



A T M E
College of Engineering



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

MODULE-5

Software Quality: Introduction, The place of software quality in project planning, Importance of software quality, Defining software quality, Software quality models, product versus process quality management.

Software Project Estimation: Observations on Estimation, Decomposition Techniques, Empirical Estimation Models.



- While quality is generally agreed to be ‘a good thing’, in practice what is meant by the ‘quality’ of a system can be vague.
- We need to define precisely what qualities we require of a system.
- However, we need to go further – we need to judge objectively whether a system meets our quality requirements and this needs measurement.

13.2 The Place of Software Quality in Project Planning

Quality will be of concern at all stages of project planning and execution, but will be of particular interest at the following points in the Step Wise framework .

Step 1:

Identify project scope and objectives Some objectives could relate to the qualities of the application to be delivered.

Step 2:

Identify project infrastructure Within this step, identifies installation standards and procedures. Some of these will almost certainly be about quality.



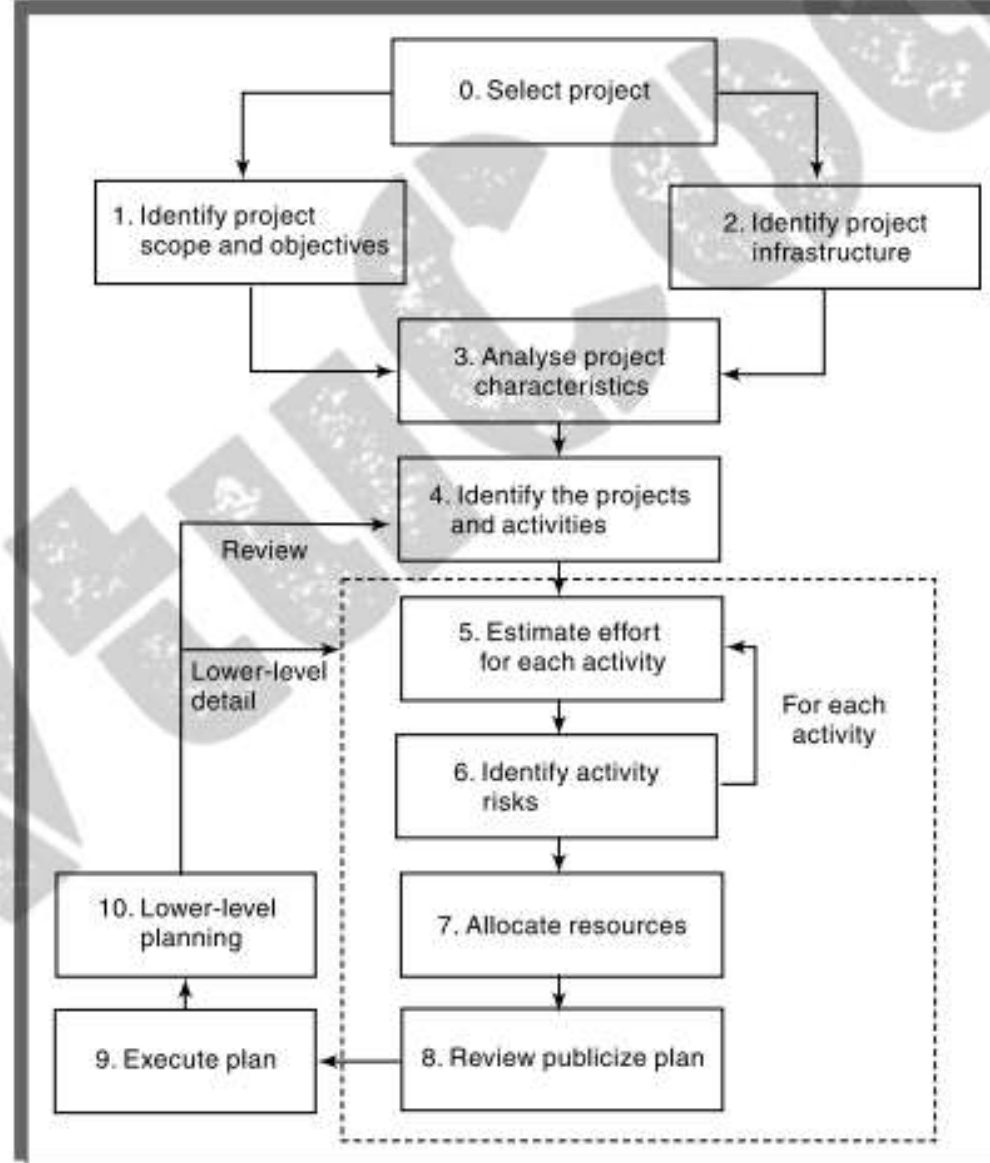
- ❖ Analyse project characteristics In activity the application to be implemented is examined to see if it has any special quality requirements.
- ❖ If, for example, it is safety critical then a range of activities could be added, such as n-version development where a number of teams develop versions 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 that the entry, exit and process requirements are identified for each activity. This is described later in this chapter.

