MODULE 5: SOFTWARE QUALITY

13.1 INTRODUCTION

Quality:

- Quality is generally agreed to be **' a good thing'** .
- In a practice what people really mean by the 'quality' of a system can be vague, undefined attribute.
- We need to define precisely what qualities we require of a system.

Objective assessment:

- We need to judge objectively whether a system meets our quality requirements and this need measurement.
- Critical for package selection, e.g., Brigette at Brightmouth College.

Now days, delivering a high-quality product is one of the major objectives of all organizations. Traditionally, the quality of a product means that how much it gets fit into its specified purpose. A product is of *good quality if it performs according to the user's requirements*. Good quality software should meet all objectives defined in the SRS document. It is the responsibility of the quality managers to ensure that the software attains a required level of quality.

Waiting until the system is complete to measure quality is too late.

During development, it's important to:

- Assess the likely quality of the final system.
- Ensure development methods will produce the desired quality.

Potential customers, like Amanda at IOE, might focus on:

- Checking if suppliers use the best development methods.
- Ensuring these methods will lead to the required quality in the final system**.**

13.2 THE PLACE OF SOFTWARE QUALITY IN PROJECT PLANNING

Quality will be of concern at all stages of project planning and execution, but will be particular interest at Stepwise framework .

Step 1 : Identifying project scope and objectives Some objective could relate to the quality of the application to be delivered.

Step 2 : Identifying project infrastructure Within this step activity 2.2 involves identifying installation standards and procedures. Some of these will almost certainly be about quality requirements.

Step 3 : Analyze project characteristics. In this activity the application to be implemented will be examined to see if it has any special quality requirements.

Example: Safety-critical applications might require additional activities such as n-version development, where multiple teams develop versions of the same software to cross-check outputs.

Step 4 : Identify the product and activities of the project. It is at that point the entry, exit and process requirement are identified for each activity.

Step 8: Review and publicize plan. At this stage the overall quality aspects of the project plan are reviewed.

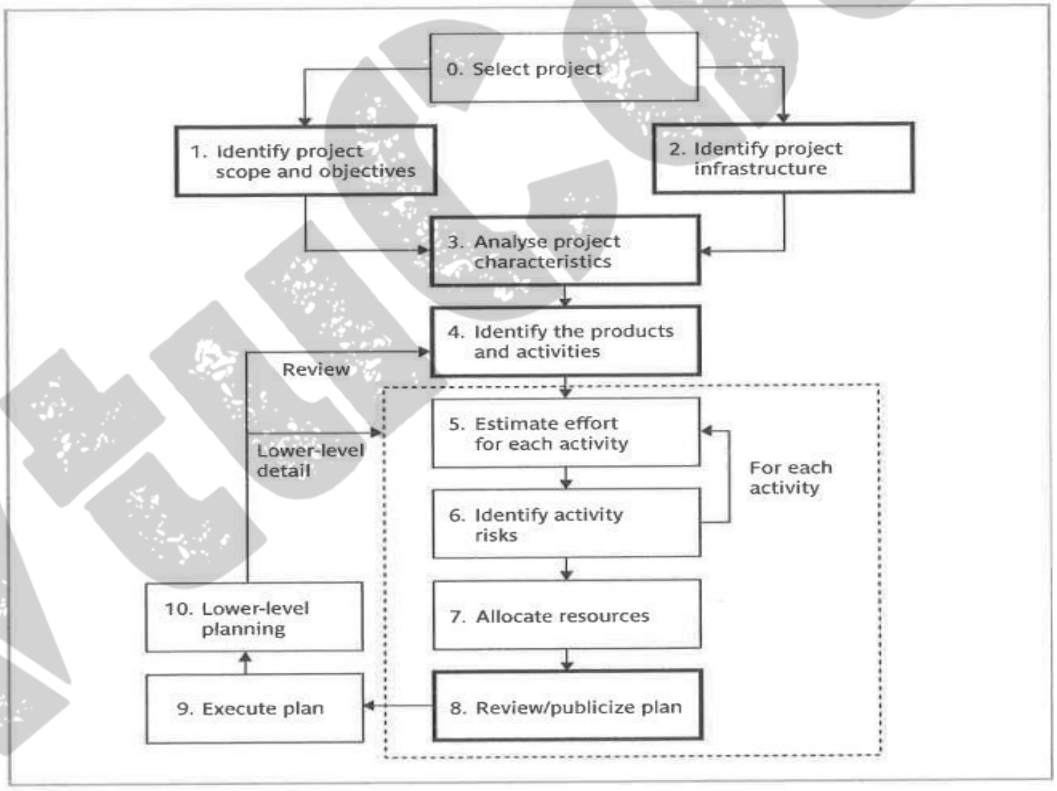


FIGURE 13.1 The place of software quality in Step Wise

13.3 THE IMPORTANCE OF SOFTWARE QUALITY

Now a days, **quality is the important aspect of all organization.** Good quality software is the requirement of all users. There are so many reasons that describe why the quality of software is important; few among of those which are most important are described below:

Increasingly criticality of software:

- The final customer or user is naturally anxious about the general quality of software especially about the reliability.
- They are concern about the safety because of their dependency on the software system such as aircraft control system are more safety critical systems.

Earlier detection of errors during development-

- As software is developed through a number of phases; output of one phase is given as input to the other one. So, if error in the initial phase is not found, then at the later stage, it is difficult to fix that error and also the cost indulged is more.

The intangibility of software :

- Difficulty in verifying the satisfactory completion of project tasks.
- Tangibility is achieved by requiring developers to produce "deliverables" that can be examined for quality.

Accumulating errors during software development :

- Errors in earlier steps can propagate and accumulate in later steps.
- Errors found later in the project are more expensive to fix.
- The unknown number of errors makes the debugging phase difficult to control.

13.4 DEFINING SOFTWARE QUALITY

Quality is a rather vague term and we need to define carefully what we mean by it. For any software system there should be three specifications.

- **A functional specification** describing what the system is to do
- A **quality specification** concerned with how well the function are to operate
- A **resource specification** concerned with how much is to be spent on the system.

Attempt to identify specific product qualities that are appropriate to software , for instance, grouped software qualities into three sets. Product operation qualities, Product revision qualities and product transition qualities.

Product operation qualities:

Correctness: The extent to which a program satisfies its specification and fulfil user objective.

Reliability: the extent to which a program can be expected to perform its intended function with required precision.

Efficiency: The amounts of computer resource required by software.

Integrity: The extent to which access to software or data by unauthorized persons can be controlled.

Usability: The effort required to learn, operate, prepare input and interprets output.

Product Revision Qualities:

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 effort required to modify an operational program.

Product Transition Qualities:

Portability: The efforts require to transfer a program from one hardware configuration and or 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.

When there is concerned about the need for a specific quality characteristic in a software product then a quality specification with the following minimum details should be drafted .

1. **Definition/Description**
   Definition: Clear definition of the quality characteristic.
   Description: Detailed description of what the quality characteristic entails.

2. **Scale o Unit of Measurement:**
   The unit used to measure the quality characteristic (e.g., faults per thousand lines of code).

3. **Test**
   Practical Test: The method or process used to test the extent to which the quality attribute exists.

4. **Minimally Acceptable**
   Worst Acceptable Value: The lowest acceptable value, below which the product would be rejected.

5. **Target Range**
   Planned Range: The range of values within which it is planned that the quality measurement value should lie.

6. **Current Value**
   Now: The value that applies currently to the quality characteristic.

Measurements Applicable to Quality Characteristics in Software

When assessing quality characteristics in software, multiple measurements may be applicable. For example, in the case of reliability, measurements could include:

1. **Availability:**
   Definition: Percentage of a particular time interval that a system is usable.
   Scale: Percentage (%).
   Test: Measure the system's uptime versus downtime over a specified period.
   Minimally Acceptable: Typically, high availability is desirable; specifics depend on system requirements.
   Target Range: E.g., 99.9% uptime.

2. **Mean Time Between Failures (MTBF):**
   Definition: Total service time divided by the number of failures.
   Scale: Time (e.g., hours, days).
   Test: Calculate the average time elapsed between system failures.
   Minimally Acceptable: Longer MTBF indicates higher reliability; minimum varies by system criticality.
   Target Range: E.g., MTBF of 10,000 hours.

3. **Failure on Demand:**
   Definition: Probability that a system will not be available when required, or probability that a transaction will fail.
   Scale: Probability (0 to 1).
   Test: Evaluate the system's response to demand or transaction processing.
   Minimally Acceptable: Lower probability of failure is desired; varies by system criticality.
   Target Range: E.g., Failure on demand probability of less than 0.01.

4. **Support Activity:**
   Definition: Number of fault reports generated and processed.
   Scale: Count (number of reports).
   Test: Track and analyze the volume and resolution time of fault reports.
   Minimally Acceptable: Lower number of fault reports indicates better reliability.
   Target Range: E.g., Less than 10 fault reports per month.

Maintainability and Related Qualities:

- **Maintainability**: How quickly a fault can be corrected once detected.
- **Changeability**: Ease with which software can be modified.
- **Analyzability**: Ease with which causes of failure can be identified, contributing to maintainability. These measurements help quantify and assess the reliability and maintainability of software systems, ensuring they meet desired quality standards.

13.5 SOFTWARE QUALITY MODELS

It is hard to directly measure the quality of a software. However, it can be expressed in terms of several attributes of a software that can be directly measured.

The quality models give a characterization (hierarchical) of software quality in terms of a set of characteristics of the software. The bottom level of the hierarchical can be directly measured, thereby enabling a quantitative assessment of the quality of the software.

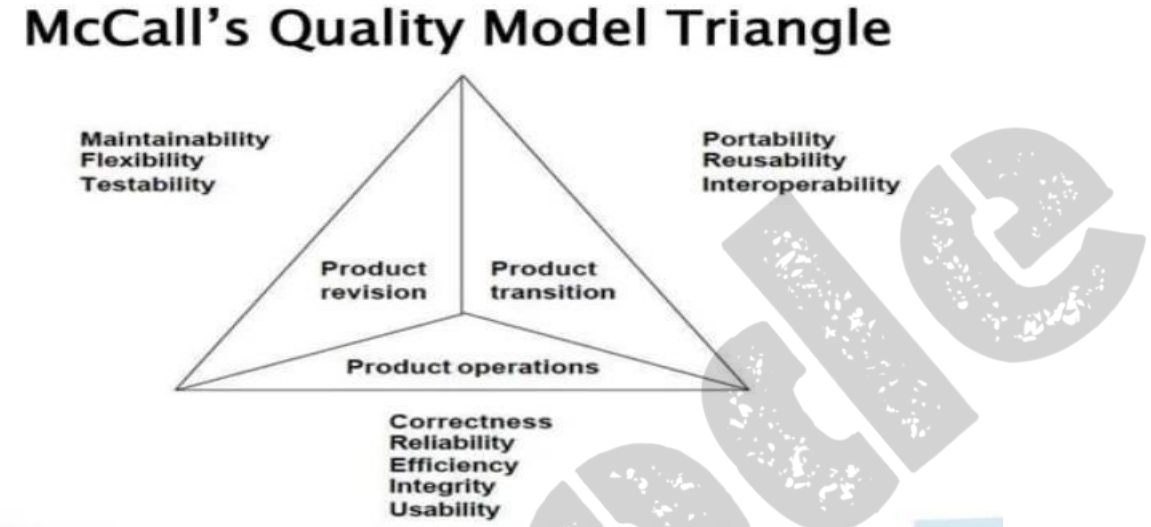
There are several well-established quality models.

1. McCall's model.
2. Dromey's Model.
3. Boehm's Model.
4. ISO 9126 Model.

Garvin's Quality Dimensions: David Gravin , a professor of Harvard Business school, defined the quality of any product in terms of eight general attributes of the product.

