2 SOFTWARE LIFE CYCLES

2.1 Categories of Software Processes

Many distinct Software processes have been defined for use in the various part of the software development and software maintenance life cycles. These processes can be categorized as follows:

1. Primary Processes: Includes software processes for development, operation and maintenance of software.
2. Supporting Processes: Are applied intermittently or continuously throughout a software product life cycle to support primary processes; they include software processes used for configuration management, quality assurance, and verification and validation.
3. Organizational Processes: they provide support for software engineering; they include training, process measurement analysis, infrastructure management, portfolio and reuse management, organizational process improvement, and management of software life cycle models.
4. Cross-project processes: Such as reuse, software product line, and domain engineering; they involve more than a single software project in an organization.
5. Project Management Processes: they include processes for planning and estimating, resource management, measuring and controlling, leading, managing risk, managing stakeholders, and coordinating the primary, supporting, organizational, and cross-project processes of software development and maintenance projects.

Software processes are also developed for particular need, such as process activities that address software quality characteristics. For example, security concerns during software development may necessitate one or more software processes to protect the security of the development environment and reduce the risks of malicious acts. Software processes may also be developed to provide adequate grounds for establishing confidence in the integrity of the software.

2.2 Software Life Cycle Models

2.3 Software Process Adaptation

Predefined Software Development Life Cycle (SDLC) and individual software processes often need to be adapted (tailored) to better serve local needs. Organizational context, innovations in technology, project size, project criticality, regulatory requirements, industry practices and corporate culture may determine needed adaptations. Adaptation may also include omitting software processes or activities from a development or product life cycle model that are clearly inapplicable to the scope of work to be accomplished.

2.4 Practical Considerations

In practice, software processes and activities are often interleaved, overlapped, and applied concurrently. Software life cycle models that specify discrete software processes, with rigorously specified entry and exit criteria and prescribed boundaries and interfaces, should be reorganized as idealizations that must be adapted to reflect the realities of software development and maintenance within the organizational context and business environment.

Another practical consideration: software processes (such as configuration management, construction, and testing) can be adapted to facilitate operation, support maintenance, migration, and retirement of the software.

Additional factors to be considered when defining and tailoring a software life cycle model include required conformance to standards, directives, and policies; customer demands; criticality of the software product; and organizational maturity and competencies. Other factors include the nature of the work (e.g., modification of existing software versus new development) and the application domain (e.g., aerospace versus hotel management).

3. SOFTWARE PROCESS ASSESSMENT AND IMPROVEMENT

Once processes have been defined and deployed, it is to be expected that as the technical and organizational environment changes, the processes will have to change as well. Additionally, parts of the processes will turn out to be less than ideal for achieving the purposes of the process, new ideas will come up how the process could be improved further, and possibly it turns out that the process is not performed as expected.

All these issues show the importance of assessing the processes regularly, in order to check that they provide the expected results and to identify any improvement opportunities. Hence, there is need to address the questions on how to examine and improve the processes used within an organization to ensure that they are adequate to achieve their relevant goals, such as projects which are performed in time, in budget and delivering the needed quality of results, and to continually improve these processes.

Software process assessments are used to evaluate the form and content of a software process, which may be specified by a standardized set of criteria. In some instances, the term "process appraisal" and "capability evaluation" are used instead of process assessment. Capability evaluations are typically performed by an acquirer or by an external agent on behalf of an acquirer. The results are used as an indicator of whether the software processes used by a supplier are acceptable to the acquirer. Performance appraisals are typically performed within an organization to identify software processes in need of improvement or to determine whether a process satisfies the criteria at a given level of process capability or maturity.

Process assessments are performed at the levels of entire organizations, organizational units within the organizations, and individual projects. Assessment may involve issues such as assessing whether software process entry and exit criteria are being met, to review risk factors and risk management, or to identify lessons learned. Process assessment is carried out using both an assessment model and an assessment method. The model can provide a norm for benchmarking comparison among projects within an organization.

A process audit differs from process assessment. **Assessments** are performed to determine levels of capability or maturity and to identify software processes to be improved. **Audits** are typically conducted to ascertain compliance with policies and standards. Audits provide management visibility into actual operations being performed in the organization so that accurate and meaningful decisions can be made concerning issues that are impacting a development project, a maintenance activity or a software-related topic.

Success factors for software process assessment and improvement within software engineering organizations include management sponsorship, planning, training, experienced and capable leaders, team commitment, expectation management, the use of change agents, plus pilot projects and experimentation with tools. Additional factors include independence of the assessor and the timeline of the assessment.

software process assessment models typically include assessment criteria for software processes that are regarded as constituting good practices. The practices may address software development processes only, or they may also include topics such as software maintenance, software project management, system engineering, or human resource management.

3.1 SOFTWARE PROCESS ASSESSMENT METHODS

A **software process assessment method** can be qualitative or quantitative. Qualitative assessments rely on the judgement of experts: quantitative assessments assign numerical scores to software processes based on analysis of objective evidence that indicates attainment of the goals and outcomes of a defined software process. For example, a quantitative assessment of the software inspection process might be performed by examining the procedural steps followed and results obtained plus data concerning defects found and time required to find and fix the defects as compared to software testing.

