MODULE-I

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 requirements (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

Software Crisis

The most visible symptoms of the software crisis are

- Late delivery, over budget
  - Product does not meet specified requirements
  - Inadequate documentation

Software delivery which some kind of errors or after the completion of the scheduled date causes huge fmancial losses and is also extremely inconvenient on one's part as history tells us. Situations like the Y2K problem which was disastrous in most of the countries is considered to be one of the catastrophic failures in terms of economic, administrative and political functioning may be termed as a Software Crisis. Thus a Software crisis may be explained as a mismatch in software deliverables and the expectations a user have from the computer system in terms of its capacity to work. During 20th century it was an emerging issue since computing flourished and software was incapable to catch up. Due to the software crisis, programmers had to struggle always in order to keep pace.

- The 2003 Northeast blackout was one of the most important power system failures in the North American history. Due to this blackout, large number of power plants failed and approximately fifty million customers faced power loss which resulted in a huge fmancial threat to currency. Afterwards, it was understood that a software bug was the reason behind this unprecedented failure in the power monitoring/noting along with the management system.
- Y2K problem basically referred to the huge amount of drawback in the processing of data after the year 2000. During 1990s, realization of the experts began to understand this important shortcoming in computer application and then in order to handle such a problem millions were spent.
- The year 1996 physically witnessed the end of Arian-5 space rocket which was made with a cost of $7000 million within a period of ten years in less than a minute after the rocket launch. It was later observed that there was a software bug in the rocket guidance system.
- One of the biggest banks of US in 1996, did wrong calculation of the credited accounts of approximately 800 customers to the amount of $9241 acs. It was later detected that the problem took place due to a programming bug in the banking software.
- At the time of the Gulf War in 1991, the U.S.A. Patriot missiles acted as a defence against Scud missiles used by Iraq. But the Patriot failed to target hit the Scud several times. Thus, twenty eight US soldiers were put to sleep in Dhahran, Saudi Arabia. An inquiry

into the incident made clear the cause of the failed target as the small bug again and finally resulted in the wrong calculation of the missile path.

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.

- ***Large software -*** It is easier to build a wall than to a house or building, likewise, as the size of software become large engineering has to step to give it a scientific process.
- ***Scalability-*** If the software processes 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 the cost of software remains high if proper process is not adapted.
- ***Dynamic Nature-*** The always growing and adapting nature of software hugely depends upon the environment in which user works. If the nature of software is always changing, new enhancements need to be done in the existing one. This is where software engineering plays a good role.
- ***Quality Management-*** Better process of software development provides better and quality software product.

Professional Software Development

What is software?

- Software is a set of a program associated with documentation and configuration data that is needed to make these programs operate correctly.
- A software system usually consists: of a number of separate programs, configuration files, which are used to set up these programs, system documentation, which describes the structure of the system, and user documentation, which explains how to use the system and web sites or users to download recent product information.
- Software products may be
  - Generic - developed to be sold to a range of different customers
  - Bespoke (custom) developed for a single customer according to their specification

What is software engineering?

- Software engineering is an engineering discipline which is concerned with all aspects of software production.
- Engineering discipline
  - Engineers make things work. They 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.
  - Engineers also recognise that they must work to organisational and financial constraints, so they look for solutions within these constraints.
- All aspects of software production
- Software engineering is not just concerned with the technical processes of software development but also with activities such as software project management and with the development of tools, methods and theories to support software production.

What is the difference between software engineering and computer science?

Computer science is concerned with theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software.

Computer science theories are currently insufficient to act as a complete underpinning for software engineering

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.
- System engineers are involved in system specification, architectural design, integration and deployment.
- Software engineering is part of this process concerned with developing the software infrastructure, control, applications and databases in the system.

What is a software process?

A software process is the set of activities and associated results that produce a software product. *4* Generic activities in all software processes are:

- Specification - what the system should do and its development constraints.
- Development - production of the software system.
- Validation - checking that the software is what the customer wants.
- Evolution - changing the software in response to changing demands.

What is a software process model?

A simplified representation of a software process, presented from a specific perspective 4 Examples of process perspectives are

