

Software Engineering

Chapter 5: Software Quality Management & Estimation

Study Notes for Engineering Students

Poly Notes Hub

Contents

- 1 5.1 Basic Quality Concepts
 - 2 5.2 Software Quality Assurance
 - 3 5.3 Statistical Software Quality Assurance
 - 4 5.4 Six Sigma Strategy
 - 5 5.5 Software Reliability
 - 6 5.6 The ISO 9000 Quality Standards
 - 7 5.7 McCall's Quality Factors
 - 8 5.8 Observations on Estimation
 - 9 5.9 The Project Planning Process, Software Scope & Feasibility, Resources
 - 10 5.10 Decomposition Techniques
 - 11 5.11 COCOMO II Model & the Make / Buy Decision
 - 12 Practice Section: 15 MCQs with Answer Key and 10 Broad Questions
-

5.1 Basic Quality Concepts

5.1.1 What is Quality?

Quality is the degree to which a product or service meets stated and implied needs, and satisfies the customer. For software, quality means that the software is **fit for use**: it conforms to explicitly stated functional and performance requirements, to explicitly documented development standards, and to implicit characteristics expected of all professionally developed software (such as ease of use and maintainability). If any of these is missing, software quality is suspect.

Quality is not an afterthought; it must be **built in** through good management, sound engineering methods and a disciplined process. It is much cheaper to prevent a defect than to find and fix it later.

5.1.2 Quality of Design and Quality of Conformance

- **Quality of design** refers to the characteristics that designers specify for a product: the grade of materials, tolerances, performance specifications. In software, it includes the requirements, specifications and design of the system. Higher-grade design gives higher design quality.
- **Quality of conformance** is the degree to which the design specifications are followed during manufacture (development). The greater the degree of conformance, the higher the level of quality of conformance. For software, it focuses on implementation: if the implementation follows the design and the system meets its requirements and performance goals, conformance quality is high.

5.1.3 Quality Control and Quality Assurance

Basis	Quality Control (QC)	Quality Assurance (QA)
Meaning	A series of inspections, reviews and tests used throughout the process to ensure each work product meets its requirements	A set of auditing and reporting functions that assess the effectiveness and completeness of quality control activities
Focus	Product (finding defects)	Process (preventing defects)
Approach	Corrective, detection-oriented	Preventive, process-oriented
Examples	Testing, technical reviews, inspections	Audits, process definition, standards, training
Responsibility	Developers and testers	Independent SQA group / management

QC includes a **feedback loop**: the process is tuned when the work products created fail to meet their specifications. QA provides management with data to gain insight into, and confidence in, product quality.

5.1.4 Quality Dimensions (Garvin)

Dimension	Meaning for software
Performance quality	Does the software deliver all content, functions and features as specified, with value to the user?
Feature quality	Does it provide features that surprise and delight users?
Reliability	Does it deliver all features without failure, available when needed?
Conformance	Does it conform to local and external software standards and to de facto design and coding conventions?
Durability	Can it be maintained or corrected without unintended side effects? Will its error rate stay stable over time?
Serviceability	Can it be maintained quickly and easily? Is the support team competent?
Aesthetics	Is the look and feel elegant and pleasing?
Perception	Is the quality judged by the user's impression of the developer's reputation and past products?

5.1.5 The Cost of Quality

The **cost of quality** includes all costs incurred in the pursuit of quality or in performing quality-related activities. It is divided into:

Category	Description	Examples
Prevention costs	Quality planning, formal technical reviews, test equipment, training	Process definition, standards, training of staff
Appraisal costs	Activities to gain insight into product condition "first time through" each process	In-process and inter-process inspection, equipment calibration, testing
Internal failure costs	Costs incurred when defects are detected before shipping	Rework, repair, failure mode analysis
External failure costs	Costs incurred when defects are found after shipping to the customer	Complaint resolution, product return and replacement, help-line support, warranty work, loss of reputation

The cost to find and fix a defect rises sharply (roughly in the ratio 1 : 10 : 100) as it moves from analysis and design, to testing, to field use. Hence the aim is to find errors as early as possible. Poor quality also brings **risks**: legal liability, loss of customers, and in safety-critical systems, harm to people.

5.1.6 Achieving Software Quality

Software quality is the result of good **project management**, sound **software engineering practice** (methods and tools), thorough **quality control** (reviews and testing), and organised **quality assurance** (standards, audits, measurement).

5.2 Software Quality Assurance

5.2.1 Definition

Software Quality Assurance (SQA) is an umbrella activity applied throughout the software process. It encompasses: a quality management approach, effective software engineering technology (methods and tools), formal technical reviews, a multi-tiered testing strategy, document control and change control, a procedure to ensure compliance with standards, and measurement and reporting mechanisms.

SQA is the planned and systematic pattern of activities needed to give confidence that the software product conforms to requirements. It is performed by an **independent SQA group** that works for management, not for the project team, so that its findings are objective.

