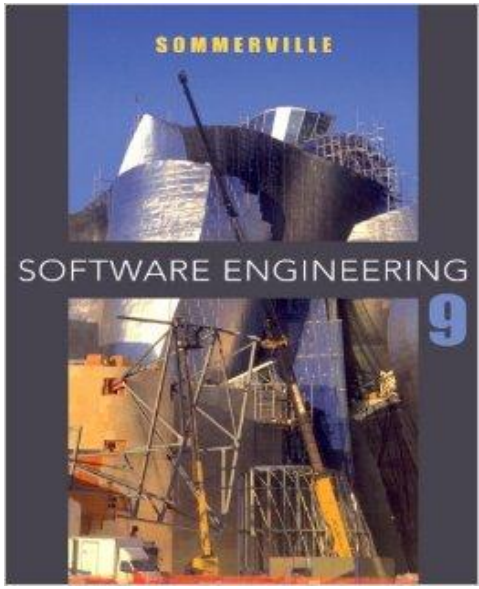
Software Project Management
CS472

Text book : **Software Engineering 9th Ed, Sommerville,**
PEARSONISBN-10 : 0-13-705346-0.



Ms. Reshma Parvez

Outline

- **Overview of Software Project Management**
- **The Management Spectrum**
- **Project Planning and Scheduling**
- **Risk Management**
- **Quality Management**
- **Software Testing Strategies**
- **Process Improvement**

A brief History

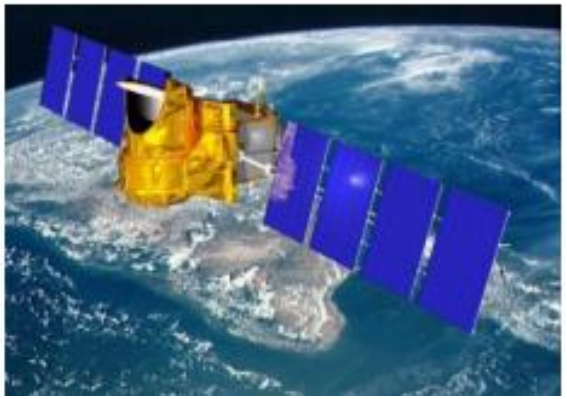
- **Second world war ➔ Repetitive tasks**

A brief History

- **Second world war**
- **Complex tasks, the problem of tasks scheduling**
- **Finding technical solution**

A brief History

- **After the second world war**
  - **Management of space projects**
  - **These methods progressively extend to all other areas**



- **AFITEP-AFNOR (1991): A project is specific approach that methodically and gradually structures a future reality and (..) that implies a goal and needs to tackle on with determinate resources**

Chapter-1

Overview of Software Project Management

WHAT IS A "PROJECT"??

- **A project is well-defined task, which is a collection of several operations done in order to achieve a goal (for example, software development and delivery). A Project can be characterized as:**
- **Every project may has a unique and distinct goal.**
- **Project is not routine activity or day-to-day operations.**
- **Project comes with a start time and end time.**
- **Project ends when its goal is achieved hence it is a temporary phase in the lifetime of an organization.**
- **Project needs adequate resources in terms of time, manpower, finance, material and knowledge-bank.**

WHAT IS SOFTWARE PROJECT ??

- A Software Project is the complete procedure of software development from requirement gathering to testing and maintenance, carried out according to the execution methodologies, in a specified period of time to achieve intended software product.

Definitions

- **Concerned with activities involved in ensuring that software is delivered on time and on schedule and in accordance with the requirements of the organizations developing and procuring the software**
- **Project management is needed because software development is always subject to budget and schedule constraints that are set by the organization developing the software**

Success Criteria

- **Deliver the software to the customer at the agreed time.**
- **Keep overall costs within budget.**
- **Deliver software that meets the customer's expectations.**
- **Maintain a happy and well-functioning development team.**

Software management distinctions

- **The product is intangible**
  - **Software cannot be seen or touched**
  - **Software project managers cannot see progress by simply looking at the artifact that is being constructed**
- **Many software projects are 'one-off' projects**
  - **Large software projects are usually different in some ways from previous projects.**
  - **Managers who have lots of previous experience may find it difficult to anticipate problems**
- **Software processes are variable and organization specific.**
  - **We still cannot reliably predict when a particular software process is likely to lead to development problems**

Management activities

- **Project planning**
  - **Project managers are responsible for planning. estimating and scheduling project development and assigning people to tasks**
- **Project planning**
  - **Project managers are usually responsible for reporting on the progress of a project to customers and to the managers of the company developing the software**
- **Risk management**
  - **Project managers assess the risks that may affect a project, monitor these risks and take action when problems arise**

Management activities

- **People management**
  - **Project managers have to choose people for their team and establish ways of working that leads to effective team performance**
- **Proposal writing**
  - **The first stage in a software project may involve writing a proposal to win a contract to carry out an item of work**
  - **The proposal describes the objectives of the project and how it will be carried out**

Risk management

- **Risk management is concerned with identifying risks and drawing up plans to minimize their effect on a project.**
- **A risk is a probability that some adverse circumstance will occur**
  - **Project risks affect schedule or resources**
  - **Product risks affect the quality or performance of the software being developed**
  - **Business risks affect the organization developing or procuring the software.**

Examples of common project, product, and business risks

Risk	Affects	Description
Staff turnover	Project	Experienced staff will leave the project before it is finished.
Management change	Project	There will be a change of organizational management with different priorities.
Hardware unavailability	Project	Hardware that is essential for the project will not be delivered on schedule.
Requirements change	Project and product	There will be a larger number of changes to the requirements than anticipated.
Specification delays	Project and product	Specifications of essential interfaces are not available on schedule.
Size underestimate	Project and product	The size of the system has been underestimated.
CASE tool underperformance	Product	CASE tools, which support the project, do not perform as anticipated.
Technology change	Business	The underlying technology on which the system is built is superseded by new technology.
Product competition	Business	A competitive product is marketed before the system is completed.

What is software engineering?

- **Software engineering is an engineering discipline that is concerned with all aspects of software production.**
- **Software engineers should adopt a systematic and organised approach to their work and use appropriate tools and techniques depending on the problem to be solved, the development constraints and the resources available.**

engineering and computer science?

- **Computer science is concerned with theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software.**
- **Computer science theories are still insufficient to act as a complete underpinning for software engineering (unlike e.g. physics and electrical engineering).**

engineering and system engineering?

- **System engineering is concerned with all aspects of computer-based systems development including hardware, software and process engineering. Software engineering is part of this process concerned with developing the software infrastructure, control, applications and databases in the system.**
- **System engineers are involved in system specification, architectural design, integration and deployment.**

Question	Answer
What is software?	Computer programs and associated documentation. Software products may be developed for a particular customer or may be developed for a general market.
What are the attributes of good software?	Good software should deliver the required functionality and performance to the user and should be maintainable, dependable, and usable.
What is software engineering?	Software engineering is an engineering discipline that is concerned with all aspects of software production.
What are the fundamental software engineering activities?	Software specification, software development, software validation, and software evolution.
What is the difference between software engineering and computer science?	Computer science focuses on theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software.
What is the difference between software engineering and system engineering?	System engineering is concerned with all aspects of computer-based systems development including hardware, software, and process engineering. Software engineering is part of this more general process.
What are the key challenges facing software engineering?	Coping with increasing diversity, demands for reduced delivery times, and developing trustworthy software.
What are the costs of software engineering?	Roughly 60% of software costs are development costs; 40% are testing costs. For custom software, evolution costs often exceed development costs.
What are the best software engineering techniques and methods?	While all software projects have to be professionally managed and developed, different techniques are appropriate for different types of system. For example, games should always be developed using a series of prototypes whereas safety critical control systems require a complete and analyzable specification to be developed. You can't, therefore, say that one method is better than another.
What differences has the Web made to software engineering?	The Web has led to the availability of software services and the possibility of developing highly distributed service-based systems. Web-based systems development has led to important advances in programming languages and software reuse.