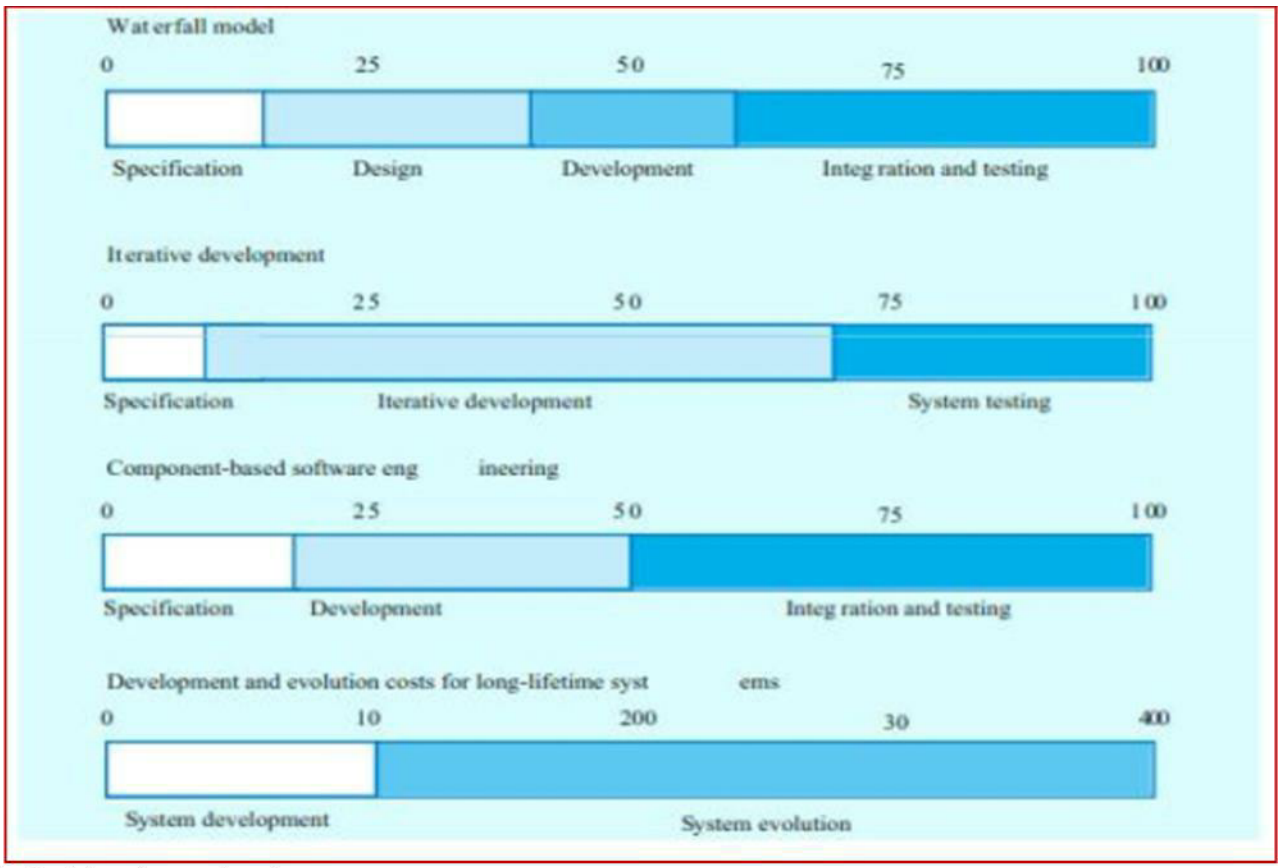
- A work low model - sequence of activities
- A data-flow model - information flow
- A role/action model - who does what

Generic process models

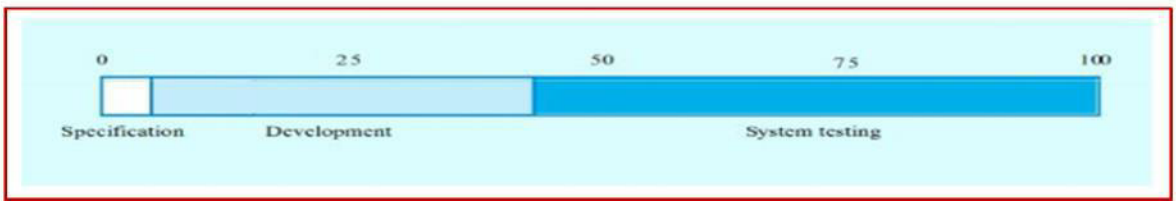
- The waterfall approach: separate process phases such as requirements specification, software design, implementation, testing and so on. After each stage is defined it is 'signed-off, and development goes on to the following stage.
- Iterative development: This approach interleaves the activities of specification, development and validation.
- Component-based software engineering (CBSE): This technique assumes that parts of the system already exist. The system development process focuses on integrating these parts rather than developing them from scratch.

What are the costs of software engineering?

- Roughly 60% of costs are development costs, 40% are testing costs. For custom software, evolution costs often exceed development costs.
- Costs vary depending on the type of system being developed and the requirements of system attributes such as performance and system reliability.
- Distribution of costs depends on the development model that is used.



✧ Product development costs



What are the attributes of good software?

- The software should deliver the required functionality and performance to the user and should be maintainable, dependable and acceptable.
- **Maintainability:** Software must evolve to meet changing needs;
- **Dependability:** Software must be trustworthy; (it has a range of characteristic, including reliability, security and safety).
- **Efficiency:** Software should not make wasteful use of system resources;
- Acceptability/Usability: Software must accepted by the users for which it was designed. This means it must be understandable, usable and compatible with other systems.

