Module 5

Project Planning

- **Software Pricing**
- **Plan Driven Development**
- **Project Scheduling**
- **Estimation Techniques**

****

Introduction

- Project Planning is one of the most important job of the project manager.
- Break down the work into parts and assign these to project 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 takes place at three stages in a project life cycle:**
  1. At the **proposal stage** when you are bidding for a contract to develop or provide a software system. You need a plan at this stage to help you decide if you have the resources to complete the work and to work out the price that you should quote to a customer.
  2. 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.
  3. **Periodically throughout the project**, when you modify your plan in light of experience gained and information from monitoring the progress of the work. You learn more about the system being implemented and capabilities of your development team. This information allows you to make more accurate estimates of how long the work will take.
- Planning at the proposal stage is inevitably speculative, as you do not usually have a complete set of requirements for the software to be developed
- Rather, you have to respond to a call for proposals based on a high-level description of the software functionality that is required
- If you win the contract, you then usually have to replan the project, taking into account changes since the proposal was made.
- When you are bidding for a contract, you have to work out the price that you will propose to the customer for developing the software.

- As a starting point for calculating this price, you need to draw up an estimate of your costs for completing the project work.
- Estimation involves working out how much effort is required to complete each activity and, from this, calculating the total cost of activities
- Many factors influence the pricing of a software project—it is not simply cost + profit.
- **There are three main parameters that you should use when computing the costs of a software development project:**

1. effort costs (the costs of paying software engineers and managers);
2. hardware and software costs, including maintenance;
3. travel and training costs.

Software Pricing

- In principle, the price of a software product to a customer is simply the cost of development plus profit for the developer.
- In practice, however, the relationship between the project cost and the price quoted to the customer is not usually so simple.
- When calculating a price, you should take broader organizational, economic, political, and business considerations into account, such as those shown in the next figure.

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.
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.

Fig: Factors affecting Software pricing

- You need to think about organizational concerns, the risks associated with the project, and the type of contract that will be used.
- These may cause the price to be adjusted upwards or downwards.
- Because of the organizational considerations involved, deciding on a project price should be a group activity involving marketing and sales staff, senior management, and project managers.

Plan-Driven Development

- Plan-driven or plan-based development is an approach to software engineering where the development process is planned in detail.
- A project plan is created that records the work to be done, who 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
- This contrasts with agile development, where many decisions affecting the development are delayed and made later, as required, during the development process

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.
- The plan should identify risks to the project and the software under development, and the approach that is taken to risk management
- Although the specific details of project plans vary depending on the type of project and organization, **plans normally include the following sections:**

1. **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
2. **Project organization**: This describes the way in which the development team is organized, the people involved, and their roles in the team.
3. **Risk analysis**: This describes possible project risks, the likelihood of these risks arising, and the risk reduction strategies that are proposed
4. **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.

5. **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.

6. **Project schedule**: This shows the dependencies between activities, the estimated time required to reach each milestone, and the allocation of people to activities.

7. **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.

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.

Fig: Project Plan Supplements

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 replanned
- The following is a UML activity diagram that shows a typical workflow for a project planning process

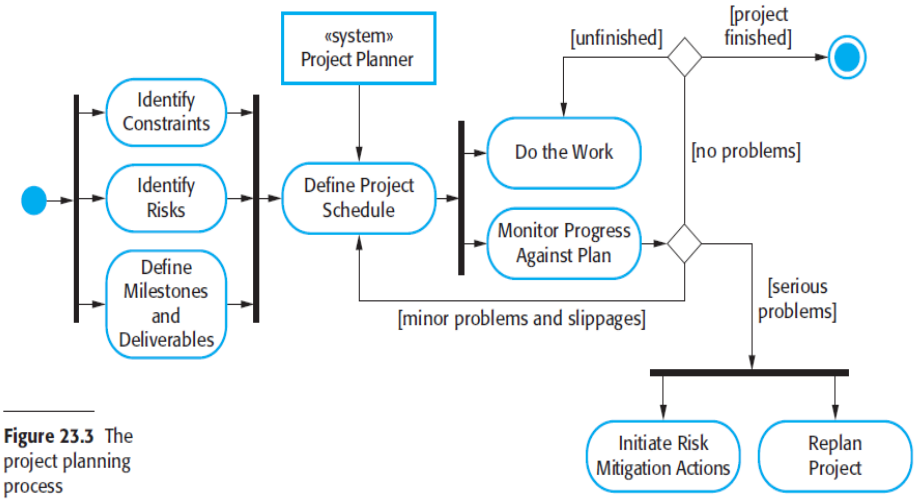

Figure 23.3 The project planning process

Fig: The Project Planning Process

- At the beginning of a planning process, you should assess the constraints affecting the project.
- The required delivery date, staff available, overall budget, available tools, and so on
- You should also identify the project 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).
- The process then enters a loop.
- You draw up an estimated schedule for the project and the activities defined in the schedule are initiated or given permission to continue.
- After some time (usually about two to three weeks), you should review progress and note discrepancies from the planned schedule
- It is important to be realistic when you are creating a project plan