Step 8:

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





13.3 The Importance of Software Quality

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

The special characteristics of software create special demands.

1. Increasing criticality of software

The final customer or user is naturally anxious about the general quality of software, especially its reliability.

This is increasingly so as organizations rely more on their computer systems and software is used in more safety-critical applications, for example to control aircraft.

2. The intangibility of software

can make it difficult to know that a project task was completed satisfactorily. Task outcomes can be made tangible by demanding that the developer produce 'deliverables' that can be examined for quality

3.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 are particularly difficult to control.



ATME

College of Engineering



13.4 Defining Software Quality

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

- ❖ Functional requirements define what the system is to do, the resource requirements specify allowable costs and the quality requirements state how well this system is to operate.
- ❖ Some qualities of a software product reflect the external view of software held by users, as in the case of usability. These external qualities have to be mapped to internal factors of which the developers would be aware. It could be argued, for example, that well-structured code is likely to have fewer errors and thus improve reliability.

• Need for Measurement

Simply defining quality is not enough; we must measure it to judge if a system meets requirements.

• Purpose of Measurement

Helps in assessing whether project goals and quality expectations are achieved.

• Good Quality Measures

A good measure should relate the observed quantity to a meaningful base (e.g., *faults per thousand lines of code* instead of total faults).

• Clarifying Quality

Attempting to measure quality helps clarify what “quality” means for that system or project.

Success Criteria

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

- Measurement answers the question: “*How do we know when we have been successful?*”

Types of Measures

- **Direct Measures:** Directly assess the quality (e.g., response time, defect rate).
- **Indirect Measures:** Assess indicators of quality (e.g., number of help desk calls → usability).

Setting Targets

- Quality measurements set performance targets for project teams.

Caution in Interpretation

- Improvement in a measured value must reflect *true* quality improvement, not manipulation of metrics.

- **Example of Misleading Measure**

Counting more errors during inspection may seem good, but could result from letting more defects reach that stage instead of preventing them earlier.

- **Conclusion**

Quality measurement must be meaningful, relevant, and genuinely reflect improvement in the product or process.

When there is concern about the need for a specific quality characteristic in a software product then a quality 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.



There could be several measurements applicable to a quality characteristic. For example, in the case of reliability, this might be measured in terms of:

- ❑ ***availability***: the percentage of a particular time interval that a system is usable;
- ❑ ***mean time between failures***: the total service time divided by the number of failures;
- ❑ ***failure on demand***: the probability that a system will not be available at the time required or the probability that a transaction will fail;
- ❑ ***support activity***: the number of fault reports that are generated and processed.



Software Quality Characteristics

• Different Quality Lists

- Various lists of software quality characteristics have been proposed by experts like **James McCall** and **Barry Boehm**.

• Lack of Agreement

- There has been **no universal agreement** on definitions of software quality attributes.

• Example – Maintainability

- Defined differently by experts:
 - Some see it as the **ease of finding and fixing errors**.
 - Others define it more broadly as the **ease of making any changes** to the software.

Example – Robustness

- Has multiple interpretations:
 - For some, it means **tolerance to incorrect input**.
 - For others, it means **ability to modify code without introducing new errors**.

Introduction of ISO 9126

- Introduced in **1991** to **standardize software quality definitions**.
- Served as a **foundation** for developing more **detailed quality standards**.

Document Details

- The original ISO 9126 was a **13-page document**, later expanded into **more comprehensive and detailed standards**.

Partly this is because people with differing motivations might be interested in software quality, namely:

- ***acquirers*** who are obtaining software from external suppliers;
- ***developers*** who are building a software product;
- ***independent evaluators*** who are assessing the quality of a software product, not for themselves but for a community of users – for example, those who might use a particular type of software tool as part of their professional practice.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

The difference between internal and external quality attributes has already been noted. ISO 9126 also introduces another type of quality – *quality in use* – for which the following elements have been identified:

- **effectiveness**: the ability to achieve user goals with accuracy and completeness;
- **productivity**: avoiding the excessive use of resources, such as staff effort, in achieving user goals;
- **safety**: within reasonable levels of risk of harm to people and other entities such as business, software, property and the environment;
- **satisfaction**: smiling users

Definition of 'Users'

- Includes not only system operators but also those who **maintain and enhance** the software.

Quality in Use Concept

- Software quality depends not only on the software itself but also on its **context of use**.