What are the key challenges facing software engineering

- Heterogeneity: Developing techniques for building software that can cope with heterogeneous platforms and execution environments;
- Delivery: Developing techniques that lead to faster delivery of software;
- Trust: Developing techniques that demonstrate that software can be trusted by its users.

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.
- **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.

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.

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 (discussed in Chapter 19) 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.

Web-based software engineering

Web-based systems are complex distributed systems but the fundamental principles of software engineering discussed previously are as applicable to them as they are to any other types of system.

- The fundamental ideas of software engineering, discussed in the previous section, apply to web -based software in the same way that they apply to other types of software system.

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.

- **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.

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 fmishing the testing of the system.
- Participation in the development of military weapons systems or

nuclear systems.

Case studies

A personal insulin pump

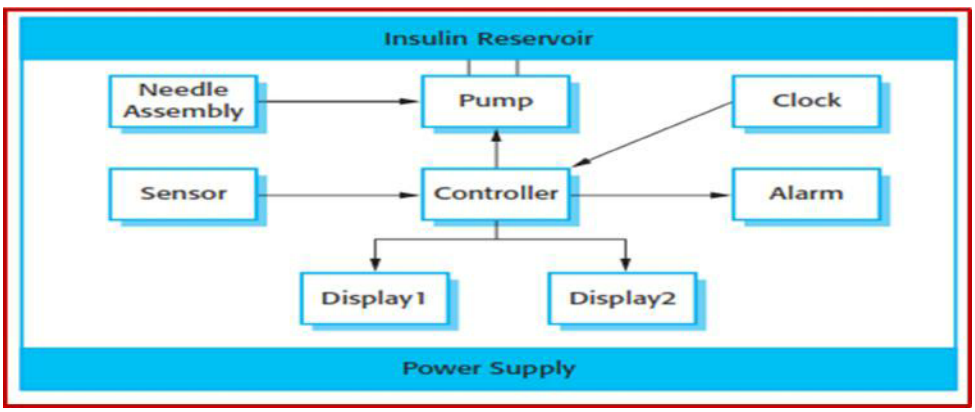
- An embedded system in an insulin pump used by diabetics to maintain blood glucose control. 4 A mental health case patient management system
- A system used to maintain records of people receiving care for mental health problems. 4 A wilderness weather station
- A data collection system that collects data about weather conditions in

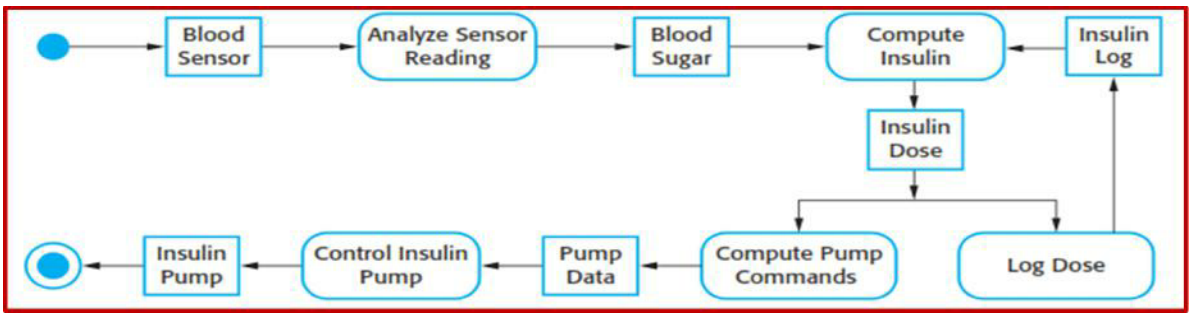
remote areas.

Insulin pump control system

Collects data from a blood sugar sensor and calculates the amount of insulin required to be injected. 4 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.





Essential high-level requirements

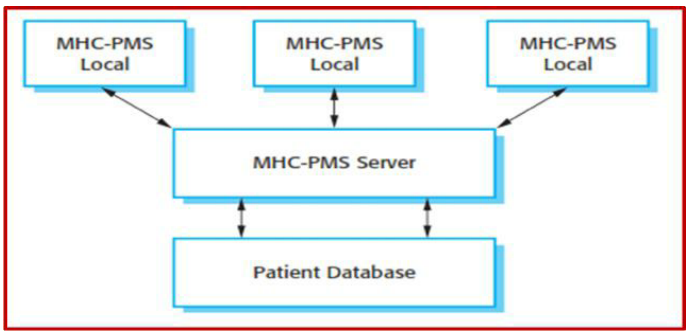
- The system shall be available to deliver insulin when required.
- The system shall perform reliably and deliver the correct amount of insulin to counteract the current level of blood sugar.
- The system must therefore be designed and implemented to ensure that the system always meets these requirements.

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.
- 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.