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.
- Both plan-based and agile processes need an initial project schedule, although the level of detail may be less in an agile project plan.
- This initial schedule is used to plan how people will be allocated to projects and to check the progress of the project against its contractual commitments
- In traditional development processes, the complete schedule is initially developed and then modified as the project progresses.
- In agile processes, there has to be an overall schedule that identifies when the major phases of the project will be completed. An iterative approach to scheduling is then used to plan each phase
- Scheduling in plan-driven projects involves breaking down the total work involved in a project into separate tasks and estimating the time required to complete each task.
- Tasks should normally last at least a week, and no longer than 2 months

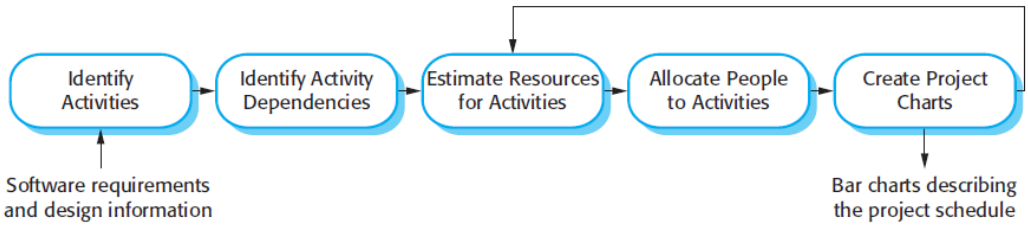

Fig: The Project Scheduling Process

Schedule representation

- Project schedules may simply be represented in a table or spreadsheet showing the tasks, effort, expected duration, and task dependencies

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)

Fig: Tasks, duration and dependencies

- However, this style of representation makes it difficult to see the relationships and dependencies between the different activities.
- For this reason, alternative graphical representations of project schedules have been developed that are often easier to read and understand.
- There are **two types of representation** that are commonly used:
    1. **Bar charts**, which are calendar-based, show who is responsible for each activity, the expected elapsed time, and when the activity is scheduled to begin and end. Bar charts are sometimes called 'Gantt charts', after their inventor, Henry Gantt
    2. **Activity networks**, which are network diagrams, show the dependencies between the different activities making up a project
- **Project activities are the basic planning element. Each activity has:**
    a. A duration in calendar days or months.
    b. An effort estimate, which reflects the number of person-days or person-months to complete the work.
    c. A deadline by which the activity should be completed.
    d. 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.
- When planning a project, you should also define milestones; that is, each stage in the project where a progress assessment can be made.
- Each milestone should be documented by a short report that summarizes the progress made and the work done

- A special kind of milestone is the production of a project deliverable.
- A deliverable is a work product that is delivered to the customer.
- It is the outcome of a significant project phase such as specification or design

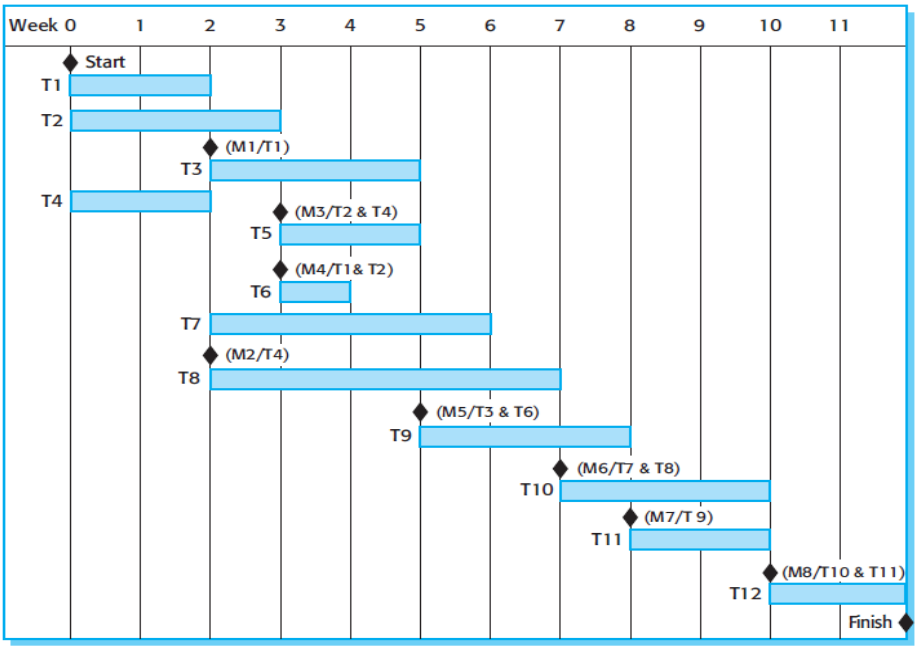

Fig: Activity bar Chart

- As well as planning the delivery schedule for the software, project managers have to allocate resources to tasks.
- The key resource is, of course, the software engineers who will do the work, and they have to be assigned to project activities.
- The resource allocation can also be input to project management tools and a bar chart generated, which shows when staff are working on the project
- People may be working on more than one task at the same time and, sometimes, they are not working on the project.
- They may be on holiday, working on other projects, attending training courses, or engaging in some other activity.
- We show part-time assignments using a diagonal line crossing the bar

