

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

MODULE 5

SOFTWARE QUALITY and SOFTWARE PROJECT ESTIMATION

INTRODUCTION

Quality:

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

Objective assessment:

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

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

Waiting until the system is complete to measure quality is too

late. During development, it's important to:

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

quality. Potential customers, like Amanda at IOE, might focus

on:

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

THE PLACE OF SOFTWARE QUALITY IN PROJECT PLANNING

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

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

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

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

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

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

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

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

THE IMPORTANCE OF SOFTWARE QUALITY

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

Increasingly criticality of software:

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

Earlier detection of errors during development-

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

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

The intangibility of software:

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

Accumulating errors during software development:

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

DEFINING SOFTWARE QUALITY

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

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

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

Product operation qualities:

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

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

Efficiency: The amounts of computer resource required by software.

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

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

Product Revision Qualities:

Maintainability: The effort required to locate and fix an error in an operational

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

program **Testability**: The effort required to test a program to ensure it performs its

intended function. **Flexibility**: The effort required to modify an operational program.

Product Transition Qualities:

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

Reusability: The extent to which a program can be used in other applications.

Interoperability: The efforts required to couple one system to another.

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

1. Definition/Description

Definition: Clear definition of the quality characteristic.

Description: Detailed description of what the quality characteristic entails.

2. Scale o Unit of Measurement:

The unit used to measure the quality characteristic (e.g., faults per thousand lines of code).

3. Test

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

4. Minimally Acceptable

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

5. Target Range

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

6. Current Value

Now: The value that applies currently to the quality characteristic.

Measurements Applicable to Quality Characteristics in Software

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

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:

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

- **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.

SOFTWARE QUALITY MODELS

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

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

There are several well-established quality models.

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

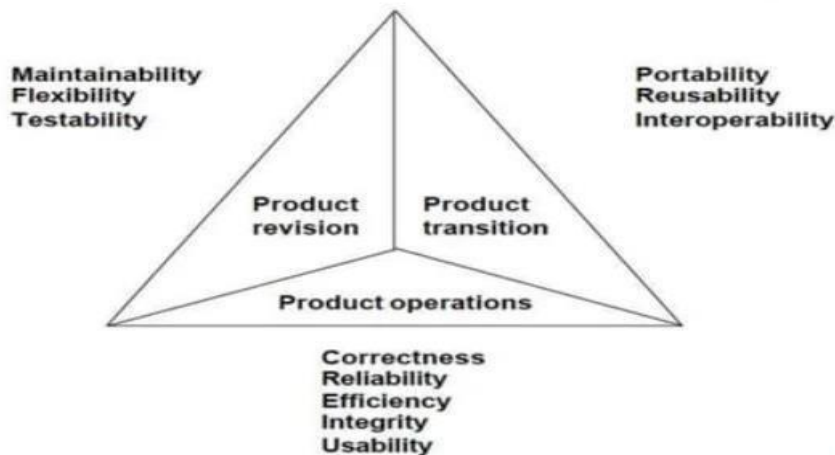
Garvin's Quality Dimensions: David Garvin, a professor of Harvard Business school, defined the quality of any product in terms of eight general attributes of the product.

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

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

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

McCall's Quality Model Triangle



on the following eleven attributes of the software:

Correctness: The extent to which a software product satisfies its specifications.

Reliability: The probability of the software product working satisfactorily over a given duration.

Efficiency: The amount of computing resources required to perform the required functions.

Integrity: The extent to which the data of the software product remain valid.

Usability: The effort required to operate the software product.

Maintainability: The ease with which it is possible to locate and fix bugs in the software product.

Flexibility: The effort required to adapt the software product to changing requirements.

Testability: The effort required to test a software product to ensure that it performs its intended function.

Portability: The effort required to transfer the software product from one hardware or software system environment to another.

Reusability: The extent to which a software can be reused in other applications.

Interoperability: The effort required to integrate the software with other software.