MHC-PMS key feature

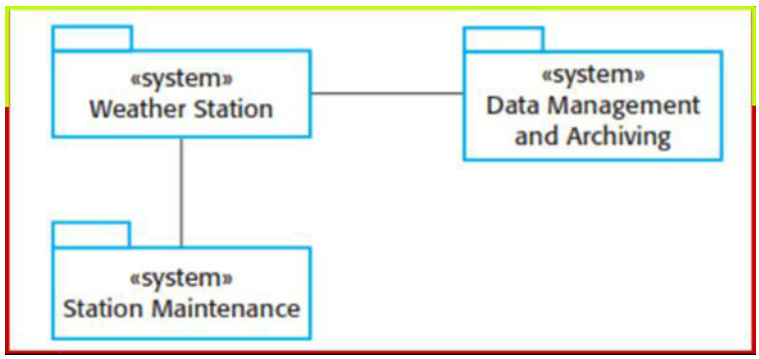
- Individual care management
  - Clinicians can create records for patients, edit the information in the system, view patient history, etc. The system supports data summaries so that doctors can quickly learn about the key problems and treatments that have been prescribed.

- **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.
- **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.

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 Processes:

Models

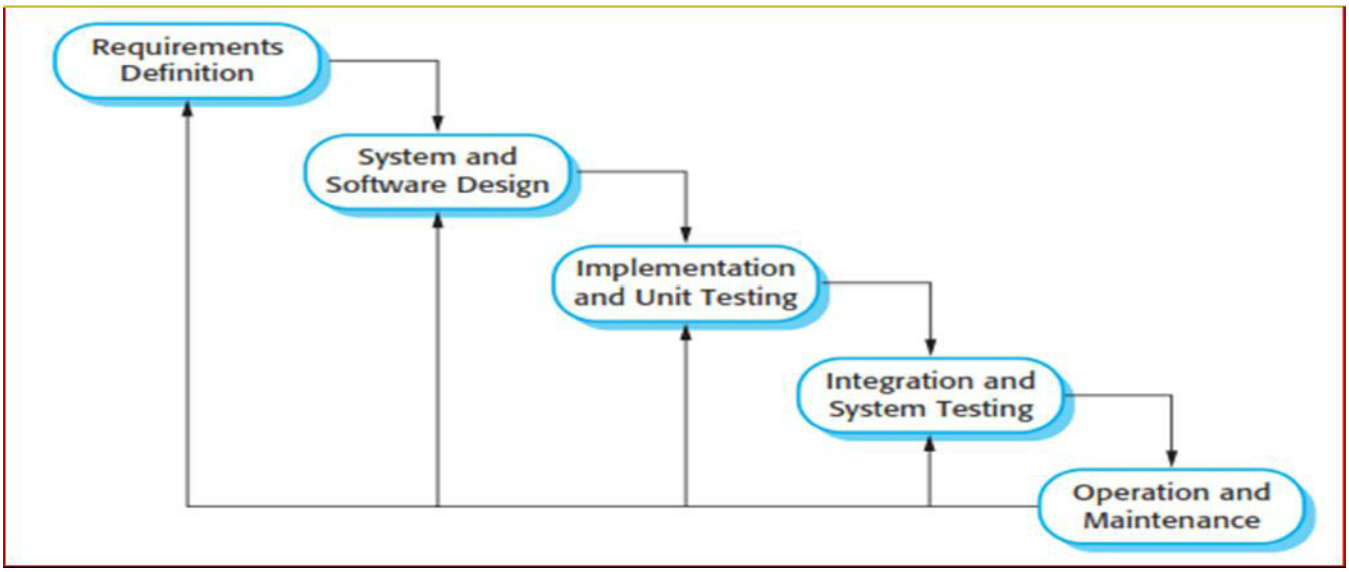
- **The waterfall model**
  Plan-driven model. Separate and distinct phases of specification and development.
- **Incremental development**
  Specification, development and validation are interleaved. May be plan-driven or agile.
- **Reuse-oriented software engineering**
  The system is assembled from 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

- **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 systems design process allocates the requirements to either hardware or software systems by establishing an overall system architecture. Software design involves identifying and describing the fundamental software system abstractions and their relationships.
- **Implementation and unit testing**
  During this stage, the software design is realized as a set of programs or program units. Unit testing involves verifying that each unit meets its specification.

