Module 1

Introduction: Software Crisis, Need for Software Engineering. Professional Software Development, Software Engineering Ethics, Case Studies.
Software Processes: Models: Waterfall Model **(Sec 2.1.1)**, Incremental Model **(Sec 2.1.2)** and Spiral Model **(Sec 2.1.3)**, Process activities.
Requirements Engineering: Requirements Engineering Processes **(Chap 4)**.Requirements Elicitation and Analysis **(Sec 4.5).** Functional and non-functional requirement, **(Sec 4.1)**.The software Requirements Document **(Sec 4.2)**. Requirements Specification **(Sec 4.3)**.Requirements validation **(Sec 4.6)**. Requirements Management **(Sec 4.7)**.

Introduction

1.1 Software crisis

Software crisis is a term used in the early days of computing science for the difficulty of writing useful and efficient computer programs in the required time. The software crisis was due to the rapid increases in computer power and the complexity of the problems that could be tackled.

The crisis manifested itself in several ways:

- Projects running over-budget
- Projects running over-time
- Software was very inefficient
- Software was of low quality
- Software often did not meet requirements
- Projects were unmanageable and code difficult to maintain
- Software was never delivered

1.2 Need for Software engineering

The need of software engineering arises because of higher rate of change in user requirements and environment on which the software is working. Following are some of the needs stated:

- **Increasing demands** As new software engineering techniques help us to build larger, more complex systems, the demands change. Systems have to be built and delivered more quickly, larger, even more complex systems are required, and systems have to have new capabilities that were previously thought to be impossible. Existing software

engineering methods cannot cope and new software engineering techniques have to be developed to meet new these new demands.

- **Low expectations** It is relatively easy to write computer programs without using software engineering methods and techniques. Many companies have drifted into software development as their products and services have evolved. They do not use software engineering methods in their everyday work. Consequently, their software is often more expensive and less reliable than it should be. Software engineering education and training is needed to address above problem.

- **Large software -** It is easier to build a wall than a house or building, likewise, as the size of the software becomes large, engineering has to step to give it a scientific process.

- **Scalability-** If the software process were not based on scientific and engineering concepts, it would be easier to re-create new software than to scale an existing one.

- **Cost-** As hardware industry has shown its skills and huge manufacturing has lower down the price of computer and electronic hardware. But, cost of the software remains high if proper process is not adapted.

- **Dynamic Nature-** Always growing and adapting nature of the software hugely depends upon the environment in which the user works. If the nature of software is always changing, new enhancements need to be done in the existing one. This is where the software engineering plays a good role.

- **Quality Management-** Better process of software development provides better and quality software product.

1.3 Professional Software Development

Software engineering is intended to support professional software development, rather than individual programming. It includes techniques that support program specification, design, and evolution, none of which are normally relevant for personal software development.

- ✓ Many people think that software is simply another word for computer programs. when we are talking about software engineering
- ✓ **Software** is not just the programs themselves but also all associated documentation and configuration data that is required to make these programs operate correctly.
- ✓ A professionally developed software system is often more than a single program. The system usually consists of a number of separate programs and configuration files that are used to set up these programs.

- ✓ It may include system documentation, which describes the structure of the system; user documentation, which explains how to use the system, and websites for users to download recent product information.

Types of Software products

Software Engineers develop software products (i.e software that can be sold to customer)

- **Generic products-**Stand-alone systems that are marketed and sold to any customer who wishes to buy them. Specification is controlled by developing organization.
  Examples – PC software such as graphics programs, project management tools; CAD software.

- **Customized products -**Software that is commissioned by a specific customer to meet their own needs. Specification is owned by buying organization.
  Examples – Attendance management system for colleges, embedded control systems, air traffic control software and traffic monitoring systems.

1.3.1 Software engineering

Software engineering is an engineering discipline that is concerned with **all aspects of software production** from the early stages of **system specification to maintaining** the system after it has gone into use. In this definition, there are two key phrases:

1. **Engineering discipline**: Engineers make things work. Apply theories, methods and tools where these are appropriate, but they use them selectively and always try to discover solutions to problems even when there are no applicable theories and methods, work to organizational and financial constraints.

2. ***All aspects of software production*** Software engineering is not just concerned with the technical processes of software development. It also includes software project management and the development of tools, methods and theories to support software production.

- ➢ **Importance of software engineering**

1. 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**.
2. **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**

There are four fundamental activities that are common to all software processes.

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

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.
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.

figure 1.1 frequently asked questions about software engineering

Product characteristics	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 utilization, 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.

Essential attributes of good software

There are Three General issues that affect most software

1. **Heterogeneity**

   Increasingly, systems are required to operate as distributed systems across networks that include different types of computer and mobile devices.

2. **Business and social change (demands for reduced delivery time)**

   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.

3. **Security and trust**

   As software is intertwined with all aspects of our lives, it is essential that we can trust that software.

1.3.2 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.

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. Example: salary payment system, phone billing system.
- **Entertainment systems**
  - These are systems that are primarily for personal use and which are intended to entertain the user.
- **Systems for modelling 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.

1.3.3 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 – changes in the ways that web based systems are engineered:

- ✓ 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.

1.4 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 out with 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

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:

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

Ethical dilemmas

- ✓ Disagreement in principle with the policies of senior management.
- ✓ Your employer acts in an unethical way and releases a safety-critical system without finishing the testing of the system.
- ✓ Participation in the development of military weapons systems or nuclear systems.

1.5 Case studies

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

Case study 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 working of this system is as shown in fig 1.3.

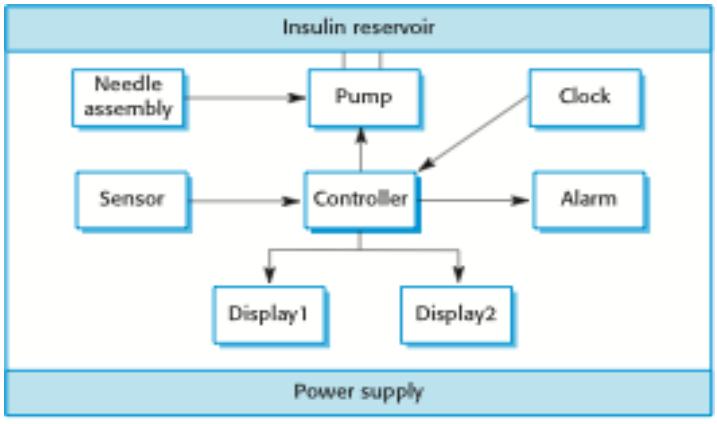

Figure 1.3 :- Insulin pump hardware architecture

Figure 1.4 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.

Case study 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 overall system is as shown in fig 1.5.
- ✓ 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.