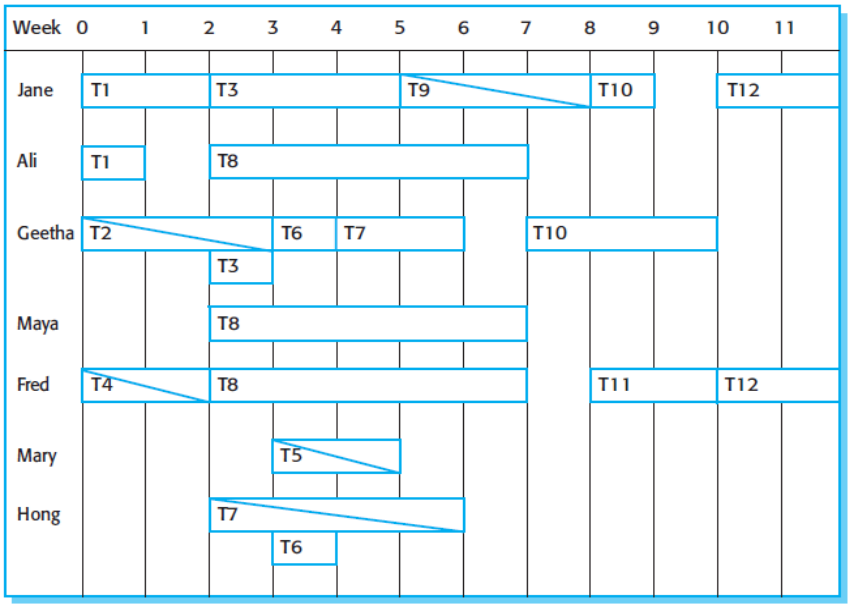

Fig: staff allocation chart

Estimation techniques

- Project schedule estimation is difficult
- You may have to make initial estimates on the basis of a high-level user requirements definition.
- The software may have to run on unfamiliar computers or use new development technology.
- The people involved in the project and their skills will probably not be known
- There are so many uncertainties that it is impossible to estimate system development costs accurately during the early stages of a project.
- **There are two types of technique that can be used to do this:**

1. ***Experience-based techniques*:** The estimate of future effort requirements is based on the manager's experience of past projects and the application domain
2. ***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.

Algorithmic Cost Modeling

- Algorithmic cost modeling uses a mathematical formula to predict project costs based on estimates of the project size; the type of software being developed; and other team, process, and product factors.
- An algorithmic cost model can be built by analyzing the costs and attributes of completed projects, and finding the closest-fit formula to actual experience.
- Algorithmic models for estimating effort in a software project are mostly based on a simple formula:
- **Effort = A x Size^{B} x M**
- A is a constant factor which depends on local organizational practices and the type of software that is developed
- Size may be either an assessment of the code size of the software or a functionality estimate expressed in function or application points
- The value of exponent B usually lies between 1 and 1.5.
- M is a multiplier made by combining process, product, and development attributes, such as the dependability requirements for the software and the experience of the development team.
- **All algorithmic models have similar problems:**
    1. It is often difficult to estimate Size at an early stage in a project, when only the specification is available
    2. The estimates of the factors contributing to B and M are subjective. Estimates vary from one person to another, depending on their background and experience of the type of system that is being developed
- Accurate code size estimation is difficult at an early stage in a project because the size of the final program depends on design decisions that may not have been made when the estimate is required
- The programming language used for system development also affects the number of lines of code to be developed. A language like Java might mean that more lines of code are necessary than if C (say) was used
- Algorithmic cost models are a systematic way to estimate the effort required to develop a system.

Quality Management

- **Software Quality**
- **Software Standards**
- **Reviews and Inspections**
- **Software Measurement and Metrics**

****

- Problems with software quality were initially discovered in the 1960s with the development of the first large software systems
- Delivered software was slow and unreliable, difficult to maintain and hard to reuse.
- Dissatisfaction with this situation led to the adoption of formal techniques of software quality management
- The quality management techniques, in conjunction with new software technologies and better software testing, have led to significant improvements in the general level of software quality
- Software quality management for software systems **has three principal concerns**:

1. **At the organizational level,** quality management is concerned with establishing a framework of organizational processes and standards that will lead to high quality software. This means that the quality management team should take responsibility for defining the software development processes to be used and standards that should apply to the software and related documentation, including the system requirements, design, and code.
2. **At the project level**, quality management involves the application of specific quality processes, checking that these planned processes have been followed, and ensuring that the project outputs are conformant with the standards that are applicable to that project.
3. **Quality management at the project level** 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.

- The terms **'quality assurance' and 'quality control'** are widely used in manufacturing industry.
- **Quality assurance (QA)** is the definition of processes and standards that should lead to high-quality products and the introduction of quality processes into the manufacturing process.
- **Quality control (QC)** is the application of these quality processes to weed out products that are not of the required level of quality.
- The QA team in most companies is responsible for managing the release testing process.
- This means that they manage the testing of the software before it is released to customers.
- They are responsible for checking that the system tests provide coverage of the requirements and that proper records are maintained of the testing process.

