WEEK 5

SOFTWARE RELIABILITY AND QUALITY ASSURANCE

Achmad Benny Mutiara
amutiara@staff.gunadarma.ac.id

Content

5.1 VERIFICATION AND VALIDATION

- **Verification and validation (V and V)** is the name given to the **checking and analysis process** that ensures that software **conforms to** its specifications and **meets** the needs of the customers who are paying for that software.

- Verification and validation is a whole life-cycle process.
  - It starts with requirements reviews and continues through design reviews and code inspection to product testing.
  - There should be V and V activities at each stage of the software development process.
  - These activities ensure that the results of process activities are as specified.

- Verification and validation is not the same thing although they are easily confused. The difference between them is succinctly expressed by Boehm (1979):

  - **'Validation: Are we building the right product?'**
  - **'Verification: Are we building the product right?'**

- **Verification involves** checking that the software conforms to its specifications. You should check that the system *meets its specified functional and non-functional requirements*.

- **Validation, however, is a more general process**; you should ensure that the software *meets the expectations of the customer*. It goes beyond checking conformance of the system to **its specifications** to showing that the software does what the customer expects.

- Within the V and V process, *two techniques of system checking and analysis* may be used.

1. **Software inspections.**
   - ***Software inspection analyzes and checks system representations***, such as the requirements document, design diagrams, and the program source code. They may be applied at all stages of the process.
   - Inspections may be supplemented by some automatic analysis of the source text of a system or associated documents.
   - Software inspections and automated analyzes are static V and V techniques as they do not require the system to be executed.
2. **Software testing.**
   - ***Software testing involves executing an implementation*** of the software with test data and examining the outputs of the software and its operational behavior to check that it is performing as required.
   - Testing is a dynamic technique of verification and validation because it works with an executable representation of the system.

5.1.1 Verification

Verification *is the process of determining whether the output of one phase of software development confirms to that of its previous phase.*

OR

Verification *involves checking of functional and non-functional requirements to ensure that the software confirms to its specifications.*

5.1.2 Validation

Validation *is the process of determining whether a fully developed system confirms to its requirement specifications.*

OR

Validation *is an analysis process that is done after checking conformance of the system to its specifications.*

- Thus, the goal of the verification and validation process is to establish confidence in the customer that the software system is ***'fit for the customer.'*** It doesn't mean that the software system is free from errors.

5.2 SOFTWARE QUALITY ASSURANCE

- **The aim of** the Software Quality Assurance (SQA) process is to develop a high quality software product.
- ***Software Quality Assurance*** *is a set of activities designed to evaluate the process by which software is developed and/or maintained.*
- **Quality assurance** is *a planned and systematic pattern of all actions necessary to provide adequate confidence that the item or product conforms to established technical requirements (IEE83).*
- **The purpose of** a software quality assurance group is to provide assurance that the procedures, tools, and techniques used during product development and modification are adequate to provide the desired level of confidence in the work products.

The process of the SQA:

1. Defines the requirements for software controlled system fault/failure detection, isolation, and recovery;

2. Reviews the software-development processes and products for software error prevention and/or controlled change to reduced functionality states; and

3. Defines the process for measuring and analyzing defects as well as reliability and maintainability factors.

5.2.1 SQA Objectives

The various objectives of SQA are as follows:

- Quality management approach.
- Measurement and reporting mechanisms.
- Effective software-engineering technology.
- A procedure to assure compliance with software-development standards where applicable.
- A multi-testing strategy is drawn.
- Formal technical reviews that are applied throughout the software process.

5.2.2 SQA Goals

The major goals of SQA are as follows:

- SQA activities are planned.
- Non-compliance issues that cannot be resolved within the software project are addressed by senior management.
- Adherence of software products and activities to the applicable standards, procedures, and requirements is verified objectively.
- Affected groups and individuals are informed of SQA activities and results.

5.2.3 SQA Plan

- **An SQA plan** defines the quality processes and procedures that should be used. This involves selecting and instantiating standards for products and processes and defining the required quality attributes of the system.
- **The SQA plan** provides a roadmap for instituting software quality assurance. Developed by the SQA group (or the software team if a SQA group does not exist), the plan serves as a template for SQA activities that are instituted for each software project.
- **The quality plan** should select those organizational standards that are appropriate to a particular product and development process. New standards may have to be defined if the project uses new methods and tools.

An outline structure for a quality plan includes:

1. *Product introduction:* A description of the product, its intended markets, and the quality expectations for the product.
2. *Product plans:* The critical release dates and responsibilities for the product along with plans for distribution and product servicing.
3. *Process descriptions:* The development and service processes that should be used for product development and management.
4. *Quality goals:* The quality goals and plans for the product including an identification and justification of critical product quality attributes.
5. *Risks and risk management:* The key risks that might affect product quality and the actions to address these risks.

Preparation of a Software Quality Assurance Plan for each software project is a primary responsibility of the software quality assurance group. **Topics in a Software Quality Assurance Plan include**:

- Purpose-scope of plan;
- List of references to other documents;
- Management, including organization, tasks, and responsibilities;
- Documentation to be produced;
- Standards, practices, and conventions;
- Reviews and audits;
- Testing;
- Problem reporting and corrective action;
- Tools, techniques, and methodologies;
- Code, media, and supplier control;
- Records collection, maintenance, and retention;
- Training;
- Risk management—the methods of risk management that are to be used.

5.3 SOFTWARE QUALITY

- We examine the qualities that are pertinent to software products and software production processes. These qualities will become our goals in the practice of software engineering.
- The basic goal of software engineering is to produce quality software.
  - **Software quality** is a broad and important field of software engineering addressed by several standardization bodies, such as ISO, IEEE, ANSI, etc.

5.3.1 Definition of Software Quality

Software quality is the:

> *Conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software.*

The above definition emphasizes three important points:

1. **Software requirements** are the foundation from which quality is measured. Lack of conformance to requirements is lack of quality.
2. **Specified standards** define a set of development criteria that guide the manner in which software is engineered. If the criteria are not followed, lack of quality will almost surely result.
3. **There is a set of implicit requirements that often goes unmentioned**. If software conforms to its explicit requirements but fails to meet implicit requirements, software quality is suspect.

5.3.2 Classification of Software Qualities

- There are many desirable software qualities. Some of these apply both to the product and to the process used to produce the product.
- **The user wants** the software products to be reliable, efficient, and easy to use.
- **The producer of the software wants** it to be verifiable, maintainable, portable, and extensible.
- **The manager of the software project** wants the process of software development to be productive and easy to control.

In this section, we consider two different classifications of software-related qualities: internal versus external and product versus process.

External versus Internal Qualities

- We can divide software qualities into external and internal qualities.
- The external qualities are visible to the users of the system:
- The internal qualities are those that concern the developers of the system.
- In general, users of the software only care about the external qualities, but it is the internal qualities, which deal largely with the structure of the software, that help developers achieve the external qualities.
  - For example, the internal quality of verifiability is necessary for achieving the external quality of reliability. In many cases, however, the qualities are related closely and the distinction between internal and external is not sharp.

Product and Process Qualities

- We use a process to produce the software product.
- We can also attribute some qualities to the process, although process qualities often are closely related to product qualities.
  - For example, if the process requires careful planning of system test data before any design and development of the system starts, products reliability will increase.
- Some qualities, such as efficiency, apply both to the product and to the process.

- It is interesting to examine the word product here. It usually refers to what is delivered to the customer.
- Even though this is an acceptable definition from the customer's perspective, it is not adequate for the developer who requires a general definition of a software product that encompasses not only the object code and the user manual that are delivered to the customer but also the requirements, design, source code, test data, etc. In fact, it is possible to deliver different subsets of the same product to different customers.

