Module-05

Chapter:-1 - Software Quality

Introduction to System Quality

Concept of Quality:

- Generally agreed to be beneficial.
- Often vaguely defined in practice.
- Requires precise definition of required qualities.

Objective Assessment:

- Judging if a system meets quality requirements needs measurement.
- Critical for package selection, e.g., Brigette at Brightmouth College.

Development Perspective:

- 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.

Emphasis on Development Methods:

- 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.

The place of software quality in project planning

Quality Concerns:

- Present at all stages of project planning and execution.
- Key points in the Step Wise framework where quality is particularly emphasized:

A. Step 1: Identify Project Scope and Objectives

- Objectives may include qualities of the application to be delivered.

B. Step 2: Identify Project Infrastructure

- Activity 2.2 involves identifying installation standards and procedures, often related to quality.

C. Step 3: Analyze Project Characteristics

- Activity 3.2 involves analyzing other project characteristics, including quality-based ones.
- 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.

D. Step 4: Identify the Products and Activities of the Project

- Identify entry, exit, and process requirements for each activity.

E. Step 5: Review and Publicize Plan

- Review the overall quality aspects of the project plan at this stage.

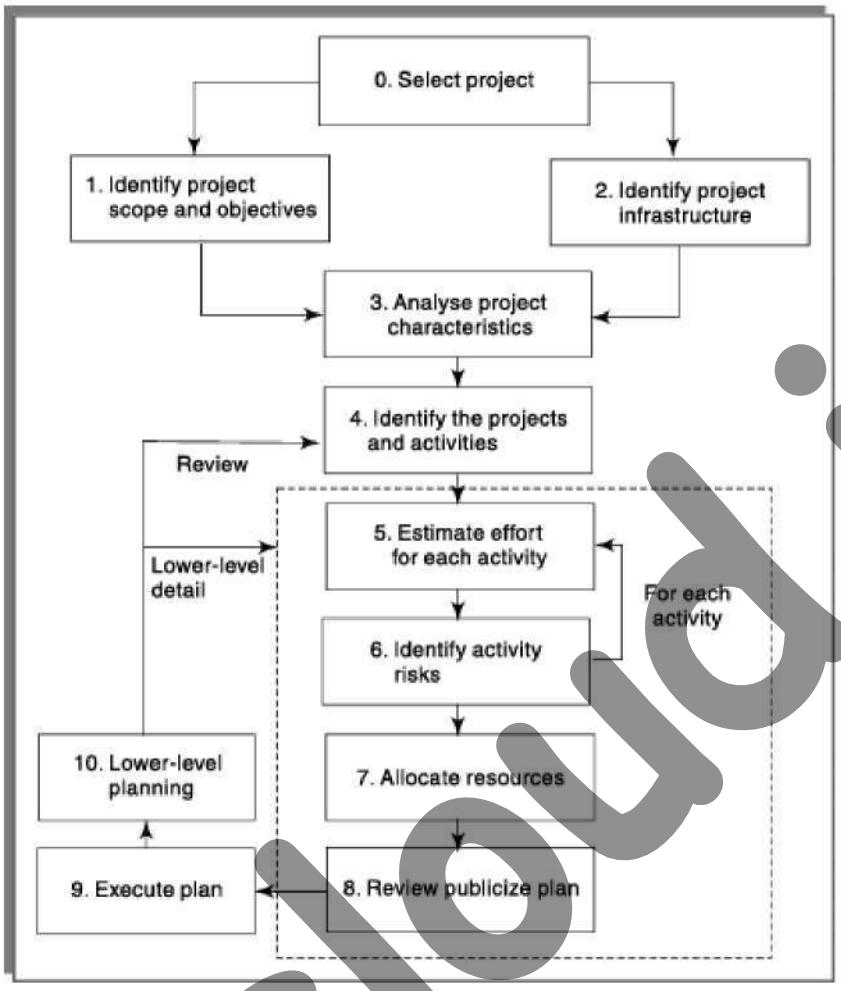


FIGURE 13.1 The place of software quality in Step Wise

Importance of software quality

Special Characteristics of Software Affecting Quality Management

General Expectation:

- Quality is a concern for all producers of goods and services.

Special Demands in Software:

1. **Increasing Criticality of Software**

   - Final customers and users are increasingly concerned about software quality, particularly reliability.

- Greater reliance on computer systems and use in safety-critical applications (e.g., aircraft control).

2. **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.
3. **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.

Defining Software Quality

System Requirements:

- **Functional Requirements:** Define what the system is to do.
- **Resource Requirements:** Specify allowable costs.
- **Quality Requirements:** State how well the system is to operate.

External and Internal Qualities:

- **External Qualities:** Reflect the user's view, such as usability.
- **Internal Factors:** Known to developers, such as well-structured code, which may enhance reliability.

Measuring Quality:

- **Necessity of Measurement:** To judge if a system meets quality requirements, its qualities must be measurable.

- **Good Measure:** Relates the number of units to the maximum possible (e.g., faults per thousand lines of code).

- **Clarification Through Measurement:** Helps to define and communicate what quality really means, effectively answering "how do we know when we have been successful?"

Direct vs. Indirect Measures:

- **Direct Measurement:** Measures the quality itself (e.g., faults per thousand lines of code).
- **Indirect Measurement:** Measures an indicator of the quality (e.g., number of user inquiries at a help desk as an indicator of usability).

Setting Targets:

- **Impact on Project Team:** Quality measurements set targets for team members.
- **Meaningful Improvement:** Ensure that improvements in measured quality are meaningful.
    - Example: Counting errors found in program inspections may not be meaningful if errors are allowed to pass to the inspection stage rather than being eradicated earlier.

Drafting a Quality Specification for Software Products

When there's concern about a specific quality characteristic in a software product, a quality specification should include the following details:

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

2. **Scale**
    - **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.

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 Software Quality Characteristics

1. **Functionality:**
   - **Definition:** The functions that a software product provides to satisfy user needs.
   - **Sub-characteristics:** Suitability, accuracy, interoperability, security, compliance.
2. **Reliability:**
   - **Definition:** The capability of the software to maintain its level of performance under stated conditions.
   - **Sub-characteristics:** Maturity, fault tolerance, recoverability.
3. **Usability:**
   - **Definition:** The effort needed to use the software. ○ **Sub-characteristics:** Understandability, learnability, operability, attractiveness.
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.
6. **Portability:**
   - **Definition:** The ability of the software to be transferred from one environment to another.
   - **Sub-characteristics:** Adaptability, install ability, co-existence.

Quality in Use (Additional Aspect)

- **Definition:** Focuses on how well the software supports specific user goals in a specific context of use.
- **Elements:** Effectiveness, productivity, safety, satisfaction.

ISO 14598

- **Purpose:** Describes procedures for assessing the degree to which a software product conforms to the ISO 9126 quality characteristics.
- **Flexibility:** Can be adapted to assess software against different quality characteristics if needed.

