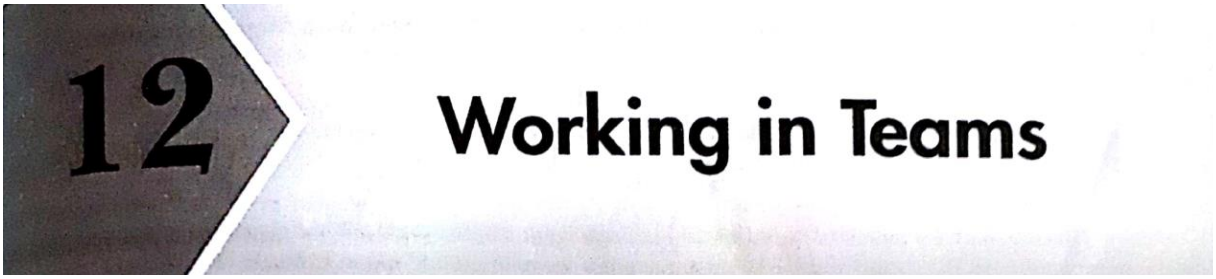
Unit V



Becoming a team

Five basic stages of development:

- Forming
- Storming
- Norming
- Performing
- Adjourning

Classification associated with Tuckman and Jensen

1. The members of the group get to know one another and try to set up some ground rules about behaviour
2. Storming – conflicts arise as various members of the group try to exert leadership and the group's methods of working are established
3. Norming – conflicts are largely settled and a feeling of group identity emerges
4. Performing – the emphasis is now on the tasks at hand
5. Adjourning – the group disbands

One way of attempting to accelerate this process is through team-building exercises.

This is covered in Section 12.2 of the text book.

Balanced teams

- Meredith Belbin studied the performance of top executives carrying out group work at the Hendon Management Centre
- Tried putting the 'best' people together in 'Apollo' teams – almost invariably did badly
- Identified the need for a balance of skills and management roles in a successful team

Management team roles

- The co-ordinator – good at chairing meetings
- The 'plant' – an idea generator
- The monitor-evaluator – good at evaluating ideas
- The shaper – helps direct team's efforts
- The team worker – skilled at creating a good working environment

Belbin management roles – continued

- The resource investigator – adept at finding resources, including information
- The completer-finisher – concerned with getting tasks completed
- The implementer – a good team player who is willing to undertake less attractive tasks if they are needed for team success
- The specialist – the 'techie' who likes to acquire knowledge for its own sake

- The 'specialist' was added by Belbin in 1996.
- A person can have elements of more than one of the types, but usually one or two predominate. According to Belbin about 30% cannot be classified under any heading at all.
- Problems can occur if, for example, you have two or more shapers and no effective co-ordinator – there are likely to be clashes that are difficult to resolve.

Group performance

Some tasks are better carried out collectively while other tasks are better delegated to individuals

- *Additive tasks* – the effort of each participant is summed
- *Compensatory tasks* – the judgements of individual group members are summed – errors of some compensated for by judgements of others

An example of an additive task is clearing snow – in practice there might be some diseconomies of scale if there were too many people involved

Compensatory tasks – using groups of developers to estimate the development effort needed to develop new software components

- *Disjunctive tasks* – there is only one correct answer – someone must:
  - Come up with right answer
  - Persuade the other that they are right
- *Conjunctive* – the task is only finished when all components have been completed

Software development would tend to be conjunctive – all the components have to be completed before the system as a whole is deemed to be complete

'Social loafing'

- Tendency for some team participants to 'coast' and let others do the work
- Also tendency not to assist other team members who have problems
- Suggested counter-measures:
  - Make individual contributions identifiable
  - Consciously involve group members ('loafer' could in fact just be shy!)
  - Reward 'team players'

Barriers to good team decisions

- Inter-personal conflicts – see earlier section on team formation
- Conflicts tend to be a dampened by emergence of *group norms* – shared group opinions and attitudes
- *Risky shift* – people in groups are more likely to make risky decisions than they would as individuals

Delphi approach

To avoid dominant personalities intruding the

following approach is adopted

1. Enlist co-operation of experts
2. Moderator presents experts with problem
3. Experts send in their recommendations to the moderator
4. Recommendations are collated and circulated to all experts
5. Experts comment on ideas of others and modify their own recommendation if so moved
6. If moderator detects consensus, stop; else back to 4

The Delphi approach was originally developed by the RAND Corporation in the late 1960s. The development of email clearly makes the application of the method much easier to organize.

Team 'heedfulness'

- Where group members are aware of the activities of other members that contribute to overall group success
- Impression of a 'collective mind'
- Some attempts to promote this:
  - Egoless programming
  - Chief programmer teams
  - XP
  - Scrum

Egoless programming

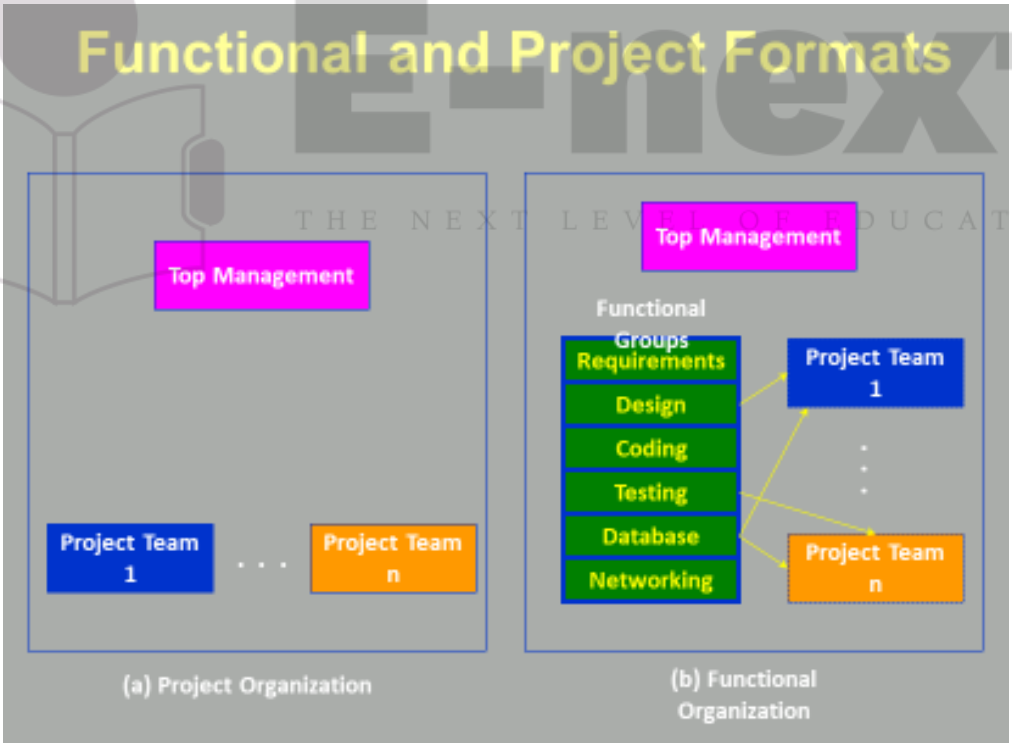
- Gerry Weinberg noted a tendency for programmers to be protective of their code and to resist perceived criticisms by others of the code
- Encouraged programmers to read each others code
- Argued that software should become communal, not personal – hence 'egoless programming'

Organization and Team Structures

- Two important issues that are critical to the effective functioning of every organization are:
  - **Department structure:** How is a department organized into teams?
  - **Team structure:** How are the individual project teams structured?

Department Structure

- **Functional format:**
  - Each functional group comprises of developers having expertise in some specific task or functional area.
- **Project format:**
  - The same team carries out all the project activities.



Functional versus project formats

- Ease of staffing
- Production of good quality documents

- Job specialization
- Efficient handling of the problems associated with manpower turnover
- Career planning

Matrix Format

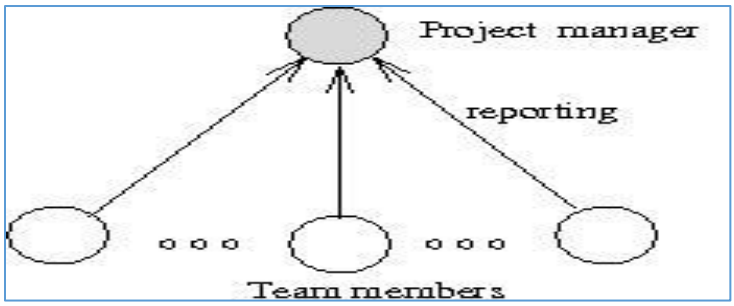
- The pool of functional specialists are assigned to different projects as needed.