Example Scenario (IOE Case)

- Maintenance job reporting varies by **type of equipment** (e.g., photocopiers vs. printers).
- Different inputs are required to calculate maintenance costs.

Usage Distribution Example

- Initially, **95%** of jobs involve photocopiers, and **5%** involve printers.

Software Maturity

- Even after testing, some **errors may remain** in the operational system.
- As errors are reported and corrected, the software becomes **more mature** (fewer faults over time).

Effect of Change in Usage

- If printer maintenance jobs suddenly increase, **new faults** may appear in less-used parts of the software.

Reason for Fault Increase

- Previously untested or rarely used code paths become **active**, revealing hidden bugs.

Dynamic Quality Requirements

- Changes in how software is used can lead to **changes in quality requirements**.

Key Insight

- Software quality is **not static**; it evolves with **usage patterns** and **operational context**.



ISO 9126 identifies six major external software quality characteristics:

- **functionality**, which covers the functions that a software product provides to satisfy user needs;
- **reliability**, which relates to the capability of the software to maintain its level of performance;
- **usability**, which relates to the effort needed to use the software;
- **efficiency**, which relates to the physical resources used when the software is executed;
- **maintainability**, which relates to the effort needed to the make changes to the software;
- **portability**, which relates to the ability of the software to be transferred to a different environment.

ISO 9126 suggests sub-characteristics for each of the primary characteristics. They are useful as they clarify what is meant by each of the main characteristics.

Characteristic	Sub-characteristics
Functionality	Suitability
	Accuracy
	Interoperability
	Functionality compliance
	Security

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

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

Characteristic	Sub-characteristics
Usability	Understandability
	Learnability
	Operability
	Attractiveness
	Usability compliance

- DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING, AI & ML
- Note how ‘learnability’ is distinguished from ‘operability’.
 - A software tool could be easy to learn but time consuming to use because, say, it uses a large number of nested menus.
 - This might be fine for a package used intermittently, but not where the system is used for many hours each day. In this case ‘learnability’ has been incorporated at the expense of ‘operability’.
 - ‘Attractiveness’ is a recent addition to the sub-characteristics of usability and is especially important where users are not compelled to use a particular software product, as in the case of games and other entertainment products.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

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

- ‘Analysability’ is the ease with which the cause of a failure can be determined. ‘Changeability’ is the quality that others call ‘flexibility’: the latter name is a better one as ‘changeability’ has a different connotation in plain English – it might imply that the suppliers of the software are always changing it!
- ‘Stability’, on the other hand, does not refer to software never changing: it means that there is a low risk of a modification to the software having unexpected effects.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

Characteristic	Sub-characteristics
Portability	Adaptability
	Installability
	Coexistence
	Replaceability
	Portability compliance

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

- 'Portability compliance' relates to those standards that have a bearing on portability. The use of a standard programming language common to many software/hardware environments would be an example of this. 'Replaceability' refers to the factors that give 'upwards compatibility' between old software components and the new ones. 'Downwards' compatibility is not implied by the definition.
- 'Coexistence' refers to the ability of the software to share resources with other software components; unlike 'interoperability', no direct data passing is necessarily involved.

Once the requirements for the software product have been established, the following steps are suggested

1. ***Judge the importance of each quality characteristic for the application*** Thus reliability will be of particular concern with safety-critical systems while efficiency will be important for some real-time systems.
2. ***Select the external quality measurements within the ISO 9126 framework relevant to the qualities prioritized above*** Thus for reliability mean time between failures would be an important measurement, while for efficiency, and more specifically 'time behaviour', response time would be an obvious measurement.
3. ***Map measurements onto ratings that reflect user satisfaction*** For response time.

- **4. Identify the relevant internal measurements and the intermediate products in which they appear** This would only be important where software was being developed, rather than existing software being evaluated.
- For new software, the likely quality of the final product would need to be assessed during development.
- For example, where the external quality in question was time behaviour, at the software design stage an estimated execution time for a transaction could be produced by examining the software code and calculating the time for each instruction in a typical execution of the transaction.
- In our view the mappings between internal and external quality characteristics and measurements suggested in ISO 9126 standard are the least convincing elements in the approach.

5. Overall assessment of product quality

Combining Quality Ratings:

- It is difficult to combine ratings of different software quality characteristics into a single overall rating.

Reason for Difficulty:

- Different quality characteristics are measured in different ways, making comparison and combination complex.

Trade-offs Between Qualities:

- Improvement in one quality may reduce another.
- Example: Enhancing **efficiency** (time behaviour, resource utilization) may reduce **portability**.

