used in a long-term, measurement-based process improvement program in the U.S. space agency NASA.

The GQM paradigm (Figure 26.4) is used in process improvement to help answer three critical questions:

1. Why are we introducing process improvement?
2. What information do we need to help identify and assess improvements?
3. What process and product measurements are required to provide this information?

These questions are directly related to the abstractions (goals, questions, metrics) in the GQM paradigm:

1. *Goals* A goal is something that the organization is trying to achieve. It should not be directly concerned with process attributes but rather with how the process affects products or the organization itself. Examples of goals might be an improved

level of process maturity (see Section 26.5), shorter product development time, or increased product reliability.

2. *Questions* These are refinements of goals where specific areas of uncertainty related to the goals are identified. Normally, a goal will have a number of associated questions that need to be answered. Examples of questions related to the goal of shortening product development times might be "Where are the bottlenecks in our current process?", "How can the time required to finalize product requirements with customers be reduced?", and "How many of our tests are effective in discovering product defects?"

3. *Metrics* These are the measurements that need to be collected to help answer the questions and to confirm whether or not process improvements have achieved the desired goal. To help answer the above questions, you might collect data on the time taken to complete each process activity (normalized by system size), the number of formal communications between clients and customers for each requirements change, and the number of defects discovered per test run.

The advantage of using the GQM approach in process improvement is that it separates organizational concerns (the goals) from specific process concerns (the questions). It provides a basis for deciding what data should be collected and suggests that collected data should be analyzed in different ways, depending on the question it is intended to answer.

The GQM approach has been developed and combined with the SEI's capability maturity model (Paulk et al., 1995) in the AMI (Analyze, Measure, Improve) method of software process improvement. The developers of the AMI method propose a staged approach to process improvement, where measurement is started after an organization has introduced some standardization into its processes, as opposed to beginning measurement straight away. The AMI handbook (Pulford et al., 1996) provides guidelines and practical advice on implementing measurement-based process improvement.

As I discussed in Chapter 24, interpreting what measurements actually mean is sometimes problematic. For example, say you measure the average time taken to repair reported bugs in software that has been delivered for external testing. This is the time between the team receiving an error report and the time when this report is formally marked as 'cleared'. You introduce a new web-based tool for error reporting and, after this tool has been in use for some time, you observe that the time to repair reported bugs has been reduced.

You may then assert that the introduction of the error reporting tools has actually reduced the time to repair bugs. When you observe changes to a metric, it is always tempting to attribute these changes to the process changes that you have introduced. However, it is dangerous to make simplistic assumptions about improvements. Changes in a metric could be caused by something completely different, such as a change of the people in the project team, changes to the project

Analysis of process practice

One approach to process analysis is to use questionnaires to discover the extent to which good software engineering practices are used. Therefore, for some stage in the process, such as requirements engineering, you may identify what practices are most appropriate for the type of system being developed in a company and ask questions about how widely these are used. This is the approach that I suggested in my book on requirements engineering process improvement (Sommerville and Sawyer, 1997).

http://www.SoftwareEngineering-9.com/Web/ProcImp/goodpractice.html

schedule, or management changes. It may also be the case that the team's practice changes simply because it is being measured. In the case of the error reporting tools, some of the reasons why the change has been observed include:

1. The new system may have reduced overhead and so more time is available to repair bugs. This has led to a reduction in average 'bug repair' times. The process improvement may have made a genuine difference.
2. The new system may have made no difference to the actual time taken to fix bugs but it may have made it easier to record information. Therefore, the bug repair times are more accurately measured with the new system. There has been no actual change in the average time to fix bugs.
3. The measurements before the new system was introduced were, perhaps, made part way through the testing of a system. The bugs that were easiest and quickest to fix had already been fixed and only 'hard bugs' remained that took longer to repair. However, after the bug reporting system was introduced, measurements were made at the beginning of the testing of the new system and the bugs being fixed were the 'easy bugs', which could be repaired quickly.
4. A new manager of the testing team may have instructed team members to report user interface inconsistencies as bugs, whereas before these were ignored. This meant that many more 'easy bugs' were reported that could be fixed quickly.

Measurement is a way of generating evidence about a process and process changes. However, this evidence has to be interpreted along with other information about the process before you can be sure that process changes are effective. You should always use measurement in conjunction with qualitative assessment of changes. This involves talking to the people involved in the process about the changes that have been introduced and getting their impression of the effectiveness of these changes. Not only does this reveal other factors that may have influenced the process, it reveals the extent to which the team has adopted the proposed changes and how these have affected actual development practice.