5.3.3 Software Quality Attributes

- Software quality is comprised of six main attributes (called characteristics). At the top level, for software products, these attributes can be defined as follows:
  1. **Functionality*:*** The capability to provide functions which meet stated and implied needs when the software is used.
  2. **Reliability:** The capability to maintain a specified level of performance.
  3. **Usability:** The capability to be understood, learned, and used.
  4. **Efficiency:** The capability to provide appropriate performance relative to the amount of resources used.
  5. **Maintainability:** The capability to be modified for purposes of making corrections, improvements, or adaptation.
  6. **Portability:** The capability to be adapted for different specified environments without applying actions or means other than those provided for this purpose in the product.

5.3.4 McCall's Quality Factors

- McCall, Richards, and Walters [MCC77] propose a useful categorization of factors that affect software quality. These software quality factors, shown in Figure 4.2, focus on three important aspects of a software product: its operational characteristics, its ability to undergo change, and its adaptability to new environments.

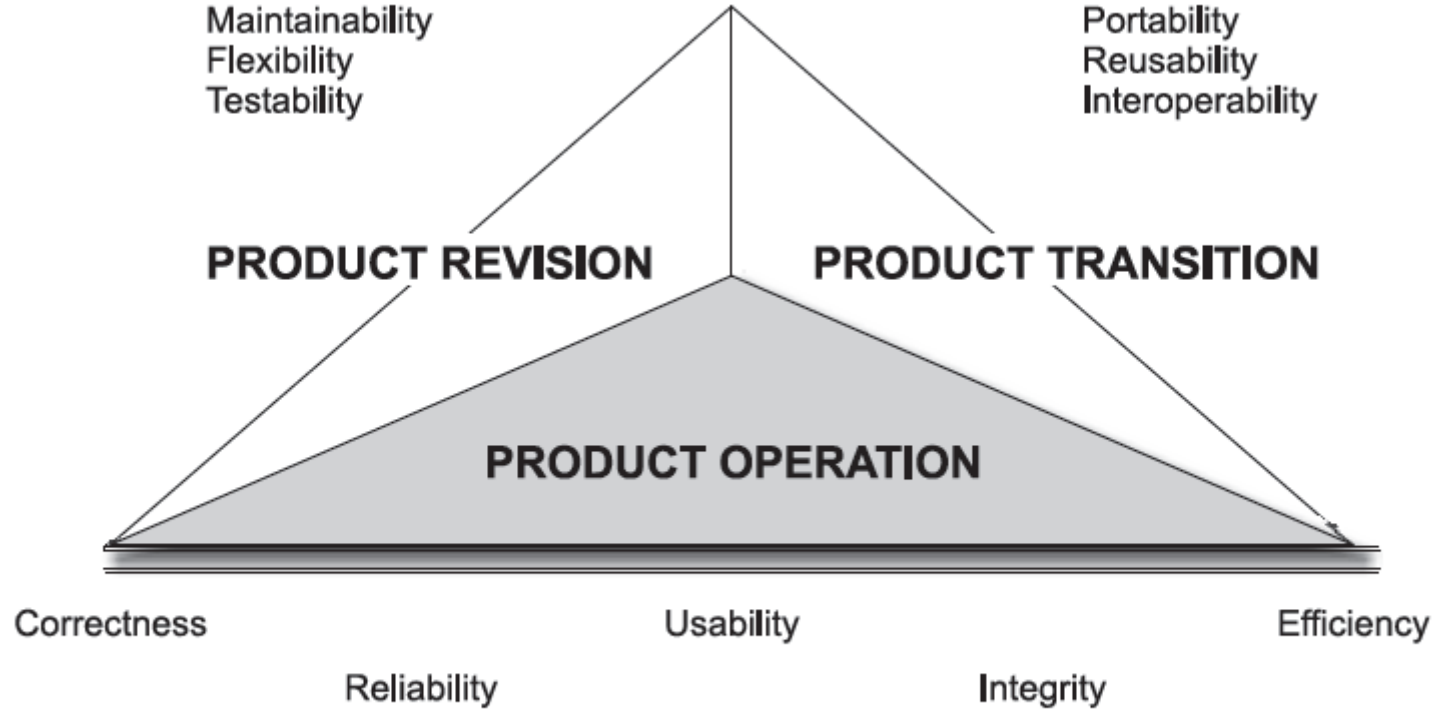

FIGURE 4.2 McCall's Software Quality Factors

- One attempt to identify specific product qualities that are appropriate to software has been that of James A. McCall. He grouped software qualities into three sets of quality factors:
  - Product operation qualities;
  - Product revision qualities; and
  - Product transition qualities.
- The definitions below are those given by McCall, but the reader may come across others. These are not all inclusive: sometimes other qualities might be of interest.

Product Operation Quality Factors

- *Correctness: The extent to which a program satisfies its specifications and* fulfills the user's objectives.
- *Reliability: The extent to which a program can be expected to perform its* intended function with required precision.
- *Efficiency: The amount of computer resources required by the software.*
- *Integrity: The extent to which access to software or data by unauthorized* persons can be controlled.
- *Usability: The effort required for learning, operating, preparing input, and* interpreting output.

Product Revision Quality Factors

- *Maintainability: The effort required to locate and fix an error in an operational* program.
- *Testability: The effort required to test a program to ensure it performs its* intended function.
- *Flexibility: The efforts required to modify an operational program.*

Product Transition Quality Factors

- *Portability: The effort required for transferring a program from one hardware* configuration and software system environment to another.
- *Reusability: The extent to which a program can be used in other applications.*
- *Interoperability: The efforts required to couple one system to another.*

5.3.5 Software Quality Criteria

- The software quality criteria of various quality factors are depicted in Table 4.1.

TABLE 4.1 Software Quality Criteria

Quality Factor	Software Quality Criteria
Correctness	Traceability, consistency, completeness
Reliability	Error tolerance, consistency, accuracy, simplicity
Efficiency	Execution efficiency, storage efficiency
Integrity	Access control, access audit
Usability	Operability, training, communicativeness, input/output volume, input/output area
Maintainability	Consistency, simplicity, conciseness, modularity, self-descriptiveness
Testability	Simplicity, modularity, instrumentation, self-descriptiveness
Flexibility	Modularity, generality, expandability, self-descriptiveness
Portability	Modularity, self-descriptiveness, machine independence, software system independence
Reusability	Generality, modularity, software system independence, machine independence, self-descriptiveness
Interoperability	Modularity, communications commonality, data commonality

5.3.6 Representative Qualities

- In this section, we present the most important qualities of software products and processes.

1. Correctness.

  – A program is functionally correct if it behaves according to the specification of the functions it should provide (called functional requirements specifications). It is common simply to use the term “correct” rather than “functionally correct”; similarly, in this context, the term “specifications” implies “functional requirements specification.” We will follow this convention when the context is clear.

  – The definition of correctness assumes that a specification of the system is available and that it is possible to determine unambiguously whether or not a program meets the specifications. With most current software systems, no such specification exists. If a specification does exist, it is usually written in an informal style using natural language.

2. *Reliability.*

– *Informally, software is reliable if the user can depend on it. The* specialized literature on software reliability defines reliability in terms of statistical behavior—the probability that the software will operate as expected over a specified time interval.

– Figure 4.3 illustrates the relationship between reliability and correctness. This figure shows that the set of all reliable programs includes the set of correct programs, but not vice versa.