Purpose of Quality Assessment:

- Conducted for various reasons such as **software development, acquisition, or independent assessment.**

During Development Phase:

- Assessment helps developers focus on key quality requirements.
- Aims to identify weaknesses early.
- An overall quality rating is **not necessary** at this stage.

3.7 Product versus Process Quality Management

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

Product-Based Approach:

- Focuses on measuring qualities of the final software product.
- Supported by PRINCE2 project management methodology.

Measurement Challenge:

- Easier to measure product qualities **after completion** than during development.
- Hard to predict final quality using attributes of intermediate products.

Alternative Approach:

- Evaluate the **quality of the development process** instead of just the final product.

System Development Process:

- Consists of multiple linked activities, where the output of one stage becomes the input for the next.

Sources of Errors:

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

- Errors can occur at any stage due to:
 - Defects in process (e.g., logic mistakes by developers).
 - Poor communication or unclear transfer of information between stages.

Cost of Errors:

- Errors not corrected early become **more expensive** to fix later.
- Each development phase passed increases rework effort.

Impact of Late Error Detection:

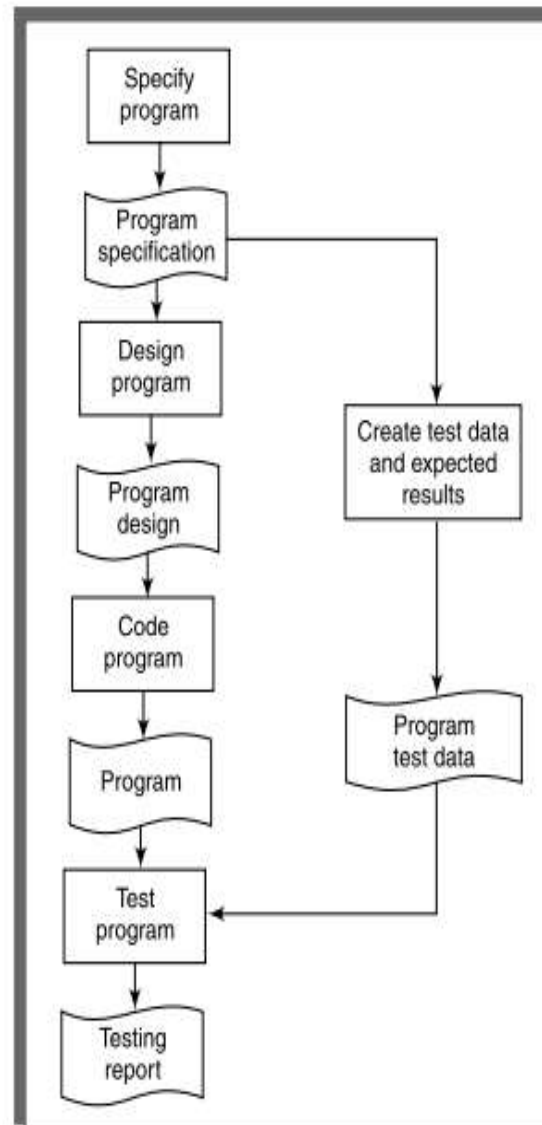
- An error in specification found during testing causes rework in all intermediate stages.
- Later stages are more detailed and less flexible to accommodate changes.

Errors should therefore be eradicated by careful examination of the deliverables of each step before they are passed on.

One way of doing this is by having the following process requirements for each step.

- 1. Entry requirements**, which have to be in place before an activity can start. An example would be that a comprehensive set of test data and expected results be prepared and approved before program testing can commence.
- 2. Implementation requirements**, which define how the process is to be conducted. In the testing phase, for example, it could be laid down that whenever an error is found and corrected, all test runs must be repeated, even those that have previously been found to run correctly

An example of the sequence of processes and deliverables



3.Exit requirements, which have to be fulfilled before an activity is deemed to have been completed. For example, for the testing phase to be recognized as being completed, all tests will have to have been run successfully with no outstanding error



13.8 Quality Management Systems

BS EN ISO 9001:2000

- **Outsourcing Concern:**

IOE may hire an external contractor for the maintenance contracts subsystem.

Main concern: ensuring the **quality of contractor's deliverables**.

- **Quality Control (QC):**

- Involves **rigorous testing** of all contractor-produced software.
- Requires **rework** if defects are found.
- **Time-consuming** process.

Quality Assurance (QA):

- Focuses on ensuring the **contractor's own QC processes** are effective.
- IOE checks whether the contractor has a proper **Quality Management System (QMS)** in place.

Nature of ISO Standards:

- Concerned with **certification of the development process**, not the end product.
- ISO 9000 → defines QMS fundamentals and terminology.
- ISO 9001 → application of QMS to product/service creation.
- ISO 9004 → focuses on **process improvement**.