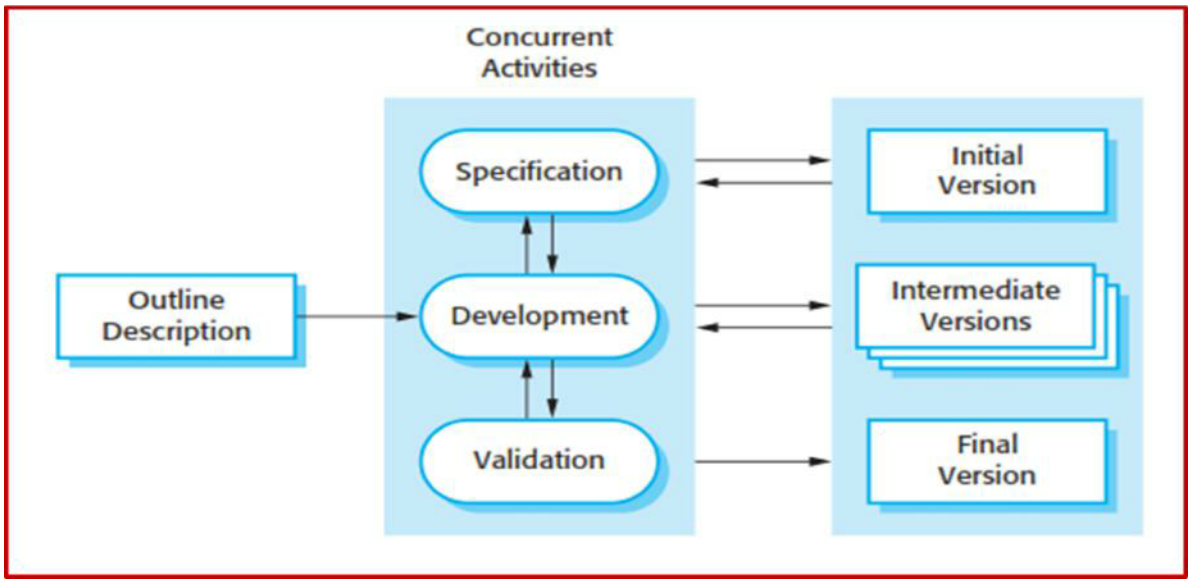
- **Integration and system testing**
  The individual program units or programs are integrated and tested as a complete system to ensure that the software requirements have been met. After testing, the software system is delivered to the customer.
- **Operation and maintenance**
  The system is installed 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.



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 software development, which is a fundamental part of agile approaches, is better than a waterfall approach for most business, e-commerce, and personal systems.
- Incremental development reflects the way that we solve problems.
- We rarely work out a complete problem solution in advance but move toward a solution in a series of steps, backtracking when we realize that we have made a mistake. By developing the software incrementally, it is cheaper and easier to make changes in the software as it is being developed.
- Each increment or version of the system incorporates some of the functionality that is needed by the customer.
- Generally, the early increments of the system include the most important or most urgently required functionality. This means that 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.
- 4 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

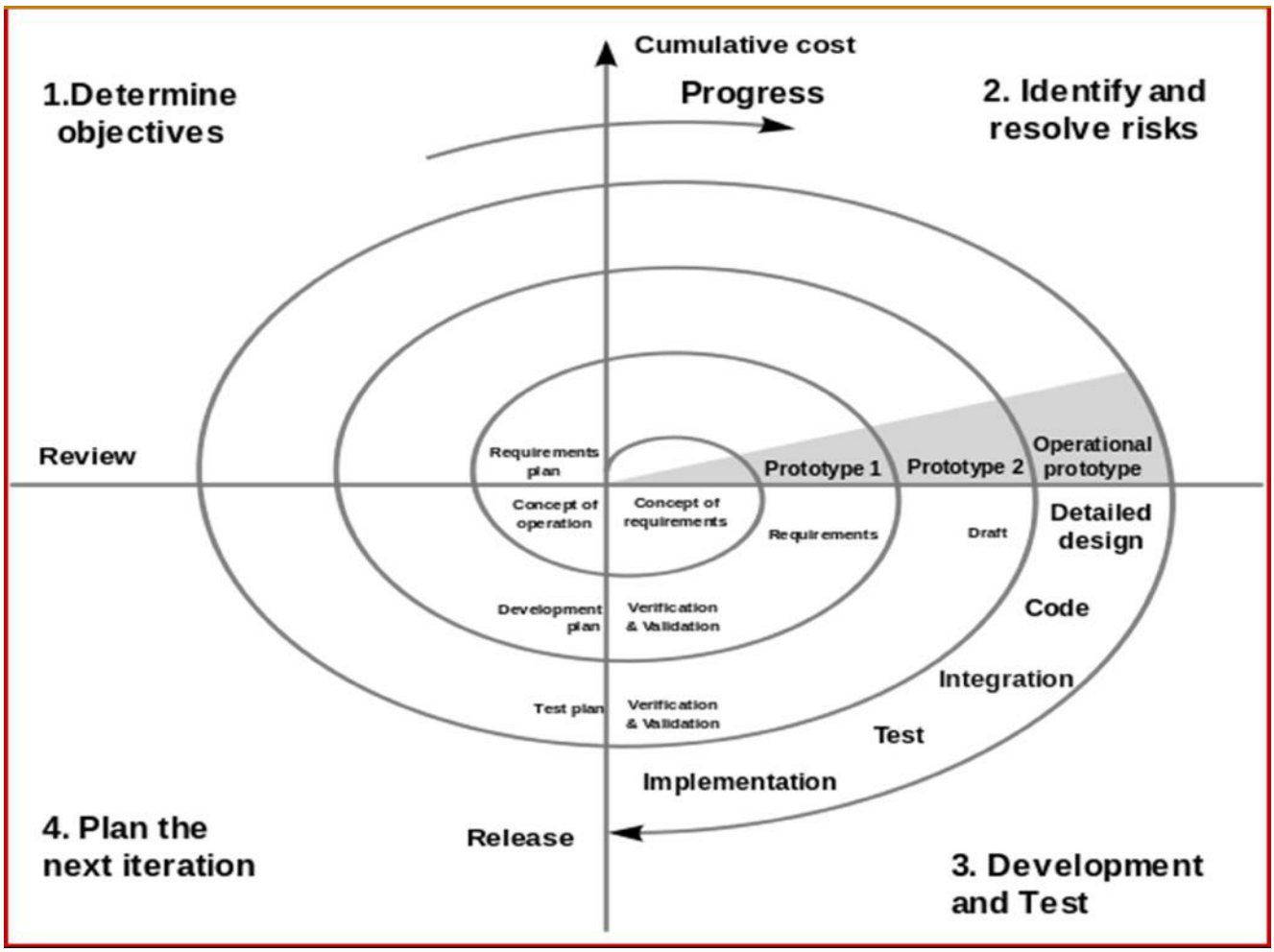
The process is not visible.