26.3 Process analysis

Process analysis is the study of processes to help understand their key characteristics and how such processes are performed in practice by the people involved. I have suggested in Figure 26.3 that process analysis follows process measurement. This is a simplification because, in reality, these activities are intertwined. You need to carry out some analysis to know what to measure, and, when making measurements, you inevitably develop a deeper understanding of the process being measured.

Process analysis has a number of closely related objectives:

1. To understand the activities involved in the process and the relationships between these activities.
2. To understand the relationships between the process activities and the measurements that have been made.
3. To relate the specific process or processes that you are analyzing to comparable processes elsewhere in the organization, or to idealized processes of the same type.

During process analysis you are trying to understand what is going on in a process. You are looking for information about that process's problems and inefficiencies. You should also be interested in the extent that the process is used, the software tools used to support the process, and how the process is influenced by organizational constraints. Figure 26.5 shows some of the aspects of the process that you may investigate during process analysis.

The most commonly used techniques of process analysis are:

1. *Questionnaires and interviews* The engineers and managers working on a project are questioned about what actually goes on. The answers to a formal questionnaire are refined during personal interviews with those involved in the process. As I discuss below, the discussion may be structured around models of software processes.
2. *Ethnographic studies* Ethnographic studies (see Chapter 4), where process participants are observed as they work, may be used to understand the nature of software development as a human activity. Such analysis reveals subtleties and complexities that may not be revealed by questionnaires and interviews.

Each of these approaches has advantages and disadvantages. Questionnaire-based analysis can be carried out fairly quickly once the right questions have been identified. However, if the questions are badly worded or inappropriate, you may end up with an incomplete or inaccurate understanding of the process. Furthermore, questionnaire-based analysis may appear to them as a form of assessment or appraisal. The engineers being questioned may therefore give the answers that they think you want to hear rather than the truth about the process used.

Process aspect	Questions
Adoption and standardization	Is the process documented and standardized across the organization? If not, does this mean that any measurements made are specific only to a single process instance? If processes are not standardized, then changes to one process may not be transferable to comparable processes elsewhere in the company.
Software engineering practice	Are there known, good software engineering practices that are not included in the process? Why are they not included? Does the lack of these practices affect product characteristics, such as the number of defects in a delivered software system?
Organizational constraints	What are the organizational constraints that affect the process design and the ways that the process is performed? For example, if the process involves dealing with classified material, there may be activities in the process to check that classified information is not included in any material due to be released to external organizations. Organizational constraints may mean that possible process changes cannot be made.
Communications	How are communications managed in the process? How do communication issues relate to the process measurements that have been made? Communication problems are a major issue in many processes and communication bottlenecks are often the reasons for project delays.
Introspection	Is the process reflective (i.e., do the actors involved in the process explicitly think about and discuss the process and how it might be improved)? Are there mechanisms through which process actors can propose process improvements?
Learning	How do people joining a development team learn about the software processes used? Does the company have process manuals and process training programs?
Tool support	What aspects of the process are and aren't supported by software tools? For unsupported areas, are there tools that could be deployed cost-effectively to provide support? For supported areas, are the tools effective and efficient? Are better tools available?

Figure 26.5 Aspects of process analysis

Interviews with the people involved in the process are more open-ended than questionnaires. You start with a prepared script of questions but adapt these according to the responses that you get from different people. If you give participants an opportunity to discuss issues more widely, you may find that the process participants talk about process problems, the ways that the process is changed in practice, and so on.

In almost all processes, the people involved make local changes to adapt the process to suit local circumstances. Ethnographic analysis is more likely than interviews to discover the true process used. However, this type of analysis can be a prolonged activity that can last several months. It relies on external observation of the process as it is being enacted. To do a complete analysis, you have to be involved from the initial stages of a project through to product delivery and maintenance. For

large projects, this could take several years, so it is clearly impractical to do a complete ethnographic analysis of the processes in a large project. Ethnographic analysis is actually most useful when an in-depth understanding of process fragments is required. Once you identify areas that need further investigation from interview material, you can then do a focused ethnographic study to discover process details.

When analyzing a process, it is often useful to start with a process model that defines the activities in the process and the inputs and outputs of these activities. The model may also include information about the process actors—the people or roles responsible for performing activities, and the critical deliverables that must be produced. You can use an informal notation to describe process models or more formal tabular notations, UML activity diagrams, or a business process modeling notation such as BPMN (discussed in Chapter 19). There are lots of examples of process models in this book that I use to present and describe software processes.