Criticisms and Controversies:

- Misconception: certification means product quality is guaranteed.
- Stephen Halliday noted it only ensures adherence to self-defined standards.
- Certification can be **costly and time-consuming**, disadvantaging small firms.
- Risk of **means–ends inversion** – focusing more on certification than real product quality.

Working of the Standard:

- Identify **quality requirements**.
- Establish a **system to monitor** fulfillment of these requirements.
- Ensure **corrective action** is taken when deviations occur.



An overview of BS EN ISO 9001:2000 QMS requirements

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

The standard is built on a foundation of the following principles:

- understanding the requirements of customers so that they can be met, or even exceeded;
- leadership to provide the unity of purpose and direction needed to achieve quality objectives;
- involvement of staff at all levels;
- a focus on the individual processes which create intermediate or deliverable products and services;
- a focus on the systems of interrelated processes that create delivered products and services; Remember that these standards are designed for all kinds of production – not just software development. So ware Quality 301
- continuous improvement of processes;
- decision making based on factual evidence;
- building mutually beneficial relationships with suppliers

These principles are applied through cycles which involve the following activities:

1. Determining the needs and expectations of the customer.
2. Establishing a quality policy, that is, a framework which allows the organization's objectives in relation to quality to be defined.
3. Design of the processes which will create the products (or deliver the services) which will have the qualities implied in the organization's quality objectives;
4. Allocation of the responsibilities for meeting these requirements for each element of each process;
5. Ensuring that resources are available to execute these processes properly;
6. Design of methods for measuring the effectiveness and efficiency of each process in contributing to the organization's quality objectives;
7. Gathering of measurements;
8. Identification of any discrepancies between the actual measurements and the target values;
9. Analysis and elimination of the causes of discrepancies.

More detailed ISO 9001 requirements include:

- Documentation of objectives, procedures plans, and records relating to the actual operation of processes. The documentation must be subject to a change control system that ensures that it is current. Essentially one needs to be able to demonstrate to an outsider that the QMS exists and is actually adhered to.
- **Management responsibility** – the organization needs to show that the QMS and the processes that produce goods and services conforming to the quality objectives are actively and properly managed.
- **Resources** – an organization must ensure that adequate resources, including appropriately trained staff and appropriate infrastructure, are applied to the processes.

- Production should be characterized by:

- ✓ Planning;

- ✓ Determination and review of customer requirement

- Effective communications between the customer and supplier;

- Design and development being subject to planning, control and review;

- Requirements and other information used in design being adequately and clearly recorded;

- Design outcomes being verified, validated and documented in a way that provides sufficient information for those who have to use the designs;

- Changes to the designs should be properly controlled;

- DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML
- ✓ Adequate measures to specify and evaluate the quality of purchased components;
 - ✓ Production of goods and the provision of services should be under controlled conditions to ensure adequate provision of information, work instruction, equipment, measurement devices, and post-delivery activities;
 - ✓ Measurement — to demonstrate that products conform to standards, and the

26.5 SOFTWARE PROJECT ESTIMATION

- Software cost and effort estimation will never be an exact science.
- Too many variables—human, technical, environmental, political—can affect the ultimate cost of software and effort applied to develop it.
- However, software project estimation can be transformed from a black art to a series of systematic steps that provide estimates with acceptable risk.

To achieve reliable cost and effort estimates, a number of options arise:

1. Delay estimation until late in the project (obviously, we can achieve 100 percent accurate estimates after the project is complete!).
2. Base estimates on similar projects that have already been completed.
3. Use relatively simple decomposition techniques to generate project cost and effort estimates.
4. Use one or more empirical models for software cost and effort estimation

Early Cost Estimates:

- Though cost estimates are needed upfront, they are uncertain because information is limited in early stages.
- The longer you wait, the more accurate your estimates become.

Decomposition Approach:

- The project is divided into **major functions** and **related software engineering activities**.
- Estimation is done step-by-step for each component.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

Empirical Estimation Models:

- Based on **historical data and experience**.
- Use mathematical relationships:
- $d = f(v_i)$ where d = effort/cost/time and v_i = independent parameters (like LOC or Function Points).

Automated Estimation Tools:

- Implement decomposition and empirical models.
- Use inputs such as **team experience, development environment, and software characteristics**.
- Generate cost and effort estimates automatically.

Importance of Historical Data:

- Reliable estimation depends on accurate **past project data**.
- Without historical data, estimates become **unreliable**.

Use of Metrics:

- Software metrics (like those in Chapter 25) form the **basis of historical estimation data**.

Errors in Estimates:

- Estimates can go wrong if **past projects differ** from the current one in size, environment, or constraints.
- Past experience doesn't always guarantee future accuracy.