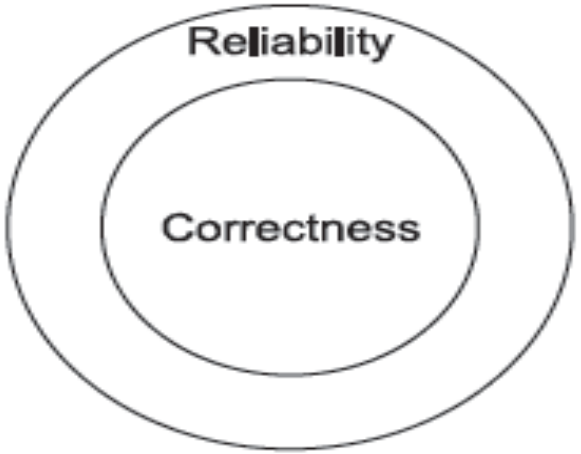

FIGURE 4.3 Relationship Between Correctness and Reliability in the Ideal Case

3. Robustness.

- *A program is robust if it behaves "reasonably," even in* circumstances that were not anticipated in the requirements specification— for example, when it encounters incorrect input data or some hardware malfunction (say, disk crash).
- Obviously, robustness is a difficult-to-define quality; after all, if we could state precisely what we should do to make an application robust, we would be able to specify its "reasonable" behavior completely. The robustness would become equivalent to correctness (or reliability in the sense of Figure 4.3).

4. Performance.

- *Performance is important because it affects the usability of the* system.
  - If a software system is too slow, it reduces the productivity of the users, possibly to the point of not meeting their needs.
  - If a software system uses too much disk space, it may be too expensive to run.
  - If a software system uses too much memory, it may affect the other applications that are run on the same system, or it may run slowly while the operating system tries to balance the memory usage of the different applications. Performance is also important because it affects the scalability of a software system.

5. *Verifiability.*

- A software system is verifiable if its properties can be verified easily. For example, it is important to be able to verify the correctness or the performance of a software system.
- Verifiability is usually an internal quality, although it sometimes becomes an external quality also. For example, in many security-critical applications, the customer requires the verifiability of certain properties. The highest level of the security standard for a trusted computer system requires the verifiability of the operating system kernel.

6. *Repairability.*

- A software system is repairable if it allows the correction of its defects with a limited amount of work. In many engineering products, repairability is a major design goal.
  - For example, automobile engines are built with the parts that are most likely to fail as the most accessible. In computer hardware engineering, there is a subspecialty called **Repairability, Availability, and Serviceability (RAS).**

7. *Evolvability.*

- ***Like other engineering products, software products are modified*** over time to provide new functions or to change existing functions. Indeed, the fact that software is so malleable makes modifications extremely easy to apply to an implementation.

8. *Understandability.*

- ***Some software systems are easier to understand than*** others. Of course, some tasks are inherently more complex than others. Given tasks of inherently similar difficulty, we can follow certain guidelines to produce more understandable designs and to write more understandable programs. For example, abstraction and modularity enhance a system's understandability.

9. *Interoperability.*

- ***"Interoperability" refers to the ability of a system to coexist*** and cooperate with other systems. With interoperability, a vendor can produce different products and allow the user to combine them if necessary. This makes it easier for the vendor to produce the products, and it gives the user more freedom in exactly what functions to pay for and to combine. Interoperability can be achieved through standardization of interfaces.

10. *Productivity.*

- ***Productivity is a quality of the software-production process; it*** measures the efficiency of the process and, as we said before, is the performance quality applied to the process. An efficient process results in faster delivery of the product.
- Productivity offers many trade-offs in the choice of a process. For example, a process that requires specialization of individual team members may lead to productivity in producing a certain product, but not in producing a variety of products.

- Software reuse is a technique that leads to the overall productivity of an organization that is involved in developing many products, but developing reusable modules is harder than developing modules for one's own use, thus reducing the productivity of the group that is developing reusable modules as part of their product development.

11. Timeliness.

- ***Timeliness is a process-related quality that refers to the ability to*** deliver a product on time. Timeliness requires careful scheduling, accurate estimation of work, and clearly specified and verifiable milestones.

12. Visibility.

- ***A software-development process is visible if all of its steps and its*** current status are documented clearly. Another term used to characterize this property is transparency. The idea is that the steps and the status of the project are available and easily accessible for external examination.

5.3.7 Importance of Software Quality

- We would expect quality to be a concern of all producers of goods and services. However, the special characteristics of software, and in particular, its intangibility and complexity, make special demands.

1. ***Increasing Criticality of Software.*** The final customer or user is naturally anxious about the general quality of software, especially its reliability. This is increasingly the case as organizations become more dependent on their computer systems and software is used more and more in areas which are safety critical; for example, to control aircraft.

2. ***The Intangibility of Software.*** This makes it difficult to know whether a particular task in a project has been completed satisfactorily. The results of these tasks can be made tangible by demanding that the developers produce‘deliverables’ that can be examined for quality.

3. ***Accumulating Errors During Software Development.*** *As computer system* development is made up of a number of steps where the output from one step is the input to the next, the errors in the earlier deliverables will be added to those in the later steps leading to an accumulating detrimental effect, and generally, the later in a project that an error is found the more expensive it will be to fix. In addition, because the number of errors in the system is unknown the debugging phases of a project are particularly difficult to control.

For these reasons quality management is an essential part of effective overall project management.

5.5 CAPABILITY MATURITY MODEL (SEI-CMM)

- SEI-CMM stands for **Software Engineering Institute Capability Maturity Model.**
- The CMM was developed by the Software Engineering Institute (SEI) of the Carnegie Mellon University in the USA, which is engaged in a long-term program of software-process improvement.
  - It is a model that helps in judging **the maturity** of a software process of an organization.
  - It also helps **to identify the main process** that will help in increasing the maturity of these processes.
  - It has become **a standard for assessing and improving software processes**.

- SEI-CMM can be used in two ways: **capability evaluation** and **software-process assessment.**
  - Capability evaluation and software-process assessment differ in motivation, objective, and the final use of the result.
  - **Capability evaluation** provides **a way to assess** the software-process capability of an organization.
  - The results of capability evaluation indicate the likely contractor performance if the contractor is awarded a work.
  - Therefore, the results of the software-process capability assessment can be used to select a contractor.
  - On the other hand, **software-process assessment** is used by an organization with the objective to improve its process capability. Thus, this type of assessment is for purely internal use.

Five levels of process maturing have been proposed for the software industry by SEI-CMM, which indicate the sophistication and quality of their software production practices. These levels are defined as follows:

1. **Level 1 (Initial).**
   - **The software process is adhoc, and even chaotic at time.**
   - The organization whose success depends on individual effort and whose processes are not defined and documented come under this level.
   - Organizations at this level can benefit most by improving project management, quality assurance, and change control.
2. **Level 2 (Repeatable).**
   - **Here basic management practices, such as tracking** costs, schedules, and functionality are established **but not** the procedures for doing it.
   - Here, the efforts done previously for the success of a project may repeat.
   - **Some of the characteristics of a process at this level are**: project commitments are realistic and based on past experience with similar projects, costs and schedules are tracked and problems resolved when they arise, formal configuration control mechanisms are in place, and software project standards are defined and followed.

3. Level 3 (Defined).

- **The organization-wide software process includes** management and engineering procedures. These procedures are well defined, documented, standardized, and integrated. All projects make use of the documented and approved version of the organization process for software development and maintenance. But the process and practices are not analyzed quantitatively. In this process both the development and management processes are formal. **ISO 9000 aims at achieving this level**.

4. Level 4 (Managed).

- **At this level, the focus is on software metrics.**
- **Two** types of metrics are collected.
- **Product metrics** measure the **characteristics of the product being developed**, such as its size, reliability, time complexity, understandability, etc.
- **Process metrics** reflect **the effectiveness** of the process being used, such as the average defect correction time, productivity, the average number of defects found per hour of inspection, the average number of failures detected during testing per LOC, and so forth.

– The process metrics are used to check if a project performed satisfactorily. Thus, the results of process measurements are used to evaluate project performance rather than to improve the process. Software processes and products are quantitatively understood, measured, and controlled using detailed procedures.

5. Level 5 (Optimized).

– **At this level, an organization is committed to** continuous process improvement.

– **Process improvement is budgeted and planned and is an integral part of the organization's process.** The organizations have the means to identify weaknesses and strengthen the process proactively, with the goal of preventing the occurrence of defects.