	Project			
Functional group	#1	#2	#3	
#1	2	0	3	Functional manager 1
#2	0	5	3	Functional manager 2
#3	0	4	2	Functional manager 3
#4	1	4	0	Functional manager 4
#5	0	4	6	Functional manager 5
	Project manager 1	Project manager 2	Project manager 3	

Team Structure

- We consider only three team structures:
  - Democratic,
  - Chief programmer,
  - Mixed team

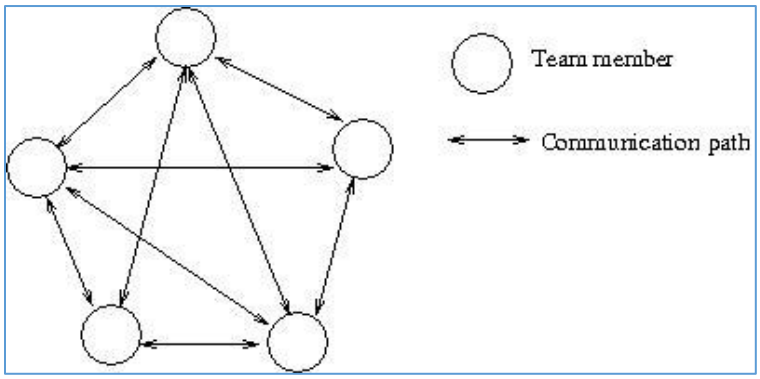
Chief programmer teams



- Fred Brooks was concerned about the need to maintain 'design consistency' in large software systems
- Appointment of key programmers, **Chief Programmers**, with responsibilities for defining requirements, designing, writing and test software code

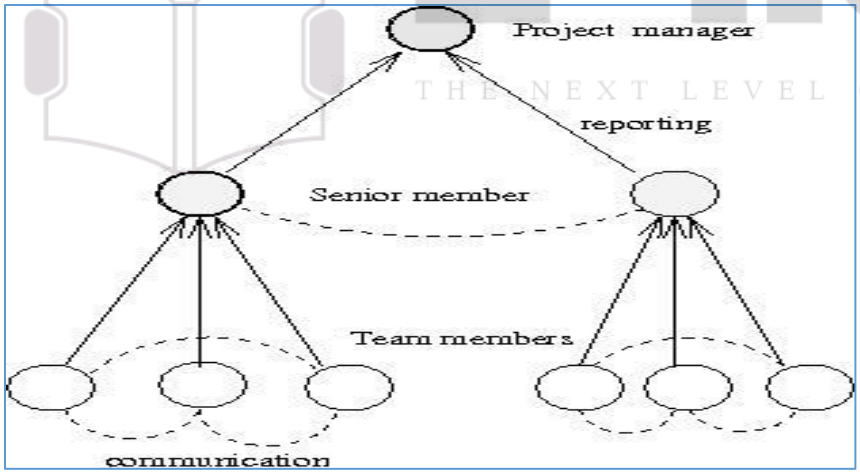
- Assisted by a support team: **co-pilot** – shared coding, **editor** who made typed in new or changed code, **program clerk** who wrote and maintained documentation and **tester**
- Problem – finding staff capable of the chief programmer role

Democratic Team



- Does not enforce any formal team hierarchy.
- Decisions are taken based on discussions,
  - any member is free to discuss with any other member
- Since a lot of debate and discussions among the team members takes place,
  - for large team sizes significant overhead is incurred

Mixed Control Team Structure



- Incorporates both hierarchical reporting and democratic set up.

Extreme programming

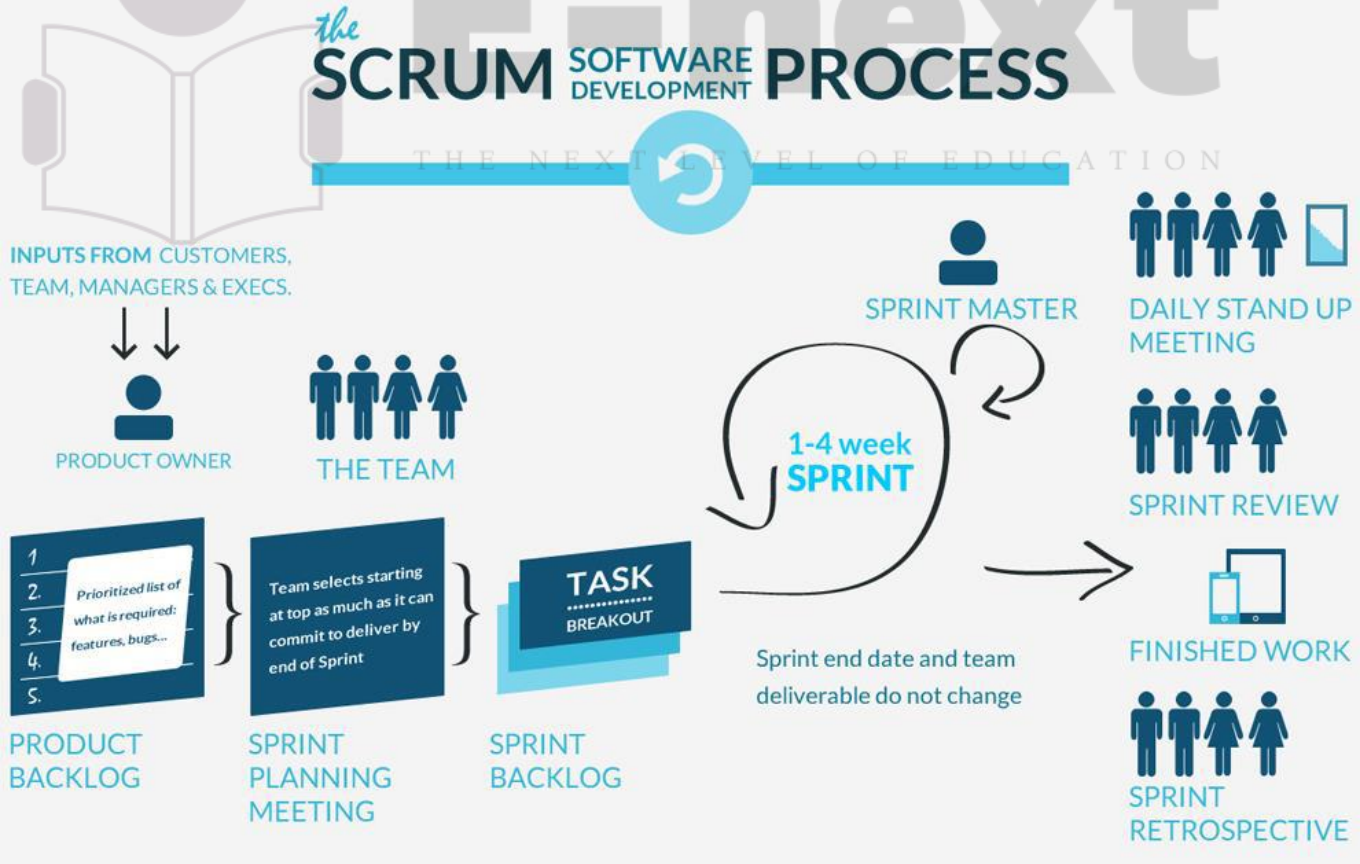
XP can be seen as an attempt to improve team heedfulness and reduce the length of communication paths (the time between something being recorded and it being used)

- Software code enhanced to be self-documenting
- Software regularly refactored to clarify its structure

- Test cases/expected results created *before* coding – acts as a supplementary specification
- Pair programming – a development of the co-pilot concept

Scrum

- Named as an analogy to a rugby scrum – all pushing together
- Originally designed for new product development where 'time-to-market' is important
- 'Sprints' increments of typically one to four weeks
- Daily 'scrums' – daily stand-up meetings of about 15 minutes
- Unlike XP, requirements are frozen during a sprint
- At the beginning of the sprint there is a sprint planning meeting where requirements are prioritized
- At end of sprint, a review meeting where work is reviewed and requirements may be changed or added to



Co-ordination of dependencies

- The previous discussion on team heedfulness focused (mainly) in communication inside the team
- What sort of communications are needed between teams and other units
- Co-ordination theory has identified the following types of coordination:
  - *Shared resources. e.g.* where several projects need the services of scarce technical experts for certain parts of the project.
  - *Producer-customer ('right time') relationships.* A project activity may depend on a product being delivered first.
  - *Task-subtask dependencies.* In order to complete a task a sequence of subtasks have to be carried out.
  - *Accessibility ('right place') dependencies.* This type of dependency is of more relevance to activities that require movement over a large geographical area, but arranging the delivery and installation of IT equipment might be identified as such.
  - *Usability ('right thing') dependencies.* Broader concern than the design of user interfaces: relates to the general question of *fitness for purpose*, e.g. the satisfaction of business requirements.
  - *Fit requirements.* This is ensuring that different system components work together effectively.

Why 'virtual projects'?

The physical needs of software developers (according to an IBM report):

- 100 square feet of floor space
- 30 square feet of work surface
- Dividers at least 6 feet high to muffle noise
- Demarco and Lister found clear statistical links between noise and coding error rates
- One answer: send the developers home!

In practice, most organizations (in the UK at least) pay little attention to creating the optimal software development environment; often they are constrained by the existing structure of buildings.

One solution is to encourage working from home. Taken a step further people at a distance, even in other continents, can be employed.

Possible advantages

- Can use staff from developing countries – lower costs
- Can use short term contracts:
  - Reduction in overheads related to use of premises
  - Reduction in staff costs, training, holidays, pensions etc.
- Can use specialist staff for specific jobs

Most of the advantages are related to cost reduction.

Further advantages

- Productivity of home workers can be higher – fewer distractions
- Can take advantage of time zone differences e.g. overnight system testing

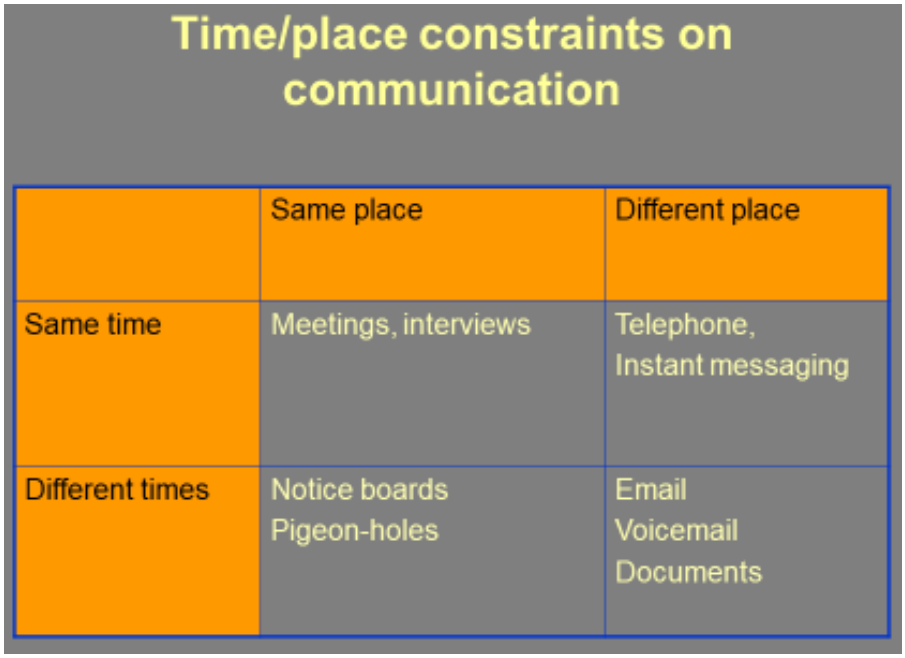
Some challenges

- Work requirements have to be carefully specified
- Procedures need to be formally documented
- Co-ordination can be difficult
- Payment methods need to be modified – piece-rates or fixed price, rather then day-rates

Most of the challenges relate to the organization of staff work. Things have to be spelled out in advance.