- 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

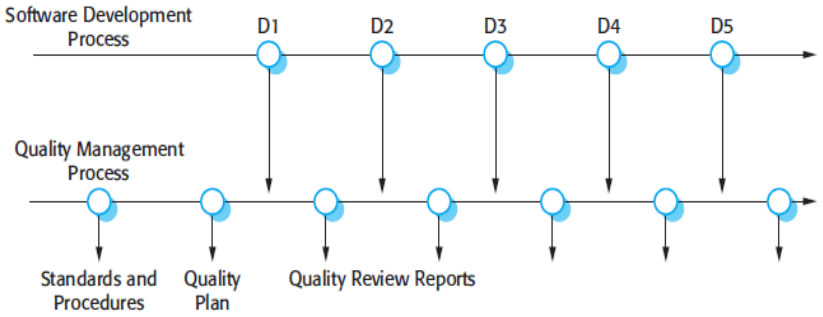

Fig: Quality management and software development

- The QA 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.
- Ideally, the quality management team should not be associated with any particular development group, but should rather have organization-wide responsibility for quality management.
- They should be independent and report to management above the project manager level.
- The reason for this is that project managers have to maintain the project budget and schedule. If problems arise, they may be tempted to compromise on product quality so that they meet their schedule

Quality Planning

- Quality planning is the process of developing a quality plan for a project.
- The quality plan should set out the desired software qualities and describe how these are to be assessed.
- Without this definition, engineers may make different and sometimes conflicting assumptions about which product attributes reflect the most important quality characteristics
- Humphrey (1989), in his classic book on software management, suggests an outline structure for a quality plan. This includes:

1. **Product introduction:** A description of the product, its intended market, and the quality expectations for the product.
2. **Product plans:** The critical release dates and responsibilities for the product, along with plans for distribution and product servicing.

3. **Process descriptions:** The development and service processes and standards that should be used for product development and management.
4. **Quality goals:** The quality goals and plans for the product, including an identification and justification of critical product quality attributes.
5. **Risks and risk management:** The key risks that might affect product quality and the actions to be taken to address these risks.

- Quality plans, which are developed as part of the general project planning process, differ in detail depending on the size and the type of system that is being developed.
- However, when writing quality plans, you should try to keep them as short as possible.
- If the document is too long, people will not read it and this will defeat the purpose of producing the quality plan.
- Standards and processes are important but quality managers should also aim to develop a 'quality culture' where everyone responsible for software development is committed to achieving a high level of product quality.
- They should encourage teams to take responsibility for the quality of their work and to develop new approaches to quality improvement

Software Quality

- The fundamentals of quality management were established by manufacturing industry in a drive to improve the quality of the products that were being made
- As part of this, they developed a definition of 'quality', which was based on conformance with a detailed product specification (Crosby, 1979) and the notion of tolerances
- The underlying assumption was that products could be completely specified and procedures could be established that could check a manufactured product against its specification.
- Of course, products will never exactly meet a specification so some tolerance was allowed. If the product was 'almost right', it was classed as acceptable
- Software quality is not directly comparable with quality in manufacturing.
- The idea of tolerances is not applicable to digital systems and, for the following reasons, it may be impossible to come to an objective conclusion about whether or not a software system meets its specification

- It is difficult to write complete and unambiguous software specifications. Software developers and customers may interpret the requirements in different ways and it may be impossible to reach agreement on whether or not software conforms to its specification.
- Specifications usually integrate requirements from several classes of stakeholders.
- These requirements are inevitably a compromise and may not include the requirements of all stakeholder groups.
- The excluded stakeholders may therefore perceive the system as a poor quality system, even though it implements the agreed requirements.
- It is impossible to measure certain quality characteristics (e.g., maintainability) directly and so they cannot be specified in an unambiguous way
- Because of these problems, the assessment of software quality is a subjective process where the quality management team has to use their judgment to decide if an acceptable level of quality has been achieved.
- The quality management team has to consider whether or not the software is fit for its intended purpose.
- This involves answering questions about the system's characteristics. **For example**:

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

- The subjective quality of a software system is largely based on its non-functional characteristics.
- This reflects practical user experience—if the software's functionality is not what is expected, then users will often just work around this and find other ways to do what they want to do.
- However, if the software is unreliable or too slow, then it is practically impossible for them to achieve their goals.

- Software quality is not just about whether the software functionality has been correctly implemented, but also depends on non-functional system attributes.
- **Boehm, et al. (1978) suggested that there were 15 important software quality attributes**

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

Fig: software quality attributes

- 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
- It may be that efficiency is critical and other factors have to be sacrificed to achieve this.
- If you have stated this in the quality plan, the engineers working on the development can cooperate to achieve this.
- The plan should also include a definition of the quality assessment process
- An assumption that underlies software quality management is that the quality of software is directly related to the quality of the software development process
- This again comes from manufacturing systems where product quality is intimately related to the production process.
- A manufacturing process involves configuring, setting up, and operating the machines involved in the process.
- Once the machines are operating correctly, product quality naturally follows.
- You measure the quality of the product and change the process until you achieve the quality level that you need