Characteristic	Sub-characteristics
Functionality	Suitability
	Accuracy
	Interoperability
	Functionality compliance
	Security

ISO 9126 Sub-characteristics: Compliance and Interoperability

Compliance

- **Definition:** Refers to the degree to which the software adheres to application-related standards or legal requirements, such as auditing standards.
- **Purpose:** Added as a sub-characteristic to all six primary ISO 9126 external characteristics.
- **Example:** Ensuring software meets specific industry regulations or international standards relevant to its functionality, reliability, usability, efficiency, maintainability, and portability.

Interoperability

- **Definition:** Refers to the ability of the software to interact with other systems effectively.
- **Clarification:** ISO 9126 uses "interoperability" instead of "compatibility" to avoid confusion with another characteristic called "replaceability".
- **Importance:** Ensures seamless integration and communication between different

software systems or components.

- **Sub-characteristics:** Includes aspects like interface compatibility, data exchange capabilities, and standards compliance to facilitate effective interaction.

Characteristic	Sub-characteristics
Reliability	Maturity
	Fault tolerance
	Recoverability
	Reliability compliance

ISO 9126 Sub-characteristics: Maturity and Recoverability

Maturity

- **Definition:** Refers to the frequency of failure due to faults in a software product over time.
- **Implication:** Higher maturity suggests that the software has been extensively tested and used, leading to the discovery and removal of more faults.
- **Measurement:** Typically assessed by the rate of failure incidents reported per unit of time or usage.

Recoverability

- **Definition:** Refers to the capability of the software to restore the system to its normal operation after a failure or disruption.
- **Clarification:** ISO 9126 distinguishes recoverability from security, which focuses on controlling access to the system.
- **Importance:** Ensures that the software can recover from failures or disruptions swiftly and effectively to minimize downtime and data loss.
- **Sub-characteristics:** Includes aspects like fault tolerance, resilience, and recovery time objectives (RTOs).

Importance of Distinction

- **Security:** Focuses on access control and protecting the system from unauthorized access, ensuring confidentiality, integrity, and availability.
- **Recoverability:** Focuses on system resilience and the ability to recover from failures, ensuring continuity of operations.

Characteristic	Sub-characteristics
Usability	Understandability
	Learnability
	Operability
	Attractiveness
	Usability compliance

ISO 9126 Sub-characteristics: Learnability and Attractiveness

Learnability

- **Definition:** Refers to the ease with which users can learn to operate the software and accomplish basic tasks.
- **Focus:** Primarily on the initial phase of user interaction with the software.
- **Measurement:** Assessed by the time it takes for new users to become proficient with the software, often measured in training hours or tasks completed.

Operability

- **Definition:** Refers to the ease with which users can operate and navigate the software efficiently.
- **Focus:** Covers the overall usability of the software during regular use and over extended periods.

- **Measurement:** Assessed by the user's experience with everyday tasks and efficiency in completing them.

Importance of Distinction

- **Learnability vs. Operability:** Learnability focuses on initial user experience and how quickly users can grasp the basics of using the software. Operability, on the other hand, emphasizes the efficiency and ease of use over extended periods and during regular use.
- **Attractiveness:** A recent addition under usability, attractiveness focuses on the aesthetic appeal and user interface design, particularly relevant in software products where user engagement and satisfaction are crucial, such as games and entertainment applications.

Application in Software Evaluation

- **Learnability:** Critical for software that requires quick adoption and minimal training, ensuring users can start using the software effectively from the outset.
- **Operability:** Crucial for software used intensively or for extended periods, focusing on efficiency, ease of navigation, and user comfort.

- **Attractiveness:** Important in consumer-facing software where user engagement, satisfaction, and retention depend on the software's visual appeal and user interface design.

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

ISO 9126 Sub-characteristics: Analysability, Changeability, and Stability

Analysability

- **Definition:** Refers to the ease with which the cause of a failure in the software can be determined.
- **Focus:** Helps in diagnosing and understanding software failures or issues quickly and accurately.
- **Importance:** Facilitates efficient debugging and troubleshooting during software maintenance and support phases.

Changeability

- **Definition:** Also known as flexibility, changeability refers to the ease with which software can be modified or adapted to changes in requirements or environment.
- **Focus:** Emphasizes the software's ability to accommodate changes without introducing errors or unexpected behaviors.

- **Importance:** Enables software to evolve and remain relevant over time by supporting modifications and updates effectively.

Stability

- **Definition:** Refers to the software's ability to undergo changes or modifications without introducing unexpected errors or side effects.
- **Focus:** Ensures that modifications or updates to the software do not destabilize its existing functionality or compromise its reliability.
- **Importance:** Provides confidence in the software's ability to maintain operational integrity despite ongoing changes or updates.

Clarification of Terms

- **Changeability vs. Flexibility:** Changeability focuses on the software's ability to be modified without causing issues, while flexibility emphasizes its capacity to adapt to diverse requirements or environments.
- **Stability:** Ensures that software modifications are managed effectively to minimize risks of unintended consequences or disruptions.

Application in Software Development

- **Analysability:** Essential for efficient debugging and problem-solving, ensuring quick resolution of software issues.
- **Changeability:** Critical for agile software development, supporting iterative changes and continuous improvement.
- **Stability:** Ensures reliability and consistency during software updates or modifications, maintaining overall system integrity.

Characteristic	Sub-characteristics
Portability	Adaptability
	Installability
	Coexistence
	Replaceability
	Portability compliance

ISO 9126 Quality Characteristics: Portability Compliance, Replaceability, and Coexistence

Portability Compliance

- **Definition:** Refers to the adherence of the software to standards that facilitate its transferability and usability across different platforms or environments.
- **Focus:** Ensures that the software can run efficiently and effectively on various hardware and software configurations without needing extensive modifications.
- **Importance:** Facilitates broader deployment and reduces dependency on specific hardware or software configurations.

Replaceability

- **Definition:** Refers to the ability of the software to replace older versions or components without causing disruptions or compatibility issues.
- **Focus:** Emphasizes upward compatibility, allowing new versions or components to seamlessly integrate with existing systems or environments.
- **Importance:** Supports evolutionary software development by enabling updates and enhancements without requiring extensive rework.

Coexistence

- **Definition:** Refers to the ability of the software to peacefully share resources and operate alongside other software components within the same environment.
- **Focus:** Does not necessarily involve direct data exchange but ensures compatibility and non-interference with other software components.
- **Importance:** Enables integration of the software into complex IT ecosystems without conflicts or performance degradation.

Clarification of Terms

- **Portability Compliance:** Ensures software can run across different platforms by adhering to common standards and programming languages.
- **Replaceability:** Supports compatibility and seamless integration of new software versions or components with existing systems.
- **Coexistence:** Ensures harmonious operation of the software alongside other components within the same environment.

Guidelines for Use of 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.

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.

External Quality Measurements:

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

Mapping Measurements to User Satisfaction:

- 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

Internal Measurements and Intermediate Products:

- Internal measurements like code execution times can help predict external qualities like response time during software design and development.
- Predicting external qualities from internal measurements is challenging and often requires validation in the specific environment where the software will operate.

