JAWAHARLAL COLLEGE OF ENGINEERING AND TECHNOLOGY

(Approved by AICTE, Affiliated to APJ Abdul Kalam Technological University, Kerala)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

(NBA Accredited)

COURSE MATERIAL

CST 309 MANAGEMENT OF SOFTWARE SYSTEMS

VISION OF THE INSTITUTION

Emerge as a centre of excellence for professional education to produce high quality engineers and entrepreneurs for the development of the region and the Nation.

MISSION OF THE INSTITUTION

- To become an ultimate destination for acquiring latest and advanced knowledge in the multidisciplinary domains.
- To provide high quality education in engineering and technology through innovative teaching-learning practices, research and consultancy, embedded with professional ethics.
- To promote intellectual curiosity and thirst for acquiring knowledge through outcome based education.

- To have partnership with industry and reputed institutions to enhance the employability skills of the students and pedagogical pursuits.
- To leverage technologies to solve the real life societal problems through community services.

ABOUT THE DEPARTMENT

- Established in: 2008
- Courses offered: B.Tech in Computer Science and Engineering
- Affiliated to the A P J Abdul Kalam Technological University.

DEPARTMENT VISION

To produce competent professionals with research and innovative skills, by providing them with the most conducive environment for quality academic and research oriented undergraduate education along with moral values committed to build a vibrant nation.

DEPARTMENT MISSION

- Provide a learning environment to develop creativity and problem solving skills in a professional manner.
- Expose to latest technologies and tools used in the field of computer science.
- Provide a platform to explore the industries to understand the work culture and expectation of an organization.
- Enhance Industry Institute Interaction program to develop the entrepreneurship skills.
- Develop research interest among students which will impart a better life for the society and the nation.

PROGRAMME EDUCATIONAL OBJECTIVES

Graduates will be able to

- Provide high-quality knowledge in computer science and engineering required for a computer professional to identify and solve problems in various application domains.
- Persist with the ability in innovative ideas in computer support systems and transmit the knowledge and skills for research and advanced learning.
- Manifest the motivational capabilities, and turn on a social and economic commitment to community services.

PROGRAM OUTCOMES (POS)

Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

COURSE OUTCOMES

SUBJECT CODE: C210	
COURSE OUTCOMES	
C305.1	To identify the basic structure and functional units of a digital computer. And analyze the effect of addressing modes on the execution time of a program
C305.2	To design processing unit using the concepts of ALU and control logic design.
C305.3	To select appropriate interfacing standards for I/O devices.
C305.4	To identify the pros and cons of different types of Memory systems and understand mapping functions.
C305.5	To select appropriate interfacing standards for I/O devices.
C305.6	To identify the roles of various functional units of a computer in instruction execution. And analyze the types of control logic design in processors.

PROGRAM SPECIFIC OUTCOMES (PSO)

The students will be able to

- Use fundamental knowledge of mathematics to solve problems using suitable analysis methods, data structure and algorithms.
- Interpret the basic concepts and methods of computer systems and technical specifications to provide accurate solutions.
- Apply theoretical and practical proficiency with a wide area of programming knowledge, design new ideas and innovations towards research.

CO PO MAPPING

Note: H-Highly correlated=3, M-Medium correlated=2,L-Less correlated=1

CO'S	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
C305.1	3	2	2	-		-	-	-	-	-	-	2
C305.2	3	3	-	-	2	-	-	-	-	-	-	2
C305.3	3	2	-	-	2	-	-	-	-	-	-	2
C305.4	3	2	2	-		-	-	-	-	-	-	2
C305.5	3	3	3	-	-	-	-	-	-	-	-	-
C305.6	3	2	-	-	2	-	-	-	-	-	-	2
C305	3	2.3	2.3		2							2

CO PSO MAPPING

CO'S	PSO1	PSO2	PSO3
C305.1	3	2	2
C305.2	3	2	2
C305.3	3	2	2
C305.4	2	2	2
C305.5	2	3	2
C305.6	2	3	2
C305	2.5	2.3	2

MODULE 1 : Introduction to Software Engineering (7 hours)

- 1. Introduction to Software Engineering - Professional software development, Software engineering ethics
- Software process models - The waterfall model, Incremental development. Process activities - Software specification, Software design and implementation, Software validation, Software evolution. Coping with change - Prototyping, Incremental delivery, Boehm's Spiral Model.
- Agile software development - Agile methods, agile manifesto - values and principles. Agile development techniques, Agile Project Management.
- Case studies : An insulin pump control system. Mentcare - a patient information system for mental health care.

1.1 Professional software development

- Software is not just a program themselves but also all associated documentation and configuration data .
- What is a software Engineering?

Frequently asked questions about software engineering

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.

Frequently asked questions about software engineering

Question	Answer
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.

Software products

- Generic products
  - Stand-alone systems that are marketed and sold to any customer who wishes to buy them.
  - Examples – PC software such as graphics programs, project management tools; CAD software; software for specific markets such as appointments systems for dentists.
  - Organization that develops the software controls the software specification.
- Customized products(bespoke)
  - Software that is commissioned by a specific customer to meet their own needs.
  - Examples – embedded control systems, air traffic control software, traffic monitoring systems.
  - Specification is developed and controlled by the organization ie buying the software.

Essential attributes of good software

Product characteristic	Description
Maintainability	Software should be written in such a way so that it can evolve to meet the changing needs of customers. This is a critical attribute because software change is an inevitable requirement of a changing business environment.
Dependability and security	Software dependability includes a range of characteristics including reliability, security and safety. Dependable software should not cause physical or economic damage in the event of system failure. Malicious users should not be able to access or damage the system.
Efficiency	Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, memory utilisation, etc.
Acceptability	Software must be acceptable to the type of users for which it is designed. This means that it must be understandable, usable and compatible with other systems that they use.

1.1.1 Software Engineering

- Software engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification through to maintaining the system after it has gone into use.
- Engineering discipline
  - Using appropriate theories and methods to solve problems within the organizational and financial constraints.
- All aspects of software production
  - Not just technical process of development. Also project management and the development of tools, methods etc. to support software production.

Importance of software engineering

- More and more, individuals and society rely on advanced software systems. We need to be able to produce reliable and trustworthy systems economically and quickly.
- It is usually cheaper, in the long run, to use software engineering methods and techniques for software systems rather than just write the programs as if it was a personal programming project. For most types of system, the majority of costs are the costs of changing the software after it has gone into use.

Software process activities

- Software specification, where customers and engineers define the software that is to be produced and the constraints on its operation.
- Software development, where the software is designed and programmed.
- Software validation, where the software is checked to ensure that it is what the customer requires.
- Software evolution, where the software is modified to reflect changing customer and market requirements.

General issues that affect most software

- Heterogeneity
  - Increasingly, systems are required to operate as distributed systems across networks that include different types of computer and mobile devices.
- Business and social change
  - Business and society are changing incredibly quickly as emerging economies develop and new technologies become available. They need to be able to change their existing software and to rapidly develop new software.
- Security and trust
  - As software is intertwined with all aspects of our lives, it is essential that we can trust that software

1.1.2 Software Engineering diversity

- There are many different types of software system and there is no universal set of software techniques that is applicable to all of these.
- The software engineering methods and tools used depend on the type of application being developed, the requirements of the customer and the background of the development team.

Software Engineering fundamentals

- Some fundamental principles apply to all types of software system, irrespective of the development techniques used:
  - Systems should be developed using a managed and understood development process. Of course, different processes are used for different types of software.
  - Dependability and performance are important for all types of system.
  - Understanding and managing the software specification and requirements (what the software should do) are important.
  - Where appropriate, you should reuse software that has already been developed rather than write new software.

Application types

- Stand-alone applications
  - These are application systems that run on a local computer, such as a PC. They include all necessary functionality and do not need to be connected to a network.
- Interactive transaction-based applications
  - Applications that execute on a remote computer and are accessed by users from their own PCs or terminals. These include web applications such as e-commerce applications.
- Embedded control systems
  - These are software control systems that control and manage hardware devices. Numerically, there are probably more embedded systems than any other type of system.

- Batch processing systems
  - These are business systems that are designed to process data in large batches. They process large numbers of individual inputs to create corresponding outputs.
- Entertainment systems
  - These are systems that are primarily for personal use and which are intended to entertain the user.
- Systems for modeling and simulation
  - These are systems that are developed by scientists and engineers to model physical processes or situations, which include many, separate, interacting objects.
- Data collection systems
  - These are systems that collect data from their environment using a set of sensors and send that data to other systems for processing.
- Systems of systems
  - These are systems that are composed of a number of other software systems

1.1.3 Software Engineering and the Web

- The Web is now a platform for running application and organizations are increasingly developing web-based systems rather than local systems.
- Web services allow application functionality to be accessed over the web.
- Cloud computing is an approach to the provision of computer services where applications run remotely on the ‘cloud’.
  - Users do not buy software buy pay according to use

Web software Engineering

- Software reuse is the dominant approach for constructing web-based systems.
  - When building these systems, you think about how you can assemble them from pre-existing software components and systems.
- Web-based systems should be developed and delivered incrementally.
  - It is now generally recognized that it is impractical to specify all the requirements for such systems in advance.
- User interfaces are constrained by the capabilities of web browsers.
  - Technologies such as AJAX allow rich interfaces to be created within a web browser but are still difficult to use. Web forms with local scripting are more commonly used.

1.2 Software engineering ethics

- Software engineering involves wider responsibilities than simply the application of technical skills.
- Software engineers must behave in an honest and ethically responsible way if they are to be respected as professionals.
- Ethical behaviour is more than simply upholding the law but involves following a set of principles that are morally correct.

Issues of professional responsibility

- Confidentiality
  - Engineers should normally respect the confidentiality of their employers or clients irrespective of whether or not a formal confidentiality agreement has been signed.
- Competence
  - Engineers should not misrepresent their level of competence. They should not knowingly accept work which is outwith their competence.
- Intellectual property rights
  - Engineers should be aware of local laws governing the use of intellectual property such as patents, copyright, etc. They should be careful to ensure that the intellectual property of employers and clients is protected.
- Computer misuse
  - Software engineers should not use their technical skills to misuse other people's computers. Computer misuse ranges from relatively trivial (game playing on an employer's machine, say) to extremely serious (dissemination of viruses).

ACM/IEEE Code of Ethics

- The professional societies in the US have cooperated to produce a code of ethical practice.
- Members of these organisations sign up to the code of practice when they join.
- The Code contains eight Principles related to the behaviour of and decisions made by professional software engineers, including practitioners, educators, managers, supervisors and policy makers, as well as trainees and students of the profession.

Rationale for the code of ethics

- *Computers have a central and growing role in commerce, industry, government, medicine, education, entertainment and society at large. Software engineers are those who contribute by direct participation or by teaching, to the analysis, specification, design, development, certification, maintenance and testing of software systems.*
- *Because of their roles in developing software systems, software engineers have significant opportunities to do good or cause harm, to enable others to do good or cause harm, or to influence others to do good or cause harm. To ensure, as much as possible, that their efforts will be used for good, software engineers must commit themselves to making software engineering a beneficial and respected profession.*

The ACM/IEEE Code of Ethics

Software Engineering Code of Ethics and Professional Practice

- ACM/IEEE-CS Joint Task Force on Software Engineering Ethics and Professional Practices

PREAMBLE

- The short version of the code summarizes aspirations at a high level of the abstraction; the clauses that are included in the full version give examples and details of how these aspirations change the way we act as software engineering professionals. Without the aspirations, the details can become legalistic and tedious; without the details, the aspirations can become high sounding but empty; together, the aspirations and the details form a cohesive code.

- Software engineers shall commit themselves to making the analysis, specification, design, development, testing and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety and welfare of the public, software engineers shall adhere to the following Eight Principles:

-

Ethical principles

1. PUBLIC - Software engineers shall act consistently with the public interest.
2. CLIENT AND EMPLOYER - Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.
3. PRODUCT - Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
4. JUDGMENT - Software engineers shall maintain integrity and independence in their professional judgment.
5. MANAGEMENT - Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
6. PROFESSION - Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.
7. COLLEAGUES - Software engineers shall be fair to and supportive of their colleagues.
8. SELF - Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

CASE STUDIES

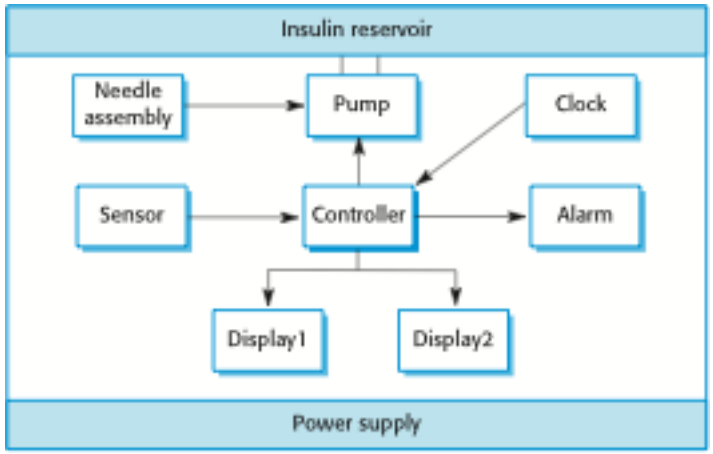
1.3 CASE STUDIES

- A personal insulin pump
  - An embedded system in an insulin pump used by diabetics to maintain blood glucose control.
- A mental health case patient management system
  - A system used to maintain records of people receiving care for mental health problems.
- A wilderness weather station
  - A data collection system that collects data about weather conditions in remote areas.

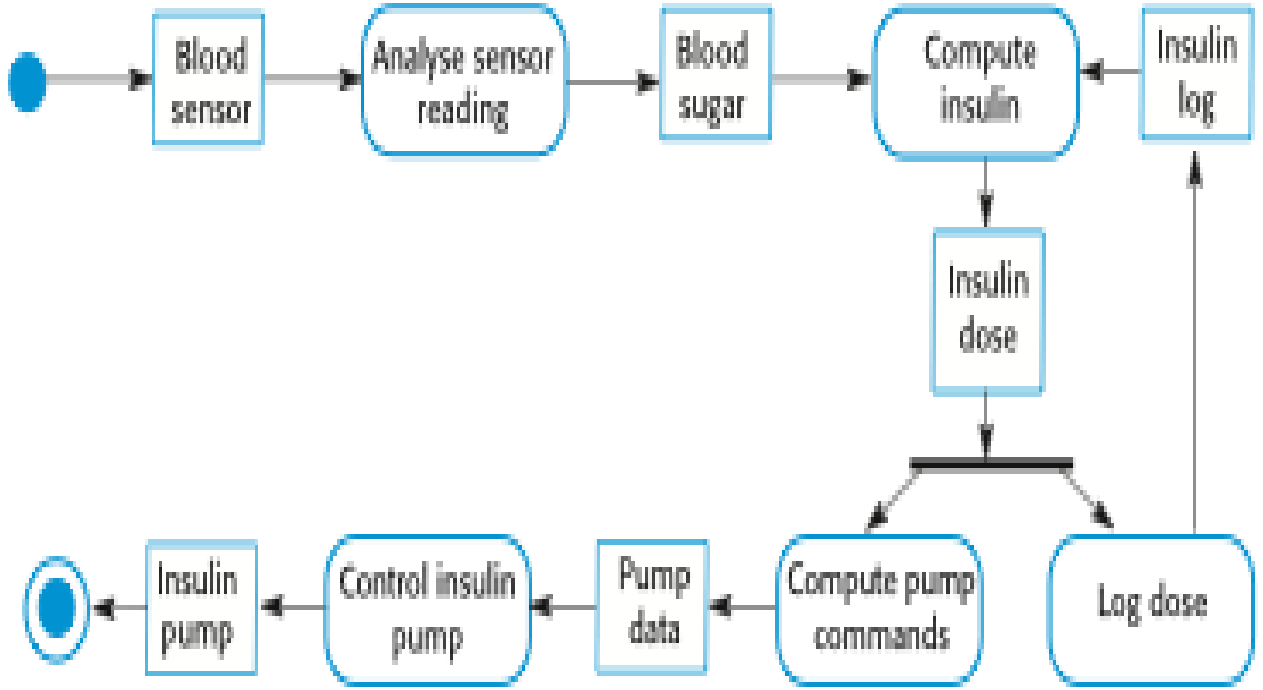
1.3.1 Insulin pump control system

- Collects data from a blood sugar sensor and calculates the amount of insulin required to be injected.
- Calculation based on the rate of change of blood sugar levels.
- Sends signals to a micro-pump to deliver the correct dose of insulin.
- Safety-critical system as low blood sugars can lead to brain malfunctioning, coma and death; high-blood sugar levels have long-term consequences such as eye and kidney damage.

Insulin pump hardware architecture



Activity model of the insulin pump



Essential high-level requirements

- The system shall be available to deliver insulin when required.
- The system shall perform reliably and deliver the correct amount of insulin to counteract the current level of blood sugar.
- The system must therefore be designed and implemented to ensure that the system always meets these requirements.

1.3.2 A patient information system for mental health care

- A patient information system to support mental health care is a medical information system that maintains information about patients suffering from mental health problems and the treatments that they have received.
- Most mental health patients do not require dedicated hospital treatment but need to attend specialist clinics regularly where they can meet a doctor who has detailed knowledge of their problems.
- To make it easier for patients to attend, these clinics are not just run in hospitals. They may also be held in local medical practices or community centres.

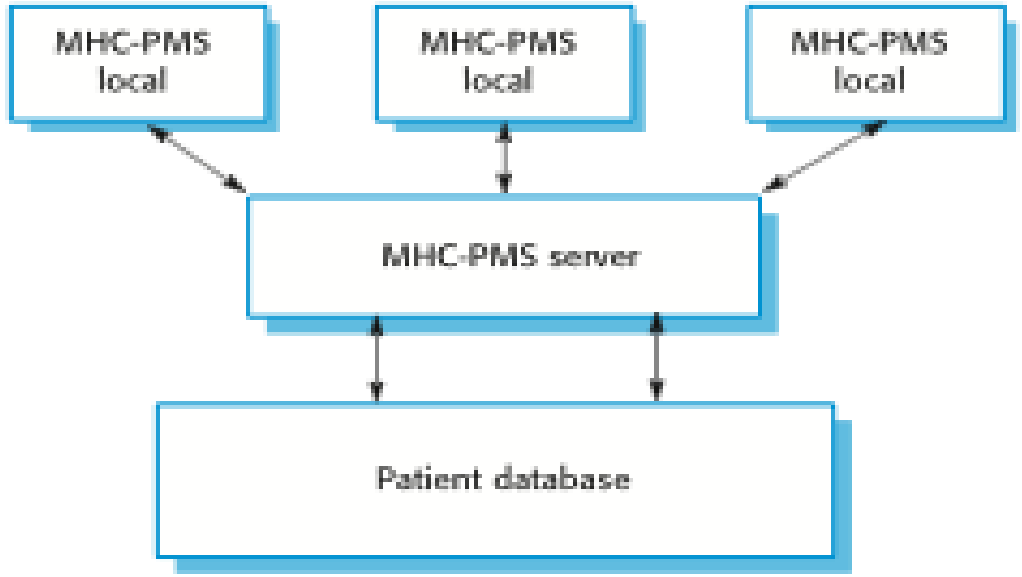
MHC-PMS

- The MHC-PMS (Mental Health Care-Patient Management System) is an information system that is intended for use in clinics.
- It makes use of a centralized database of patient information but has also been designed to run on a PC, so that it may be accessed and used from sites that do not have secure network connectivity.
- When the local systems have secure network access, they use patient information in the database but they can download and use local copies of patient records when they are disconnected.

MHC-PMS goals

- To generate management information that allows health service managers to assess performance against local and government targets.
- To provide medical staff with timely information to support the treatment of patients.

The organization of the MHC-PMS



MHC-PMS key features

- Individual care management
  - Clinicians can create records for patients, edit the information in the system, view patient history, etc. The system supports data summaries so that doctors can quickly learn about the key problems and treatments that have been prescribed.
- Patient monitoring
  - The system monitors the records of patients that are involved in treatment and issues warnings if possible problems are detected.
- Administrative reporting
  - The system generates monthly management reports showing the number of patients treated at each clinic, the number of patients who have entered and left the care system, number of patients sectioned, the drugs prescribed and their costs, etc.

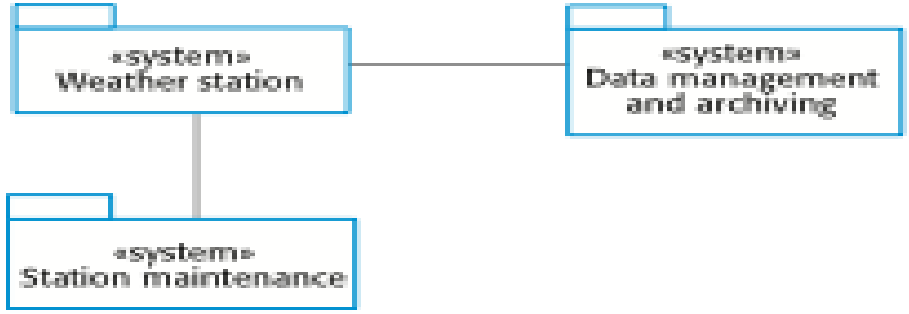
MHC-PMS concerns

- Privacy
  - It is essential that patient information is confidential and is never disclosed to anyone apart from authorised medical staff and the patient themselves.
- Safety
  - Some mental illnesses cause patients to become suicidal or a danger to other people. Wherever possible, the system should warn medical staff about potentially suicidal or dangerous patients.
  - The system must be available when needed otherwise safety may be compromised and it may be impossible to prescribe the correct medication to patients.

1.3.3 Wilderness weather station

- The government of a country with large areas of wilderness decides to deploy several hundred weather stations in remote areas.
- Weather stations collect data from a set of instruments that measure temperature and pressure, sunshine, rainfall, wind speed and wind direction.
  - The weather station includes a number of instruments that measure weather parameters such as the wind speed and direction, the ground and air temperatures, the barometric pressure and the rainfall over a 24-hour period. Each of these instruments is controlled by a software system that takes parameter readings periodically and manages the data collected from the instruments.

The weather station’s environment



Weather information system

- The weather station system
  This is responsible for collecting weather data, carrying out some initial data processing and transmitting it to the data management system.
- The data management and archiving system
  This system collects the data from all of the wilderness weather stations, carries out data processing and analysis and archives the data.
- The station maintenance system
  This system can communicate by satellite with all wilderness weather stations to monitor the health of these systems and provide reports of problems.

Additional software functionality

- Monitor the instruments, power and communication hardware and report faults to the management system.
- Manage the system power, ensuring that batteries are charged whenever the environmental conditions permit but also that generators are shut down in potentially damaging weather conditions, such as high wind.
- Support dynamic reconfiguration where parts of the software are replaced with new versions and where backup instruments are switched into the system in the event of system failure.

SOFTWARE PROCESS MODELS

The software process

- A structured set of activities required to develop a software system.
- Many different software processes but all involve:
  - Specification – defining what the system should do;
  - Design and implementation – defining the organization of the system and implementing the system;
  - Validation – checking that it does what the customer wants;
  - Evolution – changing the system in response to changing customer needs.
- A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective.

Software process descriptions

- When we describe and discuss processes, we usually talk about the activities in these processes such as specifying a data model, designing a user interface, etc. and the ordering of these activities.
- Process descriptions may also include:
  - Products, which are the outcomes of a process activity;
  - Roles, which reflect the responsibilities of the people involved in the process;
  - Pre- and post-conditions, which are statements that are true before and after a process activity has been enacted or a product produced.

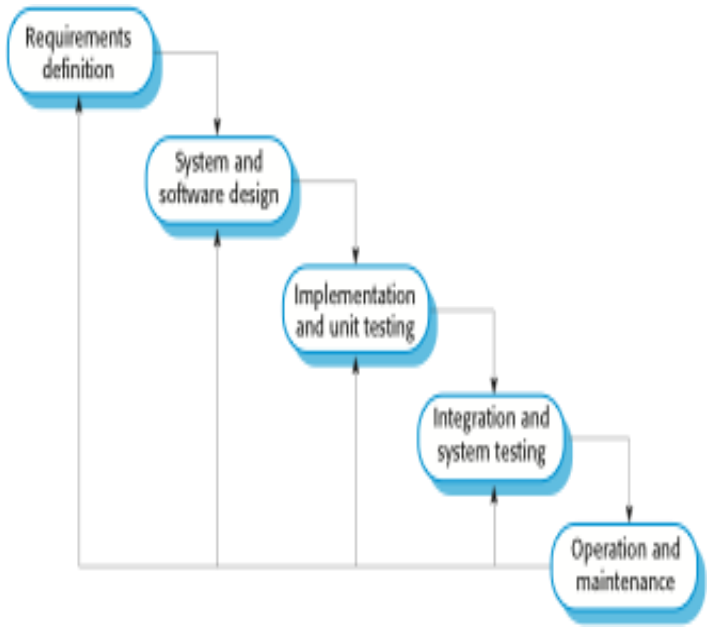
Plan-driven and agile processes

- Plan-driven processes are processes where all of the process activities are planned in advance and progress is measured against this plan.
- In agile processes, planning is incremental and it is easier to change the process to reflect changing customer requirements.
- In practice, most practical processes include elements of both plan-driven and agile approaches.
- There are no right or wrong software processes.

Software process models

- The waterfall model
  - Plan-driven model. Separate and distinct phases of specification and development.
- Incremental development
  - Specification, development and validation are interleaved. May be plan-driven or agile.
- Reuse-oriented software engineering
  - The system is assembled from existing components. May be plan-driven or agile.
- In practice, most large systems are developed using a process that incorporates elements from all of these models.

The waterfall model



Waterfall model phases

There are separate identified phases in the waterfall model:

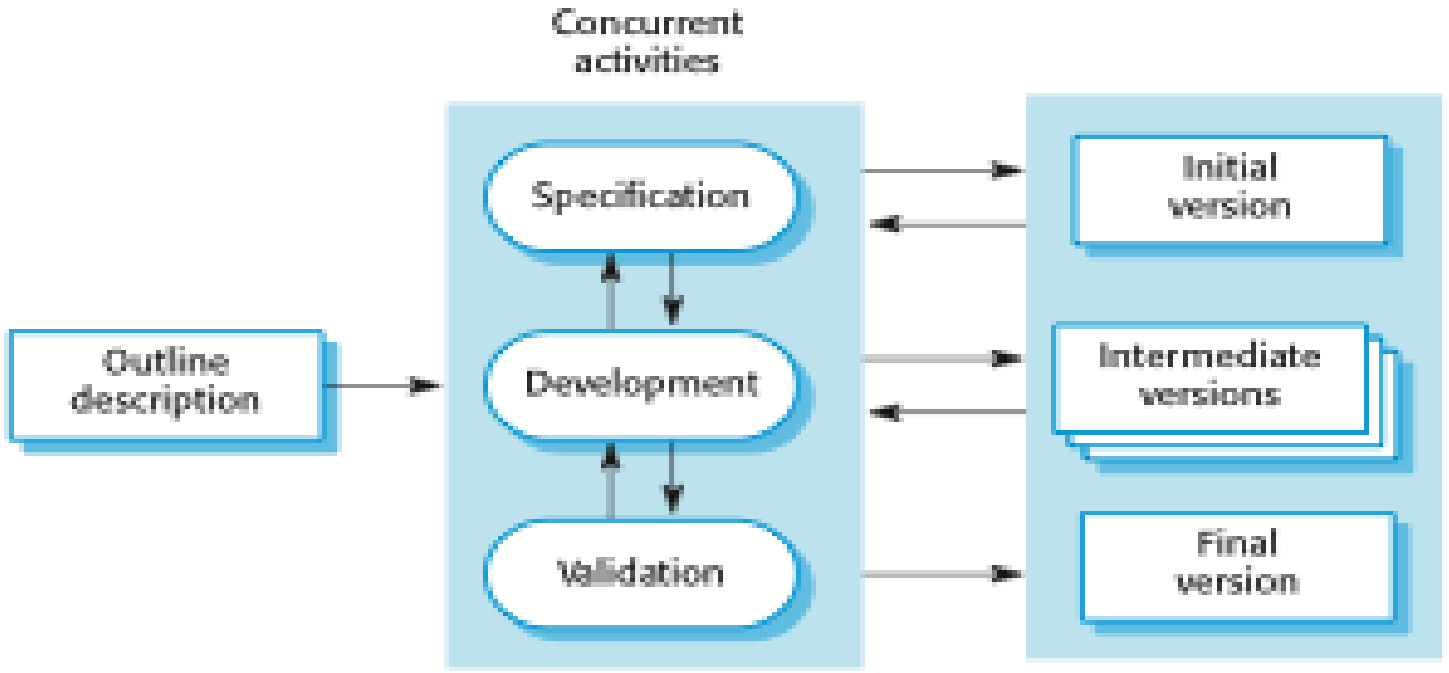
- Requirements analysis and definition
- System and software design
- Implementation and unit testing
- Integration and system testing
- Operation and maintenance

The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway. In principle, a phase has to be complete before moving onto the next phase.

Waterfall model problems

- Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.
  - Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.
  - Few business systems have stable requirements.
- The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites.
  - In those circumstances, the plan-driven nature of the waterfall model helps coordinate the work.

Incremental development



Incremental development

- Incremental Development is based on the idea of developing an initial implementation, exposing this to user comment and evolving it through several versions until an adequate system has been developed. Specification, development and validation activities are interleaved rather than separate, with rapid feedback across activities.
- It is better for business,e-commerce and software systems.
- The customer can evaluate the system at a relatively early stage in the development to see if it delivers what is required. If not, then only the current increment has to be changed and possibly new functionality defined for later increments.

Incremental development benefits

- The cost of accommodating changing customer requirements is reduced.
  - The amount of analysis and documentation that has to be redone is much less than is required with the waterfall model.
- It is easier to get customer feedback on the development work that has been done.
  - Customers can comment on demonstrations of the software and see how much has been implemented.
- More rapid delivery and deployment of useful software to the customer is possible.
  - Customers are able to use and gain value from the software earlier than is possible with a waterfall process.

Incremental development problems

- The process is not visible.
  - Managers need regular deliverables to measure progress. If systems are developed quickly, it is not cost-effective to produce documents that reflect every version of the system.
- System structure tends to degrade as new increments are added.
  - Unless time and money is spent on refactoring to improve the software, regular change tends to corrupt its structure. Incorporating further software changes becomes increasingly difficult and costly.

