MODULE-V

Project Planning: Software pricing (Sec 23.1). Plan-driven development (Sec 23.2). Project scheduling (Sec 23.3): Estimation techniques (Sec 23.5).

Quality management: Software quality (Sec 24.1). Reviews and inspections (Sec 24.3). Software measurement and metrics (Sec 24.4). Software standards (Sec 24.2)

Project Planning

Software pricing

- Project planning involves breaking down the work into parts and assign these to project team members, anticipate problems that might arise and prepare tentative solutions to those problems.
- The project plan, which is created at the start of a project, is used to communicate how the work will be done to the project team and customers, and to help assess progress on the project.

Project planning

- Project planning involves breaking down the work into parts and assigns these to project team members, anticipate problems that might arise and prepare tentative solutions to those problems.
- The project plan, which is created at the start of a project, is used to communicate how the work will be done to the project team and customers, and to help assess progress on the project.

Planning stages

- At the proposal stage, when you are bidding for a contract to develop or provide a software system.
- During the project startup phase, when you have to plan who will work on the project, how the project will be broken down into increments, how resources will be allocated across your company, etc.
- Periodically throughout the project, when you modify your plan in the light of experience gained and information from monitoring the progress of the work.

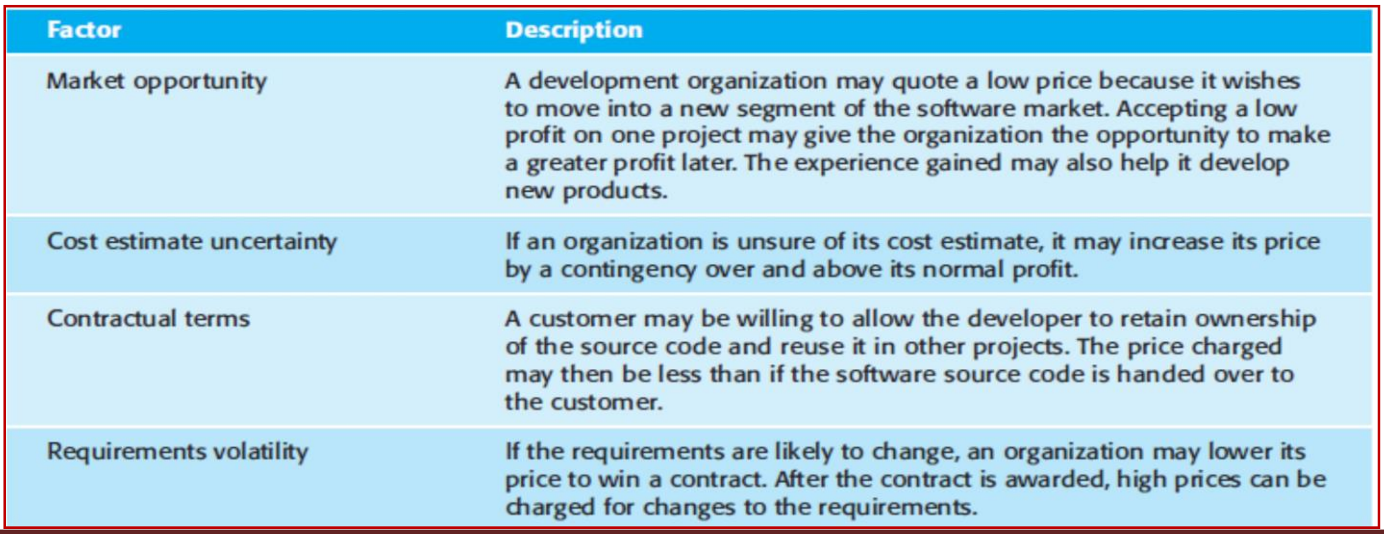
Proposal planning

- Planning may be necessary with only outline software requirements.
- The aim of planning at this stage is to provide information that will be used in setting a price for the system to customers.

Software pricing

- Estimates are made to discover the cost, to the developer, of producing a software system.
    - You take into account, hardware, software, travel, training and effort costs.
- There is not a simple relationship between the development cost and the price charged to the customer.
- Broader organizational, economic, political and business considerations influence the price charged.

Factors affecting software pricing



Factor	Description
Market opportunity	A development organization may quote a low price because it wishes to move into a new segment of the software market. Accepting a low profit on one project may give the organization the opportunity to make a greater profit later. The experience gained may also help it develop new products.
Cost estimate uncertainty	If an organization is unsure of its cost estimate, it may increase its price by a contingency over and above its normal profit.
Contractual terms	A customer may be willing to allow the developer to retain ownership of the source code and reuse it in other projects. The price charged may then be less than if the software source code is handed over to the customer.
Requirements volatility	If the requirements are likely to change, an organization may lower its price to win a contract. After the contract is awarded, high prices can be charged for changes to the requirements.

Factor	Description
Financial health	Developers in financial difficulty may lower their price to gain a contract. It is better to make a smaller than normal profit or break even than to go out of business. Cash flow is more important than profit in difficult economic times.

Plan-driven development

- Plan-driven or plan-based development is an approach to software engineering where the development process is planned in detail.
  - Plan-driven development is based on engineering project management techniques and is the 'traditional' way of managing large software development projects.
- A project plan is created that records the work to be done, which will do it, the development schedule and the work products.
- Managers use the plan to support project decision making and as a way of measuring progress.

Plan-driven development – pros and cons

- The arguments in favor of a plan-driven approach are that early planning allows organizational issues (availability of staff, other projects, etc.) to be closely taken into account, and that potential problems and dependencies are discovered before the project starts, rather than once the project is underway.
- The principal argument against plan-driven development is that many early decisions have to be revised because of changes to the environment in which the software is to be developed and used.

Project plans

- In a plan-driven development project, a project plan sets out the resources available to the project, the work breakdown and a schedule for carrying out the work.
- Plan sections
  - Introduction: This briefly describes the objectives of the project and sets out the constraints (e.g., budget, time, etc.) that affect the management of the project.
  - Project organization: This describes the way in which the development team is organized, the people involved, and their roles in the team.
  - Risk analysis: This describes possible project risks, the likelihood of these risks arising, and the risk reduction strategies that are proposed
  - Hardware and software resource requirements: This specifies the hardware and support software required to carry out the development. If hardware has to be bought, estimates of the prices and the delivery schedule may be included.
  - Work breakdown: This sets out the breakdown of the project into activities and identifies the milestones and deliverables associated with each activity. Milestones are key stages in the project where progress can be assessed; deliverables are work products that are delivered to the customer.
  - Project schedule: This shows the dependencies between activities, the estimated time required to reach each milestone, and the allocation of people to activities.
  - Monitoring and reporting mechanisms: This defines the management reports that should be produced, when these should be produced, and the project monitoring mechanisms to be used.

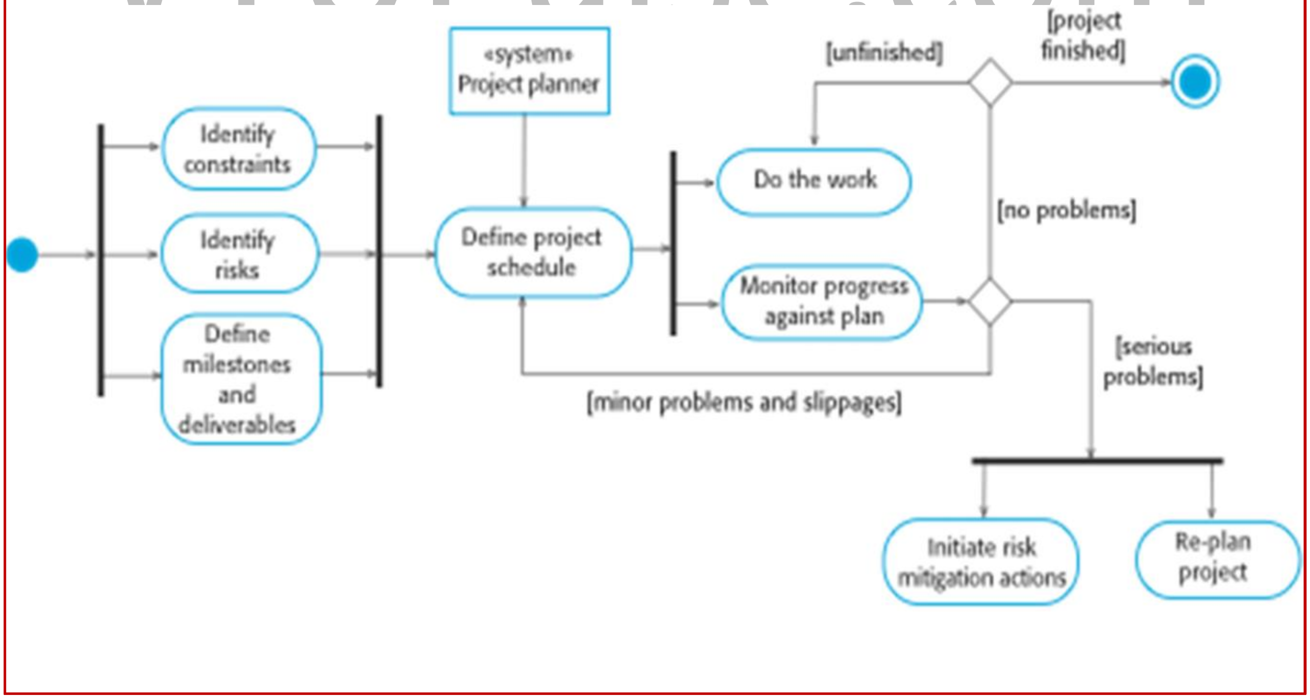
Project plan supplements

Plan	Description
Quality plan	Describes the quality procedures and standards that will be used in a project.
Validation plan	Describes the approach, resources, and schedule used for system validation.
Configuration management plan	Describes the configuration management procedures and structures to be used.
Maintenance plan	Predicts the maintenance requirements, costs, and effort.
Staff development plan	Describes how the skills and experience of the project team members will be developed.