- 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.

Spiral Model



Identification

This phase starts with gathering the business requirements in the baseline spiral. In the subsequent spirals as the product matures, identification of system requirements, subsystem requirements and unit requirements are all done in this phase.

This phase also includes understanding the system requirements by continuous communication between the customer and the system analyst. At the end of the spiral, the product is deployed in the identified market.

Design

The Design phase starts with the conceptual design in the baseline spiral and involves architectural design, logical design of modules, physical product design and the fmal design in the subsequent spirals.

Construct or Build

The Construct phase refers to production of the actual software product at every spiral. In the baseline spiral, when the product is just thought of and the design is being developed a POC (Proof of Concept) is developed in this phase to get customer feedback.

- Then in the subsequent spirals with higher clarity on requirements and design details a working model of the software called build is produced with a version number. These builds are sent to the customer for feedback.

Evaluation and Risk Analysis

Risk Analysis includes identifying, estimating and monitoring the technical feasibility and management risks, such as schedule slippage and cost overrun. After testing the build, at the end of first iteration, the customer evaluates the software and provides feedback.

Spiral Model Application

When there is a budget constraint and risk evaluation is important.

- For medium to high-risk projects.
- Long-term project commitment because of potential changes to economic priorities as the requirements change with time.
- Customer is not sure of their requirements which are usually the case.
- Requirements are complex and need evaluation to get clarity.
- New product line which should be released in phases to get enough customer feedback.
- Significant changes are expected in the product during the development cycle.

The advantages of the Spiral SDLC Model are as follows

- Changing requirements can be accommodated.
- Allows extensive use of prototypes.
- Requirements can be captured more accurately.
- Users see the system early.
- Development can be divided into smaller parts and the risky parts can be developed earlier which helps in better risk management.

The disadvantages of the Spiral SDLC Model are as follows

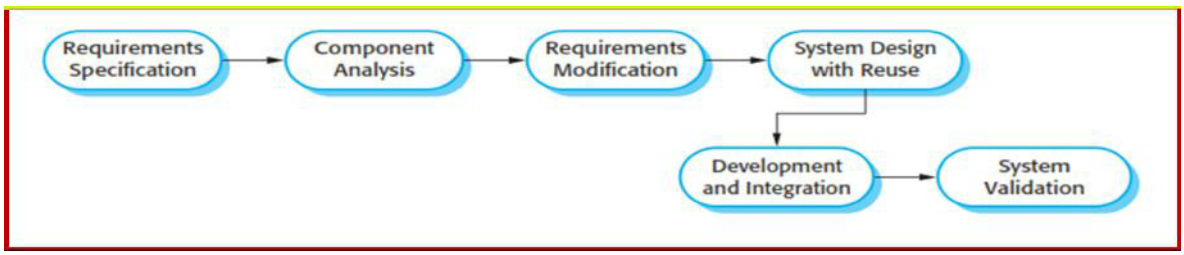
- Management is more complex.
- End of the project may not be known early.
- Not suitable for small or low risk projects and could be expensive for small projects.
- Process is complex
- Spiral may go on indefinitely.

- Large number intermediate stages requires excessive documentation.

Reuse-oriented software engineering

- Based on systematic reuse where systems are integrated from existing components or COTS (Commercial-off-the-shelf) systems.
- Process stages
  - Component analysis;
  - Requirements modification;
  - System design with reuse;
  - Development and integration.

- Reuse is now the standard approach for building many types of business system



Types** of **software component

- Web services that are developed according to service standards and which are available for remote invocation.
- Collections of objects that are developed as a package to be integrated with a component framework such as .NET or J2EE.
- Stand-alone software systems (COTS) that are configured for use in a particular environment.