Challenges with Mapping Internal and External Quality Characteristics:

- ISO 9126 acknowledges that correlating internal code metrics to external quality characteristics like reliability can be difficult.
- This challenge is addressed in a technical report rather than a full standard, indicating ongoing research and development in this area.

TABLE 13.1 Mapping measurements to user satisfaction

Response time (seconds)	Rating
<2	Exceeds expectation
2–5	Within the target range
6–10	Minimally acceptable
>10	Unacceptable

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.

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

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:

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 fa)	Prod ud A		Product B	
		Quality score (W	yVeijhtctl score (a x b)	Quality score (0)	Weighted score (a X c)
Usability	3	1	3	3	9
Efficiency	4	2	8	2	S
Maintainability	2	3	6	1	2
Overall			17		19

Certainly, when conducting a quality assessment on behalf of a user community or professional body, several considerations come into play:

1. **Objective Assessment:** The goal is to provide an objective evaluation of software tools that support the working practices of the community's members. This assessment aims to be independent of individual user environments and preferences.
2. **Variability in Results:** The outcome of such assessments can vary significantly based on the weightings assigned to each software characteristic. Different stakeholders within the community may have varying requirements and priorities.
3. **Caution in Interpretation:** It's crucial to exercise caution in interpreting and applying the results of these assessments. While they strive for objectivity, they are still influenced by the criteria set and the relative importance assigned to each quality characteristic.
4. **Community Needs:** Understanding the specific needs and expectations of the user community is essential. The assessment should align closely with the community's goals and the functionalities they require from the software tools being evaluated.
5. **Transparency and Feedback:** Providing transparency in the assessment process and gathering feedback from community members can enhance the credibility and relevance of the evaluation results. This helps ensure that the assessment adequately reflects the needs and perspectives of the community.

Product and process metrics

Understanding the differences between product metrics and process metrics is crucial in software development:

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.

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.

Quality Management systems

ISO 9001:2000, now superseded by newer versions but still relevant in principle, outlines standards for Quality Management Systems (QMS). Here's a detailed look at its key aspects and how it applies to software development:

ISO 9001:2000 Overview

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.

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.

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.

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.

SEI capability maturity model (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:

- **Capability Evaluation:** Used by contract awarding authorities (like the US DoD) to assess potential contractors' capabilities to predict performance if awarded a contract.
- **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:

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.

TABLE 13.4 CMMI key process areas

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

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.

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.

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:
  - Evaluate their current processes against a recognized standard. ○ Identify strengths and weaknesses in their processes.
  - 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.

TABLE 13.5 ISO **15504** framework for process capability

Level	Attribute	Conunenls
0. Incomplete		The process is not implemented or is unsuccessful
I. 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 de lined 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

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 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**

- **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).

- **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.
- **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.

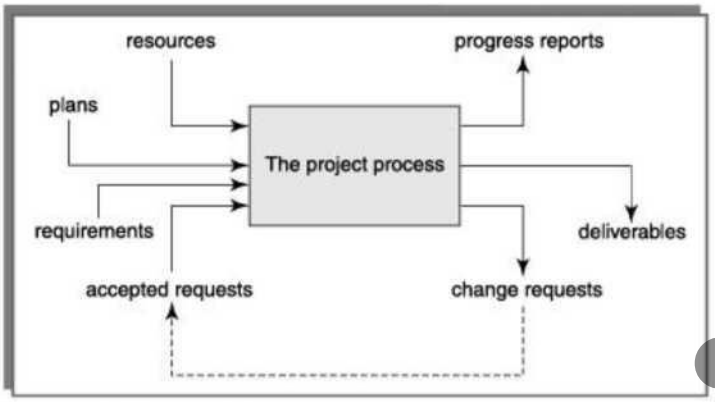


FIGURE 13.3 The project as a 'closed box'

The next step would be to define carefully the processes involved in each stage of the development life cycle – see Figure 13.4. 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.

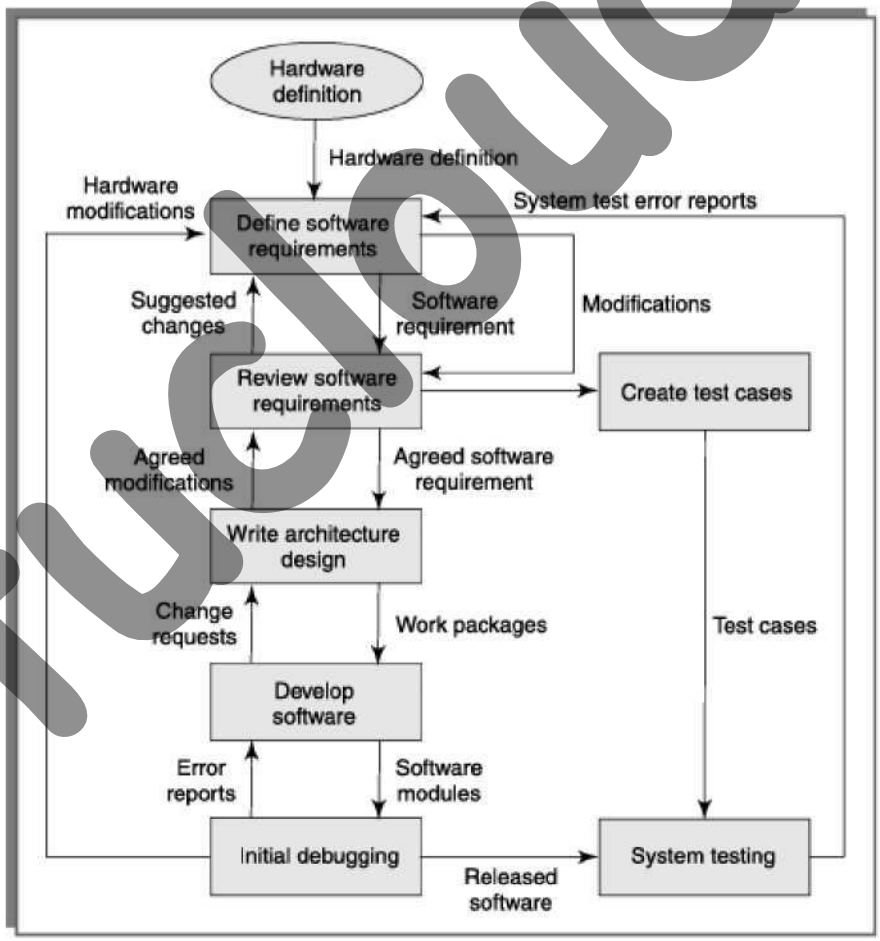


FIGURE 13.4 A process diagram

Six Sigma

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. **Analyse:**
   - **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.
  - 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.

Techniques to enhance software quality

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

1. **Increasing Visibility:**
   - Early practices like Gerald Weinberg's 'egoless programming' promoted code review among programmers, enhancing visibility into each other's work.
   - Modern practices extend this visibility to include walkthroughs, inspections, and formal reviews at various stages of development, ensuring early detection and correction of defects.
2. **Procedural 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.
3. **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.
4. **Inspections:**
   - Inspections are critical in ensuring quality at various development stages, not just in coding but also in documentation and test case creation.
   - Techniques like Fagan inspections, pioneered by IBM, formalize the review process with trained moderators leading discussions to identify defects and improve quality.