Chapter- 2

The Management Spectrum

Managing people

- **People are an organisation's most important assets.**
- **The tasks of a manager are essentially people-oriented. Unless there is some understanding of people, management will be unsuccessful.**
- **Poor people management is an important contributor to project failure.**

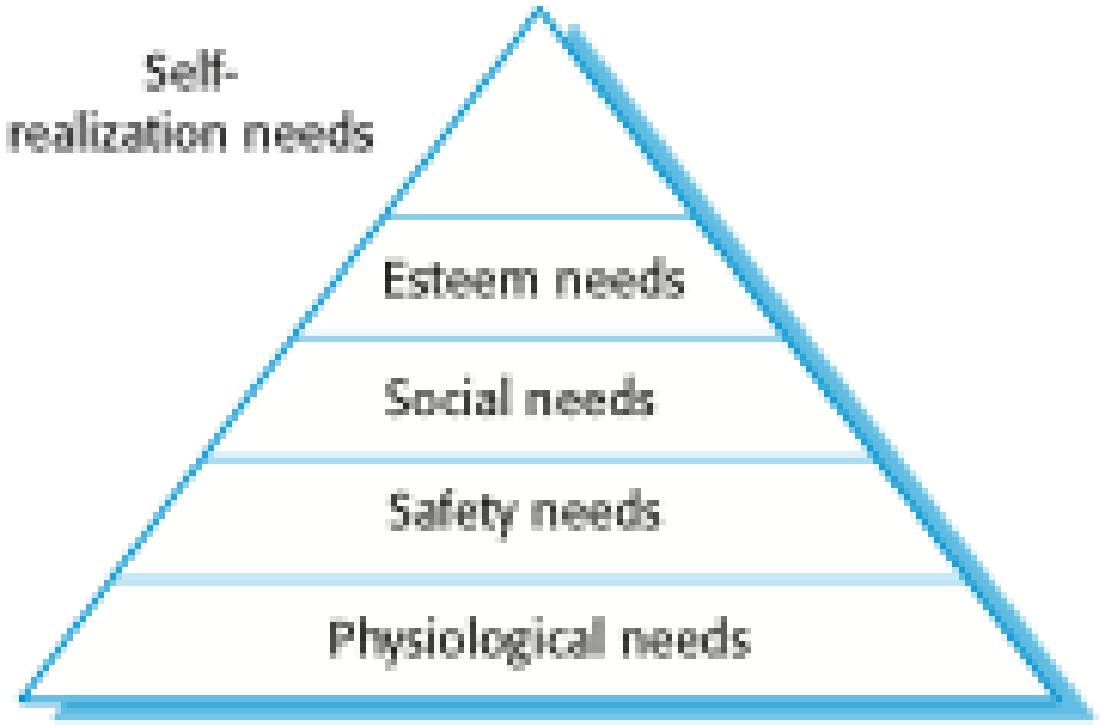
People management factors

- **Consistency**
  - **Team members should all be treated in a comparable way without favourites or discrimination.**
- **Respect**
  - **Different team members have different skills and these differences should be respected.**
- **Inclusion**
  - **Involve all team members and make sure that people's views are considered.**
- **Honesty**
  - **You should always be honest about what is going well and what is going badly in a project.**

Motivating people

- **An important role of a manager is to motivate the people working on a project**
- **Motivation means organizing the work and the working environment to encourage people to work effectively**
- **Motivation is a complex issue but it appears that their are different types of motivation based on:**
  - **Basic needs (e.g. food, sleep, etc.);**
  - **Personal needs (e.g. respect, self-esteem);**
  - **Social needs (e.g. to be accepted as part of a group).**

Human needs hierarchy



Need satisfaction

- **In software development groups, basic physiological and safety needs are not an issue.**
- **Social**
  - **Provide communal facilities;**
  - **Allow informal communications e.g. via social networking**
- **Esteem**
  - **Recognition of achievements**
  - **Appropriate rewards**
- **Self-realization**
  - **Training - people want to learn more;**
  - **Responsibility.**

Personality types

- **The needs hierarchy is almost certainly an over-simplification of motivation in practice.**
- **Motivation should also take into account different personality types:**
  - **Task-oriented;**
  - **Self-oriented;**
  - **Interaction-oriented.**

Personality types

- **Task-oriented**
  - **The motivation for doing the work is the work itself;**
- **Self-oriented**
  - **The work is a means to an end which is the achievement of individual goals - e.g. to get rich, to play tennis, to travel etc.;**
- **Interaction-oriented**
  - **The principal motivation is the presence and actions of co-workers. People go to work because they like to go to work.**

Personality types

- **Task-oriented**
  - **The motivation for doing the work is the work itself;**
- **Self-oriented**
  - **The work is a means to an end which is the achievement of individual goals - e.g. to get rich, to play tennis, to travel etc.;**
- **Interaction-oriented**
  - **The principal motivation is the presence and actions of co-workers. People go to work because they like to go to work.**

Motivation balance

- **Individual motivations are made up of elements of each class.**
- **The balance can change depending on personal circumstances and external events.**
- **However, people are not just motivated by personal factors but also by being part of a group and culture.**
- **People go to work because they are motivated by the people that they work with.**

Teamwork

- **Most software engineering is a group activity**
  - **The development schedule for most non-trivial software projects is such that they cannot be completed by one person working alone.**
- **A good group is cohesive and has a team spirit. The people involved are motivated by the success of the group as well as by their own personal goals.**
- **Group interaction is a key determinant of group performance.**
- **Flexibility in group composition is limited**
  - **Managers must do the best they can with available people.**

Teamwork

- **Most software engineering is a group activity**
  - **The development schedule for most non-trivial software projects is such that they cannot be completed by one person working alone.**
- **A good group is cohesive and has a team spirit. The people involved are motivated by the success of the group as well as by their own personal goals.**
- **Group interaction is a key determinant of group performance.**
- **Flexibility in group composition is limited**
  - **Managers must do the best they can with available people.**

Group organization

- **The way that a group is organized affects the decisions that are made by that group**
- **The ways that information is exchanged and the interactions between the development group and external project stakeholders**
- **Key questions include:**
  - **Should the project manager be the technical leader of the group?**
  - **Who will be involved in making critical technical decisions, and how will these be made?**
  - **How will interactions with external stakeholders and senior company management be handled?**
  - **How can groups integrate people who are not co-located?**
  - **How can knowledge be shared across the group?**

Group organization

- **Small software engineering groups are usually organized informally without a rigid structure.**
- **For large projects, there may be a hierarchical structure where different groups are responsible for different sub-projects.**
- **Agile development is always based around an informal group on the principle that formal structure inhibits information exchange**

Informal groups

- **The group acts as a whole and comes to a consensus on decisions affecting the system.**
- **The group leader serves as the external interface of the group but does not allocate specific work items.**
- **Rather, work is discussed by the group as a whole and tasks are allocated according to ability and experience.**
- **This approach is successful for groups where all members are experienced and competent.**

Group communications

- **Good communications are essential for effective group working.**
- **Information must be exchanged on the status of work, design decisions and changes to previous decisions.**
- **Good communications also strengthens group cohesion as it promotes understanding.**

Group communications

- **Group size**
  - **The larger the group, the harder it is for people to communicate with other group members**
- **Group structure**
  - **Communication is better in informally structured groups than in hierarchically structured groups.**
- **Group composition**
  - **Communication is better when there are different personality types in a group**
- **The physical work environment**
  - **Good workplace organisation can help encourage communications**

Chapter- 3

Project Planning and Scheduling

Project planning

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

Project stages

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

Software pricing

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

Factors affecting software pricing

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

Factors affecting software pricing

Factor	Description
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.

Plan-driven development

- **Plan-driven or plan-based development is an approach to software engineering where the development process is planned in detail**
  - **Plan-driven development is based on engineering project management techniques and is the 'traditional' way of managing large software development projects.**
- **A project plan is created that records the work to be done, 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.**

Plan-driven development – pros and cons

☺ **early planning allows organizational issues (availability of staff, other projects, etc.) to be closely taken into account**

☺ **Potential problems and dependencies are discovered before the project starts, rather than once the project is underway.**

☹ **Many early decisions have to be revised because of changes to the environment in which the software is to be developed and used.**

Project Plan

- **A project plan sets out the resources available to the project, the work breakdown and a schedule for carrying out the work.**
- **Plan sections**
  - **Introduction**
  - **Project organization**
  - **Risk analysis**
  - **Hardware and software resource requirements**
  - **Work breakdown**
  - **Project schedule**
  - **Monitoring and reporting mechanisms**