A typical **method of software process assessment** includes planning, fact-finding (by collecting evidence through questionnaires, interviews, and observation of work practices) collection and validation of process data, and analysis and reporting. Process assessment may rely on the subjective, qualitative judgement of the assessor, or on the objective presence or absence of defined artifacts, records, and other evidence.

The activities performed during a software process assessment and the distribution of effort for assessment activities are different depending on the purpose of the software process assessment. Software process assessments may be undertaken to develop capability ratings used to make recommendations for process improvements or may be undertaken to obtain a process maturity rating in order to qualify for a contract or award.

The quality of assessment results depends on the software process assessment method, the integrity and quality of the obtained data, the assessment team's capability and objectivity, and the evidence examined during the assessment. The goal of a software process assessment is to gain insight that will establish the current status of a process or processes and provide a basis for process improvement; performing a software process assessment by following a checklist for conformance without gaining insights adds little value.

3.2 SOFTWARE PROCESS IMPROVEMENT MODELS

Software process improvement models emphasize iterative cycles of continuous improvement. A software process improvement cycle typically involves the subprocesses of measuring, analyzing, and changing. The Plan-Do-Check-Act model is well-known iterative approach to software process improvement. Improvement activities include identifying and prioritizing desired improvements (planning); introducing an improvement, including change management and training (doing); evaluating the improvement as compared to previous or exemplary process results and costs (checking); and making further modifications (acting).

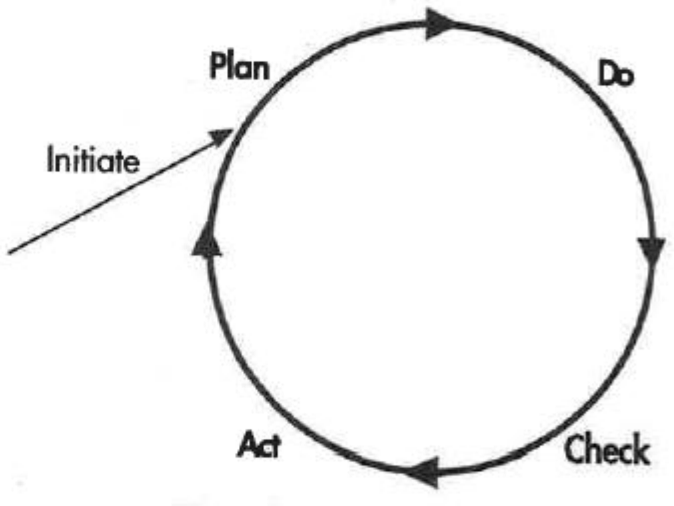

Figure 3.1: The Kaizen iterations of process improvement

The Plan-Do-Check-Act process improvement model can be applied, for example, to improve software processes that enhance defect prevention.

3.2.1 Capability Maturity Model (CMM)

The **capability maturity model,** developed by the Software Engineering Institute, is designed to help both development organizations and customers (government organizations or companies who acquire software). Software organizations need to understand the quality of their software process and how to improve it. Organizations contracting for software need ways to evaluate a potential contractor's capability to carry out the work.

The CMM has four intended uses to help organizations improve their software process capabilities:

1. Identify improvements

2. Identify risks in selecting contractors

3. Implement a process improvement program

4. Guide definition and development of the software process

The capability maturity model describes the characteristics of a mature software process. The model also shows how an immature software process can evolve into a well-managed mature one. The overall structure of the model is shown in Figure 3.2. Major components of the CMM, as shown in the figure, include:

• Maturity level: five levels on the path to a mature software process.

• Process capability: capability refers to expected results, that is, what can we predict from this organization's next project based on their current process capability?

• Key process areas: a cluster of related activities that, when performed collectively, achieve a set of goals considered important for enhancing process capability.

• Goals: the high-level objectives to be achieved by the key practices for that specific key process area.

• Key practices: the policies, procedures, and activities that most significantly contribute to the institutionalization and implementation of the key process area.

• Questions: yes/no questions that sample the key practices.