– Best software-engineering and management practices are used throughout the organization.

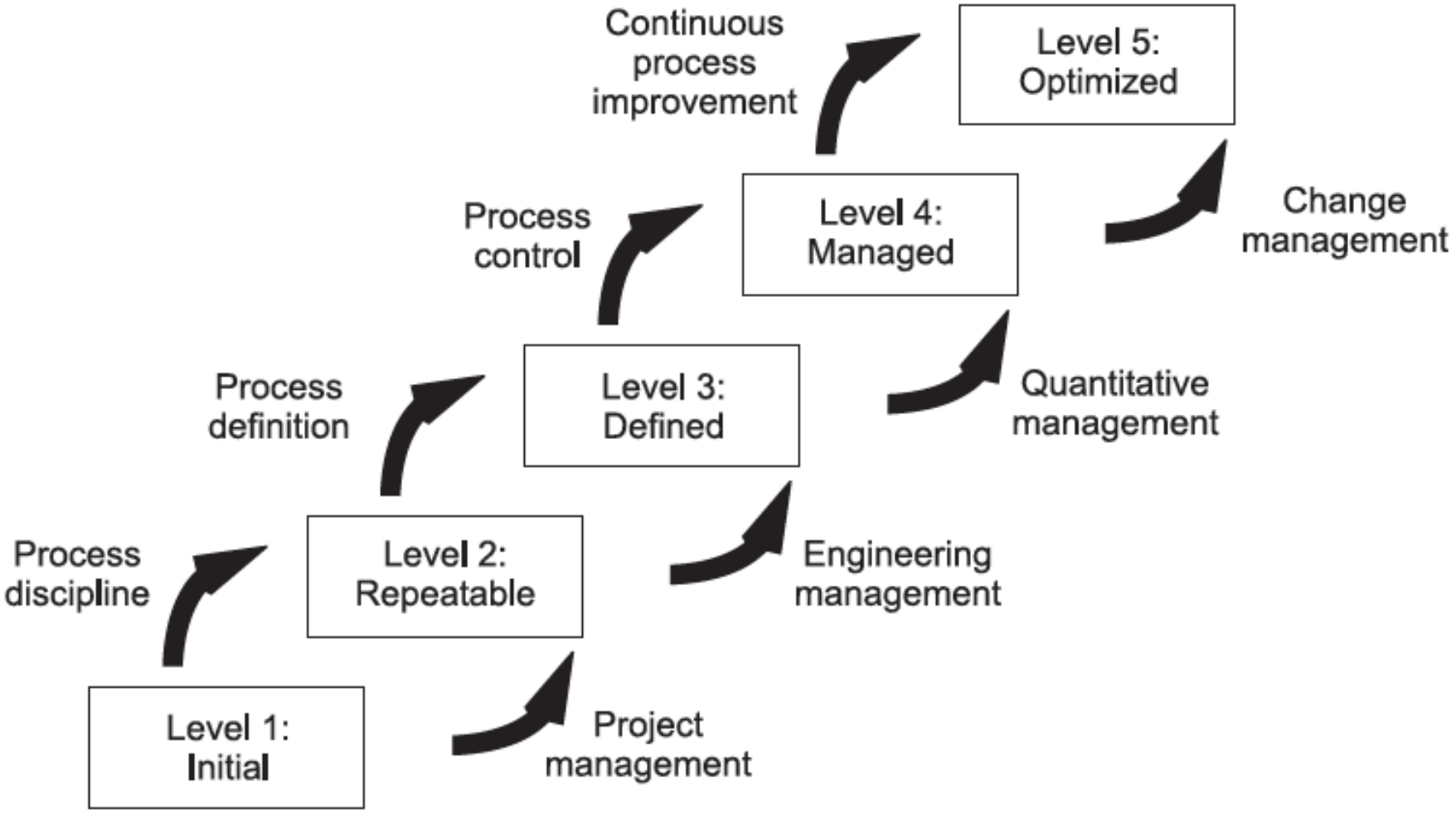

FIGURE 4.4 Software Engineering Institute Levels of Maturity

Except for Level 1, each maturity level is characterized by **several Key Process Areas (KPAs) that indicate the areas an organization** should focus on to improve its software process at the next level. This is shown in Table 4.2:

TABLE 4.2

MM level	Characteristics	Focus	Key Process Areas
1 (Initial)	Chaotic, unrepeatable, high risk of non-performance	Competent people	Not applicable
2 (Repeatable)	Methodical in umbrella activities and processes. Performance is repeated but not improved	Project management	Configuration management, quality assurance, sub-contract management, project tracking and oversight, project planning
3 (Defined)	Improved performance in cost, schedule quality, and risk management	Definition of processes	Peer reviews, inter-group coordination, software product engineering, training program, integrated software management, and organization process definition
4 (Managed)	Learns from project experience, and continuous performance improvement in all key areas of the project	Product and process quality	Quality management, process measurement, and analysis
5 (Optimized)	All-around performance, improvement through learning. High performance in all quality attributes and risk management through RMMM	Continuous process improvement	Process change management, technology innovation, and defect prevention

5.5 INTERNATIONAL STANDARD ORGANIZATION (ISO)
5.5.1 Introduction to the ISO

- The International Organization for Standardization (ISO) is a group of worldwide federations of national standards bodies from some 100 countries.
- The ISO is a non-governmental organization established in 1947.
- The ISO-9000 standard specifies quality-assurance elements in generic terms, which can be applied to any business, regardless of the product or services being offered.
  - In order to register for one of the quality-assurance system models contained in the ISO-9000, **third-party auditors** examine an organization's quality system and operations for compliance to the standard and for effective operations.
  - Upon successful audit, the organization receives a certificate from a registered body represented by the auditors. Thereafter, semi-annual audits ensure conformance to the standard.

- The ISO-9000 standard views an organization as a set of interrelated processes.
  - In order to pass the criteria for ISO-9000 compliance, the processes must address the identified areas, and document and practice them. When a process is documented, it is better understood, controlled, and improved. **However**, the ISO-9000 standard does not specify how an organization should implement its quality system. Therefore, the biggest challenge is to design and implement a quality-assurance system that meets the standard and gels well with the products/services of the organization.
- The **ISO-9001** is **a quality-assurance standard that is specific to software engineering.**
  - It specifies **20 standards** with which an organization must comply for an effective implementation of the quality assurance system.
- The ISO-9000 series of standards is a set of documents dealing with quality systems that can be used for quality assurance purposes.
  - The ISO-9000 series is not just a software standard. It is a series of five related standards that are applicable to a wide variety of industrial activities, including design/development, production, installation, and servicing.

5.5.2 ISO-9000 Mission

- The mission of the ISO is to promote the development of standardization and related activities to facilitate the international exchange of goods and services, and to develop cooperation in the spheres of intellectual, scientific, technological, and economic activity. The ISO published its ISO-9000 standard in 1988. ISO-9000 consists of **three standards for external quality assurance**. These are ISO-9001, ISO- 9002, and ISO-9003.

1. ISO-9001.

– The ISO-9001 is an international quality-management system. The ISO-9001 is mainly related to the software industry. It lays down the standards for designing, developing, servicing, and producing a standard quality of goods. It is **also applicable to** most software-development organizations.

2. ISO-9002.

- The ISO-9002 is basically related to manufacturing only and is silent on designing issues. Examples of this category of industries include steel- and car-manufacturing industries that buy the product and plant designs from external sources and are involved in only manufacturing those products. Therefore, the **ISO-9002 is not applicable to software-development organizations.**

3. ISO-9003.

- The ISO-9003 standard **applies to the service industry**. The organizations who are involved in only installation of products, services, and testing of products are eligible for the ISO-9003 certification.