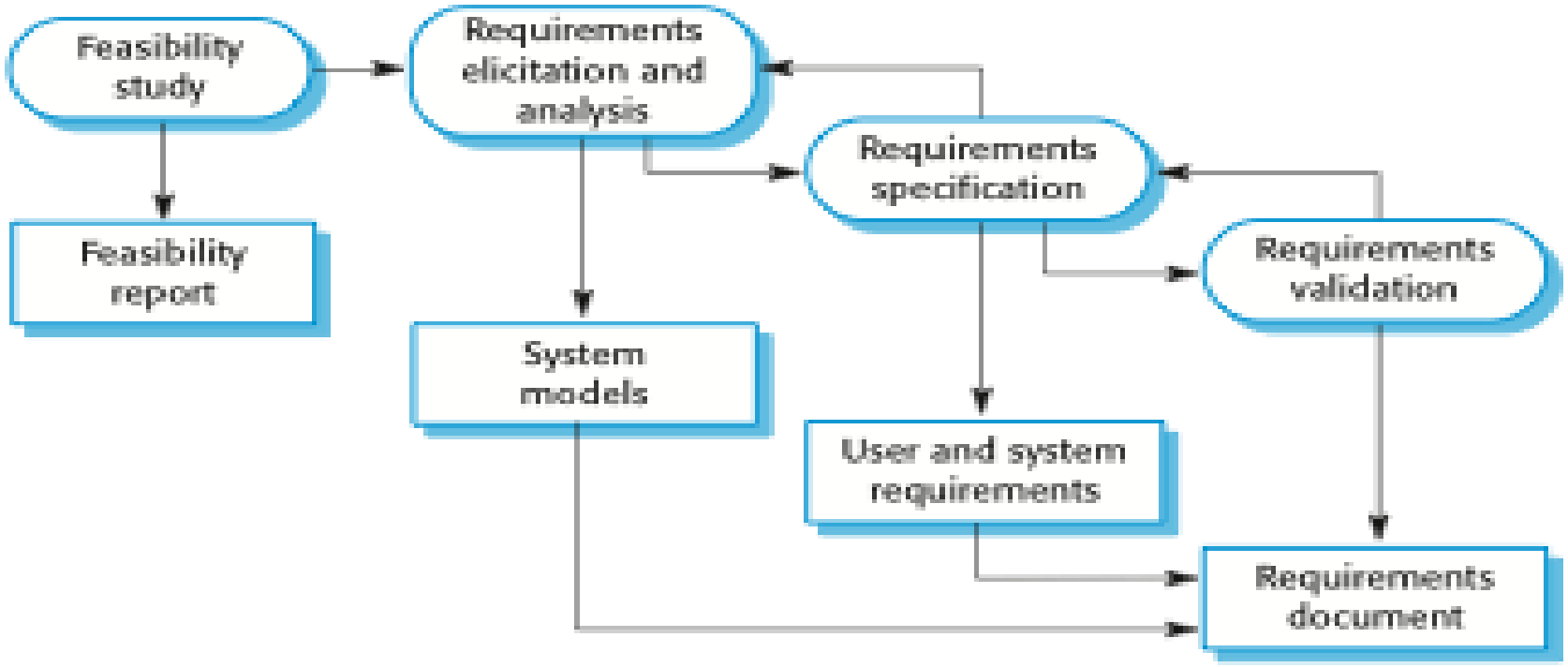
Process activities

- Real software processes are inter-leaved sequences of technical, collaborative and managerial activities with the overall goal of specifying, designing, implementing and testing a software system.
- The four basic process activities of specification, development, validation and evolution are organized differently in different development processes.
- In the waterfall model, they are organized in sequence, whereas in incremental development they are inter-leaved.

Software specification/requirement Engg

- The process of establishing and defining what services are required from the system and identifying the constraints on the system's operation and development.
- Is a particularly critical stage of the software process as errors at this stage inevitably lead to later problems in system design and implementation.
- RE process aims to produce an agreed requirements document that specifies a system satisfying stakeholder requirements.
- Requirements are presented at two levels: End users and customers need a high level statement of the requirements; system developers need a more detailed system specification.

The requirements engineering process



Software specification

- The process of establishing what services are required and the constraints on the system's operation and development.
- Requirements engineering process
  - Feasibility study
    - Is it technically and financially feasible to build the system?
    - Developed within the existing budgetary constraints.(cost effective)
  - Requirements elicitation and analysis
    - What do the system stakeholders require or expect from the system?
    - Observations from existing systems, discussions with potential users ,task analysis.
    - This may involve the development of one or more models and prototypes
  - Requirements specification
    - Is the activity of translating the information gathered during the analysis activity into a document
    - Two types of requirements
    - User requirements :are abstract statements of the system requirements for the customer and end user of the system.
    - System requirements are a more detailed description of the functionality to be provided.
  - Requirements validation
    - Checking the validity of the requirements(consistent/complete)

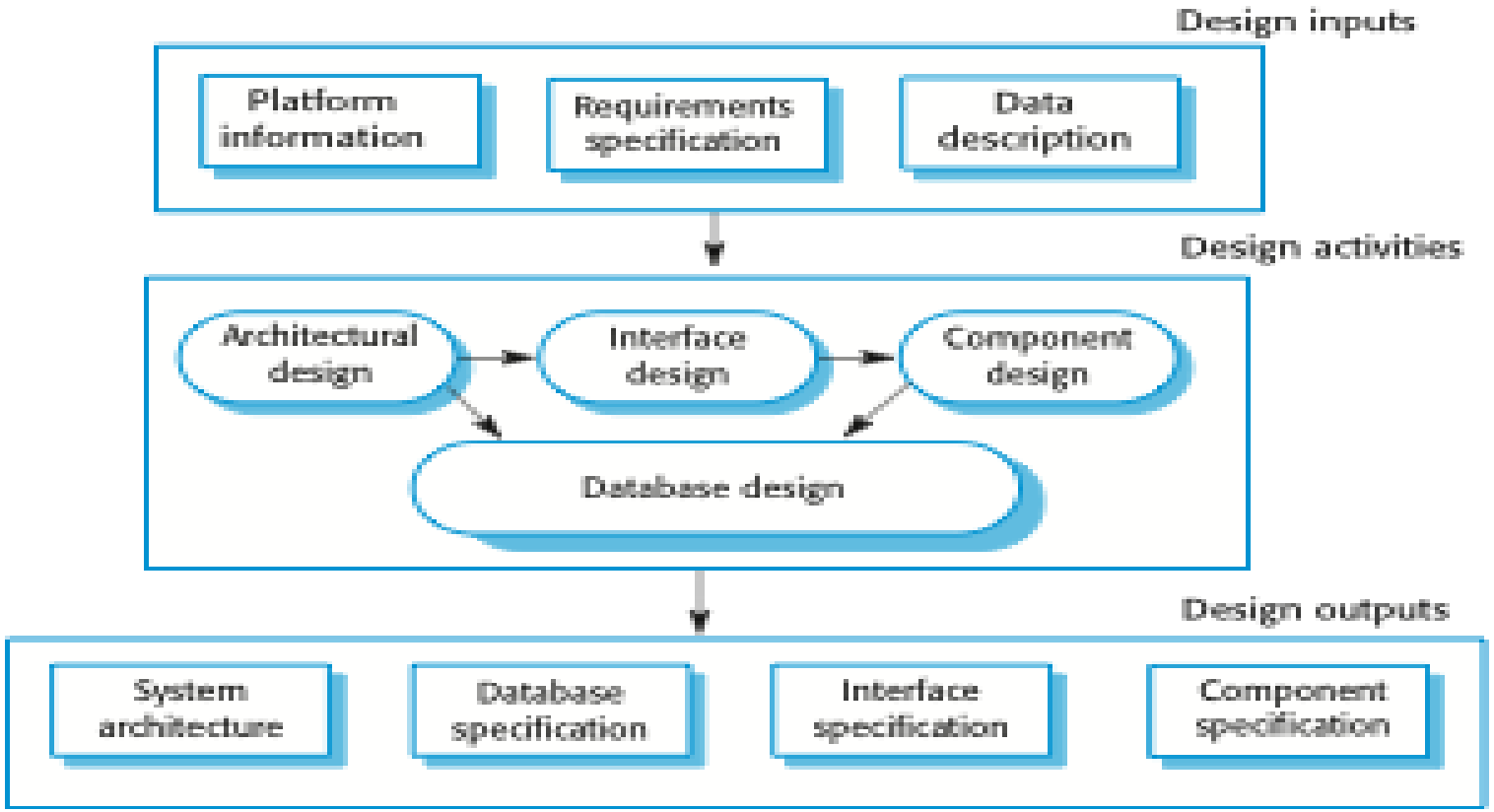
Software design and implementation

- The process of converting the system specification into an executable system.
- Software design
  - Design a software structure that realises the specification;
- Implementation
  - Translate this structure into an executable program;
- The activities of design and implementation are closely related and may be inter-leaved.

Software design and implementation

- Software platform-the environment in which software will execute.
- Information about this platform is an essential input to the design process,as designers must decide how best to integrate it with the software ‘s environment.
- The requirement specification is the description of the functionality the software must provide and its performance and dependability requirements.
- If the system is to process existing data, then the description of that data may be included in the platform specification.
- Otherwise ,the data description must be an input to the design process so that the system data organization to be defined.

A general model of the design process



Design activities

- *Architectural design*, where you identify the overall structure of the system, the principal components (sometimes called sub-systems or modules), their relationships and how they are distributed.
- *Interface design*, where you define the interfaces between system components. This interface specification must be unambiguous.
- *Component design*, where you take each system component and design how it will operate.
- *Database design*, where you design the system data structures and how these are to be represented in a database. ,depends on whether an existing database is to be reused or a new database is to be created.

- Design outputs
  - System Architecture
  - Database Specification
  - Interface specification
  - Component specification

Software validation

- Verification and validation (V & V) is intended to show that a system conforms to its specification and meets the requirements of the system customer.
- Validation involves checking and review processes and system testing.
- System testing involves executing the system with test cases that are derived from the specification of the real data to be processed by the system.
- Testing is the most commonly used V & V activity.

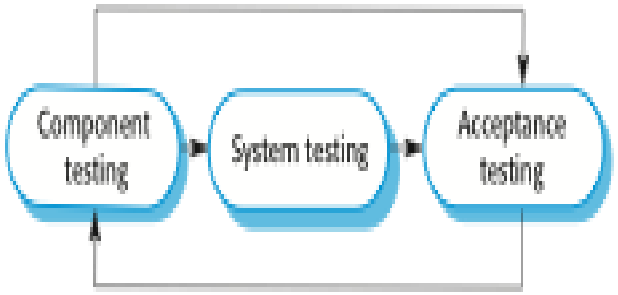
Stages of testing

- 3 stage process

System components are tested(component defects are discovered early in the process)

then the integrated system is tested,(interface problems are found when the system is integrated)

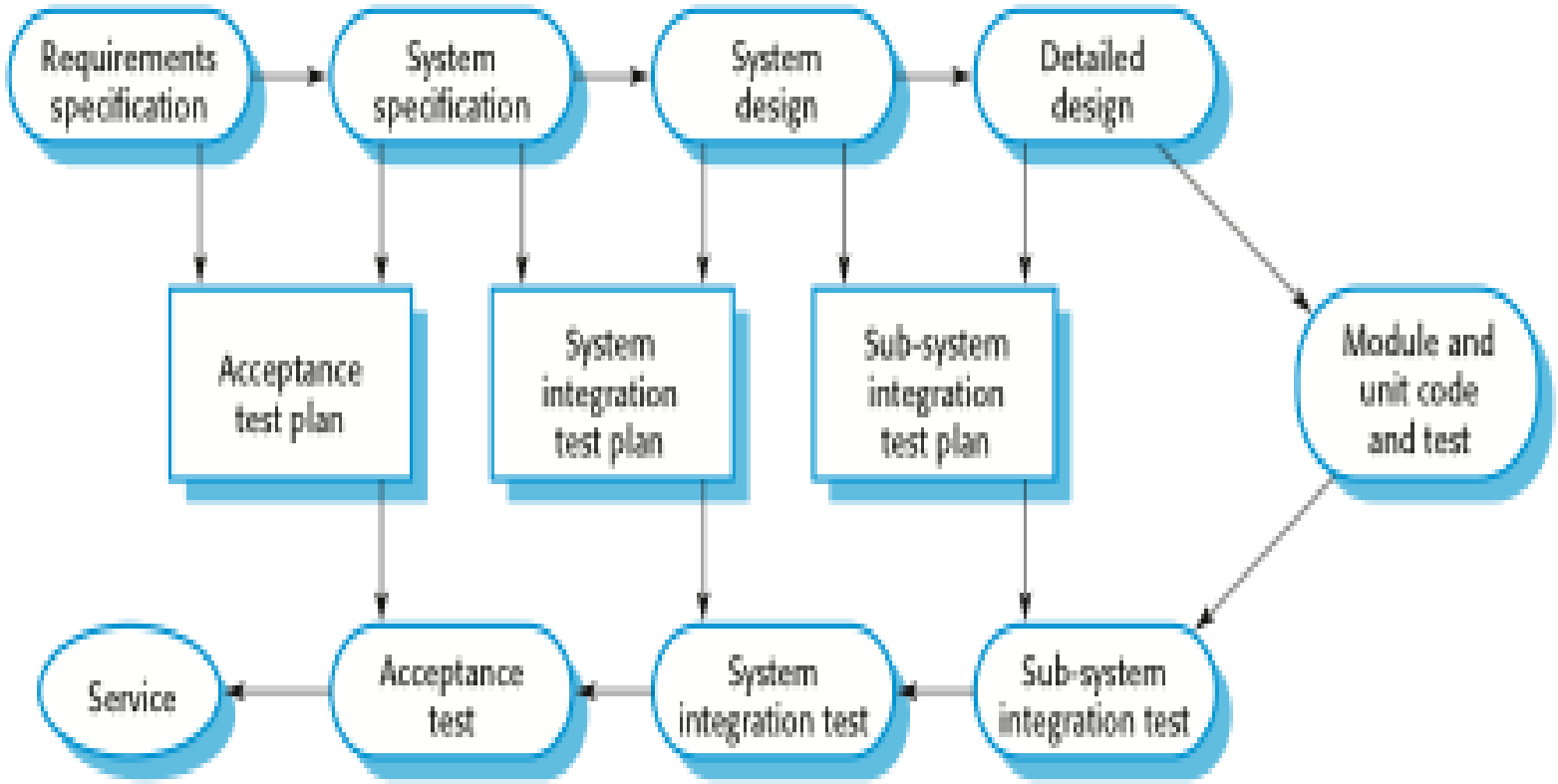
Finally the system is tested with the customers data.



Testing stages

- Development or component testing
  - Individual components are tested independently;
  - Components may be functions or objects or coherent groupings of these entities.
  - Test automation tools such as JUnit that can rerun component tests when new versions of the components are created, are commonly used.
- System testing
  - Testing of the system as a whole.
  - Concerned with showing the system meets its functional and non functional requirements , Testing of emergent properties is particularly important.
- Acceptance testing(alpha testing)
  - This is the final stage in the testing process before the system is accepted for operational use.
  - Testing with customer data to check that the system meets the customer’s needs.

Testing phases in a plan-driven software process(v model of development)



- Acceptance testing(alpha testing)

  - Alpha Testing is a **type of software testing performed to identify bugs** before releasing the product to real users or to the public. Alpha Testing is one of the user acceptance testing.
  - Custom systems are developed for a single client
  - This alpha testing process continues until the system developer and the client agree that the delivered system is an acceptable implementation of requirements.

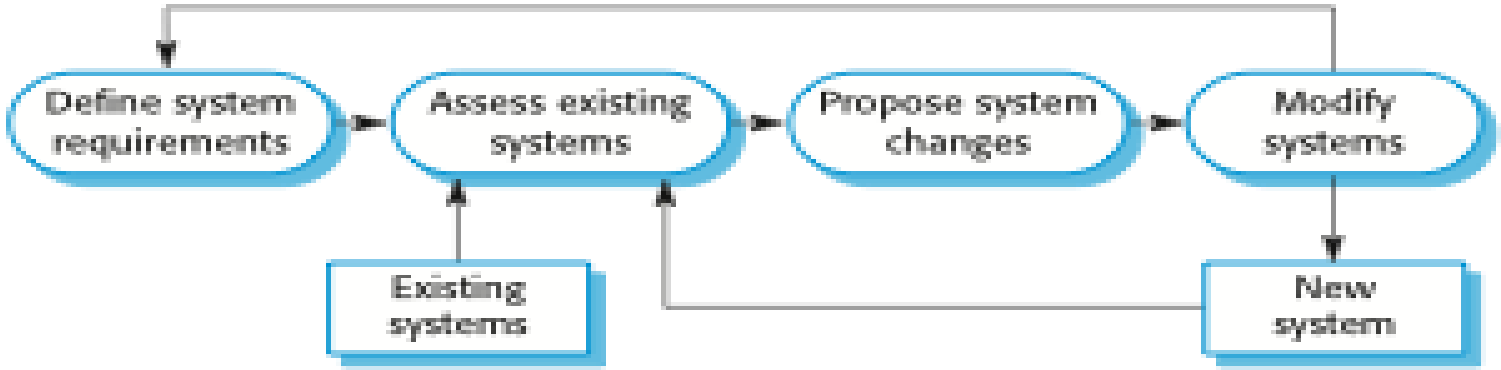
  Beta testing

  - When software is to be marketed as a software product ,beta testing is used.
  - Beta Testing is performed by **real users** of the software application in a real environment.
  - This involves delivering a system to a number of potential users who agree to use that system.
  - They report problem to system developers.
  - This exposes the product to real use and detects errors that may not have been anticipated by the system builders.
  - After this feedback,the system is modified and released either for further beta testing or general sale.

Alpha Testing	Beta Testing
Alpha testing involves both the white box and black box testing.	Beta testing commonly uses black box testing.
Alpha testing is performed by testers who are usually internal employees of the organization.	Beta testing is performed by clients who are not part of the organization.
Alpha testing is performed at developer's site.	Beta testing is performed at end-user of the product.
Reliability and security testing are not checked in alpha testing.	Reliability, security and robustness are checked during beta testing.
Alpha testing ensures the quality of the product before forwarding to beta testing.	Beta testing also concentrates on the quality of the product but collects users input on the product and ensures that the product is ready for real time users.
Alpha testing requires a testing environment or a lab.	Beta testing doesn't require a testing environment or lab.
Alpha testing may require long execution cycle.	Beta testing requires only a few weeks of execution.
Developers can immediately address the critical issues or fixes in alpha testing.	Most of the issues or feedback collected from beta testing will be implemented in future versions of the product.

Software evolution

- Software is inherently flexible and can change.
- As requirements change through changing business circumstances, the software that supports the business must also evolve and change.
- Although there has been a demarcation between development and evolution (maintenance) this is increasingly irrelevant as fewer and fewer systems are completely new.



Key points

- Software processes are the activities involved in producing a software system. Software process models are abstract representations of these processes.
- General process models describe the organization of software processes. Examples of these general models include the ‘waterfall’ model, incremental development, and reuse-oriented development.
- Requirements engineering is the process of developing a software specification.
- Design and implementation processes are concerned with transforming a requirements specification into an executable software system.
- Software validation is the process of checking that the system conforms to its specification and that it meets the real needs of the users of the system.
- Software evolution takes place when you change existing software systems to meet new requirements. The software must evolve to remain useful.

Coping with change

- Change is inevitable in all large software projects.
  - Business changes lead to new and changed system requirements
  - New technologies open up new possibilities for improving implementations
  - Changing platforms require application changes
- Change leads to rework so the costs of change include both rework (e.g. re-analyzing requirements) as well as the costs of implementing new functionality

Reducing the costs of rework

- Change avoidance, where the software process includes activities that can anticipate possible changes before significant rework is required.
  - For example, a prototype system may be developed to show some key features of the system to customers.
- Change tolerance, where the process is designed so that changes can be accommodated at relatively low cost.
  - This normally involves some form of incremental development. Proposed changes may be implemented in increments that have not yet been developed. If this is impossible, then only a single increment (a small part of the system) may have be altered to incorporate the change.

Two ways of coping with change and changing system requirements:

Prototyping

Increment delivery

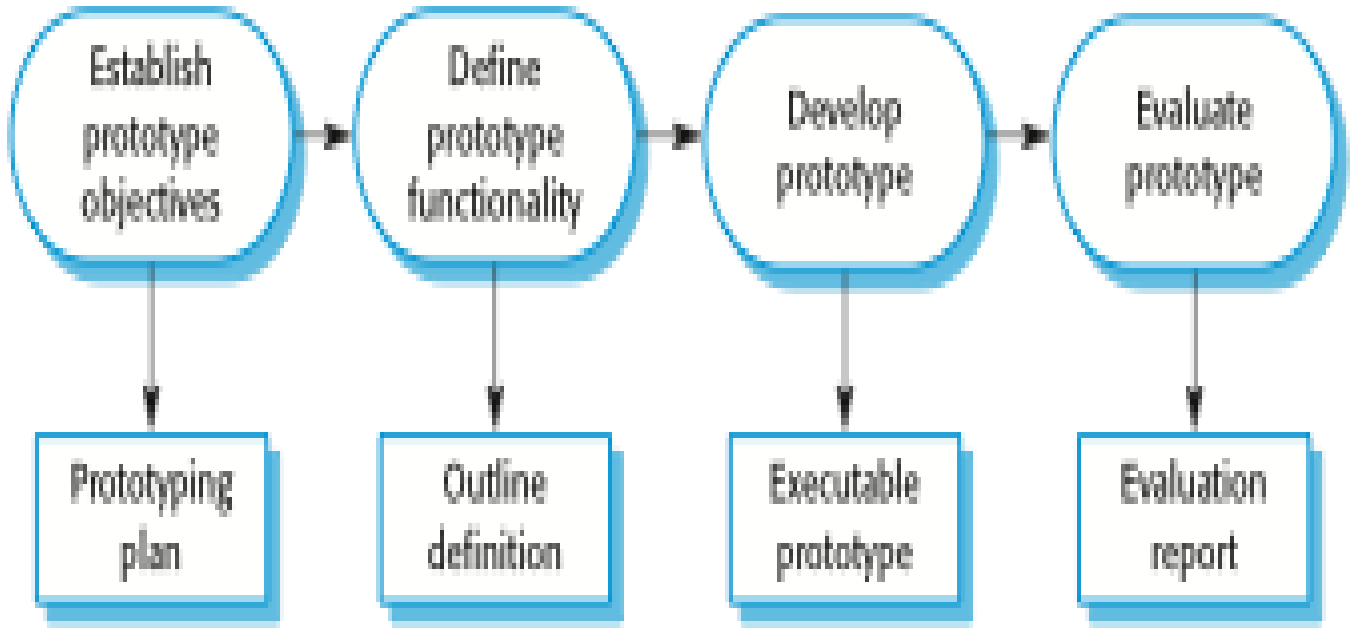
Software prototyping

- A prototype is an initial version of a system used to demonstrate concepts and try out design options, and find out more about the problem and its possible solutions.
- Where a version of the system or part of the system is developed quickly to check the customer requirements .
- Rapid ,iterative development of the prototype is essential ,so that costs are controlled and system stakeholders can experiment with the prototype early in the software process.
- A prototype can be used in:
  - The requirements engineering process to help with requirements elicitation and validation;
  - In design processes to explore particular software solutions options and develop a UI design;
  - In the testing process to run back-to-back tests.

Benefits of prototyping

- Improved system usability.
- A closer match to users' real needs.
- Improved design quality.
- Improved maintainability.
- Reduced development effort.

The process of prototype development



Prototype development process

- May be based on rapid prototyping languages or tools
- May involve leaving out functionality
  - Prototype should focus on areas of the product that are not well-understood;
  - Error checking and recovery may not be included in the prototype;
  - Focus on functional rather than non-functional requirements such as reliability and security

Throw-away prototypes

- Prototypes should be discarded after development as they are not a good basis for a production system:
  - It may be impossible to tune the system to meet non-functional requirements;
  - Prototypes are normally undocumented;
  - The prototype structure is usually degraded through rapid change;
  - The prototype probably will not meet normal organisational quality standards.

Incremental development and delivery

- Incremental development
  - Develop the system in increments and evaluate each increment before proceeding to the development of the next increment;
  - Normal approach used in agile methods;
  - Evaluation done by user/customer proxy.
- Incremental delivery
  - Deploy an increment for use by end-users;
  - More realistic evaluation about practical use of software;
  - Difficult to implement for replacement systems as increments have less functionality than the system being replaced.

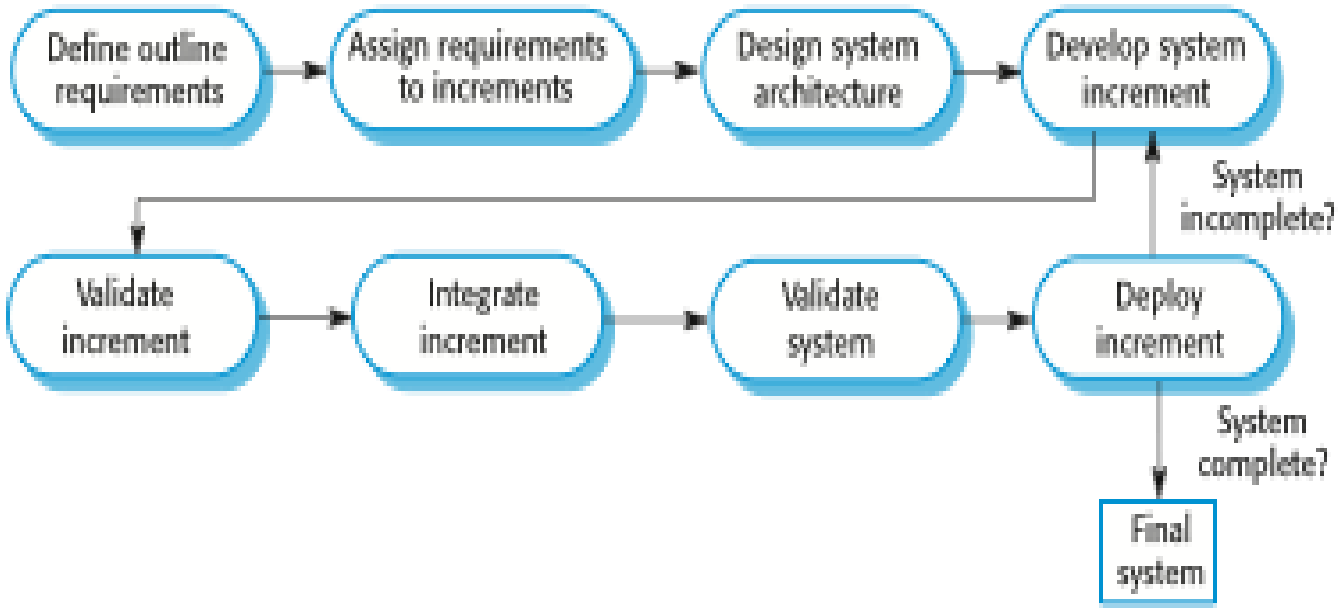
Incremental delivery

- Rather than deliver the system as a single delivery, the development and delivery is broken down into increments with each increment delivering part of the required functionality.

Incremental delivery

- Is an approach to software development where some of the developed increments are delivered to the customer and deployed for use in an operational environment.
- In an incremental delivery process ,the customer identify the services to be provided by the system.
- They identify which services are most/less important.
- User requirements are prioritised and the highest priority requirements are included in early increments.
- Once the development of an increment is started, the requirements are frozen though requirements for later increments can continue to evolve.
- Once an increment is completed and delivered, customers can put into service. This means that they take early delivery of part of the system functionality.
- They can experiment with the system and helps to clarify their requirements for later system increments.
- As new increments are completed ,they are integrated with the existing increments so that functionality improves with each delivered increment.

Incremental delivery



Incremental delivery advantages

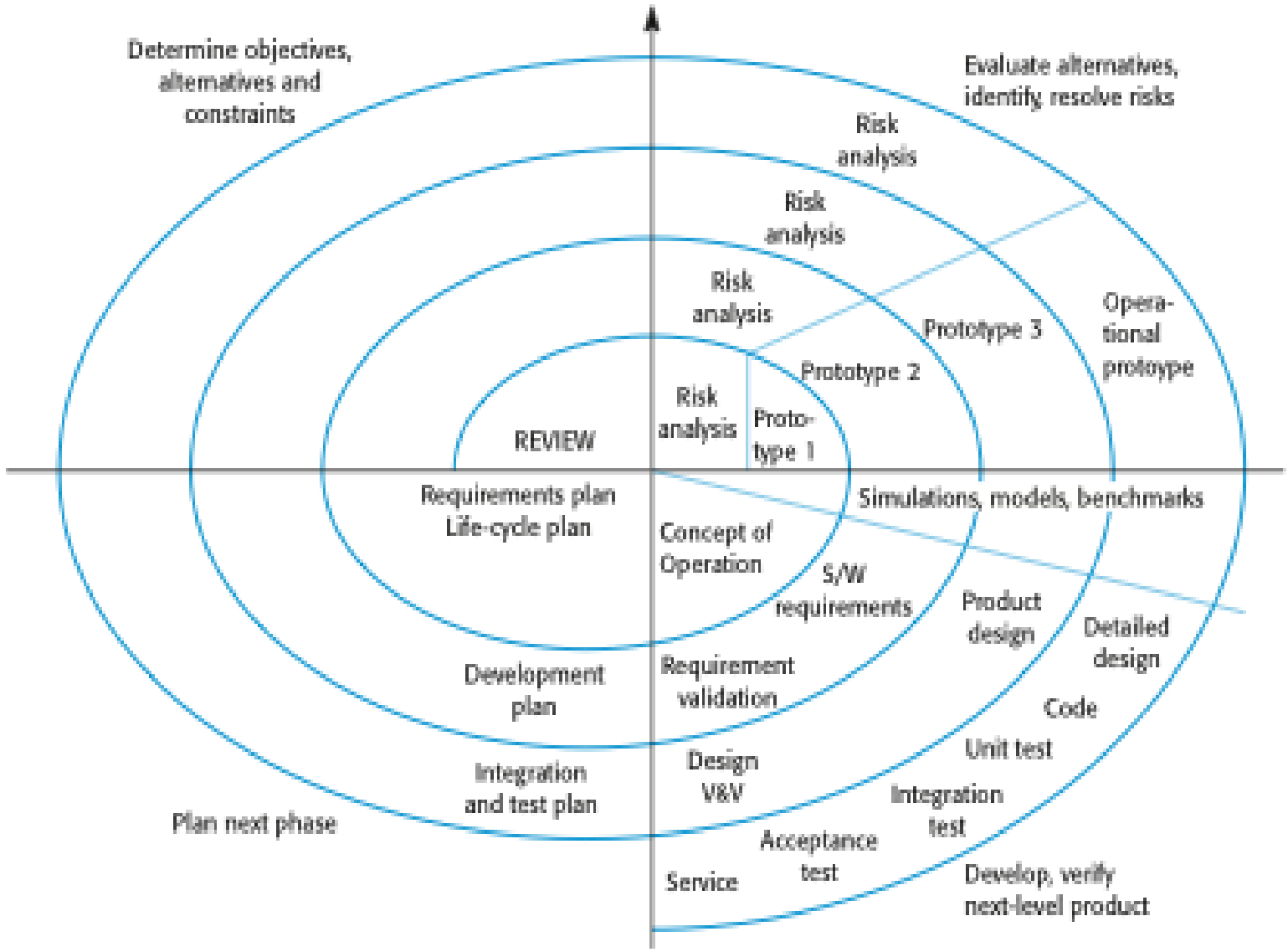
- Customer value can be delivered with each increment so system functionality is available earlier.
- Early increments act as a prototype to help elicit requirements for later increments.
- Lower risk of overall project failure.
- The highest priority system services tend to receive the most testing.