5. **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.
6. **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 the Fagan method

- Inspections are carried out on all major deliverables.
- All types of defect are noted - not just logic or function errors.
- Inspections can be carried out by colleagues at all levels except the very top.
- Inspections are carried out using a predefined set of steps.
- Inspection meetings do not last for more than two hours.
- The inspection is led by a moderator who has had specific training in the technique.
- The other participants have defined roles. For example, one person will act as a recorder and note all defects found, and another will act as reader and take the other participants through the document under inspection.
- Checklists 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

It seems like you're discussing formal methods in software development and the concept of software quality circles. Here's a summary of the points covered:

Formal Methods in Software Development

- **Definition**: Formal methods use unambiguous, mathematically based specification languages like Z and VDM. They define preconditions (input conditions) and postconditions (output conditions) for procedures.
- **Purpose**: These methods ensure precise and unambiguous specifications and allow for mathematical proof of algorithm correctness based on specified conditions.
- **Adoption**: Despite being taught widely in universities, formal methods are rarely used in mainstream software development. Object Constraint Language (OCL) is a newer, potentially more accessible development in this area.

Software Quality Circles (SWQC)

- **Purpose**: SWQCs are adapted from Japanese quality practices to improve software development processes by reducing errors.
- **Structure**: Consist of 4 to 10 volunteers in the same department who meet regularly to identify, analyze, and solve work-related problems.
- **Process**: The group selects a problem, identifies its causes, and proposes solutions. Management approval may be required for implementing improvements.
- **Benefits**: Enhances team collaboration, spreads best practices, and focuses on continuous process improvement.

If you have any specific questions or if there's more you'd like to explore on these topics or related areas, feel free to ask!

The process you're describing involves the compilation of most probable error lists, which is a proactive approach to improving software development processes. Here's a

breakdown of the steps involved:

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.

If you have more questions or need further elaboration on any aspect, feel free to ask!

Lessons Learned Report

The concept of Lessons Learned reports and Post Implementation Reviews (PIRs) are crucial for organizational learning and continuous improvement in project management. Here's a breakdown of these two types of reports:

- **Purpose**: The Lessons Learned report is prepared by the project manager immediately after the completion of the project. Its purpose is to capture insights and experiences gained during the project execution.
- **Content**: Typically, a Lessons Learned report includes:
  - **Successes and Failures**: What worked well and what didn't during the project.
  - **Challenges Faced**: Difficulties encountered and how they were overcome. ○ **Best Practices**: Practices or approaches that proved effective. ○ **Lessons Learned**: Key takeaways and recommendations for future projects.

- **Audience**: The report is usually intended for internal stakeholders involved in similar future projects, such as project managers, team members, and organizational leadership.
- **Follow-up**: One common issue is the lack of follow-up on the recommendations provided in these reports. This can occur due to a lack of dedicated resources or organizational structure for implementing these lessons.

Post Implementation Review (PIR)

- **Purpose**: A PIR takes place after a significant period of operation of the new system (typically after it has been in use for some time). Its focus is on evaluating the effectiveness of the implemented system rather than the project process itself.
- **Timing**: Conducted by someone who was not directly involved in the project to ensure neutrality and objectivity.
- **Content**: A PIR includes:
    - **System Performance**: How well the system meets its intended objectives and user needs.
    - **User Feedback**: Feedback from users on system usability and functionality.
    - **Improvement Recommendations**: Changes or enhancements suggested to improve system effectiveness.
- **Audience**: The audience typically includes stakeholders who will benefit from insights into the system's operational performance and areas for improvement.
- **Outcome**: Recommendations from a PIR often lead to changes aimed at enhancing the effectiveness and efficiency of the system.

Importance of Both Reports

- **Learning and Improvement**: Both reports contribute to organizational learning by capturing valuable insights and experiences.
- **Continuous Improvement**: They provide a basis for making informed decisions and improvements in future projects and system implementations.

Testing

The text discusses the planning and management of testing in software development, highlighting the challenges of estimating the amount of testing required due to unknowns, such as the number of bugs left in the code.

It introduces the V-process model as an extension of the waterfall model, emphasizing the importance of validation activities at each development stage.

1. **Quality Judgement**:
   - The final judgement of software quality is based on its correct execution.
2. **Testing Challenges**:
   - Estimating the remaining testing work is difficult due to unknown bugs in the code.
3. **V-Process Model**:
   - Introduced as an extension of the waterfall model.
   - Diagrammatic representation provided in Figure 13.5.
   - Stresses the necessity for validation activities matching the project creation activities.
4. **Validation Activities**:
   - Each development step has a matching validation process.
   - Defects found can cause a loop back to the corresponding development stage for rework.
5. **Discrepancy Handling**:
   - Feedback should occur only when there is a discrepancy between specified requirements and implementation.
   - Example: System designer specifies a calculation method; if a developer misinterprets it, the discrepancy is caught during system testing.
6. **System Testing**:
   - Original designers are responsible for checking that software meets the specified requirements, discovering any misunderstandings by developers.

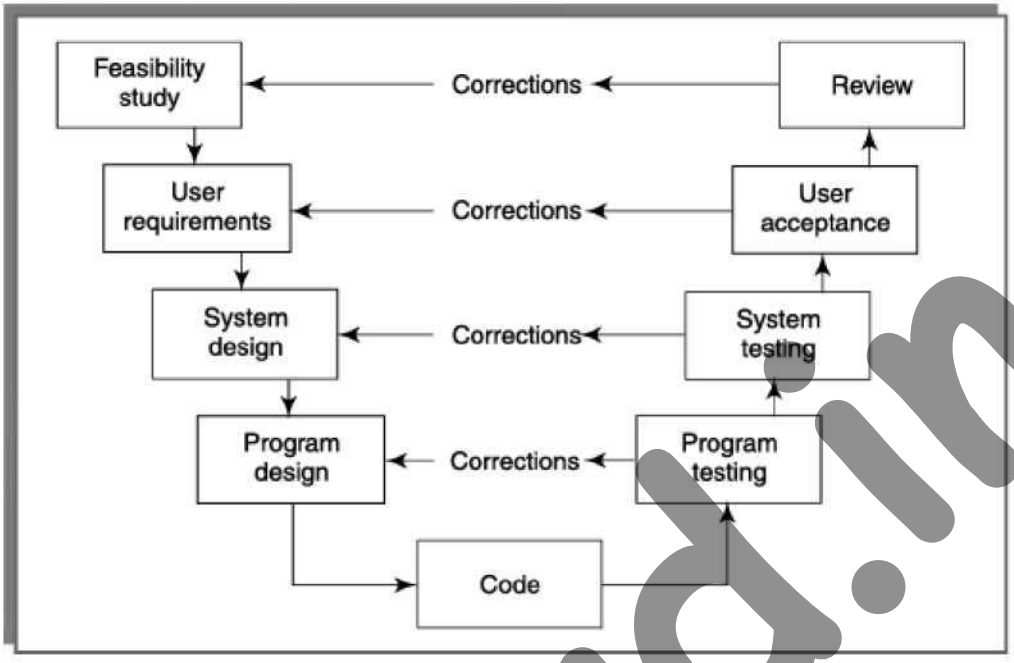


FIGURE 13.5 The V-process model

Framework for Planning:

- The V-process model provides a structure for making early planning decisions about testing.