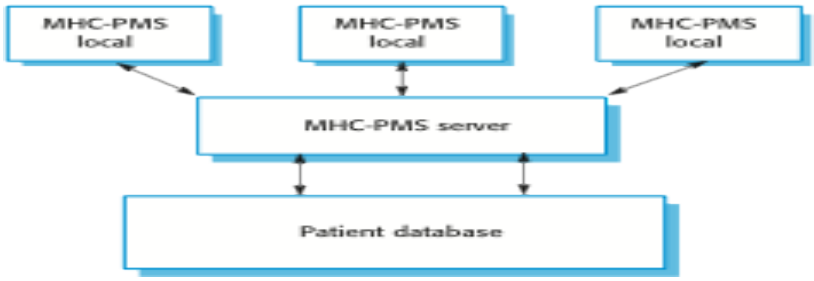

Figure 1.5 The organization of the MHC-PMS

- **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.

<u>Case study 3:</u> 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.

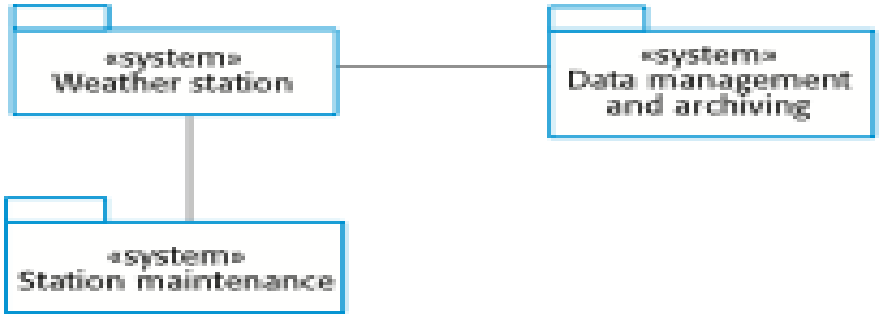

Figure 1.6 The weather station's environment

- **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.

2. 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.

2.1 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.

✓ Spiral model
- Process is represented as a spiral rather than as a sequence of activities with backtracking.

In practice, most large systems are developed using a process that incorporates elements from all of these models.

1) The waterfall model or Software life cycle

Cascade from one phase to another, this model is known as the **waterfall model or software life cycle.** The principal stages of the model map onto fundamental development activities:

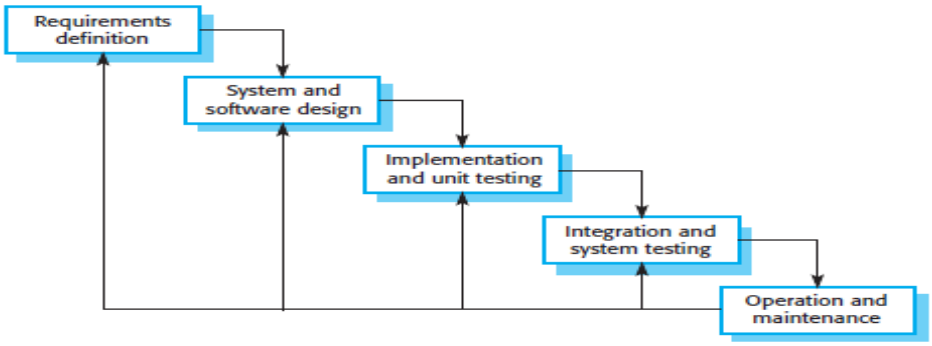

Figure 2.1 The waterfall model

- **Requirements analysis and definition**
  - The **system's services, constraints and goals** are established by consultation with **system users**.
  - They are then defined in detail and serve as a system specification.
- **System and software design**
  - The system design process partitions the requirements to either **hardware or software systems.**
  - **Software design** involves **identifying and describing** the fundamental **software system abstractions and their relationships.**
- **Implementation and unit testing**
  - The software design is realised as a **set of programs or program units**.
  - Unit testing involves verifying that each **unit meets its specification.**
- **Integration and system testing**
  - The individual program **units** or programs **are integrated and tested** as a complete system to ensure that the software requirements have been met.
  - After testing, the software system is **delivered to the customer.**
- **Operation and maintenance**
  - This is the **longest life-cycle phase**. The system is installed and put into practical use.
  - Maintenance involves **correcting errors** which were not discovered in earlier stages of the life cycle, improving the implementation of system units and enhancing the system's services as new requirements are discovered.

The result of each phase is **one or more documents** that are approved .The following **phase should not start** until the **previous phase has finished**. In practice, these stages **overlap and feed information** to each other.

The **software process is not a simple linear** model but involves a **sequence of iterations** of the development activities. Because of the costs of producing and approving documents, iterations are costly and involve significant rework.

During the **final life-cycle phase (operation and maintenance)**, the software is put into use. Errors and omissions in the original software requirements are discovered. Program and design errors emerge and the need for new functionality is identified. The system must therefore evolve to remain useful. Making these changes (software maintenance) may involve repeating previous process stages.

- **Advantages:**

1. The documentation is produced at each phase and that it fits with other engineering process models.
2. Easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process.
3. Phases are processed and completed one at a time.
4. Works well for smaller projects where requirements are very well understood.

- **Disadvantages:**

1. Its **inflexible partitioning** of the project into distinct stages.
2. **Premature freezing of requirements** may mean that the system won't do what the user wants. It may also lead to badly structured systems as design problems are circumvented by implementation tricks.
3. **Commitments must be made at an early stage in the process**, which makes it difficult to respond to changing customer requirements.
4. Once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.

2) Incremental development

Incremental development is based on the idea of **developing an initial implementation**, exposing this to **user comment** and **refining** it through **many versions** until an adequate system has been developed (Figure 2.2). **Specification, development and validation activities are interleaved** rather than separate, with rapid feedback across activities.

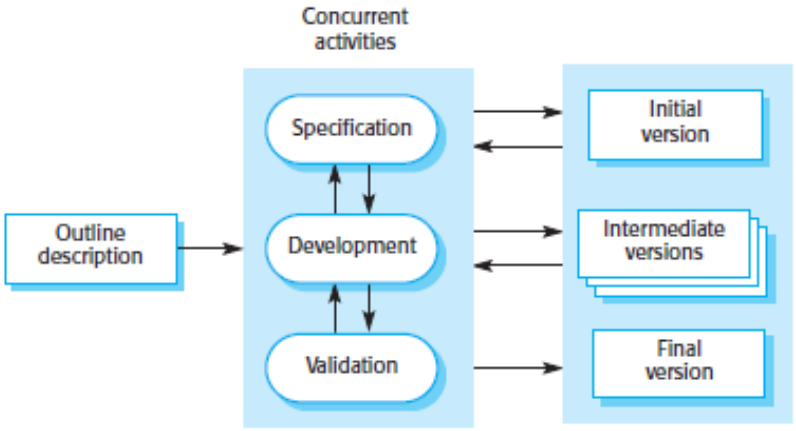

Figure 2.2 Incremental development

➢ **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.