5.2.2 Elements of SQA

- **Standards:** ISO 9001 and IEEE standards must be adopted, and the SQA group ensures conformance.
- **Reviews and audits:** technical reviews uncover errors; audits check that quality guidelines are followed.
- **Testing:** a multi-tiered test strategy to find errors; SQA ensures tests are properly planned and conducted.
- **Error and defect collection and analysis:** data on errors is analysed to understand how they are introduced and which activities can remove them.
- **Change management:** uncontrolled change is a major source of confusion and defects.
- **Education:** training of the team in best practice.
- **Vendor management:** quality requirements are included in contracts with suppliers of outsourced components.
- **Security management and safety:** ensure appropriate processes for protection and for hazard analysis.
- **Risk management:** ensure that risk management activities are properly conducted and contingency plans exist.

5.2.3 SQA Goals

- **Requirements quality:** correctness, completeness and consistency of the requirements model; reviews and prototypes are used.
- **Design quality:** every design element is examined for conformance to the requirements model; quality attributes such as modularity and low coupling are assessed.
- **Code quality:** source code and related work products conform to local coding standards and exhibit maintainability.
- **Quality control effectiveness:** resources are allocated so that QC finds the highest number of defects at the lowest cost.

5.2.4 SQA Activities

- Prepare an **SQA plan** for the project, developed during project planning and reviewed by all stakeholders.

- Participate in the development of the project's **software process description**, to ensure it complies with organisational policy and standards.
- **Review software engineering activities** to verify compliance with the defined software process; identify, document and track deviations.
- **Audit designated software work products** to verify compliance with those defined as part of the process; review selected work products and track deviations.
- **Ensure that deviations are documented** and handled according to a documented procedure.
- **Record any noncompliance** and report it to senior management; noncompliance items are tracked until resolved.

Contents of an SQA plan: purpose and scope; management structure, tasks and responsibilities; documentation to be produced; applicable standards, practices and conventions; reviews and audits; testing; problem reporting and corrective action; tools and techniques; code, media and supplier control; records retention; training; risk management.

5.2.5 Formal Technical Reviews (FTR)

An FTR is a software quality control activity performed by software engineers and others. Its objectives are to uncover errors in function, logic or implementation; verify that the software meets its requirements; ensure that it conforms to predefined standards; achieve software developed in a uniform manner; and make projects more manageable. Every FTR also serves as a training ground for junior engineers.

Types of FTR include the **walkthrough** (informal, led by the author, for understanding and feedback) and the **inspection** (formal, led by a trained moderator, with defined roles, checklists and metrics; the most effective defect-removal technique).

Review guidelines

- Review the product, not the producer.
- Set an agenda and maintain it.
- Limit debate and rebuttal; record issues for offline resolution.
- Enunciate problem areas, but do not attempt to solve every problem noted.
- Take written notes.
- Limit the number of participants (typically 3 to 5) and insist on advance preparation.
- Develop a checklist for each product that is likely to be reviewed.
- Allocate resources and schedule time for FTRs.
- Conduct meaningful training for all reviewers.
- Review your early reviews.

At the end of the review, all attendees decide whether to **accept the product without modification**, **reject it due to serious errors**, or **accept it provisionally** (minor errors to be corrected, no further review needed).

5.3 Statistical Software Quality Assurance

Statistical SQA reflects a growing trend throughout industry to become more quantitative about quality. It implies that information about software errors and defects is collected and categorised; an attempt is made to trace each error to its underlying cause; the **Pareto principle** is used to isolate the "vital few" causes (80 percent of defects trace to 20 percent of causes); and corrective action is applied to the vital few.

5.3.1 Steps in Statistical SQA

- 1 Information about software errors and defects is **collected and categorised**.
- 2 An attempt is made to **trace each error and defect to its underlying cause** (for example, nonconformance to specifications, design error, violation of standards, poor communication with the customer).
- 3 Using the **Pareto principle**, isolate the 20 percent (the "vital few") of causes that account for 80 percent of all defects.
- 4 Once the vital few causes have been identified, **move to correct the problems** that have caused the errors and defects.

5.3.2 Typical Categories of Error Causes

Code	Cause
IES	Incomplete or erroneous specifications
MCC	Misinterpretation of customer communication
IDS	Intentional deviation from specifications
VPS	Violation of programming standards
EDR	Error in data representation
ICI	Inconsistent component interface
EDL	Error in design logic
IET	Incomplete or erroneous testing
IID	Inaccurate or incomplete documentation
PLT	Error in programming language translation of design
HCI	Ambiguous or inconsistent human-computer interface
MIS	Miscellaneous

Example: if a collected data summary shows that IES, MCC and EDR together account for about 53 percent of all errors, then they are the vital few. Corrective actions might be to improve requirements gathering techniques (use of prototypes, use cases), improve communication with customers, and adopt better data modelling methods. Causes such as IDS or MIS, which contribute little, are ignored for now.

5.3.3 Error Indices

To develop an overall picture, the **error index (EI)** and **phase index (PI)** may be computed, weighting serious errors (S), moderate errors (M) and minor errors (T) differently, so that the effectiveness of each phase's QA and the overall trend (improving or worsening) can be tracked. A software organisation also tracks **defect removal efficiency (DRE)**:

DRE = E / (E + D), where E is the number of errors found before delivery and D is the number of defects found after delivery. The ideal value is 1.

