25 Process improvement

Objectives

The objective of this chapter is to explain how software processes can be improved to produce better software. When you have read this chapter, you will:

- understand the principles of software process improvement and why process improvement is worthwhile;
- understand how software process factors influence software quality and the productivity of software developers;
- understand the SEI's Process Capability Maturity Model (CMM) which may be used to assess the quality of the software process in large organisations;
- understand why CMM-based improvement is not applicable to all types of software process.

Contents

Over the past few years, there has been a great deal of interest in the software engineering community in process improvement. Process improvement means understanding existing processes and changing these processes to improve product quality and/or reduce costs and development time. Most of the literature on process improvement has focused on improving processes to improve product quality and, in particular, to reduce the number of defects in delivered software. Once this has been achieved cost or schedule reduction might become the principal improvement goals.

As discussed in Chapter 24, there is a strong relationship between the quality of the developed software product and the quality of the software process used to create that product. By improving the software process, it is hoped that the related product quality is correspondingly enhanced. Software processes are inherently complex and involve a very large number of activities. Like products, processes also have attributes or characteristics as shown in Figure 25.1.

It is not possible to make process improvements that optimise all process attributes simultaneously. For example, if a rapid development process is required then it may be necessary to reduce the process visibility. Making a process visible means producing documents at regular intervals. This will slow down the process.

Process improvement does not simply mean adopting particular methods or tools or using some model of a process that has been used elsewhere. Although organisations which develop the same type of software clearly have much in common, there are always local organisational factors, procedures and standards which influence the process. The simple introduction of published process improvements is

Figure 25.1 Process characteristics

Process characteristic	Description
Understandability	To what extent is the process explicitly defined and how easy is it to understand the process definition?
Visibility	Do the process activities culminate in clear results so that the progress of the process is externally visible?
Supportability	To what extent can the process activities be supported by CASE tools?
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 organisational requirements or identified process improvements?
Rapidity	How fast can the process of delivering a system from a given specification be completed?

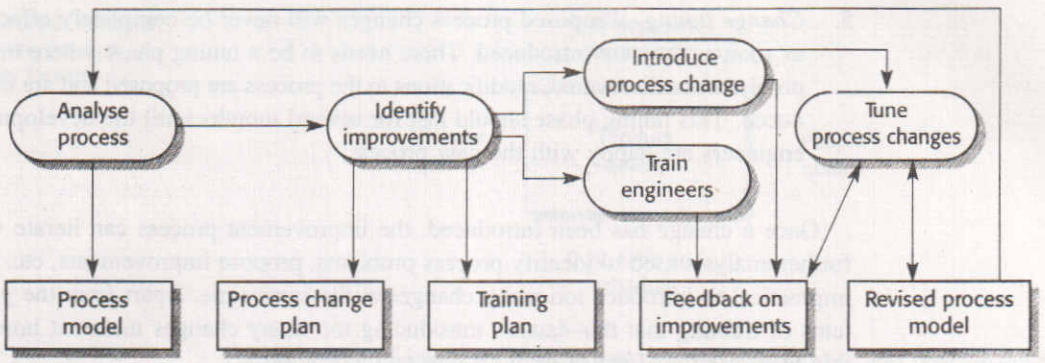


Figure 25.2 The process improvement process

unlikely to be successful. Process improvement should always be seen as an activity that is specific to an organisation or a part of a larger organisation.

A generic model of the process improvement process is illustrated in Figure 25.2. Process improvement is a long-term, iterative process. Each of the activities shown in Figure 25.2 might last several months. Successful process improvement requires organisational commitment and resources. It must be sanctioned by senior management and must have an associated budget to support improvements.

There are a number of key stages in the process improvement process:

1. *Process analysis* Process analysis involves examining existing processes and producing a process model to document and understand the process. In some cases, it may be possible to analyse the process quantitatively Measurements made during the analysis add extra information to the process model. Quantitative analysis before and after changes have been introduced allows an objective assessment of the benefits (or the problems) of the process change.

2. *Improvement identification* This stage is concerned with using the results of the process analysis to identify quality, schedule or cost bottlenecks where process factors might adversely influence the product quality. Process improvement should focus on loosening these bottlenecks by proposing new procedures, methods and tools to address the problems.

3. *Process change introduction* Process change introduction means putting new procedures, methods and tools into place and integrating them with other process activities. It is important to allow enough time to introduce changes and to ensure that these changes are compatible with other process activities and with organisational procedures and standards.

4. *Process change training* Without training, it is not possible to gain the full benefits from process changes. They may be rejected by the managers and engineers responsible for development projects. All too commonly, process changes have been imposed without adequate training and the effects of these changes have been to degrade rather than improve product quality.

5. *Change tuning* Proposed process changes will never be completely effective as soon as they are introduced. There needs to be a tuning phase where minor problems are discovered, modifications to the process are proposed and are introduced. This tuning phase should last for several months until the development engineers are happy with the new process.

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

25.1 Process and product quality

As discussed in Chapter 24, process improvement is based on the assumption that the critical factor that influences product quality is the quality of the product development process. These ideas of process improvement stemmed from the work of an American engineer called W. E. Deming who worked with Japanese industry after World War 2 to improve quality. Japanese industry has been committed to a process of continuous process improvement (the Japanese word for this is *kaizen*) for many years. This has largely contributed to the generally 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 process is improved with the aim of reducing the number of product defects. The process is improved until it is repeatable. That is, the results of the process are predictable and the number of defects reduced. It is then standardised and a further improvement cycle begins.