3) Spiral development

The **spiral model of the software process** (Figure 2.3) was originally proposed by **Boehm**. Rather than represent the software process as a sequence of activities with some backtracking from one activity to another, the process is represented as a spiral.

Each loop in the spiral represents a phase of the software process. Each loop in the spiral is split into four sectors:

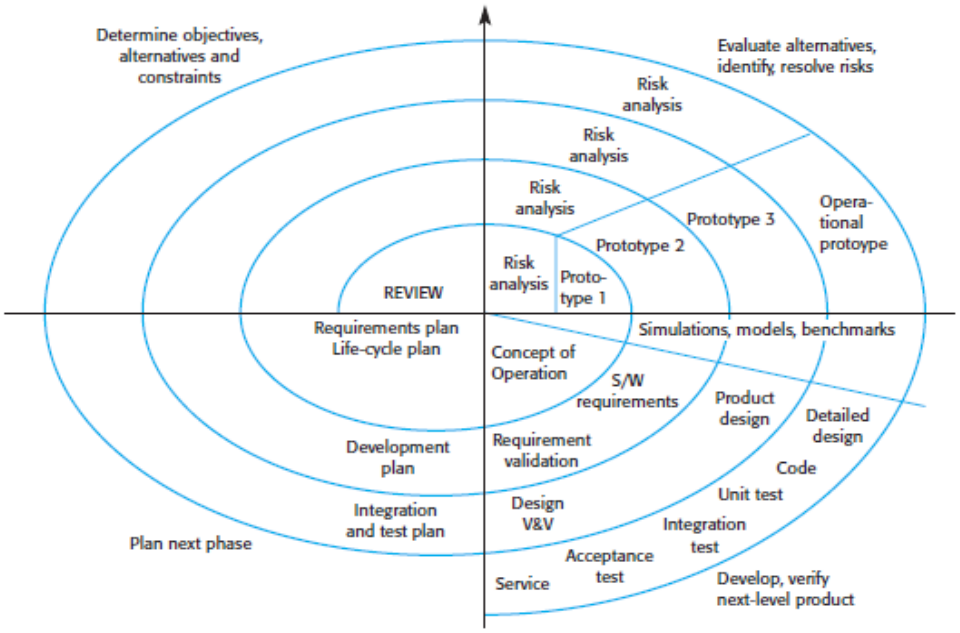

Figure 2.3 Boehm's spiral model of the software process

1. ***Objective setting*** Specific objectives for that phase of the project are defined. Constraints on the process and the product are identified and a detailed management plan is drawn up. Project risks are identified. Alternative strategies, depending on these risks, may be planned.

2. ***Risk assessment and reduction*** For each of the identified project risks, a detailed analysis is carried out. Steps are taken to reduce the risk. For example, if there is a risk that the requirements are inappropriate, a prototype system may be developed.

3. ***Development and validation*** After risk evaluation, a development model for the system is chosen. For example, if user interface risks are dominant, an appropriate development model might be evolutionary prototyping. If safety risks are the main consideration, development based on formal transformations may be the most appropriate and so on. The waterfall model may be the most appropriate development model if the main identified risk is sub-system integration.

4. *Planning* The project is reviewed and a decision made whether to continue with a further loop of the spiral. If it is decided to continue, plans are drawn up for the next phase of the project.
The main difference between the spiral model and other software process models is the explicit recognition of risk in the spiral model. For example, if the intention is to use a new programming language, a risk is that the available compilers are unreliable or do not produce sufficiently efficient object code.

- ✓ A cycle of the spiral begins by elaborating objectives such as performance and functionality.
- ✓ Alternative ways of achieving these objectives and the constraints imposed on each of them are then enumerated.
- ✓ Each alternative is assessed against each objective and sources of project risk are identified.
- ✓ The next step is to resolve these risks by information-gathering activities such as more detailed analysis, prototyping and simulation.
- ✓ Once risks have been assessed, some development is carried out, followed by a planning activity for the next phase of the process.

1.7 Process activities

There are 4 major activates involved in developing a good software,
1. software specification
2. software design and implementation
3. software validation
4. software evolution

- ➢ **Software specification**

Software specification or requirements engineering is the process of understanding and defining what services are required from the system and identifying the constraints on the system's operation and development.
The requirements engineering process is shown in Figure 2.4

There are four main phases in the requirements engineering process:

1. *Feasibility study* An estimate is made of whether the identified user needs may be satisfied using current **software and hardware technologies.** The study considers whether the proposed system will be **cost-effective** from a business point of view and whether it can be developed within existing budgetary constraints. A feasibility study should be relatively **cheap and quick.** The result should inform the decision of whether to go ahead with a more detailed analysis.

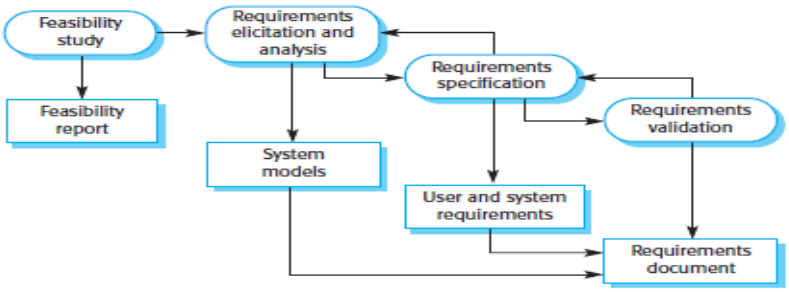

Figure 2.4The requirements engineering process

2. *Requirements elicitation and analysis* This is the process of deriving the system requirements through **observation of existing systems,** discussions with potential users and procurers, task analysis and so on. This may involve the **development of one or more system models and prototypes**. These help the analyst understand the system to be specified.

3. *Requirements specification* The activity of translating the information gathered during the analysis activity into a document that defines a set of requirements. Two types of requirements may be included in this document.

- ***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.

4.Requirements validation This activity checks the requirements for **realism, consistency and completeness**. During this process, errors in the requirements document are inevitably discovered. It must then be modified to correct these problems.

- **Software design and implementation**

The implementation stage of software development is the process of **converting a system specification into an executable system.** It always involves processes of software design and programming.

Figure 2.5 is a model of this process showing the design descriptions that may be produced at various stages of design. This diagram suggests that the stages of the design process are sequential. In fact, design process activities are interleaved. Feedback from one stage to another and consequent design rework is inevitable in all design processes.