5.4 Six Sigma Strategy

5.4.1 Introduction

Six Sigma is the most widely used strategy for statistical quality assurance in industry. Originally popularised by **Motorola** in the 1980s, the term "six sigma" is derived from six standard deviations, which corresponds to **3.4 defects per million opportunities (DPMO)**. It is a rigorous, disciplined, data-driven methodology for eliminating defects and driving processes toward near-perfection.

Sigma level	Defects per million opportunities (DPMO)	Yield
1 sigma	about 690,000	31 percent
2 sigma	about 308,000	69 percent
3 sigma	about 66,800	93.3 percent
4 sigma	about 6,210	99.38 percent
5 sigma	about 230	99.977 percent
6 sigma	3.4	99.99966 percent

5.4.2 Core Steps (DMAIC)

The Six Sigma methodology defines three **core steps**:

- **Define** customer requirements, deliverables and project goals through well-defined methods of customer communication.
- **Measure** the existing process and its output to determine current quality performance (collect defect metrics).
- **Analyse** defect metrics and determine the vital few causes.

For an **existing** software process that requires improvement, two additional steps are added (**DMAIC**):

- **Improve** the process by eliminating the root causes of defects.
- **Control** the process to ensure that future work does not reintroduce the causes of defects.

For **new** software development (a process being designed), the **DMADV** variant (also called DFSS, Design for Six Sigma) is used:

- **Design** the process to avoid root causes of defects and to meet customer requirements.
- **Verify** that the process model will, in fact, avoid defects and meet customer requirements.

5.4.3 Roles in a Six Sigma Organisation

Role	Responsibility
Executive / Champion	Sponsors projects, removes barriers and provides resources
Master Black Belt	Trains and mentors black belts; full-time Six Sigma expert
Black Belt	Leads improvement projects; applies statistical tools
Green Belt	Works on projects part-time alongside normal job

5.4.4 Six Sigma in Software

Six Sigma is applied to software to reduce defects in requirements, design, code and documentation, shorten cycle times and improve customer satisfaction. Tools include Pareto charts, cause-and-effect (fishbone) diagrams, control charts, histograms, scatter diagrams and process maps.

Basis	Six Sigma	ISO 9000
Focus	Defect reduction and process variation	Conformance to a documented quality management system
Approach	Data-driven, statistical	Documentation and audit based
Goal	Near-perfect quality (3.4 DPMO)	Consistent processes and customer satisfaction

5.5 Software Reliability

5.5.1 Definition

Software reliability is defined as the **probability of failure-free operation of a computer program in a specified environment for a specified time**. For example, a reliability of 0.999 over 8 hours of execution means that, in 1000 such runs, the software is expected to work correctly in 999. A **failure** is any nonconformance to software requirements. Unlike hardware, software failure is caused by **design faults** rather than wear, so reliability can grow as defects are removed.

5.5.2 Measures of Reliability and Availability

Measure	Meaning / formula
MTTF (Mean Time To Failure)	Average time the system operates before a failure occurs
MTTR (Mean Time To Repair)	Average time needed to repair the system after a failure
MTBF (Mean Time Between Failures)	MTBF = MTTF + MTTR
Failure rate	Number of failures per unit time; the inverse of MTTF for a constant rate
ROCOF	Rate of occurrence of failures
POFOD	Probability of failure on demand (for systems called occasionally)
Availability	Availability = $\text{MTTF} / (\text{MTTF} + \text{MTTR}) \times 100$ percent ; the probability that the program is operating according to requirements at a given point in time

Example: if MTTF = 90 hours and MTTR = 10 hours, then MTBF = 100 hours and availability = $90 / 100 = 90$ percent.

MTBF is a more useful measure than defects per KLOC because the user experiences failures, not defects; each defect has a different failure rate.

5.5.3 Software Safety

Software safety is a software quality assurance activity that focuses on the identification and assessment of potential hazards that may affect software negatively and cause an entire system to fail. Hazards are identified and categorised by criticality and risk; **fault tree analysis** is used to trace hazards to their causes; and design features (interlocks, redundancy, fail-safe modes) are specified to eliminate or control them. Reliability is concerned with failures in general, while safety is concerned with failures that can lead to a mishap.

5.5.4 Ways to Improve Reliability

- **Fault avoidance:** good design practice, information hiding, strongly typed languages, formal methods, reviews.
- **Fault detection and removal:** thorough testing, static analysis, and debugging.
- **Fault tolerance:** run-time techniques such as exception handling, N-version programming (design diversity), recovery blocks and checkpointing, so that the system keeps operating despite faults.
- Use **reliability growth models** (for example, Jelinski-Moranda, Musa) to estimate and predict reliability as defects are fixed during testing.
- Use an **operational profile** (how users actually use the system) to focus testing on the most frequently used functions.

5.6 The ISO 9000 Quality Standards

5.6.1 Introduction

A **quality assurance system** may be defined as the organisational structure, responsibilities, procedures, processes and resources for implementing quality management. **ISO 9000** describes quality assurance elements in generic terms that can be applied to any business, whether it manufactures aircraft, food or software. The standards are published by the **International Organization for Standardization (ISO)**.