Where manufacturing is involved, the process/product relationship is very obvious. Improving a process so that defects are avoided will lead to better products. This link is less obvious when the product is intangible and dependent, to some extent, on intellectual processes which cannot be automated. Software quality is not dependent on a manufacturing process but on a design process where individual human considerations are significant. In some classes of product, the process used is the most significant determinant of product quality. However, for innovative applications in particular, the people involved in the process may be more important than the process used.

For software products (or any other intellectual products such as books, films, etc. where quality principally depends on the design), there are four factors that can affect product quality. These are shown in Figure 25.3.

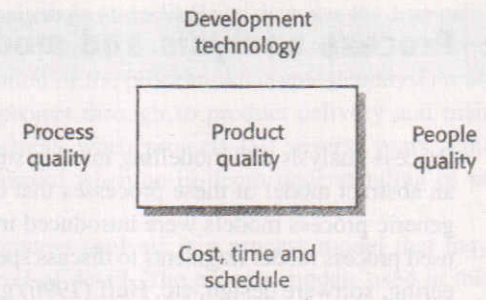


Figure 25.3 Principal software product quality factors

The influence of each of these factors depends on the size and type of the project. For very large systems that are composed of separate sub-systems, developed by different teams, the principal determinant of 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, as 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. Therefore, particularly skilled or talented individuals don't usually have a dominant effect 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. If the team has a high level of ability and experience, the quality of the product is likely to be high. 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 team cannot devote a lot of time to tedious administrative procedures. Engineers spend much of their time designing and programming the system, so good tools can significantly affect their productivity. For large projects, a basic level of development technology is essential for information management. Paradoxically, however, sophisticated tools are less important. Team members spend a relatively smaller proportion of their time in development activities and more time communicating and understanding other parts of the system. This is the dominant factor affecting their productivity. Development tools make no difference to this.

The base of the rectangle in Figure 25.3 is absolutely critical. If a project, irrespective of size, is under-budgeted or planned with an unrealistic delivery schedule, the product quality will be affected. A good process requires resources for its effective implementation. If these resources are insufficient, the process cannot work effectively. 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.

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 organisations must compete to survive. Many software projects are deliberately under-budgeted in order to win a development contract. Pricing to win (see Chapter 23) is an inevitable consequence of a competitive system. It is not surprising that product quality under such a system is difficult to control.

25.2 Process analysis and modelling

Process analysis and modelling involve studying existing processes and developing an abstract model of these processes that captures their key characteristics. Several generic process models were introduced in Chapter 1. Throughout the book, I have used process model fragments to discuss specific activities such as requirements engineering, software design, etc. Huff (1996) gives a good overview of the general topic of software process modelling.

Process analysis is concerned with studying existing processes to understand the relationships between different parts of the process. The initial stages of process analysis are inevitably qualitative where the analyst is simply trying to discover the key features of the model. Later stages may be more quantitative. The process is analysed in more detail using various metrics that are automatically or manually collected. After analysis, the process may be described using a process model.

The starting point for process analysis should be whatever 'formal' process model is used. Many organisations have such a formal model which may be imposed on them by the software customer. This standard defines the critical activities and life-cycle deliverables which must be produced.

Formal models can serve as a useful starting point for process analysis. However, they rarely include enough detail or reflect the real activities of software development. Formal process models are fairly abstract and only define the principal process activities and deliverables. It is usually necessary to 'look inside' the model to find the real process that is being enacted. Furthermore, the actual process which is followed often differs significantly from the formal model, although the specified deliverables will usually be produced.

Techniques of process analysis include:

1. *Questionnaires and interviews* The engineers 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.
2. *Ethnographic studies* As discussed in Chapter 6, ethnographic studies may be used to understand the nature of software development as a human activity. Such analysis reveals subtleties and complexities which may not be discovered using other techniques.

Each of these approaches has advantages and disadvantages. Questionnaire-based analysis can be carried out fairly quickly once the right questions have been discovered. However, if the questions are badly worded or inappropriate, you will end up with an incomplete or inaccurate model of the process. Furthermore, questionnaire-based analysis appears like a form of assessment. The engineers being questioned may give the answers which they think the questioner wants to hear rather than the truth about the process used.

Ethnographic analysis is more likely to discover the true process used. However, is a prolonged and expensive activity which can last for several months. It relies on external observation of the process. A complete analysis must continue from the initial stages of a project through to product delivery and maintenance. This will probably be impractical, when projects last several years. Ethnographic analysis, therefore, is most useful when an in-depth understanding of process fragments is required.

The output of process analysis is a process model that may be expressed at a greater or lesser level of detail. The process models used in this book are abstract, simplified models which present a generic view of the processes concerned. At this abstract level, these processes are the same in many different organisations. However, these generic models have different instantiations depending on the type of software being developed and the organisational environment. Detailed process models are not usually transferable from one organisation to another.

Generic process models are a useful basis for discussing processes. However, they do not include enough information for process analysis and improvement. Process improvement needs information about activities, deliverables, people, communications, schedules and other organisational processes that affect the software development process. Figure 25.4 explains what might be included in a detailed process model.