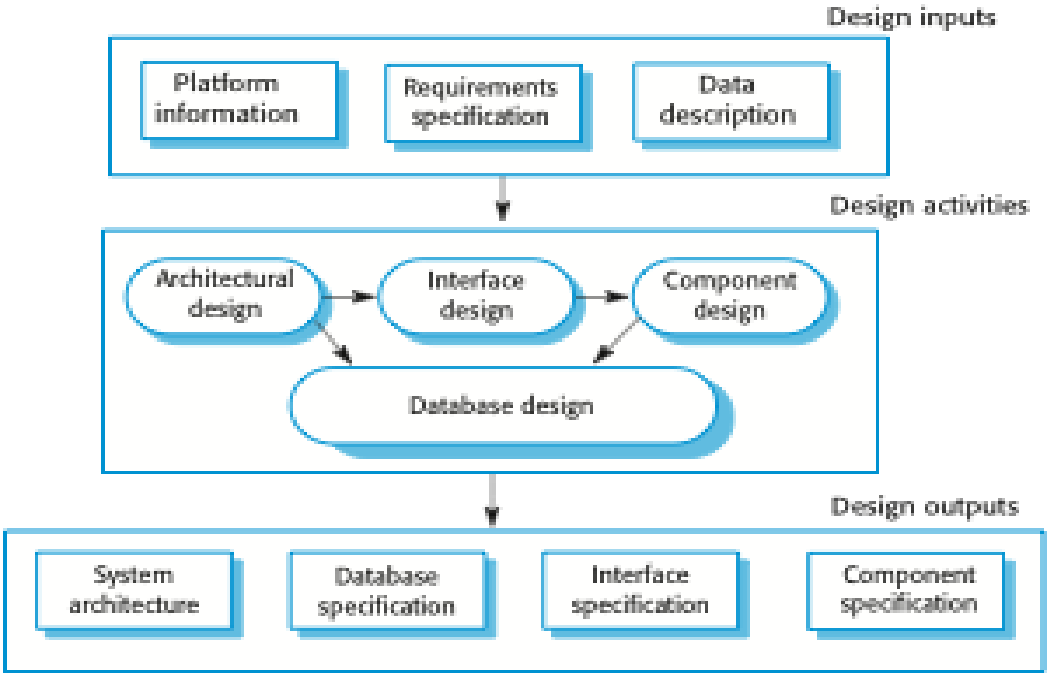

Figure 2.5 A general model of the design process

- The specific design process activities are:

1. ***Architectural design*** Architectural design is a creative process where you try to establish a system organization that will satisfy the functional and non-functional system requirements. The sub-systems making up the system and their relationships are identified and documented.
2. ***Abstract specification*** For each sub-system, an abstract specification of its services and the constraints under which it must operate is produced.
3. ***Interface design*** For each sub-system, its interface with other sub-systems is designed and documented. This interface specification must be unambiguous as it allows the sub-system to be used without knowledge of the sub-system operation.
4. ***Component design*** Services are allocated to components and the interfaces of these components are designed.
5. ***Data structure design*** The data structures used in the system implementation are designed in detail and specified.
6. ***Algorithm design*** The algorithms used to provide services are designed in detail and specified.

- Possible adaptations are:

1. The last two stages of design data structure and algorithm design may be delayed until the implementation process.

2. If an exploratory approach to design is used, the system interfaces may be designed after the data structures have been specified.
3. The abstract specification stage may be skipped, although it is usually an essential part of critical systems design.

- **Software validation**

Software validation or, more generally, verification and validation (V & V) is intended to show that a system conforms to its specification and that the system meets the expectations of the customer buying the system.
Figure 2.6 shows a three-stage testing process where system components are tested, the integrated system is tested and, finally, the system is tested with the customer's data.

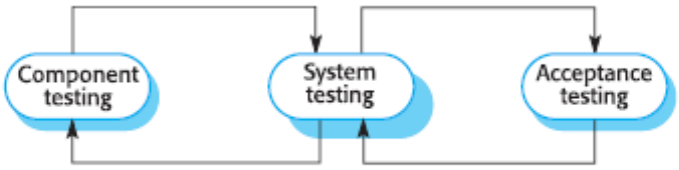

Figure 2.6:- The testing process

The stages in the testing process are:

1. Development testing(*unit testing)* Individual components are tested to ensure that they operate correctly. Each component is tested independently, without other system components. Components may be simple entities such as functions or object classes, or may be coherent groupings of these entities.

2. *System testing* The components are integrated to make up the system. This process is concerned with finding errors that result from unanticipated interactions between components and component interface problems. It is also concerned with validating that the system meets its functional and non-functional requirements and testing the emergent system properties. For large systems, this may be a multistage process where components are integrated to form sub-systems that are individually tested before they are themselves integrated to form the final system.

3. *Acceptance testing* Acceptance testing is the final stage in the testing process before the system is accepted for operational use. The system is tested with data supplied by the system customer rather than with simulated test data. Acceptance testing may reveal errors and omissions in the system requirements definition because the real data exercise the system in different ways from the test data. Acceptance testing may also reveal requirements problems where the system's facilities do not really meet the user's needs or the system performance is unacceptable.

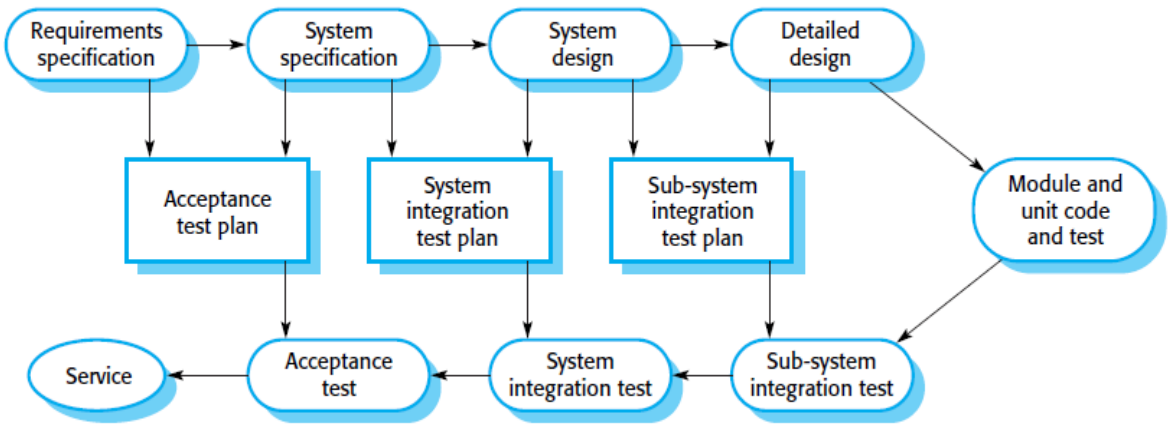

Figure 2.7 Testing phases in the software process

- Later stages of testing involve integrating work from a number of programmers and must be planned in advance. Figure 2.7 illustrates how test plans are the link between testing and development activities.