5.6.2 The ISO 9000 Family

Standard	Purpose
ISO 9000	Fundamentals and vocabulary of quality management systems
ISO 9001	Requirements for a quality management system; the standard against which organisations are certified
ISO 9004	Guidelines for performance improvement and sustained success
ISO 19011	Guidelines for auditing management systems
ISO 90003 (earlier ISO 9000-3)	Guidelines for applying ISO 9001 to computer software

5.6.3 Principles and Requirements

ISO 9001 follows a **Plan-Do-Check-Act (PDCA)** cycle and is built on quality management principles such as customer focus, leadership, engagement of people, process approach, improvement, evidence-based decision making and relationship management.

For an organisation to be ISO 9001 compliant it must establish policies and procedures for: management responsibility; the quality system itself; contract review; design control; document and data control; purchasing; control of customer-supplied product; product identification and traceability; process control; inspection and testing; control of inspection, measuring and test equipment; inspection and test status; control of nonconforming product; corrective and preventive action; handling, storage, packaging, preservation and delivery; control of quality records; internal quality audits; training; servicing; and statistical techniques.

5.6.4 The ISO Certification (Registration) Process

- The organisation documents its quality system (quality manual, procedures, records).
- It implements the system and conducts **internal audits**.
- An accredited **registration body (auditor)** audits the system and, if satisfied, issues a **certificate**.
- **Surveillance audits** are carried out periodically; the certificate is renewed typically every three years.

5.6.5 Benefits and Limitations

Benefits	Limitations
International recognition and customer confidence	Does not guarantee a high-quality product, only a documented, consistent process
Consistent, documented processes	Can lead to heavy paperwork and bureaucracy
Better organisation and fewer errors	Does not prescribe specific software engineering techniques
Often a condition for tenders and exports	Cost of certification and maintenance

5.6.6 ISO 9001 versus SEI CMM

Basis	ISO 9001	SEI CMM / CMMI
Applicability	All industries	Software (and systems) organisations
Result	Certified or not (pass / fail)	Rating on maturity levels 1 to 5
Focus	Minimum acceptable quality system	Continuous improvement path
Basis of assessment	Compliance to documented system	Capability of key process areas

5.7 McCall's Quality Factors

McCall, Richards and Walters proposed a useful categorisation of factors that affect software quality, focusing on three aspects of a software product: its **operational characteristics**, its ability to **undergo change**, and its **adaptability to new environments**. There are **11 quality factors** organised into three categories, answering the questions "How does it run?", "Can it be changed?" and "Can it be moved?".

5.7.1 The Three Categories and Eleven Factors

Category	Factor	Meaning
Product Operation (how the product runs)	Correctness	The extent to which a program satisfies its specification and fulfils the customer's mission objectives
	Reliability	The extent to which a program can be expected to perform its intended function with required precision
	Efficiency	The amount of computing resources and code required to perform its function
	Integrity	The extent to which access to software or data by unauthorised persons can be controlled
	Usability	The effort required to learn, operate, prepare input for and interpret output of a program
Product Revision (can it be changed)	Maintainability	The effort required to locate and fix an error in a program
	Flexibility	The effort required to modify an operational program
	Testability	The effort required to test a program to ensure it performs its intended function
Product Transition (can it be moved)	Portability	The effort required to transfer the program from one hardware or software environment to another
	Reusability	The extent to which a program (or parts) can be reused in other applications
	Interoperability	The effort required to couple one system to another

Memory aid: Operation has 5 factors, Revision has 3 and Transition has 3 (5 + 3 + 3 = 11).

5.7.2 Quality Metrics

McCall also defined a set of **metrics** to measure the factors, by grading questions on a scale of 0 (low) to 10 (high) and combining them in a weighted form: $F_q = c_1 \times m_1 + c_2 \times m_2 + \dots + c_n \times m_n$, where F_q is the software quality factor, c_i are regression coefficients and m_i are the metrics that influence the factor. Because many metrics are subjective, and not all factors are important for every project, the model is applied selectively.

5.7.3 Other Quality Models (brief)

- **FURPS** (Hewlett-Packard): Functionality, Usability, Reliability, Performance, Supportability.
- **ISO 9126 / ISO 25010**: functionality, reliability, usability, efficiency, maintainability, portability (and security, compatibility in the newer version).

5.8 Observations on Estimation

5.8.1 Why Estimate?

Software **project estimation** predicts the effort (person-months), duration and cost needed to build the software. Planning requires a **roadmap** and estimation provides it. Estimation is not an exact science, since so many variables (human, technical, environmental, political) can affect cost and effort. Even so, a project cannot be managed without reasonable estimates. Estimation of resources, cost and schedule requires **experience, access to good historical information (metrics) and the courage to commit to quantitative predictions** when qualitative information is all that exists.

5.8.2 Factors that Affect Estimation Risk

- **Project complexity:** the higher the complexity, the harder the estimate; but complexity is relative to the experience of the team.
- **Project size:** as size increases, the interdependency among elements grows, so the error increases more than proportionally.
- **Degree of structural uncertainty:** how well requirements are established, and how well functions are defined.
- **Availability of historical information:** the more historical data, the lower the estimation risk.

5.8.3 Options for Achieving Reliable Estimates

- **Delay estimation** until late in the project (when all is known). This is attractive but impractical.
- **Base estimates on similar projects** already completed (works if the new project is truly similar).
- Use relatively simple **decomposition techniques** to generate cost and effort estimates.
- Use one or more **empirical models** (for example COCOMO) for software cost and effort estimation.