Incremental delivery problems

- Most systems require a set of basic facilities that are used by different parts of the system.
  - As requirements are not defined in detail until an increment is to be implemented, it can be hard to identify common facilities that are needed by all increments.
- The essence of iterative processes is that the specification is developed in conjunction with the software.
  - However, this conflicts with the procurement model of many organizations, where the complete system specification is part of the system development contract.

Boehm’s spiral model

- Process is represented as a spiral rather than as a sequence of activities with backtracking.
- Each loop in the spiral represents a phase in the process.
- The innermost loop might be concerned with system feasibility,next is requirement definition,system design,….
- No fixed phases such as specification or design - loops in the spiral are chosen depending on what is required.
- Risks are explicitly assessed and resolved throughout the process.


Determine objectives, alternatives and constraints
Evaluate alternatives, identify, resolve risks
Risk analysis
Risk analysis
Risk analysis
Risk analysis
Prototype 3
Prototype 2
Opera-tional prototype
Proto-type 1
REVIEW
Requirements plan
Life-cycle plan
Concept of Operation
Simulations, models, benchmarks
S/W requirements
Product design
Detailed design
Development plan
Requirement validation
Code
Unit test
Integration and test plan
Design V&V
Integration test
Plan next phase
Acceptance test
Service
Develop, verify next-level product

Spiral model sectors

- Objective setting
  - Specific objectives for the phase are identified.
  - Constraints on the process and the product are identified and a detailed management plan is drawn up.
  - Project risks are identified.
- Risk assessment and reduction
  - Risks are assessed and activities put in place to reduce the key risks.
- Development and validation
  - A development model for the system is chosen which can be any of the generic models.
- Planning
  - The project is reviewed and the next phase of the spiral is planned.

Spiral model usage

- Spiral model has been very influential in helping people think about iteration in software processes and introducing the risk-driven approach to development.
- In practice, however, the model is rarely used as published for practical software development.

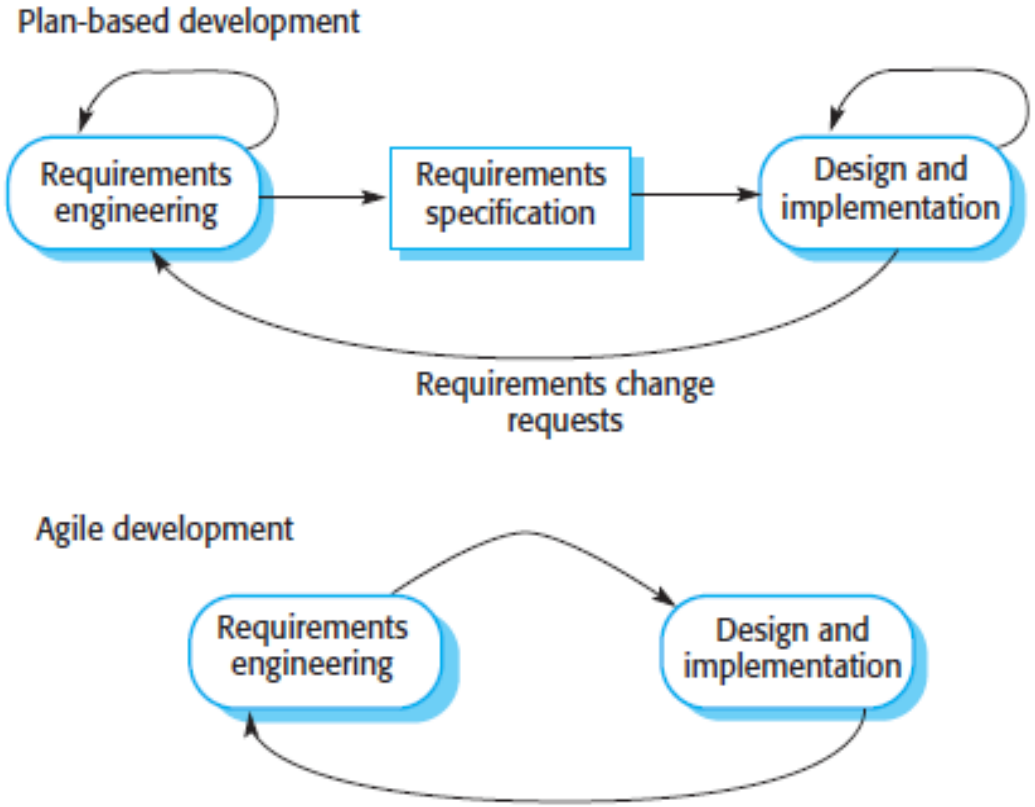
AGILE software development

- Rapid development and delivery is now often the most important requirement for software systems
  - Businesses operate in a fast –changing requirement and it is practically impossible to produce a set of stable software requirements
  - Software has to evolve quickly to reflect changing business needs.
- Agile development
  - Specification, design and implementation are inter-leaved
  - System is developed as a series of versions with stakeholders involved in version evaluation
  - **Extensive tool support is used to support the development process.**
  - User interfaces are often developed using an IDE and graphical toolset.

Agile methods

- Dissatisfaction with the overheads involved in software design methods of the 1980s and 1990s led to the creation of agile methods. These methods:
  - Focus on the code rather than the design
  - Are based on an iterative approach to software development
  - Are intended to deliver working software quickly and evolve this quickly to meet changing requirements.
- The aim of agile methods is to reduce overheads in the software process (e.g. by limiting documentation) and to be able to respond quickly to changing requirements without excessive rework.

Plan driven/agile



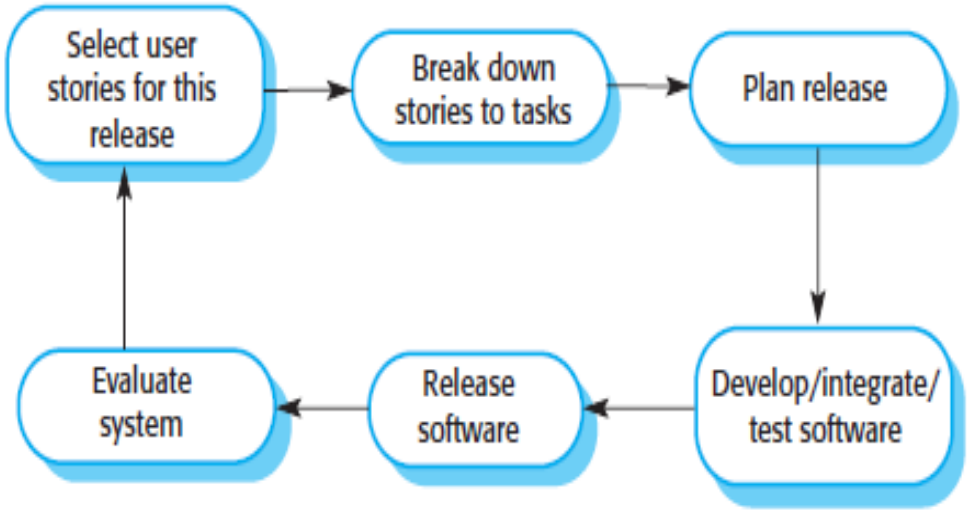
Agile manifesto

- *We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:*
  - *Individuals and interactions over processes and tools*
    Working software over comprehensive documentation
    Customer collaboration over contract negotiation
    Responding to change over following a plan
- *That is, while there is value in the items on the right, we value the items on the left more.*

The principles of agile methods

Principle	Description
Customer involvement	Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system.
Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment.
People not process	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.
Embrace change	Expect the system requirements to change and so design the system to accommodate these changes.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.

Agile development techniques:XP release cycle



In XP, requirements are expressed as scenarios (called user stories), which are implemented directly as a series of tasks. Programmers work in pairs and develop tests for each task before writing the code. All tests must be successfully executed when new code is integrated into the system. There is a short time gap between releases of the system.

Extreme programming was an agile practices that were summarized and reflect the principles of the agile manifesto:

- 1. Incremental development is supported through small, frequent releases of the system. Requirements are based on simple customer stories or scenarios that are used as a basis for deciding what functionality should be included in a system increment.
- 2. Customer involvement is supported through the continuous engagement of the customer in the development team. The customer representative takes part in the development and is responsible for defining acceptance tests for the system.
- 3. People, not process, are supported through pair programming, collective ownership of the system code, and a sustainable development process that does not involve excessively long working hours.
- 4. Change is embraced through regular system releases to customers, test-first development, refactoring to avoid code degeneration, and continuous integration of new functionality.
- 5. Maintaining simplicity is supported by constant refactoring that improves code quality and by using simple designs that do not unnecessarily anticipate future changes to the system.

Extreme programming practices

Principle or practice	Description
Incremental planning	Requirements are recorded on story cards and the stories to be included in a release are determined by the time available and their relative priority. The developers break these stories into development 'Tasks'. See Figures 3.5 and 3.6.
Small releases	The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.
Simple design	Enough design is carried out to meet the current requirements and no more.
Test-first development	An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented.
Refactoring	All developers are expected to refactor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable.

Pair programming	Developers work in pairs, checking each other's work and providing the support to always do a good job.
Collective ownership	The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers take responsibility for all of the code. Anyone can change anything.
Continuous integration	As soon as the work on a task is complete, it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.
Sustainable pace	Large amounts of overtime are not considered acceptable as the net effect is often to reduce code quality and medium term productivity
On-site customer	A representative of the end-user of the system (the customer) should be available full time for the use of the XP team. In an extreme programming process, the customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.

XP practices

- **User stories:**
- Software requirements always change. In Agile methods , requirements elicitation is integrated with development by the idea of "**user stories**" where a user story is a scenario of use that might be experienced by a system user.
- After the discussion of development team with customer, they develop a "**story card**" that briefly describes a story that encapsulates the customer needs. The development team then aims to implement that scenario in a future release of the software.
- User stories may be used in planning system iterations. Once the story cards have been developed, the development team breaks these down into tasks and estimates the effort and resources required for implementing each task.
- This usually involves discussions with the customer to refine the requirements. The customer then prioritizes the stories for implementation, choosing those stories that can be used immediately to deliver useful business support.
- The intention is to identify useful functionality that can be implemented in about two weeks, when the next release of the system is made available to the customer.

- If changes are required for a system that has already been delivered, new story cards are developed and again, the customer decides whether these changes should have priority over new functionality.
- User stories can be helpful in getting users involved in suggesting requirements during an initial predevelopment requirements elicitation activity.
- **Cons:**
- The principal problem with user stories is completeness. It is difficult to judge if enough user stories have been developed to cover all of the essential requirements of a system.
- It is also difficult to judge if a single story gives a true picture of an activity. Experienced users are often so familiar with their work that they leave things out when describing it.
- **Refactoring:**
- Changes will always have to be made to the code being developed. Refactoring means that the programming team look for possible improvements to the software and implements them immediately.
- Refactoring improves the software structure and readability and avoids the structural deterioration that naturally occurs when software is changed.

Test-first development:

- Extreme Programming developed a new approach to program testing to address the difficulties of testing without a specification. Testing is automated and is central to the development process, and development cannot proceed until all tests have been successfully executed. The key features of testing in XP are:

1. test-first development:

- Write test before write the code.
- Writing tests implicitly defines both an interface and a specification of behaviour for the functionality being developed.
- Problems of requirements and interface misunderstandings are reduced.
- Test-first development requires there to be a clear relationship between system requirements and the code implementing the corresponding requirements.
- In XP, this relationship is clear because the story cards representing the requirements are broken down into tasks and the tasks are the principal unit of implementation.
- In test-first development, the task implementers have to thoroughly understand the specification so that they can write tests for the system.
- This means that ambiguities and omissions in the specification have to be clarified before implementation begins. It also avoids the problem of “test-lag.” This may happen when the developer of the system works at a faster pace than the tester.

2. Incremental test development from scenarios,

- Develop each tasks, so that the development schedule can be maintained.

3. User involvement in the test development and validation, and

- The role of the customer in the testing process is to help develop acceptance tests for the stories that are to be implemented in the next release of the system.

4. The use of automated testing frameworks.

- Test automation is essential for test-first development. Tests are written as executable Components before the task is implemented. These testing components should be stand-alone, should simulate the submission of input to be tested, and should check that the result meets the output specification.
- An automated test framework is a system that makes it easy to write executable tests and submit a set of tests for execution. JUnit is a widely used example of an automated testing framework for Java programs.

Pair programming:

- The programming pair sits at the same computer to develop the software. However, the same pair do not always program together. Rather, pairs are created dynamically so that all team members work with each other during the development process.

Pair programming has a number of advantages.

1. **It supports the idea of collective ownership and responsibility for the system**. This reflects Weinberg's idea of egoless programming where the software is owned by the team as a whole and individuals are not held responsible for problems with the code. Instead, the team has collective responsibility for resolving these problems.

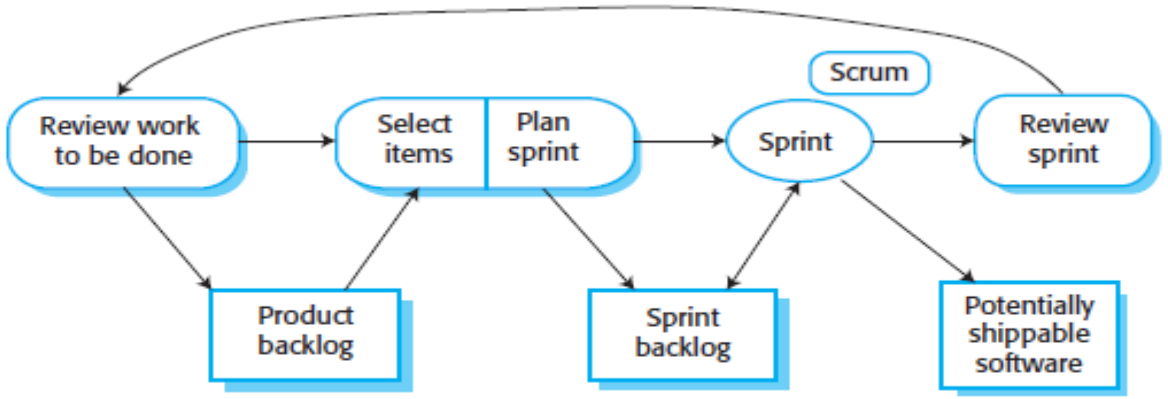
2. It acts as an informal review process because each line of code is looked at by at least

two people.

3. It encourages refactoring to improve the software structure.

Agile Project Management

- The principal responsibility of software project managers is to manage the project so that the software is delivered on time and within the planned budget for the project.
- **Scrum**
- The Scrum approach is a general agile method and focus is on managing iterative development rather than specific agile practices.
- The Scrum Process



The Sprint Cycle

- Each process iteration produces a product increment that could be delivered to customers.
- The starting point for planning is the **product backlog**, which is the list of work to be done on the project. —the list of items such as product features, requirements, user stories and engineering improvement that have to be worked on by the Scrum team.
- The product owner has a responsibility to ensure the level of specification is appropriate for the work to be done.
- Each sprint cycle lasts a fixed length of time, which is usually between 2 and 4 weeks. At the beginning of each cycle, the Product Owner prioritizes the items on the product backlog to define which are the most important items to be developed in that cycle.
- Sprints are never extended to take account of unfinished work. Items are returned to the product backlog if these cannot be completed within the allocated time for the sprint.
- The whole team is then involved in selecting which of the highest priority items they believe can be completed. They then estimate the time required to complete these items. To make these estimates, they use the velocity attained in previous sprints, that is, how much of the backlog could be covered in a single sprint. This leads to the creation of a **sprint backlog**— the work to be done during that sprint.
- The team self-organizes to decide who will work on what, and the sprint begins.

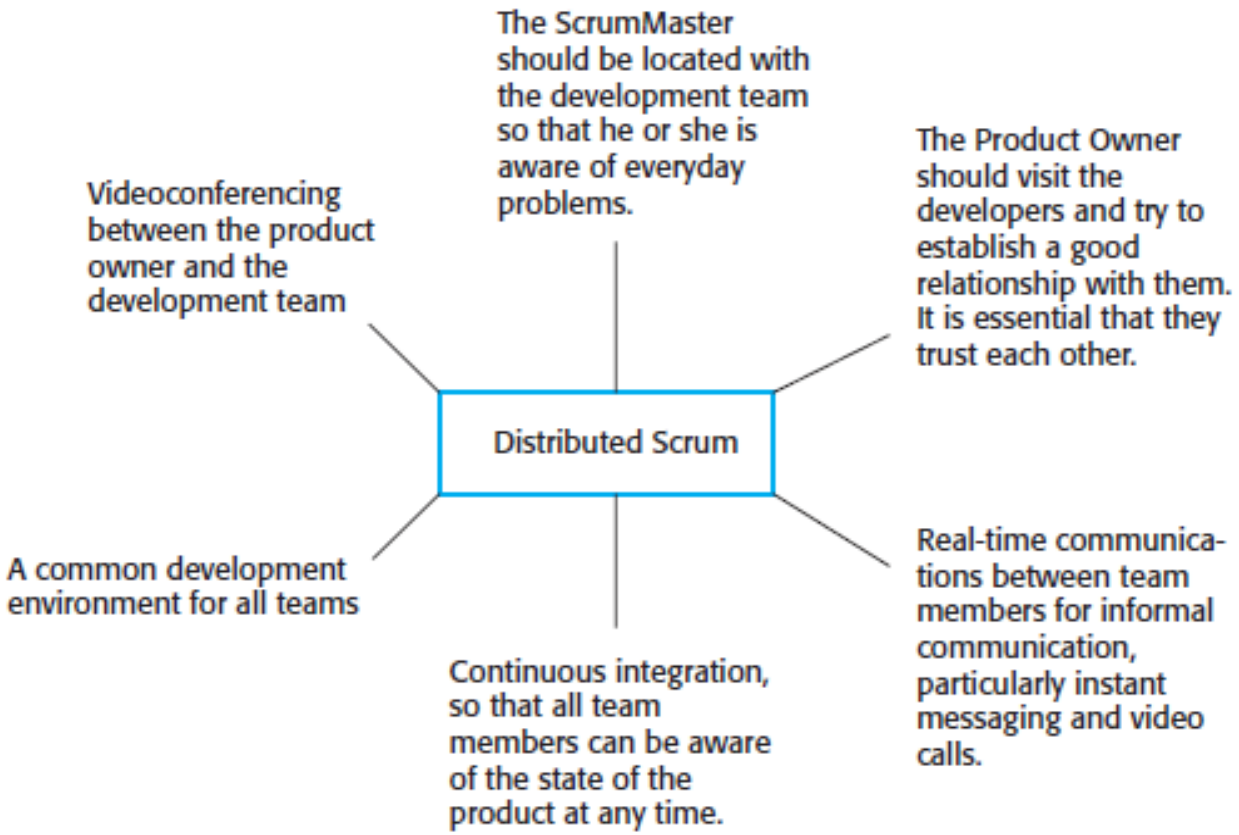
Teamwork in Scrum

- The ‘Scrum master’ is a facilitator who arranges daily meetings, tracks the backlog of work to be done, records decisions, measures progress against the backlog and communicates with customers and management outside of the team.
- The whole team attends short daily meetings (scrum)where all team members share information, describe their progress since the last meeting, problems that have arisen and what is planned for the following day.
- This means that everyone on the team knows what is going on and, if problems arise, can re-plan short-term work to cope with them, there is no top-down direction from the Scrum Master.
- Everyone participates in this short-term planning; the daily interactions among Scrum teams may be coordinated using a Scrum board. This is an office whiteboard that includes information and post-it notes about the Sprint backlog, work done, unavailability of staff, and so on. This is a shared resource for the whole team, and anyone can change or move items on the board. It means that any team member can, at a glance, see what others are doing and what work remains to be done.
- At the end of each sprint, there is a review meeting, which involves the whole team. This meeting has two purposes. First, it is a means of process improvement. The team reviews the way they have worked and reflects on how things could have been done better. Second, it provides input on the product and the product state for the product backlog review that precedes the next sprint

Scrum benefits

- The product is broken down into a set of manageable and understandable chunks.
- Unstable requirements do not hold up progress.
- The whole team have visibility of everything and consequently team communication is improved.
- Customers see on-time delivery of increments and gain feedback on how the product works.
- Trust between customers and developers is established and a positive culture is created in which everyone expects the project to succeed.

- For offshore development, the product owner is in a different country from the development team, which may also be distributed. Figure shows the requirements for Distributed Scrum



Key points

- Agile methods are incremental development methods that focus on rapid development, frequent releases of the software, reducing process overheads and producing high-quality code. They involve the customer directly in the development process.
- The decision on whether to use an agile or a plan-driven approach to development should depend on the type of software being developed, the capabilities of the development team and the culture of the company developing the system.
- Extreme programming is a well-known agile method that integrates a range of good programming practices such as frequent releases of the software, continuous software improvement and customer participation in the development team.

Key points

- A particular strength of extreme programming is the development of automated tests before a program feature is created. All tests must successfully execute when an increment is integrated into a system.
- The Scrum method is an agile method that provides a project management framework. It is centred round a set of sprints, which are fixed time periods when a system increment is developed.
- Scaling agile methods for large systems is difficult. Large systems need up-front design and some documentation.

Module 2 :

Requirement Analysis and Design (8 hours)

Functional and non-functional requirements, Requirements engineering processes. Requirements elicitation, Requirements validation, Requirements change,Traceability Matrix.

Developing use cases, Software Requirements Specification Template, Personas, Scenarios, User stories, Feature identification.

Design concepts - Design within the context of software engineering, Design Process, Design concepts, Design Model.

Architectural Design - Software Architecture, Architectural Styles, Architectural considerations, Architectural Design

Component level design - What is a component?, Designing Class-Based Components, Conducting Component level design, Component level design for web-apps

. Template of a Design Document as per “IEEE Std 1016-2009 IEEE Standard for Information Technology Systems Design Software Design Descriptions”.

Case study: The Ariane 5 launcher failure.

Requirements engineering

- The process of establishing the services that the customer requires from a system and the constraints under which it operates and is developed.
- The requirements themselves are the descriptions of the system services and constraints that are generated during the requirements engineering process.
- It may range from a high-level abstract statement of a service or of a system constraint to a detailed mathematical functional specification.

Types of requirement

- User requirements
  - Statements in natural language plus diagrams of the services the system provides and its operational constraints. Written for customers.
- System requirements
  - A structured document setting out detailed descriptions of the system's functions, services and operational constraints. Defines what should be implemented so may be part of a contract between client and contractor.

User and system requirements

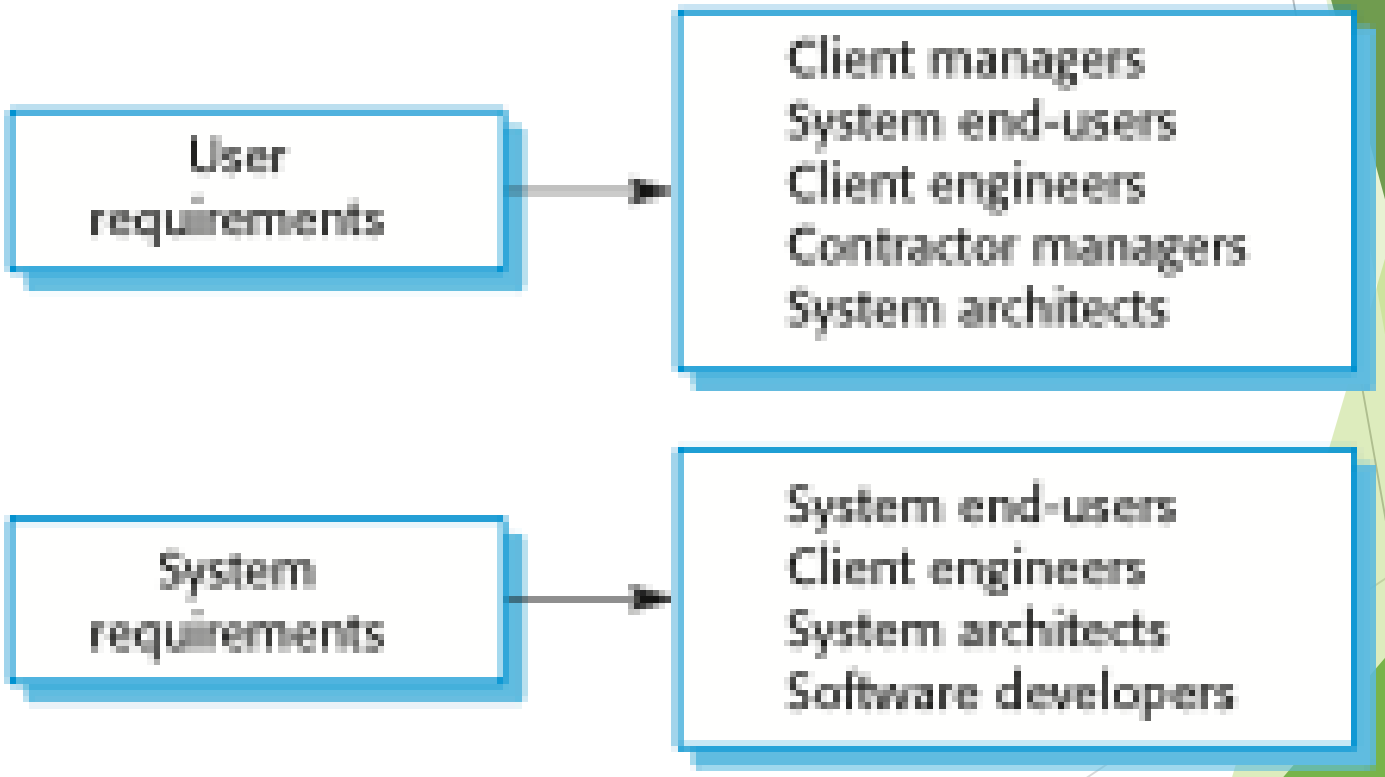
User requirement definition

1. The MHC-PMS shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month.

System requirements specification

1.1 On the last working day of each month, a summary of the drugs prescribed, their cost and the prescribing clinics shall be generated.
1.2 The system shall automatically generate the report for printing after 17.30 on the last working day of the month.
1.3 A report shall be created for each clinic and shall list the individual drug names, the total number of prescriptions, the number of doses prescribed and the total cost of the prescribed drugs.
1.4 If drugs are available in different dose units (e.g. 10mg, 20 mg, etc.) separate reports shall be created for each dose unit.
1.5 Access to all cost reports shall be restricted to authorized users listed on a management access control list.

Readers of different types of requirements specification



Functional and non-functional requirements

- Functional requirements
  - Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.
  - May state what the system should not do.
- Non-functional Requirement
- constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.
  - Often apply to the system as a whole rather than individual features or services.
- Domain requirements
  - Constraints on the system from the domain of operation

Functional requirements

- Describe functionality or system services.
- Depend on the type of software, expected users and the type of system where the software is used.
- Functional user requirement high-level statements of what the system should do.
- Functional system requirements should describe the system services in detail.