Process models are a good way of focusing attention on the activities in a process and the information transfer between these activities. These process models do not have to be formal or complete—their purpose is to provoke discussion rather than document the process in detail. Discussion with people involved in the process and observations of that process are often structured around a set of questions about the formal process model. Examples of these questions might be:

1. What activities take place in practice but are not shown in the model? Inevitably, models are incomplete but if different people identify different missing activities, this tells you that the process is not performed consistently across the organization.
2. Are there process activities, shown in the model, that you (the process actor) think are inefficient? In what ways are they inefficient and how might these be improved? How do these inefficient activities affect process measurements that may have been made?
3. What happens when things go wrong? Does the team continue to follow the process defined in the model, or is the process abandoned and emergency action taken? If the process is abandoned, this suggests that the software engineers do not believe that the process is good enough or that it does not have sufficient flexibility to handle exceptions.
4. Who are the actors involved at different stages in the process and how do they communicate? What bottlenecks commonly occur in the information exchange?
5. What tool support is used for the activities shown in the model? Is this effective and universally used? How could tool support be improved?

When you have completed an analysis of the software process, you should have a deeper understanding of that process and the potential for process improvements in the future. You should also understand the constraints on process improvement and how these can limit the scope of improvements that may be introduced.

Software process modeling

Software process modeling started in the early 1980s (Osterweil, 1987) with the long-term objective of using process models as a way of organizing and coordinating tool support for the process. A process model should include information about process activities, inputs and outputs, and the actors involved in the process. Special-purpose software process modeling notations were developed but these have been largely supplanted by notations for business process modeling such as BPMN (White, 2004) or UML activity models.

http://www.SoftwareEngineering-9.com/Web/ProcImp/spm.html

26.3.1 Process exceptions

Software processes are complex entities. There may be a defined process model in an organization but this can only represent the situation where the development team is not faced with any unanticipated problems. In reality, unanticipated problems are a fact of everyday life for project managers. The 'ideal' process model must be modified dynamically as solutions to these problems are found. Examples of the kinds of exception that a project manager may have to deal with include:

- several key people becoming ill at the same time, just before a critical project review;
- a serious breach in computer security that means all external communications are out of action for several days;
- a company reorganization, which means that managers have to spend much of their time working on organizational matters rather than on project management;
- an unanticipated request to write a proposal for a new project that means effort must be transferred from the current project to proposal writing.

Essentially, an exception will affect and usually alter, in some way, the resources, budgets, or schedules of a project. It is difficult to predict all exceptions in advance and to incorporate them into a formal process model. You therefore often have to work out how to handle exceptions and then dynamically change the 'standard' process to cope with these unexpected circumstances.

26.4 Process change

Process change involves making modifications to the existing process. As I have suggested, you may do this by introducing new practices, methods, or tools; changing the ordering of process activities; introducing or removing deliverables from the process; improving communications; or by introducing new roles and responsibilities. Process

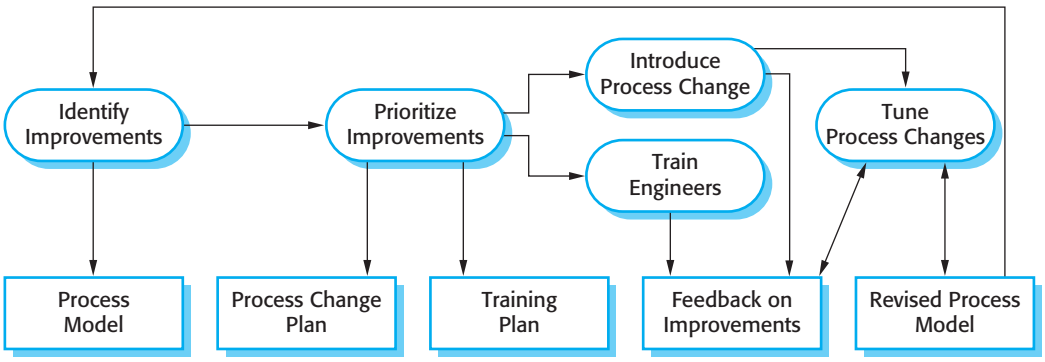

Figure 26.6 The process change process

changes should be driven by improvement goals such as 'reduce the number of defects discovered during integration testing by 25 percent'. After the changes have been implemented, you use the process measurements to assess the effectiveness of the changes.