More challenges

- Possible lack of trust when there is no face-to-face contact
- Assessment of quality of delivered products needs to be rigorous
- Different time zones can cause communication and co-ordination problems



Other factors influencing communication genres

- Size and complexity of information – favours documents
- Familiarity of context e.g. terminology – where low, two-way communication favoured
- Personally sensitive – it has to be face-to-face communication here

Other factors that would influence the choice of communication methods.

Best method of communication depends on stage of project

- Early stages
  - Need to build trust
  - Establishing context
  - Making important 'global' decisions
  - *Favours same time/ same place*
- Intermediate stages
  - Often involves the paralled detailed design of components
  - Need for clarification of interfaces etc
  - *Favours same time/different place*

Different stages of a project would favour different modes of communication

- Implementation stages
  - Design is relatively clear
  - Domain and context familiar
  - Small amounts of operational data need to be exchanged
  - Favours different time/different place communications e.g. e-mail
- Face to face co-ordination meetings – the 'heartbeat' of the project

Communications plans

- As we have seen choosing the right communication methods is crucial in a project
- Therefore, a good idea to create a **communication plan**
- **Stages** of creating a communication plan
  - Identify all the major stakeholders for the project – see chapter 1
  - Create a plan for the project – see chapter 3
  - Identify stakeholder and communication needs for each stage of the project
  - Document in a communication plan

Content of a communication plan

For each communication event and channel, identify:

- *What*. This contains the name of a particular communication event, e.g, 'kick-off meeting', or channel, e.g. 'project intranet site'.
- *Who/target*. The target audience for the communication.
- *Purpose*. What the communication is to achieve.
- *When/frequency*. If the communication is by means of a single event, then a date can be supplied. If the event is a recurring one, such as a progress meeting then the frequency should be indicated.
- *Type/method*. The nature of the communication, e.g., a meeting or a distributed document.

- *Responsibility*. The person who initiates the communication.

Leadership: types of authority

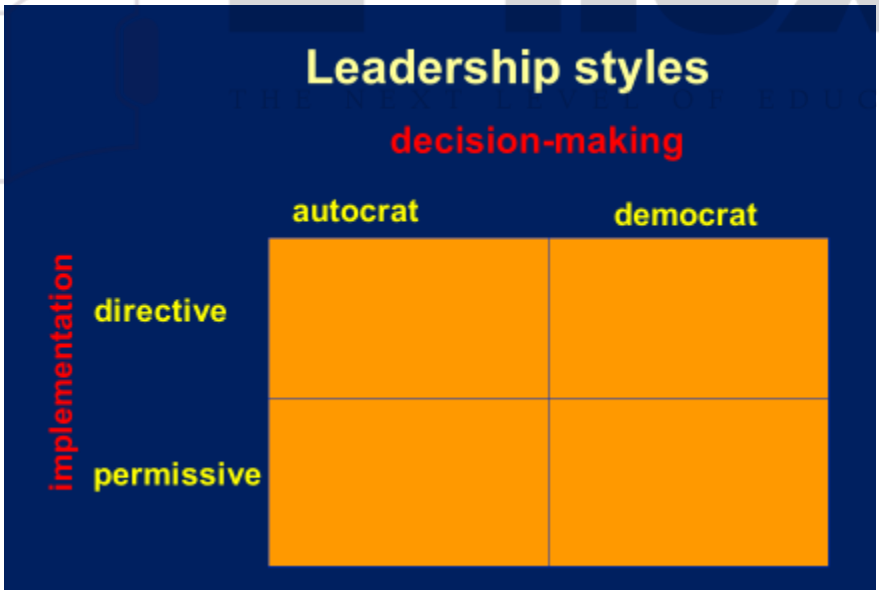
Position power

- Coercive power – able to threaten punishment
- Connection power – have access to those who do have power
- Legitimate power – based on a person's title conferring a special status
- Reward power – able to reward those who comply

Leadership: types of power

Personal power

- *Expert power:* holder can carry out specialist tasks that are in demand
- *Information power*: holder has access to needed information
- *Referent power*: based on personal attractiveness or charisma



Leadership styles

- Task orientation – focus on the work in hand
- People orientation – focus on relationships

- Where there is uncertainty about the way job is to be done or staff are inexperienced they welcome task oriented supervision
- Uncertainty is reduced – people orientation more important
- Risk that with reduction of uncertainty, managers have time on their hands and become more task oriented (interfering)

Essentially staff want hands-on management when they need guidance. Once the know the job they want to be left to get on with it!

Software product quality

The importance of software quality

- Increasing criticality of software
- The intangibility of software
- Project control concerns:
  - errors accumulate with each stage
  - errors become more expensive to remove the later they are found
  - it is difficult to control the error removal process (e.g. testing)

- Increasing criticality of software – e.g .software is increasingly being used in systems that can threaten or support human life and well-being
- The intangibility of software – it is difficult for observers to judge the quality of software development, especially during its early stages
- Project control concerns:
- The products of one sub-process in the development process are the inputs to subsequent sub-processes, thus
- errors accumulate with each stage e.g. at the design stage, the specification errors are incorporated into the design, and at the coding stage specification and design errors are incorporated into the software

- errors become more expensive to remove the later they are found
- it is difficult to control the error removal process (e.g. testing)
- See Section 13.3

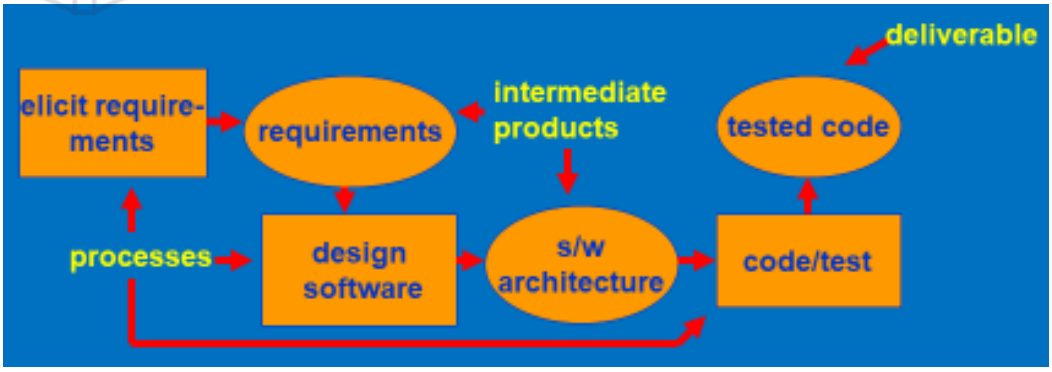
Quality specifications

Where there is a specific need for a quality, produce a quality specification

- Definition/description of the quality
- Scale: the unit of measurement
- Test: practical test of extent of quality
- Minimally acceptable: lowest acceptable value, if compensated for by higher quality level elsewhere
- Target range: desirable value
- Now: value that currently applies

ISO standards: development life cycles

A development life cycle (like ISO 12207) indicates the sequence of *processes* that will produce the software *deliverable* and the *intermediate products* that will pass between the processes.



The *deliverables* are the products that are handed over to the client at the end of the project, typically the executable code.

Intermediate products are things that are produced during the project, but which are not (usually) handed to the client at the end. Typically they are things that are produced by one sub-process (e.g. a requirements document created by the requirements elicitation and analysis processes) and used by others (e.g. a design process which produces a design that fulfils the requirements).

These sub-processes will fit into the overall framework of a *development cycle*.

Some software quality models focus on evaluating the quality of software products, others on the processes by which the products are created.

ISO standards

ISO 9126 Software product quality

Attributes of software product quality

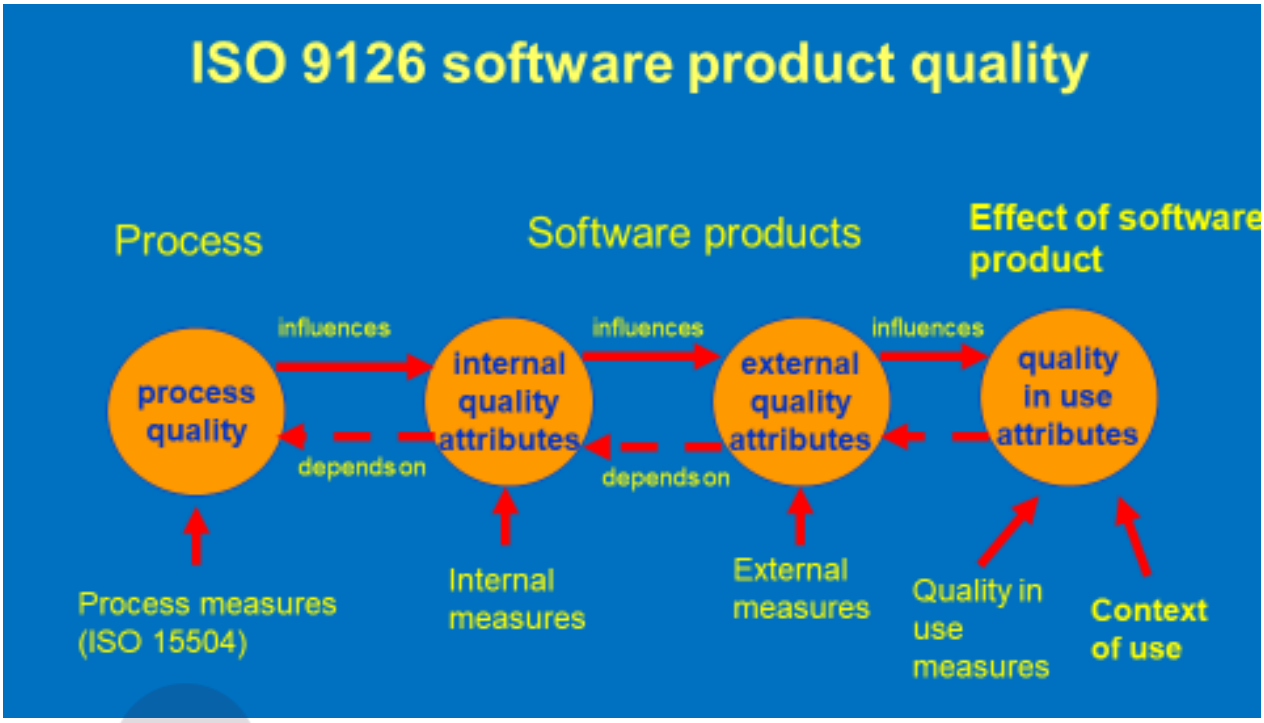
- External qualities i.e. apparent to the user of the deliverable
- Internal qualities i.e. apparent to the developers of the deliverables and the intermediate products

ISO 14598 Procedures to carry out the assessment of the product qualities defined in ISO 9126

ISO 9126 focuses on the definition of software quality, while ISO 14598 focuses on the way that the quality, once defined, is assessed. The ISO 9000 series of standards establishes requirements for quality management systems for the creation/supply of all types of goods and services while the ones mentioned here relate specifically to software.