- Performance: How well if performs the jobs.
- Features: How well it supports the required features.
- Reliability: Probability of a product working satisfactorily within a specified period of time.
- Conformance: Degree to which the product meets the requirements.
- Durability: Measure of the product life.
- Serviceability: Speed and effectiveness maintenance.
- Aesthetics: The look and feel of the product.
- Perceived quality: User's opinion about the product quality.

1) **McCall' Model:** McCall defined the quality of a software in terms of three broad parameters: its operational characteristics, how easy it is to fix defects and how easy it is to part it to different platforms. These three high-level quality attributes are defined based on the following eleven attributes of the software:



Correctness: The extent to which a software product satisfies its specifications.
Reliability: The probability of the software product working satisfactorily over a given duration.
Efficiency: The amount of computing resources required to perform the required functions.
Integrity: The extent to which the data of the software product remain valid.
Usability: The effort required to operate the software product.
Maintainability: The ease with which it is possible to locate and fix bugs in the software product.
Flexibility: The effort required to adapt the software product to changing requirements.
Testability: The effort required to test a software product to ensure that it performs its intended function.
Portability: The effort required to transfer the software product from one hardware or software system environment to another.
Reusability: The extent to which a software can be reused in other applications.
Interoperability: The effort required to integrate the software with other software.

2) **Dromey's model:** Dromey proposed that software product quality depends on four major high-level properties of the software: Correctness, internal characteristics, contextual characteristics and certain descriptive properties. Each of these high-level properties of a software product, in turn depends on several lower-level quality attributes. Dromey's hierarchical quality model is shown in Fig 13.2

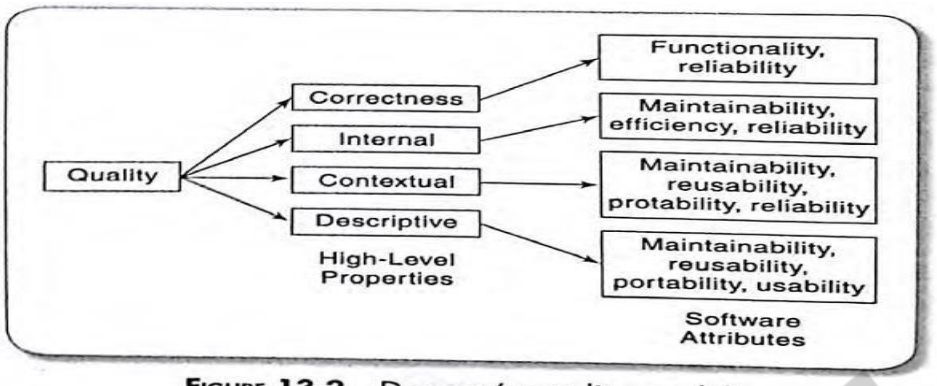


FIGURE 13.2 Dromey's quality model

3) Boehm's Model: Boehm's suggested that the quality of a software can be defined based on these high-level characteristics that are important for the users of the software. These three high-level characteristics are the following:

As-is -utility: How well (easily, reliably and efficiently) can it be used?

Maintainability: How easy is to understand, modify and then retest the software?

Portability: How difficult would it be to make the software in a changed environment?

Boehm's expressed these high-level product quality attributes in terms of several measurable product attributes. Boehm's hierarchical quality model is shown in Fig 13.3 .

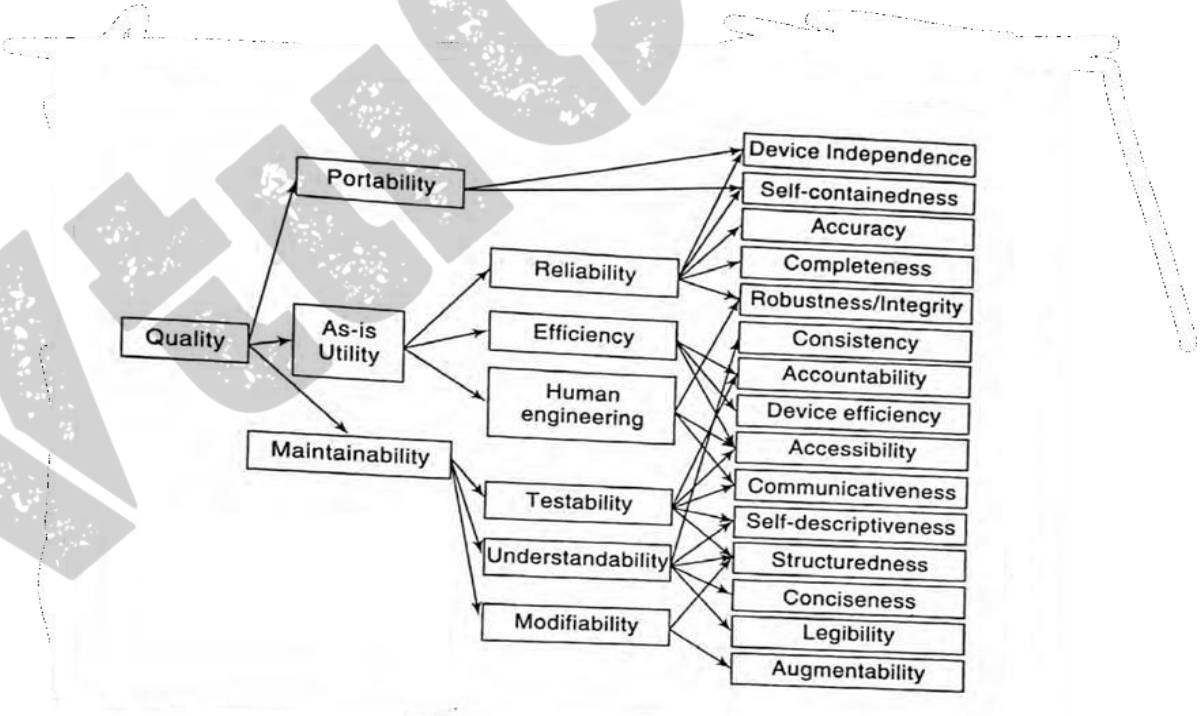


FIGURE 13.3 Boehm's quality model

13.6 ISO 9126

- ISO 9126 standards was first introduced in 1991 to tackle the question of the definition of software quality. The original 13-page document was designed as a foundation upon which further more detailed standard could be built. ISO9126 documents are now very lengthy.

Motivation might be-

- **Acquires** who are obtaining software from external suppliers
- **Developers** who are building a software product
- **Independent evaluators** who are accessing the quality of a software product, not for themselves but for a community of user.

- ISO 9126 also introduces another type of elements – quality in use- for which following element has been identified
  - **Effectiveness** : the ability to achieve user goals with accuracy and completeness;
  - **Productivity** : avoiding the excessive use of resources, such as staff effort, in achieving user goals;
  - **Safety**: within reasonable levels of risk of harm to people and other entities such as business, software, property and the environment
  - **Satisfaction**: smiling users

ISO 9126 is a significant standard in defining software quality attributes and providing a framework for assessing them. Here are the key aspects and characteristics defined by ISO 9126:

ISO 9126 identifies six major external software quality **characteristics**:

1. **Functionality:**
   Definition: The functions that a software product provides to satisfy user needs.
   Sub-characteristics: Suitability, accuracy, interoperability, security, compliance.

Characteristic	Sub-characteristics
Functionality	Suitability
	Accuracy
	Interoperability
	Functionality compliance
	Security

'**Functionality Compliance'** refers to the degree to which the software adheres to application-related standard or legal requirements. Typically, these could be auditing requirement. '**Interoperability**' refers to the ability of software to interact with others.

2. Reliability:

Definition: The capability of the software to maintain its level of performance under stated conditions**.**

Sub-characteristics: Maturity, fault tolerance, recoverability**.**

Characteristic	Sub-characteristics
Reliability	Maturity
	Fault tolerance
	Recoverability
	Reliability compliance

Maturity refers to frequency of failures due to fault in software more identification of fault more changes to remove them. **Recoverability** describes the control of access to a system.

3. Usability:

Definition: The effort needed to use the software**.**

Sub-characteristics: Understandability, learnability, operability, attractiveness.

Characteristic	Sub-characteristics
Usability	Understandability
	Learnability
	Operability
	Attractiveness
	Usability compliance

Understand ability is a clear quality to grasp. Although the definition attributes that bear on the user efforts for recognizing the logical concept and its applicability in our actually makes it less clear. **Learnability** has been distinguished from **operability**. A software tool might be easy to learn but time consuming to use say it uses a large number of nested menus. This is for a package that is used only intermittently but not where the system is used or several hours each day by the end user. In this case learnability has been incorporated at the expense of operability

4. **Efficiency**:

   Definition: The ability to use resources in relation to the amount of work done**.**

   Sub-characteristics: Time behavior, resource utilization.

5. **Maintainability:**

   Definition: The effort needed to make changes to the software.

   Sub-characteristics: Analyzability, modifiability, testability.

Characteristic	Sub-characteristics
Efficiency	Time behaviour
	Resource utilization
	Efficiency compliance
Maintainability	Analysability
	Changeability
	Stability
	Testability
	Maintainability compliance

Analysability is the quality that McCall called diagnose ability, the ease with which the cause of failure can be determined. **Changeability** is the quality that others call flexibility : the latter name implies suppliers of the software are always changing it. **Stability** means that there is a low risk of a modification to software having unexpected effects.

6. **Portability:**

   Definition: The ability of the software to be transferred from one environment to another**.** \

   Sub-characteristics: Adaptability, install ability, co-existence.

Characteristic	Sub-characteristics
Portability	Adaptability
	Installability
	Coexistence
	Replaceability
	Portability compliance

Portability compliance relates to those standards that have a bearing on portability. **Replaceability** refers to the factors that give upward compatibility between old software components and the new ones. **'Coexistence'** refers to the ability of the software to share resources with other software components; unlike' interoperability', no direct data passing is necessarily involved

ISO 9126 provides guidelines for the use of the quality characteristics.

ISO 9126 provides structured guidelines for assessing and managing software quality characteristics based on the specific needs and requirements of the software product. It emphasizes the variation in importance of these characteristics depending on the type and context of the software product being developed.

Steps Suggested After Establishing Requirements

Once the software product requirements are established, ISO 9126 suggests the following steps:

1. **Specify Quality Characteristics**: Define and prioritize the relevant quality characteristics based on the software's intended use and stakeholder requirements.
2. **Define Metrics and Measurements**: Establish measurable criteria and metrics for evaluating each quality characteristic, ensuring they align with the defined objectives and user expectations.
3. **Plan Quality Assurance Activities**: Develop a comprehensive plan for quality assurance activities, including testing, verification, and validation processes to ensure adherence to quality standards.
4. **Monitor and Improve Quality:** Continuously monitor software quality throughout the development lifecycle, identifying areas for improvement and taking corrective actions as necessary.
5. **Document and Report:** Document all quality-related activities, findings, and improvements, and provide clear and transparent reports to stakeholders on software quality status and compliance.

1. **Judge the importance of each quality characteristic for the application.**
   - Identify key quality characteristics (e.g., Usability, Efficiency, Maintainability).
   - Assign importance ratings to these characteristics based on their relevance to the application.

Importance of Quality Characteristics:

Reliability: Critical for safety-critical systems where failure can have severe consequences. Measures like mean time between failures (MTBF) are essential.

Efficiency: Important for real-time systems where timely responses are crucial. Measures such as response time are key indicators.

2. **Select the external quality measurements within the ISO 9126 framework relevant to the qualities prioritized above.**

Determine appropriate **external quality measurements** that correspond to each quality characteristic.

- **Reliability**: Measure with MTBF or similar metrics.
- **Efficiency**: Measure with response time or time behavior metrics.