Cross-Checking Techniques:

- Multiple estimation methods should be used **together** to verify and improve reliability.
- Decomposition and empirical methods act as **mutual cross-checks**.

Divide-and-Conquer Strategy:

- Breaking down large estimation problems into smaller parts improves **accuracy and manageability**.



26.6 DECOMPOSITION TECHNIQUES

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

Estimation as Problem Solving:

- Software project estimation is treated as a **problem-solving activity**.
- The goal is to determine **cost and effort** for a software project.

Complexity of the Problem:

- The estimation problem is often **too complex** to handle as a single unit.
- Therefore, it must be **divided into smaller parts**.

Decomposition Approach:

- The problem is **decomposed** into smaller, more **manageable subproblems**.
- This makes the estimation process clearer and more accurate.



The accuracy of a software project estimate is predicated on a number of things:

- (1) The degree to which you have properly estimated the size of the product to be built;
- (2) The ability to translate the size estimate into human effort, calendar time, and dollars.
- (3) The degree to which the project plan reflects the abilities of the software team;
- (4) The stability of product requirements and the environment that supports the software engineering effort.

- In this section author consider the software sizing problem. Because a project estimate is only as good as the estimate of the size of the work to be accomplished, sizing represents your first major challenge as a planner.
- In the context of project planning, size refers to a quantifiable outcome of the software project. If a direct approach is taken, size can be measured in ***lines of code(LOC)***. If an indirect approach is chosen, size is represented as ***function points (FP)***.

Putnam and Myers suggest four different approaches to the sizing problem:

- ✓ ***Fuzzy logic*** sizing. This approach uses the approximate reasoning techniques that are the cornerstone of fuzzy logic. To apply this approach, the planner must identify the type of application, establish its magnitude on a qualitative scale, and then refine the magnitude within the original range.
- ✓ **Function point sizing**. The planner develops estimates of the information domain characteristics.

✓ **Standard component sizing.** Software is composed of a number of different “standard components” that are generic to a particular application area.

For example, the standard components for an information system are subsystems, modules, screens, reports, interactive programs, batch programs, files, LOC, and object-level instructions.

The project planner estimates the number of occurrences of each standard component and then uses historical project data to estimate the delivered size per standard component.

✓ **Change sizing.** This approach is used when a project encompasses the use of existing software that must be modified in some way as part of a project. The planner estimates the number and type (e.g., reuse, adding code, changing code, deleting code) of modifications that must be accomplished.



Estimation Variables:

- Lines of Code (LOC) and Function Points (FP) are used as measures for computing productivity metrics.

Two Uses of LOC and FP:

- (1) As estimation variables to size software components.
- (2) As baseline metrics from past projects to project cost and effort.

Common Approach:

- Start with a bounded software scope.
- Decompose it into smaller, estimable problem functions.

Sizing Components:

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

- Estimation can be done using LOC, FP, or other units like classes, objects, or business processes.

Baseline Productivity Metrics:

- Apply productivity measures like LOC/person-month (LOC/pm) or FP/person-month (FP/pm) to derive cost or effort.

Combining Estimates:

- Individual function estimates are combined to get the overall project estimate.

Domain-Based Productivity:

- Productivity varies widely; use domain-specific averages.

LOC Estimation:

- Requires detailed decomposition for accurate LOC estimates.
- Higher partitioning improves accuracy.

FP Estimation:

- Focuses on **information domain characteristics**: inputs, outputs, data files, inquiries, and external interfaces.
- Also includes 14 complexity adjustment factors.

• **Estimation by Range:**

- Estimate optimistic, most likely, and pessimistic values for each function or domain element.
- Provides a measure of uncertainty in estimation.

• **Historical Data Use:**

- Past project data should guide estimation; intuition may be used when data is unavailable.

26.6.3 An Example of LOC-Based Estimation

Purpose of Example:

- Demonstrates the use of **Lines of Code (LOC)** and **Function Point (FP)** estimation techniques.
- **Type:**
- Software for **Computer-Aided Design (CAD)** of **mechanical components**.

Execution Environment:

- Runs on an **engineering workstation**.

Peripheral Interfaces:

- Interfaces with **mouse, digitizer, high-resolution color display, laser printer, and plotter**.

Software Scope (Preliminary Statement):

- Accepts **2D and 3D geometric data** from engineers.
- Provides an **interactive user interface** with good **human-machine interaction design**.
- Maintains **geometric data** and related information in a **CAD database**.
- Includes **design analysis modules** for output generation and visualization.
- Controls and interacts with **multiple graphics peripherals**.

Estimation Focus:

- The above functional description will be **decomposed into modules/functions**.
- **LOC values** will be estimated for each module to compute total effort and cost.