- 2) **Dromey's model:** Dromey proposed that software product quality depends on four major high-level properties of the software: Correctness, internal characteristics, contextual

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

characteristics and certain descriptive properties. Each of these high-level properties of a software product, in turn depends on several lower-level quality attributes. Dromey's hierarchical quality model is shown in Fig 13.2

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

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

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

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

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

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

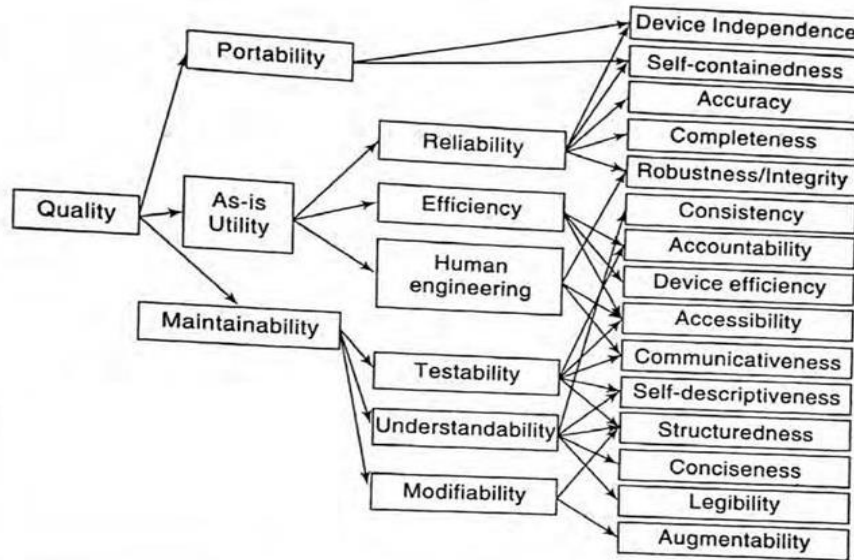


FIGURE 13.3 Boehm's quality model

ISO 9126

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

Motivation might be-

- **Acquires** who are obtaining software from external suppliers
 - **Developers** who are building a software product
 - **Independent evaluators** who are accessing the quality of a software product, not for themselves but for a community of user.
- ISO 9126 also introduces another type of elements – quality in use- for which following element has been identified
 - **Effectiveness** : the ability to achieve user goals with accuracy and completeness;
 - **Productivity** : avoiding the excessive use of resources, such as staff effort, in achieving user goals;

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

- **Safety:** within reasonable levels of risk of harm to people and other entities such as business, software, property and the environment
- **Satisfaction:** smiling users

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

ISO 9126 identifies six major external software quality characteristics:

1. Functionality:

Definition: The functions that a software product provides to satisfy user needs.

Sub-characteristics: Suitability, accuracy, interoperability, security, compliance.

Characteristic	Sub-characteristics
Functionality	Suitability
	Accuracy
	Interoperability
	Functionality compliance
	Security

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

2. Reliability:

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

Sub-characteristics: Maturity, fault tolerance, recoverability.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

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

3. Usability:

Definition: The effort needed to use the software.

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

Characteristic	Sub-characteristics
Usability	Understandability
	Learnability
	Operability
	Attractiveness
	Usability compliance

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

4. Efficiency:

Definition: The ability to use resources in relation to the amount of work done. **Sub-characteristics:** Time behavior, resource utilization.

5. Maintainability:

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

Sub-characteristics: Analyzability, modifiability, testability.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

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

6. Portability:

Definition: The ability of the software to be transferred from one environment to another. \ **Sub-characteristics: Adaptability, install ability, co-existence.**

Characteristic	Sub-characteristics
Portability	Adaptability
	Installability
	Coexistence
	Replaceability
	Portability compliance

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

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

ISO 9126 provides structured guidelines for assessing and managing software quality characteristics based on the specific needs and requirements of the software product. It

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

emphasizes the variation in importance of these characteristics depending on the type and context of the software product being developed.

Steps Suggested After Establishing Requirements

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

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

Importance of Quality Characteristics:

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

Efficiency: Important for real-time systems where timely responses are crucial. Measures

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

such as response time are key indicators.

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

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

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

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

For response time, user satisfaction could be mapped as follows (hypothetical example): Excellent: Response time < 1 second

Good: Response time 1-3 seconds

Acceptable: Response time 3-5 seconds

Poor: Response time > 5 seconds

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

TABLE 13.1 Mapping measurements to user satisfaction

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

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

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

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

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

Based on the ISO 9126 framework and your points:

1. Measurement Indicators Across Development Stages:

- **Early Stages:** Qualitative indicators like checklists and expert judgments are used to assess compliance with predefined criteria. These are subjective and based on qualitative assessments.
- **Later Stages:** Objective and quantitative measurements become more prevalent as the software product nears completion. These measurements provide concrete data about software performance and quality attributes.

2. Overall Assessment of Product Quality:

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

3. Purpose of Quality Assessment:

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

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

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

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

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

TABLE 13.2 Mapping response times onto user satisfaction

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

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

1. Rating for User Satisfaction:

- Products are evaluated based on mandatory quality levels that must be met. Beyond these mandatory levels, user satisfaction ratings in the range of 0 to 5 are assigned for other desirable characteristics.
- Objective measurements of functions are used to determine different levels of user satisfaction, which are then mapped to numerical ratings (see Table 13.2 for an example).