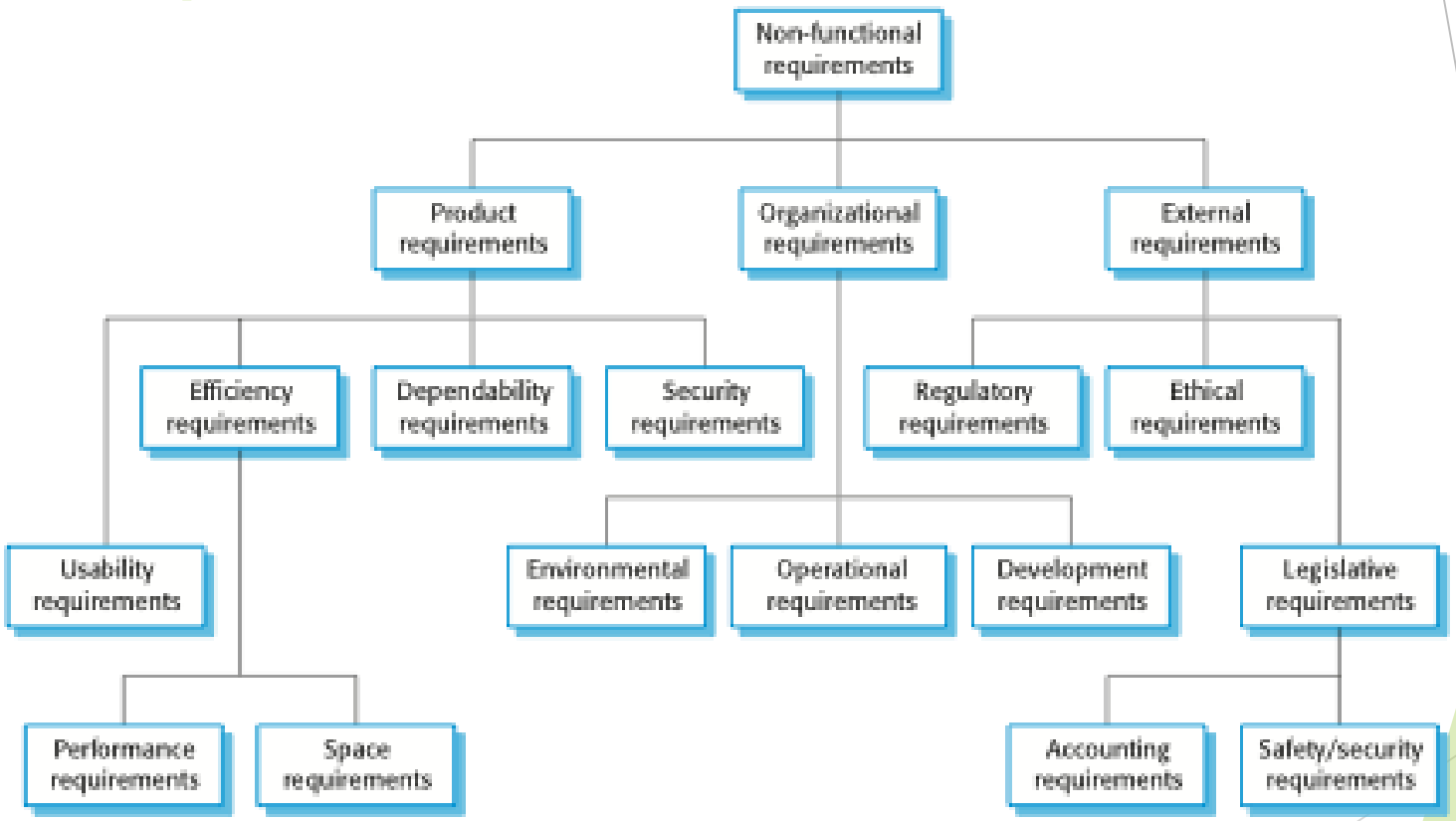
Functional requirements for the MHC-PMS

- A user shall be able to search the appointments lists for all clinics.
- The system shall generate each day, for each clinic, a list of patients who are expected to attend appointments that day
- Each staff member using the system shall be uniquely identified by his or her 8-digit employee number.

Non-functional requirements

- These define system properties and constraints e.g. reliability, response time and storage requirements. Constraints are I/O device capability, system representations, etc.
- Process requirements may also be specified mandating a particular IDE, programming language or development method.
- Non-functional requirements may be more critical than functional requirements. If these are not met, the system may be useless.

Types of nonfunctional requirement



Non-functional classifications

- Product requirements
  - Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, etc.
- Organizational requirements
  - Requirements which are a consequence of organisational policies and procedures e.g. process standards used, implementation requirements, etc.
- External requirements
  - Requirements which arise from factors which are external to the system and its development process e.g. interoperability requirements, legislative requirements, etc.

Usability requirements

- The system should be easy to use by medical staff and should be organized in such a way that user errors are minimized. (Goal)
- Medical staff shall be able to use all the system functions after fourr hours training. After this training, the average number of errors made by experienced users shall not exceed two per hour of system use. (Testable non-functional requirement)

Metrics for specifying nonfunctional requirements

Property	Measure
Speed	Processed transactions/second User/event response time Screen refresh time
Size	Mbytes Number of ROM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems

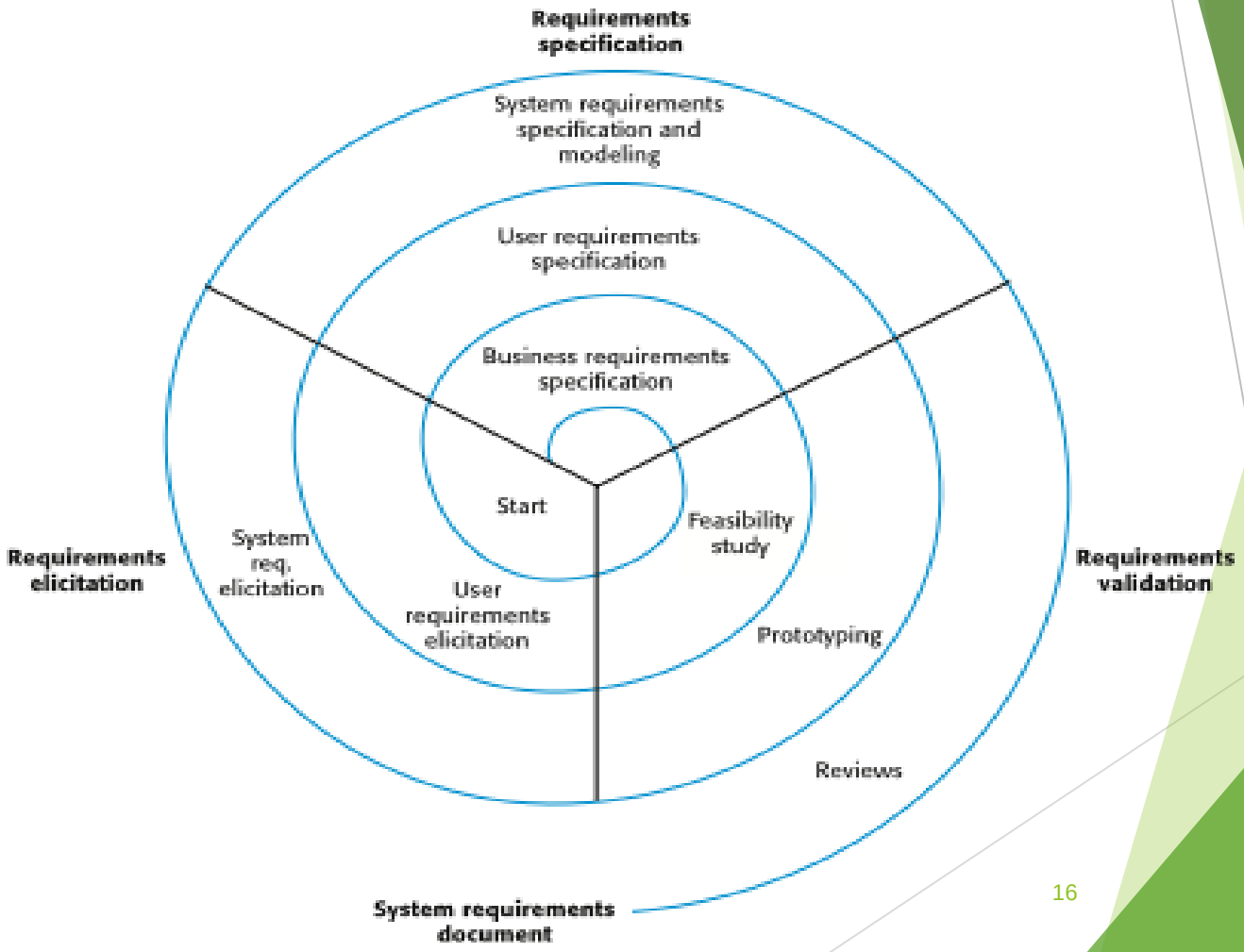
Key points

- Requirements for a software system set out what the system should do and define constraints on its operation and implementation.
- Functional requirements are statements of the services that the systems must provide are descriptions of how some computations must be carried out.
- Non-functional requirements often constrain the system being developed and the development process being used.
- They often relate to the emergent properties of the system and therefore apply to the system as a whole.

Requirements engineering processes

- The processes used for RE vary widely depending on the application domain, the people involved and the organisation developing the requirements.
- However, there are a number of generic activities common to all processes
  - Requirements elicitation;
  - Requirements analysis;
  - Requirements validation;
  - Requirements management.
- In practice, RE is an iterative activity in which these processes are interleaved.

A spiral view of the requirements engineering process



1. Requirements elicitation and analysis

- Sometimes called requirements elicitation or requirements discovery.
- Involves technical staff working with customers to find out about the application domain, the services that system should provide and the system's operational constraints.
- May involve end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc. These are called *stakeholders*.

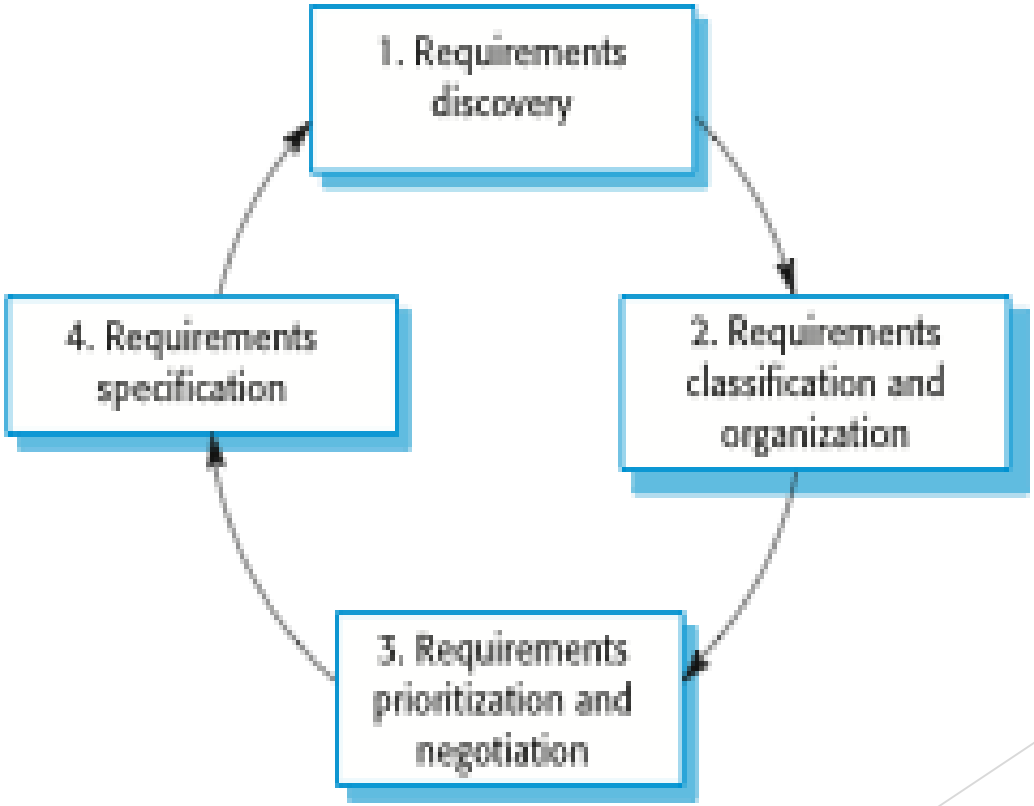
Problems of requirements analysis

- Stakeholders don't know what they really want.
- Stakeholders express requirements in their own terms.
- Different stakeholders may have conflicting requirements.
- Organisational and political factors may influence the system requirements.
- The requirements change during the analysis process. New stakeholders may emerge and the business environment may change.

Requirements elicitation and analysis

- Software engineers work with a range of system stakeholders to find out about the application domain, the services that the system should provide, the required system performance, hardware constraints, other systems, etc.
- Stages include:
  - **Requirements discovery and understanding:** process of interacting with stake holders to discover their requirements.
  - **Requirements classification and organization** :this activity takes the unstructured collection of requirements, groups related requirements and organizes them into coherent clusters.
  - **Requirements prioritization and negotiation**: when multiple stakeholders are involved ,requirements will conflict. This activity is concerned with prioritizing requirements and finding and resolving requirements conflicts through negotiation.
  - **Requirements specification(documentation)**:requirements are documented and input into the next round of spiral.

The requirements elicitation and analysis process



Requirements discovery(elicitation techniques)

- The process of gathering information about the required and existing systems and distilling the user and system requirements from this information.
- Interaction is with system stakeholders from managers to external regulator
- Systems normally have a range of stakeholders.

Stakeholders in the MHC-PMS

- Patients whose information is recorded in the system.
- Doctors who are responsible for assessing and treating patients.
- Medical receptionists who manage patients' appointments.
- IT staff who are responsible for installing and maintaining the system.

Reqmt Elicitation Techniques:1.Interviewing

- Formal or informal interviews with stakeholders are part of most RE processes.
- Types of interview
- Closed interviews : stakeholders answers based on pre-determined list of questions
  - Open interviews :in which there is no predefined agenda,where various issues are explored with stakeholders.
- Effective interviewing
  - Be open-minded, avoid pre-conceived ideas about the requirements and are willing to listen to stakeholders.
  - Prompt the interviewee to get discussions going using a springboard question, a requirements proposal, or by working together on a prototype system.

2. Stories and Scenarios

- Scenarios are real-life examples of how a system can be used.
- They should include
  - A description of the starting situation;
  - A description of the normal flow of events;
  - A description of what can go wrong;
  - Information about other concurrent activities;
  - A description of the state when the scenario finishes.

3. Ethnography

- A social scientist spends a considerable time observing and analysing how people actually work.
- Social and organisational factors of importance may be observed.
- Ethnographic studies have shown that work is usually richer and more complex than suggested by simple system models.

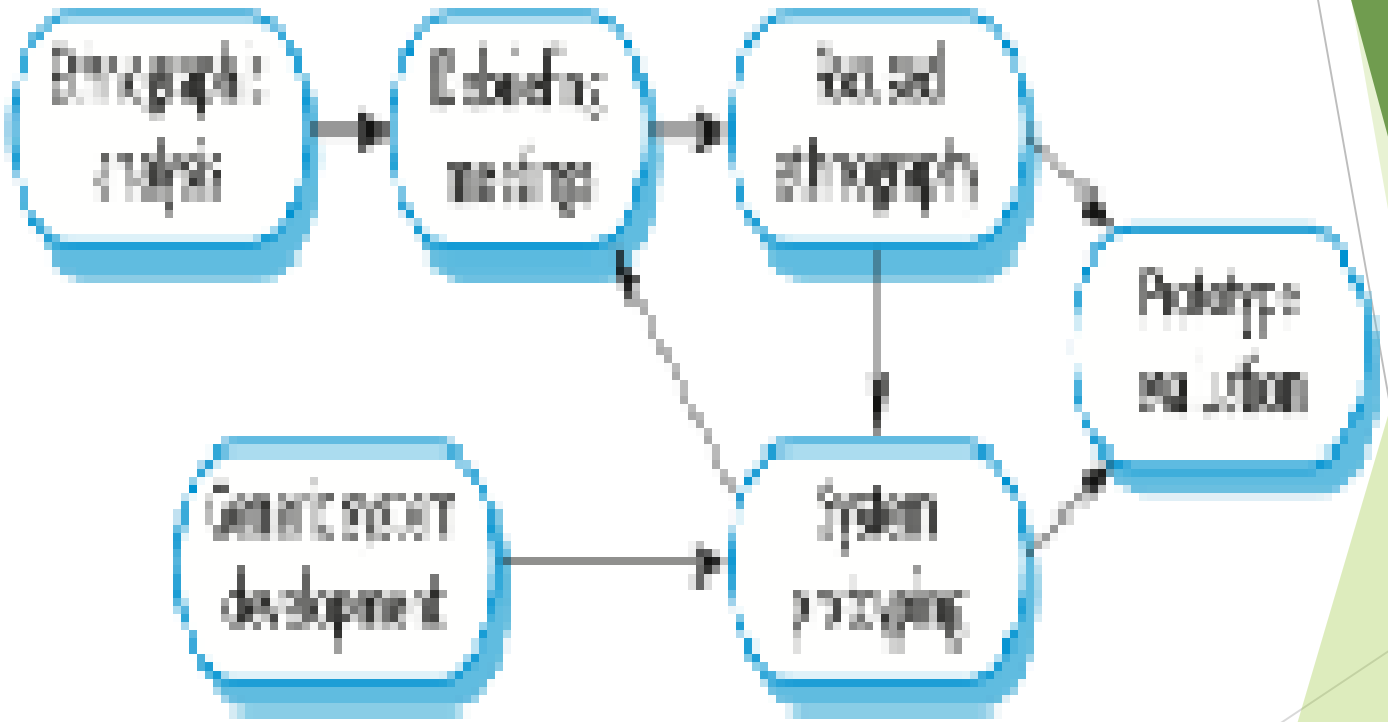
Scope of ethnography

- Requirements that are derived from cooperation and awareness of other people's activities.
  - Awareness of what other people are doing leads to changes in the ways in which we do things.

Focused ethnography

- Developed in a project studying the air traffic control process
- Combines ethnography with prototyping
- The problem with ethnography is that it studies existing practices which may have some historical basis which is no longer relevant.

Ethnography and prototyping for requirements analysis



2. Requirements specification

- The process of writing don the user and system requirements in a requirements document.
- User requirements have to be understandable by end-users and customers who do not have a technical background.
- System requirements are more detailed requirements and may include more technical information.
- The requirements may be part of a contract for the system development
  - It is therefore important that these are as complete as possible.

Ways of writing a system requirements specification

Notation	Description
Natural language	The requirements are written using numbered sentences in natural language. Each sentence should express one requirement.
Structured natural language	The requirements are written in natural language on a standard form or template. Each field provides information about an aspect of the requirement.
Design description languages	This approach uses a language like a programming language, but with more abstract features to specify the requirements by defining an operational model of the system. This approach is now rarely used although it can be useful for interface specifications.
Graphical notations	Graphical models, supplemented by text annotations, are used to define the functional requirements for the system; UML use case and sequence diagrams are commonly used.
Mathematical specifications	These notations are based on mathematical concepts such as finite-state machines or sets. Although these unambiguous specifications can reduce the ambiguity in a requirements document, most customers don't understand a formal specification. They cannot check that it represents what they want and are reluctant to accept it as a system contract

Natural language specification

- Requirements are written as natural language sentences supplemented by diagrams and tables.
- Used for writing requirements because it is expressive, intuitive and universal.

Guidelines for writing requirements

- Invent a standard format and use it for all requirements.
- Use language in a consistent way. Use shall for mandatory requirements, should for desirable requirements.
- Use text highlighting to identify key parts of the requirement.
- Avoid the use of computer jargon.
- Include an explanation (rationale) of why a requirement is necessary.

Problems with natural language

- Lack of clarity
  - Precision is difficult without making the document difficult to read.
- Requirements confusion
  - Functional and non-functional requirements tend to be mixed-up.
- Requirements amalgamation
  - Several different requirements may be expressed together.

Structured specifications

- An approach to writing requirements where the freedom of the requirements writer is limited and requirements are written in a standard way.
- This works well for some types of requirements

Form-based specifications

- Definition of the function or entity.
- Description of inputs and where they come from.
- Description of outputs and where they go to.
- Description of the action to be taken.
- Pre and post conditions (if appropriate).
- The side effects (if any) of the function.

Tabular specification

- Used to supplement natural language.
- Particularly useful when you have to define a number of possible alternative courses of action.

Use cases

- Use-cases are a scenario based technique in the UML which identify the actors in an interaction and which describe the interaction itself.

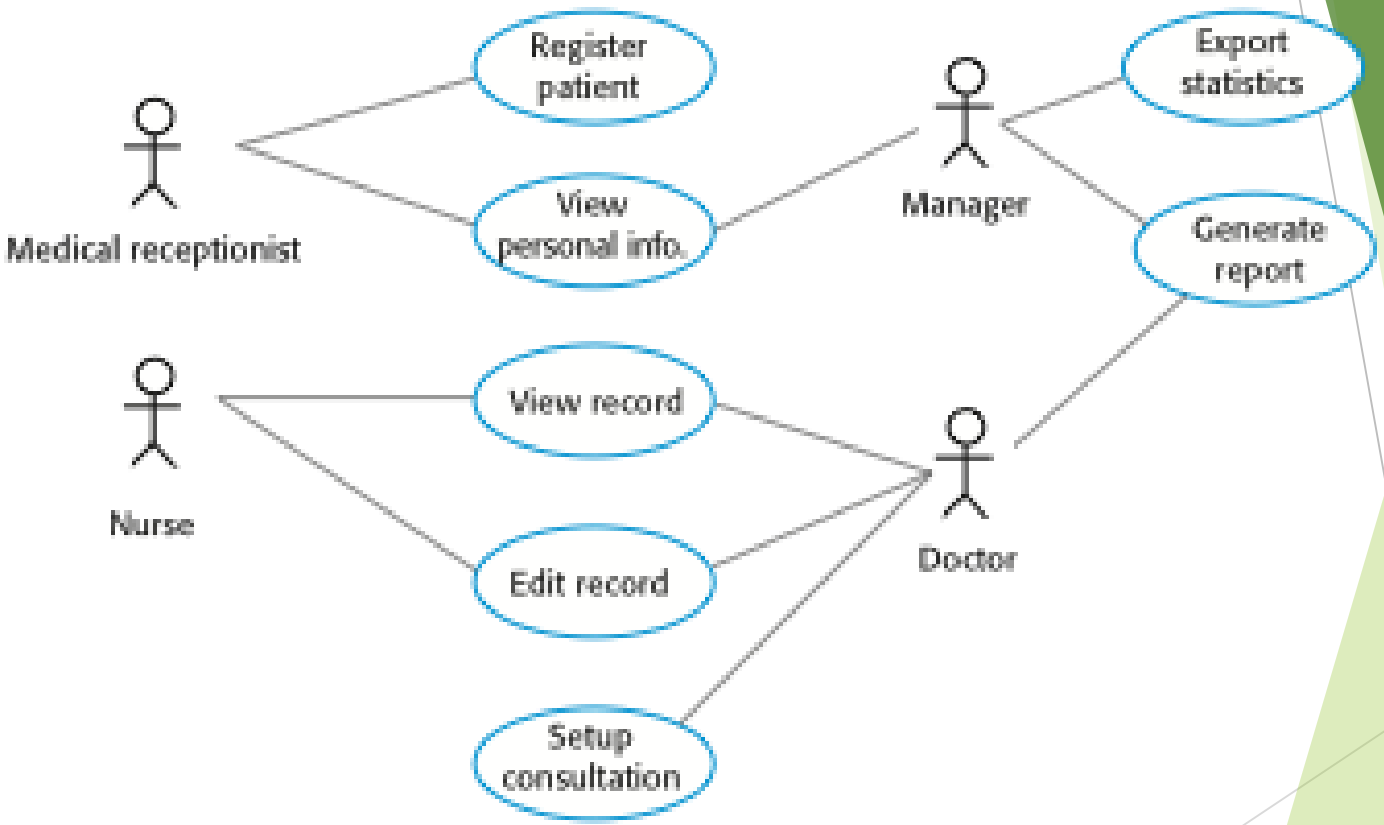
Developing Use cases

- A use case tells a stylized story about how an end user (playing one of a number of possible roles) interacts with the system under a specific c set of circumstances.
- The story may be narrative text, an outline of tasks or interactions, a template-based description, or a diagrammatic representation.
- The first step in writing a use case is to define the set of “actors” that will be

involved in the story.

- Actors are the different people (or devices) that use the system or product within the context of the function and behavior that is to be described.
- Actors represent the roles that people (or devices) play as the system operates.
- An actor is anything that communicates with the system or product and that is external to the system itself.

Use cases for the MHC-PMS



3. Requirements validation

- Concerned with demonstrating that the requirements define the system that the customer really wants.
- Requirements error costs are high so validation is very important.

Requirements checking

- Validity. Does the system provide the functions which best support the customer's needs?
- Consistency. Are there any requirements conflicts?
- Completeness. Are all functions required by the customer included?
- Realism. Can the requirements be implemented given available budget and technology
- Verifiability. Can the requirements be checked?

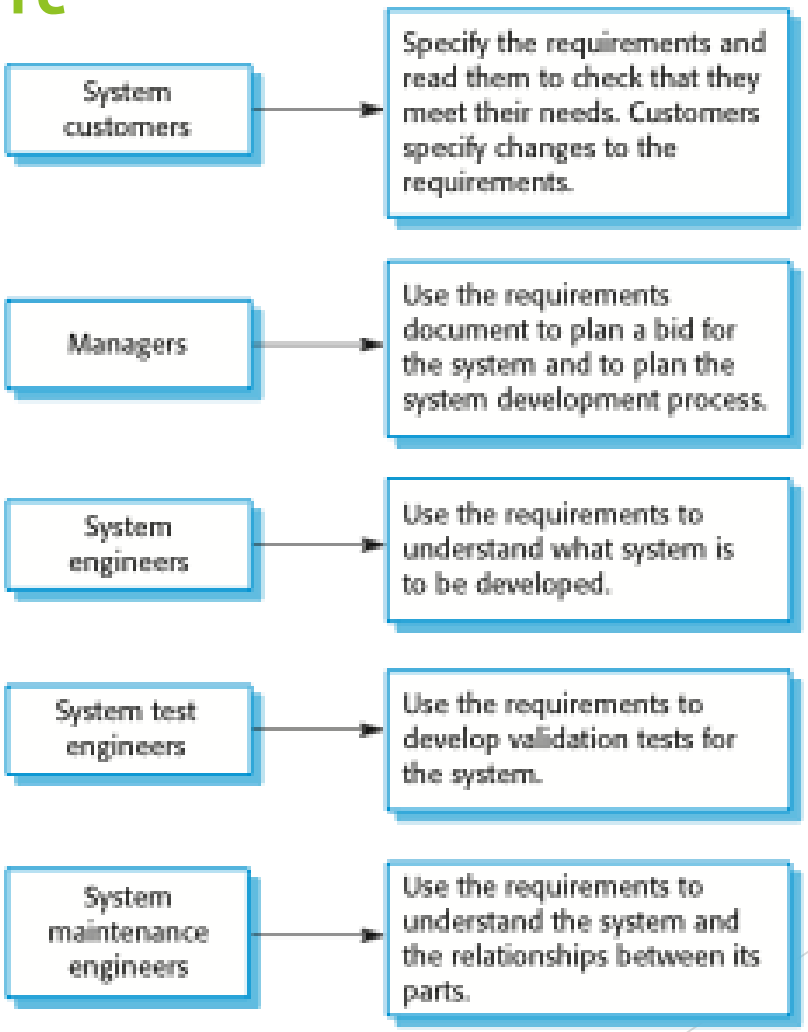
Requirements validation techniques

- Requirements reviews
  - Systematic manual analysis of the requirements:requirements are analysed systematically by a team of reviewers who check for errors and inconsistencies.
- Prototyping
  - Using an executable model of the system to check requirements.
- Test-case generation
  - Developing tests for requirements to check testability.

The software requirements document

- The software requirements document is the official statement of what is required of the system developers.
- Should include both a definition of user requirements and a specification of the system requirements.

Users of a requirements document



The structure of a requirements document

Chapter	Description
Preface	This should define the expected readership of the document and describe its version history, including a rationale for the creation of a new version and a summary of the changes made in each version.
Introduction	This should describe the need for the system. It should briefly describe the system's functions and explain how it will work with other systems. It should also describe how the system fits into the overall business or strategic objectives of the organization commissioning the software.
Glossary	This should define the technical terms used in the document. You should not make assumptions about the experience or expertise of the reader.
User requirements definition	Here, you describe the services provided for the user. The nonfunctional system requirements should also be described in this section. This description may use natural language, diagrams, or other notations that are understandable to customers. Product and process standards that must be followed should be specified.
System architecture	This chapter should present a high-level overview of the anticipated system architecture, showing the distribution of functions across system modules. Architectural components that are reused should be highlighted.

The structure of a requirements document

Chapter	Description
System requirements specification	This should describe the functional and nonfunctional requirements in more detail. If necessary, further detail may also be added to the nonfunctional requirements. Interfaces to other systems may be defined.
System models	This might include graphical system models showing the relationships between the system components and the system and its environment. Examples of possible models are object models, data-flow models, or semantic data models.
System evolution	This should describe the fundamental assumptions on which the system is based, and any anticipated changes due to hardware evolution, changing user needs, and so on. This section is useful for system designers as it may help them avoid design decisions that would constrain likely future changes to the system.
Appendices	These should provide detailed, specific information that is related to the application being developed; for example, hardware and database descriptions. Hardware requirements define the minimal and optimal configurations for the system. Database requirements define the logical organization of the data used by the system and the relationships between data.
Index	Several indexes to the document may be included. As well as a normal alphabetic index, there may be an index of diagrams, an index of functions, and so on.

INFO

Software Requirements Specification Template

A *software requirements specification* (SRS) is a work product that is created when a detailed description of all aspects of the software to be built must be specified before the project is to commence. It is important to note that a formal SRS is not always written. In fact, there are many instances in which effort expended on an SRS might be better spent in other software engineering activities. However, when software is to be developed by a third party, when a lack of specification would create severe business issues, or when a system is extremely complex or business critical, an SRS may be justified.

Karl Wiegers [Wie03] of Process Impact Inc. has developed a worthwhile template (available at **www.processimpact.com/process_assets/srs_template.doc**) that can serve as a guideline for those who must create a complete SRS. A topic outline follows:

Table of Contents

Revision History

1. **Introduction**
   - 1.1 Purpose
   - 1.2 Document Conventions
   - 1.3 Intended Audience and Reading Suggestions
   - 1.4 Project Scope
   - 1.5 References
2. **Overall Description**
   - 2.1 Product Perspective
   - 2.2 Product Features
   - 2.3 User Classes and Characteristics
   - 2.4 Operating Environment
   - 2.5 Design and Implementation Constraints
   - 2.6 User Documentation
   - 2.7 Assumptions and Dependencies
3. **System Features**
   - 3.1 System Feature 1
   - 3.2 System Feature 2 (and so on)
4. **External Interface Requirements**
   - 4.1 User Interfaces
   - 4.2 Hardware Interfaces
   - 4.3 Software Interfaces
   - 4.4 Communications Interfaces
5. **Other Nonfunctional Requirements**
   - 5.1 Performance Requirements
   - 5.2 Safety Requirements
   - 5.3 Security Requirements
   - 5.4 Software Quality Attributes
6. **Other Requirements**

Appendix A: Glossary
Appendix B: Analysis Models
Appendix C: Issues List

A detailed description of each SRS topic can be obtained by downloading the SRS template at the URL noted in this sidebar.

Requirements management

- Requirements management is the process of managing changing requirements during the requirements engineering Process and system management
- New requirements emerge as a system is being developed and after it has gone into use.
- You need to keep track of individual requirements and maintain links between dependent requirements so that you can assess the impact of requirements changes. You need to establish a formal process for making change proposals and linking these to system requirements.