3. **Map measurements onto ratings that reflect user satisfaction. for example, the mappings might be as in Table 13.1 .**

   For response time, user satisfaction could be mapped as follows (hypothetical example):
   Excellent: Response time < 1 second
   Good: Response time 1-3 seconds
   Acceptable: Response time 3-5 seconds
   Poor: Response time > 5 seconds

4. **Identify the relevant internal measurements and the intermediate products in which they appear.**

   - Identify and track internal measurements such as cyclomatic complexity, code coverage, defect density, etc.
   - Relate these measurements to intermediate products like source code, test cases, and documentation.

5. **Overall assessment of product quality: To what extent is it possible to combine ratings for different quality characteristics into a single overall rating for the software?**

   - Use weighted quality scores to assess overall product quality.
   - Focus on key quality requirements and address potential weaknesses early to avoid the need for an overall quality rating later.

Based on the ISO 9126 framework and your points:

1. Measurement Indicators Across Development Stages:

- **Early Stages**: Qualitative indicators like checklists and expert judgments are used to assess compliance with predefined criteria. These are subjective and based on qualitative assessments.

- **Later Stages**: Objective and quantitative measurements become more prevalent as the software product nears completion. These measurements provide concrete data about software performance and quality attributes.

2. Overall Assessment of Product Quality:

- Combining ratings for different quality characteristics into a single overall rating for software is challenging due to:
  - Different measurement scales and methodologies for each quality characteristic.
  - Sometimes, enhancing one quality characteristic (e.g., efficiency) may compromise another (e.g., portability).
- Balancing these trade-offs can be complex and context-dependent.

3. Purpose of Quality Assessment:

Software Development: Assessment focuses on identifying weaknesses early, guiding developers to meet quality requirements, and ensuring continuous improvement throughout the development lifecycle.

Software Acquisition: Helps in evaluating software products from external suppliers based on predefined quality criteria.

Independent Assessment: Aims to provide an unbiased evaluation of software quality for stakeholders like regulators or consumers.

It seems like you're describing a method for evaluating and comparing software products based on their quality characteristics. Here's a summary and interpretation of your approach:

TABLE 13.2 Mapping response times onto user satisfaction

Response time (seconds)	Quality score
<2	5
2–3	4
4–5	3
6–7	2
8–9	1
>9	0

The table 13.2 shows how different response times are mapped to quality scores on a scale of 0-5, with shorter response times receiving higher scores. A rating scale (e.g., 1-5) is used to reflect the importance of various quality characteristics.

1. **Rating for User Satisfaction:**
   - Products are evaluated based on mandatory quality levels that must be met. Beyond these mandatory levels, user satisfaction ratings in the range of 0 to 5 are assigned for other desirable characteristics.
   - Objective measurements of functions are used to determine different levels of user satisfaction, which are then mapped to numerical ratings (see Table 13.2 for an example).
2. **Importance Weighting**:
   - Each quality characteristic (e.g., usability, efficiency, maintainability) is assigned an importance rating on a scale of 1 to 5.
   - These importance ratings reflect how critical each quality characteristic is to the overall evaluation of the software product.
3. **Calculation of Overall Score:**
   - Weighted scores are calculated for each quality characteristic by multiplying the quality score by its importance weight.
   - The weighted scores for all characteristics are summed to obtain an overall score for each software product.
4. **Comparison and Preference Order:**
   - Products are then ranked in order of preference based on their overall scores. Higher scores indicate products that are more likely to satisfy user requirements and preferences across the evaluated quality characteristics.

This method provides a structured approach to evaluating software products based on user satisfaction ratings and importance weights for quality characteristics. It allows stakeholders to compare and prioritize products effectively based on their specific needs and preferences.

TABLE 13.3 Weighted quality scores

Product quality	Importance rating (a)	Product A		Product B	
		Quality score (b)	Weighted score (a × b)	Quality score (c)	Weighted score (a × c)
Usability	3	1	3	3	9
Efficiency	4	2	8	2	8
Maintainability	2	3	6	1	2
Overall			17		19

- This table 13.3 provides a comparison of two products (A and B) based on weighted quality scores.
- Each product quality (Usability, Efficiency, Maintainability) is given an importance rating.
- Product A and B are scored for each quality, and these scores are multiplied by the importance rating to obtain weighted scores.

- The total weighted scores are summed for each product to determine their overall ranking.

It involves assessing software products by assigning quality scores to various characteristics, weighting these scores by their importance, and summing them to get an overall score. This approach helps to objectively compare products based on user satisfaction and key quality metrics.

By assigning scores to various qualities, weighting them by their importance, and summing these to get an overall score, it provides a comprehensive way to compare and rank software products. This ensures that both essential and desirable characteristics are considered in the assessment, leading to a more balanced and objective evaluation.

13.7 PRODUCT AND PROCESS METRICS

Users assess the quality of a software product based on its external attributes, whereas during development, the developers assess the product's quality based on various internal attributes.

The internal attributes may measure either some aspects of product or of the development process(called process metrics).

1. **Product Metrics:**

   Purpose: Measure the characteristics of the software product being developed.

   Examples:

   Size Metrics: Such as Lines of Code (LOC) and Function Points, which quantify the size or complexity of the software.

   Effort Metrics: Like Person-Months (PM), which measure the effort required to develop the software.

   Time Metrics: Such as the duration in months or other time units needed to complete the development.

2. **Process Metrics:**

   Purpose: Measure the effectiveness and efficiency of the development process itself**.**

   Examples:

   Review Effectiveness: Measures how thorough and effective code reviews are in finding defects.

   Defect Metrics: Average number of defects found per hour of inspection, average time taken to correct defects, and average number of failures detected during testing per line of code.

   Productivity Metrics: Measures the efficiency of the development team in terms of output per unit of effort or time.

   Quality Metrics: Such as the number of latent defects per line of code, which indicates the robustness of the software after development.

Differences:

- **Focus**: Product metrics focus on the characteristics of the software being built (size, effort, time), while process metrics focus on how well the development process is performing (effectiveness, efficiency, quality).
- **Use**: Product metrics are used to gauge the attributes of the final software product, aiding in planning, estimation, and evaluation. Process metrics help in assessing and improving the development process itself, aiming to enhance quality, efficiency, and productivity.
- **Application**: Product metrics are typically applied during and after development phases to assess the product's progress and quality. Process metrics are applied throughout the development lifecycle to monitor and improve the development process continuously.

By employing both types of metrics effectively, software development teams can better manage projects, optimize processes, and deliver high-quality software products that meet user expectations.

13.8 PRODUCT VERSUS PROCESS QUALITY MANAGEMENT

In software development, managing quality can be approached from two main perspectives: product quality management and process quality management. Here's a breakdown of each approach and their key aspects:

Product Quality Management

Product quality management focuses on evaluating and ensuring the quality of the software product itself. This approach is typically more straightforward to implement and measure after the software has been developed.

Aspects:

1. Measurement Focus: Emphasizes metrics that assess the characteristics and attributes of the final software product, such as size (LOC, function points), reliability (defects found per LOC), performance (response time), and usability (user satisfaction ratings).

2. Evaluation Timing: Product quality metrics are often measured and evaluated after the software product has been completed or at significant milestones during development.

3. Benefits:

- Provides clear benchmarks for evaluating the success of the software development project.
- Facilitates comparisons with user requirements and industry standards.
- Helps in identifying areas for improvement in subsequent software versions or projects.

4. Challenges:

- Predicting final product quality based on intermediate stages (like early code modules or prototypes) can be challenging.
- Metrics may not always capture the full complexity or performance of the final integrated product.

Process Quality Management

Process quality management focuses on assessing and improving the quality of the development processes used to create the software. This approach aims to reduce errors and improve efficiency throughout the development lifecycle.

Aspects:

1. Measurement Focus: Emphasizes metrics related to the development processes themselves, such as defect detection rates during inspections, rework effort, productivity (e.g., lines of code produced per hour), and adherence to defined standards and procedures.

2. Evaluation Timing: Process quality metrics are monitored continuously throughout the development lifecycle, from initial planning through to deployment and maintenance.

3. Benefits:

- Helps in identifying and correcting errors early in the development process, reducing the cost and effort of rework.
- Facilitates continuous improvement of development practices, leading to higher overall quality in software products.
- Provides insights into the effectiveness of development methodologies and practices used by the team.

4. Challenges:

- Requires consistent monitoring and analysis of metrics throughout the development lifecycle.
- Effectiveness of process improvements may not always translate directly into improved product quality without careful management and integration.

Integration and Synergy

- While product and process quality management approaches have distinct focuses, they are complementary.
- Effective software development teams often integrate both approaches to achieve optimal results.

- By improving process quality, teams can enhance product quality metrics, leading to more reliable, efficient, and user-friendly software products.

13.9 QUALITY MANGEMENT SYSTEMS

BS EN ISO 9001:2000

The British Standards institution (BSI) have engaged in the creation of standards for quality management systems. The British Standards is now called BS EN ISO 9001:2000, which is identical to the international standard, ISO 9001:2000. ISO 9000 describes the fundamental features of a quality management system (QMS) and its terminology. ISO 9001 describes how a QMS can be applied to a creation of products and provision of services. ISO 9004 applies to process improvement.

An overview of BS EN ISO 9001:2000 QMS requirements

ISO 9001:2000 is part of the ISO 9000 series, which sets forth guidelines and requirements for implementing a Quality Management System (QMS). The focus of ISO 9001:2000 is on ensuring that organizations have effective processes in place to consistently deliver products and services that meet customer and regulatory requirements.

Key Elements:

1. **Fundamental Features:**
   - Describes the basic principles of a QMS, including customer focus, leadership, involvement of people, process approach, and continuous improvement.
   - Emphasizes the importance of a systematic approach to managing processes and resources.

2. **Applicability to Software Development:**
   - ISO 9001:2000 can be applied to software development by ensuring that the development processes are well-defined, monitored, and improved.
   - It focuses on the development process itself rather than the end product certification (unlike product certifications such as CE marking).

3. **Certification Process:**
   - Organizations seeking ISO 9001:2000 certification undergo an audit process conducted by an accredited certification body.
   - Certification demonstrates that the organization meets the requirements of ISO 9001:2000 and has implemented an effective QMS.

4. **Quality Management Principles:**
   - **Customer focus:** Meeting customer requirements and enhancing customer satisfaction.
   - **Leadership**: Establishing unity of purpose and direction.
   - **Involvement of people:** Engaging the entire organization in achieving quality objectives.
   - **Process approach**: Managing activities and resources as processes to achieve desired outcomes.
   - **Continuous improvement:** Continually improving QMS effectiveness.

Application to Software Development

In software development scenarios, ISO 9001:2000 helps organizations:

- Define and document processes related to software development, testing, and maintenance.
- Establish quality objectives and metrics for monitoring and evaluating software development processes.
- Implement corrective and preventive actions to address deviations from quality standards.
- Ensure that subcontractors and external vendors also adhere to quality standards through effective quality assurance practices.

Criticisms and Considerations

Perceived Value: Critics argue that ISO 9001 certification does not guarantee the quality of the end product but rather focuses on the process.

Cost and Complexity: Obtaining and maintaining certification can be costly and time-consuming, which may pose challenges for smaller organizations.

Focus on Compliance: Some organizations may become overly focused on meeting certification requirements rather than improving overall product quality.

Despite these criticisms, ISO 9001:2000 provides a structured framework that, when implemented effectively, can help organizations improve their software development processes and overall quality management practices. Measurement - to demonstrate that products conform to standards, and the QMS is effective, and to improve the effectiveness of processes that create products or services.

It emphasizes continuous improvement and customer satisfaction, which are crucial aspects in the competitive software industry.