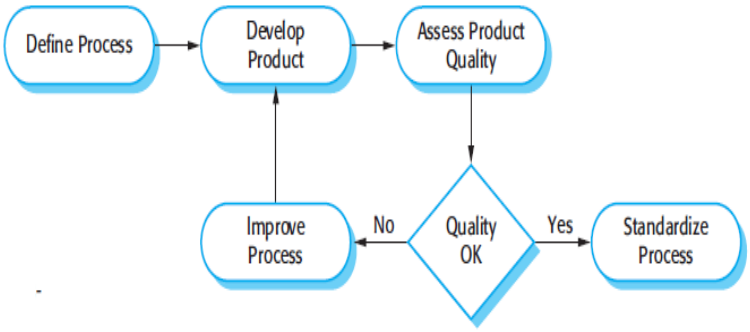

Fig: Process Based Quality

- In software development, therefore, the relationship between process quality and product quality is more complex.
- Software development is a creative rather than a mechanical process, so the influence of individual skills and experience is significant
- External factors, such as the novelty of an application or commercial pressure for an early product release, also affect product quality irrespective of the process used

Software Standards

- Software standards play a very important role in software quality management
- An important part of quality assurance is the definition or selection of standards that should apply to the software development process or software product.
- As part of this QA process, tools and methods to support the use of these standards may also be chosen
- Once standards have been selected for use, project-specific processes have to be defined to monitor the use of the standards and check that they have been followed
- **Software standards are important for three reasons:**

1. Standards capture wisdom that is of value to the organization. They are based on knowledge about the best or most appropriate practice for the company. This knowledge is often only acquired after a great deal of trial and error. Building it into a standard helps the company reuse this experience and avoid previous mistakes.
2. Standards provide a framework for defining what 'quality' means in a particular setting. Software quality is subjective, and by using standards you establish a basis for deciding if a required level of quality has been achieved. Of course, this depends

on setting standards that reflect user expectations for software dependability, usability, and performance

3. Standards assist continuity when work carried out by one person is taken up and continued by another. Standards ensure that all engineers within an organization adopt the same practices. Consequently, the learning effort required when starting new work is reduced.

- There are two related types of software engineering standard that may be defined and used in software quality management:
- ***Product standards:*** *These* apply to the software product being developed. They include document standards, such as the structure of requirements documents, documentation standards, such as a standard comment header for an object class definition, and coding standards, which define how a programming language should be used.
- ***Process standards:*** *These* define the processes that should be followed during software development. They should encapsulate good development practice. Process standards may include definitions of specification, design and validation processes, process support tools, and a description of the documents that should be written during these processes
- Standards have to deliver value, in the form of increased product quality.
- There is no point in defining standards that are expensive in terms of time and effort to apply, that only lead to marginal improvements in quality.
- **Product standards** have to be designed so that they can be applied and checked in a cost-effective way
- **Process standards** should include the definition of processes that check that product standards have been followed
- National and international bodies such as the U.S. DoD, ANSI, BSI, NATO, and the IEEE support the production of standards.
- These are general standards that can be applied across a range of projects.
- Bodies such as NATO and other defence organizations may require that their own standards be used in the development contracts that they place with software companies
- Quality management teams that are developing standards for a company should normally base these company standards on national and international standards.

- Using international standards as a starting point, the quality assurance team should draw up a standards 'handbook'. This should define the standards that are needed by their organization

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

Fig: Product and Process Standards

- Software engineers sometimes consider standards to be overprescriptive and not really relevant to the technical activity of software development.
- This is particularly likely when project standards require tedious documentation and work recording
- To minimize dissatisfaction and to encourage buy-in to standards, quality managers who set the standards should therefore take the following steps:

1. ***Involve software engineers in the selection of product standards:*** *If developers* understand why standards have been selected, they are more likely to be committed to these standards. Ideally, the standards document should not just set out the standard to be followed, but should also include commentary explaining why standardization decisions have been made
2. ***Review and modify standards regularly to reflect changing technologies:*** standards are expensive to develop and they tend to be enshrined in a company standards handbook. Because of the costs and discussion required, there is often a reluctance to change them. A standard handbook is essential but it should evolve to reflect changing circumstance and technology.
3. ***Provide software tools to support standards*****:** If tool support is available, very little effort is required to follow the software development standards.

The ISO 9001 standards framework

- There is an international set of standards that can be used in the development of quality management systems in all industries, called ISO 9000.

- ISO 9000 standards can be applied to a range of organizations from manufacturing through to service industries.
- ISO 9001, the most general of these standards, applies to organizations that design, develop, and maintain products, including software.
- The ISO 9001 standard was originally developed in 1987, with its most recent revision in 2008
- The ISO 9001 standard is not itself a standard for software development but 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.
- The major revision of the ISO 9001 standard in 2000 reoriented the standard around nine core processes