4.5.3 Why is ISO certification required by the software industry?

There are several reasons why the software industry must get an ISO certification. Some of the important reasons include:

- It is sign of customer confidence. This certification has become a standard for international bidding.
- It is a motivating factor for business organizations.
- It makes processes more focused, efficient, and cost-effective.
- It helps in designing high-quality repeatable software products.
- It highlights weaknesses and suggests corrective measures for improvements.
- It facilitates the development of optimal processes and total quality measurement.
- It emphasizes the need for proper documentation.

5.5.4 How does an organization obtain ISO-9000 certification?

An organization intending to obtain ISO-9000 certification applies to a ISO-9000 registrar for registration. The ISO-9000 registration process consists of the following stages:

1. ***Application.*** *Once an organization decides to go for ISO-9000 certification, it* applies to a registrar for registration.
2. ***Pre-assessment.*** *During this stage, the registrar makes a rough assessment of* the organization.
3. ***Document Review and Adequacy of Audit.*** *During this stage, the registrar* reviews the documents submitted by the organization and makes suggestions for possible improvements.
4. ***Compliance Audit.*** *During this stage, the registrar checks whether the* suggestions made by it during review have been complied by the organization or not.
5. ***Registration.*** *The registrar awards the ISO-9000 certificate after successful* completion of all previous phases.
6. ***Continued Surveillance.*** *The registrar continues to monitor the organization,*though only periodically.

5.5.5 Benefits of ISO-9000 Certification

Benefits of ISO-9000 certification include:

1. ***Continuous Improvement.*** Business ISO-9000 certification forces an organization to focus on "how they do business." Each procedure and work instruction must be documented and thus becomes the springboard for continuous improvement.

2. ***Eliminate Variation.*** Documented processes are the basis for repetition and help eliminate variation within the process. As variation is eliminated, efficiency improves. As efficiency improves, the cost of quality is reduced.

3. ***Higher Real and Perceived Quality***
   - *With the development of solid corrective and preventative measures,* permanent, company-wide solutions to quality problems are found.
   - *This results in higher quality.*

4. ***Boost Employee Morale.*** Employee morale is increased as they are asked to take control of their processes and document their work processes.

5. ***Improved Customer Satisfaction.*** Customer satisfaction, and more importantly customer loyalty, grows as a company transforms from a reactive organization to a proactive, preventative organization. It becomes a company people want to do business with.

6. ***Increased Employee Participation.*** Reduced problems resulting from increased employee participation, involvement, awareness, and systematic employee training.

7. ***Better Product and Services.*** Better products and services result from continuous improvement processes.

8. ***Greater Quality Assurance.*** Fosters the understanding that quality, in and of itself, is not limited to a quality department but is everyone's responsibility.

9. ***Improved Profit Levels.*** Improved profit levels result as productivity improves and rework costs are reduced.

10. ***Improved Communication.*** Improved communications both internally and externally which improves quality, efficiency, on-time delivery, and customer/ supplier relations.

11. ***Reduced Cost.*** *ISO standards result in reduced costs of the product.*

12. ***Competitive Edge.*** By offering higher customer services, ISO-9000 standards add a competitive edge.

5.5.6 Limitations of ISO-9000 Certification

The limitations of ISO-9000 certification include:

- ISO-9000 **does not provide** any guidelines for defining an appropriate process.
- ISO-9000 certification process **is not foolproof and no international accrediting agency exists.**
- ISO-9000 **does not automatically lead to continuous process improvement**, i.e., it **does not automatically lead to TQM**.

4.5.7 Uses of ISO

- ISO certification is used as **a reference model for a contract between the external independent parties**. It must be remembered that ISO certification **does not guarantee** a high-quality product. It focuses mainly on the processes. ISO is now being used as the standard and basis of evaluation for international bidding. The ISO-9000 specifies a set of guidelines for repeatable organizations. A repeatable organization is one where the production process is person-independent.

5.5.8 Salient Features of ISO-9001 Requirements

The salient features of ISO-9001 requirements include:

- All documents concerned with the development of a software product should be properly managed, authorized, and controlled.
- Proper plans should be prepared and then progress against these plans should be monitored.
- Important documents should be independently checked and reviewed for effectiveness and correctness.
- The product should be tested against its specifications.

4.5.9 ISO-9126

- Over the years, various lists of software quality characteristics have been put forward, such as those of McCall, described previously and of Boehm.
  - A difficulty has been the lack of agreed definitions of the qualities of good software.
  - The term ‘maintainability’ has been used, for example, to refer to the ease with which an error can be located and corrected in a piece of software, and also in a wider sense to include the ease of making any changes.
  - For some, ‘robustness’ has meant the software’s tolerance of incorrect input, while for others it has meant the ability to change program code without introducing unexpected errors.
  - The ISO-9126 standard was published in 1991 to tackle the question of the definition of software quality. This 13-page document was designed as a foundation upon which further, more detailed, standards could be built.

ISO-9126 identifies six software quality characteristics:

- **Functionality**, which covers the functions that a software product provides to satisfy user needs;
- **Reliability**, which relates to the capability of the software to maintain its level of performance;
- **Usability**, which relates to the effort needed to use the software;
- **Efficiency**, which relates to the physical resources used when the software is executed;
- **Maintainability**, which relates to the effort needed to make changes to the software;
- **Portability**, which relates to the ability of the software to be transferred to a different environment.

- The ISO-9126 has sub-characteristics for each of the primary characteristics. It is indicative of the difficulties of gaining widespread agreement that these subcharacteristics are outside the main standards and are provided for 'information only.' They are useful as they clarify what is meant by the main characteristics.

Characteristic Functionality

- **Sub-characteristics: *Suitability, Accuracy, Interoperability, Compliance, Security***
  - **Compliance** refers to the degree to which the software adheres to application-related standards or legal requirements. Typically these are auditing requirements.
  - Interoperability and security are good illustrations of the efforts of the ISO-9126 to clarify terminology.
  - '**Interoperability**' refers to the ability of the software to interact with other systems. The framers of the ISO-9126 have chosen this word rather than compatibility because the latter causes confusion with the characteristic referred to by the ISO-9126 as 'replace ability' (see below).

Characteristic Reliability

- **Sub-characteristics:** ***Maturity, Fault tolerance, Recoverability***
  - **Maturity** refers to the **frequency of failure due to faults** in a software product, the implication being that the more the software has been used, the more faults will have been uncovered and removed.
  - It is also interesting to note that **recoverability** has been clearly distinguished from security which describes the control of accessto a system.

Characteristic Usability

- **Sub-characteristics: *Understandability, Learnability, Operability***
  - **Understandability** is a pretty clear quality to grasp, although the definition "attributes that bear on the users' efforts for recognizing the logical concept and its applicability" in our view actually makes it less clear!
  - Note how **learnability** is **distinguished from operability**.
    - A software tool could be easy to learn but time-consuming to use because, perhaps, it uses a large number of nested menus.
    - This might be fine for a package used intermittently, but not where the system is used for many hours each day.
    - **In this case**, learnability has been incorporated at the expense of operability.

Characteristics Efficiency

- **Sub-characteristics:** Time behavior, Resource behavior

Characteristics Maintainability

- **Sub-characteristics: *Analyzability, Changeability, Stability, Testability***
  - **Analyzability** is the quality that McCall called diagnosability, the ease with which the cause of a failure can be determined.
  - **Changeability** is the quality that others have called flexibility: the latter name is perhaps a better one as changeability has a slightly different connotation in plain English—it implies that the suppliers of the software are always changing it!
  - **Stability**, on the other hand, **does not mean** that the software **never changes**: *It means that there is a low risk of a modification to the software having unexpected effects*.