Good practice is to use **at least two methods** and reconcile the results. If the estimates differ widely, there is insufficient information; the scope must be re-examined.

5.8.4 Three-Point Estimation

For each function or task, three values are estimated: an **optimistic** value (a), a **most likely** value (m), and a **pessimistic** value (b). The expected value is:

$$EV = (a + 4m + b) / 6$$

This gives a weighted average (beta distribution) with greatest weight on the most likely value. *Example:* a = 4600, m = 6900, b = 8600 gives $EV = (4600 + 27600 + 8600) / 6 = 6800$.

5.8.5 What Is Estimated?

The main estimates are **size** (LOC, function points, object points), **effort** (person-months), **schedule** (calendar time) and **cost** (money). Effort and cost are derived from size and productivity data. Estimates should be revisited as the project progresses, and **the cone of uncertainty** shows that early estimates may be off by a factor of 2 or more, narrowing as requirements become clear.

5.9 The Project Planning Process

5.9.1 Objective of Planning

The objective of software project planning is to provide a framework that enables the manager to make **reasonable estimates of resources, cost and schedule**. These estimates are made within a limited time frame at the beginning of a project and should be **updated regularly** as the project progresses. Planning also defines best-case and worst-case scenarios so that outcomes can be bounded.

Planning task set (outline): establish project scope; determine feasibility; analyse risks; define required resources (human, reusable software, environment); estimate cost and effort; develop a project schedule.

5.9.2 Software Scope

Software scope describes the functions and features to be delivered to end users; the data that are input and output; the content presented to users; and the **performance, constraints, interfaces and reliability** that bound the system. Scope can be defined by a narrative description obtained after communication with all stakeholders, or by a set of use cases developed by end users. Functions in the scope statement are evaluated and refined to provide more detail before estimation begins, because both cost and schedule are functionally oriented. Performance considerations (processing and response time requirements) and constraints (limitations imposed by hardware, memory or other systems) are also assessed.

A good scope statement answers: What is the **context** (how the software fits into a larger system)? What are the **information objectives** (inputs and outputs)? What are the **function and performance** requirements? Is it **bounded** and unambiguous?

5.9.3 Feasibility

Once scope is understood, the next question is: "**Can we build software to meet this scope? Is the project feasible?**" Many software projects fail because of unrealistic promises. A feasibility study examines:

Type of feasibility	Question
Technical	Is the necessary technology available, and does the team have the skills? Can the performance requirements be met?
Economic (financial)	Do the benefits outweigh the costs? Is there a positive return on investment?
Operational	Will the system work in the organisation, and will users accept and use it?
Schedule	Can the system be delivered within the time required?
Legal	Does it comply with laws, regulations, licensing and contracts?
Resource	Are the people, tools and budget available?

If a project is judged infeasible, the scope may be reduced or the project cancelled. It is better to find this out early.

5.9.4 Resources

The second planning task is estimation of the resources required. Resources are described by four characteristics: a **description** of the resource, a statement of **availability**, the **time when** the resource will be required, and the **duration** of time it will be applied. Three major categories exist:

1. Human resources

Both organisational position (manager, senior engineer) and specialty (database, telecommunications) are specified. The number of people required is determined only after the estimate of development effort is made. Team size, skills and experience are factors.

2. Reusable software resources

Component-based development encourages reuse. Bennatan suggests four categories of reusable components:

Category	Description	Risk
Off-the-shelf components	Existing software acquired from a third party or developed internally for a past project; ready to use, fully validated	Low; usually only minor modification needed
Full-experience components	Existing specifications, designs, code or test data developed for past projects similar to the current one; team members have full experience in the application area	Low to moderate
Partial-experience components	Existing specifications, designs, code or test data related to the current project but requiring substantial modification; team has limited experience	Moderate to high
New components	Components that must be built by the team specifically for this project	Depends on novelty

3. Environment resources

The software engineering environment (SEE), also called the project support environment, incorporates hardware and software. **Hardware** provides the platform for tools needed to produce work products; **software tools** (CASE tools, compilers, IDEs, version control, testing tools) assist development. Where a product is built for specialised hardware (such as embedded systems), a **target environment** and sometimes special simulators and test equipment are also required.

5.10 Decomposition Techniques

Software project estimation is a form of problem solving, and in most cases the problem to be solved (developing a cost and effort estimate) is too complex to be considered in one piece. Hence we **decompose** the problem, re-characterising it as a set of smaller (and hopefully more manageable) problems. There are two views: decomposition of the **problem** (problem-based estimation) and of the **process** (process-based estimation).

5.10.1 Software Sizing

The accuracy of an estimate depends on how well the planner estimates the **size** of the product. Size may be measured in lines of code (LOC), function points (FP), object points or objects/components. Approaches to sizing include:

- **"Fuzzy-logic" sizing:** the planner identifies the application type, establishes its magnitude on a qualitative scale (small, medium, large), then refines within the range.
- **Function point sizing:** the planner develops estimates of the information domain characteristics.
- **Standard component sizing:** the software is composed of standard components (subsystems, modules, screens, reports); the planner estimates the number of each and uses historical data on size.
- **Change sizing:** used when an existing system is being modified; the number and type of changes are estimated.