Types of quality assessment

- During software development:
  - To assist developers to build software with the required qualities
- During software acquisition:
  - To allow a customer to compare and select the best quality product
- For a particular community of users:
  - Independent evaluation by assessors rating a software product



Internal quality attributes are the things that developers would be aware of during the project. They could be qualities in the intermediate products that are created. External quality attributes are the qualities of the final products that are delivered to the users. Thus users would be very aware of these. A key task is mapping these two types of quality.

Acceptable quality in software depends on the use to which the software is put. For example, a higher standard of reliability would be expected of a software component that was very heavily used and the continued functioning of which was essential to the organization, than of a rarely used software tool for which there were many alternatives.

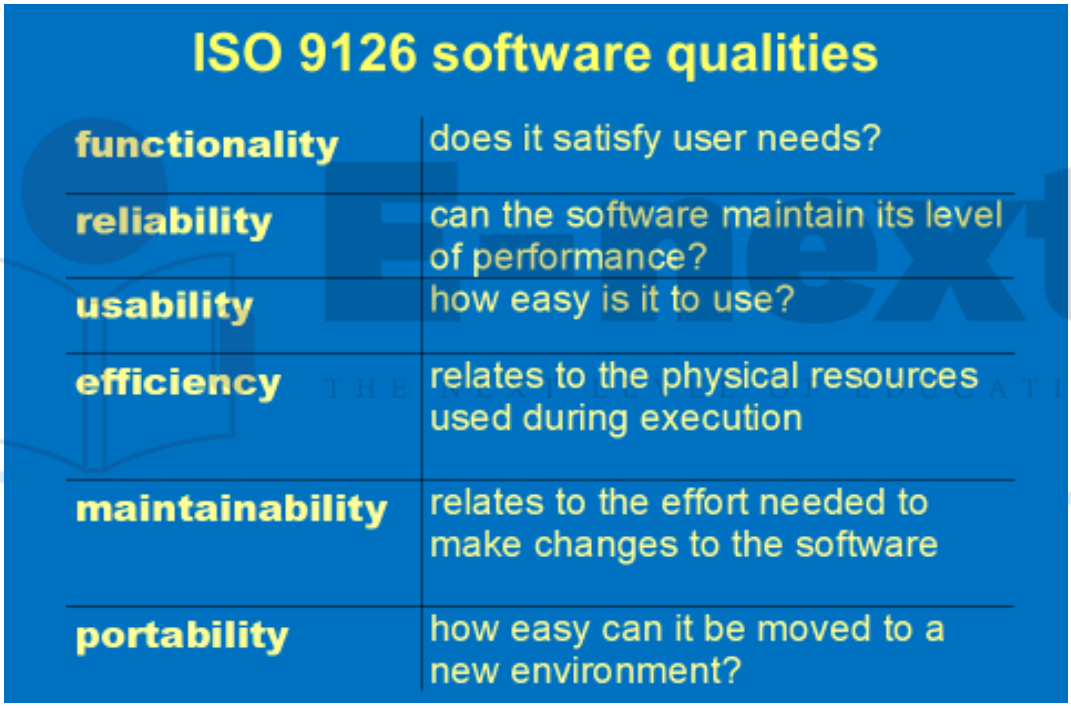
Quality in use

- Effectiveness – ability to achieve user goals with accuracy and completeness
- Productivity – avoids excessive use of resources in achieving user goals
- Safety – within reasonable levels of risk of harm to people, business, software, property, environment etc,
- Satisfaction – happy users!

'users' include those maintain software as well as those who operate it.

Quality Models

- It is a model of the software:
  - Constructed with the objective to describe, assess and/or predict quality.
- Popular models:
  - Garvin's model
  - Boehm's model
  - ISO 9126
  - Dromey's Model



These are the six top-level qualities identified in ISO9126.

Note that 'functionality' is listed as a quality. Sometimes 'functional' requirements are distinguished from 'non-functional' (i.e. quality) requirements.

Sub-characteristics of Functionality

- Suitability
- Accuracy

- Interoperability
  - ability of software to interact with other software components
- Functionality compliance
  - degree to which software adheres to application-related standards or legal requirements e.g audit
- Security
  - control of access to the system

In the case of interoperability, a good example is the ability to copy and paste content between different Microsoft products such as Excel and Word.

Sub-characteristics of Reliability

- Maturity
  - frequency of failure due to faults - the more the software has been used, the more faults will have been removed
- Fault-tolerance
- Recoverability
  - note that this is distinguished from 'security' - see above
- Reliability compliance

– complies with standards relating to reliability

Maturity – in general you would expect software that has been in operation for a relatively long time and had many users to be more reliable than brand-new software which has only a few users. In the first case, one would hope that most of the bugs had been found by the users and had been fixed by the developers.

Sub-characteristics of Usability

- Understandability
  - easy to understand?
- Learnability
  - easy to learn?

- Operability
  - easy to use?
- Attractiveness – this is a recent addition
- Usability compliance
  - compliance with relevant standards

Note that there may need to be a trade-off between 'learnability' and 'operability' – an analogy would be in learning shorthand writing: it takes a long time to do, but once you have done it you can write very quickly indeed.

'Attractiveness' is a new addition – it is important where the use of software is optional – if users don't like the software then they won't use it.

Sub-characteristics of Efficiency

- Time behaviour
  - e.g. response time
- Resource utilization
  - e.g. memory usage
- Efficiency compliance
  - compliance with relevant standards

Sub-characteristics of Maintainability

- "Analysability"
  - ease with which the cause of a failure can be found
- Changeability
  - how easy is software to change?
- Stability
  - low risk of modification having unexpected effects
- "Testability"
- Maintainability conformance

Testability can be particularly important with embedded software, such as that used to control aircraft. Software that simulates the actions of the aircraft in terms of inputs to the control system etc would need to be built.

Sub-characteristics of portability

- Adaptability
- "Installability"
- Co-existence
  - Capability of co-existing with other independent software products
- "Replaceability"
  - factors giving 'upwards' compatibility - 'downwards' compatibility is excluded
- Portability conformance
  - Adherence to standards that support portability

Replaceability - A new version of a software product should be able to deal correctly with all the inputs that the previous versions could deal with. However there could be new features that need new inputs that the old system would not be able to deal with.

Using ISO 9126 quality standards (development mode)

- Judge the importance of each quality factor for the application
  - for example, safety critical systems - *reliability* very important
  - real-time systems - *efficiency* important
- Select relevant external measurements within ISO 9126 framework for these qualities, for example
  - mean-time between failures for reliability
  - response-time for efficiency

Using ISO 9126 quality standards

- Map measurement onto ratings scale to show degree of user satisfaction – for example response time

response (secs)	rating
<2	Exceeds requirement
2-5	Target range
6-10	Minimally acceptable
>10	Unacceptable

- Identify the relevant internal measurements and the intermediate products in which they would appear
  - For example, at software design stage the estimated execution time for a transaction could be calculated

Using ISO9126 approach for application software selection

- Rather than map engineering measurement to qualitative rating, map it to a score
- Rate the importance of each quality in the range 1-5
- Multiply quality and importance scores

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

Weighted quality scores

		Product A		Product B	
Product quality	Importance rating (a)	Quality score (b)	Weighted score (a x b)	Quality score (c)	Weighted score (a x c)
usability	3	1	3	3	9
efficiency	4	2	8	2	8
maintain-ability	2	3	6	1	2
Overall totals			17		19

This suggests that Product B is slightly more likely to meet the users' quality needs.

How do we achieve product quality?

- The problem: quality attributes tend to *retrospectively* measurable
- Need to be able to examine processes by which product is created beforehand
- The production process is a network of sub-processes
  - Output from one process forms the input to the next
  - Errors can enter the process at any stage

Correction of errors

- Errors are more expensive to correct at later stages
  - need to rework more stages
  - later stages are more detailed and less able to absorb change
- Barry Boehm
  - Error typically 10 times more expensive to correct at coding stage than at requirements stage
  - 100 times more expensive at maintenance stage

The additional time needed to change products later is partly because more products in the project life cycle have to change e.g. the specification, design and code, rather than just the specification. Also the later products of the project tend to be more detailed and therefore more complicated.

For each activity, *define*:

- Entry requirements
  - these have to be in place before an activity can be started
  - example: 'a comprehensive set of test data and expected results be prepared and independently reviewed against the system requirement before program testing can commence'

We are moving into process quality now. However, the entry requirements could well relate to the quality of the products of other processes that this process will need to use.

- Implementation requirements
  - these define how the process is to be conducted
  - example 'whenever an error is found and corrected, *all* test runs must be completed, including those previously successfully passed'
- Exit requirements
  - an activity will not be completed until these requirements have been met
  - example: 'the testing phase is finished only when all tests have been run in succession with no outstanding errors'

The exit requirements, however, could well relate to characteristics of the products created by the process in question.