The timing of and the dependencies between activities, deliverables and communications must also be recorded in a process model. Sometimes activities can be carried out in parallel and sometimes in sequence. They may be interleaved so that the same engineer is involved in several activities. Deliverables may be dependent on other deliverables or on some communications between engineers working on the process.

In the examples of process models in this book, I show the approximate sequence of activities from left to right. Activities that may be carried out in parallel are, as far as possible, aligned vertically.

Detailed process models are extremely complex. It is very difficult to construct a single model that incorporates all of the above elements. To illustrate the complexity of a model, consider the process fragments below. These describe the process of testing a single module in a large system which uses a strictly controlled configuration management process (see Chapter 29). The software being tested and the test data are under configuration control. Figure 25.5 shows the role responsible for the testing process, process inputs and outputs and pre- and post-conditions.

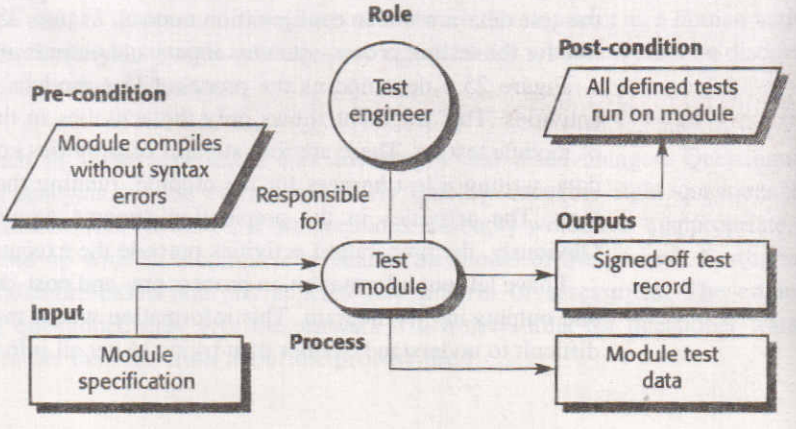
Figure 25.6 decomposes the process 'Test module' into a number of separate activities. This fragment shows only the activities in the relatively simple activity of module testing. There are four streams of activities concerned with preparing test data, writing a test harness for the module, running the tests and reporting on the tests. The activities in the preparation streams would normally be interleaved. Obviously, the preparation activities precede the execution and reporting activities.

I have left out information on process pre- and post-conditions and process inputs and outputs in this diagram. This information would make the model complex and difficult to understand. Rather than trying to get all information into a single model,

Figure 25.4 Elements of a process model

Process model element	Description
Activity (represented by a round-edged rectangle with no drop shadow)	An activity has a clearly defined objective, entry and exit conditions. Examples of activities are preparing a set of test data to test a module, coding a function or a module, proofreading a document, etc. Generally, an activity is atomic, i.e. it is the responsibility of one person or group. It is not decomposed into sub-activities.
Process (represented by a round-edged rectangle with drop shadow)	A process is a set of activities which have some coherence and whose objective is generally agreed within an organisation. Examples of processes are requirements analysis, architectural design, test planning, etc.
Deliverable (represented by a rectangle with drop shadow)	A deliverable is a tangible output of an activity which is predicted in a project plan.
Condition (represented by a parallelogram)	A condition is either a pre-condition which must hold before a process or activity can start or a post-condition which holds after a process or activity has finished.
Role (represented by a circle with drop shadow)	A role is a bounded area of responsibility. Examples of roles might be configuration manager, test engineer, software designer, etc. One person may have several different roles and a single role may be associated with several different people.
Exception (not shown in examples here but may be represented as a double-edged box)	An exception is a description of how to modify the process if some anticipated or unanticipated event occurs. Exceptions are often undefined and it is left to the ingenuity of the project managers and engineers to handle the exception.
Communication (represented by an arrow)	An interchange of information between people or between people and supporting computer systems. Communications may be informal or formal. Formal communications might be the approval of a deliverable by a project manager; informal communications might be the interchange of electronic mail to resolve ambiguities in a document.