5.10.2 Problem-Based Estimation: LOC and FP

The planner begins with a bounded statement of scope and decomposes the software into **problem functions** that can each be estimated individually. LOC or FP (the estimation variable) is estimated for each function. Baseline **productivity metrics** (for example LOC/person-month or FP/person-month, and cost per LOC or FP) are applied to the estimation variable to derive cost or effort for each function, and the function estimates are combined to produce an overall estimate.

LOC-based estimation. For each function, the planner gives optimistic, most likely and pessimistic LOC estimates, and the expected value is computed with $EV = (a + 4m + b) / 6$.

Example: total estimated size = 33,200 LOC. Using an average productivity of 620 LOC per person-month and a labour rate of 8,000 per person-month: estimated effort = $33,200 / 620$, about **54 person-months**; estimated cost = $54 \times 8,000$, about **430,000**.

Function-Point (FP) based estimation. FP, proposed by Albrecht, measures the functionality delivered, independent of language. It is computed from the **information domain values**:

Information domain value	Meaning
External Inputs (EI)	Each input that provides distinct application-oriented data to the software (screens, forms, files)
External Outputs (EO)	Each output that provides application-oriented information to the user (reports, screens, error messages)
External Inquiries (EQ)	An online input that results in an immediate software response in the form of an online output
Internal Logical Files (ILF)	Each logical grouping of data that resides within the application's boundary (a table, a file)
External Interface Files (EIF)	Logical groupings of data that reside outside the application but are used by it

Each count is multiplied by a **weighting factor** based on complexity:

Measurement parameter	Simple	Average	Complex
External Inputs (EI)	3	4	6
External Outputs (EO)	4	5	7

Measurement parameter	Simple	Average	Complex
External Inquiries (EQ)	3	4	6
Internal Logical Files (ILF)	7	10	15
External Interface Files (EIF)	5	7	10

The weighted sum is the **count total** (unadjusted FP). It is then adjusted using 14 **value adjustment factors** F_i (for example data communications, distributed processing, performance, heavily used configuration, transaction rate, online data entry, end-user efficiency, online update, complex processing, reusability, installation ease, operational ease, multiple sites, facilitation of change), each rated from 0 (not important) to 5 (absolutely essential):

$$FP = \text{Count total} \times [0.65 + 0.01 \times \text{Sum}(F_i)]$$

Example: count total = 320 and $\text{Sum}(F_i) = 52$ gives $FP = 320 \times (0.65 + 0.52) = 320 \times 1.17$, about **374 FP**. With a productivity of 6.5 FP per person-month, effort is $374 / 6.5$, about **58 person-months**.

Basis	LOC-based	FP-based
Measures	Physical size of code	Functionality delivered
Language dependence	Dependent on programming language	Language independent
Available	Only after coding (estimated earlier from analogy)	Can be estimated early from the requirements
Weakness	Difficult to estimate early; differs across languages	Subjective complexity ratings; needs training
Productivity measure	LOC per person-month	FP per person-month

5.10.3 Process-Based Estimation

The most common technique for estimating a project is to base the estimate on the **process** that will be used. The process is decomposed into a relatively small set of **activities or tasks** (for example communication, planning, modelling, construction, deployment), and the effort required to accomplish each is estimated. The steps are:

- 1 Obtain the functions from the problem-based decomposition (the project scope).
- 2 Estimate the **effort (person-months)** required to accomplish each framework activity for each function.
- 3 Apply average labour rates (money per person-month) to the effort estimated for each activity, since rates differ for different activities.
- 4 Compute the total cost and effort for each function and each framework activity.
- 5 **Compare** the process-based result with the problem-based (LOC / FP) estimate. If the two estimates agree, there is good reason to believe they are reliable; if they differ widely, the cause (insufficient information, misuse of historical data) must be investigated.

5.11 COCOMO II Model & the Make / Buy Decision

5.11.1 Empirical Estimation Models

An **empirical estimation model** uses formulas derived from data collected from past projects: $E = A + B \times (ev)^C$, where E is effort, A, B, C are empirically derived constants and ev is the estimation variable (LOC or FP). **COCOMO (CONstructive COst MOdel)**, proposed by **Barry Boehm** in 1981, is the most widely used. It estimates effort, development time and cost from the size of the product and a set of cost drivers.

COCOMO 81 (reference). The original model had three levels (Basic, Intermediate, Detailed) and three project classes:

Project class	Description	Effort (person-months)	Time (months)
Organic	Small team, familiar environment, small projects (up to about 50 KLOC)	$2.4 \times (KLOC)^{1.05}$	$2.5 \times (Effort)^{0.38}$
Semi-detached	Mixed experience, medium size, intermediate constraints	$3.0 \times (KLOC)^{1.12}$	$2.5 \times (Effort)^{0.35}$
Embedded	Tight hardware, software and operational constraints; complex	$3.6 \times (KLOC)^{1.20}$	$2.5 \times (Effort)^{0.32}$