BS EN ISO 9001:2000 outlines comprehensive requirements for implementing a Quality Management System (QMS). Here's an overview of its key requirements and principles:

Principles of BS EN ISO 9001:2000

1. **Customer Focus:**
   Understanding and meeting customer requirements to enhance satisfaction.
2. **Leadership:**
   Providing unity of purpose and direction for achieving quality objectives.
3. **Involvement of People:**
   Engaging employees at all levels to contribute effectively to the QMS.
4. **Process Approach:**
   - ❖ Focusing on individual processes that create products or deliver services.
   - ❖ Managing these processes as a system to achieve organizational objectives.
5. **Continuous Improvement**:
   - ❖ Continually enhancing the effectiveness of processes based on objective measurements and analysis.
6. **Factual Approach to Decision Making:**
   Making decisions based on analysis of data and information.
7. **Mutually Beneficial Supplier Relationships:**
   Building and maintaining good relationships with suppliers to enhance capabilities and performance.

Activities in BS EN ISO 9001:2000 Cycle

1. **Understanding Customer Needs:**
   Identifying and defining customer requirements and expectations.
2. **Establishing Quality Policy:**
   Defining a framework for quality objectives aligned with organizational goals.
3. **Process Design:**
   Designing processes that ensure products and services meet quality objectives.
4. **Allocation of Responsibilities:**
   Assigning responsibilities for meeting quality requirements within each process.
5. **Resource Management:**
   Ensuring adequate resources (human, infrastructure, etc.) are available for effective process execution.
6. **Measurement and Monitoring:**
   - ❖ Designing methods to measure and monitor process effectiveness and efficiency.
   - ❖ Gathering data and identifying discrepancies between actual performance and targets.

7. **Analysis and Improvement:**
   Analyzing causes of discrepancies and implementing corrective actions to improve processes continually.

Detailed Requirements

1. Documentation:

- Maintaining documented objectives, procedures (in a quality manual), plans, and records that demonstrate adherence to the QMS.
- Implementing a change control system to manage and update documentation as necessary.

2. **Management Responsibility:**
   Top management must actively manage the QMS and ensure that processes conform to quality objectives.
3. **Resource Management:**
   Ensuring adequate resources, including trained personnel and infrastructure, are allocated to support QMS processes.
4. **Production and Service Delivery:**
   - ❖ Planning, reviewing, and controlling production and service delivery processes to meet customer requirements.
   - ❖ Communicating effectively with customers and suppliers to ensure clarity and alignment on requirements.
5. **Measurement, Analysis, and Improvement:**
   - ❖ Implementing measures to monitor product conformity, QMS effectiveness, and process improvements.
   - ❖ Using data and information to drive decision-making and enhance overall organizational performance.

13.10 PROCESS CAPABILITY MODELS

The evolution of quality assurance paradigms from product assurance to process assurance marks a significant shift in how organizations ensure quality in their outputs. Here's an overview of some key concepts and methodologies related to process-based quality management:

Historical Perspective

1. **Before the 1950s:**
   Quality assurance primarily focused on extensive testing of finished products to identify defects.

2. **Shift to Process Assurance**:
    - ❖ Later paradigms emphasize that ensuring a good quality process leads to good quality products.
    - ❖ Modern quality assurance techniques prioritize recognizing, defining, analyzing, and improving processes.

Total Quality Management (TQM)

1. **Definition:**
    - ❖ TQM focuses on continuous improvement of processes through measurement and redesign.
    - ❖ It advocates that organizations continuously enhance their processes to achieve higher levels of quality.

Business Process Reengineering (BPR)

1. **Objective:**
    - ❖ BPR aims to fundamentally redesign and improve business processes.
    - ❖ It seeks to achieve radical improvements in performance metrics, such as cost, quality, service, and speed

To manage quality during development, process -based techniques are very important. SEI CMM, CMMI, ISO 15504, and Six Sigma are popular capability models.

Process Capability Models

1. **SEI Capability Maturity Model (CMM) and CMMI:**
    - ❖ Developed by the Software Engineering Institute (SEI), CMM and CMMI provide a framework for assessing and improving the maturity of processes.
    - ❖ They define five maturity levels, from initial (ad hoc processes) to optimized (continuous improvement).
    - ❖ CMMI (Capability Maturity Model Integration) integrates various disciplines beyond software engineering.
2. **ISO 15504 (SPICE):**
    - ❖ ISO/IEC 15504, also known as SPICE (Software Process Improvement and Capability determination), is an international standard for assessing and improving process capability.
    - ❖ It provides a framework for evaluating process maturity based on process attributes and capabilities.

3. **Six Sigma:**
    - ❖ Six Sigma focuses on reducing defects in processes to a level of 3.4 defects per million opportunities (DPMO).

- ❖ It emphasizes data-driven decision-making and process improvement methodologies like DMAIC (Define, Measure, Analyze, Improve, Control).

Importance of Process Metrics

During Product Development:

- ❖ Process metrics are more meaningfully measured during product development compared to product metrics.
- ❖ They help in identifying process inefficiencies, bottlenecks, and areas for improvement early in the development lifecycle.

1. SEI Capability Maturity Model (SEI CMM)

The SEI Capability Maturity Model (CMM) is a framework developed by the Software Engineering Institute (SEI) to assess and improve the maturity of software development processes within organizations.

It categorizes organizations into five maturity levels based on their process capabilities and practices:

SEI CMM Levels:

1. **Level 1: Initial**

   Characteristics:
   - ❖ Chaotic and ad hoc development processes.
   - ❖ Lack of defined processes or management practices.
   - ❖ Relies heavily on individual heroics to complete projects.

   Outcome:
   - ❖ Project success depends largely on the capabilities of individual team members.
   - ❖ High risk of project failure or delays.

2. **Level 2: Repeatable**

   Characteristics:
   - ❖ Basic project management practices like planning and tracking costs/schedules are in place.
   - ❖ Processes are somewhat documented and understood by the team.

   Outcome:
   - ❖ Organizations can repeat successful practices on similar projects.
   - ❖ Improved project consistency and some level of predictability.

3. **Level 3: Defined**

Characteristics:

- Processes for both management and development activities are defined and documented.
- Roles and responsibilities are clear across the organization.
- Training programs are implemented to build employee capabilities.
- Systematic reviews are conducted to identify and fix errors early.

Outcome:

- Consistent and standardized processes across the organization.
- Better management of project risks and quality.

4. **Level 4: Managed**

Characteristics:

- Processes are quantitatively managed using metrics.
- Quality goals are set and measured against project outcomes.
- Process metrics are used to improve project performance.

Outcome:

- Focus on managing and optimizing processes to meet quality and performance goals.
- Continuous monitoring and improvement of project execution.

5. **Level 5: Optimizing**

Characteristics:

- Continuous process improvement is ingrained in the organization's culture.
- Process metrics are analyzed to identify areas for improvement.
- Lessons learned from projects are used to refine and enhance processes.
- Innovation and adoption of new technologies are actively pursued.

Outcome:

- Continuous innovation and improvement in processes.
- High adaptability to change and efficiency in handling new challenges.
- Leading edge in technology adoption and process optimization.

Use of SEI CMM:

1) **Capability Evaluation:** Used by contract awarding authorities (like the US DoD) to assess potential contractors' capabilities to predict performance if awarded a contract.
2) **Process Assessment:** Internally used by organizations to improve their own process capabilities through assessment and recommendations for improvement. SEI CMM has been instrumental not only in enhancing the software development practices within organizations but also in establishing benchmarks for industry standards.

It encourages organizations to move from chaotic and unpredictable processes (Level 1) to optimized and continuously improving processes (Level 5), thereby fostering better quality, efficiency, and predictability in software development efforts.

Key process areas

The Capability Maturity Model Integration (CMMI) is an evolutionary improvement over its predecessor, the Capability Maturity Model (CMM). Here's an overview of CMMI and its structure:

2. CMMI(Capability Maturity Model Integration):

Evolution and Purpose of CMMI

1. **Evolution from CMM to CMMI:**
   CMM Background: The original Capability Maturity Model (CMM) was developed in the late 1980s by the Software Engineering Institute (SEI) at Carnegie Mellon University, primarily to assess and improve the software development processes of organizations, particularly those contracting with the US Department of Defense.

Expansion and Adaptation: Over time, various specific CMMs were developed for different domains such as software acquisition (SA-CMM), systems engineering (SE-CMM), and people management (PCMM). These models provided focused guidance but lacked integration and consistency.

2. **Need for Integration**:
   Challenges: Organizations using multiple CMMs faced issues like overlapping practices, inconsistencies in terminology, and difficulty in integrating practices across different domains.
   CMMI Solution: CMMI (Capability Maturity Model Integration) was introduced to provide a unified framework that could be applied across various disciplines beyond just software development, including systems engineering, product development, and services.

Structure and Levels of CMMI

1. **Levels of Process Maturity:** Like CMM, CMMI defines five maturity levels, each representing a different stage of process maturity and capability.
   These levels are:
   - Level 1: Initial (similar to CMM Level 1)
   - Level 2: Managed (similar to CMM Level 2)
   - Level 3: Defined (similar to CMM Level 3)
   - Level 4: Quantitatively Managed (an extension of CMM Level 4)
   - Level 5: Optimizing (an extension of CMM Level 5)

2. **Key Process Areas (KPAs):**

   Definition: Similar to CMM, each maturity level in CMMI is characterized by a set of Key Process Areas (KPAs). These KPAs represent clusters of related activities that, when performed collectively, achieve a set of goals considered important for enhancing process capability.
   Gradual Improvement: KPAs provide a structured approach for organizations to incrementally improve their processes as they move from one maturity level to the next.
   Integration across Domains: Unlike the specific CMMs for various disciplines, CMMI uses a more abstract and generalized set of terminologies that can be applied uniformly across different domains.

Level	Key process areas
1. Initial	Not applicable
2. Managed	Requirements management, project planning and monitoring and control, supplier agreement management, measurement and analysis, process and product quality assurance, configuration management
3. Defined	Requirements development, technical solution, product integration, verification, validation, organizational process focus and definition, training, integrated project management, risk management, integrated teaming, integrated supplier management, decision analysis and resolution, organizational environment for integration
4. Quantitatively managed	Organizational process performance, quantitative project management
5. Optimizing	Organizational innovation and deployment, causal analysis and resolution

TABLE 13.4 CMMI key process areas

Benefits of CMMI

- **Broad Applicability:** CMMI's abstract nature allows it to be applied not only to software development but also to various other disciplines and industries.
- **Consistency and Integration:** Provides a unified framework for improving processes, reducing redundancy, and promoting consistency across organizational practices.
- **Continuous Improvement:** Encourages organizations to continuously assess and refine their processes to achieve higher levels of maturity and performance.

3. IS0 15504 Process assessment

ISO/IEC 15504, also known as SPICE (Software Process Improvement and Capability determination), is a standard for assessing and improving software development processes. Here are the key aspects of ISO 15504 process assessment:

Process Reference Model

• **Reference Model:** ISO 15504 uses a process reference model as a benchmark against which actual processes are evaluated. The default reference model is often ISO 12207, which outlines the processes in the software development life cycle (SDLC) such as requirements analysis, architectural design, implementation, testing, and maintenance.

Process Attributes

Nine Process Attributes: ISO 15504 assesses processes based on nine attributes, which are:

1. Process Performance (PP): Measures the achievement of process-specific objectives.

2. Performance Management (PM): Evaluates how well the process is managed and controlled.

3. Work Product Management (WM): Assesses the management of work products like requirements specifications, design documents, etc.

4. Process Definition (PD): Focuses on how well the process is defined and documented.

5. Process Deployment (PR): Examines how the process is deployed within the organization.

6. Process Measurement (PME): Evaluates the use of measurements to manage and control the process.

7. Process Control (PC): Assesses the monitoring and control mechanisms in place for the process.

8. Process Innovation (PI): Measures the degree of innovation and improvement in the process.

9. Process Optimization (PO): Focuses on optimizing the process to improve efficiency and effectiveness.

Processes are assessed on the basis of nine process attributes - see Table .l 3.5.