Acceptance testing has two types

1. **Alpha testing :** Custom systems are developed for a **single client**. The alpha testing process continues until the system developer and the **client agree** that the delivered system is an acceptable implementation of the system requirements.
2. **Beta testing:** When a system is to be **marketed as a software product**, involves delivering a system to a number of **potential customers** who agree to use that system. They report problems to the system developers. This exposes the product to real use and detects errors . After this feedback, the system is modified and released either for further beta testing or for general sale.

➢ **Software evolution**

- The flexibility of software systems is one of the main reasons why more and more software is being incorporated in large, complex systems.
- Changes can be made to software at any time during or after the system development.
- Few software systems are now completely new systems, and it makes much more sense to see development and maintenance as a continuum.
- Rather than two separate processes, it is more realistic to think of software engineering as an evolutionary process (Figure 2.8) where software is continually changed over its lifetime in response to changing requirements and customer needs.

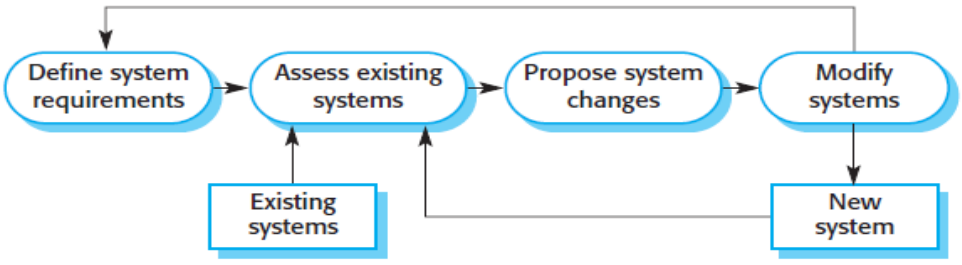

Figure 2.8 System evolution

3. Requirements engineering

The software requirements are description of features and functionalities of the target system. Requirements convey the expectations of users from the software product. The requirements can be obvious or hidden, known or unknown, expected or unexpected from client's point of view.

Requirements engineering processes

- ✓ The processes used for requirement engineering 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.

➢ Requirement Engineering

The process to gather the software requirements from client, analyze, and document them is known as requirement engineering.

The goal of requirement engineering is to develop and maintain sophisticated and descriptive 'System Requirements Specification' document

➢ 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 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., 10 mg, 20 mg) 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.

Figure3.1:-User and system requirements

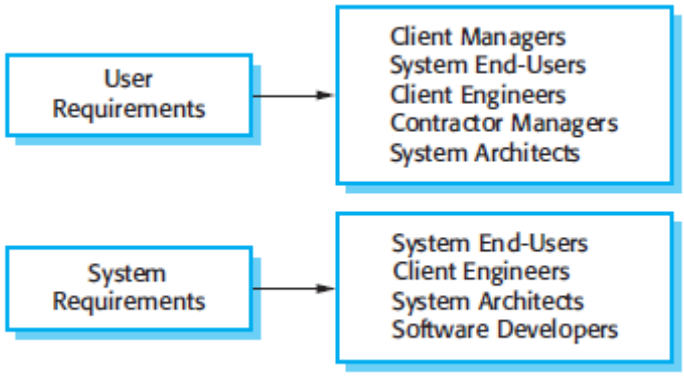

Figure 3.2:- Readers of different types of requirements specification

3.1 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 the 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.***
- ✓ The software requirements document is an agreed statement of the system requirements. It should be organized so that both system customers and software developers can use it.
- ✓ The requirements engineering process is an iterative process including requirements elicitation, specification and validation.
- ✓ Requirements elicitation and analysis is an iterative process that can be represented as a spiral of activities – requirements discovery, requirements classification and organization, requirements negotiation and requirements documentation.

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.

- **<u>Stages include:</u>**

➢ Requirements discovery
  - o Interacting with stakeholders to discover their requirements. Domain requirements are also discovered at this stage.

➢ Requirements classification and organisation
  - o Groups related requirements and organises them into coherent clusters.

- Prioritisation and negotiation
  - Prioritising requirements and resolving requirements conflicts.
- Requirements specification
  - Requirements are documented and input into the next round of the spiral.

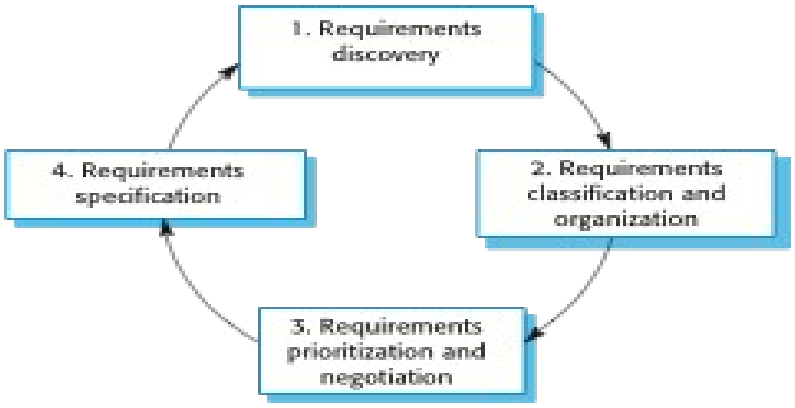

Figure 3.3 :- The requirements elicitation and analysis process

3.1.1 Problems of requirements elicitation

- ✓ 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 change.

3.1.2 Requirements discovery

- ✓ 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 regulators.
- ✓ Systems normally have a range of stakeholders.

EXAMPLE : In MHC-PMS Stakeholders,

- Patients whose information is recorded in the system.
- Doctors who are responsible for assessing and treating patients.
- Nurses who coordinate the consultations with doctors and administer some treatments.
- Medical receptionists who manage patients' appointments.

- IT staff who are responsible for installing and maintaining the system.
- Stakeholders in the MHC-PMS
- A medical ethics manager who must ensure that the system meets current ethical guidelines for patient care.
- Health care managers who obtain management information from the system.
- Medical records staffwho are responsible for ensuring that system information can be maintained and preserved, and that record keeping procedures have been properly implemented.

3.1.3 Interviewing

✓ Formal or informal interviews with stakeholders are part of most RE processes.