2. Importance Weighting:

- Each quality characteristic (e.g., usability, efficiency, maintainability) is assigned an importance rating on a scale of 1 to 5.
- These importance ratings reflect how critical each quality characteristic is to the overall evaluation of the software product.

3. Calculation of Overall Score:

- Weighted scores are calculated for each quality characteristic by multiplying the quality score by its importance weight.
- The weighted scores for all characteristics are summed to obtain an overall score for each software product.

4. Comparison and Preference Order:

- Products are then ranked in order of preference based on their overall scores. Higher scores indicate products that are more likely to satisfy user requirements and

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

preferences across the evaluated quality characteristics.

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

TABLE 13.3 Weighted quality scores

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

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

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

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

PRODUCT AND PROCESS METRICS

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

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

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

1. Product Metrics:

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

Examples:

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

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

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

2. Process Metrics:

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

Review Effectiveness: Measures how thorough and effective code reviews are in finding defects. **Defect Metrics:** Average number of defects found per hour of inspection, average time taken to correct defects, and average number of failures detected during testing per line of code. **Productivity Metrics:** Measures the efficiency of the development team in terms of output per unit of effort or time.

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

Differences:

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

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

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
2. Software product, such as size (LOC, function points), reliability (defects found per LOC), performance(response time), and usability (user satisfaction ratings).
3. **Evaluation Timing:** Product quality metrics are often measured and evaluated after the software product has been completed or at significant milestones during development.
4. **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.

5. Challenges:

- Predicting final product quality based on intermediate stages (like early code

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

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.

Once the expected value for the estimation variable has been determined, historical LOC or FP productivity data are applied. Are the estimates correct? The only reasonable answer to this

question is: “You can’t be sure.” Any estimation technique, no matter how sophisticated, must be cross-checked with another approach. Even then, common sense and experience must prevail.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

26.6.1 An Example of LOC-Based Estimation

As an example of LOC and FP problem-based estimation techniques, I consider a software package to be developed for a computer-aided design application for mechanical components. The software is to execute on an engineering workstation and must interface with various computer graphics peripherals including a mouse, digitizer, high-resolution color display, and laser printer. A preliminary statement of software scope can be developed:

The mechanical CAD software will accept two- and three-dimensional geometric data from an engineer. The engineer will interact and control the CAD system through a user interface that will exhibit characteristics of good human/machine interface design. All geometric data and other supporting information will be maintained in a CAD database. Design analysis modules will be developed to produce the required output, which will be displayed on a variety of graphics devices. The software will be designed to control and interact with peripheral devices that include a mouse, digitizer, laser printer, and plotter.

This statement of scope is preliminary—it is not bounded. Every sentence would have to be expanded to provide concrete detail and quantitative bounding. For example, before estimation can begin, the planner must determine what “characteristics of good human/machine interface design” means or what the size and sophistication of the “CAD database” are to be.

For our purposes, assume that further refinement has occurred and that the major software functions listed in Figure 26.2 are identified. Following the decomposition technique for LOC, an estimation

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

table (Figure 26.2) is developed. A range of LOC estimates is developed for each function. For example, the range of LOC estimates for the 3D geometric analysis function is optimistic, 4600 LOC; most likely, 6900 LOC; and pessimistic, 8600 LOC. Applying Equation 26.1, the expected value for the 3D geometric analysis function is 6800 LOC. Other estimates are derived in a similar

**Estimation
for the
LOC methods**

table

Function	Estimated LOC
User interface and control facilities (UICF)	2,300
Two-dimensional geometric analysis (2DGA)	5,300
Three-dimensional geometric analysis (3DGA)	6,800
Database management (DBM)	3,350
Computer graphics display facilities (CGDF)	4,950
Peripheral control function (PCF)	2,100
Design analysis modules (DAM)	8,400
<i>Estimated lines of code</i>	33,200

Do not succumb to the temptation to use this result as your project estimate. You should derive another result using a different approach.

702

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

fashion. By summing vertically in the estimated LOC column, an estimate of 33,200 lines of code is established for the CAD system.

A review of historical data indicates that the organizational average productivity for systems of this type is 620 LOC/pm. Based on a burdened labor rate of \$8000 per month, the cost per line of code is approximately \$13. Based on the LOC estimate and the historical productivity data, the total estimated project cost is \$431,000 and the estimated effort is 54 person-months.⁷