Figure 25.5 The module testing process



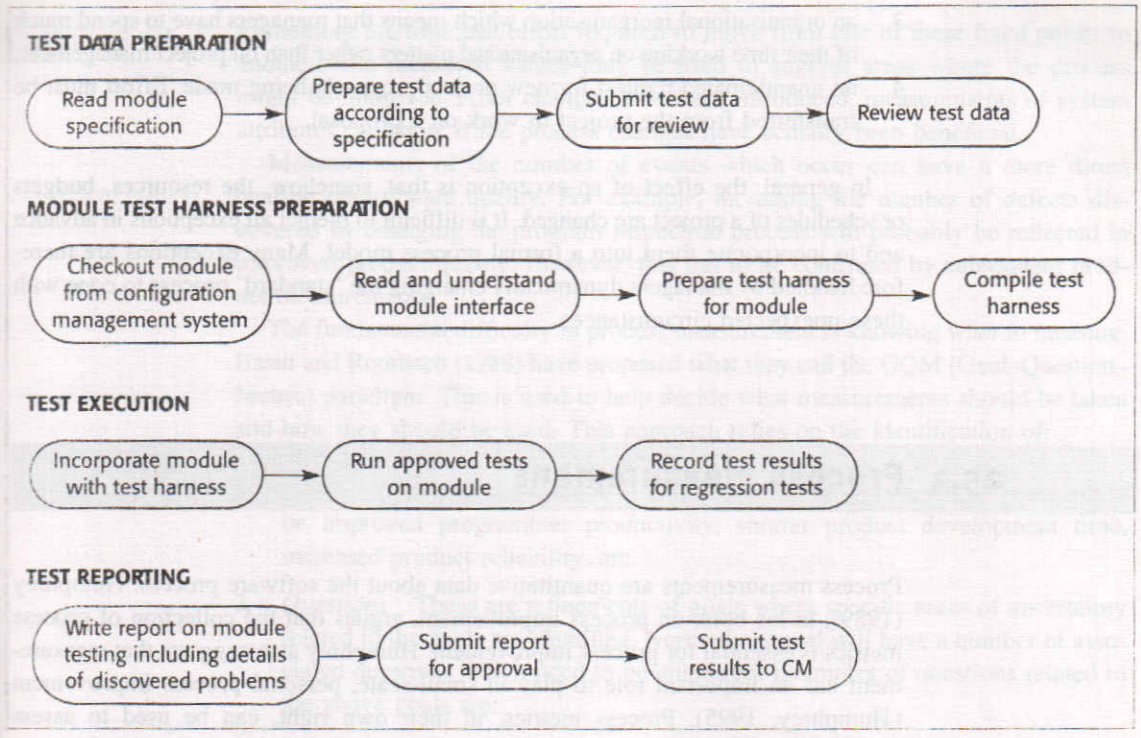


Figure 25.6 The activities involved in module testing

you may need to make several different models at different levels of abstraction. These should be related using common elements such as activities or deliverables. Some models should be primarily concerned with process activities, others with control information that drives the process execution.

25.2.1 Process exceptions

Software processes are very complex entities. While there may be a defined process model in an organisation, this can only ever represent the ideal 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 must deal with include:

1. several key people becoming ill at the same time just before a critical project review;
2. a communications processor failure which means that electronic mail is out of action for several days;

3. an organisational reorganisation which means that managers have to spend much of their time working on organisational matters rather than on project management;
4. an unanticipated request for new project proposals being made. Effort must be transferred from the project to work on a proposal.

In general, the effect of an exception is that, somehow, the resources, budgets or schedules of a project are changed. It is difficult to predict all exceptions in advance and to incorporate them into a formal process model. Many exceptions are therefore handled by managers dynamically changing the 'standard' process to cope with these unexpected circumstances.

25.3 Process measurement

Process measurements are quantitative data about the software process. Humphrey (1989), in his book on process improvement, argues that the collection of process metrics is essential for process improvement. Humphrey also suggests that measurement has an important role to play in small-scale, personal process improvement (Humphrey, 1995). Process metrics, in their own right, 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, testing time or both. However, process measurements on their own cannot be used to determine if product quality has improved. Product metrics (see Chapter 24) must also be collected and related to the process activities.

Three classes of process metric 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, etc.
2. *The resources required for a particular process* The resources might be total effort in person-days, travel costs, computer resources, etc.
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, the average number of lines of code modified in response to a requirements change, etc.

The first two types of measurement can be used to help discover if process changes have improved the efficiency of a process. Say there are fixed points in a software development process such as the acceptance of requirements, the completion of architectural design, the completion of test data generation, etc. It may be possible

to measure the time and effort required to move from one of these fixed points to another. The measured values may be used to suggest areas where the process might be improved. After changes have been introduced, measurements of system attributes can show if the process changes have actually been beneficial.

Measurements of the number of events which occur can 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.

The fundamental difficulty in process measurement is knowing what to measure. Basili and Rombach (1988) have proposed what they call the GQM (Goal–Question–Metric) paradigm. This is used to help decide what measurements should be taken and how they should be used. This approach relies on the identification of:

1. *Goals* What the organisation is trying to achieve. Examples of goals might be improved programmer productivity, shorter product development time, increased product reliability, etc.

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 which need to be answered. Examples of questions related to the above goals are:
   - how can the number of debugged lines of code be increased?
   - how can the time required to finalise product requirements be reduced?
   - how can more effective reliability assessments be made?

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. In the above examples, measurements which might be made include the productivity of individual programmers in lines of code and their level of experience, the number of formal communications between client and contractor for each requirements change and the number of tests required to cause product failure.

The advantage of this approach applied to process improvement is that it separates organisational concerns (the goals) from specific process concerns (the questions). It focuses data collection and suggests that collected data should be analysed in different ways depending on the question it is intended to answer. Basili and Green (1993) describe how this approach has been used in a long-term, measurement-based process improvement programme.

The GQM approach has been developed and combined with the SEI's capability maturity model (discussed below) in the ami method (Pulford *et al.*, 1996) of software process improvement. The developers of the ami method propose a staged approach to process improvement where measurement is introduced when an organisation has introduced some discipline into its processes. It provides guidelines and practical advice on implementing measurement-based process improvement.

25.4 The SEI Process Capability Maturity Model

The Software Engineering Institute (SEI) at Carnegie-Mellon University is a DoD-funded institute whose mission is software technology transfer. It was established to improve the capabilities of the US software industry and, specifically, the capabilities of those organisations who receive DoD funding for large defence projects. In the mid-1980s, the SEI initiated a study of ways of assessing the capabilities of contractors. They were particularly interested in contractors who were bidding for software projects funded by the US Department of Defense.

The outcome of this capability assessment work was the SEI Software Capability Maturity Model. This has been tremendously influential in convincing the software engineering community, in general, to take process improvement seriously. The SEI model classifies software processes into five different levels as shown in Figure 25.7.