There are five key stages in the process change process (Figure 26.6):

1. *Improvement identification* This stage is concerned with using the results of the process analysis to identify ways to tackle quality problems, schedule bottlenecks, or cost inefficiencies that have been identified during process analysis. You may propose new processes, process structures, methods, and tools to address process problems. For example, a company may believe that many of its software problems stem from requirements problems. Using a requirements engineering best practice guide (Sommerville and Sawyer, 1997), various requirements engineering practices that could be introduced or changed may then be identified.

2. *Improvement prioritization* This stage is concerned with assessing possible changes to the process, and prioritizing them for implementation. When many possible changes have been identified, it is usually impossible to introduce them all at once, and you must decide which are the most important. You may make these decisions based on the need to improve specific process areas, the costs of introducing a change, the impact of a change on the organization, or other factors. For example, a company may consider the introduction of requirements management processes to manage evolving requirements to be the highest-priority process change.

3. *Process change introduction* Process change introduction means putting new procedures, methods, and tools into place and integrating them with other process activities. You must allow enough time to introduce changes and ensure that these changes are compatible with other process activities and organizational procedures and standards. This may involve acquiring tools for requirements management and designing processes to use these tools.

4. *Process training* Without training, it is not possible to gain the full benefits of process changes. The engineers involved need to understand the changes that have been proposed and how to perform the new and changed processes. All too often, process changes are imposed without adequate training and the effect of these changes is to degrade rather than improve product quality. In the case of requirements management, the training might involve a discussion of the value of requirements management, an explanation of the process activities, and an introduction to the tools that have been selected.
5. *Change tuning* Proposed process changes will never be completely effective as soon as they are introduced. You need a tuning phase where minor problems can be discovered, and modifications to the process can be proposed and introduced. This tuning phase should last for several months until the development engineers are happy with the new process.

It is generally unwise to introduce too many changes at the same time. Apart from the training difficulties that this causes, introducing too many changes makes it impossible to assess the effect of each change on the process. Once a change has been introduced, the improvement process can iterate, with further analysis used to identify process problems, propose improvements, and so on.

As well as the difficulties of assessing the effectiveness of changed processes that I have discussed, there are two major difficulties that those involved in change processes may have to face:

1. *Resistance to change* Team members or project managers may resist the introduction of process changes and propose reasons why changes will not work, or delay the introduction of changes. They may, in some cases, deliberately obstruct process changes and interpret data to show the ineffectiveness of proposed process changes.
2. *Change persistence* Although it may be possible to introduce process changes initially, it is common for process innovations to be discarded after a short time and for the processes to revert to their previous state.

Resistance to change may come from both project managers and the engineers involved in the process that is being changed. Project managers often resist process change because any innovation has unknown risks associated with it. Process changes may be intended to speed up software production or reduce software defects. However, there is always the danger that these process changes will be ineffective or that the time required to introduce the changes will be longer than the time saved. Project managers are judged according to whether or not their project produces software on time and to budget. Therefore, they may prefer an inefficient but predictable process to an improved process that has organizational benefits, but which may also have short-term risks associated with it.

Engineers may resist the introduction of new processes for similar reasons, or because they see these processes as threatening their professionalism. That is, they

may feel that the new predefined process gives them less discretion and does not recognize the value of their skills and experience. They may think that the new process will mean that fewer people will be required and that they may lose their job. They may not wish to learn new skills, tools, or ways of working.

As a manager, you have to be sensitive to the feelings of the people affected when introducing process change. You have to involve the team all the way through the change process, understand their doubts, and involve them in planning the new process. By making them stakeholders in the process change, it is much more likely that they will want to make it work. Business process reengineering (Hammer, 1990; Ould, 1995), a fashion of the 1990s which involved making radical process changes, was largely unsuccessful because it failed to take the legitimate concerns of the people involved into account.

To address the concerns of project managers that process change will adversely affect project schedules and costs, you have to increase project budgets to allow for additional costs and delays resulting from the change. You also have to be realistic about the short-term benefits of the change. Changes are unlikely to lead to large-scale, immediate improvements. The benefits of process change are long term rather than short term so you have to support the process changes over several projects.

The problem of changes being introduced then subsequently discarded is a common one. Changes may be proposed by an 'evangelist' who believes strongly that the changes will lead to improvement. He or she may work hard to ensure the changes are effective and the new process is accepted. However, if the 'evangelist' leaves, then he or she may be replaced by someone who is less committed to the new process. The people involved may therefore simply revert to the previous ways of doing things. This is particularly likely if the changes that have been introduced have not been universally adopted and the full benefits of the process changes have not yet been realized.