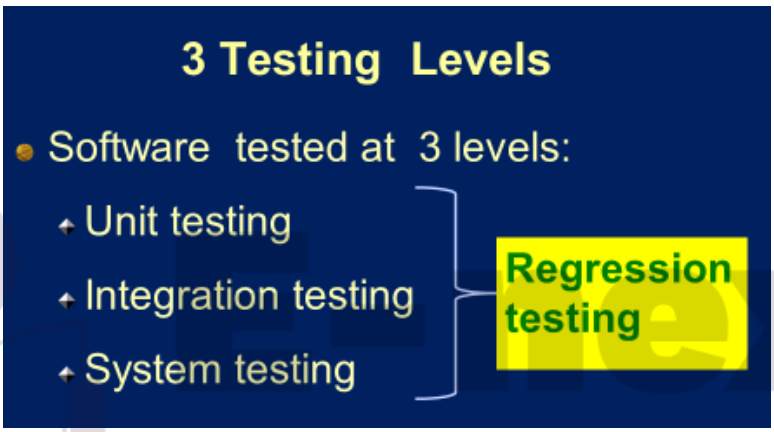
Techniques to Enhance Product Quality

- Increasing visibility
  - 'egoless programming'. Weinberg encouraged the simple practice of programmers looking at each other's code.
- Procedural structure
  - Over the years there has been the growth of
  - Methodologies

- Checking intermediate stages
  - checking the correctness of work at various stages of development

Testing

- Objective of testing is to expose as many bugs as possible.
- How is testing done?
  - Input test data to the program.
  - Observe the output: Check if the program behaves as expected



Test Levels

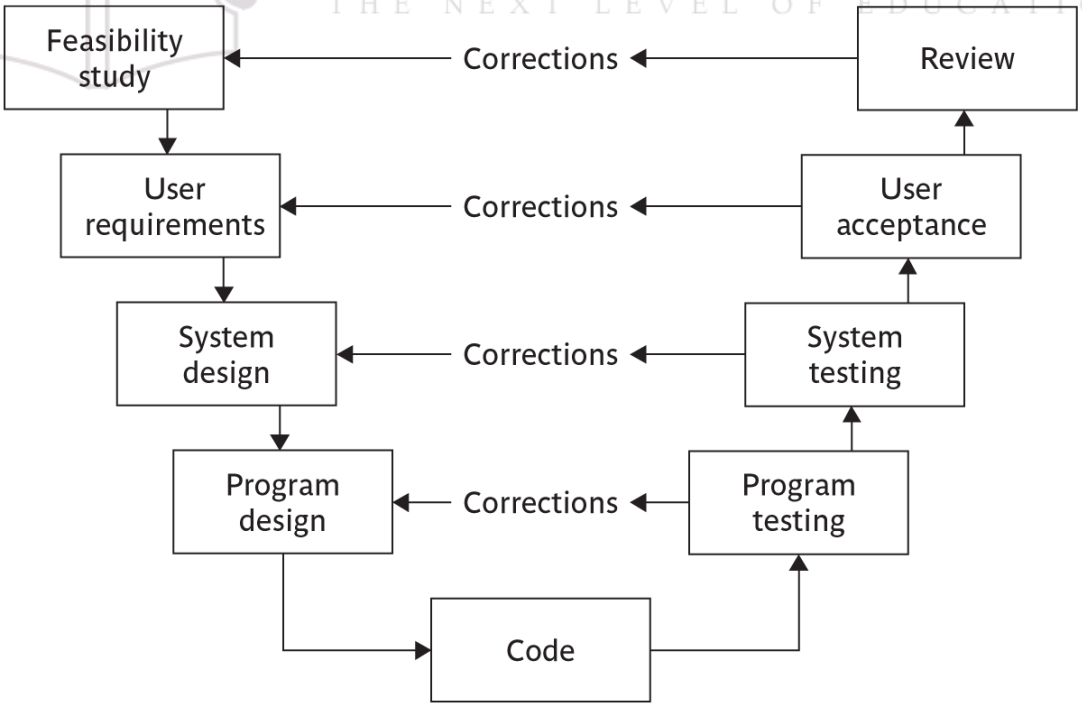
- Unit testing
  - Test each module (unit, or component) independently
  - Mostly done by developers of the modules
- Integration and system testing
  - Test the system as a whole
  - Often done by separate testing or QA team
- Acceptance testing
  - Validation of system functions by the customer

Verification versus Validation

- Verification is the process of determining:
  - Whether output of one phase of development conforms to its previous phase.
- Validation is the process of determining:
  - Whether a fully developed system conforms to its SRS document.
- Verification is concerned with phase containment of errors:
  - Whereas the aim of validation is that the final product be error free.

Testing: the V-process model

- This shown diagrammatically in the next slide
- It is an extension of the waterfall approach
- For each development stage there is a testing stage
- The testing associated with different stages serves different purposes e.g. system testing tests that components work together correctly, user acceptance testing that users can use system to carry out their work



Black box versus glass box test

- **Glass box testing**
  - The tester is aware of the internal structure of the code; can test each path; can assess percentage test coverage of the tests e.g. proportion of code that has been executed
- **Black box testing**
  - The tester is not aware of internal structure; concerned with degree to which it meets user requirements

Levels of testing

- Unit testing
- Integration testing
- System testing

Testing activities

- *Test planning*
- *Test suite design*
- *Test case execution and result checking*
- *Test reporting:*
- *Debugging:*
- *Error correction:*
- *Defect retesting*
- *Regression testing*
- *Test closure:*

Test plans

- Specify test environment
  - In many cases, especially with software that controls equipment, a special test system will need to be set up

- Usage profile
  - failures in operational system more likely in the more heavily used components
  - Faults in less used parts can lie hidden for a long time
  - Testing heavily used components more thoroughly tends to reduce number of operational failures

Management of testing

The tester executes test cases and may as a result find discrepancies between actual results and expected results – **issues**

Issue resolution – could be:

- a mistake by tester
- a fault – needs correction
- a fault – may decide not to correct: **off-specification**
- a change – software works as specified, but specification wrong: submit to change control

Decision to stop testing

The problem: impossible to know there are no more errors in code

Need to estimate how many errors are likely to be left

Bug seeding – insert (or leave) known bugs in code

Estimate of bugs left =

(total errors found)/(seeded errors found) x (total seeded errors)

Alternative method of error estimation

- Have two independent testers, A and B
- N_1 = valid errors found by A
- N_2 = valid errors found by B
- N_{12} = number of cases where same error found by A and B
- Estimate = $(N_1 \times N_2)/ N_{12}$

- Example: A finds 30 errors, B finds 20 errors. 15 are common to A and B. How many errors are there likely to be?

Test automation

- Other than reducing human effort and time in this otherwise time and effort-intensive work,
  - Test automation also significantly improves the thoroughness of testing.
- A large number of tools are at present available both in the public domain as well as from commercial sources.

Types of Testing Tools

- Capture and playback
- Automated test script
- Random input test
- Model-based test

Software reliability

- The reliability of a software product essentially denotes its *trustworthiness* or *dependability*.
  - Usually keeps on improving with time during the testing and operational phases as defects are identified and repaired.
- A reliability growth model (RGM) models how the reliability of a software product improves as failures are reported and bugs are corrected.
  - An RGM can be used to determine when during the testing phase a given reliability level will be attained, so that testing can be stopped

Software process quality

Product and Process Quality

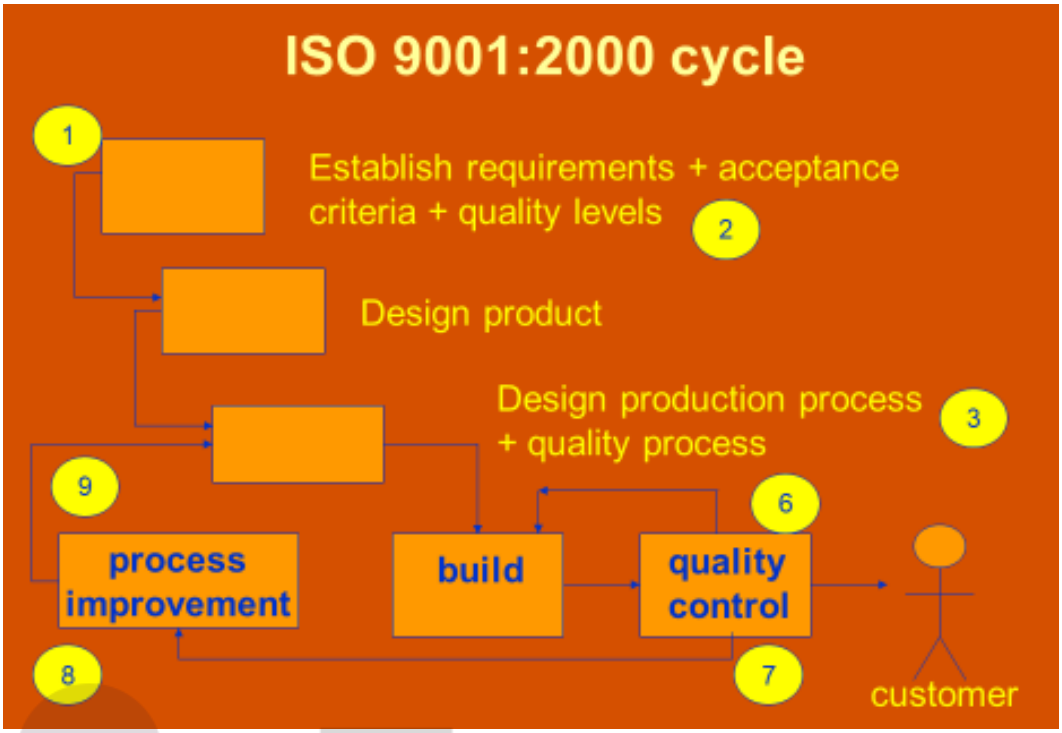
- A good process is usually required to produce a good product.
- For manufactured goods, process is the principal quality determinant.
- For design-based activity, other factors are also involved:
  - For example, the capabilities of the designers.