These five levels are defined as follows:

1. *Initial level* At this level, an organisation does not have effective management procedures or project plans. If formal procedures for project control exist, there are no organisational mechanisms to ensure that they are used consistently. The organisation may successfully develop software but the characteristics of the software (quality, etc.) and the software process (budget, schedule, etc.) will be unpredictable.
2. *Repeatable level* At this level, an organisation has formal management, quality assurance and configuration control procedures in place. It is called the repeatable level because the organisation can successfully repeat projects of the same

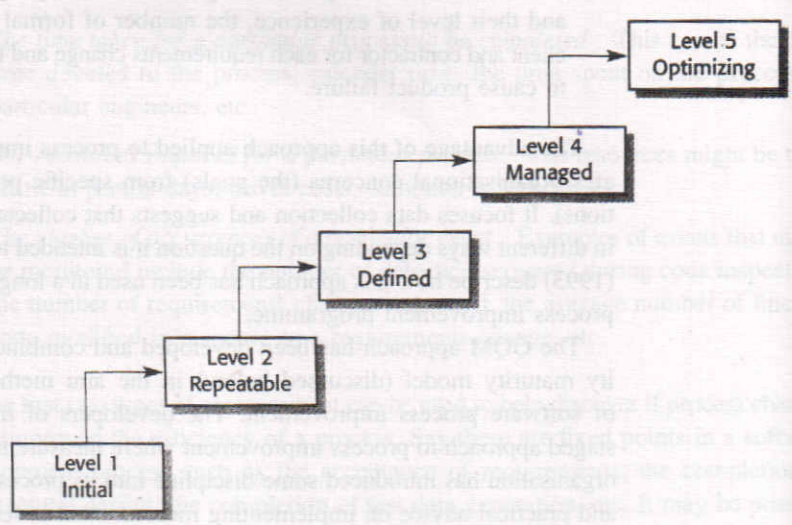


Figure 25.7 The SEI capability maturity model

type. However, there is a lack of a formal process model. Project success is dependent on individual managers motivating a team and on organisational folklore acting as an intuitive process description.

3. *Defined level* At this level, an organisation has defined its process and thus has a basis for qualitative process improvement. Formal procedures are in place to ensure that the defined process is followed in all software projects.
4. *Managed level* A Level 4 organisation has a defined process and a formal programme of quantitative data collection. Process and product metrics are collected and fed into the process improvement activity.
5. *Optimising level* At this level, an organisation is committed to continuous process improvement. Process improvement is budgeted and planned and is an integral part of the organisation's process.

The maturity levels in the initial version of the model were criticised as being too imprecise. After experience with using the model for capability evaluation as discussed in the following section, a revised version was adopted (Paulk *et al.*, 1993). The five levels were retained but were defined more specifically in terms of key process areas (Figure 25.8). Process improvement should be concerned with establishing these key processes and not with simply reaching some arbitrary level in the model. A similar approach, based on key practices, has been used to derive a requirements engineering process maturity model (Sommerville and Sawyer, 1997).

The SEI work on this model has been influenced by methods of statistical quality control in manufacturing. Humphrey (1988), in the first widely published description of the model 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.

While there are certainly some similarities, I do not think that results from manufacturing engineering can be transferred directly to software engineering. As I have already discussed in section 25.1, factors such as the skill and experience of the development engineers affect product quality. These are often as important as process factors.

The SEI maturity model is an important contribution but it should not be taken as a definitive capability model for all software processes. The model was developed to assess the capabilities of companies developing defence software. These are large, long-lifetime software systems which have complex interfaces with hardware and other software systems. They are developed by large teams of engineers and must follow the development standards and procedures laid down by the US Department of Defense.