5.11.2 COCOMO II

COCOMO II (1995, calibrated 2000) is a revised model for modern development practices: iterative and incremental processes, reuse, component-based development, rapid prototyping and use of CASE tools. It provides three sub-models, used at different stages of the project:

Model	Used when	Size measure
Application composition model	Early prototyping phase, when the user interface, system interaction and technology maturity are being explored	Object points
Early design model	After requirements are stabilised and basic architecture established; to explore alternative architectures	Function points (converted to LOC) with 7 cost drivers
Post-architecture model	During construction, when the architecture is defined and the project is under development; the most detailed model	Source lines of code (KSLOC) or FP, with 17 cost drivers

(a) Application composition model: object points. An **object point** is a weighted count of screens (at the user interface), reports and 3GL components likely to be required to build the application, each classified as simple, medium or difficult:

Object type	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8

Object type	Simple	Medium	Difficult
3GL component			10

NOP = (object points) x [(100 - %reuse) / 100], where NOP is the new object points. **Effort = NOP / PROD** person-months, where PROD is the productivity rate that depends on developer experience and ICASE (integrated CASE) maturity: very low 4, low 7, nominal 13, high 25, very high 50 NOP per person-month.

(b) Post-architecture model: the main equation.

Effort (PM) = A x (Size)^E x Product of Effort Multipliers (EM)

where **A = 2.94** (calibration constant), **Size** is in thousands of source lines of code (KSLOC), and the exponent **E = B + 0.01 x Sum(SF_i)**, with **B = 0.91** and SF the **five scale factors**:

Scale factor	Meaning
PREC	Precedentedness: how new the project is to the organisation
FLEX	Development flexibility: how rigid the requirements and process are
RESL	Architecture / risk resolution: extent of risk elimination
TEAM	Team cohesion: stakeholder communication and cooperation
PMAT	Process maturity: based on the CMM maturity level

Each scale factor is rated from very low to extra high. A project with a larger exponent (more diseconomy of scale) takes more effort per KSLOC as size grows.

Schedule (development time): TDEV = 3.67 x (PM)^F, where **F = 0.28 + 0.2 x (E - B)**. **Average team size = PM / TDEV**.

Cost drivers (effort multipliers). The post-architecture model has 17 drivers in four groups, each rated from very low to extra high, giving a multiplier (nominal = 1.0):

Group	Drivers
Product	RELY (required reliability), DATA (database size), CPLX (product complexity), RUSE (required reusability), DOCU (documentation match to life-cycle needs)
Platform	TIME (execution time constraint), STOR (main storage constraint), PVOL (platform volatility)
Personnel	ACAP (analyst capability), PCAP (programmer capability), PCON (personnel continuity), APEX (applications experience), PLEX (platform experience), LTEX (language and tool experience)
Project	TOOL (use of software tools), SITE (multi-site development), SCED (required development schedule)

Worked example: a project of 50 KSLOC, with Sum(SF) = 18 and all effort multipliers nominal (product = 1).

- $E = 0.91 + 0.01 \times 18 = 1.09$

- $PM = 2.94 \times (50)^{1.09}$, which is about 2.94×71.1 , about **209 person-months**
- $F = 0.28 + 0.2 \times (1.09 - 0.91) = 0.316$
- $TDEV = 3.67 \times (209)^{0.316}$, which is about 3.67×5.41 , about **20 months**
- Average staff = $209 / 20$, about **10 people**

5.11.3 Strengths and Limitations of COCOMO II

Strengths	Limitations
Well documented, based on a large project database	Accuracy depends on the quality of size input (KSLOC)
Handles reuse, modern processes and different stages	Many cost drivers are subjective
Allows calibration to the organisation's own data	Needs local calibration for best results
Provides effort, schedule and staffing together	Does not account for all factors (for example, requirement volatility beyond the drivers)

5.11.4 The Make / Buy Decision

In many software application areas it is often more cost effective to **acquire** computer software than to develop it. The software engineering manager faces a **make / buy decision** that can be further complicated by acquisition options:

- **Off-the-shelf (buy):** software may be purchased (or licensed) off the shelf.
- **Full-experience or partial-experience components:** existing components may be acquired and then modified to meet specific needs.
- **Contracted:** the software may be custom built by an outside contractor to meet the purchaser's specifications.
- **Build (make):** the software is developed in-house from scratch.

The steps in the decision are: develop specifications for the desired software; estimate the internal cost and delivery date; select candidate applications or contractors; compare the options on cost, schedule and risk; and decide. A **decision tree** analysis is commonly used: each path is assigned a probability and a cost, and the **expected cost** is computed as:

Expected cost = Sum of (path probability x estimated path cost)

Example (costs in thousands):

Option	Outcomes (probability, cost)	Expected cost
Build in-house	Simple (0.30, 380); Difficult (0.70, 450)	$0.30 \times 380 + 0.70 \times 450 = \mathbf{429}$
Reuse existing components	Minor changes (0.40, 275); Major changes (0.60): simple (0.20, 310) or complex (0.80, 490)	$0.40 \times 275 + 0.60 \times (0.20 \times 310 + 0.80 \times 490) = 110 + 272.4 = \mathbf{382}$
Buy off the shelf	Minor changes (0.70, 210); Major changes (0.30, 400)	$0.70 \times 210 + 0.30 \times 400 = \mathbf{267}$
Contract out	Without changes (0.60, 350); With changes (0.40, 500)	$0.60 \times 350 + 0.40 \times 500 = \mathbf{410}$

On expected cost alone, **buying** is cheapest in this example. However, the decision must also take into account non-cost criteria:

- **Delivery date:** will the product be available when needed?
- **Development experience** of the team and vendor reputation.
- **Conformance to requirements:** how well does the package match the needs?
- **Local "politics"** and organisational commitment.
- **Likelihood of change**, vendor support and the future of the product.
- **Performance and security** of the application, and sensitivity of the requirements.