Because of problems of change persistence, the CMMI model, discussed in Section 26.5, argues strongly for the institutionalization of process change. This means that process change is not dependent on individuals but that the changes become part of standard practice in the company, with company-wide support and training.

26.5 The CMMI process improvement framework

The U.S. Software Engineering Institute (SEI) was established to improve the capabilities of the American software industry. In the mid-1980s, the SEI initiated a study of ways to assess the capabilities of software contractors. The outcome of this capability assessment was the SEI Software Capability Maturity Model (CMM) (Paulk et al., 1993; Paulk et al., 1995). This has been tremendously influential in convincing the software engineering community to take process improvement seriously. The Software CMM was followed by a range of

other capability maturity models, including the People Capability Maturity Model (P-CMM) (Curtis et al., 2001) and the Systems Engineering Capability Model (Bate, 1995).

Other organizations have also developed comparable process maturity models. The SPICE approach to capability assessment and process improvement (Paulk and Konrad, 1994) is more flexible than the SEI model. It includes maturity levels comparable with the CMM levels, but also identifies processes, such as customer-supplier processes, that cut across these levels. As the level of maturity increases, the performance of these cross-cutting processes must also improve.

The Bootstrap project in the 1990s had the goal of extending and adapting the SEI maturity model to make it applicable across a wider range of companies. This model (Haase et al., 1994; Kuvaja, et al., 1994) uses the SEI's maturity levels (discussed in Section 26.5.1). It also proposes a base process model (based on the model used in the European Space Agency) that may be used as a starting point for local process definition. It includes guidelines for developing a company-wide quality system to support process improvement.

In an attempt to integrate the plethora of capability models based on the notion of process maturity (including its own models), the SEI embarked on a new program to develop an integrated capability model (CMMI). The CMMI framework supersedes the Software and Systems Engineering CMMs and integrates other capability maturity models. It has two instantiations, staged and continuous, and addresses some of the reported weaknesses in the Software CMM.

The CMMI model (Ahern et al., 2001; Chrissis et al., 2007) is intended to be a framework for process improvement that has broad applicability across a range of companies. Its staged version is compatible with the Software CMM and allows an organization's system development and management processes to be assessed and assigned a maturity level from 1 to 5. Its continuous version allows for a finer-grain classification of process maturity. This model provides a way of rating 22 process areas (see Figure 26.7) on a scale from 0 to 5.

The CMMI model is very complex, with more than 1,000 pages of description. I have radically simplified it for discussion here. The principal model components are:

1. A set of process areas that are related to software process activities. The CMMI identifies 22 process areas that are relevant to software process capability and improvement. These are organized into four groups in the continuous CMMI model. These groups and related process areas are listed in Figure 26.7.

2. A number of goals, which are abstract descriptions of a desirable state that should be attained by an organization. The CMMI has specific goals that are associated with each process area and define the desirable state for that area. It also defines generic goals that are associated with the institutionalization of good practice. Figure 26.8 shows examples of specific and generic goals in the CMMI.

3. A set of good practices, which are descriptions of ways of achieving a goal. Several specific and generic practices may be associated with each goal within a process

Category	Process area
Process management	Organizational process definition (OPD)
	Organizational process focus (OPF)
	Organizational training (OT)
	Organizational process performance (OPP)
	Organizational innovation and deployment (OID)
Project management	Project planning (PP)
	Project monitoring and control (PMC)
	Supplier agreement management (SAM)
	Integrated project management (IPM)
	Risk management (RSKM)
	Quantitative project management (QPM)
Engineering	Requirements management (REQM)
	Requirements development (RD)
	Technical solution (TS)
	Product integration (PI)
	Verification (VER)
	Validation (VAL)
Support	Configuration management (CM)
	Process and product quality management (PPQA)
	Measurement and analysis (MA)
	Decision analysis and resolution (DAR)
	Causal analysis and resolution (CAR)

Figure 26.7 Process areas in the CMMI

area. Some examples of recommended practices are shown in Figure 26.9. However, the CMMI recognizes that it is the goal rather than the way that the goal is reached that is important. Organizations may use any appropriate practices to achieve any of the CMMI goals—they do not have to adopt the practices recommended in the CMMI.

Generic goals and practices are not technical but are associated with the institutionalization of good practice. What this means depends on the maturity of the organization. At an early stage of maturity development, institutionalization may mean ensuring that plans are established and processes are defined for all software development in the company. However, for an organization with more mature, advanced processes, institutionalization may mean introducing process control using statistical and other quantitative techniques across the organization.