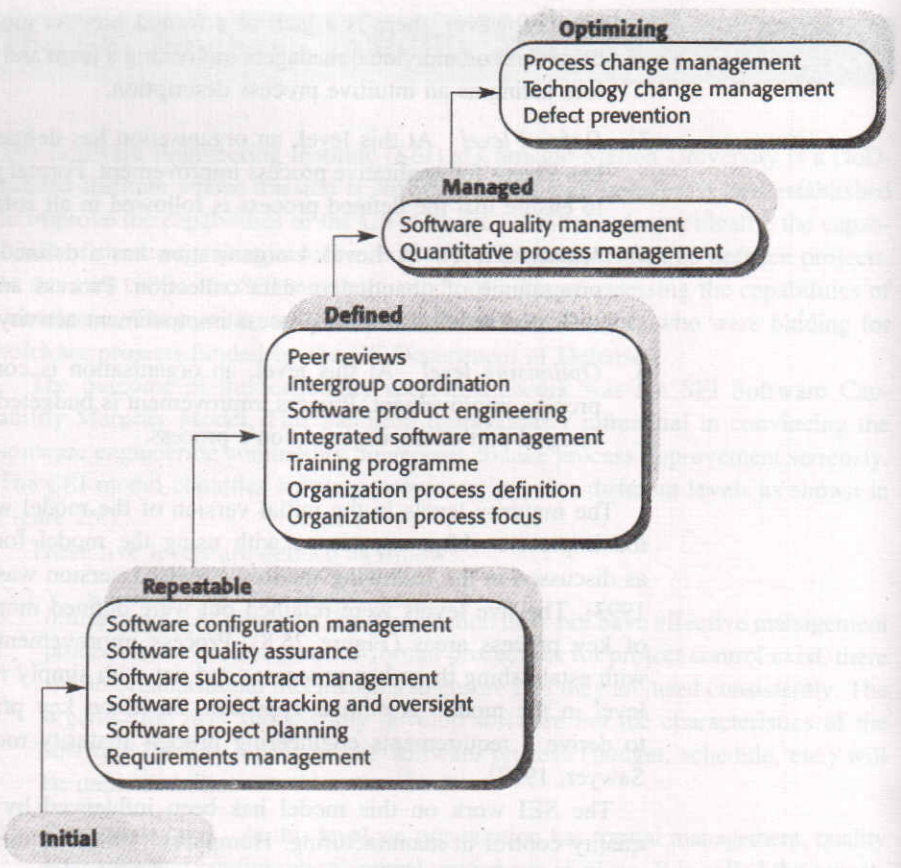


Figure 25.8 Key Process Areas (©1993 IEEE)

The first three levels of the SEI model are relatively simple to understand. The key process areas include practices which are currently used in industry. Some organisations have reached the higher levels of the model (Diaz and Sligo, 1997) but the standards and practices that are applicable at that level are not widely understood. In some cases, the best practice might diverge from the SEI model because of local organisational circumstances.

Problems at the higher levels do not negate the usefulness of the SEI model. Most organisations are at lower levels of process maturity. There are, however, three more serious problems with the SEI model. These may mean that it is not a good predictor of an organisation's capability to produce high-quality software.

The major problems with the capability maturity model are:

1. The model focuses exclusively on project management rather than product development. It does not take into account an organisation's use of technologies such as prototyping, formal or structured methods, tools for static analysis, etc.
2. It excludes risk analysis and resolution as a key process technology (Bollinger and McGowan, 1991). We have discussed the importance of risk assessment

in Chapter 1. Its advantage is that it discovers problems before they seriously affect the development process.

3. The domain of applicability of the model is not defined. The authors of the model clearly recognise that the model is not appropriate for all organisations. However, they do not describe the type of organisations where they think the model should and should not be used. The consequence of this is that the model has been oversold as a way to tackle software process problems. For smaller organisations, in particular, the model is too bureaucratic. Humphrey has recognised this and has now developed smaller-scale process improvement strategies (Humphrey, 1995).

The authors of the SEI model admitted (Humphrey and Curtis, 1991) that technology assessment was excluded because they could not find any standard way of assessing technology usage. Furthermore, they suggest that the management processes that are defined in the model are essential before technology can be effectively used. Therefore, at the lower levels of the model, the use of particular technologies is not significant as far as product quality is concerned.

This is, again, an over-simplification. The effective use of technologies such as prototyping and static analysis of programs can have a significant effect on product quality. This does not depend on how these are incorporated into the software process. The routine use of tools and methods by an organisation can clearly separate it from other organisations at the same level. It may mean that its product quality is superior to those products developed by organisations at a higher maturity level.

There are significant differences between commercial and defence software development. The problems of developing very large software systems are not necessarily shared by all organisations involved in software development. In particular, commercial software development can respond more quickly to technological change. This is evident in the way it has moved towards Internet-based systems and the use of distributed architectures. Defence software, on the other hand, has a long procurement process, a long lifetime and, for good reasons, tends to be more conservative in adopting new techniques. Some of the ideas underlying the SEI model are generally relevant but the model is not completely applicable outside the domain for which it was designed.

Paulk has compared the Capability Maturity Model with the ISO 9000 standard for quality management as discussed in Chapter 24 (Paulk, 1995). He looked at the key process areas at each level of the CMM and compared these to the ISO 9000 requirements for the quality management processes which should defined (see Figure 24.2). For the vast majority of areas, there is a clear correlation between the key processes and the ISO 9000 standard. The CMM is more detailed and prescriptive and includes a framework for process improvement. This is not covered in ISO 9000. In general, organisations whose process maturity is rated at Level 2 or 3 are likely to be ISO 9000 compliant. However, because of the abstract definition of ISO 9000, some Level 1 organisations can also satisfy the ISO 9000 standard.

Based on their experience with software process maturity, the SEI have now developed a number of other capability maturity models. A systems engineering capability maturity model assesses systems engineering processes as discussed in Chapter 2; the systems acquisition capability maturity model focuses on company's acquisition processes for software and hardware; the people capability maturity model (see Chapter 22) is concerned with organisational processes for improving the capabilities and competencies of its staff. Details of all of these are available from the SEI web site which is linked from this book's web pages.

25.4.1 Capability assessment

In discussions of process maturity and the SEI model, it is sometimes forgotten that the intention of the model was to allow the US Department of Defense to assess the capabilities of software contractors. At the time of writing, there is no published requirement for contractors to have reached a given level of maturity. However, there is an assumption in the community that organisations at the higher maturity level do have an advantage in bidding for contracts. In future, organisations will probably be expected to have reached a certain level of maturity (probably Level 3) before they can bid for DoD software contracts.

The model is intended to represent the capabilities of organisations rather than the maturity of particular projects. This makes sense from a contractual point of view but because an organisation is rated at Level 1 (say), this does not mean that all of its projects are at that level. Within the organisation, there may be particular projects or groups working at a much higher maturity level.