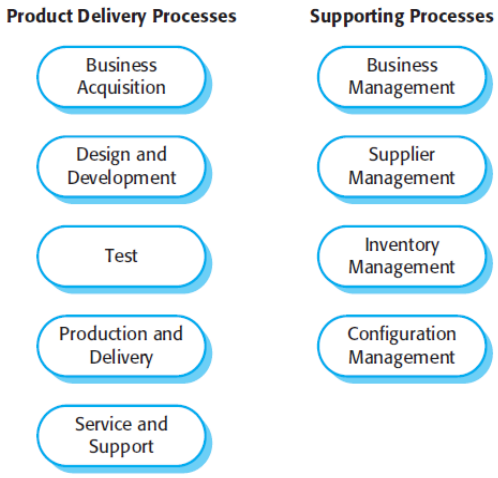

Fig: ISO 9001 core processes

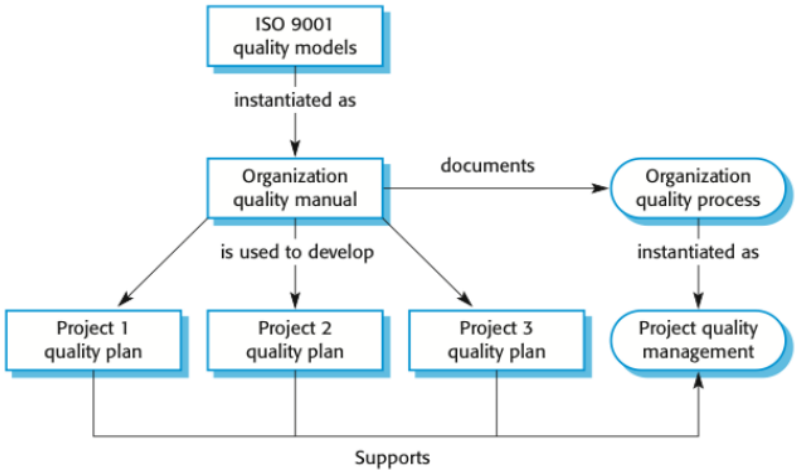

Fig: ISO 9001 and quality management

- If an organization is to be ISO 9001 conformant, it must document how its processes relate to these core processes.
- It must also define and maintain records that demonstrate that the defined organizational processes have been followed.
- The company quality manual should describe the relevant processes and the process data that has to be collected and maintained

Reviews and inspections

- Reviews and inspections are QA activities that check the quality of project deliverables.
- Reviews and Inspections may be part of the software verification and validation processes
- This involves examining the software, its documentation and records of the process to discover errors and omissions and to see if quality standards have been followed
- During a review, a group of people examine the software and its associated documentation, looking for potential problems and non-conformance with standards.
- The review team makes informed judgments about the level of quality of a system or project deliverable.
- Project managers may then use these assessments to make planning decisions and allocate resources to the development process
- Quality reviews are based on documents that have been produced during the software development process

- As well as software specifications, designs, or code, process models, test plans, configuration management procedures, process standards, and user manuals may all be reviewed
- The review should check the consistency and completeness of the documents or code under review and make sure that quality standards have been followed.
- However, reviews are not just about checking conformance to standards.
- They are also used to help discover problems and omissions in the software or project documentation.
- The conclusions of the review should be formally recorded as part of the quality management process.
- If problems have been discovered, the reviewers' comments should be passed to the author of the software or whoever is responsible for correcting errors or omissions.
- The purpose of reviews and inspections is to improve software quality, not to assess the performance of people in the development team
- Reviewing is a public process of error detection
- Mistakes that are made by individuals are revealed to the whole programming team.
- To ensure that all developers engage constructively with the review process, project managers have to be sensitive to individual concerns.
- They must develop a working culture that provides support without blame when errors are discovered.

Quality Review vs. Progress Reviews

- Although a quality review provides information for management about the software being developed, quality reviews are not the same as management progress reviews
- Progress reviews compare the actual progress in a software project against the planned progress.
- Their prime concern is whether or not the project will deliver useful software on time and on budget

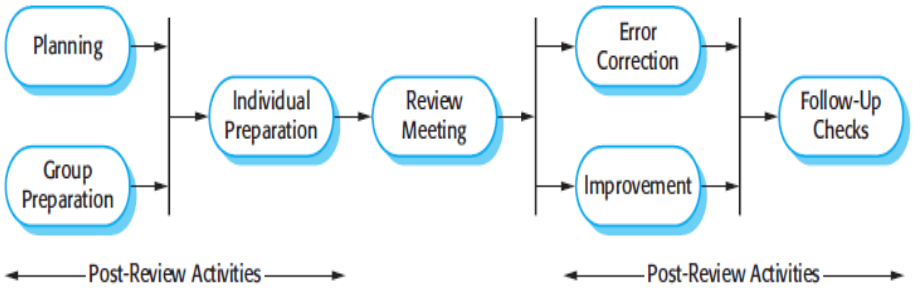

Fig: the software review process

- Although there are many variations in the details of reviews, the review process is normally structured into three phases:

1. ***Pre-review activities*:** These are preparatory activities that are essential for the review to be effective. Typically, 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.
2. ***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. The review itself should be relatively short—two hours at most. One team member should chair the review and another should formally record all review decisions and actions to be taken.
3. ***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.