- **Types of interview**

  - Closed interviews based on pre-determined list of questions
  - Open interviews 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.

✓ Interviews in practice
✓ Normally a mix of closed and open-ended interviewing.
✓ Interviews are good for getting an overall understanding of what stakeholders do and how they might interact with the system.
✓ Interviews are not good for understanding domain requirements
  - Requirements engineers cannot understand specific domain terminology;
  - Some domain knowledge is so familiar that people find it hard to articulate or think that it isn't worth articulating.

3.1.4 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.

✓ Scenario for collecting medical history in MHC-PMS
✓ Scenario for collecting medical history in MHC-PMS

3.1.5 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.
✓ A set of use cases should describe all possible interactions with the system.
✓ High-level graphical model supplemented by more detailed tabular description Sequence diagrams may be used to add detail to use-cases by showing the sequence of event processing in the system.
✓ Use cases for the MHC-PMS

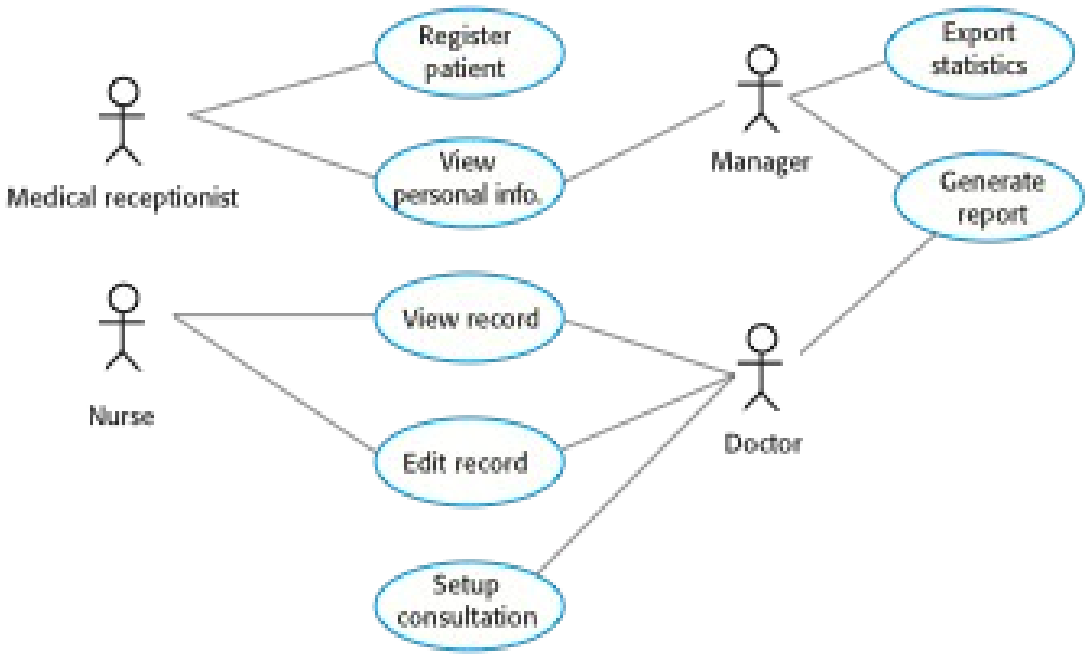

Figure 3.4 Use Case For The MHC-PMS

ACTORS	ROLE
Medical receptionist	Manages the patients register and personal information related to patients
Nurse	Record management related to patients health status.
Manager	Report management about the patients condition and framing the statistics for analysis.
Doctor	Monitor the patients, frame a case sheet

	and prescriptions.

3.1.6 Ethnography

- A social scientist spends a considerable time observing and analysing how people actually work.
- People do not have to explain or articulate their 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.

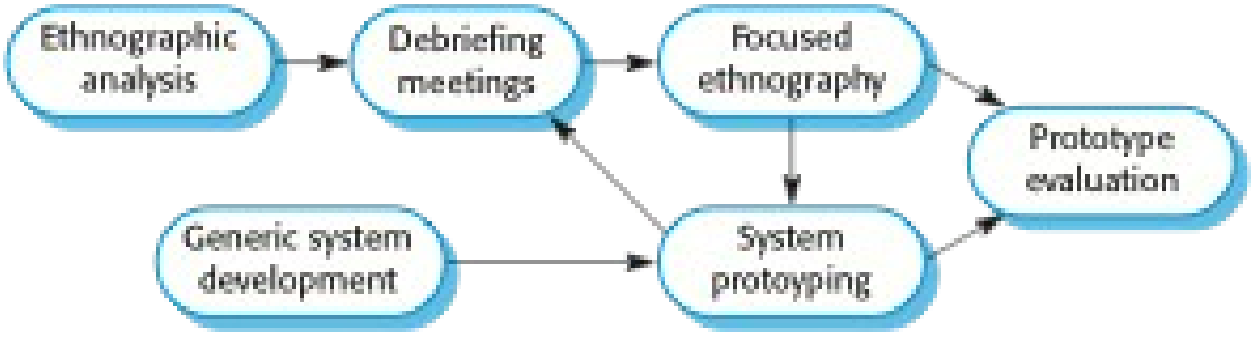

Figure: 3.5 Ethnography And Prototyping For Requirements Analysis

- **Scope of ethnography**

- Requirements that are derived from the way that people actually work rather than the way I which process definitions suggest that they ought to work.
- 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.
- Ethnography is effective for understanding existing processes but cannot identify new features that should be added to a system.

3.2 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 requirements

- 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

3.2.1 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 requirements may be high-level statements of what the system should do.
✓ Functional system requirements should describe the system services in detail.
✓ Requirements, which are related to functional aspect of software fall into this category.
✓ They define functions and functionality within and from the software system.

General Examples :

✓ Search option given to user to search from various invoices.

✓ User should be able to mail any report to management.

✓ Users can be divided into groups and groups can be given separate rights.

✓ Should comply business rules and administrative functions

- **Example: 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.

- **Imprecision**

✓ Problems arise when requirements are not precisely stated.
✓ Ambiguous requirements may be interpreted in different ways by developers and users.
✓ Consider the term ‘search’ in requirement 1
- User intention – search for a patient name across all appointments in all clinics;
- Developer interpretation – search for a patient name in an individual clinic. User chooses clinic then search.

- **Requirements completeness and consistency**

In principle, requirements should be both complete and consistent.

- ✓ Complete
  - They should include descriptions of all facilities required.
- ✓ Consistent
  - There should be no conflicts or contradictions in the descriptions of the system facilities.