Level	Attribute	Comments
0. Incomplete		The process is not implemented or is unsuccessful
1. Performed process	1.1 Process performance	The process produces its defined outcomes
2. Managed process	2.1 Performance management	The process is properly planned and monitored
	2.2 Work product management	Work products are properly defined and reviewed to ensure they meet requirements
3. Established process	3.1 Process definition	The processes to be carried out are carefully defined
	3.2 Process deployment	The processes defined above are properly executed by properly trained staff
4. Predictable process	4.1 Process measurement	Quantitatively measurable targets are set for each sub-process and data collected to monitor performance
	4.2 Process control	On the basis of the data collected by 4.1 corrective action is taken if there is unacceptable variation from the targets
5. Optimizing	5.1 Process innovation	As a result of the data collected by 4.1, opportunities for improving processes are identified
	5.2 Process optimization	The opportunities for process improvement are properly evaluated and where appropriate are effectively implemented

TABLE 13.5 ISO 15504 framework for process capability

Compatibility with CMMI

Alignment with CMMI: ISO 15504 and CMMI share similar goals of assessing and improving software development processes. While CMMI is more comprehensive and applicable to a broader range of domains, ISO 15504 provides a structured approach to process assessment specifically tailored to software development.

Benefits and Application

• **Benefits**: ISO 15504 helps organizations:

1) Evaluate their current processes against a recognized standard.

2) Identify strengths and weaknesses in their processes.

3) Implement improvements based on assessment findings.

• **Application**: The standard is used by organizations to conduct process assessments either internally for improvement purposes or externally for certification purposes.

Note: For a process to be judged to be at Level 3, for example, Levels 1 and 2 must also have been achieved.

When assessors are judging the degree to which a process attribute is being fulfilled, they allocate one of the following scores:

Level	Interpretation
N – not achieved	0 to 15% achievement
P – partially achieved	15% to 50% achievement
L – largely achieved	50% to 85% achievement
F – fully achieved	85% achievement

In order to assess the process attribute of a process at being at a certain level of achievement , indicators have to be found that provide evidence for the assessment.

In the context of assessing process attributes according to ISO/IEC 15504 (SPICE), evidence is crucial to determine the level of achievement for each attribute.

Here's how evidence might be identified and evaluated for assessing the process attributes, taking the example of requirement analysis processes:

Example of Assessing Requirement Analysis Processes

1. **Process Definition (PD):**
   - **Evidence**: A section in the procedures manual that outlines the steps, roles, and responsibilities for conducting requirements analysis.
   - **Assessment**: Assessors would review the documented procedures to ensure they clearly define how requirements analysis is to be conducted. This indicates that the process is defined (3.1 in Table 13.5).
2. **Process Deployment (PR):**
   - **Evidence**: Control documents or records showing that the documented requirements analysis process has been used and followed in actual projects.
   - **Assessment**: Assessors would look for signed-off control documents at each step of the requirements analysis process, indicating that the defined process is being implemented and deployed effectively (3.2 in Table 13.5).

Using ISO/IEC 15504 Attributes

1) **Process Performance (PP):**
   - **Evidence**: Performance metrics related to the effectiveness and efficiency of the requirements analysis process, such as the accuracy of captured requirements, time taken for analysis, and stakeholder satisfaction surveys.
   - **Assessment**: Assessors would analyze the metrics to determine if the process meets its performance objectives (e.g., accuracy, timeliness).
2) **Process Control (PC):**
   - **Evidence**: Procedures and mechanisms in place to monitor and control the requirements analysis process, such as regular reviews, audits, and corrective action reports.
   - **Assessment**: Assessors would review the control mechanisms to ensure they effectively monitor the process and address deviations promptly.
3) **Process Optimization (PO):**
   - **Evidence**: Records of process improvement initiatives, feedback mechanisms from stakeholders, and innovation in requirements analysis techniques.
   - **Assessment**: Assessors would examine how the organization identifies opportunities for process improvement and implements changes to optimize the requirements analysis process.

Importance of Evidence

- Objective Assessment: Evidence provides objective data to support the assessment of process attributes.
- Validation: It validates that the process attributes are not just defined on paper but are effectively deployed and managed.
- Continuous Improvement: Identifying evidence helps in identifying areas for improvement and optimizing processes over time.

Implementing process improvement

Implementing process improvement in UVW, especially in the context of software development for machine tool equipment, involves addressing several key challenges identified within the organization. Here's a structured approach, drawing from CMMI principles, to address these issues and improve process maturity:

Identified Issues at UVW

1. **Resource Overcommitment:**
   Issue: Lack of proper liaison between the Head of Software Engineering and Project Engineers leads to resource overcommitment across new systems and maintenance tasks simultaneously.
   Impact: Delays in software deliveries due to stretched resources**.**

2. **Requirements Volatility:**
   Issue: Initial testing of prototypes often reveals major new requirements.
   Impact: Scope creep and changes lead to rework and delays.
3. **Change Control Challenges:**
   Issue: Lack of proper change control results in increased demands for software development beyond original plans.
   Impact: Increased workload and project delays.
4. **Delayed System Testing:**
   Issue: Completion of system testing is delayed due to a high volume of bug fixes.
   Impact: Delays in product release and customer shipment.

Steps for Process Improvement

1. **Formal Planning and Control**
   Objective: Introduce structured planning and control mechanisms to assess and distribute workloads effectively.
   Actions:
   - ❖ Implement formal project planning processes where software requirements are mapped to planned work packages.
   - ❖ Define clear milestones and deliverables, ensuring alignment with both hardware and software development phases.
   - ❖ Monitor project progress against plans to identify emerging issues early.

   Expected Outcomes:
   - ❖ Improved visibility into project status and resource utilization.
   - ❖ Early identification of potential bottlenecks or deviations from planned schedules.
   - ❖ Enable better resource allocation and management across different projects.

2. **Change Control Procedures**
   Objective: Establish robust change control procedures to manage and prioritize system changes effectively.
   Actions:
   - ❖ Define a formal change request process with clear documentation and approval workflows.
   - ❖ Ensure communication channels between development teams, testing groups, and project stakeholders are streamlined for change notifications.
   - ❖ Implement impact assessment mechanisms to evaluate the effects of changes on project timelines and resources.

   Expected Outcomes:

Reduced scope creep and unplanned changes disrupting project schedules.
Enhanced control over system modifications, minimizing delays and rework.

3. **Enhanced Testing and Validation**

Objective: Improve testing and validation processes to reduce delays in system testing and bug fixes.

Actions:

- ❖ Strengthen collaboration between development and testing teams to ensure comprehensive test coverage early in the development lifecycle.
- ❖ Implement automated testing frameworks where feasible to expedite testing cycles.
- ❖ Foster a culture of quality assurance and proactive bug identification throughout the development phases.

Expected Outcomes:

- ❖ Faster turnaround in identifying and resolving bugs during testing.
- ❖ Timely completion of system testing phases, enabling on-time product releases.

Moving Towards Process Maturity Levels

Level 1 to Level 2 Transition:

Focus: Transition from ad-hoc, chaotic practices to defined processes with formal planning and control mechanisms.

Benefits: Improved predictability in project outcomes, better resource management, and reduced project risks.

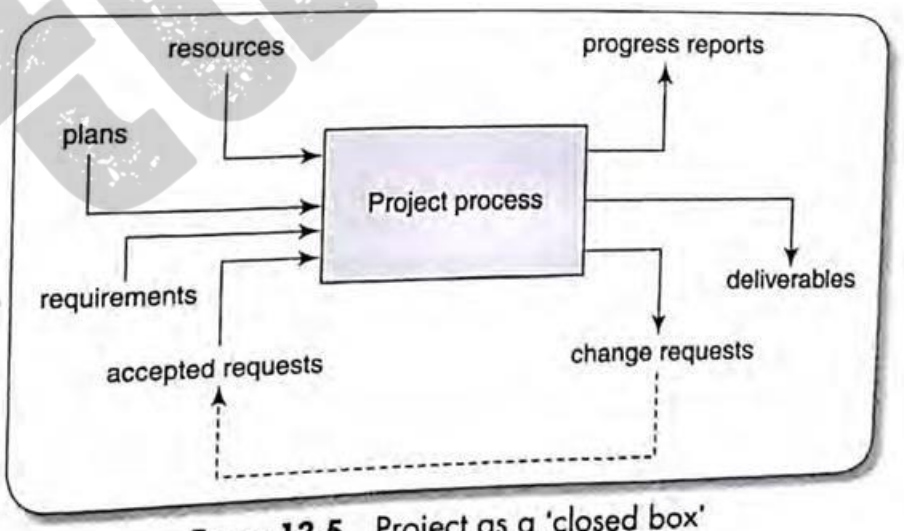


FIGURE 13.5 Project as a 'closed box'

The next step would be to identify the processes involved in each stage of the development life cycle. As in Fig 13.6. The steps of defining procedures for each development task and ensuring that they are actually carried out help to bring an organization up to Level 3.

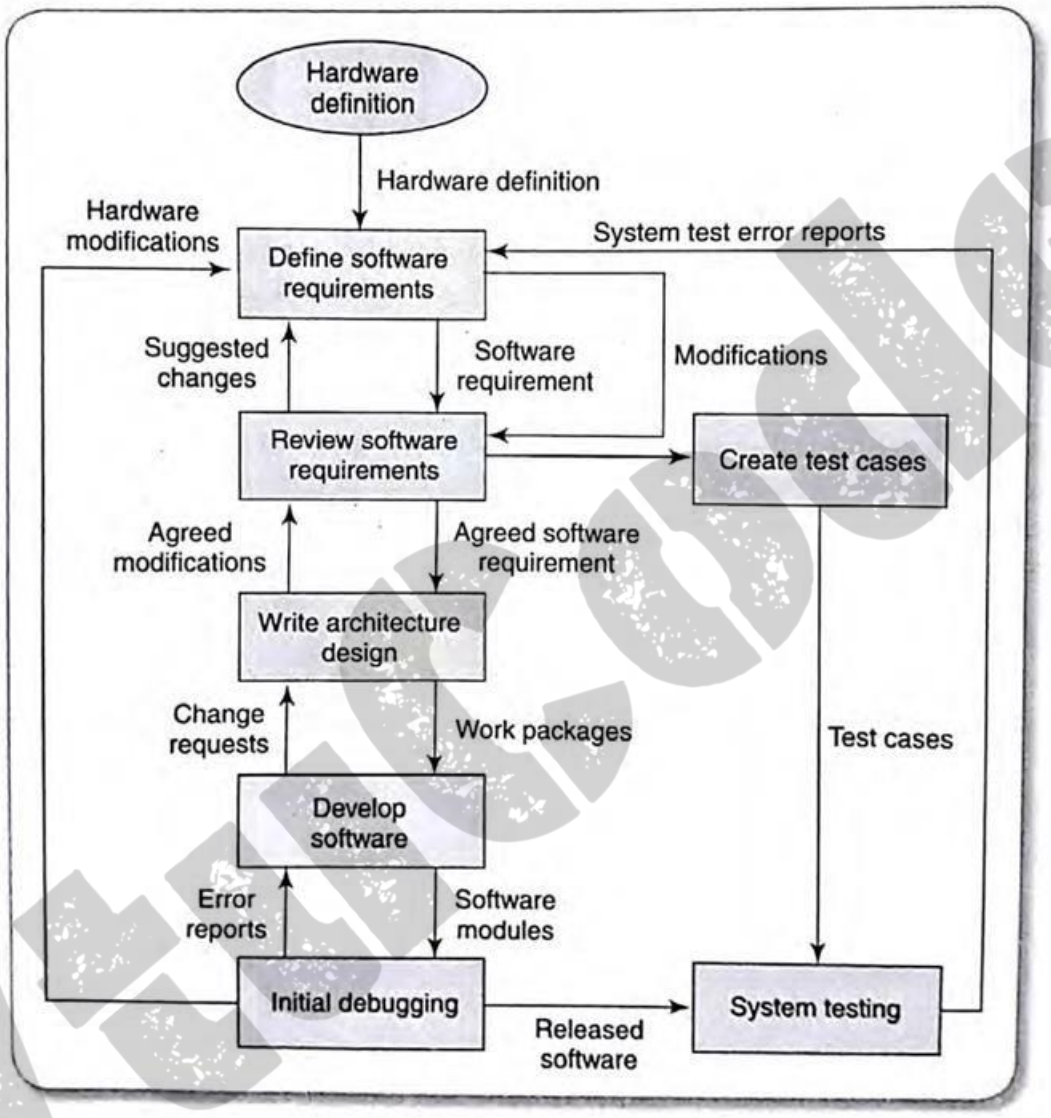


FIGURE 13.6 Process diagram

Personal Software Process (PSP)

PSP is based on the work of Watts Humphrey. PSP is suitable for individual use. PSP is a framework that helps engineers to measure and improve the way they work .It helps in developing personal skills and methods by estimating, planning, and tracking performance against plans, and provides a defined process which can be tuned by individuals.