Goal	Process area
Corrective actions are managed to closure when the project's performance or results deviate significantly from the plan.	Project monitoring and control (specific goal)
Actual performance and progress of the project are monitored against the project plan.	Project monitoring and control (specific goal)
The requirements are analyzed and validated, and a definition of the required functionality is developed.	Requirements development (specific goal)
Root causes of defects and other problems are systematically determined.	Causal analysis and resolution (specific goal)
The process is institutionalized as a defined process.	Generic goal

Figure 26.8 Process areas in the CMMI

A CMMI assessment involves examining the processes in an organization and rating these processes or process areas on a six-point scale that relates to the level of maturity in each process area. The idea is that the more mature a process, the better it is. The six-point scale assigns a level of maturity to a process area as follows:

1. *Incomplete* At least one of the specific goals associated with the process area is not satisfied. There are no generic goals at this level as institutionalization of an incomplete process does not make sense.
2. *Performed* The goals associated with the process area are satisfied, and for all processes the scope of the work to be performed is explicitly set out and communicated to the team members.
3. *Managed* At this level, the goals associated with the process area are met and organizational policies are in place that define when each process should be used. There must be documented project plans that define the project goals. Resource management and process monitoring procedures must be in place across the institution.
4. *Defined* This level focuses on organizational standardization and deployment of processes. Each project has a managed process that is adapted to the project requirements from a defined set of organizational processes. Process assets and process measurements must be collected and used for future process improvements.
5. *Quantitatively managed* At this level, there is an organizational responsibility to use statistical and other quantitative methods to control subprocesses; that is, collected process and product measurements must be used in process management.
6. *Optimizing* At this highest level, the organization must use the process and product measurements to drive process improvement. Trends must be analyzed and the processes adapted to changing business needs.

Goal	Associated practices
The requirements are analyzed and validated, and a definition of the required functionality is developed.	Analyze derived requirements systematically to ensure that they are necessary and sufficient.
	Validate requirements to ensure that the resulting product will perform as intended in the user's environment, using multiple techniques as appropriate.
Root causes of defects and other problems are systematically determined.	Select the critical defects and other problems for analysis.
	Perform causal analysis of selected defects and other problems and propose actions to address them.
The process is institutionalized as a defined process.	Establish and maintain an organizational policy for planning and performing the requirements development process.
	Assign responsibility and authority for performing the process, developing the work products, and providing the services of the requirements development process.

Figure 26.9 Goals and associated practices in the CMMI

This is a very simplified description of the capability levels and, to put these into practice, you need to work with more detailed descriptions. The levels are progressive, with explicit process descriptions at the lowest levels, through process standardization, to process change and improvement driven by measurements of the process and the software at the highest level. To improve its processes, a company should aim to increase the maturity level of the process groups that are relevant to its business.

26.5.1 The staged CMMI model

The staged CMMI model is comparable with the Software Capability Maturity Model in that it provides a means to assess an organization's process capability at one of five levels, and prescribes the goals that should be achieved at each of these levels. Process improvement is achieved by implementing practices at each level, moving from the lower to the higher levels in the model.

The five levels in the staged CMMI model are shown in Figure 26.10. They correspond to capability levels 1 to 5 in the continuous model. The key difference between the staged and the continuous CMMI models is that the staged model is used to assess the capability of the organization as a whole, whereas the continuous model measures the maturity of specific process areas within the organization.

Each maturity level has an associated set of process areas and generic goals. These reflect good software engineering and management practice and the institutionalization of process improvement. The lower maturity levels may be achieved by