Types and Amounts of Testing:

- Decisions can be made about the types and amounts of testing required from the beginning of the project.

Off-the-Shelf Software:

- If software is acquired off-the-shelf, certain stages like program design and coding are not relevant.
- Consequently, program testing would not be necessary in this scenario.

User Requirements and Acceptance Testing:

- User requirements must still be produced.

- User acceptance tests remain valid and necessary regardless of the software being off-the-shelf.

Verification versus validation

1. **Objectives**:
   - Both techniques aim to remove errors from software.
2. **Definitions**:
   - **Verification**: Ensures outputs of one development phase conform to the previous phase's outputs.
   - **Validation**: Ensures fully developed software meets its requirements specification.
3. **Objectives Clarified**:
   - **Verification Objective**: Check if artifacts produced after a phase conform to those from the previous phase (e.g., design documents conform to requirements specifications).
   - **Validation Objective**: Check if the fully developed and integrated software satisfies customer requirements.
4. **Techniques**:
   - **Verification Techniques**: Review, simulation, and formal verification. ○ **Validation Techniques**: Primarily based on product testing.
5. **Process Stages**:
   - **Verification**: Conducted during the development process to ensure development activities are correct.
   - **Validation**: Conducted at the end of the development process to ensure the final product meets customer requirements.
6. **Phase Containment of Errors**:
   - Verification aims for phase containment of errors, which is a cost-effective way to eliminate bugs and an important software engineering principle.

7. **V-Process Model Activities**:
    - All activities on the right side of the V-process model are verification activities except for the system testing block, which is a validation activity.

Testing activities

The text provides an overview of test case design approaches, levels of testing, and main testing activities in software development.

It emphasizes the differences between black-box and white-box testing, the stages of testing (unit, integration, system), and the activities involved in the testing process. **Test Case Design Approaches**

1. **Black-Box Testing**:
    - Test cases are designed using only the functional specification. ○ Based on input/output behavior without knowledge of internal structure. ○ Also known as functional testing or requirements-driven testing.
2. **White-Box Testing**:
    - Test cases are designed based on the analysis of the source code. ○ Requires knowledge of the internal structure.
    - Also known as structural testing or structure-driven testing.

Levels of Testing

1. **Unit Testing**:
    - Tests individual components or units of a program.
    - Conducted as soon as the coding for each module is complete. ○ Allows for parallel activities since modules are tested separately. ○ Referred to as testing in the small.
2. **Integration Testing**:
    - Checks for errors in interfacing between modules.

- Units are integrated step by step and tested after each integration. - Referred to as testing in the large.

3. **System Testing**:
   - Tests the fully integrated system to ensure it meets requirements. - Conducted after integration testing.

Testing Activities

1. **Test Planning**:
   - Involves determining relevant test strategies and planning for any required test bed.
   - Test bed setup is crucial, especially for embedded applications.
2. **Test Suite Design**:
   - Planned testing strategies are used to design the set of test cases (test suite).
3. **Test Case Execution and Result Checking**:
   - Each test case is executed, and results are compared with expected outcomes.
   - Failures are noted for test reporting when there is a mismatch between actual and expected results.

The text describes the detailed process and activities involved in software test reporting, debugging, error correction, defect retesting, regression testing, and test closure.

It highlights the importance of formal issue recording, the adjudication of issues, and various testing strategies to ensure software quality.

Test Reporting

1. **Issue Raising**:
   - Report discrepancies between expected and actual results.
2. **Issue Recording**:
   - Formal recording of issues and their history.

3. **Review Body Decisions**:
    - **Dismissal**: Misunderstanding of requirement by tester.
    - **Fault Identification**: Developers need to correct the issue. ○ **Incorrect Requirement**: Adding a new requirement and additional work/payment.
    - **Off-Specification Fault**: The application can operate with the error.
4. **Test Failure Notification**:
    - Informal intimation to the development team to optimize turnaround time.

Debugging and Error Correction

1. **Debugging**:
    - Identify error statements by analyzing failure symptoms.
    - Various debugging strategies are employed.
2. **Error Correction**:
    - Correct the code after locating the error through debugging.
3. **Defect Retesting**:
    - Retesting corrected code to check if the defect has been successfully addressed (resolution testing).
4. **Regression Testing**:
    - Ensures unmodified functionalities still work correctly after bug fixes. ○ Runs alongside resolution testing to check for new errors introduced by changes.

Test Closure

1. **Test Completion**:
    - Archiving documents related to lessons learned, test results, and logs for

future reference.

2. **Time-Consuming Activity**:
   - Debugging is noted as usually the most time-consuming activity in the testing process.

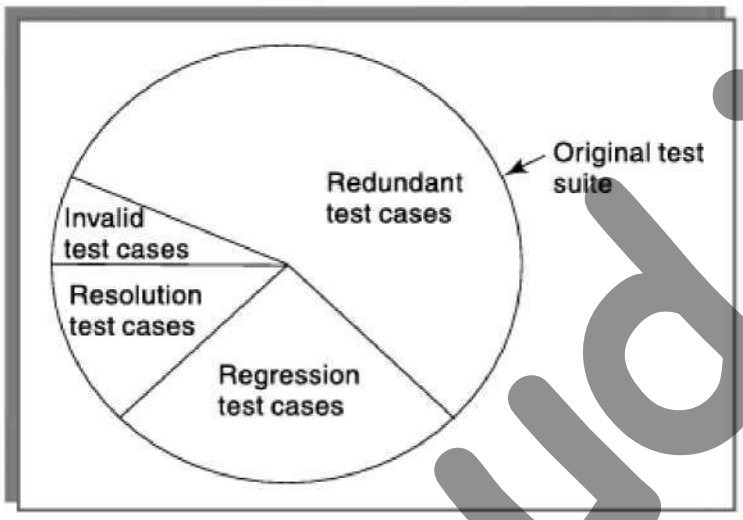


FIGURE 13.6 Types of test cases in the original test suite after a change

Who performs testing?

The text describes who performs testing in organizations, the importance and benefits of test automation, and various types of automated testing tools.

It emphasizes that while test automation can significantly reduce human effort, improve thoroughness, and lower costs, different tools have distinct advantages and challenges.

Who Performs Testing

1. **Independent Testing Groups**:
   - Provide an independent assessment of software correctness before release.
   - Can offer a final quality check but may lead developers to take less care in their work.
2. **Integrated Testing Roles**:
   - Testers work alongside developers rather than as a separate group.