The planning process

- Project planning is an iterative process that starts when you create an initial project plan during the project startup phase.
- Plan changes are inevitable.
    - As more information about the system and the project team becomes available during the project, you should regularly revise the plan to reflect requirements, schedule and risk changes.
    - Changing business goals also leads to changes in project plans. As business goals change, this could affect all projects, which may then have to be re-planned.

The project planning process



Project scheduling

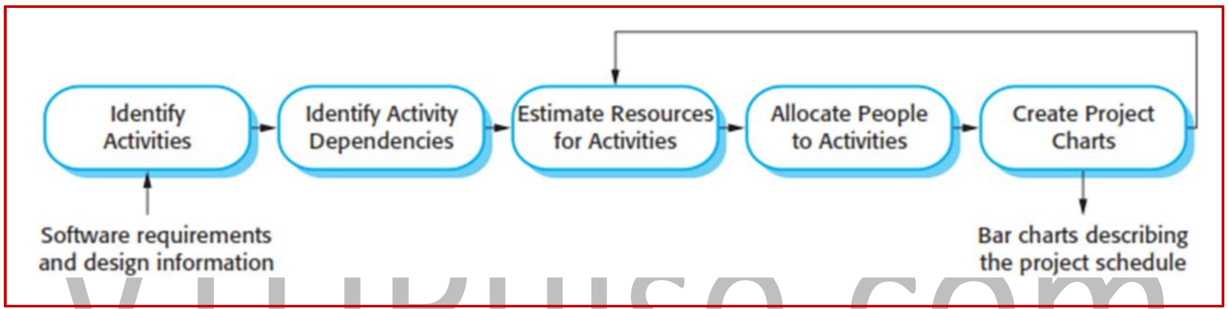
- Project scheduling is the process of deciding how the work in a project will be organized as separate tasks, and when and how these tasks will be executed.
- You estimate the calendar time needed to complete each task, the effort required and who will work on the tasks that have been identified.
- You also have to estimate the resources needed to complete each task, such as the disk space required on a server, the time required on specialized hardware, such as a simulator, and what the travel budget will be.

Project scheduling activities

- Split project into tasks and estimate time and resources required to complete each task.
- Organize tasks concurrently to make optimal use of workforce.
- Minimize task dependencies to avoid delays caused by one task waiting for another to complete.
- Dependent on project manager's intuition and experience.

Milestones and deliverables

- Milestones are points in the schedule against which you can assess progress, for example, the handover of the system for testing.
- Deliverables are work products that are delivered to the customer, e.g. a requirements document for the system.


Identify Activities
Identify Activity Dependencies
Estimate Resources for Activities
Allocate People to Activities
Create Project Charts
Software requirements and design information
Bar charts describing the project schedule

Scheduling problems

- Estimating the difficulty of problems and hence the cost of developing a solution is hard.
- Productivity is not proportional to the number of people working on a task.
- Adding people to a late project makes it later because of communication overheads.
- The unexpected always happens. Always allow contingency in planning.

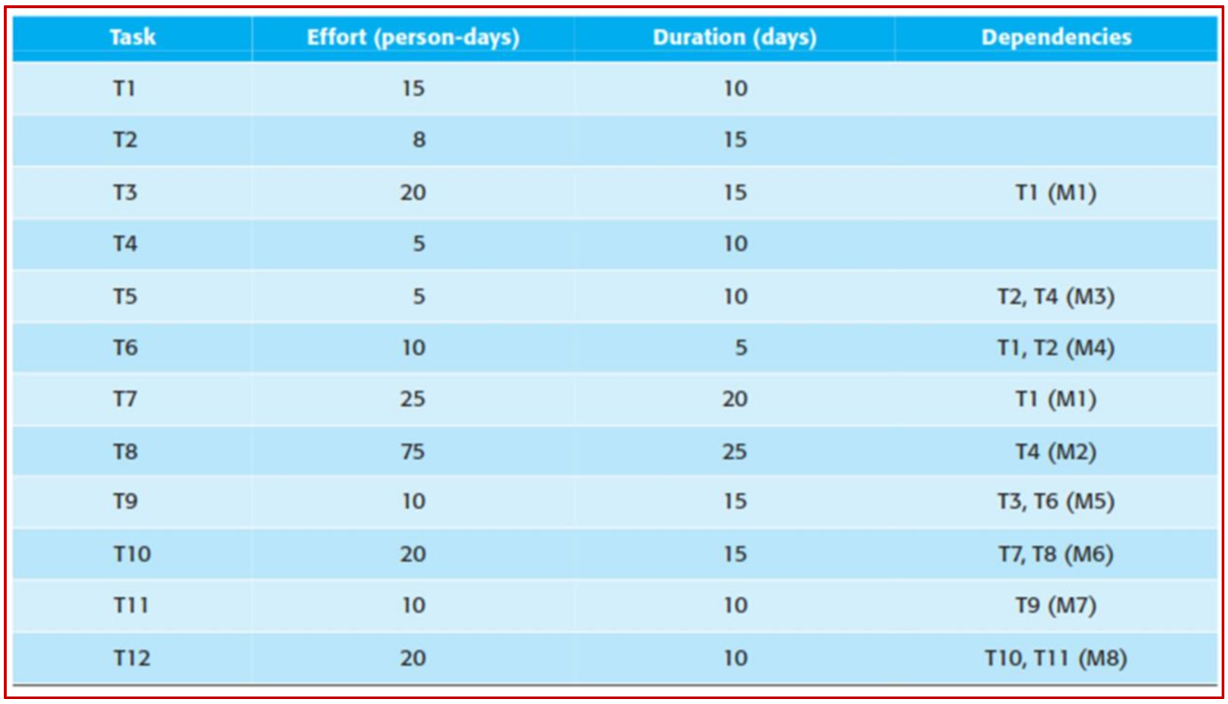
Schedule representation

- Graphical notations are normally used to illustrate the project schedule.
- These show the project breakdown into tasks. Tasks should not be too small. They should take about a week or two.
- Bar charts are the most commonly used representation for project schedules. They show the schedule as activities or resources against time.
- Project activities are the basic planning element. Each activity has:
  - Duration in calendar days or months.
  - An effort estimate, which reflects the number of person-days or person-months to complete the work.
  - A deadline by which the activity should be completed.
  - A defined endpoint. This represents the tangible result of completing the activity. This could be a document, the holding of a review meeting, the successful execution of all tests, etc.

Example:

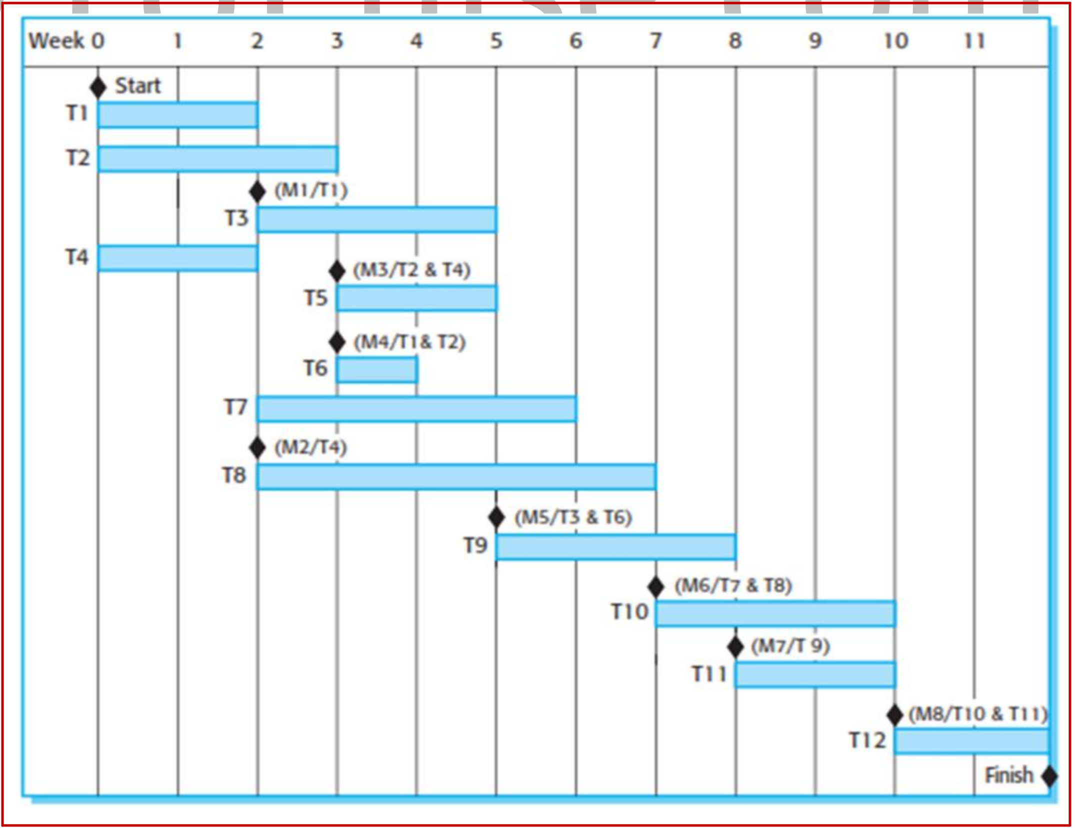
- Milestone M1 is associated with task T1 and milestone M3 is associated with a pair of tasks, T2 and T4.
- At task T3 is dependent on task T1. Task T1 must, therefore, be completed before T3 starts.
- For example, T1 might be the preparation of a component design and T3, the implementation of that design.

- Before implementation starts, the design should be complete.