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

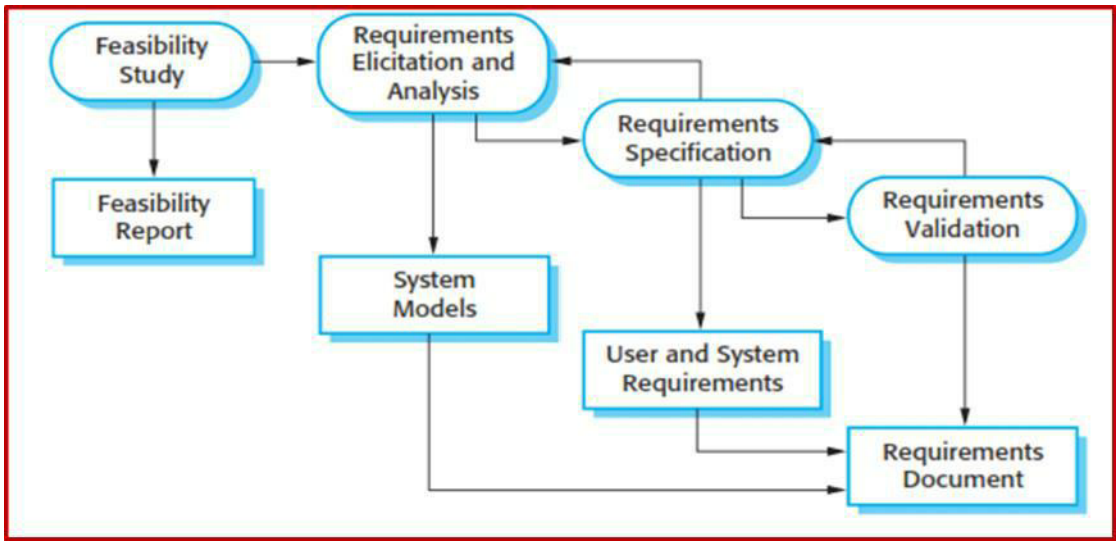
The process of establishing what services are required and the constraints on the system's operation and development.

Requirements engineering process

- **Feasibility study**
  Is it technically and financially feasible to build the system?

- **Requirements elicitation and analysis**
  What do the system stakeholders require or expect from the system?
  - Requirements specification
    - Defining the requirements in detail
  - Requirements validation
    - Checking the validity of the requirements

The requirements engineering process



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.

A general model of the design process

Design activities

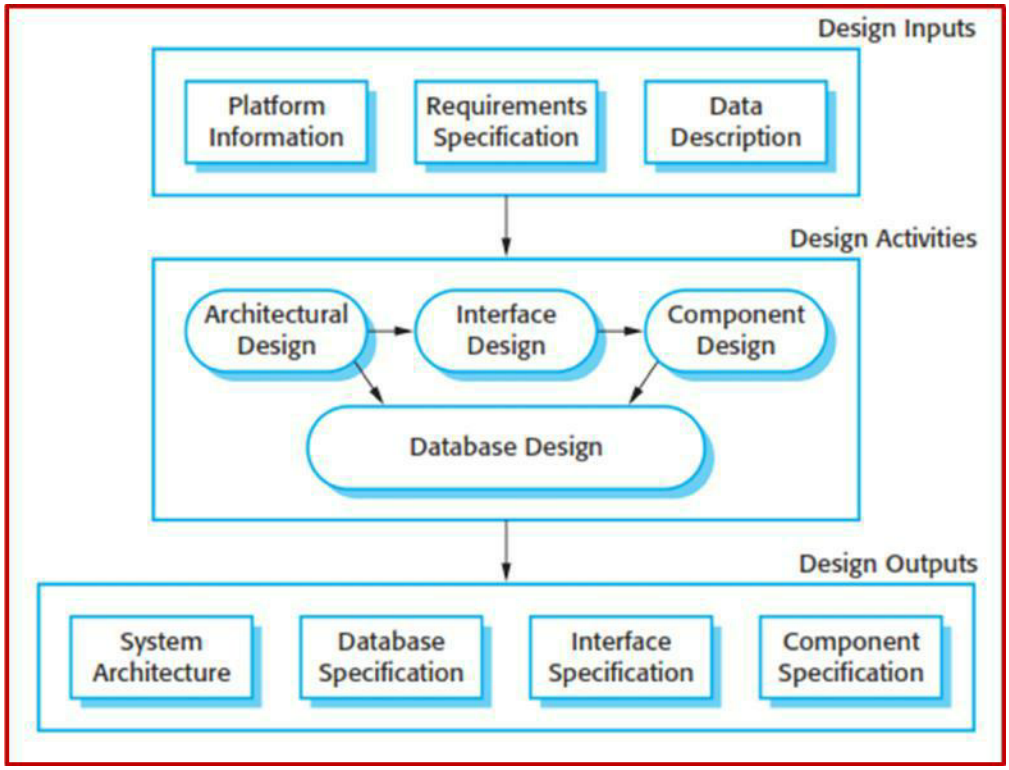
- Architectural design, where you identify the overall structure of the system, the principal components (sometimes called sub-systems or modules), their relationships and how they are distributed.
- Interface design, where you define the interfaces between system components.
- Component design, where 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.