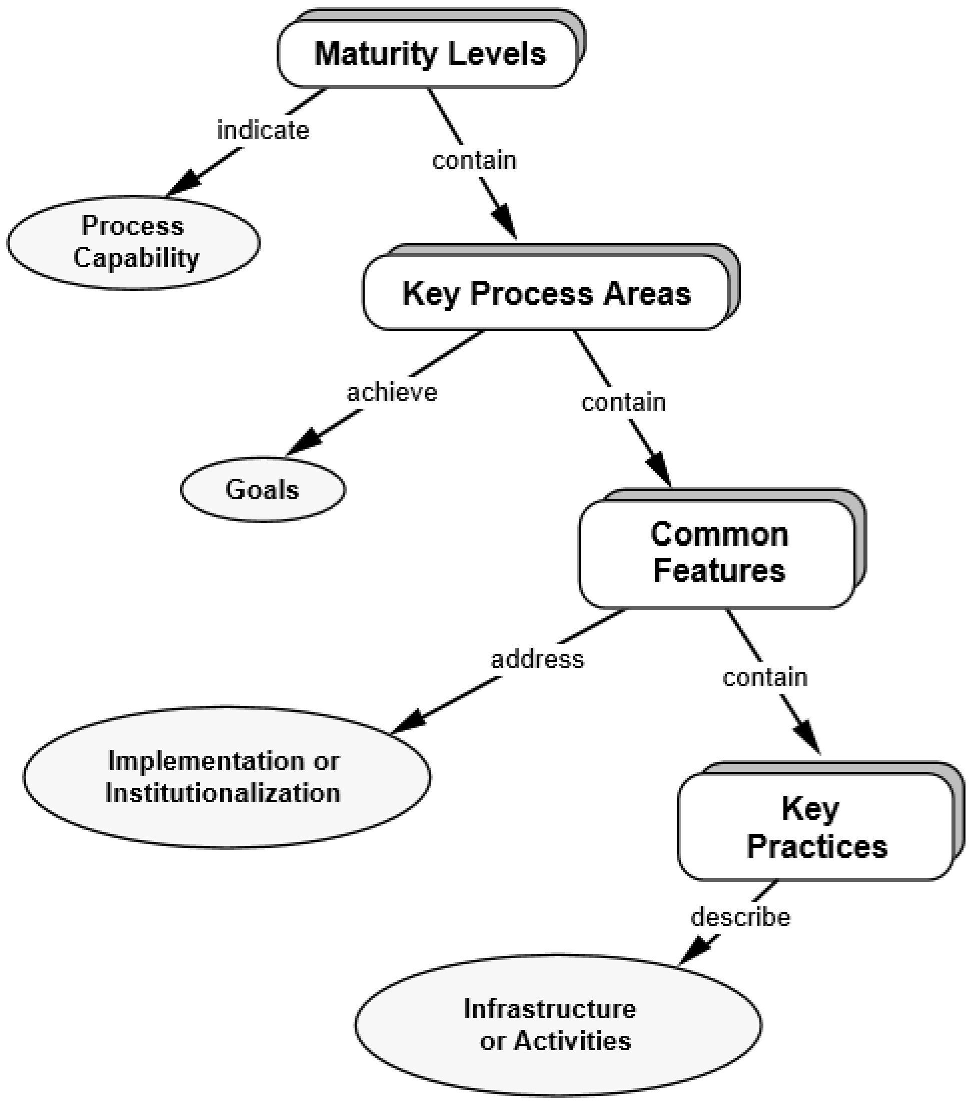

Figure 3.2: Capability maturity model (CMM) structure

Figure 3.2 shows an example, of the parts of the CMM structure to illustrate the relationships among the parts. In this figure, one sees the relationship between the components (maturity levels, key process areas, key practices, and questions). At the repeatable maturity level, in the key process area of software project planning, one of the key practices is to estimate project size. Thus, a typical question from the maturity questionnaire might be "Do you use a documented procedure to estimate software size?"

3.3 CONTINUOS AND STAGED SOFTWARE PROCESS RATINGS

Software process capability and software process maturity are typically rated using five or six levels to characterize the capability or maturity of the software processes used within an organization.

A continuous rating system involves assigning a rating to each software process of interest; a staged rating system is established by assigning the same maturity rating to all of the software processes within a specified process level. A representation of continuous and staged process levels is provided in Table 3.1. Continuous models typically use a level 0 rating; staged models typically do not.

Table 3.1: Software Process Rating Levels

Level	**Continuous Representation of Capability Levels**	**Staged Representation of Maturity Levels**
0	Incomplete	
1	Performed	Initial
2	Managed	Managed
3	Defined	Defined
4		Quantitatively Managed
5		Optimizing

In Table 3.1, the level 0 indicates that a software process is incompletely performed or may not be performed. At level 1, a software process is being performed (capability rating), or the software processes in a maturity level 1 group are being performed but on an ad hoc, informal basis. At level 2, a software process (capability rating) or the processes in maturity level 2 are being performed in a manner that provides management visibility into intermediate work products and can exert some control over transitions between processes. At level 3, a single software process or processes in a maturity level 3 group plus the process or processes in maturity level 2 are well defined and are being repeated across different projects. Level 3 of process capability or maturity provides the basis for process improvement across an organization because the processes are conducted in a similar manner. This allows collection of performance data in a uniform manner across multiple projects. At maturity level 4, quantitative measures can be applied and used for process assessment; statistical analysis may be used. At maturity level 5, the mechanisms for continuous process improvements.

Continuous and staged representations can be used to determine the order in which software processes are to be improved. In the continuous representation, the different capability levels for different software processes provide a guideline for determining the order in which software processes will be improved. In the staged representation, satisfying the goals of a set of software processes within a maturity level is accomplished for that maturity level, which provides a foundation for improving all of the software processes at the next higher level.