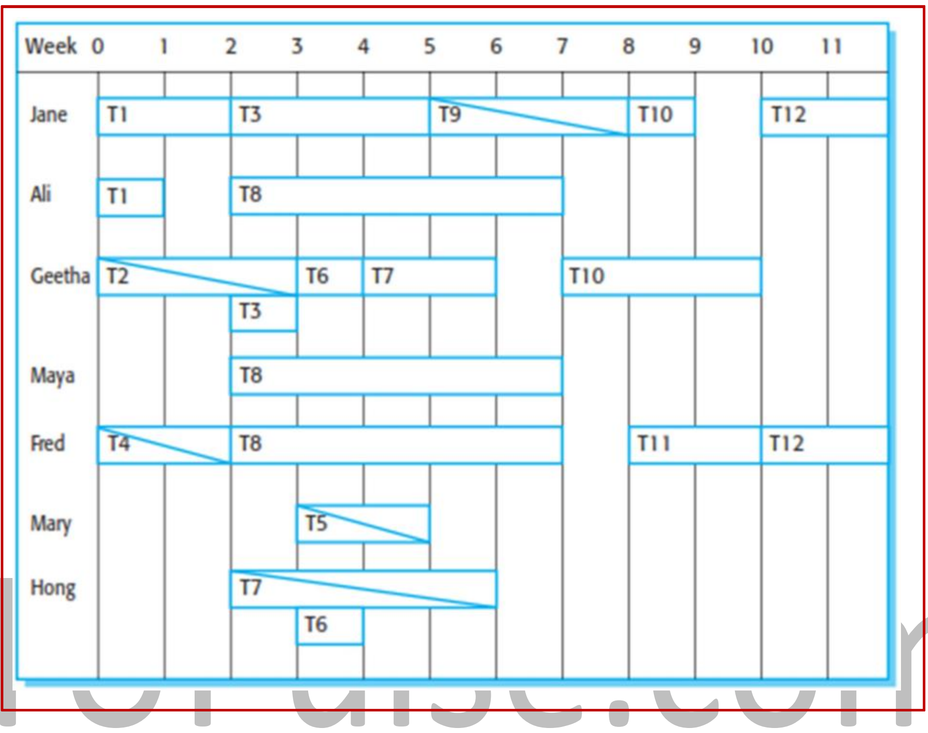


Task	Effort (person-days)	Duration (days)	Dependencies
T1	15	10	
T2	8	15	
T3	20	15	T1 (M1)
T4	5	10	
T5	5	10	T2, T4 (M3)
T6	10	5	T1, T2 (M4)
T7	25	20	T1 (M1)
T8	75	25	T4 (M2)
T9	10	15	T3, T6 (M5)
T10	20	15	T7, T8 (M6)
T11	10	10	T9 (M7)
T12	20	10	T10, T11 (M8)

- Bar chart showing a project calendar and the start and finish dates of tasks.
- Reading from left to right, the bar chart clearly shows when tasks start and end.
- The milestones (M1, M2, etc.) are also shown on the bar chart.
- Notice that tasks that are independent are carried out in parallel (e.g., tasks T1, T2, and T4 all start at the beginning of the project).



- If a task is delayed, this can obviously affect later tasks that are dependent on it.
- They cannot start until the delayed task is completed.
- Delays can cause serious problems with staff allocation, especially when people are working on several projects at the same time.
- If a task (T) is delayed, the people allocated may be assigned to other work (W).
- To complete this may take longer than the delay but, once assigned; they cannot simply be reassigned back to the original task, T. This may then lead to further delays in T as they complete W.



Estimation techniques

- Organizations need to make software effort and cost estimates. There are two types of technique that can be used to do this:
  - Experience-based techniques:
    - The estimate of future effort requirements is based on the manager's experience of past projects and the application domain. Essentially, the manager makes an informed judgment of what the effort requirements are likely to be.
  - Algorithmic cost modeling
    - In this approach, a formulaic approach is used to compute the project effort based on estimates of product attributes, such as size, and process characteristics, such as experience of staff involved.

Experience-based approaches

- Experience-based techniques rely on judgments based on experience of past projects and the effort expended in these projects on software development activities.
- Typically, you identify the deliverables to be produced in a project and the different software components or systems that are to be developed.
- You document these in a spreadsheet, estimate them individually and compute the total effort required.
- It usually helps to get a group of people involved in the effort estimation and to ask each member of the group to explain their estimate.

Algorithmic cost modelling

- Cost is estimated as a mathematical function of product, project and process attributes whose values are estimated by project managers:

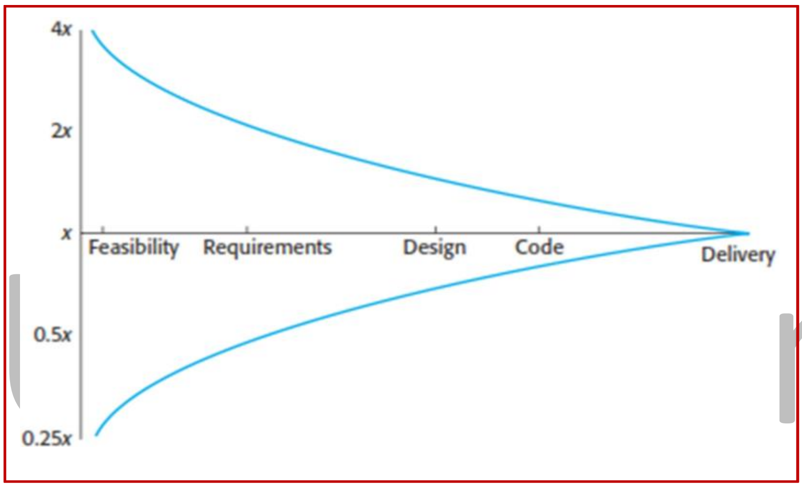
$\boxed{\text{Effort} = A \times \text{Size}^B \times M}$

- A is an organisation-dependent constant, B reflects the disproportionate effort for large projects and M is a multiplier reflecting product, process and people attributes.

- The most commonly used product attribute for cost estimation is code size.
- Most models are similar but they use different values for A, B and M.

Estimation accuracy

- The size of a software system can only be known accurately when it is finished.
- Several factors influence the final size
  - Use of COTS and components;
  - Programming language;
  - Distribution of system.
- As the development process progresses then the size estimate becomes more accurate.
- The estimates of the factors contributing to B and M are subjective and vary according to the judgment of the estimator.



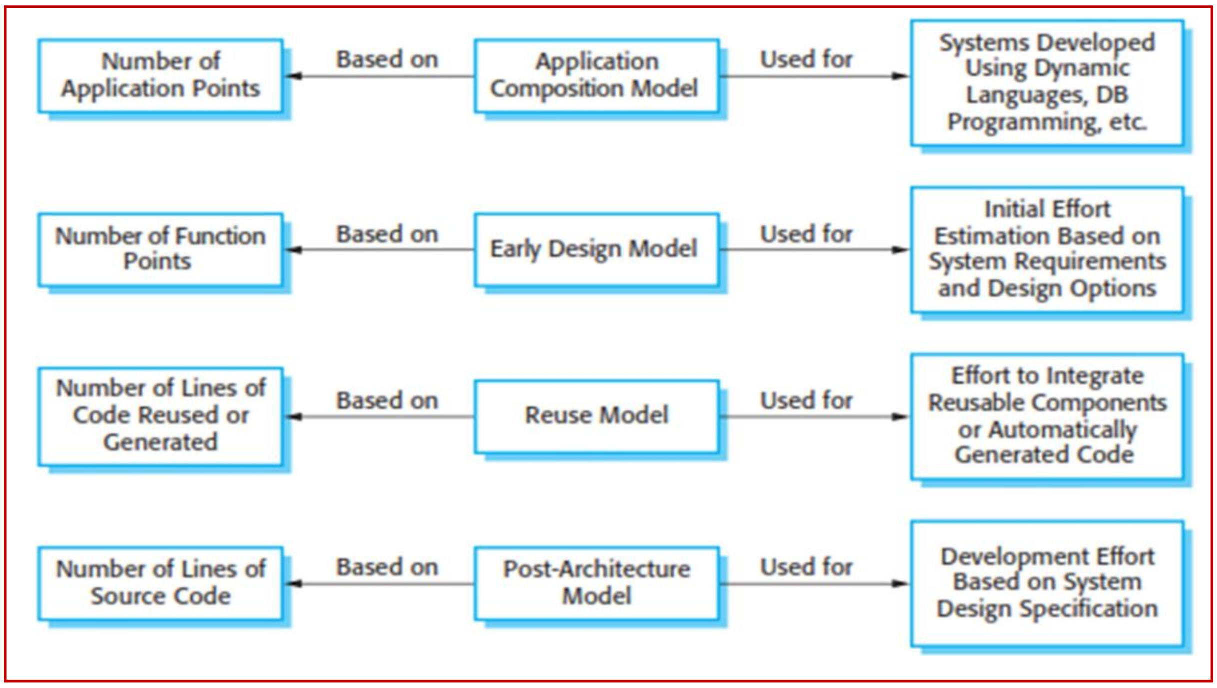
The COCOMO 2 model

- An empirical model based on project experience.
- Well-documented, 'independent' model which is not tied to a specific software vendor.
- Long history from initial version published in 1981 (COCOMO-81) through various instantiations to COCOMO 2.
- COCOMO 2 takes into account different approaches to software development, reuse, etc.

COCOMO 2 models

- COCOMO 2 incorporates a range of sub-models that produce increasingly detailed software estimates.
- The sub-models in COCOMO 2 are:
  - Application composition model.
    - Used when software is composed from existing parts.
  - Early design model.
    - Used when requirements are available but design has not yet started.
  - Reuse model.
    - Used to compute the effort of integrating reusable components.
  - Post-architecture model.
    - Used once the system architecture has been designed and more information about the system is available.