Time Management: PSP advocates that developers should rack the way they spend time. The actual time spent on a task should be measured with the help of a stop-clock to get an objective picture of the time spent. An engineer should measure the time he spends for various development activities such as designing, writing code, testing etc.

PSP Planning: Individuals must plan their project. The developers must estimate the maximum. minimum, and the average LOC required for the product. They record the plan data in a project plan summary.

The PSP is schematically shown in Figure 13.7 . As an individual developer must plan the personal activities and make the basic plans before starting the development work. While carrying out the activities of different phases of software development, the individual developer must record the log data using time measurement.

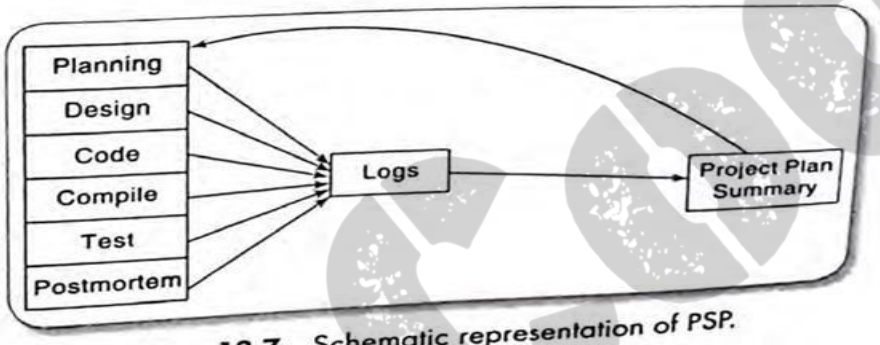


FIGURE 13.7 Schematic representation of PSP.

During post implementation project review, the developer can compare the log data with the initial plan to achieve better planning in the future projects, to improve his process etc. The four maturity levels of PSP have schematically been shown in Fig 13.8. The activities that the developer must perform for achieving a higher level of maturity have also been annotated on the diagram.

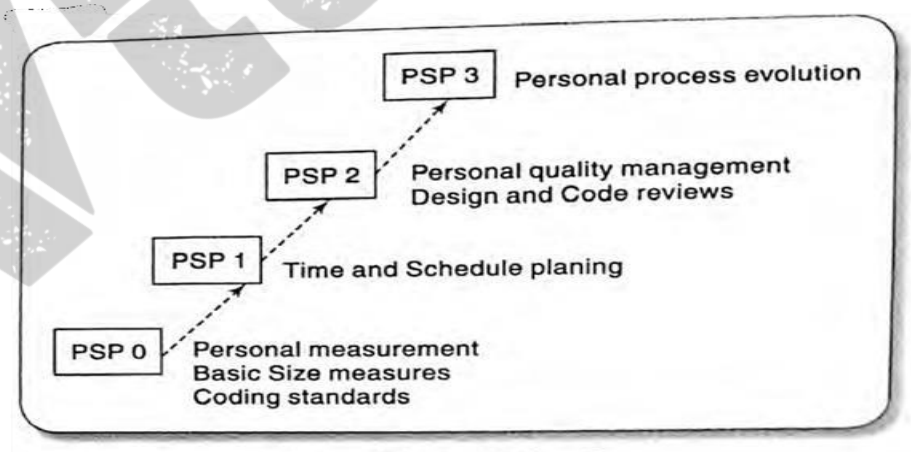


FIGURE 13.8 PSP levels

Six Sigma

- Motorola, USA , initially developed the six-sigma method in the early 1980s. The purpose of six sigma is to develop processes to do things better, faster, and at a lower cost.
- Six sigma becomes applicable to any activity that is concerned with cost, timeliness, and quality of results. Therefore, it is applicable to virtually every industry.
- Six sigma seeks to improve the quality of process outputs by identifying and removing the causes of defects and minimizing variability in the use of process.
- Six sigma is essentially a disciplined, data-driven approach to eliminate defects in any process. The statistical representation of six sigma describes quantitatively how a process is performing. To achieve six sigma, a process must not produce more than 3.4 defects per million defect opportunities.
- A six-sigma defect is defined as any system behavior that is not as per customer specifications. Total number of six sigma defect opportunities is then the total number of chances for committing an error. Sigma of a process can easily be calculated using a six-sigma calculator.

Implementing Six Sigma Methodology at UVW

UVW, a company specializing in machine tool equipment with sophisticated control software, can benefit significantly from implementing Six Sigma methodologies. Here's how UVW can adopt and benefit from

Six Sigma: Overview of Six Sigma Six Sigma is a disciplined, data-driven approach aimed at improving process outputs by identifying and eliminating causes of defects, thereby reducing variability in processes. The goal is to achieve a level of quality where the process produces no more than 3.4 defects per million opportunities.

Steps to Implement Six Sigma at UVW

1. **Define:**
   Objective: Clearly define the problem areas and goals for improvement.
   Action: Identify critical processes such as software development, testing, and deployment where defects and variability are impacting quality and delivery timelines.

2. **Measure:**
   Objective: Quantify current process performance and establish baseline metrics.
   Action: Use statistical methods to measure defects, cycle times, and other relevant metrics in software development and testing phases.

3. **Analyze:**
   Objective: Identify root causes of defects and variability in processes.
   Action: Conduct thorough analysis using tools like root cause analysis, process mapping, and statistical analysis to understand why defects occur and where process variations occur.

4. **Improve**
   Objective: Implement solutions to address root causes and improve process performance.
   Action: Develop and implement process improvements based on the analysis findings. This could include standardizing processes, enhancing communication between teams (e.g., software development and testing), and implementing better change control procedures.

5. **Control**:
   Objective: Maintain the improvements and prevent regression.
   Action: Establish control mechanisms to monitor ongoing process performance. Implement measures such as control charts, regular audits, and performance reviews to sustain improvements.

Application to UVW's Software Development

Focus Areas:

- Addressing late deliveries due to resource overcommitment.
- Managing requirements volatility and change control effectively. o Enhancing testing processes to reduce defects and delays in system testing phases.

Tools and Techniques:

- Use of DMAIC (Define, Measure, Analyse, Improve, Control) for existing process improvements.
- Application of DMADV (Define, Measure, Analyse, Design, Verify) for new process or product development to ensure high-quality outputs from the outset.

Benefits of Six Sigma at UVW

- **Improved Quality:** Reduced defects and variability in software products.
- **Enhanced Efficiency**: Streamlined processes leading to faster delivery times.
- **Cost Savings:** Reduced rework and operational costs associated with defects.

13.11 TECHNIQUES TO HELP ENHANCE SOFTWARE QUALITY

The discussion highlights several key themes in software quality improvement over time, emphasizing shifts in practices and methodologies:

Three main themes emerge:

Increasing visibility: A landmark in this movement towards making the software development process more visible was the advocacy by the American software guru, Gerald Weinberg of egoless programming. Weinberg encouraged the simple practice of programmer looking at each other code.

Procedure structure:

- Initially, software development lacked structured methodologies, but over time, methodologies with defined processes for every stage (like Agile, Waterfall, etc.) have become prevalent.
- Structured programming techniques and 'clean-room' development further enforce procedural rigor to enhance software quality

Checking intermediate stages:

- Traditional approaches involved waiting until a complete, albeit imperfect, version of software was ready for debugging.
- Contemporary methods emphasize checking and validating software components early in development, reducing reliance on predicting external quality from early design documents.

Inspections :

- Inspections are critical in ensuring quality at various development stages, not just in coding but also in documentation and test case creation.
  It is very effective way of removing superficial errors from a piece of work.
  - It motivates developers to produce better structured and self-explanatory software.
  - It helps spread good programming practice as the participants discuss the advantages and disadvantages of specific piece of code.
  - It enhances team spirit.
- Techniques like Fagan inspections, pioneered by IBM, formalize the review process with trained moderators leading discussions to identify defects and improve quality.

Japanese Quality Techniques:

- Learnings from Japanese quality practices, such as quality circles and continuous improvement, have influenced global software quality standards, emphasizing rigorous inspection and feedback loops.

Benefits of Inspections:

- Inspections are noted for their effectiveness in eliminating superficial errors, motivating developers to write better-structured code, and fostering team collaboration and spirit.
- They also facilitate the dissemination of good programming practices and improve overall software quality by involving stakeholders from different stages of development.

The general principles behind Fagan method

- Inspections are carried out on all major deliverables.
- All types of defects are noted.
- Inspection can be carried out by colleagues at all levels except the very top.
- Inspection can be carried using a predefined set of steps.
- Inspection meeting do not last for more than two hours.
- The inspection is led by a moderator who has had specific training in the techniques.
- The participants have define rules.
- Checklist are used to assist the fault-finding process.
- Material is inspected at an optimal rate of about 100 lines an hour.
- Statistics are maintained so that the effectiveness of the inspection process can be monitored.

Structured programming and clean room software development

The late 1960s marked a pivotal period in software engineering where the complexity of software systems began to outstrip the capacity of human understanding and testing capabilities. Here are the key developments and concepts that emerged during this time:

1. **Complexity and Human Limitations:**
   - Software systems were becoming increasingly complex, making it impractical to test every possible input combination comprehensively.
   - Edsger Dijkstra and others argued that testing could only demonstrate the presence of errors, not their absence, leading to uncertainty about software correctness.
2. **Structured Programming:**
   - To manage complexity, structured programming advocated breaking down software into manageable components.
   - Each component was designed to be self-contained with clear entry and exit points, facilitating easier understanding and validation by human programmers.
3. **Clean-Room Software Development:**
   - Developed by Harlan Mills and others at IBM, clean-room software development introduced a rigorous methodology to ensure software reliability.

- It involved three separate teams:
  - **Specification Team**: Gathers user requirements and usage profiles.
  - **Development Team**: Implements the code without conducting machine testing; focuses on formal verification using mathematical techniques.
  - **Certification Team**: Conducts testing to validate the software, using statistical models to determine acceptable failure rates.

4. **Incremental Development:**
   - Systems were developed incrementally, ensuring that each increment was capable of operational use by end-users.
   - This approach avoided the pitfalls of iterative debugging and ad-hoc modifications, which could compromise software reliability.

5. Verification and Validation:
   - Clean-room development emphasized rigorous verification at the development stage rather than relying on extensive testing to identify and fix errors.
   - The certification team's testing was thorough and continued until statistical models showed that the software failure rates were acceptably low.

Overall, these methodologies aimed to address the challenges posed by complex software systems by promoting structured, systematic development processes that prioritize correctness from the outset rather than relying on post hoc testing and debugging.

Clean-room software development, in particular, contributed to the evolution of quality assurance practices in software engineering, emphasizing formal methods and rigorous validation techniques.