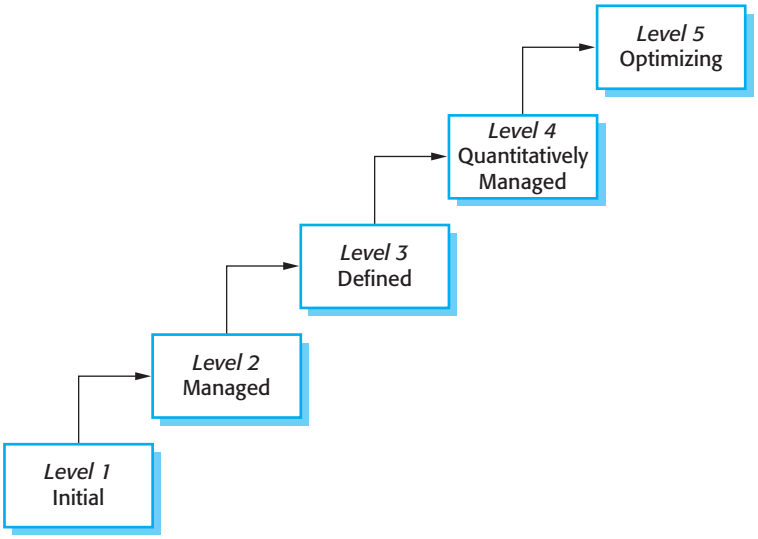

Figure 26.10 The CMMI staged maturity model

introducing good practice; however, higher levels require a commitment to process measurement and improvement.

For example, the process areas as defined in the model associated with the second level (the managed level) are:

1. *Requirements management* Manage the requirements of the project's products and product components, and identify inconsistencies between those requirements and the project's plans and work products.
2. *Project planning* Establish and maintain plans that define project activities.
3. *Project monitoring and control* Provide understanding into the project's progress so that appropriate corrective actions can be taken when the project's performance deviates significantly from the plan.
4. *Supplier agreement management* Manage the acquisition of products and services from suppliers external to the project for which a formal agreement exists.
5. *Measurement and analysis* Develop and sustain a measurement capability that is used to support management information needs.
6. *Process and product quality assurance* Provide staff and management with objective insight into the processes and associated work products.
7. *Configuration management* Establish and maintain the integrity of work products using configuration identification, configuration control, configuration status accounting, and configuration audits.

As well as these specific practices, organizations operating at the second level in the CMMI model should have achieved the generic goal of institutionalizing each of

the processes as a managed process. Examples of institutional practices associated with project planning that lead to the project planning process being a managed process are:

- Establish and maintain an organizational policy for planning and performing the project planning process.
- Provide adequate resources for performing the project management process, developing the work products, and providing the services of the process.
- Monitor and control the project planning process against the plan and take appropriate corrective action.
- Review the activities, status, and results of the project planning process with high-level management, and resolve any issues.

The advantage of the staged CMMI is that it is compatible with the software capability maturity model that was proposed in the late 1980s. Many companies understand and are committed to using this model for process improvement. It is therefore straightforward for them to make a transition from this to the staged CMMI model. Furthermore, the staged model defines a clear improvement pathway for organizations. They can plan to move from the second to the third level and so on.

The major disadvantage of the staged model (and of the Software CMM), however, is its prescriptive nature. Each maturity level has its own goals and practices. The staged model assumes that all of the goals and practices at one level are implemented before the transition to the next level. However, organizational circumstances may be such that it more appropriate to implement goals and practices at higher levels before lower-level practices. When an organization does this, a maturity assessment will give a misleading picture of its capability.

26.5.2 The continuous CMMI model

Continuous maturity models do not classify an organization according to discrete levels. Rather, they are finer-grained models that consider individual or groups of practices and assess the use of good practice within each process group. The maturity assessment is not, therefore, a single value but a set of values showing the organization's maturity for each process or process group.

The continuous CMMI considers the process areas shown in Figure 26.7 and assigns a capability assessment level from 0 to 5 (as described earlier) to each process area. Normally, organizations operate at different maturity levels for different process areas. Consequently, the result of a continuous CMMI assessment is a capability profile showing each process area and its associated capability assessment. A fragment of a capability profile that shows processes at different capability levels is shown in Figure 26.11. This shows that the level of maturity in configuration management, for example, is high, but that risk management maturity is low. A company may develop

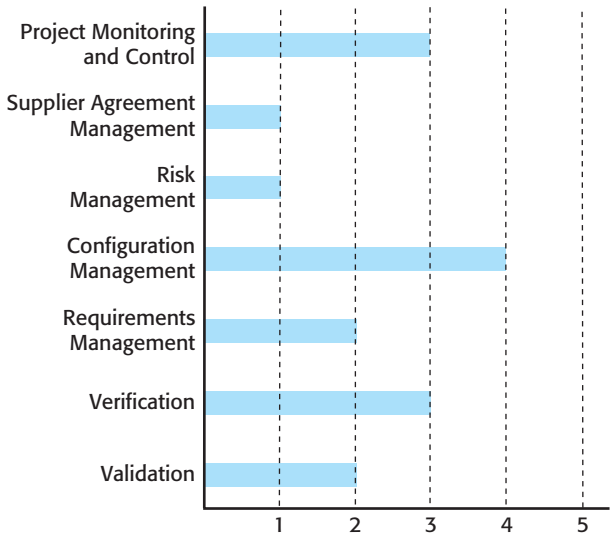

Figure 26.11 A process capability profile

actual and target capability profiles where the target profile reflects the capability level that they would like to reach for that process area.