BS EN ISO 9001:2000 and quality management systems

- ISO 9001 is one of a family of standards that specify the characteristics of a good quality management system (QMS)
- Can be applied to the creation of any type of product or service, not just IT and software
- Does NOT set universal product/service standards
- DOES specify the way in which standards are established and monitored

ISO 9001:2000 principles

1. Understanding the requirements of the customer
2. Leadership to provide unity of purpose and direction to achieve quality
3. Involvement of staff at all levels
4. Focus on the individual which create intermediate and deliverable products and services
5. Focus on interrelation of processes that deliver products and services
6. Continuous process improvement
7. Decision-making based on factual evidence
8. Mutually beneficial relationships with suppliers



These principles are applied through cycles which involve the following activities (numbers in yellow circles map to numbered items below):

1. determining the needs and expectations of the customer;

2. establishing a *quality policy*, that is, a framework which allows the organization's objectives in relation to quality to be defined;

3. design of the *processes* which will create the products (or deliver the services) which will have the qualities implied in the organization's quality objectives;

4. allocation of the responsibilities for meeting these requirements for each element of each process;

5. ensuring resources are available to execute these processes properly;

6. design of methods for measuring the effectiveness and efficiency of each process in contributing to the organization's quality objectives;

7. gathering of measurements;

8. . identification of any discrepancies between the actual measurements and the target values;

9. analysis and elimination of the causes of discrepancies.

The need to improve

- can everything be improved at one?

  no, must tackle the most important things first

- 'poor companies are poor at changing'

  some later improvements build on earlier ones

- but there are problems
  - *improvement takes up time and money*
  - *'improvement' may simply be more bureaucracy!*

Capability maturity model (CMM)

- Created by Software Engineering Institute, Carnegie Mellon University
- CMM developed by SEI for US government to help procurement
- The rationale was:
  - Include likely contractor performance as a factor in contract awards.
- Watts S. Humphrey 'Managing the software process' Addison Wesley
- Assessment is by questionnaire and interview

Over the years different version of CMM were devised for different environments, but more recently there has been an attempt to reintegrate these disparate models into one generic, yet flexible, model called CMM Integration (CMMI)

Capability maturity model 2

- Different versions have been developed for different environments e.g. software engineering
- New version CMMI tries to set up a generic model which can be used for different environments

SEI Capability Maturity Model

- Can be used in two ways:
  - Capability evaluation
  - Software process assessment.

Capability Evaluation

- Provides a way to assess the software process capability of an organization:
  - Helps in selecting a contractor
  - Indicates the likely contractor performance.

Software Process Assessment

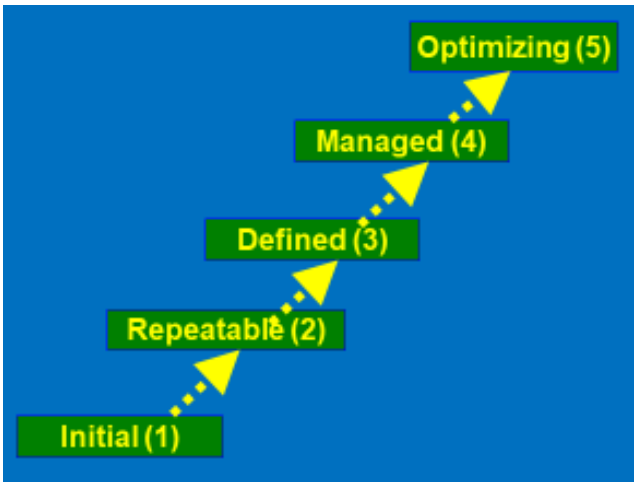
- Used by an organization to assess its current process:
  - Suggests ways to improve the process capability.
- This type of assessment is for purely internal use.

What is CMM?

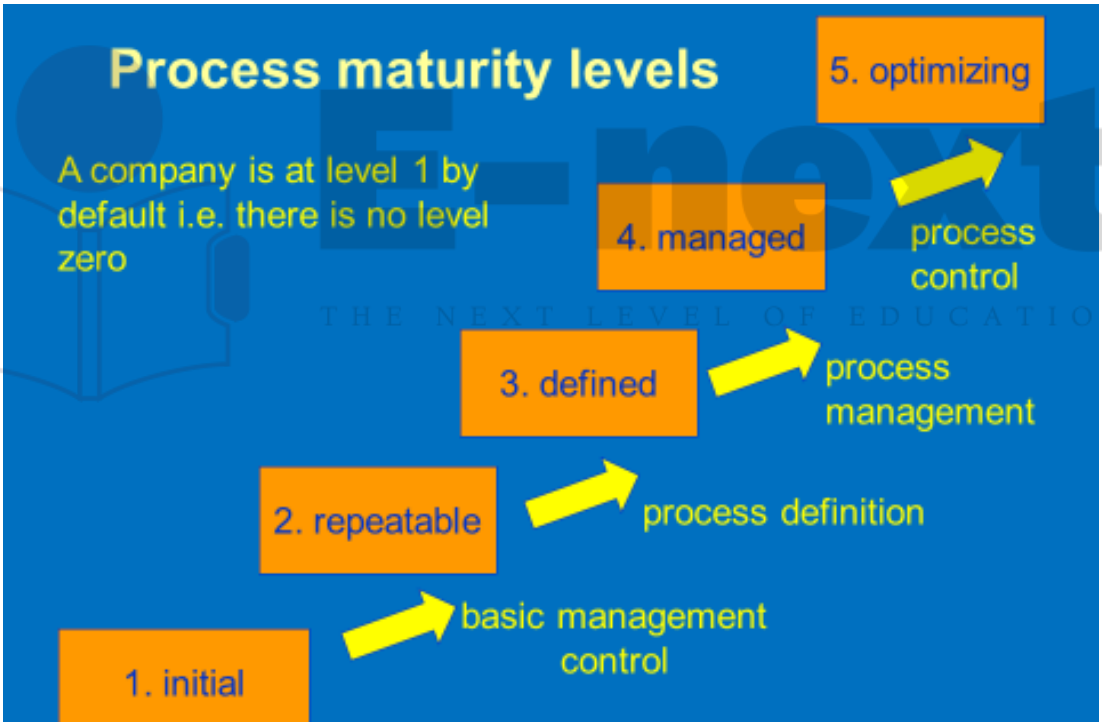
- Describes an evolutionary improvement path for software organizations from an ad hoc immature process :
  - To a mature, disciplined one.
- Provides guidance on:
  - How to control the process

- How to evolve the process

CMM Maturity Levels



- Five maturity levels:
- Stages are ordered so that improvements at one stage provide foundations for the next.
- Based on the pioneering work of Philip Crosby.



CMM Level 1 (Initial)

- Organization operates Without any formalized process or project plans
- An organization at this level is characterized by Ad hoc and chaotic activities.
- Software development processes are not defined,
- Different developers follow their own process

- The success of projects depend on individual efforts and heroics.

Level 2 (Repeatable)

- Basic project management practices are followed
  - Size and cost estimation techniques:
  - Function point analysis, COCOMO, etc.
  - Tracking cost, schedule, and functionality.
- Development process is ad hoc:
  - Not formally defined
  - Also not documented.

Level 3 (Defined)

- All management and development activities:
  - Defined and documented.
  - Common organization-wide understanding of activities, roles, and responsibilities.
- The process though defined:
  - Process and product qualities are not measured.

Level 4 (Managed)

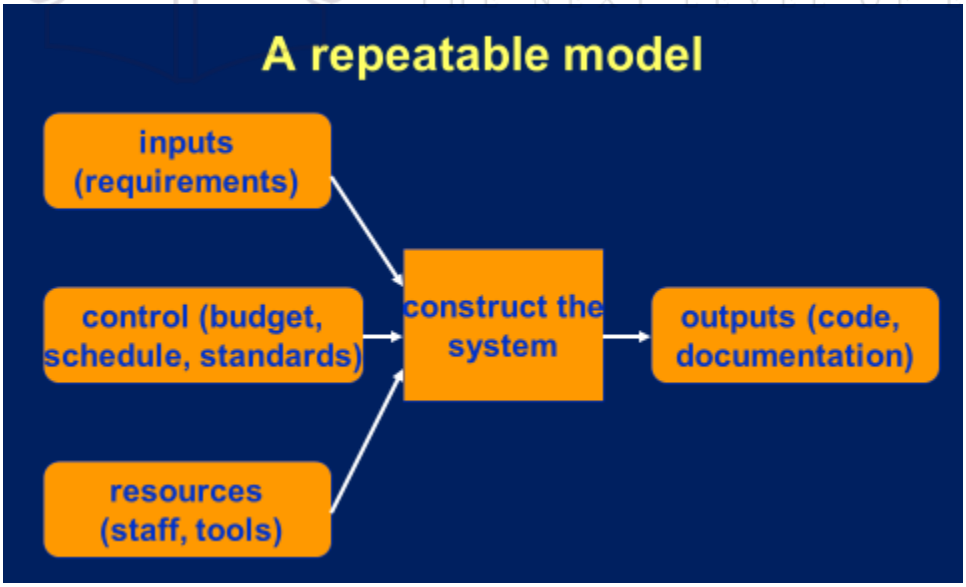
- Quantitative quality goals for products are set.
- Software process and product quality are measured:
  - The measured values are used to control the product quality.
- Results of measurement used to evaluate project performance:
  - Rather than improve process.
- Detailed measures of the software process and product quality are collected.
- Both the software process and products are quantitatively understood and controlled.

Level 5 (Optimizing)

- Statistics collected from process and product measurements are analyzed:
  - Continuous process improvement based on the measurements.
- Known types of defects are prevented from recurring by tuning the process
- Lessons learned from specific projects incorporated into the process

Key process areas

- The KPAs of a level indicate the areas that an organization at the lower maturity level needs to focus to reach this level.
- KPAs provide a way for an organization to gradually improve its quality of over several stages.
- KPAs for each stage has been carefully designed such that one stage enhances the capability already built up.
  - Trying to focus on some higher level KPAs without achieving the lower level KPAs would be counterproductive.