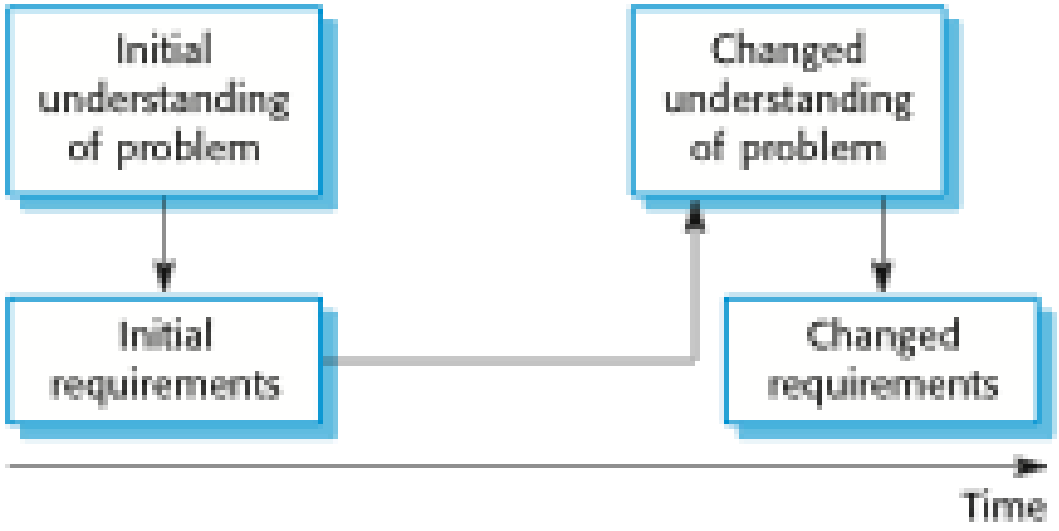
Changing requirements

- The business and technical environment of the system always changes after installation.
- The people who pay for a system and the users of that system are rarely the same people.
  - System customers impose requirements because of organizational and budgetary constraints. These may conflict with end-user requirements and, after delivery, new features may have to be added for user support if the system is to meet its goals.

Changing requirements

- Large systems usually have a diverse user community, with many users having different requirements and priorities that may be conflicting or contradictory.

Requirements evolution



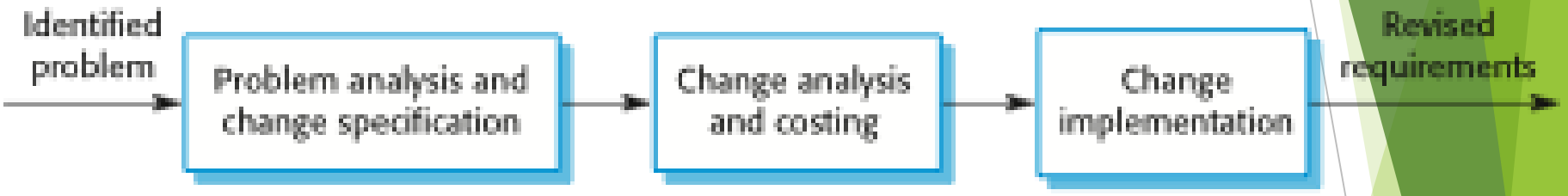
Requirements management planning

- Establishes the level of requirements management detail that is required.
- Requirements management decisions:
  - *Requirements identification* Each requirement must be uniquely identified so that it can be cross-referenced with other requirements.
  - *A change management process* This is the set of activities that assess the impact and cost of changes. I discuss this process in more detail in the following section.
  - *Traceability policies* These policies define the relationships between each requirement and between the requirements and the system design that should be recorded.
  - *Tool support* Tools that may be used range from specialist requirements management systems to spreadsheets and simple database systems.

Requirements change management

- Deciding if a requirements change should be accepted
  - *Problem analysis and change specification*
  - *Change analysis and costing*
    - The effect of the proposed change is assessed using traceability information and general knowledge of the system requirements. Once this analysis is completed, a decision is made whether or not to proceed with the requirements change.
  - Change implementation
    - The requirements document and, where necessary, the system design and implementation, are modified. Ideally, the document should be organized so that changes can be easily implemented.

Requirements change management



Traceability Matrix

- Is an Engg team that refers to documented links between Software Engg work products(eg Requirements and test cases)
- Rows of the matrix are labelled using requirement names and columns can be labelled with the name of software engg work product.
- A matrix cell is marked to indicate the presence of link between the two.

- This matrix can support a variety of engg development activities.
- They can provide continuity for developers as a project moves from one project phase to another.
- It can be used to ensure the engg work products have taken all requirements into account.

Implementation and Testing (9 hours)

Module III

Object-oriented design using the UML

- Software design and implementation is the stage in the software engineering process at which an executable software system is developed.
- Software design is a creative activity in which you identify software components and their relationships, based on a customer's requirements.
- Implementation is the process of realizing the design as a program.
- Design and implementation are closely linked, and you should normally take implementation issues into account when developing a design.

Two aims:

- 1. To show how system modeling and architectural are put into practice in developing an object-oriented software design.
- 2. To introduce important implementation issues that are not usually covered in programming books.

These include software reuse, configuration management and open-source development.

Object-oriented design using the UML

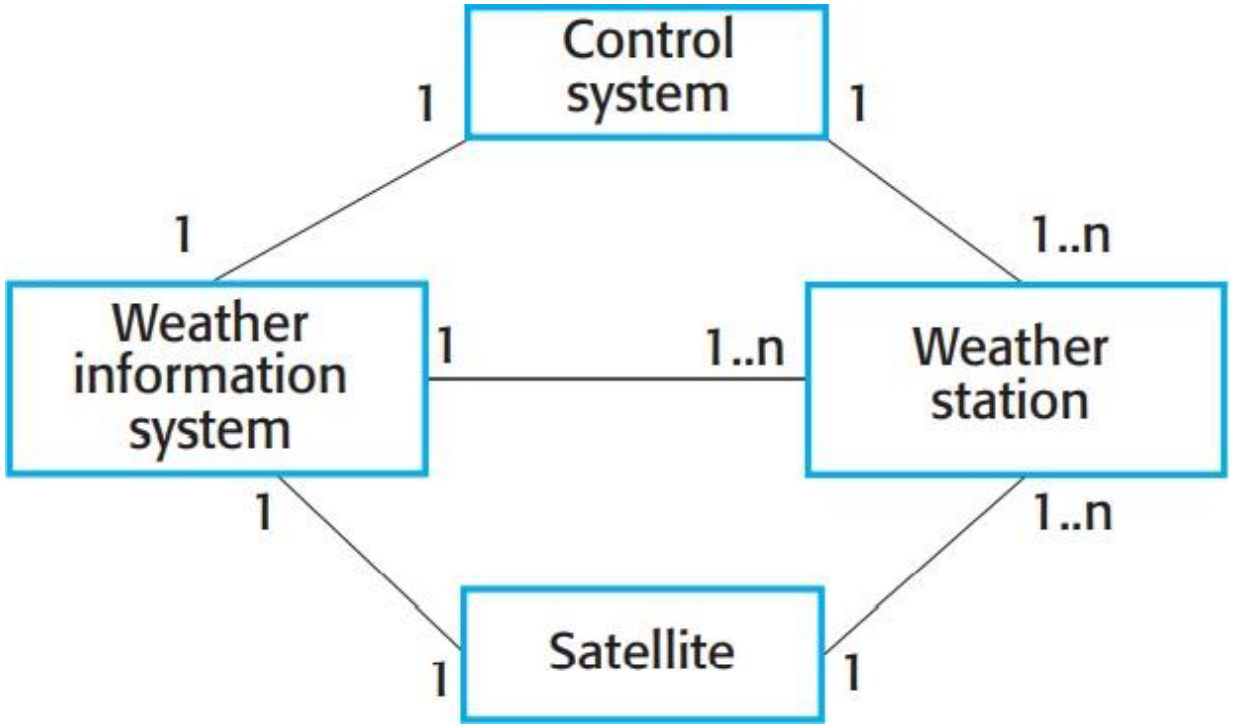
- An object-oriented system is made up of interacting objects that maintain their own local state and provide operations on that state
- Object-oriented design processes involve designing object classes and the relationships between these classes.
- Objects include both data and operations to manipulate that data.

System context and interactions

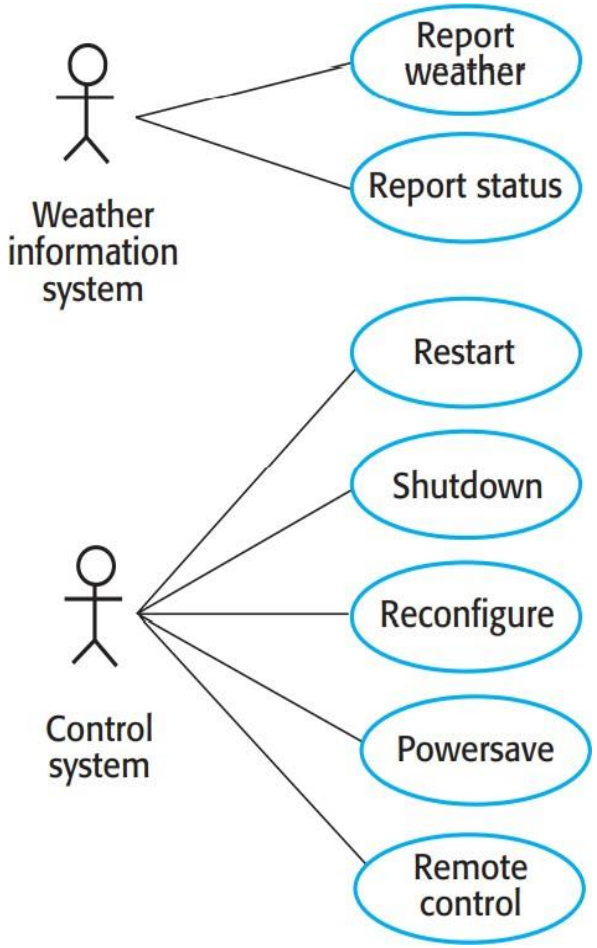
- System context models and interaction models present complementary views of the relationships between a system and its environment:
- A system context model is a structural model that demonstrates the other systems in the environment of the system being developed.
- An interaction model is a dynamic model that shows how the system interacts with its environment as it is used.

- The context model of a system may be represented using associations.
- Associations simply show that there are some relationships between the entities involved in the association.
- You can document the environment of the system using a simple block diagram, showing the entities in the system and their associations.

System context for the weather station



Weather station use cases



Usecase description-Report weather

System	Weather station
Use case	Report weather
Actors	Weather information system, Weather station
Data	The weather station sends a summary of the weather data that has been collected from the instruments in the collection period to the weather information system. The data sent are the maximum, minimum, and average ground and air temperatures; the maximum, minimum, and average air pressures; the maximum, minimum and average wind speeds; the total rainfall; and the wind direction as sampled at 5-minute intervals.
Stimulus	The weather information system establishes a satellite communication link with the weather station and requests transmission of the data.
Response	The summarized data is sent to the weather information system.
Comments	Weather stations are usually asked to report once per hour, but this frequency may differ from one station to another and may be modified in future.

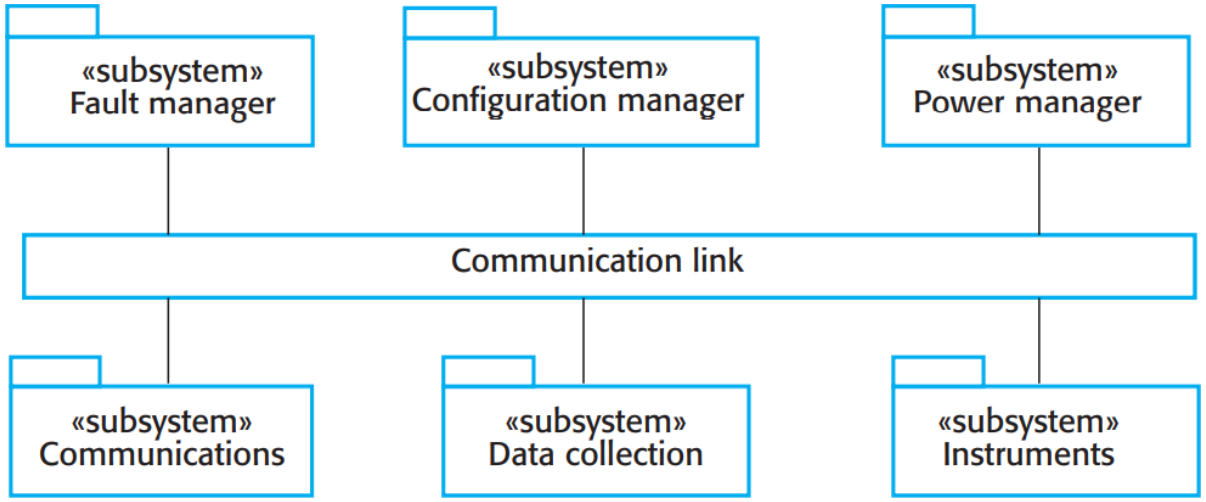
Architectural design

- Once the interactions between the software system and the system's environment have been defined,
- Identify the major components that make up the system and their interactions. Then design the system organization using an architectural pattern such as a layered or client–server model

High-level architecture of weather station

The weather station is composed of independent subsystems that communicate by broadcasting messages on a common infrastructure(Communication link here)

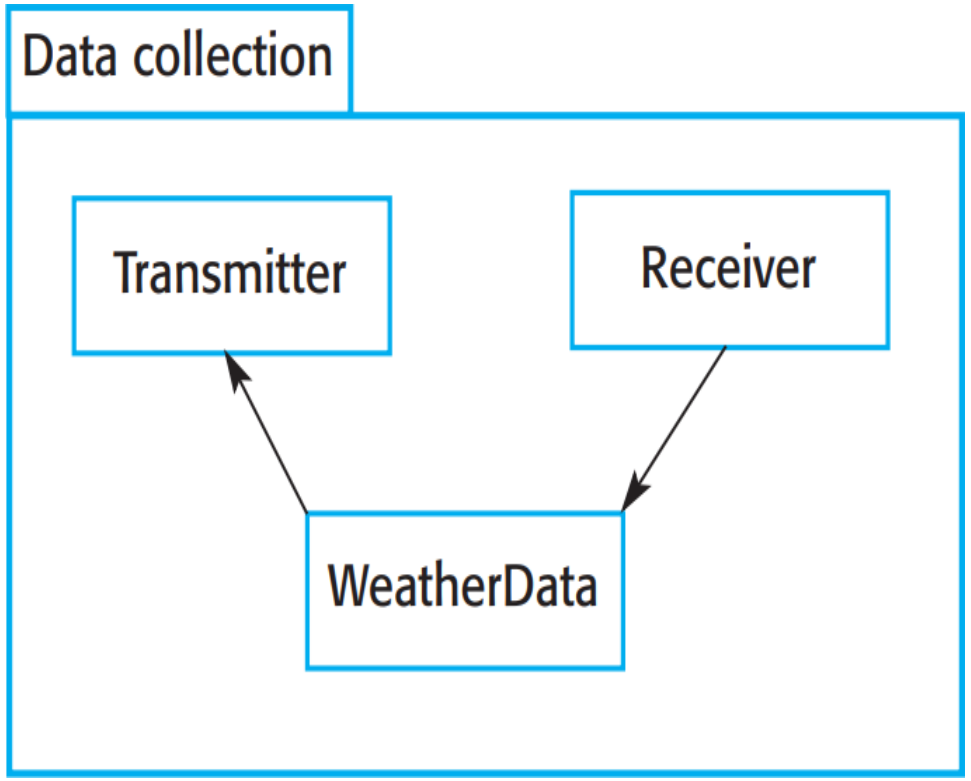
Each subsystem listens for messages on that infrastructure and picks up the messages that are intended for them. This “listener model” is a commonly used architectural style for distributed systems.



- **Benefit** :it is easy to support different configurations of subsystems because the sender of a message does not need to address the message to a particular subsystem

Architecture of data collection system

- The Transmitter and Receiver objects are concerned with managing communications
- The WeatherData object encapsulates the information that is collected from the instruments and transmitted to the weather information system.
- This arrangement follows the producer–consumer pattern



Object class identification

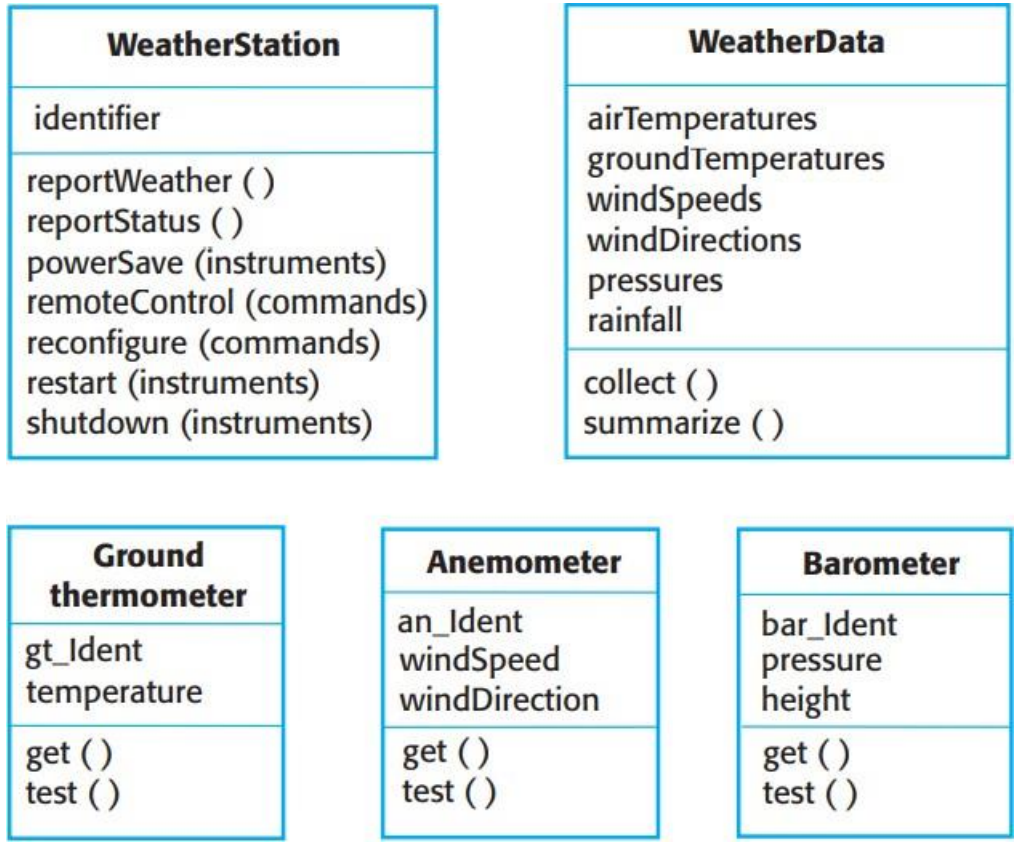
Various ways of identifying object classes in object-oriented systems :

1. Use a grammatical analysis of a natural language description of the system to be constructed. Objects and attributes are nouns; operations or services are verbs.

2. Use tangible entities (things) in the application domain such as aircraft, roles such as manager, events such as request, interactions such as meetings etc.

3. Use a scenario-based analysis where various scenarios of system use are identified and analyzed in turn.

Weather station objects



Design models

- Two kinds of design model:

    1. Structural models, which describe the static structure of the system using object classes and their relationships.

    2. Dynamic models, which describe the dynamic structure of the system and show the expected runtime interactions between the system objects.

Three UML model types:

1. Subsystem models, which show logical groupings of objects into coherent subsystems. These are represented using a form of class diagram with each subsystem shown as a package with enclosed objects. Subsystem models are structural models.

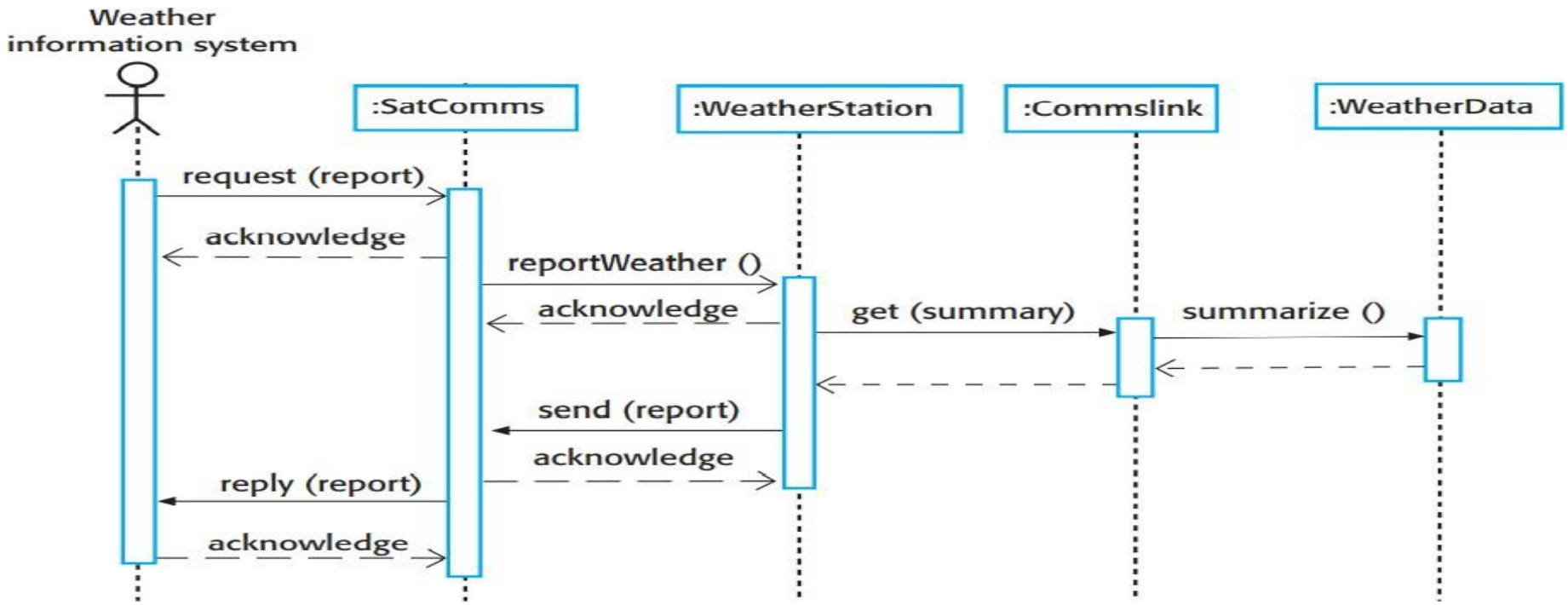
2. Sequence models, which show the sequence of object interactions. These are represented using a UML sequence or a collaboration diagram. Sequence models are dynamic models.

3. State machine models, which show how objects change their state in response to events. These are represented in the UML using state diagrams. State machine models are dynamic models.

- A subsystem model is a useful static model that shows how a design is organized into logically related groups of objects.

- Sequence models are dynamic models that describe, for each mode of interaction, the sequence of object interactions that take place.

Sequence diagram describing data collection



1. The **SatComms** object receives a request from the weather information system to collect a weather report from a weather station. It acknowledges receipt of this request. The stick arrowhead on the sent message indicates that the external system does not wait for a reply but can carry on with other processing.

2. **SatComms** sends a message to WeatherStation, via a satellite link, to create a summary of the collected weather data. Again, the stick arrowhead indicates that SatComms does not suspend itself waiting for a reply.

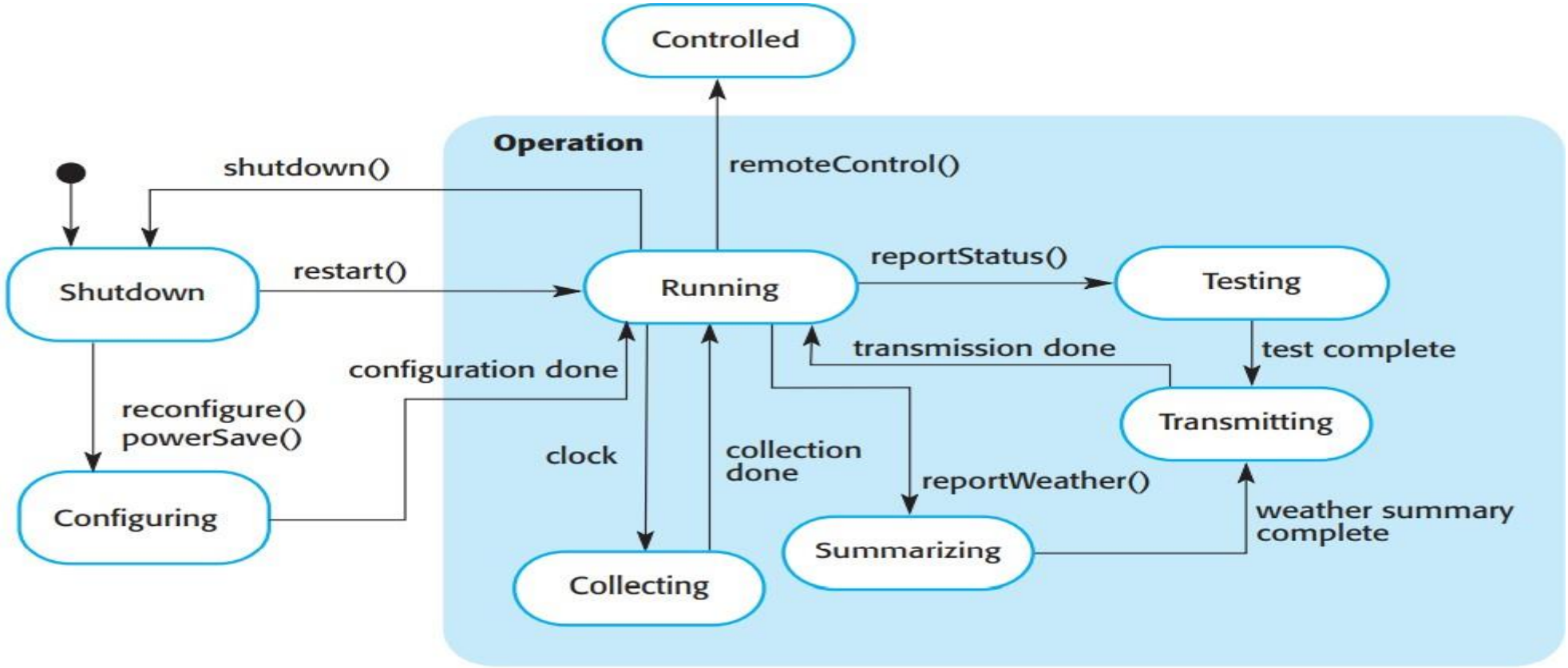
3. **WeatherStation** sends a message to a Commslink object to summarize the weather data. In this case, the squared-off style of arrowhead indicates that the instance of the WeatherStation object class waits for a reply.

4. **Commslink** calls the summarize method in the object WeatherData and waits for a reply

5. The weather data summary is computed and returned to WeatherStation via the **Commslink** object.

6. **WeatherStation** then calls the SatComms object to transmit the summarized data to the weather information system, through the satellite communications system

Weather station state diagram



1. If the system state is Shutdown, then it can respond to a restart(), a reconfigure() or a powerSave() message. The unlabeled arrow with the black blob indicates that the Shutdown state is the initial state. A restart() message causes a transition to normal operation. Both the powerSave() and reconfigure() messages cause a transition to a state in which the system reconfigures itself. The state diagram shows that reconfiguration is allowed only if the system has been shut down.

2. In the Running state, the system expects further messages. If a shutdown() message is received, the object returns to the shutdown state.

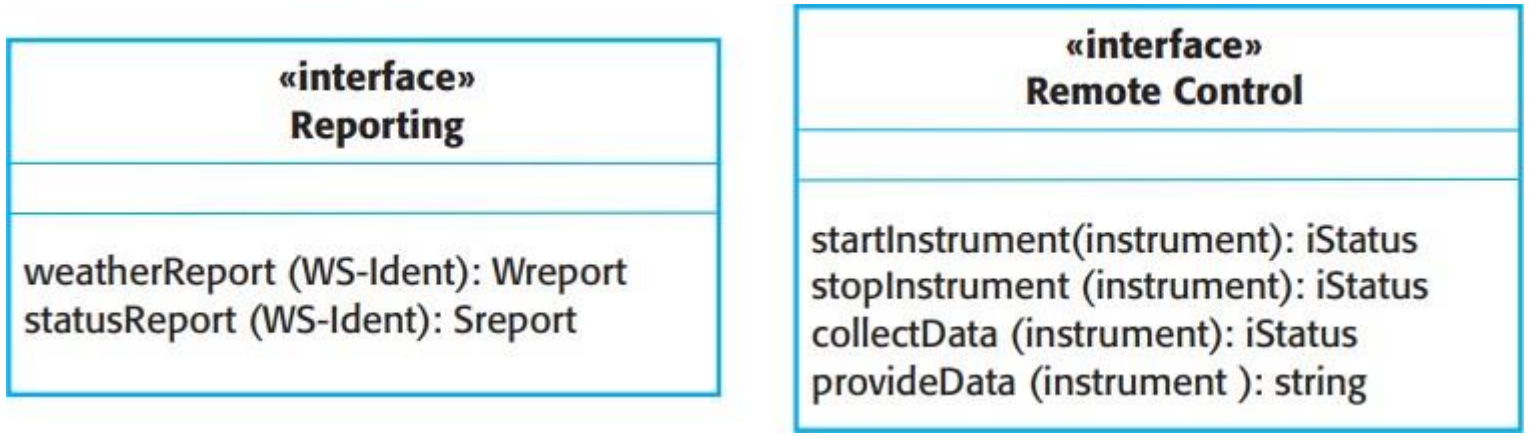
3. If a reportWeather() message is received, the system moves to the Summarizing state. When the summary is complete, the system moves to a Transmitting state where the information is transmitted to the remote system. It then returns to the Running state.

4. If a signal from the clock is received, the system moves to the Collecting state, where it collects data from the instruments. Each instrument is instructed in turn to collect its data from the associated sensors.

5. If a remoteControl() message is received, the system moves to a controlled state in which it responds to a different set of messages from the remote control room. These are not shown on this diagram.

Interface specification

Weather station interfaces



Design Patterns

- The pattern is a description of the problem and the essence of its solution, so that the solution may be reused in different settings. The pattern is not a detailed specification.
- Patterns have made a huge impact on object-oriented software design
- They have become a vocabulary for talking about a design
- Patterns are a way of reusing the knowledge and experience of other designers. Design patterns are usually associated with object-oriented design.
- The general principle of encapsulating experience in a pattern is one that is equally applicable to any kind of software design.