Characteristic Portability

- **Sub-characteristics: *Adaptability, Installability, Conformance, Replaceability***
  - **Conformance**, as distinguished from compliance, relates ***to those standards that have a bearing on portability***. The use of a standard programming language common to many software/hardware environments would be an example of conformance.
  - **Replaceability** refers to **the factors that give 'upwards compatibility'** between old software components and the new ones. **Downwards compatibility is specifically excluded from the definition.**

- **The ISO-9126 provides guidelines for the use of the quality characteristics**. Variation in the importance of different quality characteristics depending on the type of product is stressed.
- **Thus, reliability will be of particular** concern with safety critical systems while efficiency will be important for some real-time systems.
- For interactive end-user systems, the key quality might be usability.
- Once the requirements for the software product have been established the following steps are suggested:
  - **Quality metrics selection**
  - **Ratings level definition**
  - **Assessment criteria definition**

5.6 COMPARISON OF ISO-9000 CERTIFICATION AND THE SEI-CMM

Comparison of some of the key characteristics of ISO-9000 certification and the SEI-CMM model are as follows:

- The emphasis of the SEI-CMM is on continuous process improvement, whereas the ISO-9000 addresses the minimum criteria for an acceptable quality system.
- The Capability Maturity Model (CMM) is a five-level framework for measuring software-engineering practices, as they relate to a process. On the other hand the ISO-9000 defines a minimum level of generic attributes for a quality management program.
- The ISO-9000 is awarded by an international standard body. Therefore, ISO- 9000 certification can be quoted by an organization in official documents, in communication with external parties, and in tender quotations. However, SEICMM assessment is purely for internal use.

- The ISO-9000 standards were basically designed to audit manufacturing/ service organizations, whereas the CMM was developed specifically for the software industry and thus addresses several issues specific to the software industry.
- The SEI-CMM focuses strictly on software, while the ISO-9000 has a much wider scope: hardware, software, processed materials, and services.
- The SEI-CMM model provides a list of key process areas (KPAs) on which an organization at any maturity level needs to concentrate to take it from one maturity level to the next. Thus, it provides a way for achieving gradual quality improvement.
- The SEI-CMM model aims for achieving Total Quality Management (TQM), which is beyond quality assurance, whereas the ISO-9000 aims at Level 3 (defined level) of the SEI-CMM model.
- The ISO-9000 requires that procedures for handling, storage, packaging, and delivery be established and maintained, while replication, delivery, and installation are not covered in the SEI-CMM.

- The ISO-9000 requires that procedures for handling, storage, packaging, and delivery be established and maintained, while replication, delivery, and installation are not covered in the SEI-CMM.

5.7 RELIABILITY ISSUES
5.7.1 Software Reliability

Software reliability *is defined as the probability of failure-free operation of a computer program in a specified environment for a specified time.*

OR

Reliability of a software product *essentially denotes its trustworthiness or dependability.*

OR

Reliability of a software product can also be defined as the probability of the product working "correctly" over a given period of time.

- The expected curve for software is given in Figure 4.5.

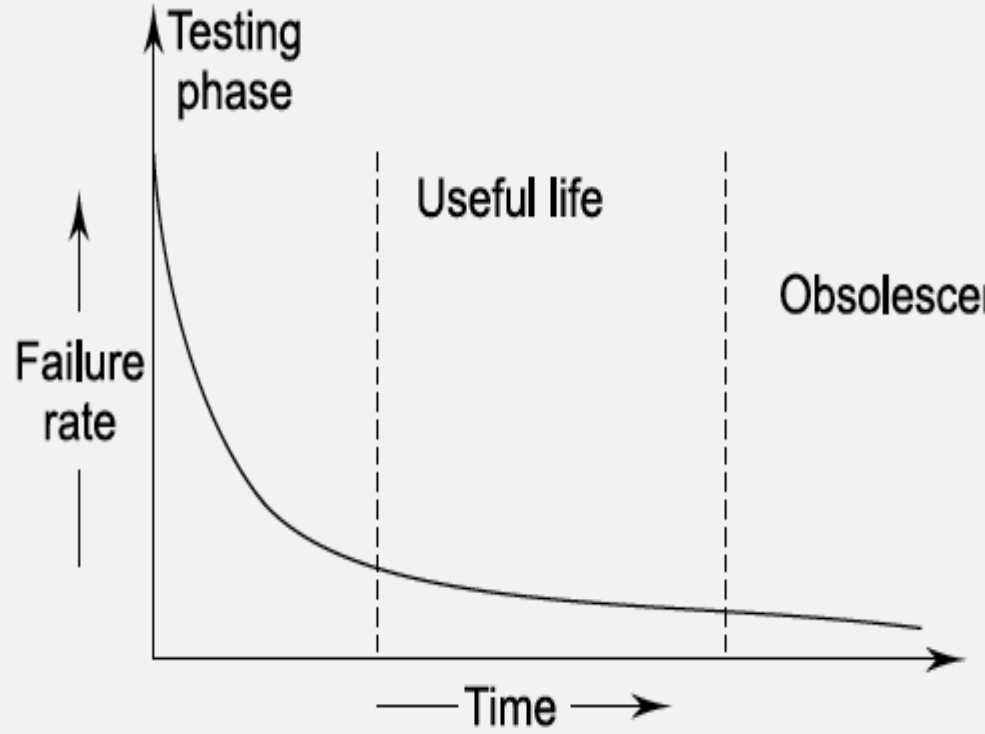

FIGURE 4.5 Software Reliability Curve (Failure Rate Versus Time)

Software may be retired only if it becomes obsolete. Some of the contributing factors are given below:

- Change in environment
- Change in infrastructure/technology
- Major change in requirements
- Increase in complexity
- Extremely difficult to maintain
- Deterioration in structure of the code
- Slow execution speed
- Poor graphical user interfaces

5.7.2 Software-Reliability Specifications

- **Reliability** is a complex concept, which should always be considered at the system level rather than the individual component level. Because the components in a system are interdependent, a failure in one component can be propagated through the system and affect the operation of other components. In a computer-based system, you have to consider three dimensions when specifying the overall system reliability:

1. ***Hardware Reliability.*** What is the probability of a hardware component failing and how long does it take to repair that component?

2. ***Software Reliability.*** How likely is it that a software component will producean incorrect output? Software failures are different from hardware failures in that software does not wear out. It can continue operating correctly after an incorrect result has been produced.

3. ***Operator Reliability. How likely is it that the operator of a system will make*** an error?

- The reliability of a system depends upon a number of factors rather than the sum of all the probabilities (failure probabilities of each factor).
- $P_S = P_A + P_R + \ldots\ldots\ldots\ldots + P_N$,

where

  - P_S = Probability of system failure
  - P_A = Probability of component A failure
  - P_B = Probability of component B failure
  - P_N = Probability of component N failure

- As the number of factors increase, the overall probability of system failure increases.

5.7.3 Reliability Terminologies

TABLE 4.3 Reliability Terminologies

Term	Description
System failure	An event that occurs at some point in time when the system does not deliver a service as expected by its users.
System error	Erroneous system behavior where the behavior of the system does not conform to its specifications.
System fault	An incorrect system state, i.e., a system state that is unexpected by the designers of the system.
Human errors or mistakes	Human behavior that results in the introduction of faults into a system.

5.7.4 Classification of Failures

- When a system specifies the reliability then the engineer should identify the failure type and consider whether it should be treated differently in specification. Many large systems are divided into small subsystems and each subsystem has different reliability requirements. It is easy and cheap to assess the reliability requirements of each subsystem separately.

TABLE 4.4 Failure Classifications

Failure Class	Description
Recoverable	The system can recover with or without operator intervention.
Transient	Occurs only with certain inputs.
Unrecoverable	The system cannot recover without operator intervention or the system may need to be restarted.
Corrupting	Failure corrupts system data.
Non-corrupting	Failure does not corrupt system data.
Permanent	Occurs for all input values while invoking a function of the system.