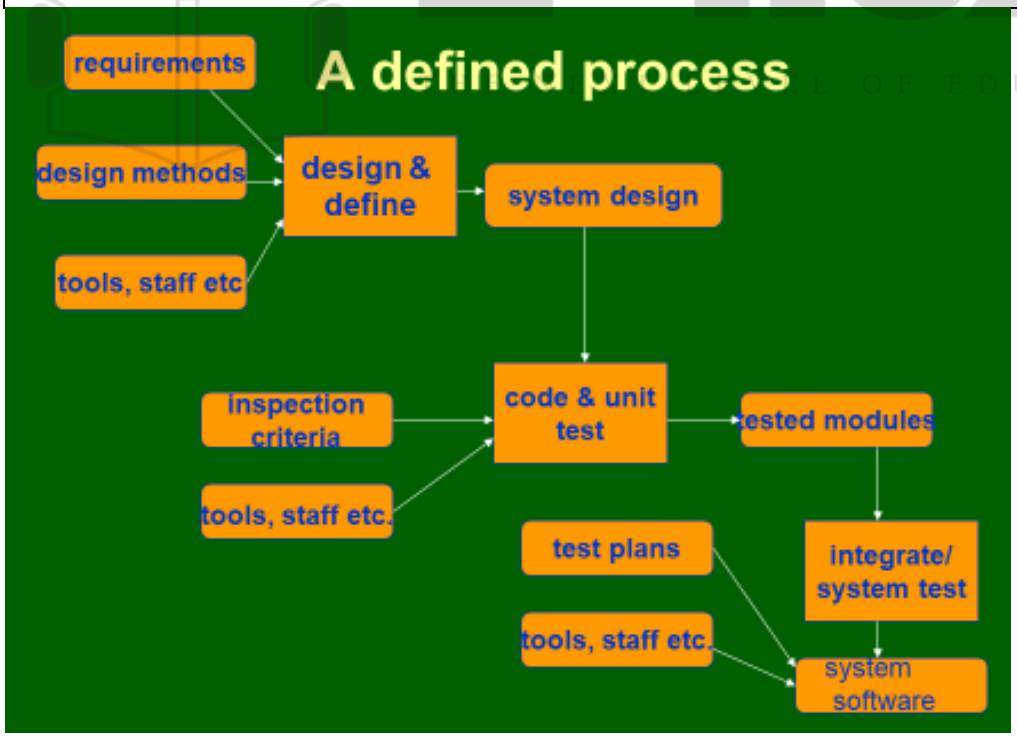
The idea is that the process is under 'statistical control'. Essentially this means that you are collecting data about the performance of the project. This allows you to plan future projects more accurately because, for example, you know approximately how long it will take you to do a new project because you have data on the productivity rates experienced by project teams doing similar work in the past.

Repeatable model KPAs

To move to this level concentrate on:

- Configuration management
- Quality assurance
- Sub-contract management
- Project planning
- Project tracking and oversight
- Measurement and analysis

- KPAs are Key Process areas.
- **Configuration management** – this is the creation and maintenance of a database of the products being created by the project. This enables the project team, among other things, to ensure that when a change is made to one product, e.g. a design document, other related documents, e.g. test cases and expected results are updated.
- **Quality assurance –** this involves setting up a group whose job it is to check that people are following the laid-down quality procedures



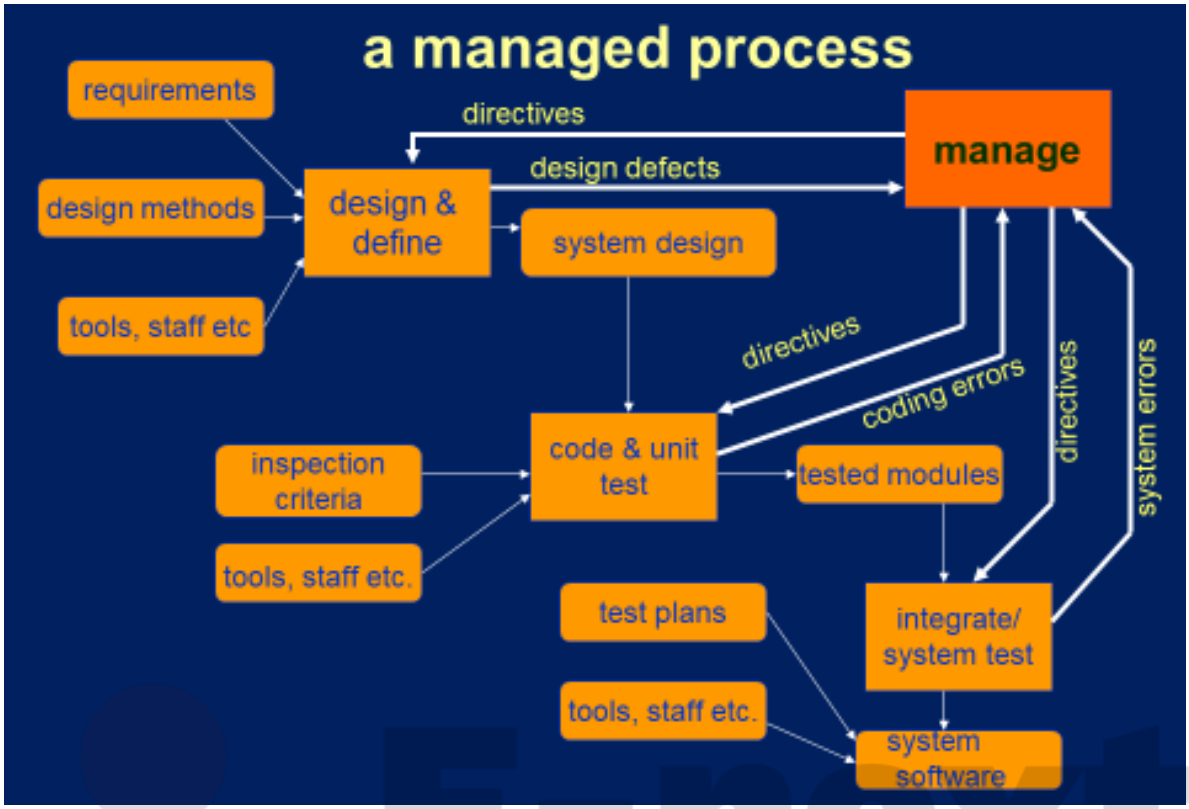
Management at the previous level tends to see the project as a whole. At this level an attempt is made to breakdown the project into a set of component sub-activities and to make the progress and effectiveness of these sub-activities visible to the wider world. Sub-activities are broken down into even lower level activities until you get to activities carried out by individuals or small teams.

Repeatable to defined KPAs

Concentrate on

- Requirements development and technical solution
- Verification and validation
- Product integration
- Risk management
- Organizational training
- Organizational process focus (function)
- Decision analysis and resolution
- Process definition
- Integrated project management

1. **Requirements analysis -** multi-stakeholder requirements evolution
2. **Technical solution –** evolutionary design and quality engineering
3. **Product integration –** continuous integration, interface control, change management
4. **Verification:** assessment to ensure that the product is produced correctly
5. **Validation:** assessment to ensure that the right product is created
6. **Risk management**
7. **Organizational training**
8. **Organizational process focus –** establishing an organizational framework for process definition
9. **Decision analysis and resolution –** systematic alternative assessment
10. **Process definition** treatment of process as a persistent, evolving asset of the organization
11. **Integrated project management** Methods for unifying the various teams and stakeholders within a project.

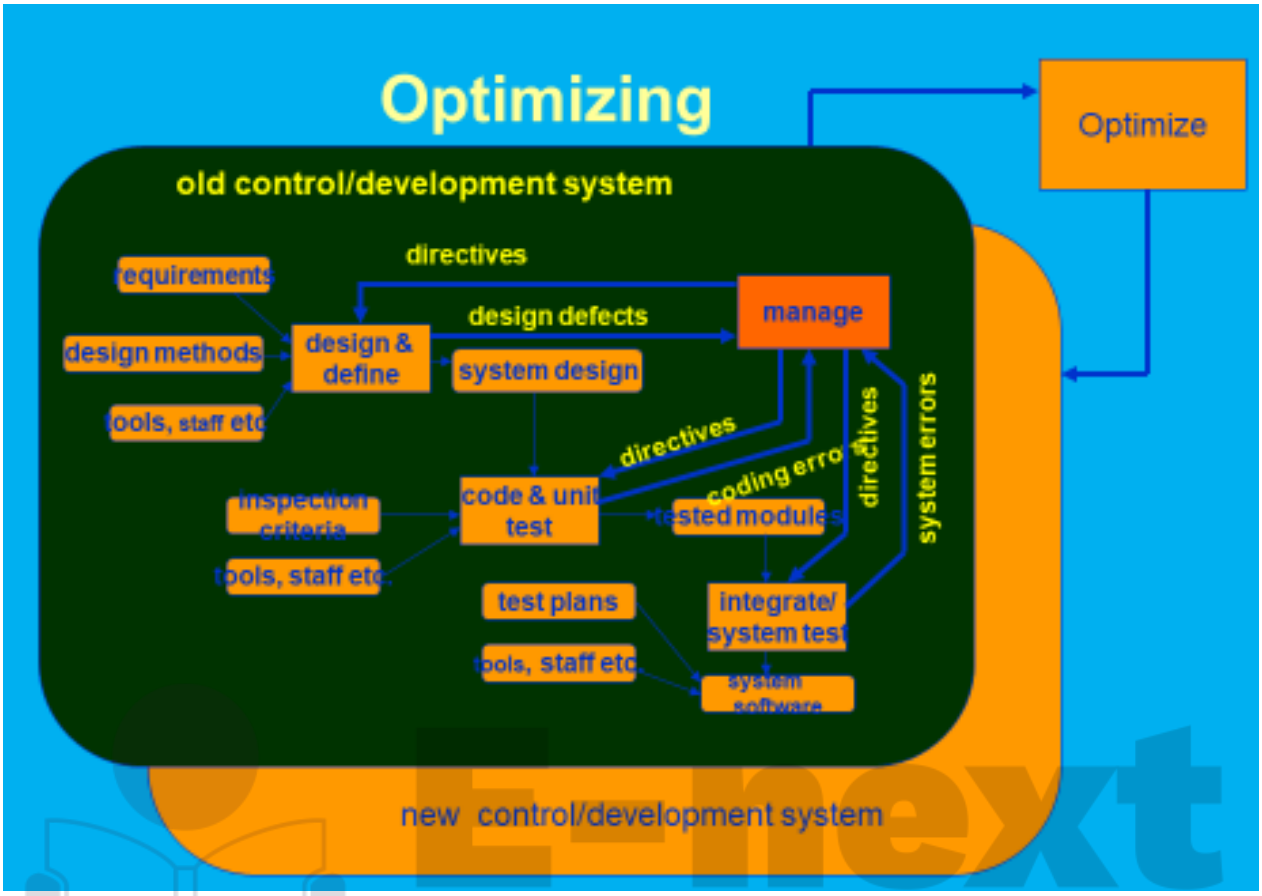


Now that you have identified individual sub-processes and are collecting data about them, you can carry management interventions as problems start to emerge and thus stop the problems evolving into major set-backs.