COCOMO estimation models

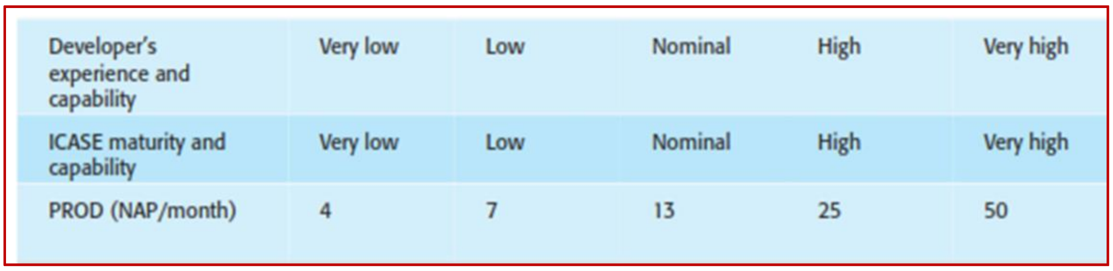


Number of Application Points
Based on
Application Composition Model
Used for
Systems Developed Using Dynamic Languages, DB Programming, etc.
Number of Function Points
Based on
Early Design Model
Used for
Initial Effort Estimation Based on System Requirements and Design Options
Number of Lines of Code Reused or Generated
Based on
Reuse Model
Used for
Effort to Integrate Reusable Components or Automatically Generated Code
Number of Lines of Source Code
Based on
Post-Architecture Model
Used for
Development Effort Based on System Design Specification

Application composition model

- □ Supports prototyping projects and projects where there is extensive reuse.
- □ Based on standard estimates of developer productivity in application (object) points/month.
- □ Takes CASE tool use into account.
- □ Formula is
  - PM = (NAP ´ (1 - %reuse/100)) / PROD
  - PM is the effort in person-months, NAP is the number of application points and PROD is the productivity.

Application-point productivity



Developer's experience and capability	Very low	Low	Nominal	High	Very high
ICASE maturity and capability	Very low	Low	Nominal	High	Very high
PROD (NAP/month)	4	7	13	25	50

Early design model

- □ Estimates can be made after the requirements have been agreed.
- □ Based on a standard formula for algorithmic models
  - □ PM = A ¯ Size^B ¯ M where
  - □ M = PERS ¯ RCPX ¯ RUSE ¯ PDIF ¯ PREX ¯ FCIL ¯ SCED;
  - □ A = 2.94 in initial calibration, Size in KLOC, B varies from 1.1 to 1.24 depending on novelty of the project, development flexibility, risk management approaches and the process maturity.
- □ Multipliers reflect the capability of the developers, the non-functional requirements, the familiarity with the development platform, etc.

- RCPX - product reliability and complexity;
- RUSE - the reuse required;
- PDIF - platform difficulty;
- PREX - personnel experience;
- PERS - personnel capability;
- SCED - required schedule;
- FCIL - the team support facilities.

The reuse model

- Takes into account black-box code that is reused without change and code that has to be adapted to integrate it with new code.
- There are two versions:
  - Black-box reuse where code is not modified. An effort estimate (PM) is computed.
  - White-box reuse where code is modified. A size estimate equivalent to the number of lines of new source code is computed. This then adjusts the size estimate for new code.

Reuse model estimates 1

- For generated code:
  - PM = (ASLOC * AT/100)/ATPROD
  - ASLOC is the number of lines of generated code
  - AT is the percentage of code automatically generated.
  - ATPROD is the productivity of engineers in integrating this code.

Reuse model estimates 2

- When code has to be understood and integrated:
  - ESLOC = ASLOC * (1-AT/100) * AAM.
  - ASLOC and AT as before.
  - AAM is the adaptation adjustment multiplier computed from the costs of changing the reused code, the costs of understanding how to integrate the code and the costs of reuse decision making.

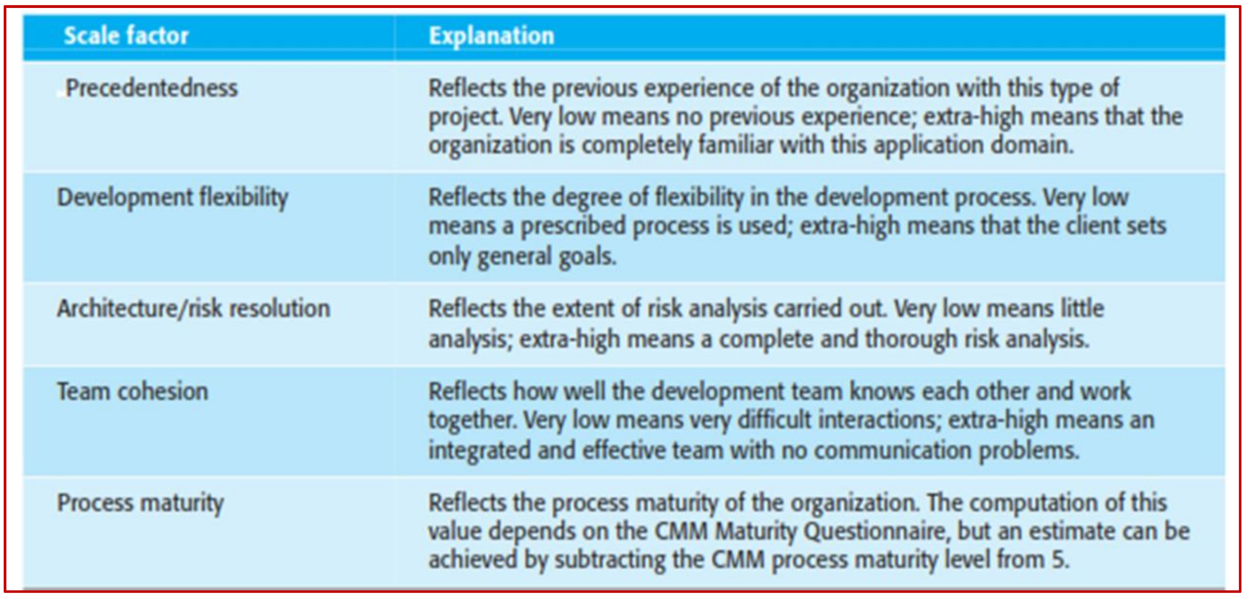
Post-architecture level

- Uses the same formula as the early design model but with 17 rather than 7 associated multipliers.
- The code size is estimated as:
  - Number of lines of new code to be developed;
  - Estimate of equivalent number of lines of new code computed using the reuse model;
  - An estimate of the number of lines of code that have to be modified according to requirements changes.

The exponent term

- This depends on 5 scale factors (see next slide). Their sum/100 is added to 1.01
- A company takes on a project in a new domain. The client has not defined the process to be used and has not allowed time for risk analysis. The company has a CMM level 2 rating.
  - Precedenteness - new project (4)
  - Development flexibility - no client involvement - Very high (1)
  - Architecture/risk resolution - No risk analysis - V. Low .(5)
  - Team cohesion - new team - nominal (3)
  - Process maturity - some control - nominal (3)
- Scale factor is therefore 1.17.

Scale factors used in the exponent computation in the post-architecture model

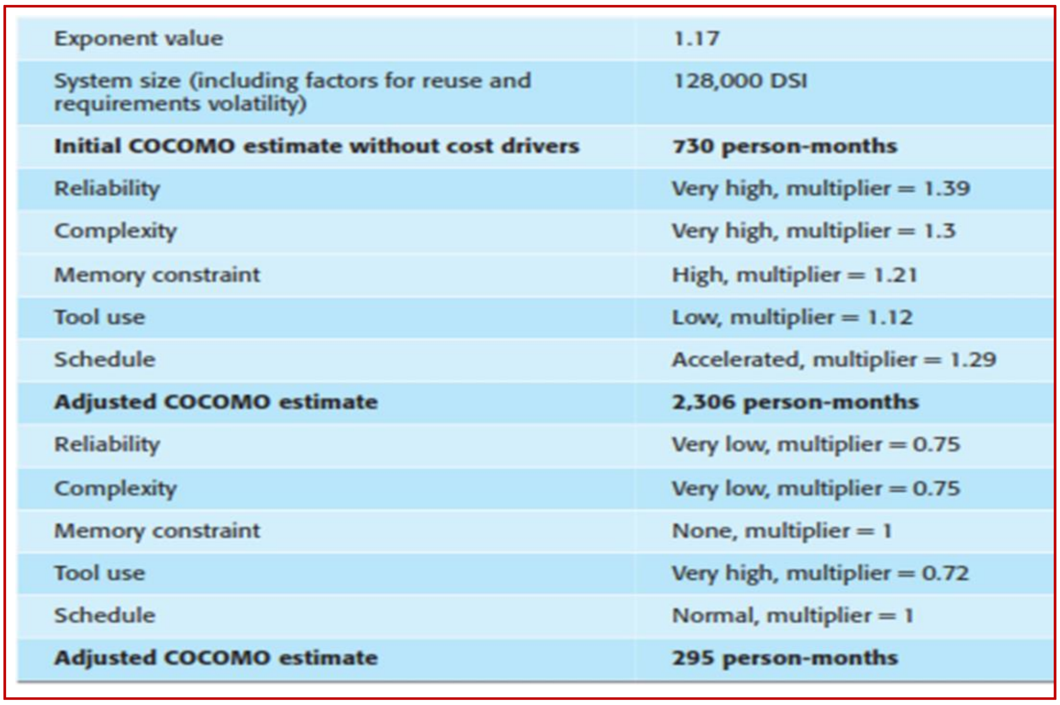


Scale factor	Explanation
Precedentedness	Reflects the previous experience of the organization with this type of project. Very low means no previous experience; extra-high means that the organization is completely familiar with this application domain.
Development flexibility	Reflects the degree of flexibility in the development process. Very low means a prescribed process is used; extra-high means that the client sets only general goals.
Architecture/risk resolution	Reflects the extent of risk analysis carried out. Very low means little analysis; extra-high means a complete and thorough risk analysis.
Team cohesion	Reflects how well the development team knows each other and work together. Very low means very difficult interactions; extra-high means an integrated and effective team with no communication problems.
Process maturity	Reflects the process maturity of the organization. The computation of this value depends on the CMM Maturity Questionnaire, but an estimate can be achieved by subtracting the CMM process maturity level from 5.