The principal advantage of the continuous model is that companies can pick and choose processes for improvement according to their own needs and requirements. In my experience, different types of organization have different requirements for process improvement. For example, a company that develops software for the aerospace industry may focus on improvements in system specification, configuration management, and validation, whereas a web development company may be more concerned with customer-facing processes. The staged model requires companies to focus on the different stages in turn. By contrast, the continuous CMMI permits discretion and flexibility, while still allowing companies to work within the CMMI improvement framework.

KEY POINTS

- The goals of process improvement are higher product quality, reduced process costs, and faster delivery of software.
- The principal approaches to process improvement are agile approaches, geared to reducing process overheads, and maturity-based approaches based on better process management and the use of good software engineering practice.

- The process improvement cycle involves process measurement, process analysis and modeling, and process change.
- Process models, which show the activities in a process and their relationships with software products, are used for process description. In practice, however, engineers involved in software development always adapt models to their local circumstances.
- Measurement should be used to answer specific questions about the software process used. These questions should be based on organizational improvement goals.
- Three types of process metrics used in the measurement process are time metrics, resource utilization metrics, and event metrics.
- The CMMI process maturity model is an integrated process improvement model that supports both staged and continuous process improvement.
- Process improvement in the CMMI model is based on reaching a set of goals related to good software engineering practice and describing, standardizing, and controlling the practices used to achieve these goals. The CMMI model includes recommended practices that may be used, but these are not obligatory.

FURTHER READING

'Can you trust software capability evaluations?' This article takes a skeptical look at the subject of capability evaluation, where a company's process maturity is assessed, and discusses why these evaluations may not give a true picture of an organization's maturity. (E. O'Connell and H. Saiedian, *IEEE Computer*, **33** (2), February 2000) http://dx.doi.org/10.1109/2.820036.

Software Process Improvement: Results and Experience from the Field. This book is a collection of papers focusing on process improvement case studies in several small and medium-sized Norwegian companies. It also includes a good introduction to the general issues of process improvement. (Conradi, R., Dybå, T., Sjøberg, D., Ulsund, T. (eds.), Springer, 2006.)

CMMI: Guidelines for Process Integration and Product Improvement, 2nd edition. A comprehensive description of the CMMI. The CMMI is large and complex and it is practically impossible to make it easy to read and understand. This book does a reasonable job by including some anecdotal and historical material, but it's still sometimes tough going. (M. B. Chrissis, M. Konrad, S. Shrum, Addison-Wesley, 2007.)

EXERCISES

26.1. What are the important differences between the agile approach and the process maturity approach to software process improvement?

26.2. Under what circumstances is product quality likely to be determined by the quality of the development team? Give examples of the types of software product that are particularly dependent on individual talent and ability.

26.3. Suggest three specialized software tools that might be developed to support a process improvement program in an organization.

26.4. Assume that the goal of process improvement in an organization is to increase the number of reusable components that are produced during development. Suggest three questions in the GQM paradigm that this might lead to.

26.5. Describe three types of software process metric that may be collected as part of a process improvement process. Give one example of each type of metric.

26.6. Design a process for assessing and prioritizing process change proposals. Document this process as a process model showing the roles involved in this process. You should use UML activity diagrams or BPMN to describe the process.

26.7. Give two advantages and two disadvantages of the approach to process assessment and improvement that is embodied in the process improvement frameworks such as the CMMI.

26.8. Under what circumstances would you recommend the use of the staged representation of the CMMI?

26.9. What are the advantages and disadvantages of using a process maturity model that focuses on goals to be achieved, rather than good practices to be introduced?

26.10. Do you think that process improvement programs, which involve measuring the work of people in the process and introducing changes into that process, can be inherently dehumanizing? What resistance to a process improvement program might arise and why?

REFERENCES

Ahern, D. M., Clouse, A. and Turner, R. (2001). *CMMI Distilled*. Reading, Mass.: Addison-Wesley.

Basili, V. and Green, S. (1993). 'Software Process Improvement at the SEL'. *IEEE Software,* **11** (4), 58–66.

Basili, V. R. and Rombach, H. D. (1988). 'The TAME Project: Towards Improvement-Oriented Software Environments'. *IEEE Trans. on Software Eng.*, **14** (6), 758–773.