Four essential elements of design patterns

1. A name that is a meaningful reference to the pattern.

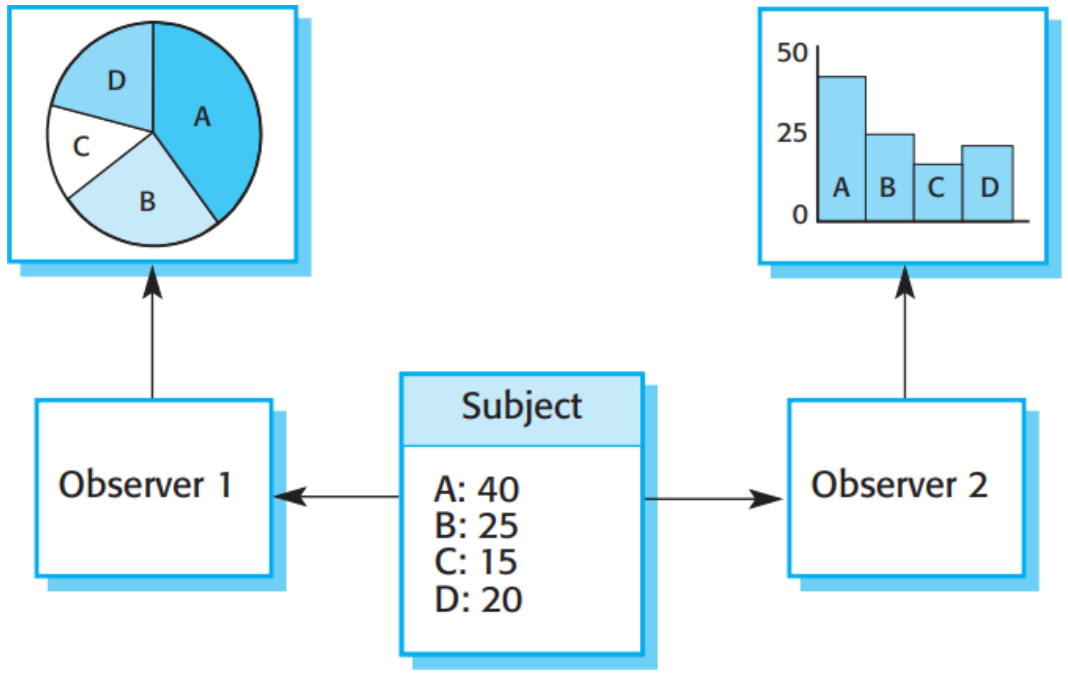
2. A description of the problem area that explains when the pattern may be applied.

3. A solution description of the parts of the design solution, their relationships and their responsibilities.It is a template for a design solution that can be instantiated in different ways. This is often expressed graphically and shows the relationships between the objects and object classes in the solution.

4. A statement of the consequences—the results and trade-offs—of applying the pattern. This can help designers understand whether or not a pattern can be used in a particular situation.

Two different graphical presentations of the same dataset:

- Graphical representations are normally used to illustrate the object classes in patterns and their relationships.
- These supplement the pattern description and add detail to the solution description.



IMPLEMENTATION ISSUES

- Software engineering includes all of the activities involved in software development from the initial requirements of the system through to maintenance and management of the deployed system.
- A critical stage of this process is, of course, system implementation, where you create an executable version of the software.
- Implementation may involve developing programs in high- or low-level programming languages or tailoring and adapting generic, off-the-shelf systems to meet the specific requirements of an organization.

Aspects of implementation

1. Reuse:
   - Most modern software is constructed by reusing existing components or systems. When you are developing software, you should make as much use as possible of existing code.
2. Configuration management:
   - During the development process, many different versions of each software component are created. If you don’t keep track of these versions in a configuration management system, you are liable to include the wrong versions of these components in your system.
3. Host-target development
   - Production software does not usually execute on the same computer as the software development environment. Rather, you develop it on one computer (the host system) and execute it on a separate computer (the target system)

1.Reuse

Different Levels of reuse:

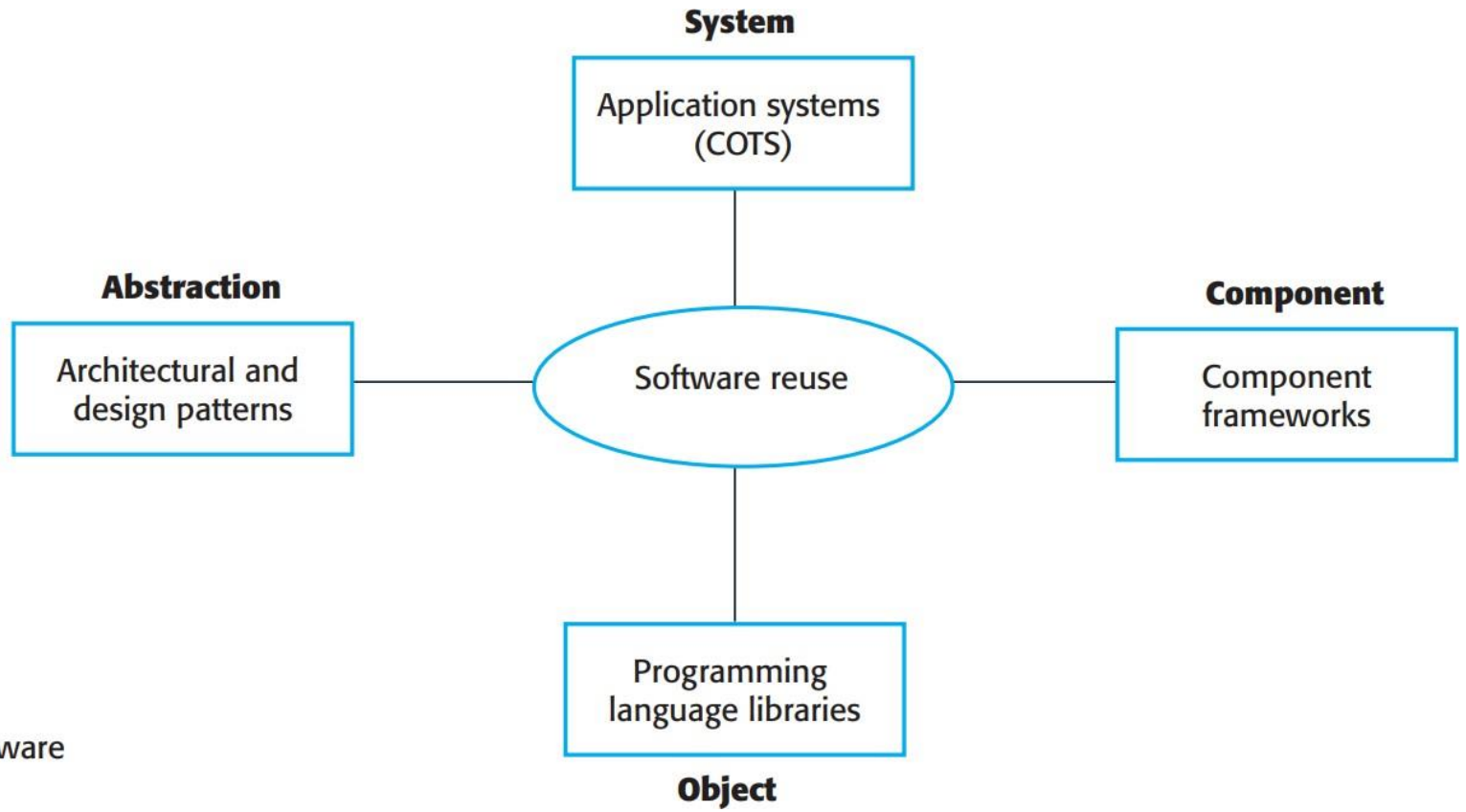
1. **The abstraction level**: At this level, you don’t reuse software directly but rather use knowledge of successful abstractions in the design of your software.eg. Design patterns and architectural patterns

2. **The object level:** At this level, you directly reuse objects from a library rather than writing the code yourself. To implement this type of reuse, you have to find appropriate libraries and discover if the objects and methods offer the functionality that you need. For example, JavaMail library.

3. **The component level:** Components are collections of objects and object classes that operate together to provide related functions and services. An example of component-level reuse is where you build your user interface using a framework.

4. **The system level:** At this level, you reuse entire application systems. This function usually involves some kind of configuration of these systems. This may be done by adding and modifying code or by using the system’s own configuration interface

Software reuse



oftware

Costs associated with reuse:

1. The costs of the time spent in looking for software to reuse and assessing whether or not it meets your needs.
2. Where applicable, the costs of buying the reusable software. For large off-theshelf systems, these costs can be very high.
3. The costs of adapting and configuring the reusable software components or systems to reflect the requirements of the system that you are developing.
4. The costs of integrating reusable software elements with each other (if you are using software from different sources) and with the new code that you have developed.

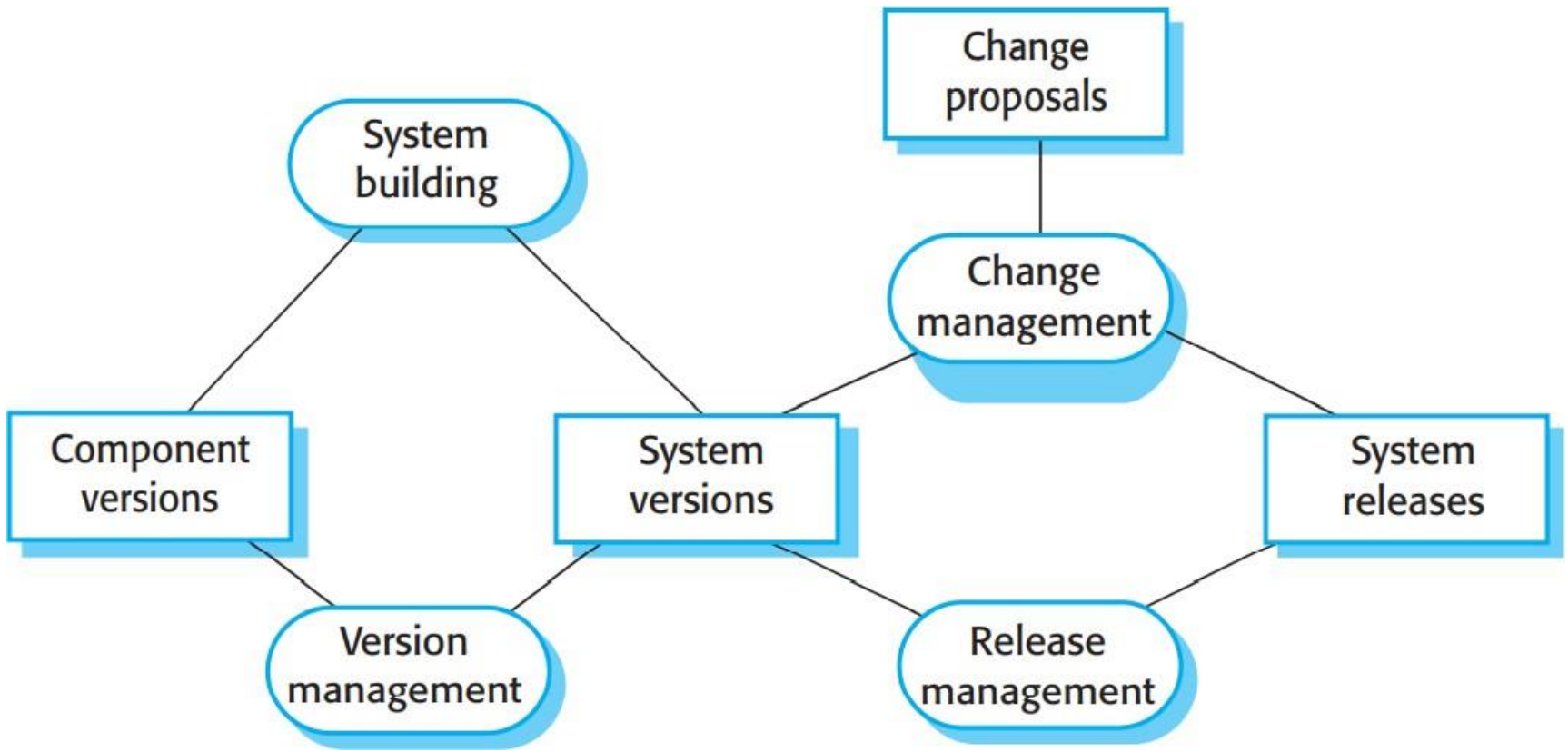
- By reusing existing software, you can develop new systems more quickly, with fewer development risks and at lower cost. As the reused software has been tested in other applications, it should be more reliable than new software
- How to reuse existing knowledge and software should be the first thing you should think about when starting a software development project.

2.Configuration management

- Configuration management is the name given to the general process of managing a changing software system. The aim of configuration management is to support the system integration process so that all developers can access the project code and documents in a controlled way, find out what changes have been made, and compile and link components to create a system.
- Software configuration management tools support each of the above activities. These tools are usually installed in an integrated development environment, such as Eclipse.

Four fundamental configuration management activities:

- 1. Version management, where support is provided to keep track of the different versions of software components. Version management systems include facilities to coordinate development by several programmers. They stop one developer from overwriting code that has been submitted to the system by someone else.
- 2. System integration, where support is provided to help developers define what versions of components are used to create each version of a system. This description is then used to build a system automatically by compiling and linking the required components.
- 3. Problem tracking, where support is provided to allow users to report bugs and other problems, and to allow all developers to see who is working on these problems and when they are fixed.
- 4. Release management, where new versions of a software system are released to customers. Release management is concerned with planning the functionality of new releases and organizing the software for distribution.


Change proposals
System building
Change management
Component versions
System versions
System releases
Version management
Release management

3. Host-target development

- Most professional software development is based on a host-target model .Software is developed on one computer (the host) but runs on a separate machine (the target).
- More generally, we can talk about a development platform (host) and an execution platform (target).
- A platform is more than just hardware. It includes the installed operating system plus other supporting software such as a database management system or, for development platforms, an interactive development environment.
- Simulators are often used when developing embedded systems.
- Simulators speed up the development process for embedded systems

A software development platform should provide a range of tools to support software engineering processes.

These may include:

1. An integrated compiler and syntax-directed editing system that allows you to create, edit, and compile code.

2. A language debugging system.

3. Graphical editing tools, such as tools to edit UML models.

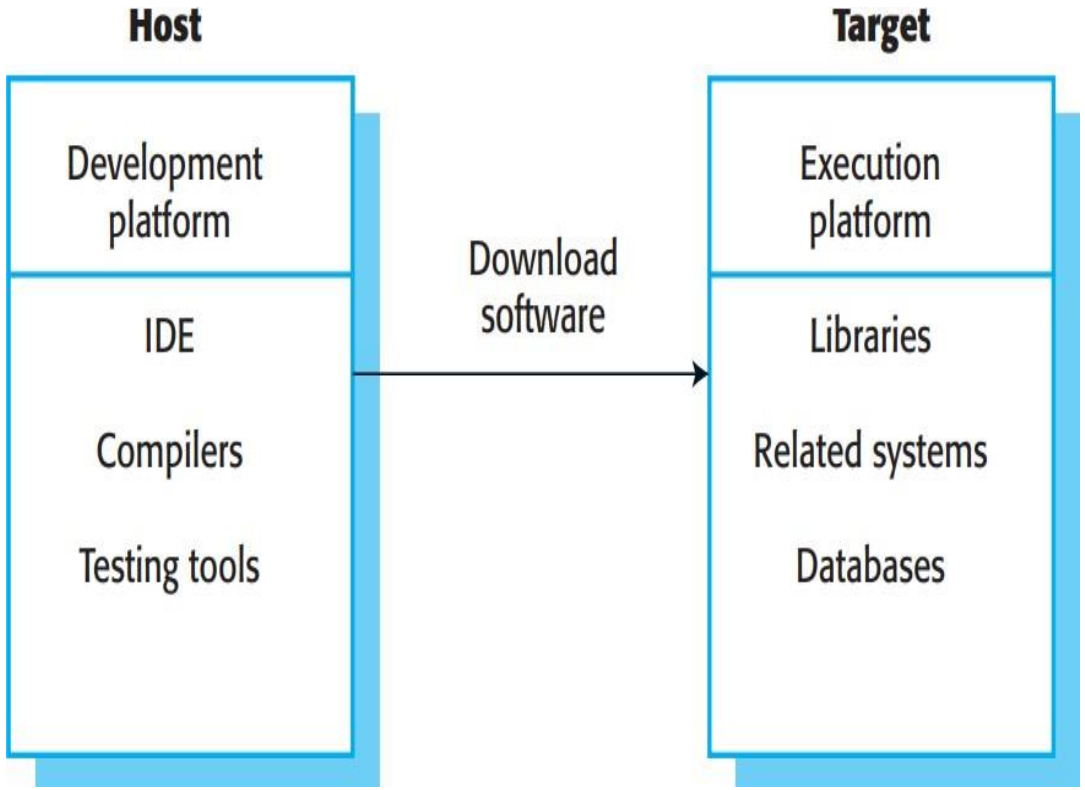
4. Testing tools, such as JUnit, that can automatically run a set of tests on a new version of a program.

5. Tools to support refactoring and program visualization.

6. Configuration management tools to manage source code versions and to integrate and build systems.

A general-purpose IDE is a framework for hosting software tools that provides data management facilities for the software being developed and integration mechanisms that allow tools to work together.

The best-known general-purpose IDE is the Eclipse environment (http://www.eclipse.org).



Issues

1. The hardware and software requirements of a component

   If a component is designed for a specific hardware architecture, or relies on some other software system, it must obviously be deployed on a platform that provides the required hardware and software support.

2. The availability requirements of the system

   High-availability systems may require components to be deployed on more than one platform. This means that, in the event of platform failure, an alternative implementation of the component is available.

3. Component communications

   If there is a lot of intercomponent communication, it is usually best to deploy them on the same platform or on platforms that are physically close to one another. This reduces communications latency—the delay between the time that a message is sent by one component and received by another.

Open-source licensing

- A fundamental principle of open-source development is that source code should be freely available.
- Legally, the developer of the code owns the code. They can place restrictions on how it is used by including legally binding conditions in an open-source software license
- Licensing issues are important because if you use open-source software as part of a software product, then you may be obliged by the terms of the license to make your own product open source..
- The open-source approach is one of several business models for software.

Most open-source licenses are variants of one of three general models:

1. The GNU General Public License (GPL).

This is a so-called reciprocal license that simplistically means that if you use open-source software that is licensed under the GPL license, then you must make that software open source.

2. The GNU Lesser General Public License (LGPL).

This is a variant of the GPL license where you can write components that link to open-source code without having to publish the source of these components. However, if you change the licensed component, then you must publish this as open source.

3. The Berkley Standard Distribution (BSD) License.

This is a nonreciprocal license, which means you are not obliged to re-publish any changes or modifications made to open-source code. You can include the code in proprietary systems that are sold. If you use open-source components, you must acknowledge the original creator of the code. eg.The MIT license

Companies managing projects that use open source should:

1. Establish a system for maintaining information about open-source components that are downloaded and used. You have to keep a copy of the license for each component that was valid at the time the component was used. Licenses may change, so you need to know the conditions that you have agreed to.

2. Be aware of the different types of licenses and understand how a component is licensed before it is used. You may decide to use a component in one system but not in another because you plan to use these systems in different ways.

3. Be aware of evolution pathways for components. You need to know a bit about the open-source project where components are developed to understand how they might change in future.

4. Educate people about open source. It’s not enough to have procedures in place to ensure compliance with license conditions. You also need to educate developers about open source and open-source licensing.

5. Have auditing systems in place. Developers, under tight deadlines, might be tempted to break the terms of a license. If possible, you should have software in place to detect and stop this.

6. Participate in the open-source community. If you rely on open-source products, you should participate in the community and help support their development.

MODULE - 4

SOFTWARE PROJECT MANAGEMENT

- Software project management is an essential part of software engineering.
- The success criteria **for project management** obviously vary from project to project, but, for most projects, **important goals are:**
    1. to deliver the software to the customer at the agreed time;
    2. to keep overall costs within budget
    3. to deliver software that meets the customer's expectations;
    4. to maintain a coherent and well-functioning development team.
- Software engineering is different from other types of engineering in a number of ways:
    1. The product is intangible.
    2. Large software projects are often "one-off" projects.
    3. Software processes are variable and organization-specific.

- It is impossible to write a standard job description for a software project manager.
- Some of the most **important factors that affect how software projects are managed are:**
    1. Company size
    2. Software customers
    3. Software size
    4. Software type
    5. Organizational culture
    6. Software development processes

- The **fundamental project management activities** that are common to all organizations:
    1. **Project planning** → Project managers are responsible for planning, estimating, and scheduling project development and assigning people to tasks.
    2. **Risk management** → Project managers have to assess the risks that may affect a project, monitor these risks, and take action when problems arise.
    3. **People management** → Project managers are responsible for managing a team of people. They have to choose people for their team and establish ways of working that lead to effective team performance.
    4. **Reporting** → 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.
    5. **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. It usually includes cost and schedule estimates and justifies why the project contract should be awarded to a particular organization or team.

RISK MANAGEMENT

- Risk management is one of the most important jobs for a project manager.
- Risk management involves anticipating risks that might affect the project schedule or the quality of the software being developed, and then taking action to avoid these risks.
- Risks can be categorized according to type of risk (technical, organizational, etc.)

- **Classification of risks according to what these risks affect:**
  1. **Project risks** → affect the project schedule or resources. An example of a project risk is the loss of an experienced system architect.
  2. **Product risks** → affect the quality or performance of the software being developed. An example of a product risk is the failure of a purchased component to perform as expected.
  3. **Business risks** → affect the organization developing or procuring the software. For example, a competitor introducing a new product is a business risk.
- For large projects, you should record the results of the risk analysis in a risk register along with a consequence analysis. This sets out the consequences of the risk for the project, product, and business.

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 company 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.
Software tool underperformance	Product	Software tools that 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.

Fig: Examples of common project, product, and business risks

- Effective risk management makes it easier to cope with problems and to ensure that these do not lead to unacceptable budget or schedule slippage.
- For small projects, formal risk recording may not be required, but the project manager should be aware of them.
- The specific risks that may affect a project depend on the project and the organizational environment in which the software is being developed.
- Software risk management is important because of the inherent uncertainties in software development.

- An outline of the process of risk management is presented in Figure. It involves several **stages:**
  1. **Risk identification** → You should identify possible project, product, and business risks.
  2. **Risk analysis** → You should assess the likelihood and consequences of these risks.
  3. **Risk planning** → You should make plans to address the risk, either by avoiding it or by minimizing its effects on the project.
  4. **Risk monitoring** → You should regularly assess the risk and your plans for risk mitigation and revise these plans when you learn more about the risk.
- The risk management process is an iterative process that continues throughout a project.

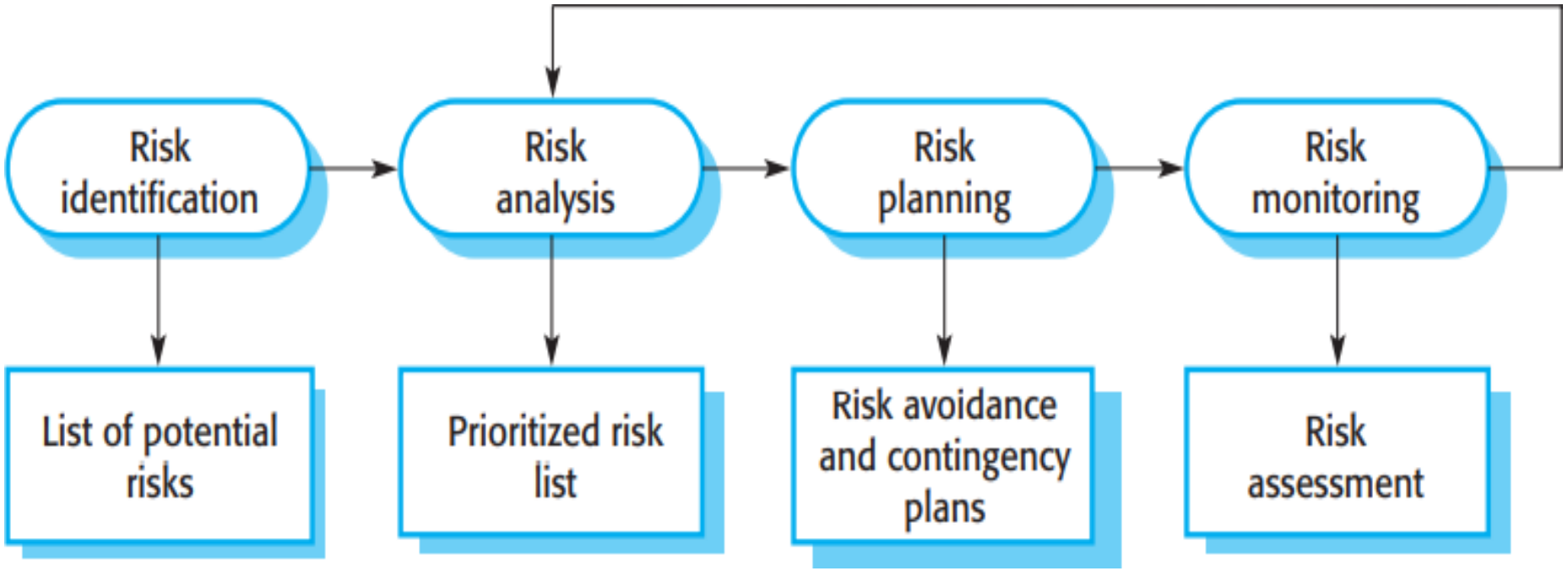


Fig: The Risk Management Process

Risk Identification:

- Risk identification is the first stage of the risk management process.
- It is concerned with identifying the risks that could pose a major threat to the software engineering process, the software being developed, or the development organization.
- Risk identification may be a team process in which a team gets together to brainstorm possible risks.
- As a starting point for risk identification, a checklist of different types of risk may be used.

- **6 types of risk** may be included in a risk checklist:
  1. **Estimation risks** → arise from the management estimates of the resources required to build the system.
  2. **Organizational risks** → arise from the organizational environment where the software is being developed.
  3. **People risks** → are associated with the people in the development team.
  4. **Requirements risks** → come from changes to the customer requirements and the process of managing the requirements change.
  5. **Technology risks** → come from the software or hardware technologies that are used to develop the system.
  6. **Tools risks** → come from the software tools and other support software used to develop the system.

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

Figure 22.3 Examples of different types of risk

Risk Analysis:

- During the risk analysis process, you have to consider each identified risk and make a judgment about the probability and seriousness of that risk.
- It is not possible to make precise, numeric assessment of the probability and seriousness of each risk.
- You should assign the risk to one of a number of bands:
    1. The probability of the risk might be assessed as insignificant, low, moderate, high, or very high.
    2. The effects of the risk might be assessed as catastrophic (threaten the survival of the project), serious (would cause major delays), tolerable (delays are within allowed contingency), or insignificant.
- You may then tabulate the results of this analysis process using a table ordered according to the seriousness of the risk.

Risk	Probability	Effects
Organizational financial problems force reductions in the project budget (5).	Low	Catastrophic
It is impossible to recruit staff with the skills required (6).	High	Catastrophic
Key staff are ill at critical times in the project (7).	Moderate	Serious
Faults in reusable software components have to be repaired before these components are reused (12).	Moderate	Serious
Changes to requirements that require major design rework are proposed (9).	Moderate	Serious
The organization is restructured so that different managements are responsible for the project (4).	High	Serious
The database used in the system cannot process as many transactions per second as expected (11).	Moderate	Serious
The time required to develop the software is underestimated (1).	High	Serious
Software tools cannot be integrated (14).	High	Tolerable
Customers fail to understand the impact of requirements changes (10).	Moderate	Tolerable
Required training for staff is not available (8).	Moderate	Tolerable
The rate of defect repair is underestimated (2).	Moderate	Tolerable
The size of the software is underestimated (3).	High	Tolerable
Code generated by code generation tools is inefficient (13).	Moderate	Insignificant

Figure 22.4 Risk types and examples

- Both the probability and the assessment of the effects of a risk may change as more information about the risk becomes available and as risk management plans are implemented. You should therefore update this table during each iteration of the risk management process.
- Once the risks have been analyzed and ranked, you should assess which of these risks are most significant.
- In general, catastrophic risks should always be considered, as should all serious risks that have more than a moderate probability of occurrence.

Risk Planning:

- The risk planning process develops strategies to manage the key risks that threaten the project.
- For each risk, you have to think of actions that you might take to minimize the disruption to the project if the problem identified in the risk occurs.
- You should also think about the information that you need to collect while monitoring the project so that emerging problems can be detected before they become serious.

- In risk planning, you have to ask “what-if” questions that consider both individual risks, combinations of risks, and external factors that affect these risks. For example, questions that you might ask are:
  1. What if several engineers are ill at the same time?
  2. What if an economic downturn leads to budget cuts of 20% for the project?
  3. What if the performance of open-source software is inadequate and the only expert on that open-source software leaves?
  4. What if the company that supplies and maintains software components goes out of business?
  5. What if the customer fails to deliver the revised requirements as predicted?
- Based on the answers to these “what-if” questions, you may devise strategies for managing the risks.

- The **possible risk management strategies** fall into 3 categories:
  1. **Avoidance strategies** → Following these strategies means that the probability that the risk will arise is reduced. An example of a risk avoidance strategy is the strategy for dealing with defective components.
  2. **Minimization strategies** → Following these strategies means that the impact of the risk is reduced. An example of a risk minimization strategy is the strategy for staff illness.
  3. **Contingency plans** → Following these strategies means that you are prepared for the worst and have a strategy in place to deal with it. An example of a contingency strategy is the strategy for organizational financial problems.
- The strategies used in critical systems ensure reliability, security, and safety, where you must avoid, tolerate, or recover from failures.

- It is best to use a strategy that avoids the risk.
- If this is not possible, you should use a strategy that reduces the chances that the risk will have serious effects.
- Finally, you should have strategies in place to cope with the risk if it arises. These should reduce the overall impact of a risk on the project or product.

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.
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 automated code generation.