Outsourcing. The decision to outsource may be strategic or tactical. Advantages include lower cost, access to specialised skills and freed internal resources; risks include loss of control, dependence on the vendor, communication problems and intellectual property concerns.

Practice Section

Part A: 15 Multiple Choice Questions

Q1. The cost of quality includes:

- (a) Only the cost of testing
- (b) Only the cost of rework
- (c) Only the cost of prevention
- (d) Prevention, appraisal and failure costs

Q2. Six Sigma corresponds to how many defects per million opportunities?

- (a) 3.4
- (b) 34
- (c) 340
- (d) 6,210

Q3. In the DMAIC approach, the letter "C" stands for:

- (a) Check
- (b) Compare
- (c) Control
- (d) Customer

Q4. MTBF is equal to:

- (a) $MTTF + MTTR$
- (b) $MTTF - MTTR$
- (c) $MTTF / MTTR$
- (d) $MTTR / MTTF$

Q5. If $MTTF = 90$ hours and $MTTR = 10$ hours, the availability is:

- (a) 80 percent
- (b) 90 percent
- (c) 95 percent
- (d) 99 percent

Q6. ISO 9001 specifies requirements for a:

- (a) Programming language
- (b) Testing tool
- (c) Quality management system
- (d) Project estimation model

Q7. Which of the following is a McCall quality factor under Product Transition?

- (a) Correctness
- (b) Maintainability
- (c) Efficiency
- (d) Portability

Q8. How many quality factors are in McCall's model?

- (a) 9
- (b) 11
- (c) 13
- (d) 15

Q9. In statistical SQA, the Pareto principle is used to identify:

- (a) The "vital few" causes of most defects
- (b) The cost of the project
- (c) The size of the team
- (d) The number of test cases

Q10. The function point formula is:

- (a) $FP = \text{Count total} \times \text{Sum}(Fi)$
- (b) $FP = \text{Count total} + 0.01 \times \text{Sum}(Fi)$
- (c) $FP = \text{Count total} \times [0.65 + 0.01 \times \text{Sum}(Fi)]$
- (d) $FP = \text{Count total} / [0.65 + 0.01 \times \text{Sum}(Fi)]$

Q11. Object points are used as the size measure in which COCOMO II sub-model?

- (a) Application composition model
- (b) Early design model
- (c) Post-architecture model
- (d) Basic COCOMO

Q12. Which is NOT a category of resources in project planning?

- (a) Human resources
- (b) Reusable software resources
- (c) Entertainment resources
- (d) Environment resources

Q13. Using the three-point formula, if $a = 4600$, $m = 6900$ and $b = 8600$, the expected value is:

- (a) 6,500
- (b) 6,700
- (c) 6,800
- (d) 7,000

Q14. In a make/buy decision tree, the expected cost of an option is calculated as:

- (a) The sum of all path costs
- (b) The sum of (path probability $\times$ path cost)
- (c) The highest path cost
- (d) The average of probabilities

Q15. In COCOMO II post-architecture model, the constant A is:

- (a) 1.05
- (b) 2.94
- (c) 3.67
- (d) 0.91

Answer Key

Q	Ans	Q	Ans	Q	Ans
1	(d)	6	(c)	11	(a)
2	(a)	7	(d)	12	(c)
3	(c)	8	(b)	13	(c)
4	(a)	9	(a)	14	(b)
5	(b)	10	(c)	15	(b)

Part B: 10 Broad Questions

- 1 Define software quality. Explain quality of design and conformance, the dimensions of quality and the cost of quality.
- 2 What is Software Quality Assurance? Explain the elements, goals and activities of SQA. Differentiate between quality control and quality assurance.
- 3 Explain formal technical reviews and the review guidelines. What is the SQA plan?
- 4 Explain statistical software quality assurance with the steps involved. Describe the typical categories of error causes with an example of the Pareto principle.
- 5 What is Six Sigma? Explain DMAIC and DMADV with the sigma levels and roles in Six Sigma.
- 6 Define software reliability. Explain MTTF, MTTR, MTBF and availability with examples. How can software reliability and safety be improved?
- 7 Explain the ISO 9000 family of quality standards. Describe the requirements, certification process, benefits and limitations. Compare ISO 9001 with SEI CMM.
- 8 Explain McCall's quality factors with a diagram or table, grouping them into product operation, revision and transition.
- 9 Explain the project planning process: software scope, feasibility and the three categories of resources. Why is estimation difficult, and how can estimation risk be reduced?
- 10 Explain decomposition techniques (LOC and FP based estimation) with examples. Describe the COCOMO II model and the make/buy decision using a decision tree.