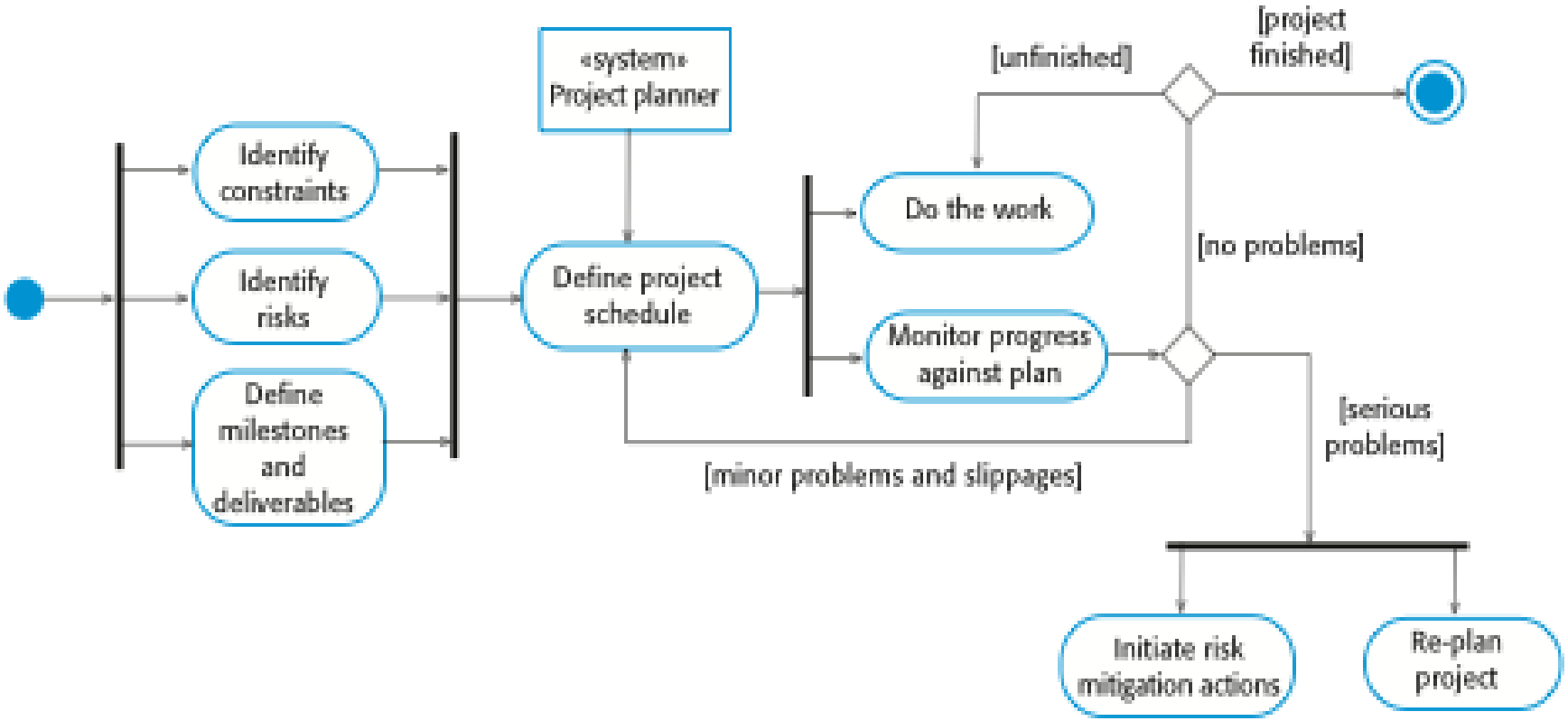
Project Plan

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

The planning process

- **Project planning is an iterative process that starts when you create an initial project plan during the project startup phase**
- **Plan changes are inevitable**
  - **Regular revision of the plan to reflect requirements, schedule and risk changes.**
  - **Changing business goals also leads to changes in project plans**

The project planning process



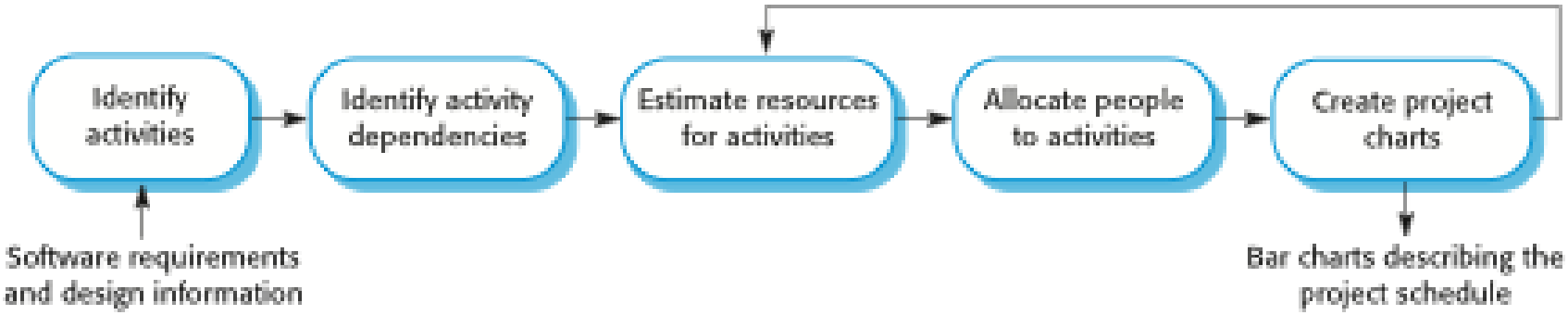
Project scheduling

- **Project scheduling is the process of deciding how the work in a project will be organized as separate tasks, and when and how these tasks will be executed.**
- **Estimating the calendar time needed to complete each task, the effort required and who will work on the tasks that have been identified.**
- **Estimating the resources needed to complete each task**

Milestones and deliverables

- **Milestone 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 project scheduling process



Scheduling problems

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

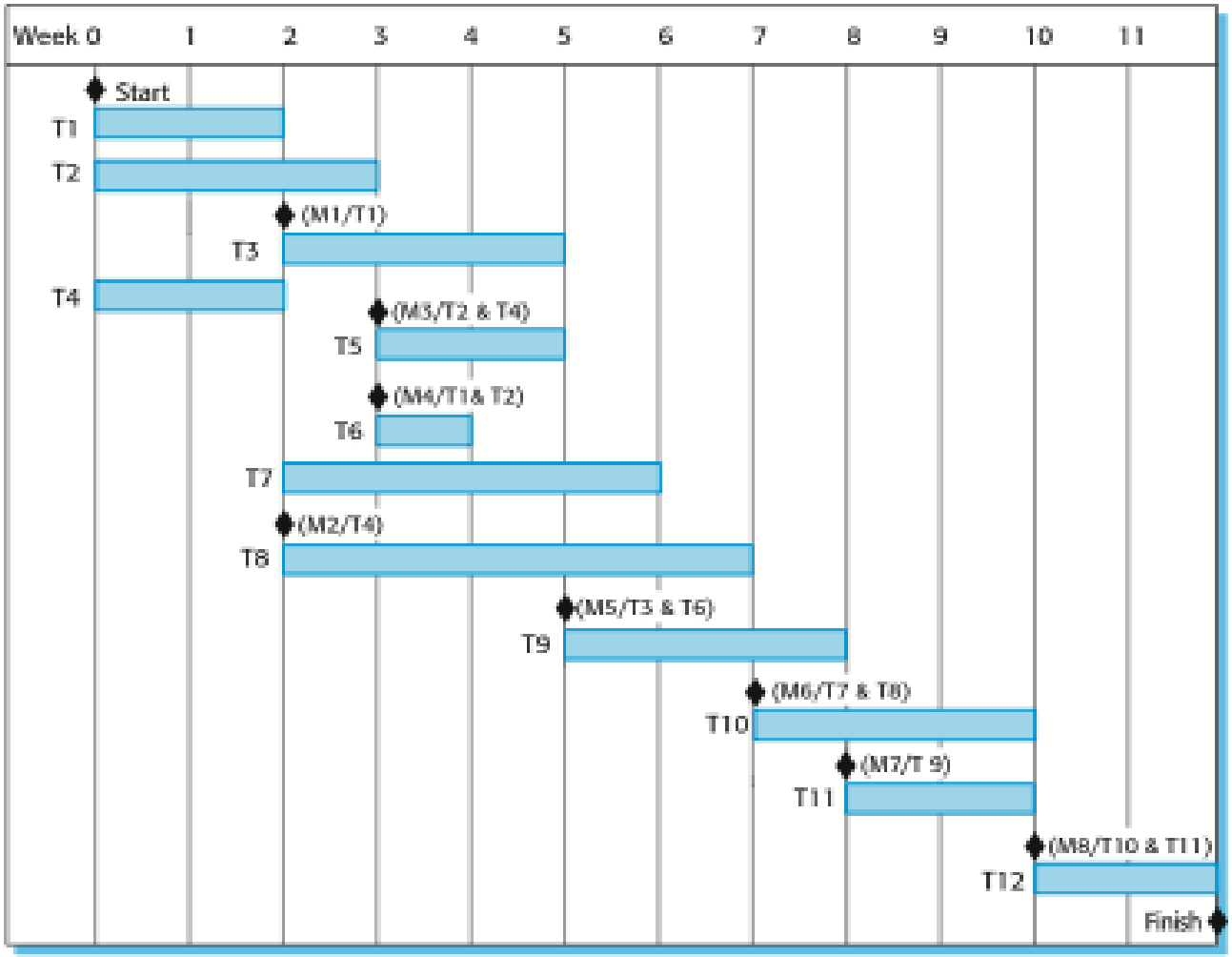
Schedule representation

- **Graphical notations are normally used to illustrate the project schedule**
- **These show the project breakdown into tasks. Tasks should not be too small. They should take about a week or two**
- **Bar charts are the most commonly used representation for project schedules. They show the schedule as activities or resources against time**