Program Inspections

- Program inspections are 'peer reviews' where team members collaborate to find bugs in the program that is being developed
- Program inspections involve team members from different backgrounds who make a careful, line-by-line review of the program source code.
- They look for defects and problems and describe these at an inspection meeting.
- Defects may be logical errors, anomalies in the code that might indicate an erroneous condition or features that have been omitted from the code
- During an inspection, a checklist of common programming errors is often used to focus the search for bugs

- You use different checklists for different programming languages because each language has its own characteristic errors.

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?

Fig: An inspection checklist

Software measurement and metrics

- Software measurement is concerned with deriving a numeric value or profile for an attribute of a software component, system, or process.
- By comparing these values to each other and to the standards that apply across an organization, you may be able to draw conclusions about the quality of software, or assess the effectiveness of software processes, tools, and methods
- For example, say an organization intends to introduce a new software-testing tool.
- Before introducing the tool, you record the number of software defects discovered in a given time.
- This is a baseline for assessing the effectiveness of the tool.

- After using the tool for some time, you repeat this process. If more defects have been found in the same amount of time, after the tool has been introduced, then you may decide that it provides useful support for the software validation process
- The long-term goal of software measurement is to use measurement in place of reviews to make judgments about software quality.
- Using software measurement, a system could ideally be assessed using a range of metrics and, from these measurements, a value for the quality of the system could be inferred. If the software had reached a required quality threshold, then it could be approved without review
- A software metric is a characteristic of a software system, system documentation, or development process that can be objectively measured.
- Examples of metrics include the size of a product in lines of code; the Fog index which is a measure of the readability of a passage of written text; the number of reported faults in a delivered software product; and the number of person-days required to develop a system component
- Software metrics may be either control metrics or predictor metrics.
- As the names imply, control metrics support process management, and predictor metrics help you predict characteristics of the software.
- Control metrics are usually associated with software processes. Examples of control or process metrics are the average effort and the time required to repair reported defects
- Predictor metrics are associated with the software itself and are sometimes known as 'product metrics'.
- Examples of predictor metrics are the cyclomatic complexity of a module, the average length of identifiers in a program, and the number of attributes and operations associated with object classes in a design
- Both control and predictor metrics may influence management decision making

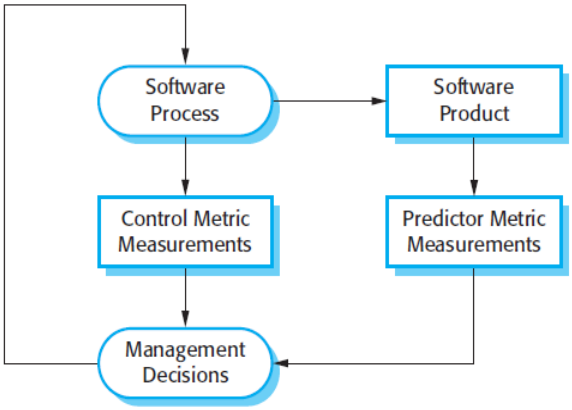

Fig: Predictor and control measurement

- There are two ways in which measurements of a software system may be used:

  1. ***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.
  2. ***To identify the system components whose quality is substandard:*** *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
- Unfortunately, it is difficult to make direct measurements of many of the software quality attributes
- Quality attributes such as maintainability, understandability, and usability are external attributes that relate to how developers and users experience the software.
- They are affected by subjective factors, such as user experience and education, and they cannot therefore be measured objectively.
- To make a judgment about these attributes, you have to measure some internal attributes of the software (such as its size, complexity, etc.) and assume that these are related to the quality characteristics that you are concerned with