Preliminary Scope:

- The initial project scope is **not bounded** and needs further **refinement** for estimation.

Need for Detail:

- Each statement (e.g., “*good human/machine interface design*”, “*CAD database*”) must be **quantitatively defined** before estimation.

Refined Scope:

- After refinement, **major software functions** are identified

LOC Estimation Table:

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

Estimation uses **decomposition technique** to assign **LOC ranges** to each function.

- **Example Function (3D Geometric Analysis):**
- **Optimistic: 4600 LOC**
- **Most likely: 6900 LOC**
- **Pessimistic: 8600 LOC**
- **Expected Value: 6800 LOC**

Total LOC Estimate:

- Summing all functions gives **33,200 LOC** for the entire CAD system.

Historical Productivity Data:

- Organizational average productivity = **620 LOC/person-month (LOC/pm)**.
- **Cost Estimation Parameters:**
- **Labor rate:** \$8000 per month.
- **Cost per LOC:** Approximately **\$13 per line**.

- **Final Estimates:**

- **Total Effort:** 54 person-months.
- **Total Cost:** \$431,000.

- **Conclusion:**

- LOC-based estimation combines **decomposition**, **historical data**, and **productivity rates** to predict **effort and cost** accurately.

26.6.4 An Example of FP-Based Estimation

Focus of Decomposition:

FP-based (Function Point-based) estimation emphasizes *information domain values* instead of software functions.

Information Domain Parameters:

- Estimation involves identifying and evaluating:
 - External Inputs
 - External Outputs
 - External Inquiries
 - Internal Logical Files
 - External Interfaces

Application Example:

- These parameters are estimated for the **CAD software** project.

Computation Method:

- The **Function Point (FP) value** is calculated using the standard FP computation technique described in Chapter 23.

Complexity Assumption:

- The **complexity weighting factor** is assumed to be **average** for this estimation.

Estimation Result:

- The outcomes of this FP-based estimation are presented in **Figure 26.3**.

FIGURE 26.3 Estimating information domain values

Information domain value	Opt.	Likely	Pess.	Est. count	Weight	FP count
Number of external inputs	20	24	30	24	4	97
Number of external outputs	12	15	22	16	5	78
Number of external inquiries	16	22	28	22	5	88
Number of internal logical files	4	4	5	4	10	42
Number of external interface files	2	2	3	2	7	15
<i>Count total</i>						320

26.6.5 Process-Based Estimation

Basis of Estimation:

The most common technique is **process-based estimation**, where the project is estimated based on the **process** to be used.

Decomposition Approach:

The process is **decomposed into smaller tasks**, and **effort** for each task is estimated individually.

Starting Point:

Estimation begins with identifying **software functions** from the **project scope** (similar to problem-based estimation).

Framework Activities:

Each software function is linked with a **set of framework activities**, which can be represented in a **tabular form** (as in Figure 26.4).

FIGURE 26.4 Process-based estimation table

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

Activity →	CC	Planning	Risk analysis	Engineering		Construction release		CE	Totals
Task →				Analysis	Design	Code	Test		
Function ↓									
UICF				0.50	2.50	0.40	5.00	n/a	8.40
2DGA				0.75	4.00	0.60	2.00	n/a	7.35
3DGA				0.50	4.00	1.00	3.00	n/a	8.50
CGDF				0.50	3.00	1.00	1.50	n/a	6.00
DBM				0.50	3.00	0.75	1.50	n/a	5.75
PCF				0.25	2.00	0.50	1.50	n/a	4.25
DAM				0.50	2.00	0.50	2.00	n/a	5.00
Totals	0.25	0.25	0.25	3.50	20.50	4.50	16.50		46.00
% effort	1%	1%	1%	8%	45%	10%	36%		

CC = customer communication CE = customer evaluation

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

Effort Estimation:

- **Effort (e.g., person-months)** is estimated for each **process activity** associated with each **software function**.

Labor Rate Application:

- **Average labor rates** (cost per unit effort) are applied, varying by task—
 - **Senior staff:** Early framework activities (higher cost).
 - **Junior staff:** Later stages like construction and release (lower cost).

Computation of Cost and Effort:

- **Total costs and efforts** for each function and framework activity are then **calculated**.

Cross-Verification:

- When performed along with **LOC (Lines of Code)** or **FP (Function Point)** estimation, results are **compared and reconciled**.

Reliability of Estimates:

- If estimates **agree**, they are considered **reliable**; if they **differ**, further **investigation and analysis** are required.



26.6.6 An Example of Process-Based Estimation

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

Example Used:

- The **CAD software** is used to demonstrate **process-based estimation**.

Project Scope:

- The **system configuration** and **software functions** remain **unchanged** as per the project scope.