Test Automation

1. **Importance of Test Automation**:
   - Testing is the most time-consuming and laborious software development activity.
   - Automation reduces human effort, time, and improves testing thoroughness.
   - Enables sophisticated test case design techniques.
2. **Benefits of Test Automation**:
   - More testing with a large number of test cases in a short period without significant cost overhead.
   - Automated test results are more reliable and eliminate human errors. ○ Simplifies regression testing by running old test cases repeatedly.

   - Reduces monotony, boredom, and errors in running the same test cases repeatedly.
   - Substantial cost and time reduction in testing and maintenance phases.

Types of Automated Testing Tools

1. **Capture and Playback Tools**:
   - Record inputs and outputs during manual execution to replay tests automatically later.
   - Useful for regression testing but can be costly to maintain when the unit under test changes.
2. **Automated Test Script Tools**:
   - Use scripts to drive automated test tools, providing inputs and recording outputs.
   - Debugging and verifying test scripts require significant effort, and changes to the unit under test necessitate updating scripts.

3. **Random Input Test Tools**:
    - Generate random test values to cover the input space, aiming to crash the unit under test.
    - Easy to implement but limited in scope as it only finds defects that crash the system.
4. **Model-Based Test Tools**:
    - Use simplified representations (models) of the program, like state or activity models, to generate tests.
    - Adequately cover the state space described by the model.

Estimating Errors Based on Code Size

1. **Historical Data**:
    - Use historical data to estimate errors per 1000 lines of code from past projects.
    - Apply this ratio to new system development to estimate potential errors based on the code size.

Error Seeding Technique

1. **Seeding Known Errors**:
    - Known errors are deliberately left in the code during desk checks. ○ Example: If 10 errors are seeded and after testing, 30 errors are found, including 6 seeded errors, 60% of seeded errors are detected. ○ This suggests that around 40% of errors are still to be detected. ○ Formula to estimate total errors:

Independent Reviews

1. **Tom Gilb's Approach**:
    - Two independent reviewers or groups are asked to inspect or test the same

code.

- They must work independently of each other.
- Three counts are collected for better error estimation.

Using these methods helps in obtaining a better estimation of latent errors, providing a clearer understanding of the remaining testing effort needed to ensure software quality.

- *n* 1. the number of valid errors found by A
- zi2, the number of valid errors found by B
- nl2, the 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. An estimate of the total number of errors (n) can be calculated by the formula:

zi = (nl X n2)/z/12

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

(30 X 20)/15 = 40

Software reliability

1. **Definition and Importance**:
   - Software reliability denotes the trustworthiness or dependability of a software product.
   - It is defined as the probability of the software working correctly over a given period of time.
   - Reliability is a crucial quality attribute for software products.
2. **Defects and Reliability**:
   - A large number of defects typically indicate unreliability.
   - Reducing defects generally improves reliability.
   - It is challenging to create a mathematical formula to relate reliability directly to the number of latent defects.
3. **Execution Frequency and Defect Impact**:
   - Errors in infrequently executed parts of the software have less impact on

overall reliability.

- Studies show that 90% of a typical program's execution time is spent on 10% of its instructions.
- The specific location of a defect (core or non-core part) affects reliability.

4. **Observer Dependency**:
   - Reliability is dependent on user behavior and usage patterns.
   - A bug may affect different users differently based on how they use the software.
5. **Reliability Improvement Over Time**:
   - Reliability usually improves during testing and operational phases as defects are identified and fixed.
   - This improvement can be modeled mathematically using Reliability Growth Models (RGM).
6. **Reliability Growth Models (RGM)**:
   - RGMs describe how reliability improves as failures are reported and bugs are corrected.
   - Various RGMs exist, including the Jelinski-Moranda model, Littlewood-Verall's model, and Goel-Okutomo's model.
   - RGMs help predict when a certain reliability level will be achieved, guiding decisions on when testing can be stopped.

Quality plans

Purpose of Quality Plans:

- Quality plans detail how standard quality procedures and standards from an organization's quality manual will be applied to a specific project.
- They ensure all quality-related activities and requirements are addressed.

Integration with Project Planning:

- If a thorough project planning approach, like Step Wise, is used, quality-related activities and requirements are often already integrated into the main planning process.
- In such cases, a separate quality plan might not be necessary.

Client Requirements:

- For software developed for external clients, the client's quality assurance staff may require a quality plan to ensure the quality of the delivered products.
- This requirement ensures that the client's quality standards are met.

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.

A quality plan might have entries for:

- purpose - scope of 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 - the methods of risk management that are to be used.