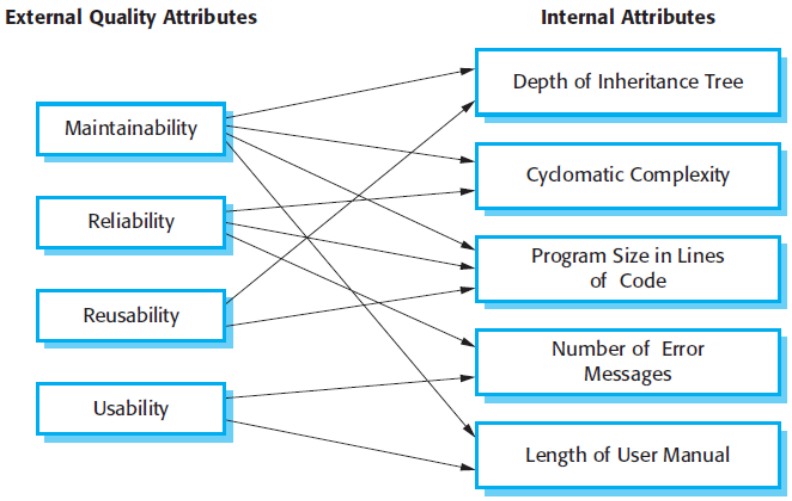

Fig: Relationship between internal and external software

- The diagram suggests that there may be relationships between external and internal attributes, but it does not say how these attributes are related.
- If the measure of the internal attribute is to be a useful predictor of the external software characteristic, **three conditions must hold**
  1. The internal attribute must be measured accurately. This is not always straightforward and it may require special-purpose tools to make the measurements
  2. A relationship must exist between the attribute that can be measured and the external quality attribute that is of interest. That is, the value of the quality attribute must be related, in some way, to the value of the attribute than can be measured.
  3. This relationship between the internal and external attributes must be understood, validated, and expressed in terms of a formula or model.
- There are several reasons why this is difficult:
  1. It is impossible to quantify the return on investment of introducing an organizational metrics program. There have been significant improvements in software quality over the past few years without the use of metrics so it is difficult to justify the initial costs of introducing systematic software measurement and assessment.
  2. There are no standards for software metrics or standardized processes for measurement and analysis. Many companies are reluctant to introduce measurement programs until such standards and supporting tools are available.
  3. In many companies, software processes are not standardized and are poorly defined and controlled. As such, there is too much process variability within the same company for measurements to be used in a meaningful way.
  4. Much of the research on software measurement and metrics 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, or by using agile methods.

5. Introducing measurement adds additional overhead to processes. This contradicts the aims of agile methods, which recommend the elimination of process activities that are not directly related to program development. Companies that have adopted agile methods are therefore not likely to adopt a metrics program.

Product Metrics

- Product metrics are predictor metrics that are used to measure internal attributes of a software system.
- Examples of product metrics include the system size, measured in lines of code, or the number of methods associated with each object class
- Product metrics fall into two classes:
- **Dynamic metrics,** which are collected by measurements made of a program in execution.
- These metrics can be collected during system testing or after the system has gone into use. An example might be the number of bug reports or the time taken to complete a computation
- **Static metrics,** which are collected by measurements made of representations of the system, such as the design, program, or documentation. Examples of static metrics are the code size and the average length of identifiers used.
- These types of metric are related to different quality attributes.
- Dynamic metrics help to assess the efficiency and reliability of a program.
- Static metrics help assess the complexity, understandability, and maintainability of a software system or system components
- There is usually a clear relationship between dynamic metrics and software quality characteristics

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.

Fig: static software product metrics

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.

Fig: the CK object oriented metrics suite

Software Component Analysis

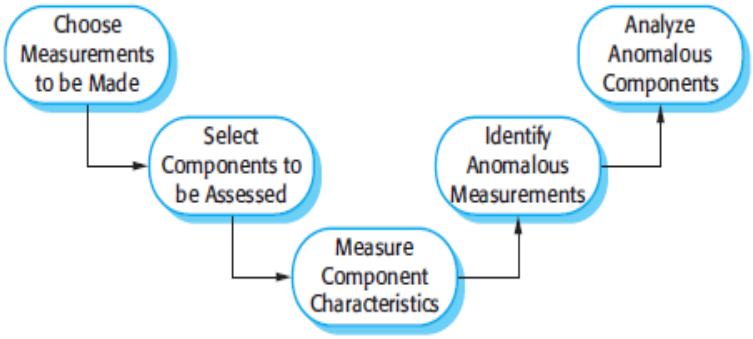

Fig: The Process of Product Measurement

- Each system component can be analyzed separately using a range of metrics.
- The values of these metrics may then be 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 key stages in this component measurement process are:

1. ***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
2. ***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.
3. ***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
4. ***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
5. ***Analyze 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.

Measurement Ambiguity

- When you collect quantitative data about software and software processes, you have to analyze that data to understand its meaning.
- It is easy to misinterpret data and to make inferences that are incorrect. You cannot simply look at the data on its own— you must also consider the context where the data is collected
- To illustrate how collected data can be interpreted in different ways, consider the scenario below, which is concerned with the number of change requests made by users of a system:

> *A manager decides to monitor the number of change requests submitted by customers based on an assumption that there is a relationship between these change requests and product usability and suitability. She assumes that the higher the number of change requests, the less the software meets the needs of the customer.*
>
> *Handling change requests and changing the software is expensive. The organization therefore decides to modify its process with the aim of improving customer satisfaction and, at the same time, reduce the costs of making changes. The intent is that the process changes will result in better products and fewer change requests.*
>
> *Process changes are initiated to increase customer involvement in the software design process. Beta testing of all products is introduced and customer-requested modifications are incorporated in the delivered product. New versions of products, developed with this modified process, are delivered. In some cases, the number of change requests is reduced. In others, it is increased. The manager is baffled and finds it impossible to assess the effects of the process changes on the product quality.*

- To understand why this kind of ambiguity can occur, you have to understand the reasons why users might make change requests:
- The software is not good enough and does not do what customers want it to do. They therefore request changes to deliver the functionality that they require.
- Alternatively, the software may be very good and so it is widely and heavily used. Change requests may be generated because there are many software users who creatively think of new things that could be done with the software
- Therefore, increasing the customer involvement in the process may reduce the number of change requests for products where the customers were unhappy.

- The process changes have been effective and have made the software more usable and suitable.
- Alternatively, however, the process changes may not have worked and customers may have decided to look for an alternative system.
- The number of change requests might decrease because the product has lost market share to a rival product and there are consequently fewer product users
- To analyze the change request data, you do not simply need to know the number of change requests.
- You need to know who made the request, how they use the software, and why the request was made. You also need information about external factors such as modifications to the change request procedure or market changes that might have an effect.
- With this information, it is then possible to find out if the process changes have been effective in increasing product quality.