Defined to managed KPAs

Concentrate on:

- Organizational process performance
- Quantitative project management



You are now in a position not only to deal with problems and quality short-falls as they emerge in a project; you can look at the processes you have designed and identify defects in them that lead to deficient products. Actions can be taken to remove the source of deficient products, not just the deficient products themselves.

Managing to optimizing: KPAs

Concentrate on:

- Causal analysis and resolution
- Organizational innovation and deployment

CMMI (Capability Maturity Model Integration)

- CMMI is the successor of the Capability Maturity Model (CMM).
- After CMMI was first released in 1990:
  - It became popular in many other domains
  - Human Resource Management (HRM).: people management (PCMM)

- software acquisition (SA-CMM)
- systems engineering (SE-CMM)

Some questions about CMMI

- Suitable only for large organizations?
  - e.g. need for special quality assurance and process improvement groups
- Defining processes may not be easy with new technology
  - how can we plan when we've not used the development method before?
- Higher CMM levels easier with maintenance environments?
- Can you jump levels?

ISO/IEC 15504 IT process assessment

- To provide guidance on the assessment of software development processes
- **Process Reference Model**: Needs a defined set of processes that represent good practice to be the benchmark
- **ISO 12207** is the default reference model
- Could use others in specific environments

It can be argued that CMMI can be seen as a particular implementation of ISO 15504. There is however some controversy over this.

ISO 15504 performance attributes

CMMI level	ISO 15504
	0. incomplete
initial	1.1.process performance – achieves defined outcome
repeatable	2.1 process management – it is planned and monitored
	2.2 work product management – control of work products

CMMI	ISO 15504
Defined	3.1. Process definition
	3.2. Process deployment
Managed	4.1. Process measurement
	4.2. Process control
Optimizing	5.1. Process innovation
	5.2. Process optimization

At Level 3 ISO 15504 distinguishes between the definition of all the processes in the development life cycle and the process or educating and managing staff to adhere to the 'best practice' processes.

Similarly at Level 4 it implies that full control of the process cannot be in place until the process is being properly and consistently measured

Level 5 Process innovation – as a result of the data collected in 4.1, opportunities for process improvements are identified Process optimization – the opportunities for process improvement are properly evaluated and where appropriate are effectively implemented.

Process Reference Model

- A defined standard approach to development
- Reflects recognized good practice
- A benchmark against which the processes to be assessed can be judged
- ISO 12207 is the default model

The Process Reference model is the benchmark against which a set of processes is to be assessed. For example, ISO 12207 specifies the features of an acceptable software requirements document. Among the things that this must produce are included '*Functional and capability specifications, including performance, physical characteristics, and environmental conditions under which the software item is to perform*'. My interpretation here is that this includes ensuring the software will execute correctly when given any of the inputs that might be expected in the environment in which it will operate.

ISO 15504 performance indicators

This is just an example of how indicators for each level *might* be identified

1.Performance

Descriptions of maximum and minimum expected input values exist

2.1 Performance management

A plan of how expected input variable ranges are to be obtained exists which is up to date

2.2 Work product management

There are minutes of a meeting where the input requirements document was reviewed and corrections were mandated

3.1 Process definition

A written procedure for input requirements gathering exists

3.2 Process deployment

A control document exists that is signed as each part of the procedure is completed

4.1. Process measurement

Collected measurement data can be collected e.g. number of changes resulting from review

4.2. Process control

Memos relating to management actions taken in the light of the above

5.1 Process innovation

Existence of some kind of 'lessons learnt' report at the end of project

5.2. Process optimization

Existence of documents assessing the feasibility of suggested process improvements and which show consultation with relevant stakeholders

Techniques to improve quality –Inspections

- When a piece of work is completed, copies are distributed to co-workers
- Time is spent individually going through the work noting defects
- A meeting is held where the work is then discussed
- A list of defects requiring re-work is produced

Inspections are a simple but very effective methods of removing errors from documents. Software code could be seen as just another type of document in this context.

Inspections - advantages of approach

- An effective way of removing superficial errors from a piece of software
- Motivates the software developer to produce better structured and self-descriptive code
- Spreads good programming practice
- enhances team-spirit
- *The main problem* maintaining the commitment of participants

'Clean-room' software development

Ideas associated with Harlan Mills at IBM

- Three separate teams:

1. Specification team – documents user requirements and usage profiles (how much use each function will have)
2. Development team – develops code but does not test it. Uses mathematical verification techniques
3. Certification team – tests code. Statistical model used to decide when to stop

- The key idea here is to avoid developers building perhaps crudely coded software components which they turn into working system by a process of throwing tests at it and making changes. Claimed that it is impossible to produce fault-free code this way – rather they should used structured approaches and formal mathematical notations and demonstrate software will work by logical verification.
- Note that usage profiles mentioned on the slide are explored further in the section on testing.

Formal methods

- Use of mathematical notations such as VDM and Z to produce unambiguous specifications
- Can prove correctness of software mathematically (cf. geometric proofs of Pythagoras' theorem)
- Newer approach use Object Constraint Language (OCL) to add detail to UML models
- Aspiration is to be able to generate applications directly from UML+OCL without manual coding – Model Driven Architectures (MDA)

Model Driven Architecture is the complete opposite to extreme programming!

Quality plans

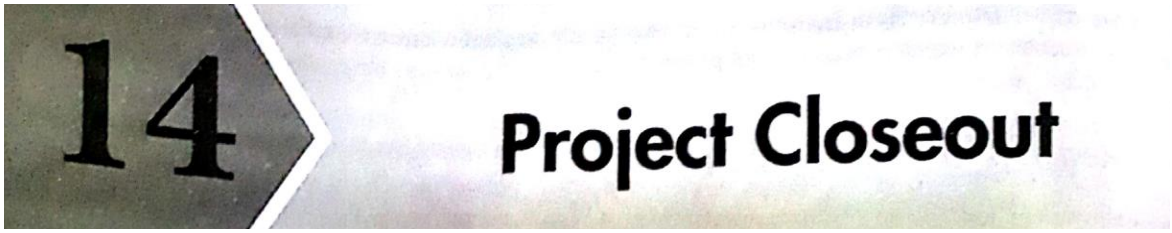
- quality standards and procedures should be documented in an organization's *quality manual*
- for each separate project, the quality needs should be assessed
- select the level of quality assurance needed for the project and document in a *quality plan*

Typical contents of a quality plan

- scope of plan
- references to other documents
- quality management, including organization, tasks, and responsibilities
- documentation to be produced
- standards, practices and conventions
- reviews and audits

more contents of a quality plan

- testing
- problem reporting and corrective action
- tools, techniques, and methodologies
- code, media and supplier control
- records collection, maintenance and retention
- training
- risk management



Project Closeout

- Every project must come to an end sometime or other
  - It is the responsibility of the project manager to decide the appropriate time to close a project.
- Project closeout activities:
  - administrative closure
  - Contract closure

- Increasing criticality of software – e.g .software is increasingly being used in systems that can threaten or support human life and well-being
- The intangibility of software – it is difficult for observers to judge the quality of software development, especially during its early stages
- Project control concerns:
- The products of one sub-process in the development process are the inputs to subsequent sub-processes, thus
- errors accumulate with each stage e.g. at the design stage, the specification errors are incorporated into the design, and at the coding stage specification and design errors are incorporated into the software
- errors become more expensive to remove the later they are found
- it is difficult to control the error removal process (e.g. testing)
- See Section 13.3

Types of Project Closure

- Two main types:
  - Normal termination: All the project goals have been successfully accomplished.
  - Premature termination: the project is unlikely to achieve its stated objectives --- This is the case for about a third of all projects

Premature Termination

- There are many reasons as to why a project may have to be prematurely terminated:
  - *Lack of resources*
  - *Changed business need of the customer*
  - *perceived benefits accruing from the project no longer remain valid*
  - *Changes to the regulatory policies*
  - *Key technologies used in the project becoming obsolete during project execution*
  - *Risks have become unacceptably high:*

Why are projects not properly closed?

- Often projects are not properly closed:
  - *Lack of interest by the project team*
  - *Underestimation of how fast know-how can get lost and how much implicit knowledge exists with the team members*
  - *Emotional factors*
  - *Indecision regarding project closure:*

Problems of Improper project closure

- *Time and cost overrun:*
  - project as a cost centre runs up expenditure
- *Locks up valuable human and other resources*
  - Redeployment of project personnel and other resources gets delayed
- *Stress on the project personnel:*

Issues associated with project termination

- Two fold:
  - Emotional

- Intellectual

- Team members may pay more attention to issues such as getting reassigned to a project of their choice and the project work can take a back seat
- Terms of contract and the list of deliverables need to be renegotiated
- Closure decision has to be effectively communicated to all stakeholders.

Project Closure Process

- Getting client acceptance
- Archiving project deliverables
- Preserving project know-how
- Performing a financial closure
- Performing postImplementation project review
- Preparing postimplementation review report
- Releasing staff

Post-implementation project review

- Conduct project survey.
- Collect objective information.
- Hold a debriefing meeting.
- Prepare post-implementation review report.
- Publish the report.

Project survey

- *Project performance*
- *Administrative performance*
- *Organizational structure*
- *Team performance*
- *Techniques of project management*

- *Risk management:*

Project Closeout Report

- Project closeout report is intended to provide a concise evaluation of the project.
- Project description: Information about the project, to give context
- What worked well
- The factors that impeded the performance of the project
- A prescription for other projects to follow