Capability assessment is based on a standard questionnaire that is designed to identify the key processes in the organisation. This is applied during an evaluation visit where project managers from a number of different projects are interviewed. After discussion of their responses to the questionnaire and refinement of these responses, an evaluation score is reached. A model of this assessment process is shown in Figure 25.9.

I think that the main problem with the current model is its stratification into levels and the judgemental association of numbers with these levels. The assessment guidelines require an organisation to have all the practices at a particular level in place before it can be accredited at that level. Thus, an organisation which has 80 per cent of Level 2 practices in place (say) and 70 per cent of Level 3 practices would receive a Level 1 rating. A better approach would be a finer-grain capability classification where the specific practices which are standard in an organisation are identified.

The SPICE approach to capability assessment and process improvement (Paulk and Konrad, 1994; El Amam *et al.*, 1997) is more flexible than the SEI model. SPICE has been proposed as an ISO standard process improvement framework. It includes maturity levels comparable with the SEI levels but also identifies key processes areas such as customer–supplier processes that cut across these levels. As the level of maturity increases, the performance of these key processes must improve.

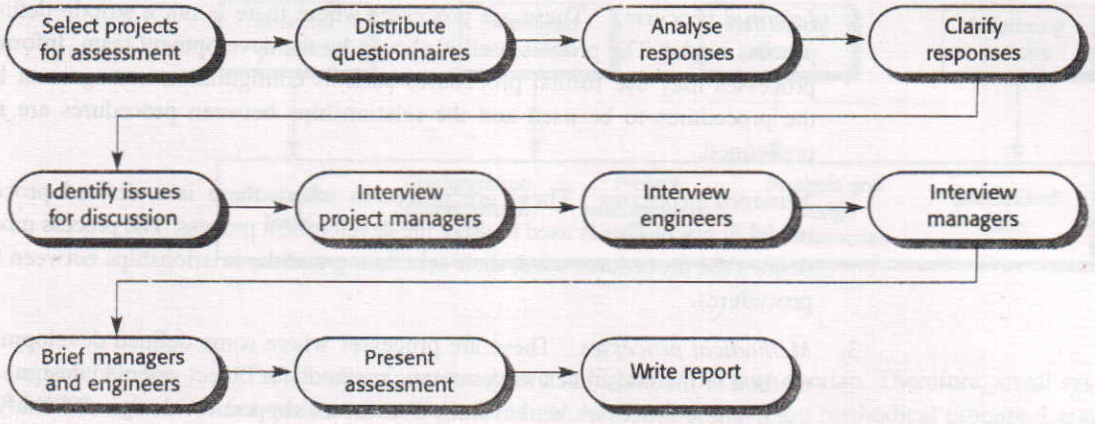


Figure 25.9
The capability assessment process

The Bootstrap project had the goal of extending and adapting the SEI maturity model to make it applicable across a wider range of companies. The Bootstrap model (Haase *et al.*, 1994; Kuvaja *et al.*, 1994) uses the SEI's maturity levels but also proposes:

- guidelines for a company-wide quality system to support process improvement,
- an important distinction between organisation, methodology and technology,
- a base process model (based on the model used in the European Space Agency) which may be adopted.

The evolution of the SEI's Capability Maturity Model is taking these alternative approaches into account. During the lifetime of this book, I anticipate that a new version of the CMM will be available which incorporates the best features of other approaches. This will become the world-wide standard for software process assessment and improvement.

25.5 Process classification

The process maturity classification proposed in the SEI model is appropriate for large, long-lifetime software projects undertaken by large organisations. There are many other types of software project and organisation where this view of process maturity should not be applied directly.

Rather than attempt to classify processes into levels, with fairly arbitrary boundaries drawn between these levels, I believe a more general approach to process classification can be applied across a broader spectrum of organisations and projects. Different types of process can be identified:

1. *Informal processes* These are processes where there is not a strictly defined process model. The process used is chosen by the development team. Informal processes may use formal procedures such as configuration management but the procedures to be used and the relationships between procedures are not predefined.

2. *Managed processes* These are processes where there is a defined process model in place. This is used to drive the development process. The process model defines the procedures used, their scheduling and the relationships between the procedures.

3. *Methodical processes* These are processes where some defined development method or methods (such as systematic methods for object-oriented design) are used. These processes benefit from CASE tool support for design and analysis processes.

4. *Improving processes* These are processes which have inherent improvement objectives. There is a specific budget for process improvements and procedures in place for introducing such improvements. As part of these improvements, quantitative process measurement may be introduced.

These classifications obviously overlap and a process may fall into several classes. For example, the process may be informal in that it is chosen by the development team. The team may choose to use a particular design method. They may also have a process-improvement capability. In this case, the process would be classified as *informal*, *methodical* and *improving*.

These classifications are useful because they serve as a basis for multi-dimensional process improvement. They help organisations choose an appropriate process for different types of product development. Figure 25.10 shows different types of product and the type of process that might be used for their development.