Formal methods

- Clean-room development, uses mathematical verification techniques. These techniques use unambiguous, mathematically based , specification language of which Z and VDM are examples. They are used to define preconditions and postconditions for each procedure.
- Precondition define the allowable states, before processing, of the data items upon which a procedure is to work.
- Post condition define the state of those data items after processing. The mathematical notation should ensure that such a specification is precise and unambiguous.

Software quality circles(SWQC)

- SWQCs are adapted from Japanese quality practices to improve software development processes by reducing errors.

- Staff are involved in the identification of sources of errors through the formation of quality circle. These can be set up in all departments of an organizations including those producing software where they are known as software quality circle(SWQC).

- A quality circle is a group of four to ten volunteers working in the same area who meet for ,say, an hour a week to identify, analyze and solve their work -related problems. One of their number is a group leader and there could be an outsider a facilitator, who can advise on procedural matters.

- Associated with quality circles is the compilation of most probable error lists. For example, at IOE, Amanda might find that the annual maintenance contracts project is being delayed because of errors in the requirements specifications.

Compilation of Most Probable Error Lists

1. Identification of Common Errors: Teams, such as those in quality circles, assemble to identify the most frequent types of errors occurring in a particular phase of software development, such as requirements specification.

2. Analysis and Documentation: The team spends time analyzing past projects or current issues to compile a list of these common errors. This list documents specific types of mistakes that have been identified as recurring.

3. Proposing Solutions: For each type of error identified, the team proposes measures to reduce or eliminate its occurrence in future projects.

For example:

Example Measure: Producing test cases simultaneously with requirements specification to ensure early validation.

Example Measure: Conducting dry runs of test cases during inspections to catch errors early in the process.

4. Development of Checklists: The proposed measures are formalized into checklists that can be used during inspections or reviews of requirements specifications. These checklists serve as guidelines to ensure that identified errors are systematically checked for and addressed.

5. Implementation and Feedback: The checklists are implemented into the software development process. Feedback mechanisms are established to evaluate the effectiveness of these measures in reducing errors and improving overall quality.

Benefits of Most Probable Error Lists

Improved Quality: By proactively identifying and addressing common errors, the quality of requirements specifications (or other phases) improves.

Efficiency: Early detection and prevention of errors reduce the need for costly corrections later in the development cycle.

Standardization: Checklists standardize the inspection process, ensuring that critical aspects are consistently reviewed. This approach aligns well with quality circles and other continuous improvement methodologies by fostering a culture of proactive problem-solving and learning from past experiences.

Lessons learnt reports

- Another way by which an organization can improve its performance is by reflecting on the performance of a project at its immediate end when the experience is still fresh. This reflection may identify lessons to be applied to future projects.
- Project Managers are required to write a **Lessons Learnt** report at the end of the project. This should be distinguished from a *Post Implementation Review (PIR).* The PIR is often produced by someone who was not involved in the original project, in order to ensure neutrality.

13.12 Testing

The final judgement of the quality of the application is whether it actually works correctly when executed.

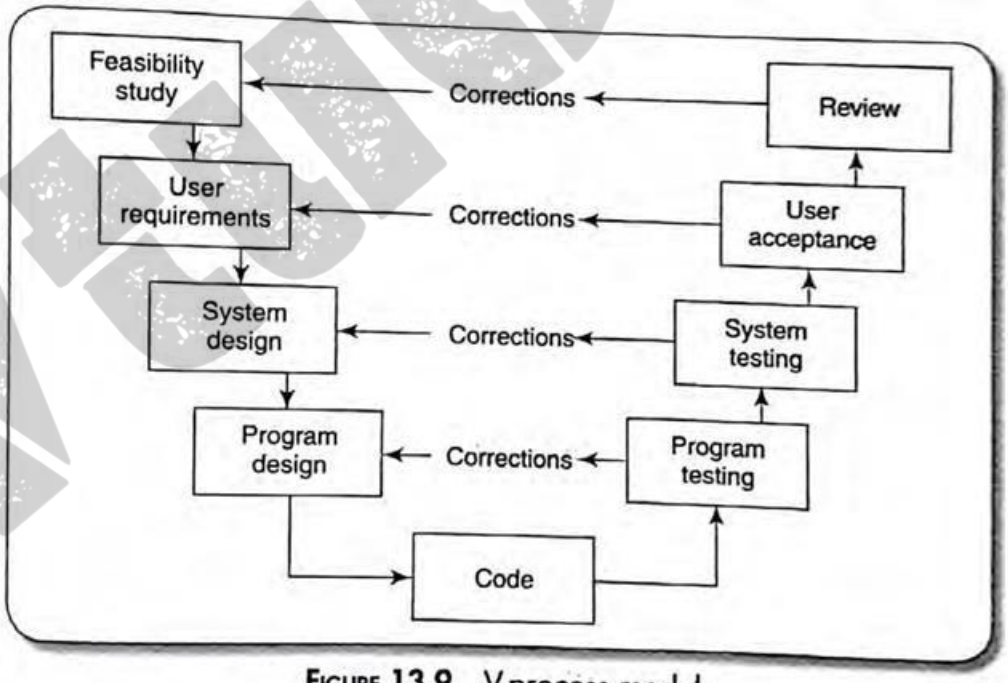


FIGURE 13.9 V-process model

- Considering the diagrammatic representation of V-Model , which stress the necessity of validation activities that match the activities creating the products of the project.
- The V-process model is expanding the activity box 'testing' in the waterfall model.
- Each step has a matching validation process which can , where defects are found, cause a loop back to the corresponding development stage and a reworking of a following step.

Verification verses Validation:

Both are bug detection techniques which helps to remove errors in software.

The main difference between these two techniques are the following:

- ❖ **Verification** is the process of determining whether the output of one phase of software development conforms to that of its previous phase. For example, a verification step can be to check if the design documents produced after the design step conform to the requirement specification.
- ❖ **Validation** is the process of determining whether fully developed software conforms to its requirements specification. Validation is applied to the fully developed and integrated software to check if it satisfies customer's requirements.

- ❖ **Verification** is carried put during development process to check of the development activities are being carried out correctly , where as
- ❖ **Validation** is carried out towards the end of the development process to check if the right product as required by the customer had been developed.

- All the boxes in the right hand of the V-process model of Fig.13.9 correspond to verification activities except the system testing block which corresponds to validation activity.

Test case Design

There are two approaches to design test cases:

1) Black-box approach
2) White-box(or glass-box) approach.

In black-box approach, test cases are designed using only the functional specification of the software. That is, test cases are designed solely based on an analysis of the input/output behavior. Hence black-box testing is also known as ***functional testing*** and also as ***requirement-driven testing.***

Design of white-box test cases requires analysis of the source code, It is also called ***structural testing or structure-driven testing.***

Levels of Testing:

A software product is normally tested at three different stages or levels. These three testing stages are:

1) Unit Testing
2) Integration Testing
3) System Testing

During unit testing, the individual components (or units) of a program are tested. For every module, unit testing is carried out as soon as the coding for it is complete. Since every module is tested separately, there is a good scope for parallel activities during unit testing.

After testing all the units individually, the units are integrated over a number of steps and tested after each step of integration (Integration testing).

Finally, the fully integration system is tested (System testing).

Testing Activities:

Testing involves performing the following main activities:

1) **Test Planning:** Test Planning consists of determining the relevant test strategies and planning for any test bed that may be required. A test bed usually includes setting up the hardware or simulator.

2) **Test Case Execution and Result Checking:** Each test case is run and the results are compared with the expected results. A mismatch between the actual result and expected results indicates a failure. The test cases for which the system fails are noted down for test reporting.

3) **Test Reporting:** When the test cases are run, the tester may raise issues, that is, report discrepancies between the expected and the actual findings. A means of formally recording these issues and their history is needed. A review body adjudicates these issues. The outcome of this scrutiny would be one of the following:
    - The issue is dismissed on the grounds that there has been a misunderstanding of a requirement by the tester.
    - The issue is identified as a fault which the developers need to correct -Where development is being done by contractors, they would be expected to cover the cost of the correction.
    - It is recognized that the software is behaving as specified, but the requirement originally agreed is in fact incorrect.
    - The issue is identified as a fault but is treated as an off-specification -It is decided that the application can be made operational with the error still in place.

4) **Debugging:** For each failure observed during testing, debugging is carried out to identify the statements that are in error.

5) **Defect Retesting:** Once a defect has been dealt with by the development team, the corrected code is retested by the testing team to check whether the defect has successfully been addressed . Defect retest is also called ***resolution testing.*** The resolution tests are a subset of the complete test suite (Fig: 13.10).

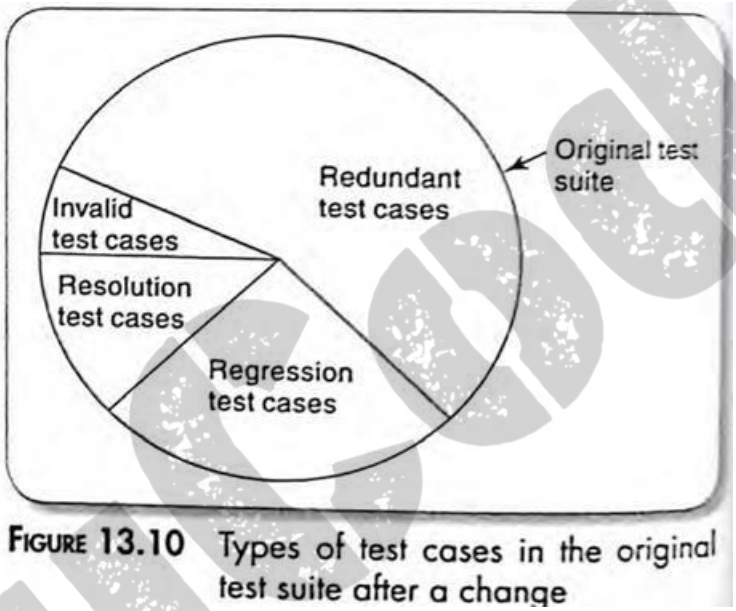


FIGURE 13.10 Types of test cases in the original test suite after a change

6) **Regression Testing:** Regression testing checks whether the unmodified functionalities still continue to work correctly. Thus, whenever a defect is corrected and the change is incorporated in the program code, the change introduced to correct an error could actually introduce errors in functionalities that were previously working correctly.

7) **Test Closure:** Once the system successfully passes all the tests, documents related to lessons learned, results, logs etc., are achieved for use as a reference in future projects.

Who Performs Testing:

Many organizations have separate system testing groups to provide an independent assessment of the correctness of software before release. In other organizations, staff is allocated to a purely testing role but work alongside the develops instead of a separate group.

Test Automation:

1) Testing is most time consuming and laborious of all software development. With the growing size of programs and the increased importance being given to product quality, test automation is drawing attention.

2) Test automation is automating one or some activities of the test process. This reduces human effort and time which significantly increases the thoroughness of testing.

3) With automation, more sophisticated test case design techniques can be deployed. By using the proper testing tools automated test results are more reliable and eliminates human errors during testing.

4) Every software product undergoes significant change overtime. Each time the code changes, it needs to be tested whether the changes induce any failures in the unchanged features. Thus the originally designed test suite need to be run repeatedly each time the code changes. Automated testing tools can be used in repeatedly running the same set of test cases.

Types of Automated Testing Tools

- **Capture and Playback Tools:** In this type of tools, the test cases are executed manually only once. During manual execution , the sequence and values of various inputs as the outputs produced are recorded. Later, the test can be automatically replayed and the results are checked against the recorded output.

  Advantage: This tool is useful for **regression testing.**

  Disadvantage: Test maintenance can be costly when the unit test changes , since some of the captured tests may become invalid.

- **Automated Test Script Tool:** Test Scripts are used to drive an automated test tool. The scripts provide input to the unit under test and record the output. The testers employ a variety of languages to express test scripts.