MODULE- 5

The Place of Software Quality in Project Planning

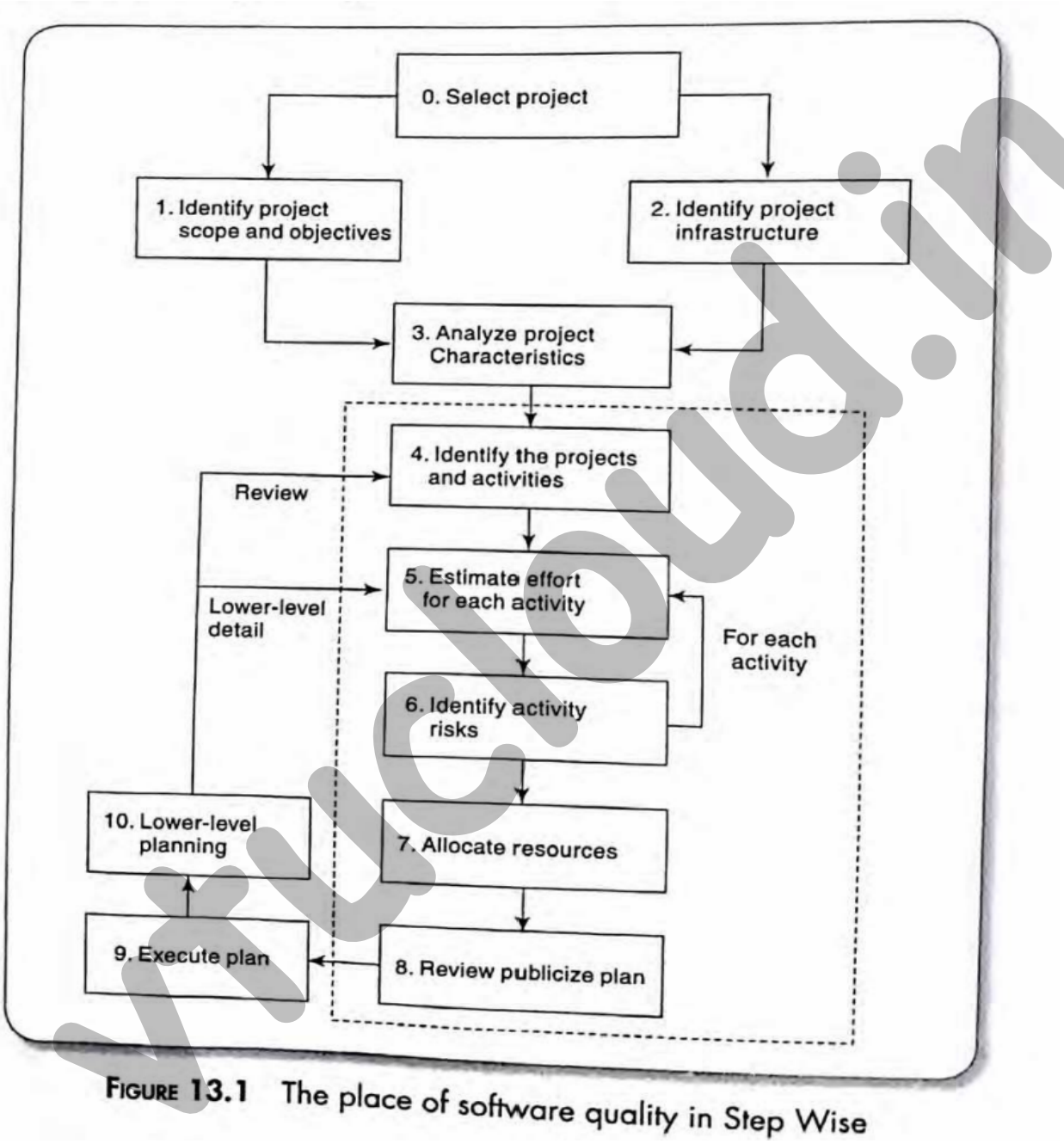


FIGURE 13.1 The place of software quality in Step Wise

Step 1:Project Scope and objective:Some objective could relate to the qualities of the application to be delivered.

Step 2: Identify project infrastructure: Identify the installation standard and procedures mostly regarding quality

Step 3: Analyze project characteristics: The application to be implemented is analyzed is examined to see if it has any special quality requirements.

For example is it is safety critical then a wide range of activities could be added, such as n-version development where a number of teams develop versions of the same software which are then run in parallel with the outputs being cross checked for discrepancies.

Step 4: Identify the products and activities of the project: It is at this point the entry, exist and process requirement are identified for each activity. Break down the project into manageable activities, ensuring each is planned with quality measures in place.

Step 5:Estimate Effort for Each Activity: Accurate effort estimation is essential to allocate sufficient resources for quality assurance activities, avoiding rushed and low-quality outputs.

Step 6: Identify Activity Risks: Identifying risks early allows for planning mitigation strategies to maintain quality throughout the project.

Step 7: Allocate Resources: Allocate resources not just for development but also for quality assurance tasks like testing, code reviews, and quality audits.

Step 8: Review and Publicize Plan: Regular reviews of the plan ensure that quality objectives are being met and any deviations are corrected promptly.

Step 9: Execute Plan: Execute the project plan with a focus on adhering to quality standards, monitoring progress, and making necessary adjustments to maintain quality.

Step 10: Lower-Level Planning:Detailed planning at lower levels should include specific quality assurance activities tailored to each phase or component of the project.

Review (Feedback Loop): Continuous review and feedback loops help in maintaining and improving quality throughout the project lifecycle.

Importance of Software Quality

We would expect quality to be concern of all procedures of goods and services.

- **Increasing criticality of software:** The final customer or user of a software is generally anxious about the 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.

- **The intangibility of software:** This make it difficult to know that a particular tasks in project has been completed satisfactory. The results of these tasks can be made tangible by demanding that the developer produce deliverables that can be examined for quality.
- **Accumulating errors during software development:** As computer system development comprises steps where the output from one step is the input to the next, the errors in the later deliverables will be added to those in the earlier steps, leading to an accumulating detrimental effect. In general, the later in a project that an error is found the more expensive it will be to fix. In addition, because the number of errors in the system is unknown, the debugging phases of a project arc particularly difficult to control.

Defining the software Quality:

Quality is a rather vague term and we need to define carefully what we mean by it.

- A functional specification describing what the system is to do
- A quality specification concerned with how well the functions are to operate

- Attempt to identify specific product qualities that are appropriate to software,

- For instance, group software qualities into three sets. **Product operation qualities, Products revision qualities and product transition qualities** A resource specification concerned with how much in to be spent on the system
- When there is concern about the need for a specific quality characteristic in a software product then a specification with the following minimum details should be drafted:

Definition/description: definition of the quality characteristic
Scale: the unit of measurement
Test: the practical test of the extent to which the attribute quality exists
Minimally acceptable: the worst value which might be acceptable if other characteristics compensated for it, and below which the product would have to be rejected out of hand
Target range: the range of values within which it is planned the quality measurement value should lie
Now: the value that applies currently

PRODUCT OPERATION QUALITIES
Correctness: The extent to which a program satisfy 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 required 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

Software Quality Models

The quality models give a characterization (often hierarchical) of software quality in terms of a set of characteristics of the software. The bottom level of the hierarchy can be directly measured, enabling a quantitative assessment of the quality of the software. There are several well-established quality Models including McCall's. Dromey's and Boehm's. Since there was no standardization among the large number of quality models that became available, the ISO 9126 model of quality was developed.

GARVIN'S QUALITY DIMENSIONS

David Garvin suggests that quality ought to be thought about by taking a third-dimensional read point that begins with an assessment of correspondence and terminates with a transcendental (aesthetic) view. Eight dimensions of product quality management will be used at a strategic level to investigate quality characteristics.

Performance: How well it performs the jobs.

Features: How well it supports the required features.

Reliability: Probability of a product working satisfactorily within a specific period of time