Multipliers

- Product attributes
  - Concerned with required characteristics of the software product being developed.
- Computer attributes
  - Constraints imposed on the software by the hardware platform.
- Personnel attributes
  - Multipliers that take the experience and capabilities of the people working on the project into account.
- Project attributes
  - Concerned with the particular characteristics of the software development project.

The effect of cost drivers on effort estimates



Exponent value	1.17
System size (including factors for reuse and requirements volatility)	128,000 DSI
Initial COCOMO estimate without cost drivers	**730 person-months**
Reliability	Very high, multiplier = 1.39
Complexity	Very high, multiplier = 1.3
Memory constraint	High, multiplier = 1.21
Tool use	Low, multiplier = 1.12
Schedule	Accelerated, multiplier = 1.29
Adjusted COCOMO estimate	**2,306 person-months**
Reliability	Very low, multiplier = 0.75
Complexity	Very low, multiplier = 0.75
Memory constraint	None, multiplier = 1
Tool use	Very high, multiplier = 0.72
Schedule	Normal, multiplier = 1
Adjusted COCOMO estimate	**295 person-months**

Project duration and staffing

- As well as effort estimation, managers must estimate the calendar time required to complete a project and when staff will be required.
- Calendar time can be estimated using a COCOMO 2 formula

$$\boxed{TDEV = 3 \times (PM)^{(0.33 + 0.2*(B - 1.01))}}$$

  - PM is the effort computation and B is the exponent computed as discussed above (B is 1 for the early prototyping model). This computation predicts the nominal schedule for the project.
- The time required is independent of the number of people working on the project.

Staffing requirements

- Staff required can't be computed by diving the development time by the required schedule.
- The number of people working on a project varies depending on the phase of the project.
- The more people who work on the project, the more total effort is usually required.
- A very rapid build-up of people often correlates with schedule slippage.

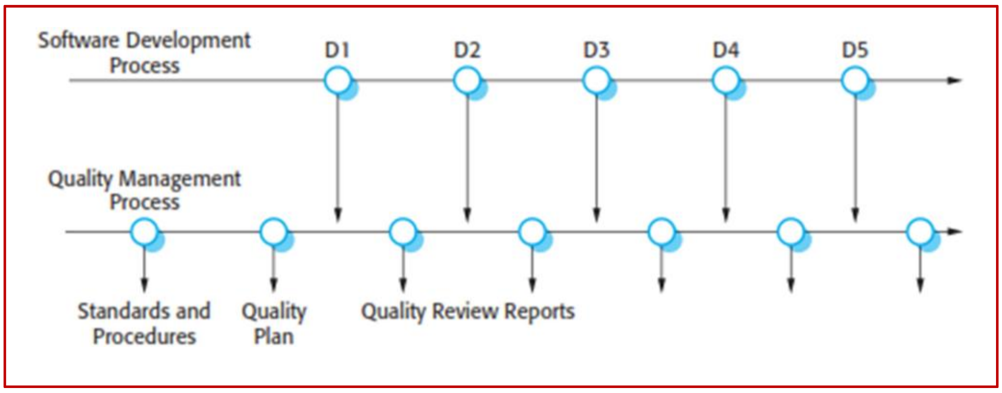
Quality Management

Software quality management

- Concerned with ensuring that the required level of quality is achieved in a software product.
- Three principal concerns:
  - At the organizational level, quality management is concerned with establishing a framework of organizational processes and standards that will lead to high-quality software.
  - At the project level, quality management involves the application of specific quality processes and checking that these planned processes have been followed.
  - At the project level, quality management is also concerned with establishing a quality plan for a project. The quality plan should set out the quality goals for the project and define what processes and standards are to be used.

Quality management activities

- Quality management provides an independent check on the software development process.
- The quality management process checks the project deliverables to ensure that they are consistent with organizational standards and goals
- The quality team should be independent from the development team so that they can take an objective view of the software. This allows them to report on software quality without being influenced by software development issues.



Quality planning

- A quality plan sets out the desired product qualities and how these are assessed and defines the most significant quality attributes.
- The quality plan should define the quality assessment process.

- It should set out which organisational standards should be applied and, where necessary, define new standards to be used.

Quality plans

- Quality plan structure
    - Product introduction;
    - Product plans;
    - Process descriptions;
    - Quality goals;
    - Risks and risk management.
- Quality plans should be short, succinct documents
    - If they are too long, no-one will read them.

Scope of quality management

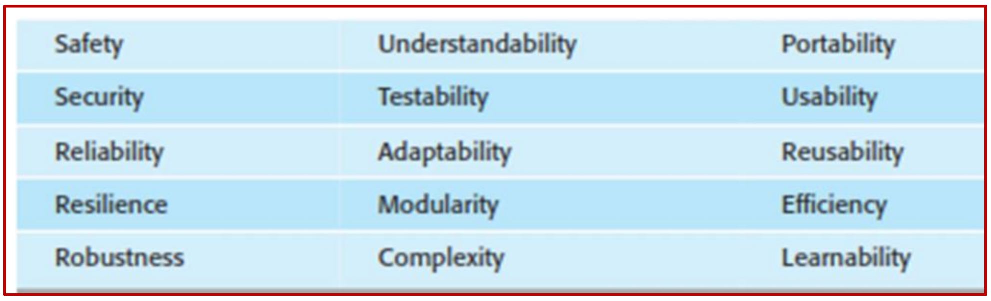
- Quality management is particularly important for large, complex systems. The quality documentation is a record of progress and supports continuity of development as the development team changes.
- For smaller systems, quality management needs less documentation and should focus on establishing a quality culture.

Software quality

- Quality, simplistically, means that a product should meet its specification.
- This is problematical for software systems
    - There is a tension between customer quality requirements (efficiency, reliability, etc.) and developer quality requirements (maintainability, reusability, etc.);
    - Some quality requirements are difficult to specify in an unambiguous way;
    - Software specifications are usually incomplete and often inconsistent.
- The focus may be 'fitness for purpose' rather than specification conformance.

Software fitness for purpose

- Have programming and documentation standards been followed in the development process?
- Has the software been properly tested?
- Is the software sufficiently dependable to be put into use?
- Is the performance of the software acceptable for normal use?
- Is the software usable?
- Is the software well-structured and understandable?



Safety	Understandability	Portability
Security	Testability	Usability
Reliability	Adaptability	Reusability
Resilience	Modularity	Efficiency
Robustness	Complexity	Learnability

Quality conflicts

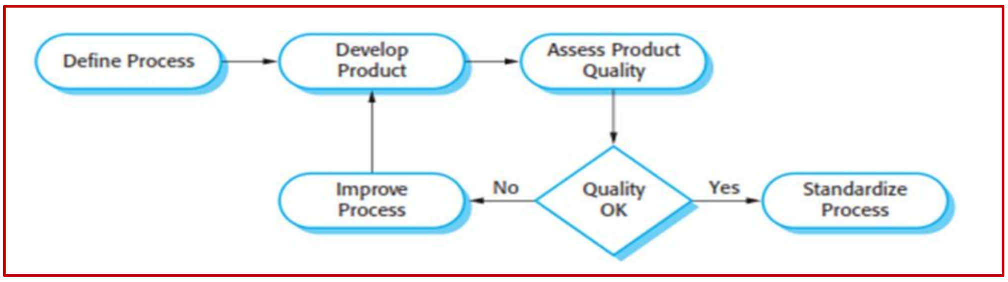
- It is not possible for any system to be optimized for all of these attributes – for example, improving robustness may lead to loss of performance.
- The quality plan should therefore define the most important quality attributes for the software that is being developed.

- The plan should also include a definition of the quality assessment process, an agreed way of assessing whether some quality, such as maintainability or robustness, is present in the product.

Process and product quality

- The quality of a developed product is influenced by the quality of the production process.
- This is important in software development as some product quality attributes are hard to assess.
- However, there is a very complex and poorly understood relationship between software processes and product quality.
  - The application of individual skills and experience is particularly important in software development;
  - External factors such as the novelty of an application or the need for an accelerated development schedule may impair product quality.

Process-based quality



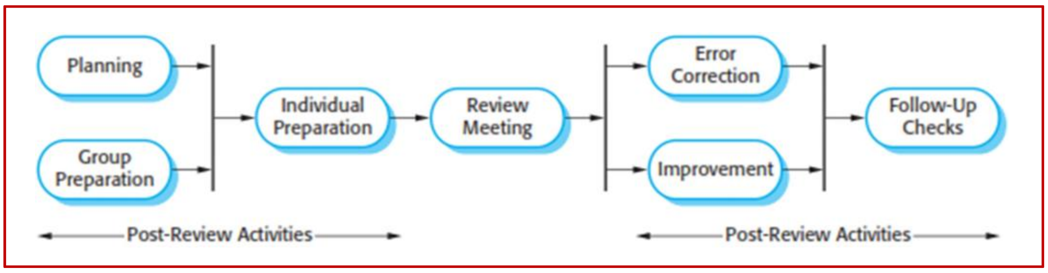
Reviews and inspections

- A group examines part or all of a process or system and its documentation to find potential problems.
- Software or documents may be 'signed off' at a review which signifies that progress to the next development stage has been approved by management.
- There are different types of review with different objectives
  - Inspections for defect removal (product);
  - Reviews for progress assessment (product and process);
  - Quality reviews (product and standards).

Quality reviews

- A group of people carefully examine part or all of a software system and its associated documentation.
- Code, designs, specifications, test plans, standards, etc. can all be reviewed.
- Software or documents may be 'signed off' at a review which signifies that progress to the next development stage has been approved by management.

The software review process



Pre-review activities:

- Pre-review activities are concerned with review planning and review preparation.
- Review planning involves setting up a review team, arranging a time and place for the review, and distributing the documents to be reviewed.
- During review preparation, the team may meet to get an overview of the software to be reviewed.
- Individual review team members read and understand the software or documents and relevant standards.

- They work independently to find errors, omissions, and departures from standards.

The review meeting

- During the review meeting, an author of the document or program being reviewed should ‘walk through’ the document with the review team.