Figure 22.5 Strategies to help manage risk

Risk Monitoring:

- Risk monitoring is the process of checking that your assumptions about the product, process, and business risks have not changed.
- You should regularly assess each of the identified risks to decide whether or not that risk is becoming more or less probable.
- You should also think about whether or not the effects of the risk have changed.
- To do this, you have to look at other factors, such as the number of requirements change requests, which give you clues about the risk probability and its effects. These factors are dependent on the types of risk.

Risk type	Potential indicators
Estimation	Failure to meet agreed schedule; failure to clear reported defects.
Organizational	Organizational gossip; lack of action by senior management.
People	Poor staff morale; poor relationships among team members; high staff turnover.
Requirements	Many requirements change requests; customer complaints.
Technology	Late delivery of hardware or support software; many reported technology problems.
Tools	Reluctance by team members to use tools; complaints about software tools; requests for faster computers/more memory, and so on.

Figure 22.6 Risk indicators

- You should monitor risks regularly at all stages in a project.
- At every management review, you should consider and discuss each of the key risks separately.
- You should decide if the risk is more or less likely to arise and if the seriousness and consequences of the risk have changed.

MANAGING PEOPLE

- The people working in a software organization are its greatest assets.
- It is expensive to recruit and retain good people.
- Software managers have to ensure that the engineers working on a project are as productive as possible.
- It is important that software project managers understand the technical issues that influence the work of software development.
- Software engineers often have strong technical skills but may lack the softer skills that enable them to motivate and lead a project development team.
- As a project manager, you should be aware of the potential problems of people management and should try to develop people management skills.

- **4 critical factors that influence the relationship between a manager and the people** that he or she manages:
    1. **Consistency** → All the people in a project team should be treated in a comparable way. No one expects all rewards to be identical, but people should not feel that their contribution to the organization is undervalued.
    2. **Respect** → Different people have different skills, and managers should respect these differences.
    3. **Inclusion** → People contribute effectively when they feel that others listen to them and take account of their proposals. It is important to develop a working environment where all views, even those of the least experienced staff, are considered.
    4. **Honesty** → As a manager, you should always be honest about what is going well and what is going badly in the team. You should also be honest about your level of technical knowledge and be willing to defer to staff with more knowledge when necessary.

Motivating People:

- As a project manager, you need to motivate the people who work with you so that they will contribute to the best of their abilities.
- In practice, **motivation** means organizing work and its environment to encourage people to work as effectively as possible.
- To provide this encouragement, you should understand a little about what motivates people.
- People are motivated by satisfying their needs. These needs are arranged in a series of levels, as shown in Figure.

Figure 22.7 Human needs hierarchy

- The lower levels of this hierarchy represent fundamental needs for food, sleep, and so on, and the need to feel secure in an environment.
- **Social need** is concerned with the need to feel part of a social grouping.
- **Esteem need** represents the need to feel respected by others, and **self-realization need** is concerned with personal development.
- People need to satisfy lower-level needs such as hunger before the more abstract, higher-level needs.
- People working in software development organizations are not usually hungry, thirsty, or physically threatened by their environment. Therefore, making sure that peoples' social, esteem, and self-realization needs are satisfied is most important from a management point of view.

1. **To satisfy social needs**, you need to give people time to meet their co-workers and provide places for them to meet. This is relatively easy when all of the members of a development team work in the same place. Social networking systems and teleconferencing can be used for remote communications.
2. **To satisfy esteem needs**, you need to show people that they are valued by the organization. Public recognition of achievements is a simple and effective way of doing this.
3. Finally, **to satisfy self-realization needs**, you need to give people responsibility for their work, assign them demanding (but not impossible) tasks, and provide opportunities for training and development where people can enhance their skills. Training is an important motivating influence as people like to gain new knowledge and learn new skills.

- Maslow’s model of motivation takes an exclusively personal viewpoint on motivation.
- It does not take adequate account of the fact that people feel themselves to be part of an organization, a professional group, and one or more cultures.
- Being a member of a cohesive group is highly motivating for most people.
- Therefore, as a manager, you also have to think about how a group as a whole can be motivated.

Case study: Motivation

Alice is a software project manager working in a company that develops alarm systems. This company wishes to enter the growing market of assistive technology to help elderly and disabled people live independently. Alice has been asked to lead a team of six developers that can develop new products based on the company's alarm technology.

Alice's assistive technology project starts well. Good working relationships develop within the team, and creative new ideas are developed. The team decides to develop a system that a user can initiate and control the alarm system from a cell phone or tablet computer. However, some months into the project, Alice notices that Dorothy, a hardware expert, starts coming into work late, that the quality of her work is deteriorating, and, increasingly, that she does not appear to be communicating with other members of the team.

Alice talks about the problem informally with other team members to try to find out if Dorothy's personal circumstances have changed and if this might be affecting her work. They don't know of anything, so Alice decides to talk with Dorothy to try to understand the problem.

After some initial denials of any problem, Dorothy admits that she has lost interest in the job. She expected that she would be able to develop and use her hardware interfacing skills. However, because of the product direction that has been chosen, she has little opportunity to use these skills. Basically, she is working as a C programmer on the alarm system software.

While she admits that the work is challenging, she is concerned that she is not developing her interfacing skills. She is worried that finding a job that involves hardware interfacing will be difficult after this project. Because she does not want to upset the team by revealing that she is thinking about the next project, she has decided that it is best to minimize conversation with them.

Figure 22.8 Individual motivation

- Psychological personality type also influences motivation.
- Bass and Dunteman (Bass and Dunteman 1963) identified **3 classifications for professional workers:**
  1. **Task-oriented people** → who are motivated by the work they do. In software engineering, these are people who are motivated by the intellectual challenge of software development.
  2. **Self-oriented people** → who are principally motivated by personal success and recognition. They are interested in software development as a means of achieving their own goals. They often have longer-term goals and they wish to be successful in their work to help realize these goals.
  3. **Interaction-oriented people** → who are motivated by the presence and actions of co-workers.

- Research has shown that interaction-oriented personalities usually like to work as part of a group, whereas task-oriented and self-oriented people usually prefer to act as individuals.
- **People Capability Maturity Model (P-CMM)** → is a framework for assessing how well organizations manage the development of their staff. It highlights best practice in people management and provides a basis for organizations to improve their people management processes. It is best suited to large rather than small, informal companies.

TEAMWORK

- As it is impossible for everyone in a large group to work together on a single problem, large teams are usually split into a number of smaller groups.
- Each group is responsible for developing part of the overall system.
- The best size for a software engineering group is 4 to 6 members, and they should never have more than 12 members.
- When groups are small, communication problems are reduced.

- Putting together a group that has the right balance of technical skills, experience, and personalities is a critical management task.
- A good group is cohesive and thinks of itself as a strong, single unit.
- The people involved are motivated by the success of the group as well as by their own personal goals.
- In a **cohesive group**, members think of the group as more important than the individuals who are group members.
  - They are loyal to the group.
  - They identify with group goals and other group members.
  - They attempt to protect the group, as an entity, from outside interference. This makes the group robust and able to cope with problems and unexpected situations.

- The **benefits of creating a cohesive group** are:
    1. The group can establish its own quality standards.
    2. Individuals learn from and support each other.
    3. Knowledge is shared.
    4. Refactoring and continual improvement is encouraged.
- Good project managers should always try to encourage group cohesiveness.
- They may try to establish a sense of group identity by naming the group and establishing a group identity and territory.
- One of the most effective ways of promoting cohesion is **to be inclusive** i.e., you should treat group members as responsible and trustworthy, and make information freely available.

- An effective way of making people feel valued and part of a group is to make sure that they know what is going on.

Case study: Team spirit

Alice, an experienced project manager, understands the importance of creating a cohesive group. As her company is developing a new product, she takes the opportunity to involve all group members in the product specification and design by getting them to discuss possible technology with elderly members of their families. She encourages them to bring these family members to meet other members of the development group.

Alice also arranges monthly lunches for everyone in the group. These lunches are an opportunity for all team members to meet informally, talk around issues of concern, and get to know each other. At the lunch, Alice tells the group what she knows about organizational news, policies, strategies, and so forth. Each team member then briefly summarizes what they have been doing, and the group discusses a general topic, such as new product ideas from elderly relatives.

Every few months, Alice organizes an "away day" for the group where the team spends two days on "technology updating." Each team member prepares an update on a relevant technology and presents it to the group. This is an offsite meeting, and plenty of time is scheduled for discussion and social interaction.

Figure 22.9 Group cohesion

- Given a stable organizational and project environment, the **3 factors that have the biggest effect on team working are:**
    1. The people in the group (**Selecting group members**)
    2. The way the group is organized (**Group organizations**)
    3. Technical and managerial communications (**Group communications**)

Selecting Group Members:

- A manager or team leader's job is to create a cohesive group and organize that group so that they work together effectively.
- This task involves selecting a group with the right balance of technical skills and personalities.
- Technical knowledge and ability should not be the only factor used to select group members.
- People who are motivated by the work are likely to be the strongest technically.
- People who are self-oriented will probably be best at pushing the work forward to finish the job.
- People who are interaction-oriented help facilitate communications within the group.

- The project manager has to control the group so that individual goals do not take precedence over organizational and group objectives.
- This control is easier to achieve if all group members participate in each stage of the project.
- Individual initiative is most likely to develop when group members are given instructions without being aware of the part that their task plays in the overall project.
- If all the members of the group are involved in the design from the start, they are more likely to understand why design decisions have been made. They may then identify with these decisions rather than oppose them.

Case study: Group composition

In creating a group for assistive technology development, Alice is aware of the importance of selecting members with complementary personalities. When interviewing potential group members, she tried to assess whether they were task-oriented, self-oriented, or interaction-oriented. She felt that she was primarily a self-oriented type because she considered the project to be a way of getting noticed by senior management and possibly being promoted. She therefore looked for one or perhaps two interaction-oriented personalities, with task-oriented individuals to complete the team. The final assessment that she arrived at was:

Alice—self-oriented
Brian—task-oriented
Chun—interaction-oriented
Dorothy—self-oriented
Ed—interaction-oriented
Fiona—task-oriented
Fred—task-oriented
Hassan—interaction-oriented

Figure 22.10 Group composition

Group Organization:

- The way a group is organized affects the group's decisions, the ways information is exchanged, and the interactions between the development group and external project stakeholders.
- Project managers are often responsible for selecting the people in the organization who will join their software engineering team.
- Getting the best possible people in this process is very important as poor selection decisions may be a serious risk to the project.
- Key factors that should influence the selection of staff are education and training, application domain and technology experience, communication ability, adaptability, and problem solving ability.

- **Important organizational questions for project managers** include the following:
    1. Should the project manager be the technical leader of the group?
    2. Who will be involved in making critical technical decisions, and how will these decisions be made? Will decisions be made by the system architect or the project manager or by reaching consensus among a wider range of team members?
    3. How will interactions with external stakeholders and senior company management be handled?
    4. How can groups integrate people who are not co-located?
    5. How can knowledge be shared across the group?

Informal Groups	Hierarchical Groups
1. Small programming groups are usually organized. 2. Group leader gets involved in the software development with the other group members. 3. The group as a whole discusses the work to be carried out, and tasks are allocated according to ability and experience. 4. More senior group members may be responsible for the architectural design. 5. Detailed design and implementation is the responsibility of the team member who is allocated to a particular task. 6. Groups are very successful, particularly when most group members are experienced and competent. Such a group makes decisions which improves cohesiveness and performance. 7. With no experienced engineers to direct the work, the result can be a lack of coordination between group members and, possibly, eventual project failure.	1. Group leader is at the top of the hierarchy. 2. Group leader has more formal authority than the group members and so can direct their work. 3. There is a clear organizational structure. 4. Decisions are made toward the top of the hierarchy and implemented by people lower down. 5. Communications are primarily instructions from senior staff; the people at lower levels of the hierarchy have relatively little communication with the managers at the upper levels. 6. These groups can work well when a well-understood problem can be easily broken down into software components that can be developed in different parts of the hierarchy. 7. This grouping allows for rapid decision making.

- In software development, effective team communications at all levels is essential:
    1. Changes to the software often require changes to several parts of the system, and this requires discussion and negotiation at all levels in the hierarchy.
    2. Software technologies change so fast that more junior staff may know more about new technologies than experienced staff. Top-down communications may mean that the project manager does not find out about the opportunities of using these new technologies. More junior staff may become frustrated because of what they see as old-fashioned technologies being used for development.
- A major challenge facing project managers is the difference in technical ability between group members.
- i.e., adopting a group model that is based on individual experts can pose significant risks.

Group Communications:

- It is absolutely essential that group members communicate effectively and efficiently with each other and with other project stakeholders.
- Good communication also helps strengthen group cohesiveness.
- Group members:
    1. Exchange information on the status of their work, the design decisions that have been made, and changes to previous design decisions.
    2. Resolve problems that arise with other stakeholders and inform these stakeholders of changes to the system, the group, and delivery plans.
    3. Come to understand the motivations, strengths, and weaknesses of other people in the group.

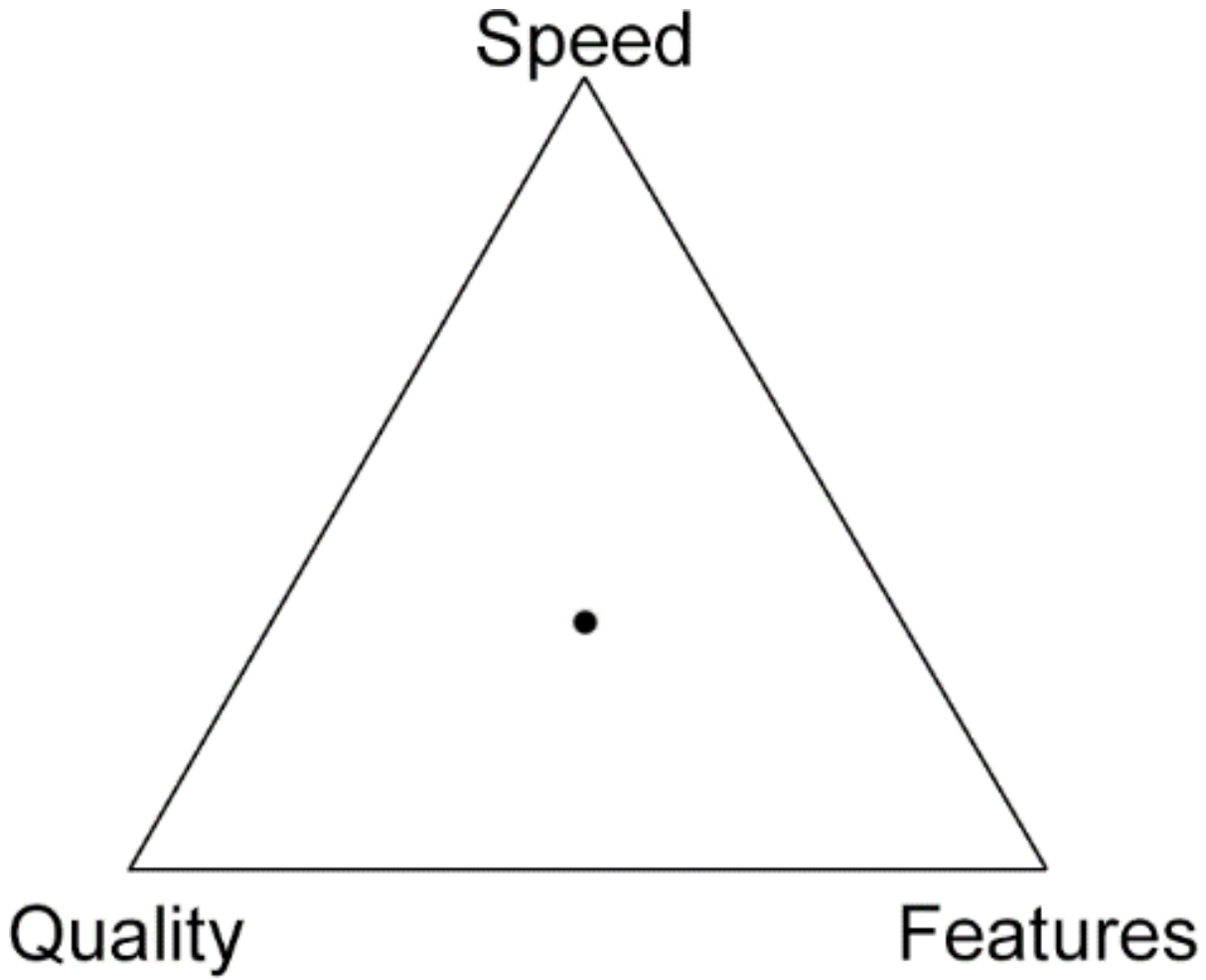
- The effectiveness and efficiency of communications are influenced by:
  1. **Group size** → As a group gets bigger, it gets harder for members to communicate effectively. The number of one-way communication links is n * (n − 1), where n is the group size.
  2. **Group structure** → People in informally structured groups communicate more effectively than people in groups with a formal, hierarchical structure.
  3. **Group composition** → People with the same personality may clash, and, as a result, communications can be inhibited.
  4. **The physical work environment** → The organization of the workplace is a major factor in facilitating or inhibiting communications.
  5. **The available communication channels** → There are many different forms of communication—face to face, email messages, formal documents, telephone, and technologies such as social networking and wikis.

- Effective communication is achieved when communications are two-way and the people involved can discuss issues and information and establish a common understanding of proposals and problems.
- All this can be done through meetings, although these meetings are often dominated by powerful personalities.
- Informal discussions when a manager meets with the team for coffee are sometimes more effective.
- Wikis and blogs allow project members and external stakeholders to exchange information, irrespective of their location. They help manage information and keep track of discussion threads, which often become confusing when conducted by email.
- You can also use instant messaging and teleconferences, which can be easily arranged, to resolve issues that need discussion.

MODULE 5

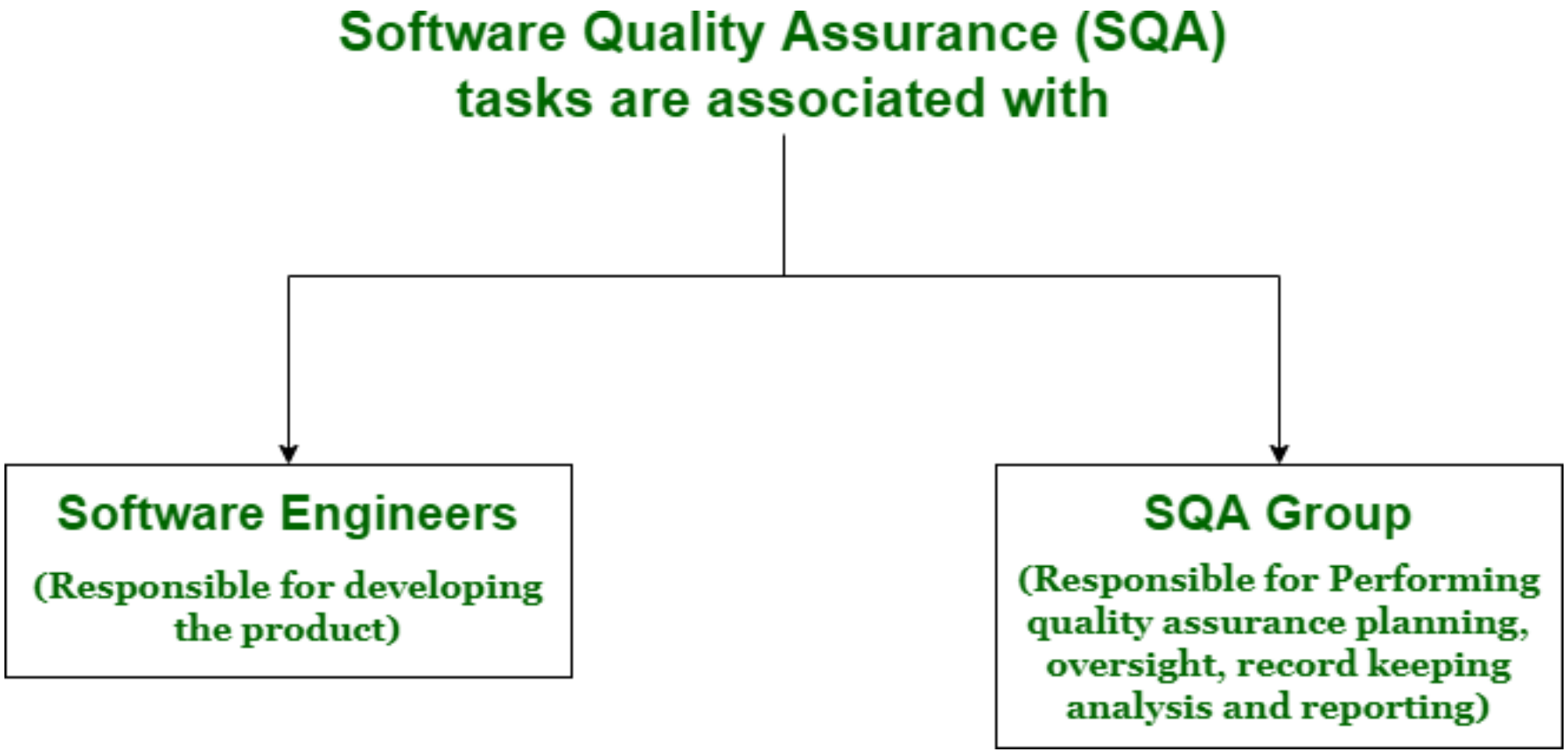

Reliability
Usability
Portability
Efficiency
Software Quality
Maintainability
Functionality

SOFTWARE QUALITY DILEMMA

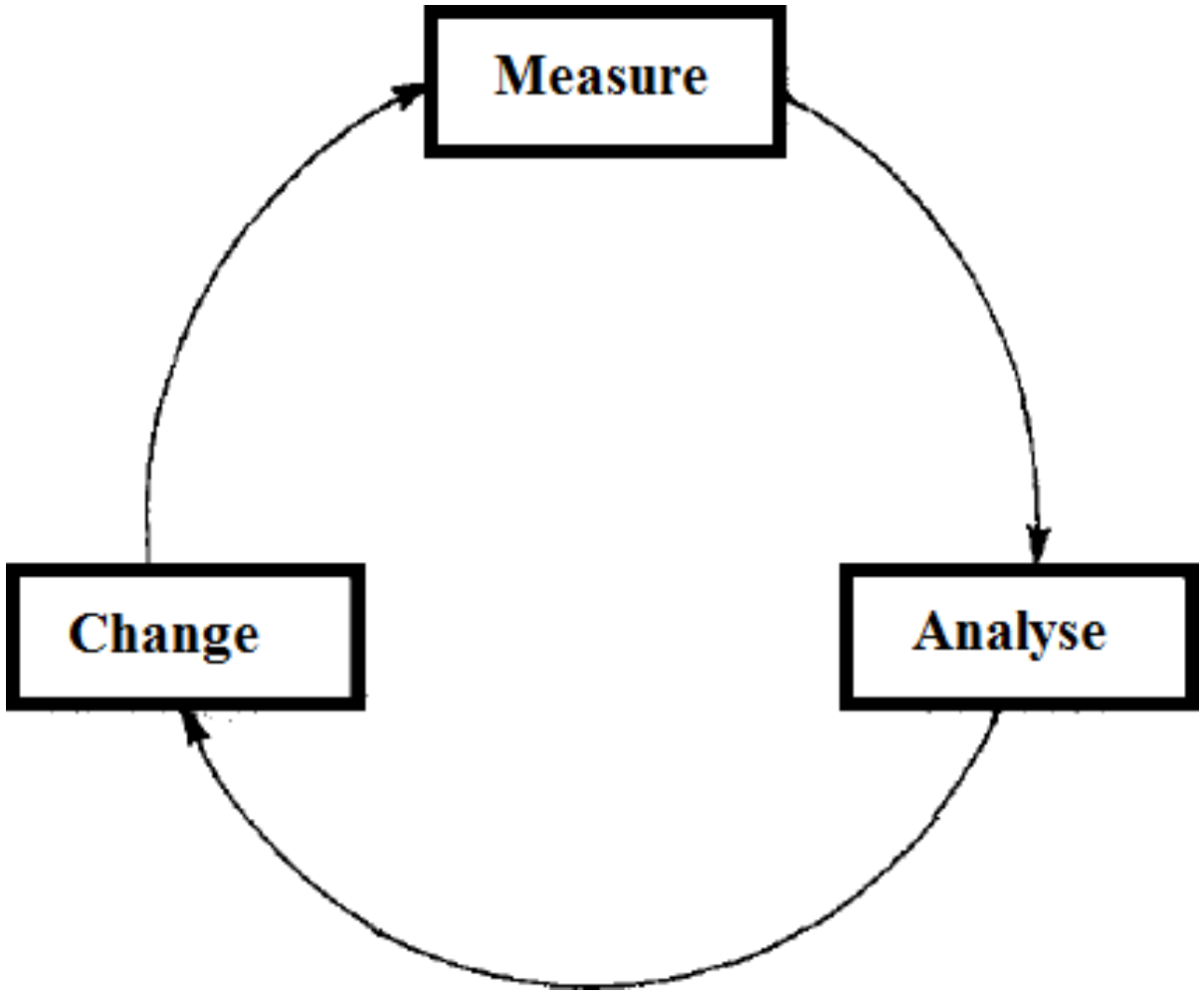


Elements of SQA

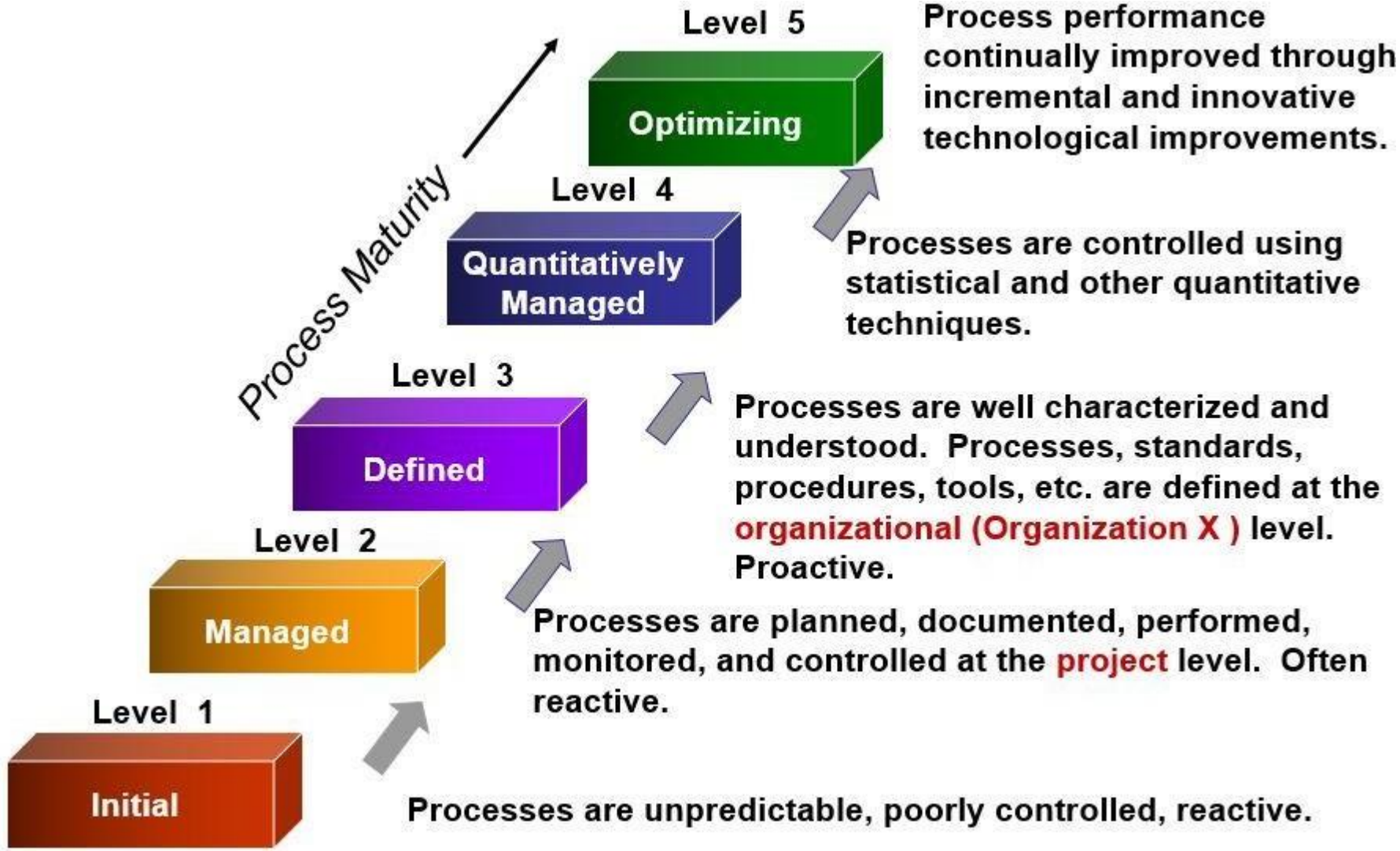
- Standards
- Reviews and Audits
- Testing
- Error/defect collection and analysis
- Change management
- Education
- Vendor management
- Security management
- Safety
- Risk management


Software Quality Assurance (SQA) tasks are associated with
Software Engineers
(Responsible for developing the product)
SQA Group
(Responsible for Performing quality assurance planning, oversight, record keeping analysis and reporting)