5.8 RELIABILITY METRICS

Reliability metrics *are used to quantitatively express the reliability of a software product.*

- Some reliability metrics, which can be used to quantify the reliability of a software product are:

1. **MTTF (Mean Time to Failure).**
   - **Components have an average lifetime and this is reflected in the most widely used hardware reliability metric, Mean Time to Failure (MTTF).**
   - The MTTF is the mean time for which a component is expected to be operational.
   - The MTTF is the average time between observed system failures.
   - An MTTF of 500 means that one failure can be expected every 500 time units. The time units are totally dependent on the system and it can even be specified in the number of transactions, as is the case of database query systems.
   - MTTF is relevant for systems with long transactions, i.e., where system processing takes a long time.
   - MTTF should be longer than transaction length. For example, it is suitable for computer-aided design systems where a designer will work on a design for several hours as well as for word-processor systems.

2. MTTR (Mean Time to Repair).

- **Once failure occurs, some time is required** to fix the error.
- **MTTR** measures the average time it takes to track the errors causing the failure and then to fix them.

OR

- MTTR is the average time to replace a defective component.
- Once a hardware component fails then the failure is usually permanent so the Mean Time to Repair (MTTR) reflects the time taken to repair or replace the component.

3. MTBF (Mean Time Between Failures).

- **We can combine the MTTF and** MTTR metrics to get the MTBF metric:
- **MTBF = MTTF + MTTR**
- Thus, a MTBF of 300 hours indicates that once a failure occurs, the next failure is expected to occur only after 300 hours. In this case, the time measurements are real time and not the execution time as in MTTF.

4. POFOD (Probability of Failure on Demand).

- **POFOD is the likelihood** that the system will fail when a service request is made.
- **A POFOD of 0.001** means that one out of a thousand service requests may result in failure.
- POFOD is an important measure for safety critical systems and should be kept as low as possible. It is relevant for many safety-critical systems with the exception of management components, such as an emergency shutdown system in a chemical plant.

5. ROCOF (Rate of Occurrences of Failure).

- **ROCOF is the frequency of** occurrence with which unexpected behavior is likely to occur.
- **A ROCOF of 2/100** means that two failures are likely to occur in each 100 operational time units.
- **This metric is sometimes called the failure intensity**. It is relevant for operating systems and transaction-processing systems where the system has to process a large number of similar requests that are relatively frequent; for example, credit-card processing systems, airlinebooking systems, etc.

6. AVAIL (Availability).

- **Availability is the probability that the system is** available for use at a given time. An availability of 0.998 means that in every 1000 time units, the system is likely to be available for 998 of these.

5.8.1 Measurements of Reliability and Availability Measurement of Reliability

- Most hardware-related reliability models are predicted on failure due to wear rather than failure due to design defects.
  - In hardware, failures due to physical wear (e.g., effect of temperature corrosion) are more likely than a design-related failure. There has been debate over the relationship between key concepts in hardware reliability and their applicability to software. Although an irrefutable link has yet to be established.
- If we consider a computer-based system, **a simple measure of reliability** is **mean-time-between-failure (MTBF**) and it can be expressed as:
- **MTBF = MTTF + MTTR,**

  where
  - **MTTF = Mean Time to Failure**
  - **MTTR = Mean Time to Repair**.

Measurement of Availability

- Many researchers argue that MTBF is a far more useful measure than defects/KLOC or defects/FP because an end-user is concerned with failures, not with the total error count. Because each error contained within a program does not have the same failure rate, the total error count provides little indication of the reliability of a system. In addition to the reliability measure, we must develop a measure of availability. Software availability is the probability that a program is operating according to requirements at a given point in time and is defined as:
- **Availability = (MTTF/(MTTF +MTTR)) × 100%**
- The MTBF reliability measure is equally sensitive to MTTF and MTTR. The availability measure is somewhat more sensitive to MTTR.

5.9 RELIABILITY GROWTH MODELING

- A reliability growth model is *a mathematical model of how software reliability improves as errors are detected and repaired.*

- A reliability growth model can be used to predict when a particular level of reliability is likely to be attained.
  - Thus, reliability growth modeling can be used to determine when to stop testing to attain a given reliability level. Although several reliability growth models have been proposed as discussed in the following sections.

5.9.1 Jelinski-Moranda Model

- The Jelinski-Moranda model is *the earliest and probably the best-known reliability model*. It proposes a failure intensity function in the form of
- $\lambda(t) = \varphi(N - i + 1)$,

  where
  - φ = Constant of proportionality
  - N = Total number of errors present
  - *i* = *Number of errors found by time interval* t_i
- This model assumes that all failures have the same failure rate. It means that failure rate is a step function and there will be an improvement in reliability after fixing a fault. So, every failure contributes equally to the overall reliability.

Here, failure intensity is directly proportional to the number of remaining errors in a program. The relation between time and failure intensity is shown in Figure 4.6.

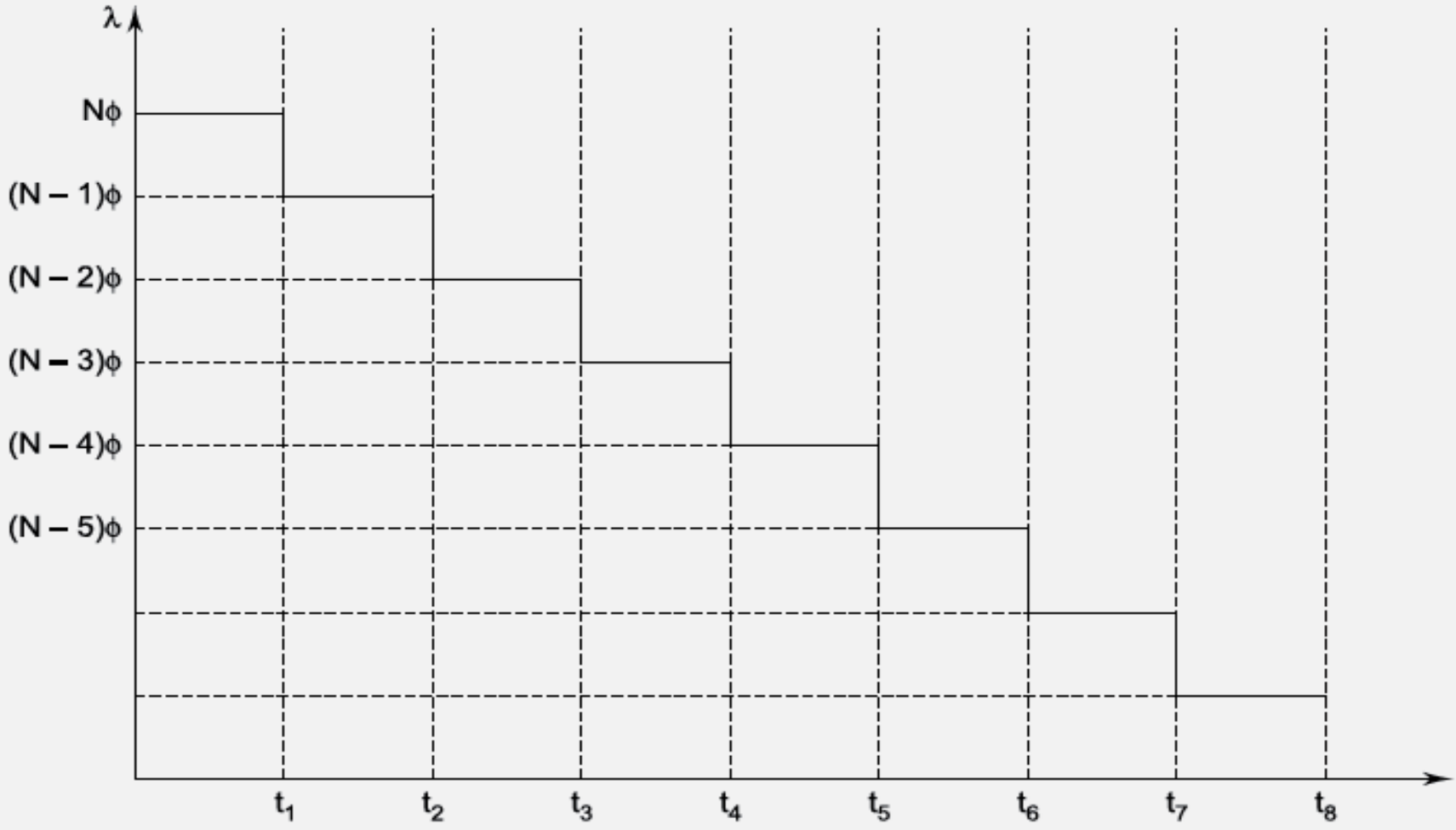

FIGURE 4.6 Relation Between T and λ

- The time interval $t_1, t_2 \ldots t_k$ *may vary in duration depending upon the occurrence* of a failure. Between the $(i-1)^{th}$ *and* i^{th} *failure, the failure intensity function is* $(N - i + 1)\varphi$.