Post-review activities

- After a review meeting has finished, the issues and problems raised during the review must be addressed.
- This may involve fixing software bugs, refactoring software so that it conforms to quality standards, or rewriting documents.

Reviews and agile methods

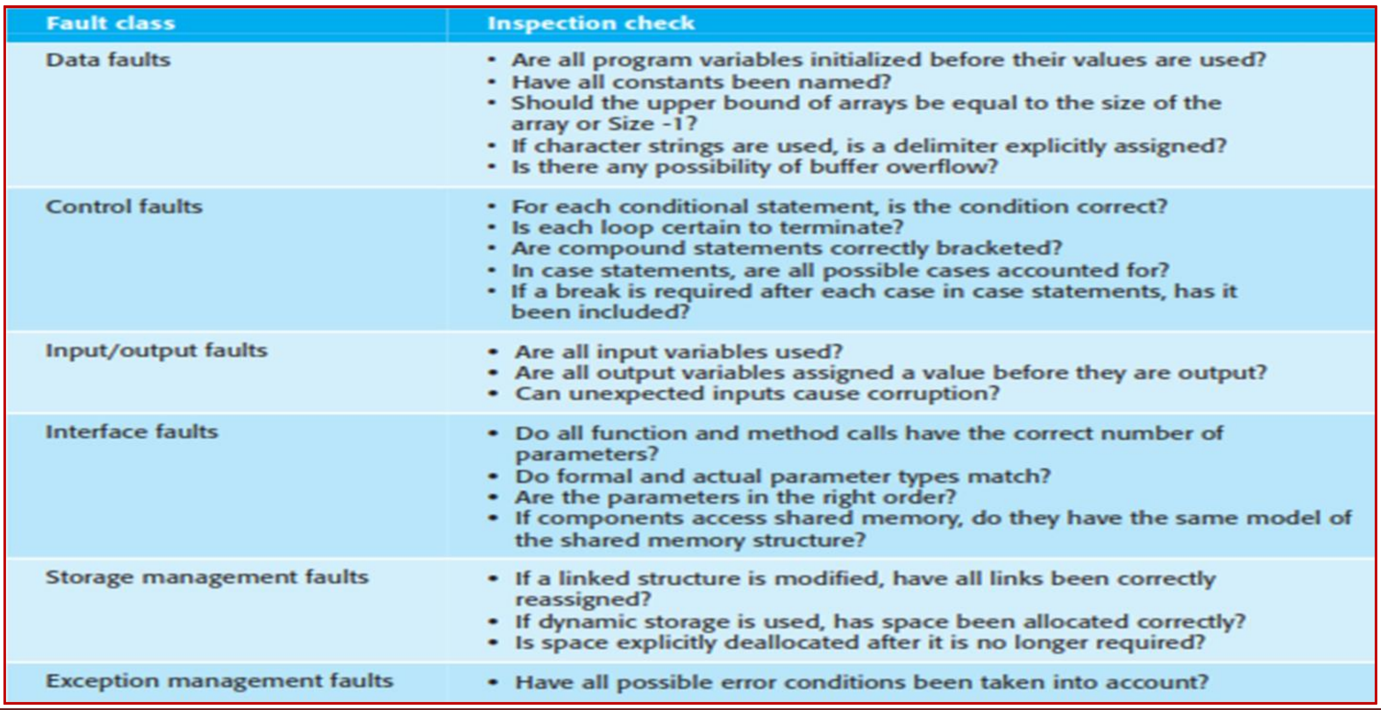
- The review process in agile software development is usually informal.
    - In Scrum, for example, there is a review meeting after each iteration of the software has been completed (a sprint review), where quality issues and problems may be discussed.
- In extreme programming, pair programming ensures that code is constantly being examined and reviewed by another team member.
- XP relies on individuals taking the initiative to improve and refactor code. Agile approaches are not usually standards-driven, so issues of standards compliance are not usually considered.

Program inspections

- These are peer reviews where engineers examine the source of a system with the aim of discovering anomalies and defects.
- Inspections do not require execution of a system so may be used before implementation.
- They may be applied to any representation of the system (requirements, design, configuration data, test data, etc.).
- They have been shown to be an effective technique for discovering program errors.

Inspection checklists

- Checklist of common errors should be used to drive the inspection.
- Error checklists are programming language dependent and reflect the characteristic errors that are likely to arise in the language.
- In general, the 'weaker' the type checking, the larger the checklist.
- Examples: Initialisation, Constant naming, loop termination, array bounds, etc.



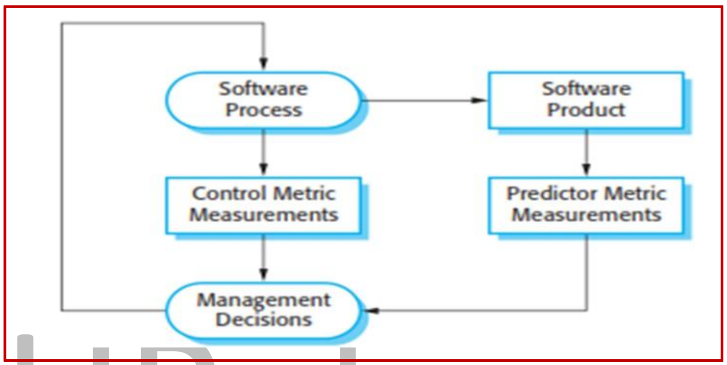
Fault class	Inspection check
Data faults	• Are all program variables initialized before their values are used? • Have all constants been named? • Should the upper bound of arrays be equal to the size of the array or Size -1? • If character strings are used, is a delimiter explicitly assigned? • Is there any possibility of buffer overflow?
Control faults	• For each conditional statement, is the condition correct? • Is each loop certain to terminate? • Are compound statements correctly bracketed? • In case statements, are all possible cases accounted for? • If a break is required after each case in case statements, has it been included?
Input/output faults	• Are all input variables used? • Are all output variables assigned a value before they are output? • Can unexpected inputs cause corruption?
Interface faults	• Do all function and method calls have the correct number of parameters? • Do formal and actual parameter types match? • Are the parameters in the right order? • If components access shared memory, do they have the same model of the shared memory structure?
Storage management faults	• If a linked structure is modified, have all links been correctly reassigned? • If dynamic storage is used, has space been allocated correctly? • Is space explicitly deallocated after it is no longer required?
Exception management faults	• Have all possible error conditions been taken into account?

Software measurement and metrics

- Software measurement is concerned with deriving a numeric value for an attribute of a software product or process.
- This allows for objective comparisons between techniques and processes.
- Although some companies have introduced measurement programmes, most organisations still don’t make systematic use of software measurement.
- There are few established standards in this area.

Software metric

- Any type of measurement which relates to a software system, process or related documentation
  - Lines of code in a program, the Fog index, number of person-days required to develop a component.
- Allow the software and the software process to be quantified.
- May be used to predict product attributes or to control the software process.
- Product metrics can be used for general predictions or to identify anomalous components.



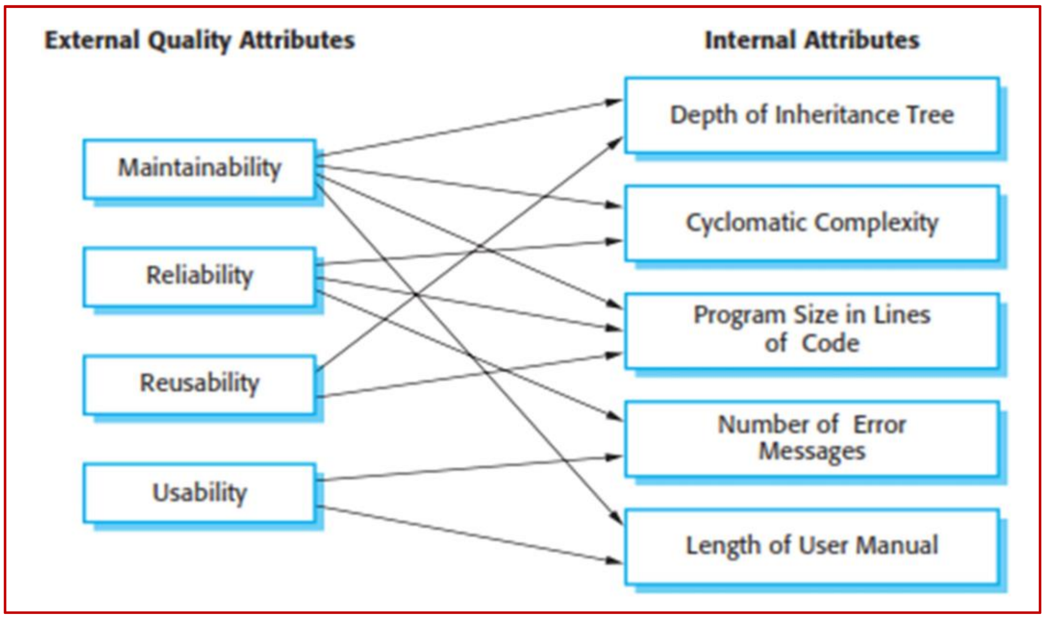
Use of measurements

- To assign a value to system quality attributes
  - By measuring the characteristics of system components, such as their cyclomatic complexity, and then aggregating these measurements, you can assess system quality attributes, such as maintainability.
- To identify the system components whose quality is sub-standard
  - Measurements can identify individual components with characteristics that deviate from the norm. For example, you can measure components to discover those with the highest complexity. These are most likely to contain bugs because the complexity makes them harder to understand.

Metrics assumptions

- A software property can be measured.
- The relationship exists between what we can measure and what we want to know. We can only measure internal attributes but are often more interested in external software attributes.
- This relationship has been formalised and validated.
- It may be difficult to relate what can be measured to desirable external quality attributes.

Relationships between internal and external software



Problems with measurement in industry

- It is impossible to quantify the return on investment of introducing an organizational metrics program.
- There are no standards for software metrics or standardized processes for measurement and analysis.
- In many companies, software processes are not standardized and are poorly defined and controlled.
- Most work on software measurement has focused on code-based metrics and plan-driven development processes. However, more and more software is now developed by configuring ERP systems or COTS.
- Introducing measurement adds additional overhead to processes.