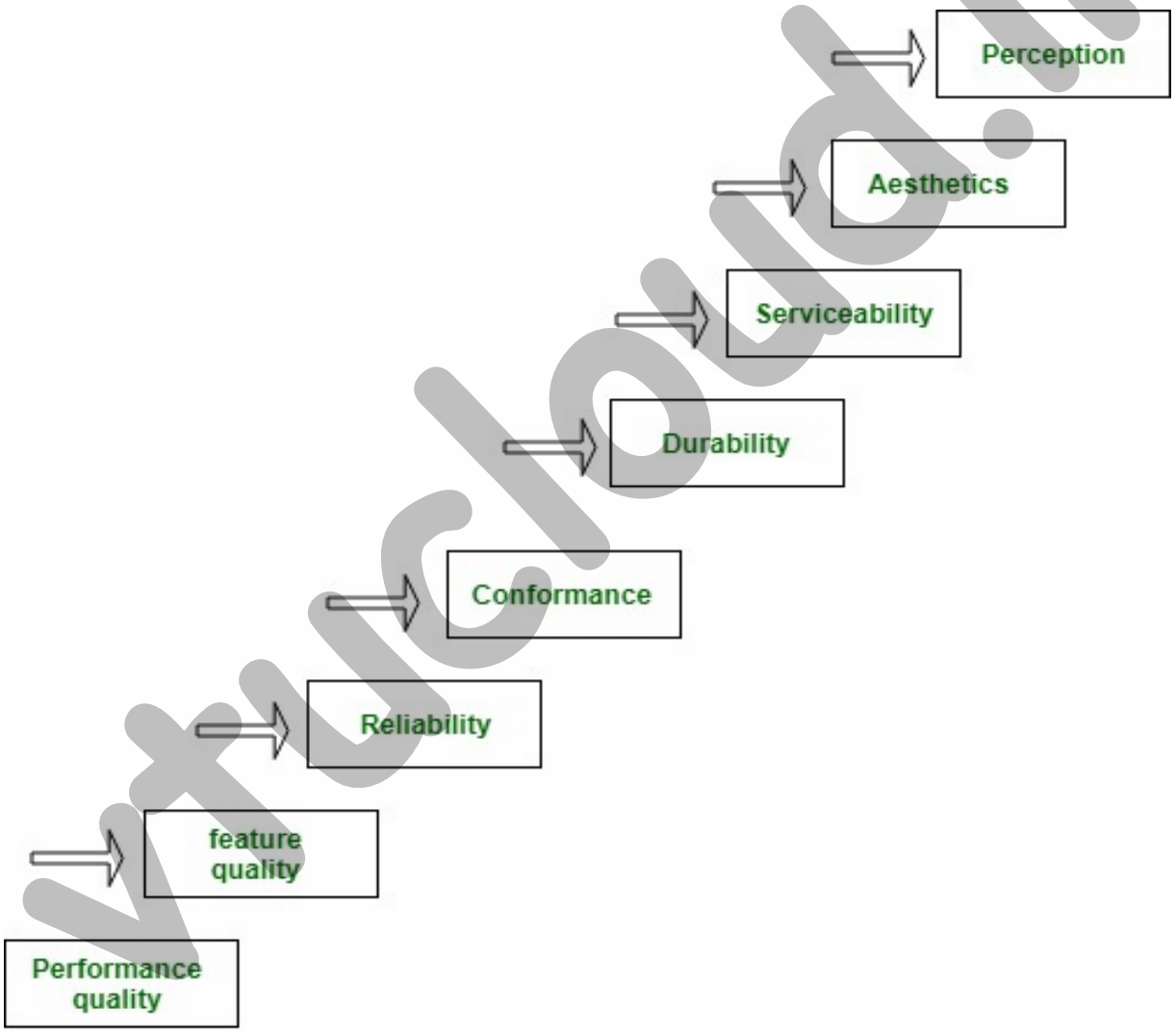
Conformance: Degree to which the product meets the requirements.

Durability: Measure of the product life

Serviceability: Speed and effectiveness maintenance.

Aesthetics: The look and feel of ihe product.

Perceived quality: User's opinion about the product quality.



Garvin's Dimentions of Quality

MCCALL MODEL

McCall's Software Quality Model was introduced in 1977. This model is incorporated with many attributes, termed software factors, which influence software. The model distinguishes between two levels of quality attributes:

- Quality Factors
- Quality Criteria

Quality Factors: The higher-level quality attributes that can be accessed directly are called quality factors. These attributes are external. The attributes at this level are given more importance by the users and managers.

Quality Criteria: The lower or second-level quality attributes that can be accessed either subjectively or objectively are called Quality Criteria. These attributes are internal. Each quality factor has many second-level quality attributes or quality criteria.

Example: The usability quality factor is divided into operability, training, communicativeness, input/output volume, and input/output rate. This model classifies all software requirements into 11 software quality factors.

The 11 factors are organized into three product quality factors: Product Operation, Product Revision, and Product Transition.

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.

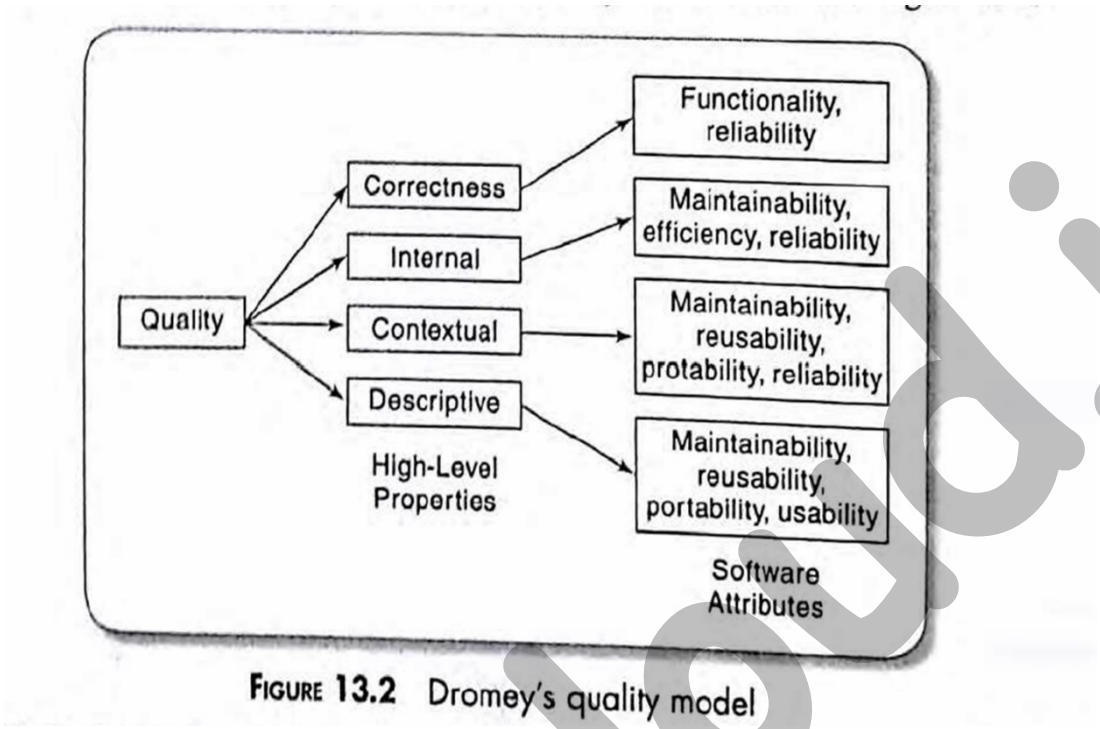


FIGURE 13.2 Dromey's quality model

BOEHM'S MODEL:

The model represents a hierarchical quality model similar to the McCall Quality Model to define software quality using a predefined set of attributes and metrics, each of which contributes to the overall quality of software. The difference between Boehm's and McCall's Models is that McCall's Quality Model primarily focuses on precise measurement of high-level characteristics, whereas Boehm's Quality Model is based on a wider range of characteristics.

Primary Uses of Boehm's Model

The highest level of Boehm's model has the following three primary uses, as stated as below:

As is the utility: The extent to which, we can use software as-is.

Maintainability: Effort required to detect and fix an error during maintenance.

Portability: Effort required to change the software to fit in a new environment.

Quality Factors Associated with Boehm's Model

The next level of Boehm's hierarchical model consists of seven quality factors associated with three primary uses, stated below:

Portability: Effort required to change the software to fit in a new environment.

Reliability: The extent to which software performs according to requirements.

Efficiency: Amount of hardware resources and code required to execute a function.

Usability (Human Engineering): Extent of effort required to learn, operate and understand functions of the software.

Testability: Effort required to verify that software performs its intended functions.

Understandability: Effort required for a user to recognize a logical concept and its applicability.

Modifiability: Effort required to modify software during the maintenance phase.