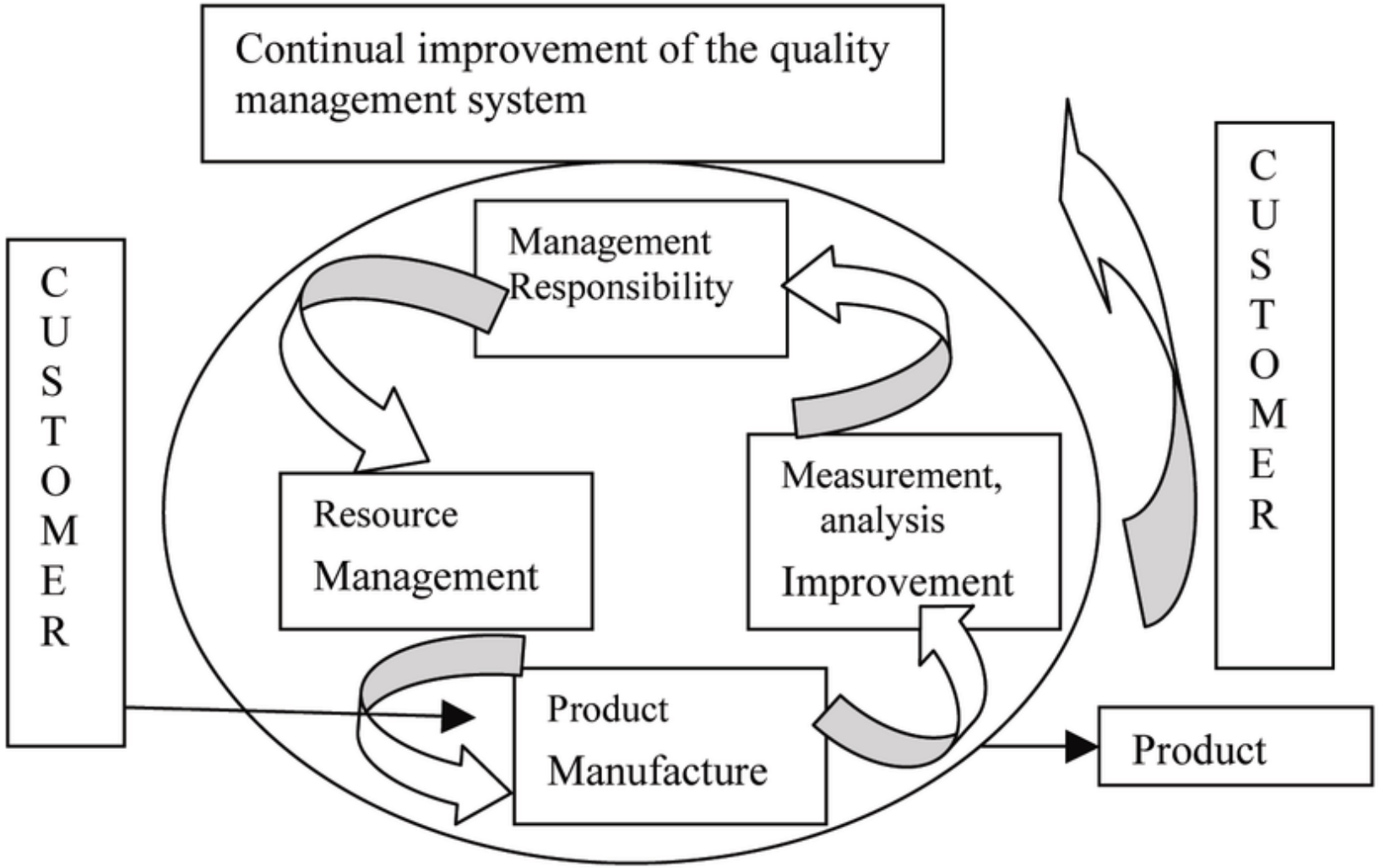
SOFTWARE PROCESS IMPROVEMENT



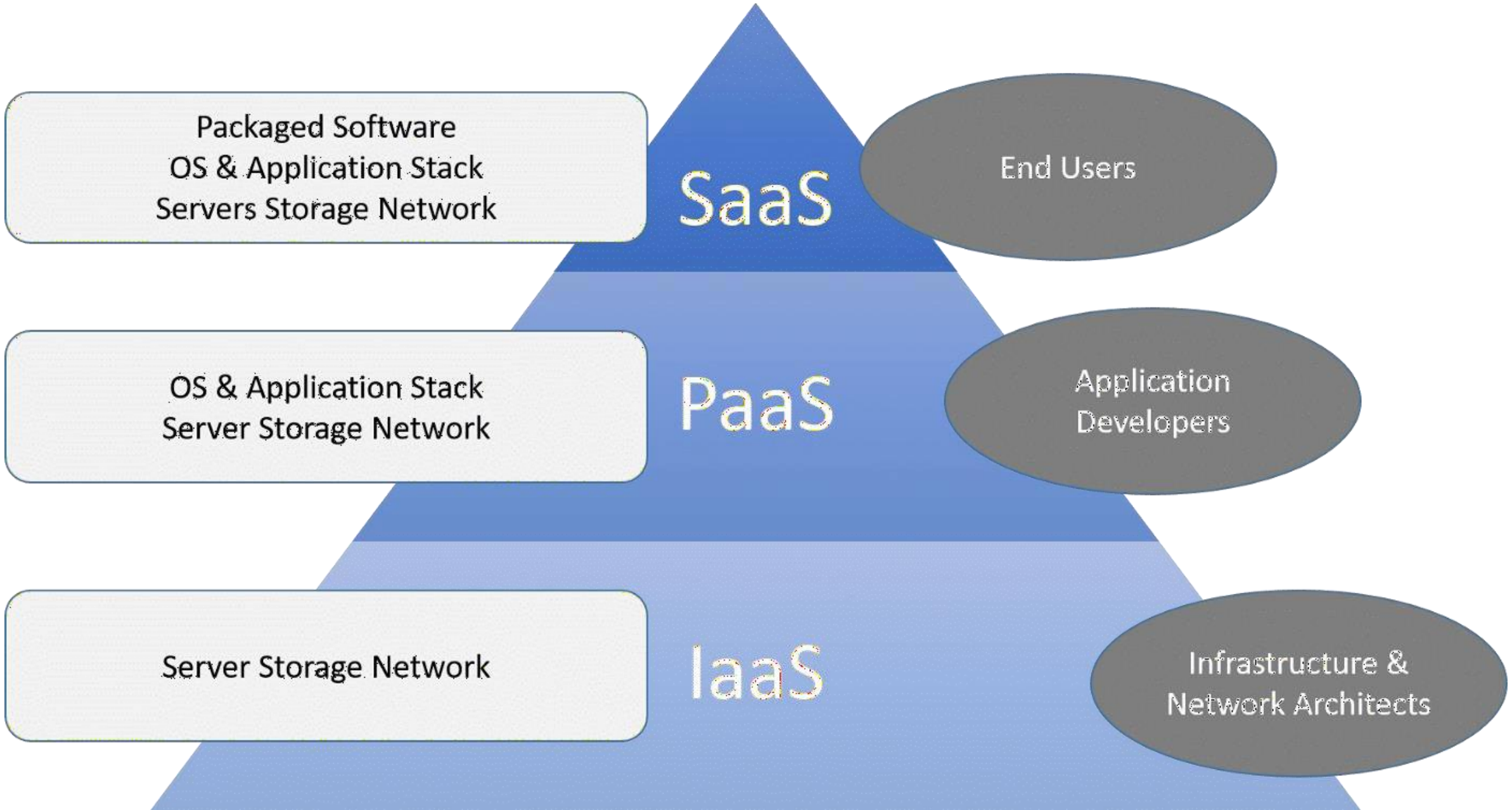
Capability Maturity Model Integration (CMMI)



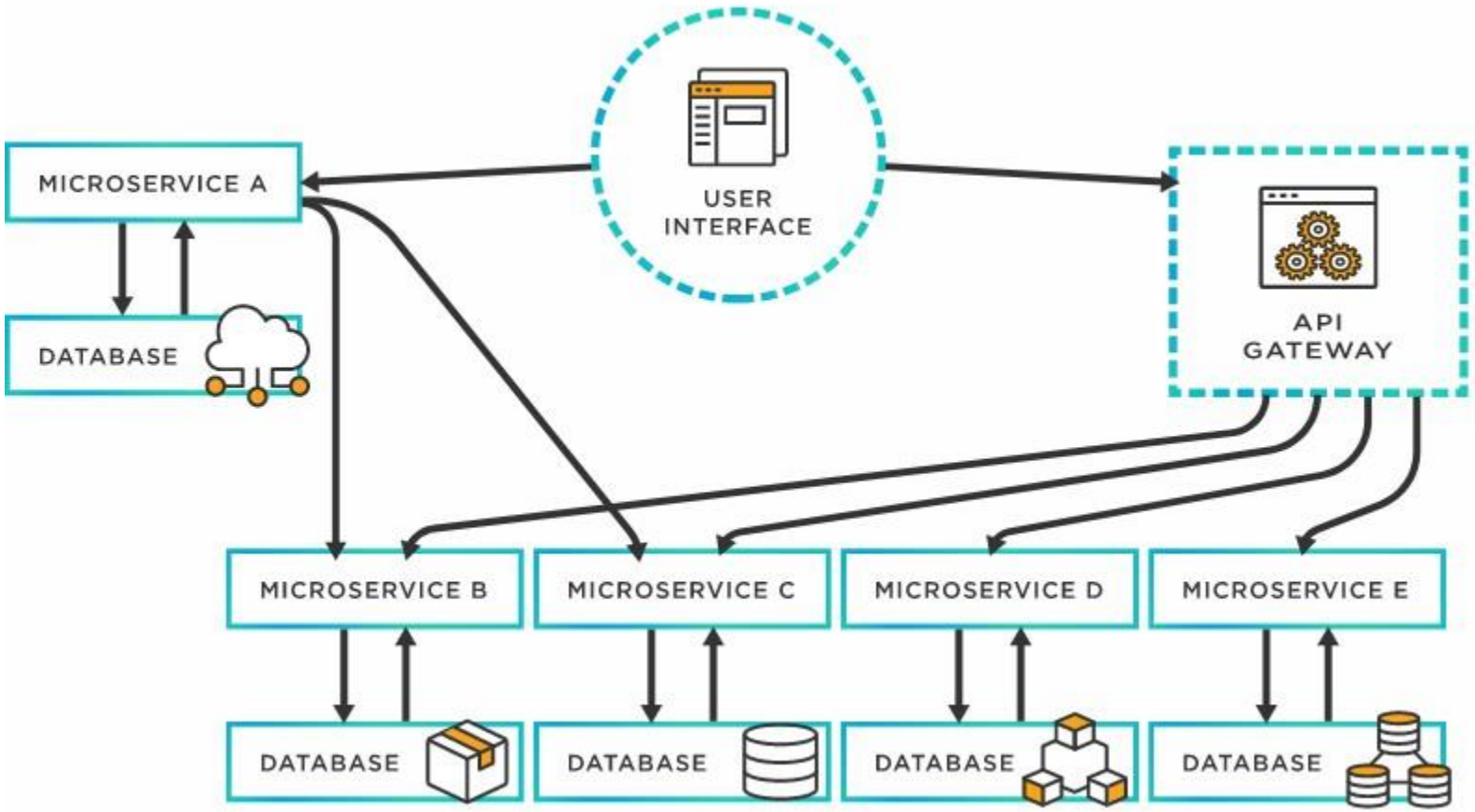
ISO 9001:2000 Quality Management System process model



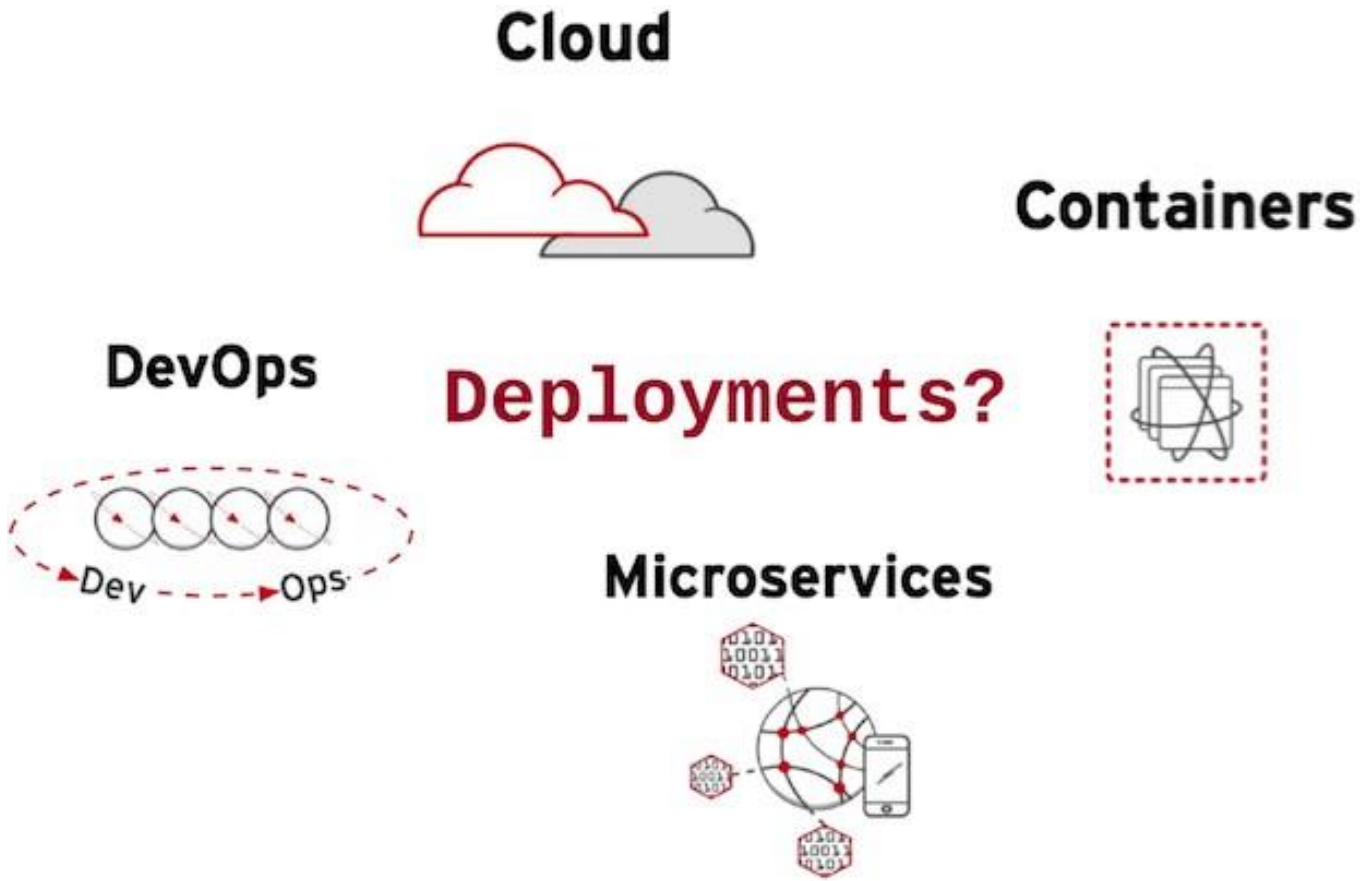
Cloud Service Models



Micro services



Micro services Deployments



CST 309	MANAGEMENT OF SOFTWARE SYSTEMS	Category	L	T	P	Credit	Year of Introduction
		PCC	3	0	0	3	2019

Preamble: This course provides fundamental knowledge in the Software Development Process. It covers Software Development, Quality Assurance, Project Management concepts and technology trends. This course enables the learners to apply state of the art industry practices in Software development.

Prerequisite: Basic understanding of Object Oriented Design and Development.

Course Outcomes: After the completion of the course the student will be able to

CO1	Demonstrate Traditional and Agile Software Development approaches **(Cognitive Knowledge Level: Apply)**
CO2	Prepare Software Requirement Specification and Software Design for a given problem. **(Cognitive Knowledge Level**: **Apply**)
CO3	Justify the significance of design patterns and licensing terms in software development, prepare testing, maintenance and DevOps strategies for a project. **(Cognitive Knowledge Level: Apply**)
CO4	Make use of software project management concepts while planning, estimation, scheduling, tracking and change management of a project, with a traditional/agile framework**. (Cognitive Knowledge Level: Apply**)
CO5	Utilize SQA practices, Process Improvement techniques and Technology advancements in cloud based software models and containers & microservices. **(Cognitive Knowledge Level: Apply**)

Mapping of course outcomes with program outcomes

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	✓	✓	✓	✓		✓						✓
CO2	✓	✓	✓	✓		✓				✓	✓	✓
CO3	✓	✓	✓	✓				✓		✓	✓	✓
CO4	✓	✓	✓	✓		✓			✓	✓	✓	✓
CO5	✓	✓	✓	✓		✓						✓

	Abstract POs defined by National Board of Accreditation		
PO#	**Broad PO**	**PO#**	**Broad PO**
PO1	Engineering Knowledge	**PO7**	Environment and Sustainability
PO2	Problem Analysis	**PO8**	Ethics
PO3	Design/Development of solutions	**PO9**	Individual and team work
PO4	Conduct investigations of complex problems	**PO10**	Communication
PO5	Modern tool usage	**PO11**	Project Management and Finance
PO6	The Engineer and Society	**PO12**	Lifelong learning

Assessment Pattern

Bloom's Category	Continuous Assessment Tests		End Semester Examination Marks
	Test1 (Percentage)	Test2 (Percentage)	
Remember	**30**	**30**	**30**
Understand	**40**	**40**	**50**
Apply	**30**	**30**	**20**
Analyse			
Evaluate			
Create			

Mark Distribution

Total Marks	CIE Marks	ESE Marks	ESE Duration
150	**50**	**100**	**3 hours**

Continuous Internal Evaluation Pattern:

Attendance : **10 marks**

Continuous Assessment Tests : **25 marks**

Continuous Assessment Assignment : **15 marks** (Each student shall identify a software development problem and prepare Requirements Specification, Design Document, Project Plan and Test case documents for the identified problem as the assignment.)

Internal Examination Pattern:

Each of the two internal examinations has to be conducted out of 50 marks.

First Internal Examination shall be preferably conducted after completing the first half of the syllabus and the Second Internal Examination shall be preferably conducted after completing the remaining part of the syllabus.

There will be two parts: Part A and Part B. Part A contains 5 questions (preferably, 2 questions each from the completed modules and 1 question from the partly covered module), having 3 marks for each question adding up to 15 marks for part A. Students should answer all questions from Part A. Part B contains 7 questions (preferably, 3 questions each from the completed modules and 1 question from the partly covered module), each with 7 marks. Out of the 7 questions in Part B, a student should answer any 5.

End Semester Examination Pattern:

There will be two parts; Part A and Part B. Part A contains 10 questions with 2 questions from each module, having 3 marks for each question. Students should answer all questions. Part B contains 2 questions from each module of which a student should answer any one. Each question can have a maximum of 2 subdivisions and carries 14 marks.

Syllabus

Module 1 : Introduction to Software Engineering (7 hours)

Introduction to Software Engineering - Professional software development, Software engineering ethics. Software process models - The waterfall model, Incremental development. Process activities - Software specification, Software design and implementation, Software validation, Software evolution. Coping with change - Prototyping, Incremental delivery, Boehm's Spiral Model. Agile software development - Agile methods, agile manifesto - values and principles. Agile development techniques, Agile Project Management. Case studies : An insulin pump control system. Mentcare - a patient information system for mental health care.

Module 2 : Requirement Analysis and Design (8 hours)

Functional and non-functional requirements, Requirements engineering processes. Requirements elicitation, Requirements validation, Requirements change, Traceability Matrix. Developing use cases, Software Requirements Specification Template, Personas, Scenarios, User stories, Feature identification. Design concepts - Design within the context of software engineering, Design Process, Design concepts, Design Model. Architectural Design - Software Architecture, Architectural Styles, Architectural considerations, Architectural Design Component level design - What is a component?, Designing Class-Based Components, Conducting Component level design, Component level design for web-apps. Template of a Design Document as per “IEEE Std 1016-2009 IEEE Standard for Information Technology Systems Design Software Design Descriptions”. Case study: The Ariane 5 launcher failure.

Module 3 : Implementation and Testing (9 hours)

Object-oriented design using the UML, Design patterns, Implementation issues, Open-source development - Open-source licensing - GPL, LGPL, BSD. Review Techniques - Cost impact of Software Defects, Code review and statistical analysis. Informal Review, Formal Technical Reviews, Post-mortem evaluations. Software testing strategies - Unit Testing, Integration Testing, Validation testing, System testing, Debugging, White box testing, Path testing, Control Structure testing, Black box testing, Testing Documentation and Help facilities. Test automation, Test-driven development, Security testing. Overview of DevOps and Code Management - Code management, DevOps automation, Continuous Integration, Delivery, and Deployment (CI/CD/CD). Software Evolution - Evolution processes, Software maintenance.

Module 4 : Software Project Management (6 hours)

Software Project Management - Risk management, Managing people, Teamwork. Project Planning, Software pricing, Plan-driven development, Project scheduling, Agile planning. Estimation techniques, COCOMO cost modeling. Configuration management, Version management, System building, Change management, Release management, Agile software management - SCRUM framework. Kanban methodology and lean approaches.

Module 5 : Software Quality, Process Improvement and Technology trends (6 hours)

Software Quality, Software Quality Dilemma, Achieving Software Quality Elements of Software Quality Assurance, SQA Tasks , Software measurement and metrics. Software Process Improvement(SPI), SPI Process CMMI process improvement framework, ISO 9001:2000 for Software. Cloud-based Software - Virtualisation and containers, Everything as a service(IaaS, PaaS), Software as a service. Microservices Architecture - Microservices, Microservices architecture, Microservice deployment.

Text Books

1. Book 1 - Ian Sommerville, Software Engineering, Pearson Education, Tenth edition, 2015.
2. Book 2 - Roger S. Pressman, Software Engineering : A practitioner’s approach, McGraw Hill publication, Eighth edition, 2014
3. Book 3 - Ian Sommerville, Engineering Software Products: An Introduction to Modern Software Engineering, Pearson Education, First Edition, 2020.

References

1. IEEE Std 830-1998 - IEEE Recommended Practice for Software Requirements SpeciÞcations
2. IEEE Std 1016-2009 IEEE Standard for Information Technology—Systems Design—Software Design Descriptions

3. David J. Anderson, Kanban, Blue Hole Press 2010
4. David J. Anderson, Agile Management for Software Engineering, Pearson, 2003
5. Walker Royce, Software Project Management : A unified framework, Pearson Education, 1998
6. Steve. Denning, The age of agile, how smart companies are transforming the way work gets done. New York, Amacom, 2018.
7. Satya Nadella, Hit Refresh: The Quest to Rediscover Microsoft's Soul and Imagine a Better Future for Everyone, Harper Business, 2017
8. Henrico Dolfing, Project Failure Case Studies: Lessons learned from other people's mistakes, Kindle edition
9. Mary Poppendieck, Implementing Lean Software Development: From Concept to Cash, Addison-Wesley Signature Series, 2006
10. StarUML documentation - https://docs.staruml.io/
11. OpenProject documentation - https://docs.openproject.org/
12. BugZilla documentation - https://www.bugzilla.org/docs/
13. GitHub documentation - https://guides.github.com/
14. Jira documentation - https://www.atlassian.com/software/jira

Course Level Assessment Questions

Course Outcome 1 (CO1):

1. What are the advantages of an incremental development model over a waterfall model?
2. Illustrate how the process differs in agile software development and traditional software development with a socially relevant case study. (Assignment question)

Course Outcome 2 (CO2):

1. How to prepare a software requirement specification?
2. Differentiate between Architectural design and Component level design.
3. How does agile approaches help software developers to capture and define the user requirements effectively?
4. What is the relevance of the SRS specification in software development?
5. Prepare a use case diagram for a library management system.

Course Outcome 3 (CO3):

1. Differentiate between the different types of software testing strategies.
2. Justify the need for DevOps practices?
3. How do design patterns help software architects communicate the design of a complex system effectively?

4. What are the proactive approaches one can take to optimise efforts in the testing phase?

Course Outcome 4 (CO4):

1. Illustrate the activities involved in software project management for a socially relevant problem?
2. How do SCRUM, Kanban and Lean methodologies help software project management?
3. Is rolling level planning in software project management beneficial? Justify your answer.
4. How would you assess the risks in your software development project? Explain how you can manage identified risks?

Course Outcome 5 (CO5):

1. Justify the importance of Software Process improvement?
2. Explain the benefits of cloud based software development, containers and microservices.
3. Give the role of retrospectives in improving the software development process.
4. Illustrate the use of project history data as a prediction tool to plan future socially relevant projects.

Model Question Paper

QP CODE:

Reg No:_____

Name :_____ **PAGES : 3**

APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

FIFTH SEMESTER B.TECH DEGREE EXAMINATION, MONTH & YEAR

Course Code: CST 309

Course Name: Management of Software Systems

Duration: 3 Hrs **Max. Marks :100**

PART A

Answer all Questions. Each question carries 3 marks

1. Why professional software that is developed for a customer is not simply the programs that have been developed and delivered.
2. Incremental software development could be very effectively used for customers who do not have a clear idea about the systems needed for their operations. Justify.
3. Identify any four types of requirements that may be defined for a software system
4. Describe software architecture
5. Differentiate between GPL and LGPL?
6. Compare white box testing and black box testing.
7. Specify the importance of risk management in software project management?
8. Describe COCOMO cost estimation model.
9. Discuss the software quality dilemma
10. List the levels of the CMMI model? **(10x3=30)**

Part B

(Answer any one question from each module. Each question carries 14 Marks)

11. (a) Compare waterfall model and spiral model **(8)**

(b) Explain Agile ceremonies and Agile manifesto **(6)**

12. (a) Illustrate software process activities with an example. **(8)**

(b) Explain Agile Development techniques and Agile Project Management **(6)**

13. (a) What are functional and nonfunctional requirements? Imagine that you are developing a library management software for your college, list eight functional requirements and four nonfunctional requirements. **(10)**

(b) List the components of a software requirement specification? **(4)**

OR

14. (a) Explain Personas, Scenarios, User stories and Feature identification? **(8)**

(b) Compare Software Architecture design and Component level design **(6)**

15. (a) Explain software testing strategies. **(8)**

(b) Describe the formal and informal review techniques. **(6)**

OR

16. (a) Explain Continuous Integration, Delivery, and Deployment CI/CD/CD) **(8)**

(b) Explain test driven development **(6)**

17. (a) What is a critical path and demonstrate its significance in a project schedule with the help of a sample project schedule. **(8)**

(b) Explain plan driven development and project scheduling. **(6)**

OR

18. (a) Explain elements of Software Quality Assurance and SQA Tasks. **(6)**

(b) What is algorithmic cost modeling? What problems does it suffer from when **(8)**

compared with other approaches to cost estimation?

19. (a) Explain elements of Software Quality Assurance and SQA Tasks. **(8)**

(b) Illustrate SPI process with an example. **(6)**

OR

20. (a) Compare CMMI and ISO 9001:2000. **(8)**

(b) How can Software projects benefit from Container deployment and Micro service deployment? **(6)**

Teaching Plan

No	Contents	No of Lecture Hrs
Module 1 : Introduction to Software Engineering (7 hours)		
1.1	Introduction to Software Engineering.[Book 1, Chapter 1]	1 hour
1.2	Software process models [Book 1 - Chapter 2]	1 hour
1.3	Process activities [Book 1 - Chapter 2]	1 hour
1.4	Coping with change [Book 1 - Chapter 2, Book 2 - Chapter 4]	1 hour
1.5	Case studies : An insulin pump control system. Mentcare - a patient information system for mental health care. [Book 1 - Chapter 1]	1 hour
1.6	Agile software development [Book 1 - Chapter 3]	1 hour
1.7	Agile development techniques, Agile Project Management.[Book 1 - Chapter 3]	1 hour
Module 2 : Requirement Analysis and Design (8 hours)		
2.1	Functional and non-functional requirements, Requirements engineering processes [Book 1 - Chapter 4]	1 hour
2.2	Requirements elicitation, Requirements validation, Requirements change, Traceability Matrix [Book 1 - Chapter 4]	1 hour
2.3	Developing use cases, Software Requirements Specification Template [Book 2 - Chapter 8]	1 hour

2.4	Personas, Scenarios, User stories, Feature identification [Book 3 - Chapter 3]	1 hour
2.5	Design concepts [Book 2 - Chapter 12]	1 hour
2.6	Architectural Design [Book 2 - Chapter 13]	1 hour
2.7	Component level design [Book 2 - Chapter 14]	1 hour
2.8	Design Document Template. Case study: The Ariane 5 launcher failure. [Ref - 2, Book 2 - Chapter 16]	1 hour
Module 3 : Implementation and Testing (9 hours)		
3.1	Object-oriented design using the UML, Design patterns [Book 1 - Chapter 7]	1 hour
3.2	Implementation issues, Open-source development - Open-source licensing - GPL, LGPL, BSD [Book 1 - Chapter 7]	1 hour
3.3	Review Techniques - Cost impact of Software Defects, Code review and statistical analysis. [Book 2 - Chapter 20]	1 hour
34	Informal Review, Formal Technical Reviews, Post-mortem evaluations. [Book 2 - Chapter 20]	1 hour
3.5	Software testing strategies - Unit Testing, Integration Testing, Validation testing, System testing and Debugging (basic concepts only). [Book 2 - Chapter 22]	1 hour
3.6	White box testing, Path testing, Control Structure testing, Black box testing. Test documentation [Book 2 - Chapter 23]	1 hour
3.7	Test automation, Test-driven development, Security testing. [Book 3 - Chapter 9]	1 hour
3.8	DevOps and Code Management - Code management, DevOps automation, CI/CD/CD. [Book 3 - Chapter 10]	1 hour
3.9	Software Evolution - Evolution processes, Software maintenance. [Book 1 - Chapter 9]	1 hour
Module 4 : Software Project Management (6 hours)		
4.1	Software Project Management - Risk management, Managing people, Teamwork [Book 1 - Chapter 22]	1 hour
4.2	Project Planning - Software pricing, Plan-driven development, Project scheduling, Agile planning [Book 1 - Chapter 23]	1 hour
4.3	Estimation techniques [Book 1 - Chapter 23]	1 hour
4.4	Configuration management [Book 1 - Chapter 25]	1 hour

4.5	Agile software management - SCRUM framework [Book 2 - Chapter 5]	1 hour
4.6	Kanban methodology and lean approaches.[Ref 9 - Chapter 2]	1 hour
Module 5 : Software Quality, Process Improvement and Technology trends (6 hours)		
5.1	Software Quality, Software Quality Dilemma, Achieving Software Quality. [Book 2 - Chapter 19]	1 hour
5.2	Elements of Software Quality Assurance, SQA Tasks , Software measurement and metrics. [Book 3 - Chapter 21]	1 hour
5.3	Software Process Improvement (SPI), SPI Process [Book 2 - Chapter 37]	1 hour
5.4	CMMI process improvement framework, ISO 9001:2000 for Software. [Book 2 - Chapter 37]	1 hour
5.5	Cloud-based Software - Virtualisation and containers, IaaS, PaaS, SaaS.[Book 3 - Chapter 5]	1 hour
5.6	Microservices Architecture - Microservices, Microservices architecture, Microservice deployment [Book 3 - Chapter 6]	1 hour