3.2..2 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.
- ✓ Requirements, which are not related to functional aspect of software, fall into this category. They are implicit or expected characteristics of software, which users make assumption of.
- ✓ Non-functional requirements include -
- Security
- Logging
- Storage
- Performance
- Cost
- Disaster recovery

- ✓ Non-functional requirements may affect the overall architecture of a system rather than the individual components. There are 2 main reasons,

  - For example, to ensure that performance requirements are met, you may have to organize the system to minimize communications between components.
  - A single non-functional requirement, such as a security requirement, may generate a number of related functional requirements that define system services that are required. It may also generate requirements that restrict existing requirements.

Types of non-functional requirements

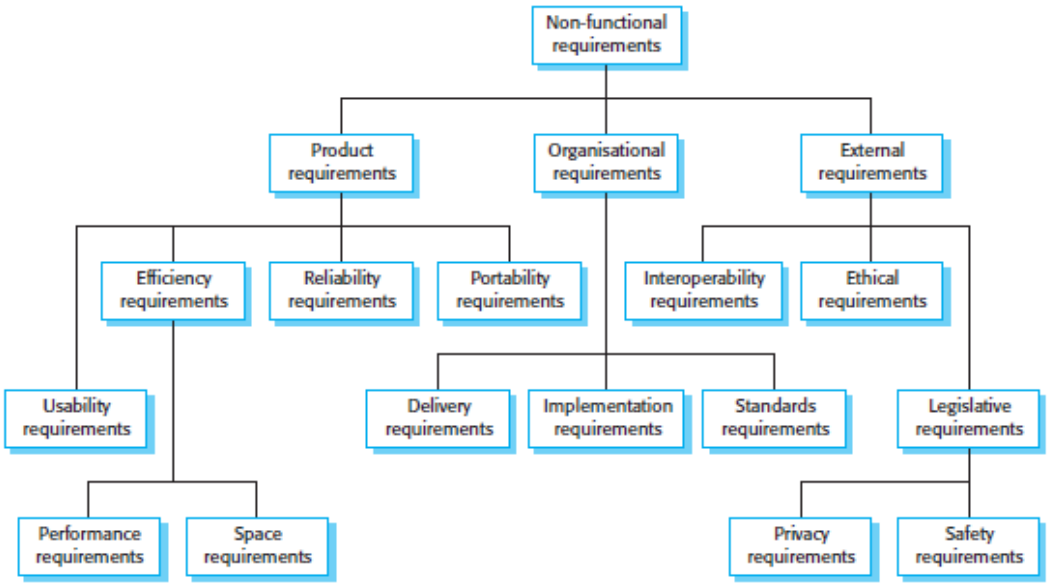

Fig 3.5 :The types of non-functional requirements

1. ***Product requirements***
   - These requirements specify product behavior.
   - **Performance requirements** on how fast the system must execute and how much memory it requires.
   - **Reliability** requirements that set out the acceptable failure rate.
   - **Portability** requirements; and usability requirements.
2. ***Organisational requirements***
   - These requirements are derived from policies and procedures in the customer's and developer's organisation.
   - **Process standards** that must be used.
   - **Implementation requirements** such as the programming language or design method used.
   - **Delivery requirements** that specify when the product and its documentation are to be delivered.
3. ***External requirements***
   - Requirements that are derived from factors external to the system and its development process.

- **Interoperability requirements** that define how the system interacts with systems in other organizations
- **Legislative requirements** that must be followed to ensure that the system operates within the law.
- **Ethical requirements** are requirements placed on a system to ensure that it will be acceptable to its users and the general public.

PRODUCT REQUIREMENT
The MHC-PMS shall be available to all clinics during normal working hours (Mon–Fri, 08.30–17.30). Downtime within normal working hours shall not exceed five seconds in any one day.

ORGANIZATIONAL REQUIREMENT
Users of the MHC-PMS system shall authenticate themselves using their health authority identity card.

EXTERNAL REQUIREMENT
The system shall implement patient privacy provisions as set out in HStan-03-2006-priv.

Figure 3.6 :-Examples of non-functional requirements in the MHC-PMS

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

Figure 3.7 Metrics for specifying nonfunctional requirements

3.3 The software requirements document (SRS- System requirements specification)

- ✓ 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.
- ✓ It is NOT a design document. As far as possible, it should set of WHAT the system should do rather than HOW it should do it.
- ✓ The requirements document has a diverse set of users, ranging from the senior management of the organization that is paying for the system to the engineers responsible for developing the software

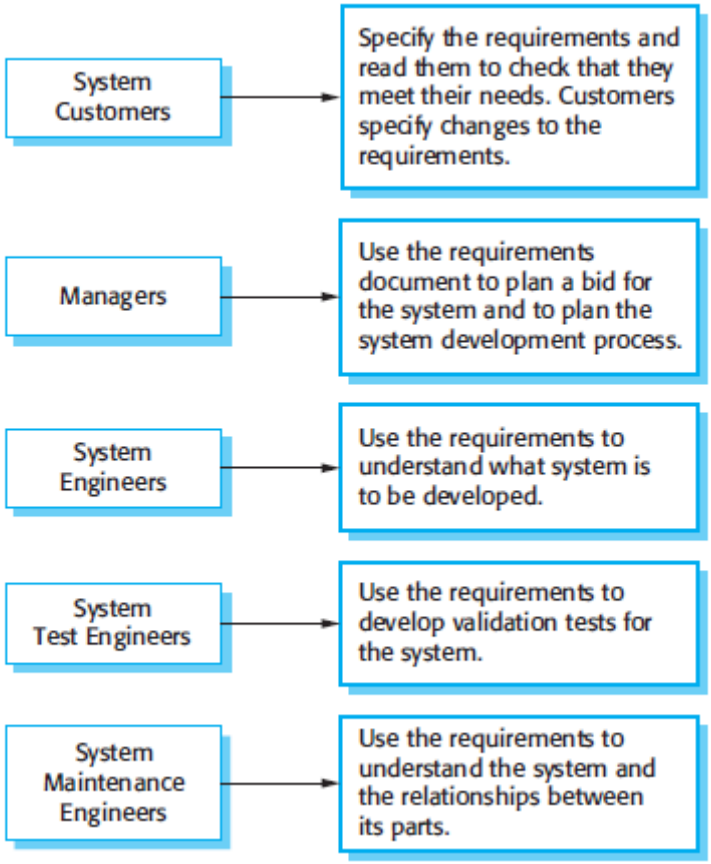

Figure 3.8 :-Users of a requirements document

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 non-functional 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.
System requirements specification	This should describe the functional and non-functional requirements in more detail. If necessary, further detail may also be added to the non-functional requirements. Interfaces to other systems may be defined.
System models	This might include graphical system models showing the relationships between the system components, 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.

Figure 3.9: The structure of a requirements document

3.4 Requirements specification

- ✓ The process of writing down 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
    - o It is therefore important that these are as complete as possible.