  Advantage: Once the test script is debugged and verified, it can be rerun a large number of times easily and cheaply.

  Disadvantage: Debugging test scripts to ensure accuracy requires significant effort.

- **Random Input Test Tools:** In this type of an automatic testing tool, test values are randomly generated to cover the input space of the unit under test. The outputs are ignored because analyzing them would be extremely expensive.

  Advantage: This is relatively easy and cost-effective for finding some types of defects.

  Disadvantage: Is very limited form of testing. It finds only the defects that crash the unit under test and not the majority of defects that do not crash but simply produce incorrect results.

- **Model-Based Test Tools:** A model is a simplified representation of program. These models can either be structural models or behavioral models. Examples of behavioral models are state models and activity models. A state model-based testing generates tests that adequately cover the state space described by the model.

Estimation of latent errors:

The methods of estimating the number of latent errors in software during testing:

1) **Using Historical Data :**If historical error data from past projects are available, you can use this data to estimate the number of errors per 1000 lines of code. This method allows you to predict the number of errors likely to be found in a new system development of a known size.

2) **Seeding Known Errors**

- During testing, seed the software with known errors.
- For example, introduce 10 known errors into the code.
- After testing, suppose 30 errors are found, of which 6 are the seeded errors.
- This implies that 60% of the seeded errors were detected.
- Thus, 40% of the errors are still undetected.
- Using this method, you can estimate the total number of errors in the software using the formula:

Estimated Total Errors=(TotalErrors Found/Seeded Errors Found)×Total Number of Seeded errors.

3) **Alternative Approach by Tom Gilb---Independent Reviews**

- Instead of seeding known errors, use independent reviewers to inspect or test the same code.
- Collect three counts:
  - n1: Number of valid errors found by reviewer A
  - n2: Number of valid errors found by reviewer B
  - n12: Number of cases where the same error is found by both A and B
- The smaller the proportion of errors found by both A and B compared to those found by only one reviewer, the larger the total number of errors likely to be in the software.

- Estimate the total number of errors using the formula:

$$n=(n1\times n2)/n12$$

This method helps in estimating the number of latent errors without deliberately introducing known errors into the software.

For example, A finds 30 errors and B finds 20 errors of which 15 are common to both A and B . Thes estimated total number of errors would be:

$$(30*20)/15=40$$

13.13 SOFTWARE RELIABILITY:

- The reliability of a software product denotes trustworthiness or dependability.
- It can be defined as the probability of its working correctly over a given period of time.

Software product having a large number of defects is unreliable. Reliability of the system will improve if the number of defects in it is reduced.

Reliability is a observer dependent, it depends on the relative frequency with which different users invoke the functionalities of a system. It is possible that because of different usage patterns of the available functionalities of software, a bug which frequently shows up for one user, may not show up at all for another user, or may show up very infrequently.

Reliability of the software keeps on improving with time during the testing and operational phases as defects are identified and repaired. The growth of reliability over the testing and operational phases can be modelled using a mathematical expression called ***Reliability Growth Model (RGM).***

Popular RGMs *(Explanation in Page 53)*

- Jelinski-Moranda model
- Littlewood-Verrall model
- Goel-Okumoto model

RGMs help predict reliability levels during the testing phase and determine when testing can be stopped.

Challenges in Software Reliability which makes which makes difficult to measure than hardware reliability.

- Dependence on the specific location of bugs
- Observer-dependent nature of reliability
- Continuous improvement as errors are detected and corrected.

Hardware vs. Software Reliability

- Hardware failures typically result from wear and tear, whereas software failures are due to bugs.

- Hardware failure often requires replacement or repair of physical components. Software failures need bug fixes in the code, which can affect reliability positively or negatively.

- **Hardware Reliability**: Concerned with stability and consistent inter-failure times.
  Software Reliability: Aims for growth, meaning an increase in inter-failure times as bugs are fixed.

- **Hardware**: Shows a "bathtub" curve where failure rate is initially high, decreases during the useful life, and increases again as components wear out. **Software**: Reliability generally improves over time as bugs are identified and fixed, leading to decreased failure rates

Figure 13.11(a): Illustrates the hardware product's failure rate over time, depicting the "bathtub" curve.
Figure 13.11(b): Shows the software product's failure rate, indicating a decline in failure rate over time due to bug fixes and improvements.

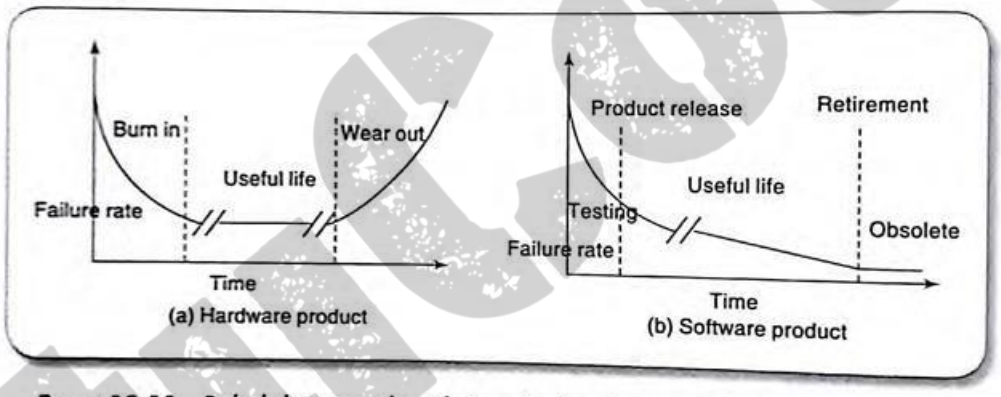


FIGURE 13.11 Reliability growth with time for hardware and software products

Software Reliability Metrics

The six metric that correlate with reliability as follows:

1) **Rate of Occurrence of Failure (ROCOF)**:

- Measures the frequency of occurrences of failures.
- Calculated as the ratio of total failures observed to the duration of observation.
- Limited applicability for non-continuously running software. Ex: Library software is idle until a book issue request is made.

2) **Mean Time to Failure (MTTF)**:

- Time between two successive failures, averaged over a large number of failures.

- Calculation involves summing up time intervals between failures and dividing by the number of failures. MTTF can be calculated as $\sum t_i/n$
- Only runtime is considered, excluding downtime for fixes.

3) **Mean Time to Repair (MTTR)**:

- Average time required to fix an error.
- Measures the time taken to track and fix failures

4) **Mean Time Between Failures (MTBF)**:

- Sum of MTTF and MTTR. –> MTBF=MTTF+MTTR.
- Indicates the expected time until the next failure after a failure is fixed.

5) **Probability of Failure on Demand (POFOD)**:

- POFOD measures likelihood of system failure when a service request is made. For example, POFOD of 0.001 means that 1 out of every 1000 service requests would result in a failure.
- Suitable for systems not required to run continuously.

6) **Availability**:

- Measures how likely the system is available for use during a given period.
- Takes into account both failure occurrences and repair times.

Types of Software Failures

- **Transient**: Failures occurring only for certain inputs while invoking a function.
- **Permanent**: Failures occurring for all inputs while invoking a function.
- **Recoverable**: System can recover without shutting down.
- **Unrecoverable**: System needs to be restarted to recover.
- **Cosmetic**: Minor irritations without incorrect result

Reliability Growth Modeling

- **Definition**: Mathematical models predicting how software reliability improves as errors are detected and fixed.
- **Application**: Used to predict reliability levels and determine when to stop testing.

1) Jelinski and Moranda Model

- **Model Characteristics**:
  - A step function model assuming reliability increases by a constant increment with each error fix.
  - Assumes perfect error fixing, meaning all errors contribute equally to reliability growth.
  - Typically, unrealistic as different errors contribute differently to growth. Reliabity growth predicted using this model has been shown in Figure 13.12.

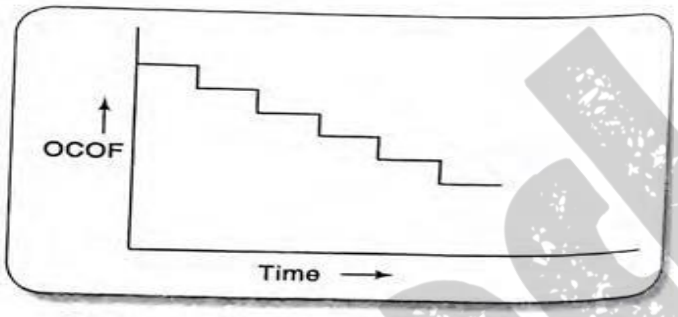


FIGURE 13.12 Jelinski–Moranda model

- **Failure Rate Equation**:
  Instantaneous failure rate, or hazard rate, is given by

  $Z(i)=K(N-i+1)Z(i) = K(N - i + 1)Z(i)=K(N−i+1)$,
  where K is a constant,
  N is the total number of errors, and
  't' is the interval between the i^{th} and (i+1) th failure.

2) Littlewood and Verall's Model

- This model allows for negative reliability growth, acknowledging that repairs can introduce new errors.
- It models the impact of error repairs on product reliability, which diminishes over time as larger contributions to reliability are addressed first.
- Uses Gamma distribution to treat an error's contribution to reliability improvement as an independent random variable.

3) Goel-Okutomo Model

- This model assumes exponentially distributed execution times between failures and models the cumulative number of failures over time. $(\mu(t))$,expected number of failures between time t and $t+\Delta t$.
- It is assumed that it follows a non-homogeneous Poisson process (NHPP) ie., expected number of occurrences for any time t to $t+\Delta t$ is proportional to the expected number of undetected errors at time t.

- The rate of failures decreases exponentially over time.
- The model assumes immediate and perfect correction once a failure is detected.

- The number of failures over time if given in Figure 13.13. The number of failures at time t can be given by $\mu(t)=N(1-e-bt)$,
  where N=Expected total number of defects in the code and
  b is the rate at which the failure rate decreases.

Graph: Illustrates the expected total number of errors over execution time, showing an initial rapid increase in detected errors, which slows down as more errors are corrected.

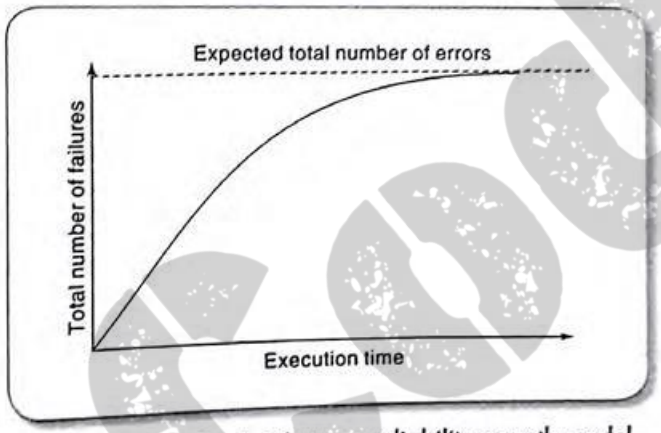


FIGURE 13.13 Goel–Okutomo reliability growth model

13.14 QUALITY PLANS

- Organizations produce quality plans for each project to show how standard quality procedures and standards from the organization's quality manual will be applied to the project.

- If quality-related activities and requirements have been identified by the main planning process, a separate quality plan may not be necessary.

- When producing software for an external client, the client's quality assurance staff might require a dedicated quality plan to ensure the quality of the delivered products.

Function of a Quality Plan:

- A quality plan acts as a checklist to confirm that all quality issues have been addressed during the planning process.

- Most of the content in a quality plan references other documents that detail specific quality procedures and standards.

Components of a Quality Plan

A quality plan might include:

- **Purpose and scope of the plan**
- **List of references to other documents**
- **Management arrangements**: 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**: Methods of risk management to be used