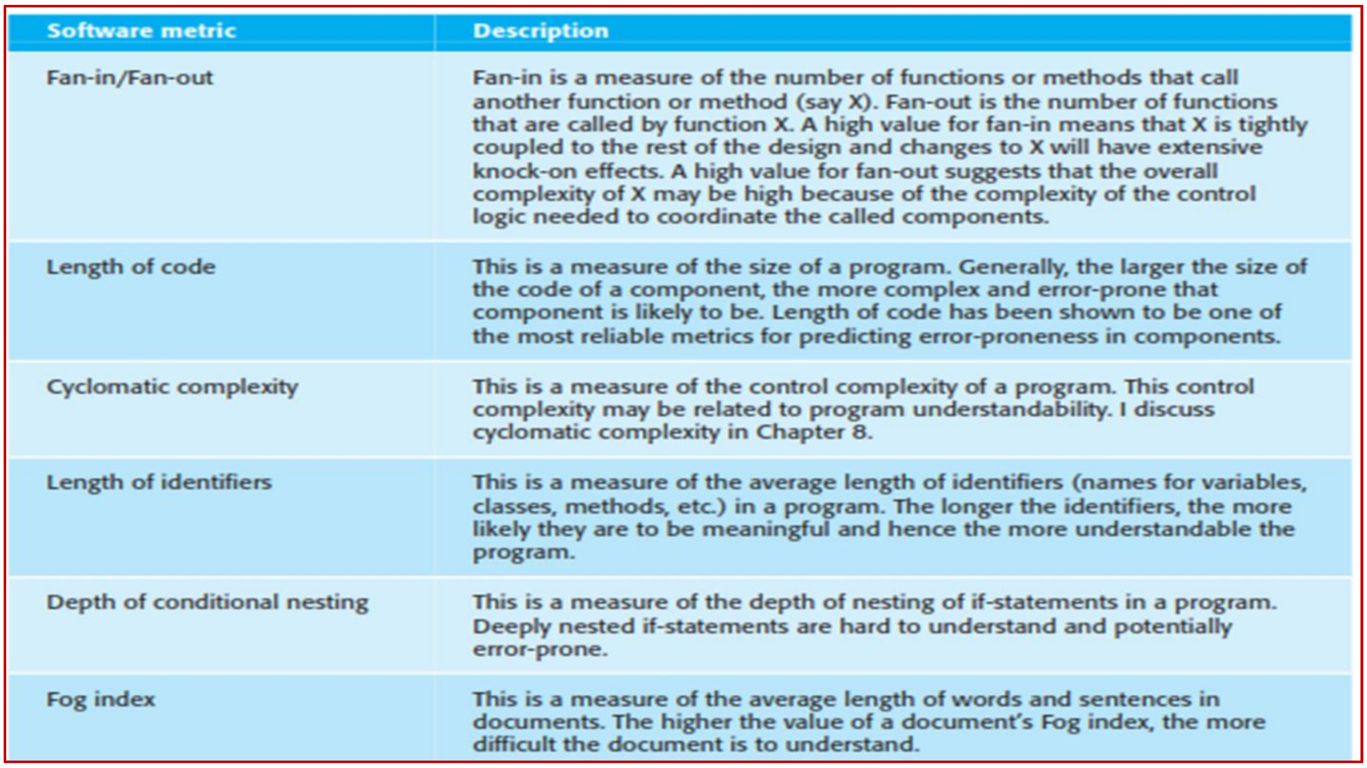
Product metrics

- A quality metric should be a predictor of product quality.
- Classes of product metric
  - Dynamic metrics which are collected by measurements made of a program in execution;
  - Static metrics which are collected by measurements made of the system representations;
  - Dynamic metrics help assess efficiency and reliability
  - Static metrics help assess complexity, understandability and maintainability.

Dynamic and static metrics

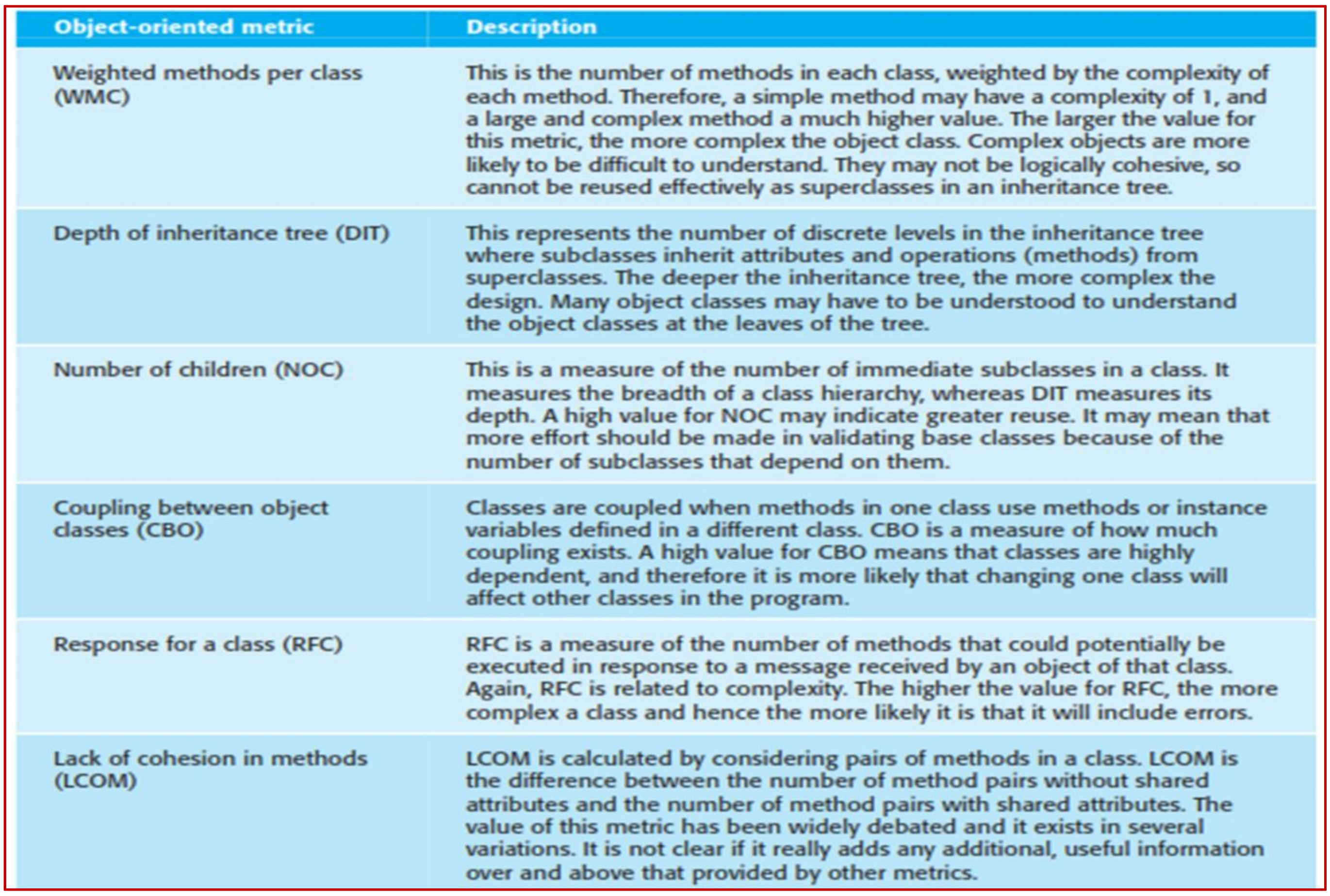
- Dynamic metrics are closely related to software quality attributes
  - It is relatively easy to measure the response time of a system (performance attribute) or the number of failures (reliability attribute).
- Static metrics have an indirect relationship with quality attributes
  - You need to try and derive a relationship between these metrics and properties such as complexity, understandability and maintainability.

Static software product metrics



Software metric	Description
Fan-in/Fan-out	Fan-in is a measure of the number of functions or methods that call another function or method (say X). Fan-out is the number of functions that are called by function X. A high value for fan-in means that X is tightly coupled to the rest of the design and changes to X will have extensive knock-on effects. A high value for fan-out suggests that the overall complexity of X may be high because of the complexity of the control logic needed to coordinate the called components.
Length of code	This is a measure of the size of a program. Generally, the larger the size of the code of a component, the more complex and error-prone that component is likely to be. Length of code has been shown to be one of the most reliable metrics for predicting error-proneness in components.
Cyclomatic complexity	This is a measure of the control complexity of a program. This control complexity may be related to program understandability. I discuss cyclomatic complexity in Chapter 8.
Length of identifiers	This is a measure of the average length of identifiers (names for variables, classes, methods, etc.) in a program. The longer the identifiers, the more likely they are to be meaningful and hence the more understandable the program.
Depth of conditional nesting	This is a measure of the depth of nesting of if-statements in a program. Deeply nested if-statements are hard to understand and potentially error-prone.
Fog index	This is a measure of the average length of words and sentences in documents. The higher the value of a document's Fog index, the more difficult the document is to understand.