Notation	Description
Natural language sentences	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.

Figure 3.10 :- Ways of writing system requirements specification

- ✓ In principle, requirements should state what the system should do and the design should describe how it does this.
- ✓ In practice, requirements and design are inseparable
    - o A system architecture may be designed to structure the requirements;
    - o The system may inter-operate with other systems that generate design requirements;

- The use of a specific architecture to satisfy non-functional requirements may be a domain requirement.
- This may be the consequence of a regulatory requirement.

3.5 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. This means that the requirements can be understood by users and customers.

- **Guidelines for writing requirements**

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

Example requirements for the insulin pump software system

3.2 The system shall measure the blood sugar and deliver insulin, if required, every 10 minutes. (*Changes in blood sugar are relatively slow so more frequent measurement is unnecessary; less frequent measurement could lead to unnecessarily high sugar levels.*)

3.6 The system shall run a self-test routine every minute with the conditions to be tested and the associated actions defined in Table 1. (*A self-test routine can discover hardware and software problems and alert the user to the fact the normal operation may be impossible.*)

- **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.

3.6 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 e.g. requirements for embedded control system but is sometimes too rigid for writing business system requirements.
- ✓

Insulin Pump/Control Software/SRS/3.3.2

Function	Compute insulin dose: Safe sugar level.
Description	Computes the dose of insulin to be delivered when the current measured sugar level is in the safe zone between 3 and 7 units.
Inputs	Current sugar reading (r2), the previous two readings (r0 and r1).
Source	Current sugar reading from sensor. Other readings from memory.
Outputs	CompDose—the dose in insulin to be delivered.
Destination	Main control loop.
Action	CompDose is zero if the sugar level is stable or falling or if the level is increasing but the rate of increase is decreasing. If the level is increasing and the rate of increase is increasing, then CompDose is computed by dividing the difference between the current sugar level and the previous level by 4 and rounding the result. If the result, is rounded to zero then CompDose is set to the minimum dose that can be delivered.
Requirements	Two previous readings so that the rate of change of sugar level can be computed.
Pre-condition	The insulin reservoir contains at least the maximum allowed single dose of insulin.
Post-condition	r0 is replaced by r1 then r1 is replaced by r2.
Side effects	None.

Figure 3.11 A structured specification of a requirement for an insulin pump

- **Tabular specification**

- ✓ Used to supplement natural language.
- ✓ Particularly useful when you have to define a number of possible alternative courses of action.
- ✓ For example, the insulin pump systems bases its computations on the rate of change of blood sugar level and the tabular specification explains how to calculate the insulin requirement for different scenarios.

Condition	Action
Sugar level falling (r2 < r1)	CompDose = 0
Sugar level stable (r2 = r1)	CompDose = 0
Sugar level increasing and rate of increase decreasing ((r2 − r1) < (r1 − r0))	CompDose = 0
Sugar level increasing and rate of increase stable or increasing ((r2 − r1) ≥ (r1 − r0))	CompDose = round ((r2 − r1)/4) If rounded result = 0 then CompDose = MinimumDose

Figure 3.12 Tabular specification of computation for an insulin pump

3.7 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
  - Fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error.

3.7.1 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 validation techniques**

Developing tests for requirements to check testability.

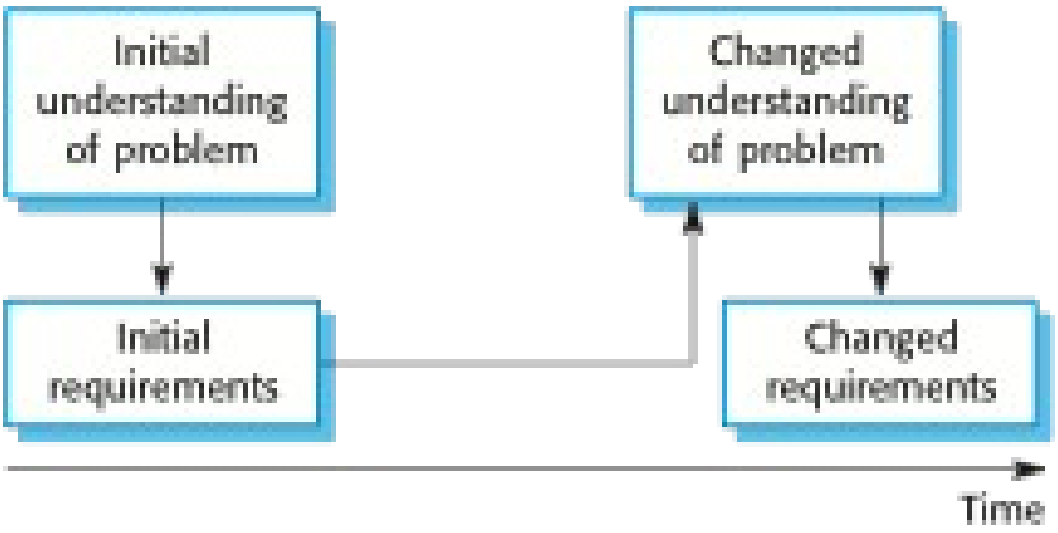

Figure : 3.14 Requirements Evolution

- ✓ Requirements reviews
- ✓ Regular reviews should be held while the requirements definition is being formulated.
- ✓ Both client and contractor staff should be involved in reviews.
- ✓ Reviews may be formal (with completed documents) or informal. Good communications between developers, customers and users can resolve problems at an early stage.

➢ **Review checks**

- ✓ Verifiability
  - o Is the requirement realistically testable?
- ✓ Comprehensibility
  - o Is the requirement properly understood?
- ✓ Traceability
  - o Is the origin of the requirement clearly stated?
- ✓ Adaptability
  - o Can the requirement be changed without a large impact on other requirements?

3.8 Requirements management

- Requirements management is the process of managing changing requirements during the requirements engineering process and system development.
- 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.

3.8.1 Requirements management planning

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

1. ***Requirements identification*** Each requirement must be uniquely identified so that it can be cross-referenced with other requirements.
2. ***A change management process*** This is the set of activities that assess the impact and cost of changes.
3. ***Traceability policies*** These policies define the relationships between each requirement and between the requirements and the system design that should be recorded.
4. ***Tool support*** Tools that may be used range from specialist requirements management systems to spreadsheets and simple database systems.

3.9 Requirements change management

Deciding if a requirements change should be accepted

1) Problem analysis and change specification

During this stage, the problem or the change proposal is analyzed to check that it is valid. This analysis is fed back to the change requestor who may respond with a more specific requirements change proposal, or decide to withdraw the request.

2) 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.

3) 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.