Process-Based Table (Figure 26.4):

- The table lists **effort estimates (in person-months)** for each **software engineering activity** associated with every **CAD function**.

Activity Breakdown:

- **Engineering and construction release** activities are further **subdivided** into **major software engineering tasks**.

Gross Estimates:

- Broad effort estimates are included for **customer communication, planning, and risk analysis.**

Effort Totals:

- **Horizontal and vertical totals** in the table show the **effort distribution** across analysis, design, coding, and testing.

Effort Distribution Insight:

- **53% of total effort** is devoted to **front-end engineering** (requirements analysis and design), highlighting its **importance.**

Cost and Effort Summary:

- **Total estimated effort: 46 person-months**
- **Average labor rate: \$8,000 per month**
- **Total project cost: \$368,000**

Flexible Costing Option:

- If needed, **different labor rates** can be applied to each **framework activity** or **engineering task** for more precise costing.



26.6.7 Estimation with Use Cases

Developing an estimation approach with use cases is problematic for the following reasons:

- ✓ Use cases are described using many different formats and styles—there is no standard form.
- ✓ Use cases represent an external view (the user's view) of the software and can therefore be written at many different levels of abstraction.
- ✓ Use cases do not address the complexity of the functions and features that are described.
- ✓ Use cases can describe complex behavior (e.g., interactions) that involve many functions and features.



Use Case Variability:

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

- Unlike **LOC (Lines of Code)** or **Function Points (FP)**, a **use case** can vary widely in effort—some may take **months**, others only **days** to implement.

Estimation Challenge:

- Although many researchers have explored **use cases** for estimation, **no standard or proven method** exists yet.

Smith's Suggestion:

- **Use cases** can be used for estimation **only when considered within a “structural hierarchy.”**

Structural Hierarchy Constraints:

- Each level of the hierarchy can contain **no more than 10 use cases**.
- Each use case should include **up to 30 distinct scenarios**.

Level of Abstraction:

- **High-level use cases** (describing large systems) involve **greater development effort** than **low-level use cases** (for subsystems).

Preparation Before Estimation:

- Determine the **hierarchy level** of use cases.
- Identify the **average length** (in pages) of each use case.
- Define the **type of software** (e.g., real-time, business, WebApp, embedded, etc.).
- Outline a **rough system architecture**

Estimation Approach:

- Once these characteristics are known, **empirical data** can relate **LOC or FP per use case** for each hierarchy level.

Effort Calculation:

- **Historical data** are then applied to **compute total development effort** based on the estimated LOC or FP.



To illustrate how this computation might be made, consider the following relationship

$$\text{LOC estimate} = N \times \text{LOC}_{\text{avg}} + [(S_a/S_h - 1) + (P_a/P_h - 1)] \times \text{LOC}_{\text{adjust}}$$

where

- N = actual number of use cases
- LOC_{avg} = historical average LOC per use case for this type of subsystem
- $\text{LOC}_{\text{adjust}}$ = represents an adjustment based on n percent of LOC_{avg} where n is defined locally and represents the difference between this project and “average” projects
- S_a = actual scenarios per use case
- S_h = average scenarios per use case for this type of subsystem
- P_a = actual pages per use case
- P_h = average pages per use case for this type of subsystem



26.6.8 An Example of Use-Case–Based Estimation

Subsystem Grouping:

- The system is divided into **three main subsystem groups**:
 - **User Interface Subsystem** – includes **UICF**.
 - **Engineering Subsystem Group** – includes **2DGA**, **3DGA**, and **DAM**.
 - **Infrastructure Subsystem Group** – includes **CGDF** and **PCF**.

User Interface Subsystem:

- Described by **6 use cases**.
- Each use case has ≤ 10 **scenarios**.
- **Average length: 6 pages** per use case.

Engineering Subsystem Group:

- Described by **10 use cases** (higher-level in structural hierarchy).
- Each use case has ≤ 20 **scenarios**.
- **Average length: 8 pages** per use case.

Infrastructure Subsystem Group:

- Described by **5 use cases**.
- Each use case has ≈ 6 **scenarios**.
- **Average length: 5 pages** per use case.

Hierarchy Insight:

- **Engineering subsystem** use cases represent **higher abstraction levels** than those of the **user interface** or **infrastructure** subsystems.

FIGURE 26.5 Use-case estimation

	use cases	scenarios	pages	scenarios	pages	LOC	LOC estimate
User interface subsystem	6	10	6	12	5	560	3,366
Engineering subsystem group	10	20	8	16	8	3100	31,233
Infrastructure subsystem group	5	6	5	10	6	1650	7,970
Total LOC estimate							42,568



A T M E
College of Engineering



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML