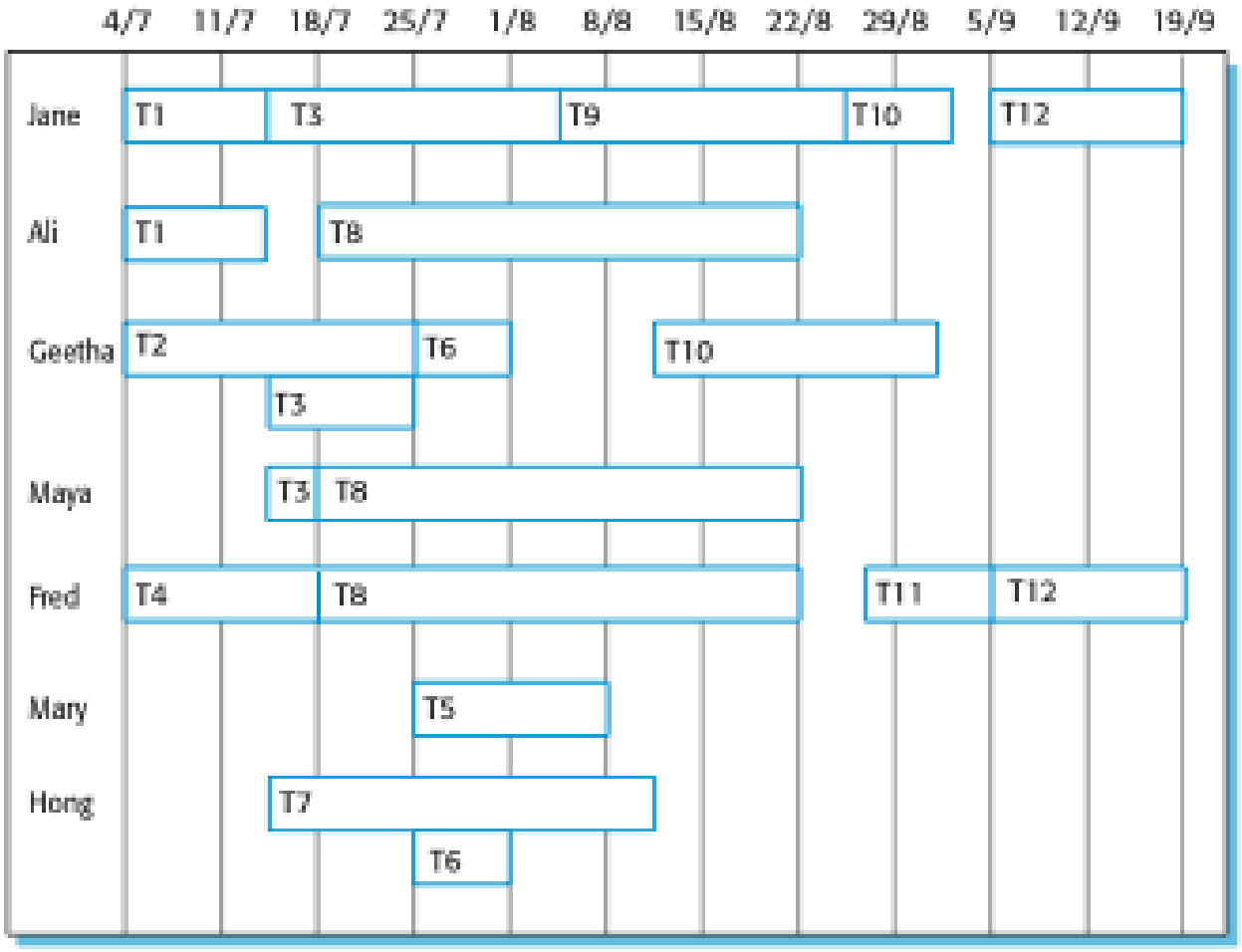
Tasks, durations, and dependencies

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)

Activity bar chart



Staff allocation bar chart



Agile planning

- **Agile methods are iterative approaches where the software is developed and delivered to customers in increments.**
- **The functionality of these increments is not planned in advance but is decided during the development.**
  - **The decision on what to include in an increment depends on progress and on the customer's priorities.**
- **The customer's priorities and requirements change so it makes sense to have a flexible plan that can accommodate these changes.**

Agile planning stages

- **Release planning, which looks ahead for several months and decides on the features that should be included in a release of a system.**
- **Iteration planning, which has a shorter term outlook, and focuses on planning the next increment of a system. This is typically 2-4 weeks of work for the team.**

Agile planning in XP

- **Incremental development is supported through small, frequent system releases.**
- **Customer involvement means full-time customer engagement with the team.**
- **Change supported through regular system releases.**

Story-based planning

- **The system specification in XP is based on user stories that reflect the features that should be included in the system.**
- **The project team read and discuss the stories and rank them in order of the amount of time they think it will take to implement the story.**

Story-based planning

- **Release planning involves selecting and refining the stories that will reflect the features to be implemented in a release of a system and the order in which the stories should be implemented.**
- **Stories to be implemented in each iteration are chosen, with the number of stories reflecting the time to deliver an iteration (usually 2 or 3 weeks).**

Estimation techniques

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

Experience-based approaches

- **Experience-based techniques rely on judgments based on experience of past projects and the effort expended in these projects on software development activities.**
- **Identifying the deliverables to be produced in a project and the different software components or systems that are to be developed.**
- **It usually helps to get a group of people involved in the effort estimation and to ask each member of the group to explain their estimate**

Algorithmic cost modelling

- **Cost is estimated as a mathematical function of product**
  - **Effort = A ´SizeB´M**
  - **A is an organisation-dependent constant, B reflects the disproportionate effort for large projects and M is a multiplier reflecting product, process and people attributes.**
- **The most commonly used product attribute for cost estimation is code size.**
- **Most models are similar but they use different values for A, B and M.**

The COCOMO 2 model

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

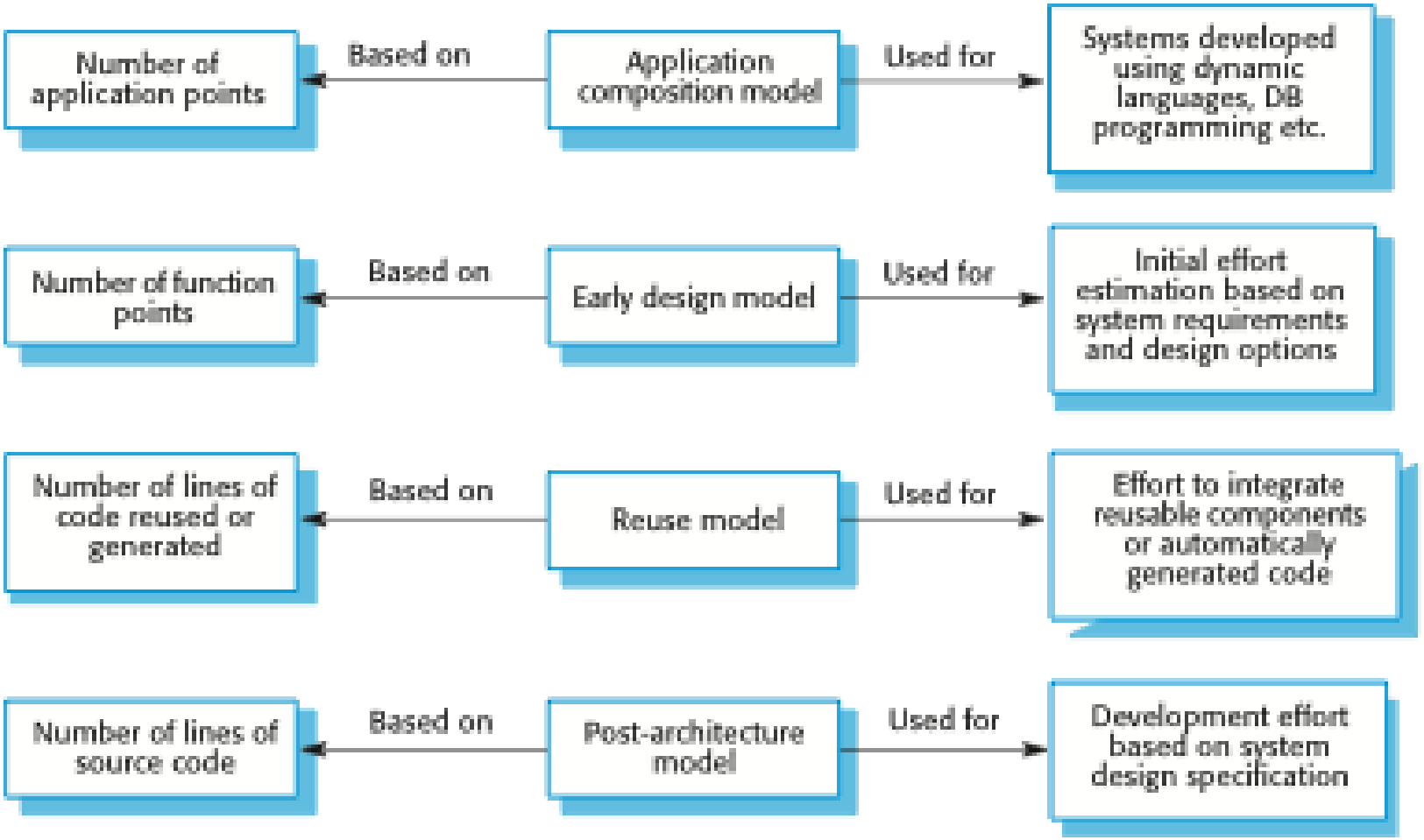
The COCOMO 2 models

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

The COCOMO 2 models

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

The COCOMO estimation model



Application composition model

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

Application-point productivity

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

Early design model

- **Estimates can be made after the requirements have been agreed.**
- **Based on a standard formula for algorithmic models**
  - **PM = A´SizeB´M where**
  - **M = PERS ´ RCPX ´ RUSE ´ PDIF ´ PREX ´ FCIL ´ SCED;**
  - **A = 2.94 in initial calibration, Size in KLOC, B varies from 1.1 to 1.24 depending on novelty of the project, development flexibility, risk management approaches and the process maturity.**

Multipliers

- **Multipliers reflect the capability of the developers, the non-functional requirements, the familiarity with the development platform, etc.**
  - **RCPX - product reliability and complexity;**
  - **RUSE - the reuse required;**
  - **PDIF - platform difficulty;**
  - **PREX - personnel experience;**
  - **PERS - personnel capability;**
  - **SCED - required schedule;**
  - **FCIL - the team support facilities**

The reuse model

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

Reuse model estimates 1

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

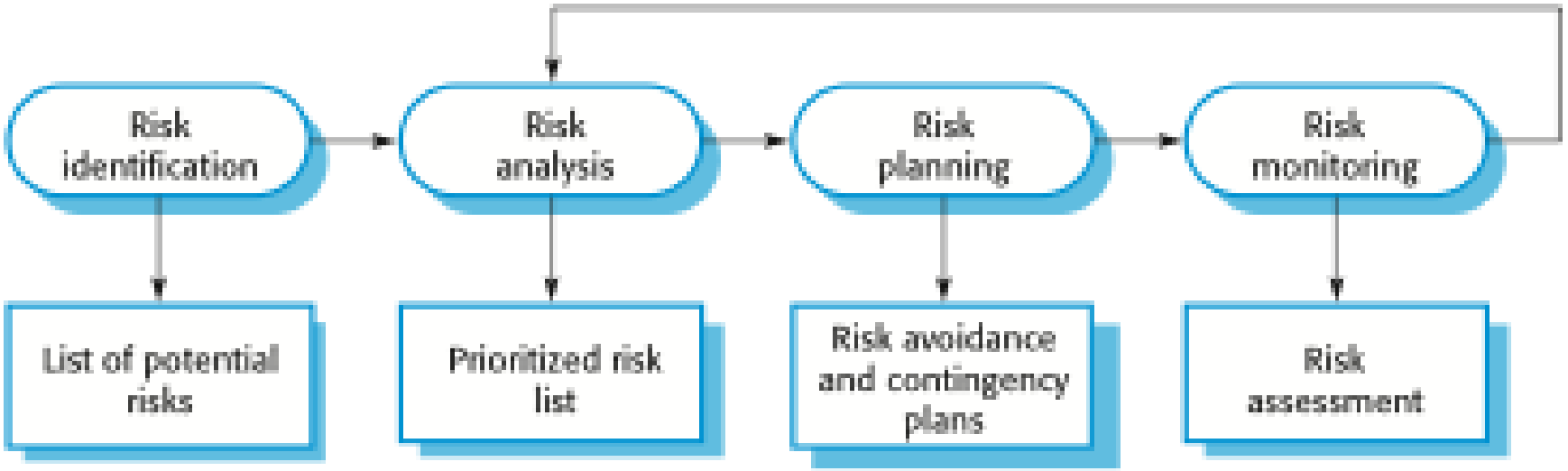
Chapter 4

Risk Management

The risk management process

- **Risk identification: Identify project, product and business risks**
- **Risk analysis: Assess the likelihood and consequences of these risks**
- **Risk planning: Draw up plans to avoid or minimize the effects of the risk**
- **Risk monitoring: Monitor the risks throughout the project**

The risk Management Process



Risk Identification

- **May be a team activities or based on the individual project manager's experience.**
- **A checklist of common risks may be used to identify risks in a project**
  - **Technology risks**
  - **People risks**
  - **Organizational risks**
  - **Requirements risks**
  - **Estimation risks**

Examples of different risk types

Risk type	Possible risks
Technology	**The database used in the system cannot process as many transactions per second as expected. (1)** **Reusable software components contain defects that mean they cannot be reused as planned. (2)**
People	**It is impossible to recruit staff with the skills required. (3)** **Key staff are ill and unavailable at critical times. (4)** **Required training for staff is not available. (5)**
Organizational	**The organization is restructured so that different management are responsible for the project. (6)** **Organizational financial problems force reductions in the project budget. (7)**
Tools	**The code generated by software code generation tools is inefficient. (8)** **Software tools cannot work together in an integrated way. (9)**
Requirements	**Changes to requirements that require major design rework are proposed. (10)** **Customers fail to understand the impact of requirements changes. (11)**
Estimation	**The time required to develop the software is underestimated. (12)** **The rate of defect repair is underestimated. (13)** **The size of the software is underestimated. (14)**

Risk analysis

- **Assess probability and seriousness of each risk.**
- **Probability may be very low, low, moderate, high or very high.**
- **Risk consequences might be catastrophic, serious, tolerable or insignificant Technology risks**

Risk types and examples

Risk	Probability	Effects
Organizational financial problems force reductions in the project budget (7).	Low	Catastrophic
It is impossible to recruit staff with the skills required for the project (3).	High	Catastrophic
Key staff are ill at critical times in the project (4).	Moderate	Serious
Faults in reusable software components have to be repaired before these components are reused. (2).	Moderate	Serious
Changes to requirements that require major design rework are proposed (10).	Moderate	Serious
The organization is restructured so that different management are responsible for the project (6).	High	Serious
The database used in the system cannot process as many transactions per second as expected (1).	Moderate	Serious

Risk planning

- **Consider each risk and develop a strategy to manage that risk.**
- **Avoidance strategies: The probability that the risk will arise is reduced**
- **Minimization strategies: The impact of the risk on the project or product will be reduced**
- **Contingency plans: If the risk arises, contingency plans are plans to deal with that risk**

Strategies to help manage risk

Risk	Strategy
Organizational financial problems	**Prepare a briefing document for senior management showing how the project is making a very important contribution to the goals of the business and presenting reasons why cuts to the project budget would not be cost-effective.**
Recruitment problems	**Alert customer to potential difficulties and the possibility of delays investigate buying-in components.**
Staff illness	**Reorganize team so that there is more overlap of work and people therefore understand each other's jobs.**
Defective components	**Replace potentially defective components with bought-in components of known reliability.**
Requirements changes	**Derive traceability information to assess requirements change impact maximize information hiding in the design.**

Strategies to help manage risk

Risk	Strategy
Organizational restructuring	**Prepare a briefing document for senior management showing how the project is making a very important contribution to the goals of the business.**
Database performance	**Investigate the possibility of buying a higher-performance database.**
Underestimated development time	**Investigate buying-in components investigate use of a program generator.**

Risk Monitoring

- **Assess each identified risks regularly to decide whether or not it is becoming less or more probable**
- **Also assess whether the effects of the risk have changed**
- **Each key risk should be discussed at management progress meetings**

Risk Indicator

Risk type	Potential indicators
Technology	**Late delivery of hardware or support software many reported technology problems.**
People	**Poor staff morale poor relationships amongst team members high staff turnover.**
Organizational	**Organizational gossip lack of action by senior management.**
Tools	**Reluctance by team members to use tools complaints about CASE tools demands for higher-powered workstations.**
Requirements	**Many requirements change requests customer complaints.**
Estimation	**Failure to meet agreed schedule failure to clear reported defects.**

Chapter 5

Quality Management

Software quality management: Three principal concerns:

- **At the organizational level, quality management is concerned with establishing a framework of organizational processes and standards that will lead to high-quality software.**
- **At the project level, quality management involves the application of specific quality processes and checking that these planned processes have been followed.**
- **At the project level, quality management is also concerned with establishing a quality plan for a project.**

Quality management activities

- **Quality management provides an independent check on the software development process.**
- **The quality management process checks the project deliverables to ensure that they are consistent with organizational standards and goals**
- **The quality team should be independent from the development team so that they can take an objective view of the software.**

Quality planning

- **A quality plan sets out the desired product qualities and defines the most significant quality attributes.**
- **The quality plan should define the quality assessment process**
- **Quality plan structure**
  - **Product introduction;**
  - **Product plans;**
  - **Process descriptions;**
  - **Quality goals;**
  - **Risks and risk management.**

Software quality

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

Process and product quality

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

Software standards

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

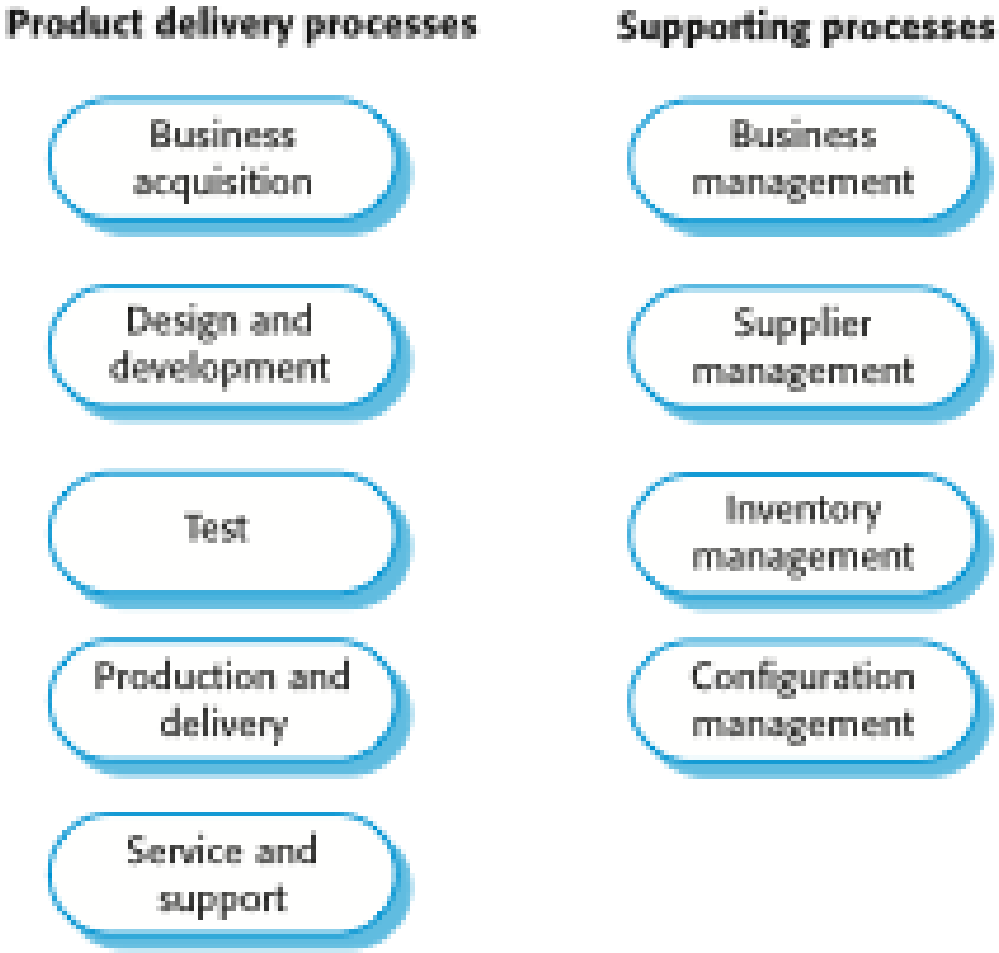
Importance of standards

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

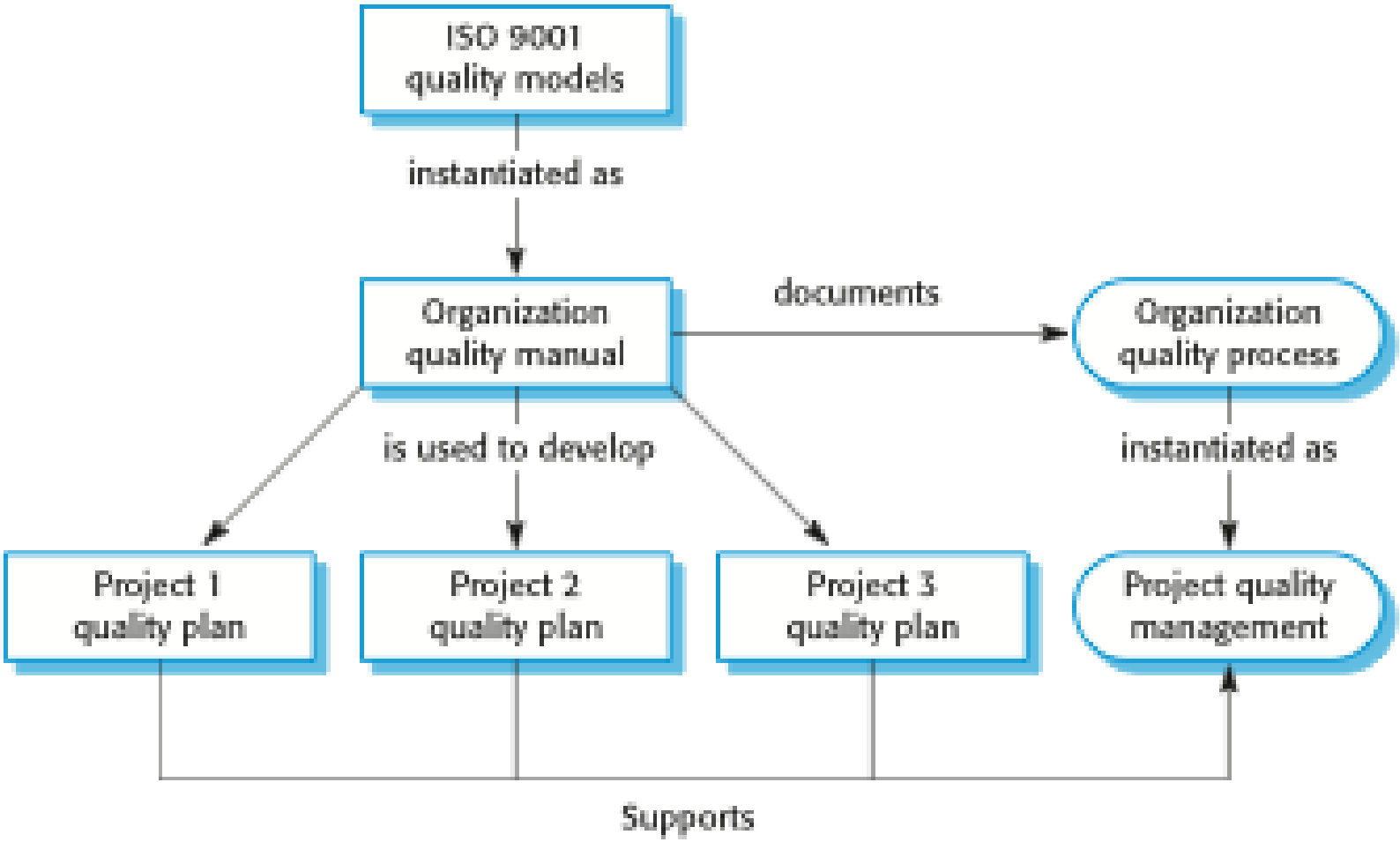
ISO 9001 standards framework

- **An international set of standards that can be used as a basis for developing quality management systems.**
- **ISO 9001, the most general of these standards, applies to organizations that design, develop and maintain products, including software.**
- **The ISO 9001 standard is a framework for developing software standards.**

ISO 9001 core processes



ISO 9001 and quality management



ISO 9001 certification

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

Reviews and inspections

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

Quality reviews

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

Quality reviews

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

Quality reviews

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

Program inspections

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

Inspection checklists

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

An inspection checklist (a)

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?

An inspection checklist (b)

Fault class	Inspection check
Interface faults	• Do all function and method calls have the correct number of parameters? • Do formal and actual parameter types match? • Are the parameters in the right order? • If components access shared memory, do they have the same model of the shared memory structure?
Storage management faults	• If a linked structure is modified, have all links been correctly reassigned? • If dynamic storage is used, has space been allocated correctly? • Is space explicitly deallocated after it is no longer required?
Exception management faults	• Have all possible error conditions been taken into account?

Software measurement and metrics

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

Metrics assumptions

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

Metrics assumptions

- **A quality metric should be a predictor of product quality.**
- **Classes of product metric**
  - **Dynamic metrics which are collected by measurements made of a program in execution;**
  - **Static metrics which are collected by measurements made of the system representations;**
  - **Dynamic metrics help assess efficiency and reliability**
  - **Static metrics help assess complexity, understandability and maintainability**
  - **Dynamic metrics are closely related to software quality attributes**
  - **Static metrics have an indirect relationship with quality attributes.**

Static software product metrics

Software metric	Description
Fan-in/Fan-out	Fan-in is a measure of the number of functions or methods that call another function or method (say X). Fan-out is the number of functions that are called by function X. A high value for fan-in means that X is tightly coupled to the rest of the design and changes to X will have extensive knock-on effects. A high value for fan-out suggests that the overall complexity of X may be high because of the complexity of the control logic needed to coordinate the called components.
Length of code	This is a measure of the size of a program. Generally, the larger the size of the code of a component, the more complex and error-prone that component is likely to be. Length of code has been shown to be one of the most reliable metrics for predicting error-proneness in components.

Static software product metrics

Software metric	Description
Cyclomatic complexity	This is a measure of the control complexity of a program. This control complexity may be related to program understandability. I discuss cyclomatic complexity in Chapter 8.
Length of identifiers	This is a measure of the average length of identifiers (names for variables, classes, methods, etc.) in a program. The longer the identifiers, the more likely they are to be meaningful and hence the more understandable the program.
Depth of conditional nesting	This is a measure of the depth of nesting of if-statements in a program. Deeply nested if-statements are hard to understand and potentially error-prone.
Fog index	This is a measure of the average length of words and sentences in documents. The higher the value of a document's Fog index, the more difficult the document is to understand.

The CK object-oriented metrics suite

Object-oriented metric	Description
Weighted methods per class (WMC)	This is the number of methods in each class, weighted by the complexity of each method. Therefore, a simple method may have a complexity of 1, and a large and complex method a much higher value. The larger the value for this metric, the more complex the object class. Complex objects are more likely to be difficult to understand. They may not be logically cohesive, so cannot be reused effectively as superclasses in an inheritance tree.
Depth of inheritance tree (DIT)	This represents the number of discrete levels in the inheritance tree where subclasses inherit attributes and operations (methods) from superclasses. The deeper the inheritance tree, the more complex the design. Many object classes may have to be understood to understand the object classes at the leaves of the tree.
Number of children (NOC)	This is a measure of the number of immediate subclasses in a class. It measures the breadth of a class hierarchy, whereas DIT measures its depth. A high value for NOC may indicate greater reuse. It may mean that more effort should be made in validating base classes because of the number of subclasses that depend on them.

The CK object-oriented metrics suite

Object-oriented metric	Description
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.

Chapter 6

Software Testing Strategies

Program testing

- **Testing is intended to show that a program does what it is intended to do and to discover program defects**
- **When you test software, you execute a program using artificial data.**
- **You check the results of the test run for errors, anomalies or information about the program's non-functional attributes.**
- **Testing is part of a more general verification and validation process, which also includes static validation techniques.**

Validation and defect testing

- **The first goal leads to validation testing**
  - **You expect the system to perform correctly using a given set of test cases that reflect the system's expected use.**
  - **Goal: To demonstrate to the developer and the system customer that the software meets its requirements**
- **The second goal leads to defect testing**
  - **The test cases are designed to expose defects. The test cases in defect testing need not reflect how the system is normally used.**
  - **Goal: To discover faults or defects in the software where its behaviour is incorrect or not in conformance with its specification**

Validation and defect testing

- **The first goal leads to validation testing**
  - **You expect the system to perform correctly using a given set of test cases that reflect the system's expected use.**
  - **Goal: To demonstrate to the developer and the system customer that the software meets its requirements**
- **The second goal leads to defect testing**
  - **The test cases are designed to expose defects. The test cases in defect testing need not reflect how the system is normally used.**
  - **Goal: To discover faults or defects in the software where its behaviour is incorrect or not in conformance with its specification**

Verification vs validation

- **Verification: "Are we building the product right".**
- **The software should conform to its specification.**
- **Validation: "Are we building the right product".**
- **The software should do what the user really requires.**

Inspections and testing

- **Software inspections Concerned with analysis of the static system representation to discover problems (static verification)**
  - **May be supplement by tool-based document and code analysis.**
- **Software testing Concerned with exercising and observing product behaviour (dynamic verification)**
  - **The system is executed with test data and its operational behaviour is observed.**

Inspections and testing

- **Software inspections Concerned with analysis of the static system representation to discover problems (static verification)**
  - **May be supplement by tool-based document and code analysis.**
- **Software testing Concerned with exercising and observing product behaviour (dynamic verification)**
  - **The system is executed with test data and its operational behaviour is observed.**
  - **These involve people examining the source representation with the aim of discovering anomalies and defects.**

Stages of testing

- **Development testing, where the system is tested during development to discover bugs and defects.**
- **Release testing, where a separate testing team test a complete version of the system before it is released to users.**
- **User testing, where users or potential users of a system test the system in their own environment.**

Development testing

- **Development testing includes all testing activities that are carried out by the team developing the system.**
  - **Unit testing, where individual program units or object classes are tested. Unit testing should focus on testing the functionality of objects or methods.**
  - **Component testing, where several individual units are integrated to create composite components. Component testing should focus on testing component interfaces.**
  - **System testing, where some or all of the components in a system are integrated and the system is tested as a whole. System testing should focus on testing component interactions.**

Development testing

- **Development testing includes all testing activities that are carried out by the team developing the system.**
  - **Unit testing, where individual program units or object classes are tested. Unit testing should focus on testing the functionality of objects or methods.**
  - **Component testing, where several individual units are integrated to create composite components. Component testing should focus on testing component interfaces.**
  - **System testing, where some or all of the components in a system are integrated and the system is tested as a whole. System testing should focus on testing component interactions.**

Unit testing

- **Unit testing is the process of testing individual components in isolation.**
- **It is a defect testing process.**
- **Units may be:**
  - **Individual functions or methods within an object**
  - **Object classes with several attributes and methods**
  - **Composite components with defined interfaces used to access their functionality**

Component testing

- **Software components are often composite components that are made up of several interacting objects.**
- **You access the functionality of these objects through the defined component interface.**
- **Testing composite components should therefore focus on showing that the component interface behaves according to its specification.**

System testing

- **System testing during development involves integrating components to create a version of the system and then testing the integrated system.**
- **The focus in system testing is testing the interactions between components.**
- **System testing checks that components are compatible, interact correctly and transfer the right data at the right time across their interfaces.**
- **System testing tests the emergent behaviour of a system.**

Test-driven development

- **Test-driven development (TDD) is an approach to program development in which you inter-leave testing and code development.**
- **Tests are written before code and 'passing' the tests is the critical driver of development.**
- **You develop code incrementally, along with a test for that increment. You don't move on to the next increment until the code that you have developed passes its test.**

Benefits of test-driven development

- **Code coverage**
  - **Every code segment that you write has at least one associated test so all code written has at least one test.**
- **Regression testing**
  - **A regression test suite is developed incrementally as a program is developed.**
- **Simplified debugging**
  - **When a test fails, it should be obvious where the problem lies. The newly written code needs to be checked and modified.**
- **System documentation**
  - **The tests themselves are a form of documentation that describe what the code should be doing.**

Release testing

- **Release testing is the process of testing a particular release of a system that is intended for use outside of the development team.**
- **The primary goal of the release testing process is to convince the supplier of the system that it is good enough for use.**
- **Release testing is usually a black-box testing process where tests are only derived from the system specification.**

Release testing and system testing

- **Release testing is a form of system testing.**
- **Important differences:**
  - **A separate team that has not been involved in the system development, should be responsible for release testing.**
  - **System testing by the development team should focus on discovering bugs in the system (defect testing). The objective of release testing is to check that the system meets its requirements and is good enough for external use (validation testing).**

Requirements tests

- **Release testing is a form of system testing.**
- **Important differences:**
  - **A separate team that has not been involved in the system development, should be responsible for release testing.**
  - **System testing by the development team should focus on discovering bugs in the system (defect testing). The objective of release testing is to check that the system meets its requirements and is good enough for external use (validation testing).**

User Testing

- **User or customer testing is a stage in the testing process in which users or customers provide input and advice on system testing.**
- **User testing is essential, even when comprehensive system and release testing have been carried out.**
  - **The reason for this is that influences from the user's working environment have a major effect on the reliability, performance, usability and robustness of a system. These cannot be replicated in a testing environment.**

Types of user testing

- **Alpha testing**
  - **Users of the software work with the development team to test the software at the developer's site.**
- **Beta testing**
  - **A release of the software is made available to users to allow them to experiment and to raise problems that they discover with the system developers.**
- **Acceptance testing**
  - **Customers test a system to decide whether or not it is ready to be accepted from the system developers and deployed in the customer environment.**

Chapter 7

Process Improvement

Process Improvement

- **Process improvement means understanding existing processes and changing these processes to increase product quality and/or reduce costs and development time.**
- **Approaches to improvement**
  - **The process maturity approach, which focuses on improving process and project management and introducing good software engineering practice.**
  - **The agile approach, which focuses on iterative development and the reduction of overheads in the software process.**

Process and product quality

- **Process quality and product quality are related and process improvement benefits arise because the quality of the product depends on its development process.**
- **A good process is usually required to produce a good product.**
- **For manufactured goods, process is the principal quality determinant.**
- **For design-based activities, other factors are also involved, especially the capabilities of the designers**

Process improvement process

- **There is no such thing as an 'ideal' or 'standard' software process that is applicable in all organizations or for all software products of a particular type.**
- **Each company has to develop its own process depending on its size, the background and skills of its staff, the type of software being developed, customer and market requirements, and the company culture.**

Process attributes

Process characteristic	**Key issues**
Understandability	To what extent is the process explicitly defined and how easy is it to understand the process definition?
Standardization	To what extent is the process based on a standard generic process? This may be important for some customers who require conformance with a set of defined process standards. To what extent is the same process used in all parts of a company?
Visibility	Do the process activities culminate in clear results, so that the progress of the process is externally visible?
Measurability	Does the process include data collection or other activities that allow process or product characteristics to be measured?
Supportability	To what extent can software tools be used to support the process activities?

Process attributes

Process characteristic	**Key issues**
Acceptability	Is the defined process acceptable to and usable by the engineers responsible for producing the software product?
Reliability	Is the process designed in such a way that process errors are avoided or trapped before they result in product errors?
Robustness	Can the process continue in spite of unexpected problems?
Maintainability	Can the process evolve to reflect changing organizational requirements or identified process improvements?
Rapidity	How fast can the process of delivering a system from a given specification be completed?

Process improvement stages

- **Process measurement**
  - **Attributes of the current process are measured. These are a baseline for assessing improvements.**
- **Process analysis**
  - **The current process is assessed and bottlenecks and weaknesses are identified.**
- **Process change**
  - **Changes to the process that have been identified during the analysis are introduced.**

Process measurements

- **Wherever possible, quantitative process data should be collected**
  - **A process may have to be defined before any measurement is possible.**
- **Process measurements should be used to assess process improvements**
  - **But this does not mean that measurements should drive the improvements. The improvement driver should be the organizational objectives.**

Process metrics

- **Time taken for process activities to be completed**
  - **E.g. Calendar time or effort to complete an activity or process.**
- **Resources required for processes or activities**
  - **E.g. Total effort in person-days.**
- **Number of occurrences of a particular event**
  - **E.g. Number of defects discovered.**

Aspects of process analysis

Process aspect	Questions
Adoption and standardization	Is the process documented and standardized across the organization? If not, does this mean that any measurements made are specific only to a single process instance? If processes are not standardized, then changes to one process may not be transferable to comparable processes elsewhere in the company.
Software engineering practice	Are there known, good software engineering practices that are not included in the process? Why are they not included? Does the lack of these practices affect product characteristics, such as the number of defects in a delivered software system?
Organizational constraints	What are the organizational constraints that affect the process design and the ways that the process is performed? For example, if the process involves dealing with classified material, there may be activities in the process to check that classified information is not included in any material due to be released to external organizations. Organizational constraints may mean that possible process changes cannot be made.

Aspects of process analysis

Process aspect	Questions
Communications	How are communications managed in the process? How do communication issues relate to the process measurements that have been made? Communication problems are a major issue in many processes and communication bottlenecks are often the reasons for project delays.
Introspection	Is the process reflective (i.e., do the actors involved in the process explicitly think about and discuss the process and how it might be improved)? Are there mechanisms through which process actors can propose process improvements?
Learning	How do people joining a development team learn about the software processes used? Does the company have process manuals and process training programs?
Tool support	What aspects of the process are and aren't supported by software tools? For unsupported areas, are there tools that could be deployed cost-effectively to provide support? For supported areas, are the tools effective and efficient? Are better tools available?

Process models

- **Process models are a good way of focusing attention on the activities in a process and the information transfer between these activities.**
- **Process models do not have to be formal or complete – their purpose is to provoke discussion rather than document the process in detail.**
- **Model-oriented questions can be used to help understand the process e.g.**
  - **What activities take place in practice but are not shown in the model?**
  - **Are there process activities, shown in the model, that you (the process actor) think are inefficient?**

Process change stages (a)

- **Improvement identification**
  - **This stage is concerned with using the results of the process analysis to identify ways to tackle quality problems, schedule bottlenecks or cost inefficiencies that have been identified during process analysis.**
- **Improvement prioritization**
  - **When many possible changes have been identified, it is usually impossible to introduce them all at once, and you must decide which are the most important.**
- **Process change introduction**
  - **Process change introduction means putting new procedures, methods and tools into place and integrating them with other process activities.**

Process change stages (b)

- **Process change training**
  - **Without training, it is not possible to gain the full benefits of process changes. The engineers involved need to understand the changes that have been proposed and how to perform the new and changed processes.**
- **Change tuning**
  - **Proposed process changes will never be completely effective as soon as they are introduced. You need a tuning phase where minor problems can be discovered, and modifications to the process can be proposed and introduced.**

The CMMI model

- **An integrated capability model that includes software and systems engineering capability assessment.**
  - **The model has two instantiations**
- **Staged where the model is expressed in terms of capability levels;**
  - **Continuous where a capability rating is computed.**

The staged CMMI model

- **Comparable with the software CMM.**
- **Each maturity level has process areas and goals. For example, the process area associated with the managed level include:**
  - **Requirements management;**
  - **Project planning;**
  - **Project monitoring and control;**
  - **Supplier agreement management;**
  - **Measurement and analysis;**
  - **Process and product quality assurance.**

The CMMI staged maturity model