The CK object-oriented metrics suite

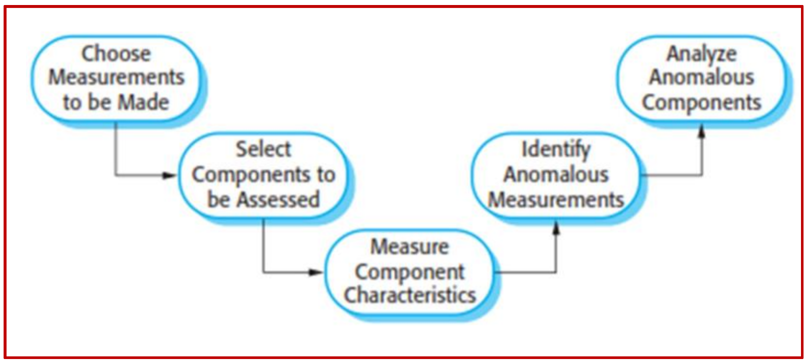


Object-oriented metric	Description
Weighted methods per class (WMC)	This is the number of methods in each class, weighted by the complexity of each method. Therefore, a simple method may have a complexity of 1, and a large and complex method a much higher value. The larger the value for this metric, the more complex the object class. Complex objects are more likely to be difficult to understand. They may not be logically cohesive, so cannot be reused effectively as superclasses in an inheritance tree.
Depth of inheritance tree (DIT)	This represents the number of discrete levels in the inheritance tree where subclasses inherit attributes and operations (methods) from superclasses. The deeper the inheritance tree, the more complex the design. Many object classes may have to be understood to understand the object classes at the leaves of the tree.
Number of children (NOC)	This is a measure of the number of immediate subclasses in a class. It measures the breadth of a class hierarchy, whereas DIT measures its depth. A high value for NOC may indicate greater reuse. It may mean that more effort should be made in validating base classes because of the number of subclasses that depend on them.
Coupling between object classes (CBO)	Classes are coupled when methods in one class use methods or instance variables defined in a different class. CBO is a measure of how much coupling exists. A high value for CBO means that classes are highly dependent, and therefore it is more likely that changing one class will affect other classes in the program.
Response for a class (RFC)	RFC is a measure of the number of methods that could potentially be executed in response to a message received by an object of that class. Again, RFC is related to complexity. The higher the value for RFC, the more complex a class and hence the more likely it is that it will include errors.
Lack of cohesion in methods (LCOM)	LCOM is calculated by considering pairs of methods in a class. LCOM is the difference between the number of method pairs without shared attributes and the number of method pairs with shared attributes. The value of this metric has been widely debated and it exists in several variations. It is not clear if it really adds any additional, useful information over and above that provided by other metrics.

Software component analysis

- System component can be analyzed separately using a range of metrics.
- The values of these metrics may then compared for different components and, perhaps, with historical measurement data collected on previous projects.
- Anomalous measurements, which deviate significantly from the norm, may imply that there are problems with the quality of these components.

The process of product measurement


Choose Measurements to be Made
Select Components to be Assessed
Measure Component Characteristics
Identify Anomalous Measurements
Analyze Anomalous Components

Choose measurements to be made:

- The questions that the measurement is intended to answer should be formulated and the measurements required to answer these questions defined. Measurements that are not directly relevant to these questions need not be collected

Select components to be assessed

- You may not need to assess metric values for all of the components in a software system. Sometimes, you can select a representative selection of components for measurement, allowing you to make an overall assessment of system quality. At other times, you may wish to focus on the core components of the system that are in almost constant use. The quality of these components is more important than the quality of components that are only rarely used.

Measure component characteristics

- The selected components are measured and the associated metric values computed. This normally involves processing the component representation (design, code, etc.) using an automated data collection tool.
- This tool may be specially written or may be a feature of design tools that are already in use.

Identify anomalous measurements

- After the component measurements have been made, you then compare them with each other and to previous measurements that have been recorded in a measurement database. You should look for unusually high or low values for each metric, as these suggest that there could be problems with the component exhibiting these values.

Analyse anomalous components

- When you have identified components that have anomalous values for your chosen metrics, you should examine them to decide whether or not these anomalous metric values mean that the quality of the component is compromised. An anomalous metric value for complexity (say) does not necessarily mean a poor quality component. There may be some other reason for the high value, so may not mean that there are component quality problems.

Measurement surprises

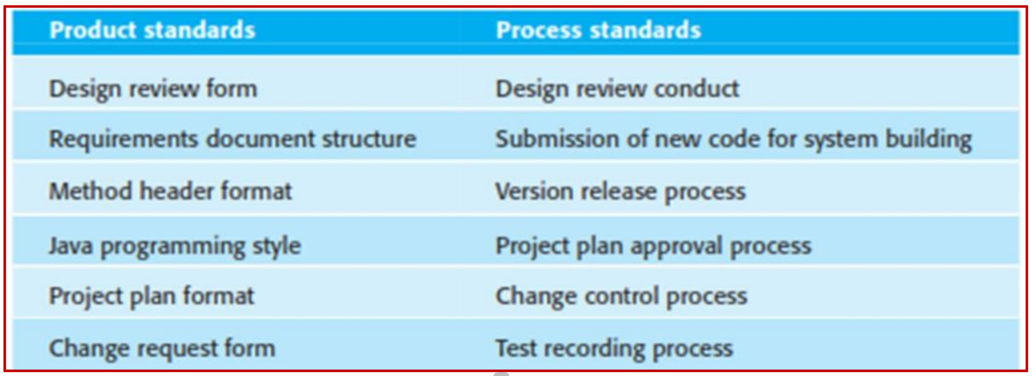
- Reducing the number of faults in a program leads to an increased number of help desk calls
  - The program is now thought of as more reliable and so has a wider more diverse market. The percentage of users who call the help desk may have decreased but the total may increase;
  - A more reliable system is used in a different way from a system where users work around the faults. This leads to more help desk calls.

Software standards

- Standards define the required attributes of a product or process. They play an important role in quality management.
- Standards may be international, national, organizational or project standards.
- Product standards define characteristics that all software components should exhibit e.g. a common programming style.
- Process standards define how the software process should be enacted.

Importance of standards

- Encapsulation of best practice- avoids repetition of past mistakes.
- They are a framework for defining what quality means in a particular setting i.e. that organization's view of quality.
- They provide continuity - new staff can understand the organisation by understanding the standards that are used.



Product standards	Process standards
Design review form	Design review conduct
Requirements document structure	Submission of new code for system building
Method header format	Version release process
Java programming style	Project plan approval process
Project plan format	Change control process
Change request form	Test recording process

Problems with standards

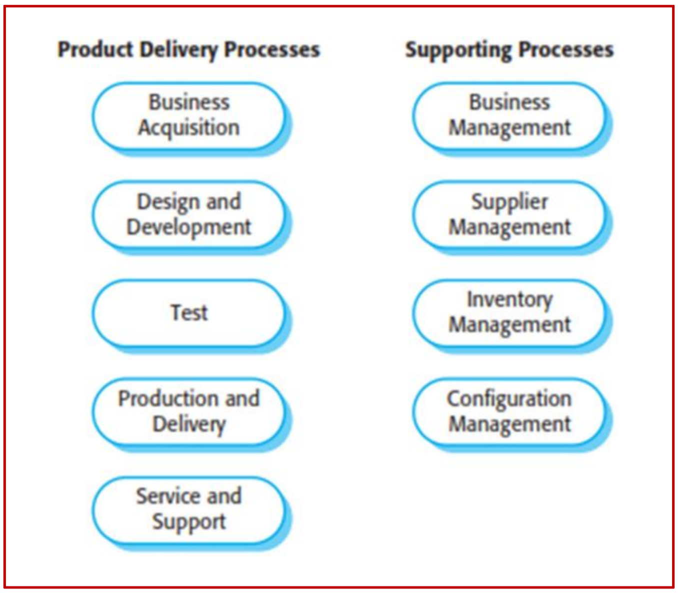
- They may not be seen as relevant and up-to-date by software engineers.
- They often involve too much bureaucratic form filling.
- If they are unsupported by software tools, tedious form filling work is often involved to maintain the documentation associated with the standards.

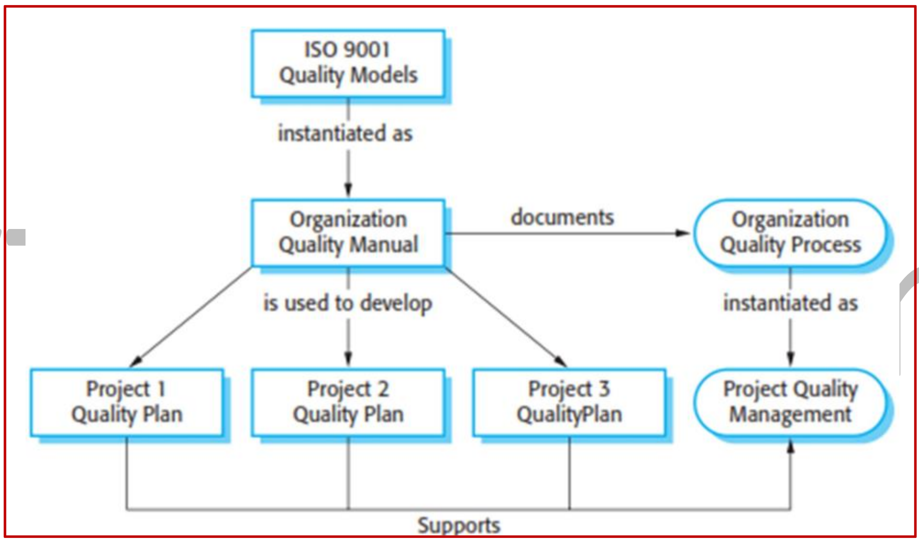
Standards development

- Involve practitioners in development. Engineers should understand the rationale underlying a standard.
- Review standards and their usage regularly. Standards can quickly become outdated and this reduces their credibility amongst practitioners.
- Detailed standards should have specialized tool support. Excessive clerical work is the most significant complaint against standards.
    - Web-based forms are not good enough.

ISO 9001 standards framework

- An international set of standards that can be used as a basis for developing quality management systems.
- ISO 9001, the most general of these standards, applies to organizations that design, develop and maintain products, including software.
- The ISO 9001 standard is a framework for developing software standards.
    - It sets out general quality principles, describes quality processes in general and lays out the organizational standards and procedures that should be defined. These should be documented in an organizational quality manual.





ISO 9001 certification

- Quality standards and procedures should be documented in an organisational quality manual.
- An external body may certify that an organisation's quality manual conforms to ISO 9000 standards.
- Some customers require suppliers to be ISO 9000 certified although the need for flexibility here is increasingly recognised.