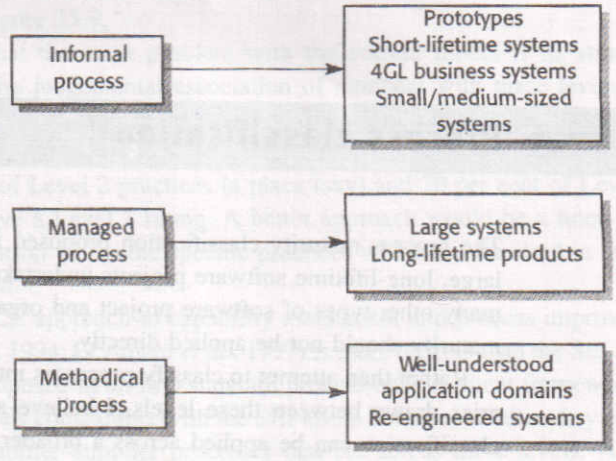


Figure 25.10 Process applicability

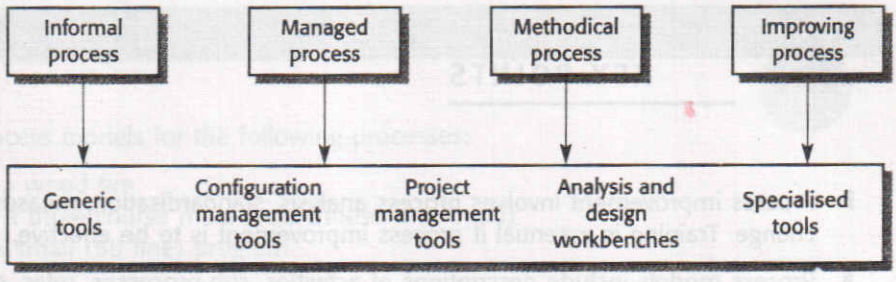


Figure 25.11 Process tool support

The classes of system shown in Figure 25.10 may overlap. Therefore, small systems which are re-engineered can be developed using a methodical process. Large systems always need a managed process. However, if the domain is not well understood, it may be difficult to choose an appropriate design method. Large systems may therefore be developed using a managed process that is not based on any particular design method.

Process classification provides a basis for choosing the right process to be used when a particular type of product is to be developed. For example, say a program is needed to support a transition from one type of computer system to another. This has a relatively short lifetime. Its development does not require the standards and management procedures which are appropriate for software which will be used for many years.

Process classification recognises that the process affects product quality. It does not assume, however, that the process is always the dominant factor. It provides a basis for improving different types of process. Different types of process improvement may be applied to the different types of process. For example, the improvements to methodical processes might be based on better method training, better integration of requirements and design, improved CASE tools, etc.

Most software processes now have some CASE tool support so they are *supported processes*. Methodical processes are now usually supported by analysis and design workbenches. However, processes may have other kinds of tool support (for example, prototyping tools, testing tools) irrespective of whether or not a structured design method is used.

The tool support that can be effective in supporting processes depends on the process classification. For example, informal processes can use generic tools such as prototyping languages, compilers, debuggers, word processors, etc. They will rarely use more specialised tools in a consistent way. Figure 25.11 shows that a spectrum of different tools can be used in software development. The effectiveness of particular tools depends on the type of process that is used.

Analysis and design workbenches are only likely to be cost-effective where a methodical process is being followed. Specialised tools are developed as part of the process improvement activity to provide specific support for improving certain process activities.

KEY POINTS

- Process improvement involves process analysis, standardisation, measurement and change. Training is essential if process improvement is to be effective.
- Process models include descriptions of activities, sub-processes, roles, exceptions, communications, deliverables and other processes.
- Measurement should be used to answer specific questions about the software process used. These questions should be based on organisational improvement goals.
- Three types of process metrics are time metrics, resource utilisation metrics and event metrics.
- The SEI process maturity model classifies software processes as initial, repeatable, defined, managed and optimising. It identifies key processes that should be used at each of these levels.
- The SEI model is appropriate for large systems developed by large teams of engineers. It should not be applied without adaptation to local circumstances.
- Processes can be classified as informal, managed, methodical and improving. This classification can be used to identify process tool support.

FURTHER READING

Trends in Software: Software Process Modelling and Technology. This book includes a good selection of overview papers which cover different aspects of software processes, including process modelling, process support and the use of the CMM. (A. Fuggetta and A. Wolf (eds.), 1996, John Wiley and Sons.)

The Capability Maturity Model for Software. This is a comprehensive description of the SEI's process assessment and improvement framework. It includes descriptions of all of the key processes and rationale for their inclusion. (M. Paulk, C, V, Weber and B. Curtis, 1995, Addison-Wesley.)

IEEE Software, **10**(4), July 1993. This is a special issue of the journal devoted to articles on process maturity. It includes a description of version 1.1 of the capability maturity model as well as articles discussing process improvement and process modelling.

EXERCISES

25.1 Suggest process models for the following processes:

- Lighting a wood fire
- Cooking a three-course meal (you choose the menu)
- Writing a small (50 line) program

25.2 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.

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

25.4 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.

25.5 Give two advantages and two disadvantages of the approach to process assessment and improvement which is embodied in the SEI process maturity model.

25.6 Suggest two application domains where the SEI capability model is unlikely to be appropriate. Give reasons why this is the case.

25.7 Consider the type of software process used in your organisation. How many of the key process areas identified in the SEI model are used? How would this model classify your level of process maturity?

25.8 Suggest three specialised software tools which might be developed to support a process improvement programme in an organisation.

25.9 Explain why a methodical process is not necessarily a managed process as defined in section 25.5.

25.10 Are process improvement programmes which involve measuring the work of people in the process and changing the process inherently dehumanising? What resistance to a process improvement programme might arise?