- **Example 4.1.** *There are 50 errors estimated to be present in a program. We have experienced 30 errors. Use the Jelinski-Moranda model to calculate the failure intensity with a given value of* $\varphi = 0.03$. *What will be the failure intensity after the experience of 40 errors?*

- **Solution.**
  - N = 50 errors; *i = 30 failures;* $\varphi = 0.03$

  We know
  - $\lambda(t) = \varphi(N - i + 1) = 0.03\ (50 - 30 + 1) = 0.63$ failures/CPU hr.
  - After 40 failures $\lambda(t) = 0.03\ (50 - 40 + 1) = 0.33$ failures/CPU hr.

- Hence, there is a continuous decrease in the failure intensity as the number of failures experienced increases.

5.9.2 Little Wood and Verall's Model

- This model allows for negative reliability growth to reflect the fact that when a repair is carried out, it may introduce additional errors.
- It also models the fact that as errors are repaired, the average improvement in reliability per repair decreases.
- It treats an error's contribution to reliability improvement as an independent random variable having Gamma distribution. This distribution models the fact that error corrections with large contributions to reliability growth are removed first. This represents diminishing return as testing continues.
- There are more complex reliability growth models that give greater accurate approximations to reliability growth. However, these models are beyond the scope of this text.

5.9.3 Step Function Model

- The simplest reliability growth model is a step function model, where it is assumed that the reliability increases by a constant increment each time an error is detected and repaired. Such a model is shown in Figure 4.7.
- However, this simple model of reliability, which implicitly assumes that all errors contribute equally to reliability growth, is highly unrealistic since we already know that corrections of different errors contribute differently to reliabilitygrowth.

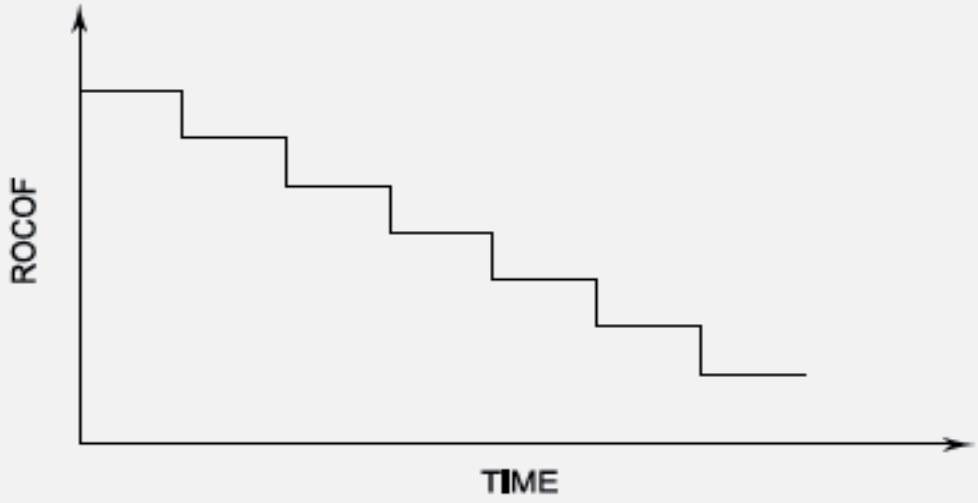

FIGURE 4.7 Step Function Model of Reliability Growth

5.10 RELIABILITY ASSESSMENT

- In general, many properties of engineering artifacts, such as reliability, are measured and verified in this way. For instance, the reliability of an electrical appliance may be measured in terms of its probability of failure within a given time. This measure is helpful whenever we cannot guarantee absence of failures absolutely. Software reliability is the probability of the failure-free operation of a computer program for a specified time in a specified environment.

- The process of measuring the reliability of a system is illustrated in Figure 4.8.

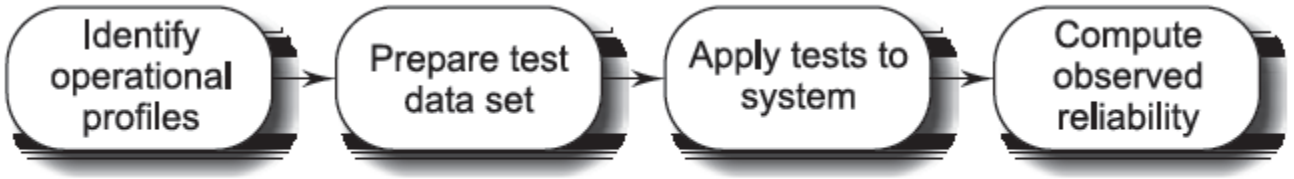

FIGURE 4.8 The Reliability Measurement Process

This process involves four stages:

- Existing systems of the same type are studied to establish an operational profile. An operational profile identifies different classes of system inputs and the probability of these inputs in normal use.
- A set of test data is constructed (sometimes with the help of test-data generators) that reflects the operational profile.
- The system is tested with these data and the number of failures is observed.
- The times of these failures are also logged; the time units chosen should be appropriate for the reliability metric used.
- After a statistically significant number of failures have been observed, the software reliability can then be computed. You can then work out the appropriate reliability metric value.

This approach to reliability measurement is not easy to apply in practice. The principal difficulties that arise are due to:

- Operational profile uncertainty. The operational profiles may not be an accurate reflection of the real use of the system.
- High costs of test-data generation. Defining a large amount of test data takes a long time if it is not possible to generate this data automatically.
- Statistical uncertainty when high reliability is specified. It is important to generate a statistically significant number of failures to allow accurate reliability measurements.

EXERCISES

1. State some of the software quality factors that are proposed by McCall.
2. Define representative qualities.
3. Describe the various classifications of software quality.
4. How do you define reliability? Discuss various models for reliability allocation.
5. What is software quality assurance?
6. What are the goals of software quality assurance?
7. What are the objectives of software quality assurance?
8. Define an SQA plan.
9. Why is it important for a software-development organization to obtain ISO-9000 certification?
10. Discuss the main requirements of the ISO-9001 and compare it with the SEI-CMM.
11. Discuss how reliability changes over the lifetime of a software product and a hardware product.
12. Define:
    (*i) Software quality*
    (*ii) Reliability metrics*
    (*iii) Software reliability*

13. Compare:
    (*i) ISO and SEI-CMM*
    (*ii) Software and hardware reliability*
14. Explain, in detail, the SEI-CMM model. Also, differentiate it with ISO.
15. Give the shortcomings of ISO-9000 certification.
16. Explain any two types of reliability growth models.
17. What is software reliability and software availability? Also, discuss how they are measured.
18. What is software quality?
19. Explain the steps an organization will take in order to obtain ISO-9000 certification.
20. Answer the following questions:
    (*i) Can a program be correct and still not be reliable? Explain.*
    (*ii) What is ISO-9000 certification?*
    (*iii) What are the salient features of ISO-9001 requirements?*
    (*iv) What is software reliability? Explain.*

Tanya Jawab

The End