Software validation

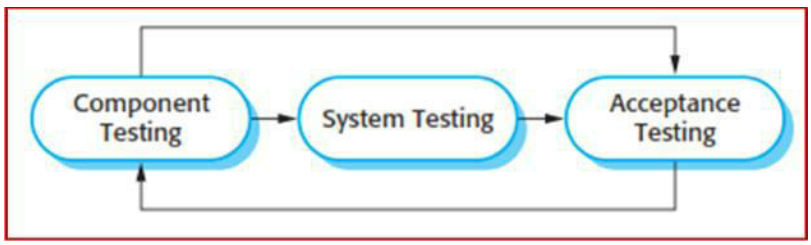
- Verification and validation (V & V) is intended to show that a system conforms to its specification and meets the requirements of the system customer.
- Involves checking and review processes and system testing.
- System testing involves executing the system with test cases that are derived from the

specification of the real data to be processed by the system.

- Testing is the most commonly used V & V activity.



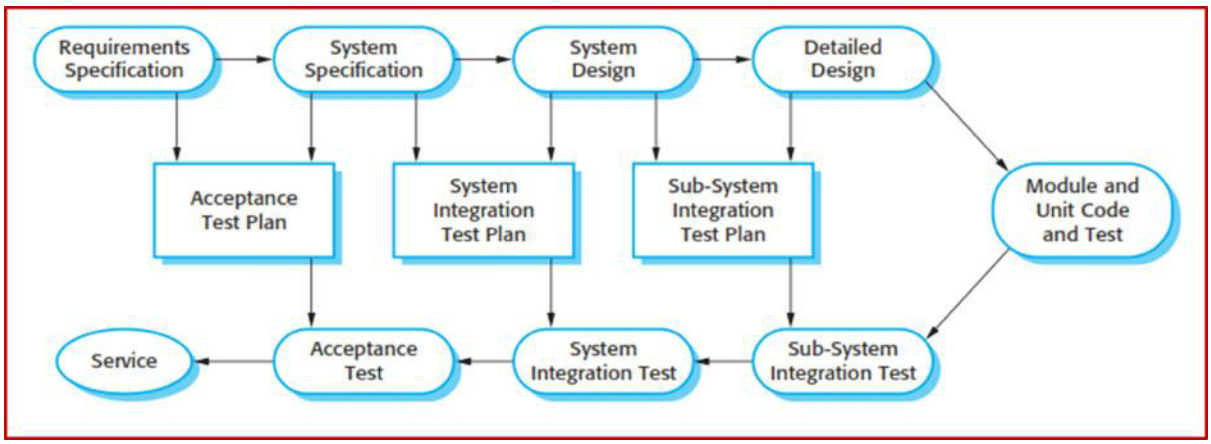
Stages** of **testing



Testing stages

- **Development or component testing** Individual components are tested independently;
  Components may be functions or objects or coherent groupings of these entities.
- **System testing**
  Testing of the system as a whole. Testing of emergent properties is particularly important.
- **Acceptance testing**
  Testing with customer data to check that the system meets the customer's needs.

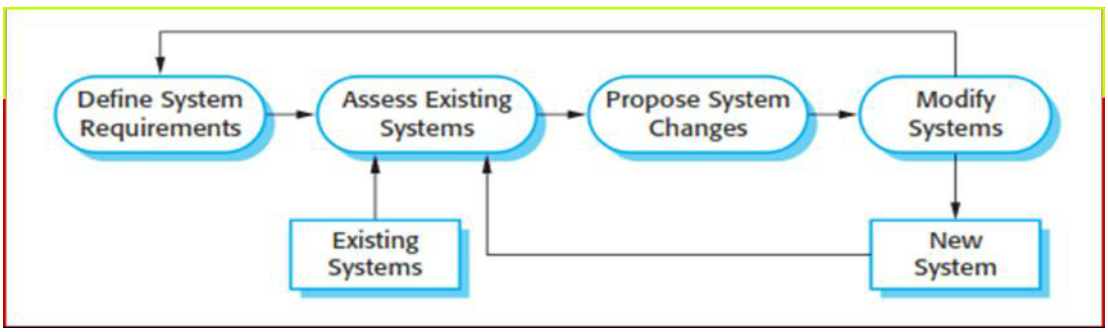
Testing phases in a plan-driven software process



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.

System evolution



Requirements Engineering:

Requirements Engineering

Processes

- 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.

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.

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 ***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. 4 Functional user requirements may be high-level statements of what the system should do. 4 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.

Requirements imprecision

Problems arise when requirements are not precisely stated.

- Ambiguous requirements may be interpreted in different ways by developers and users. 4 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.
  - In practice, it is impossible to produce a complete and consistent requirements document.

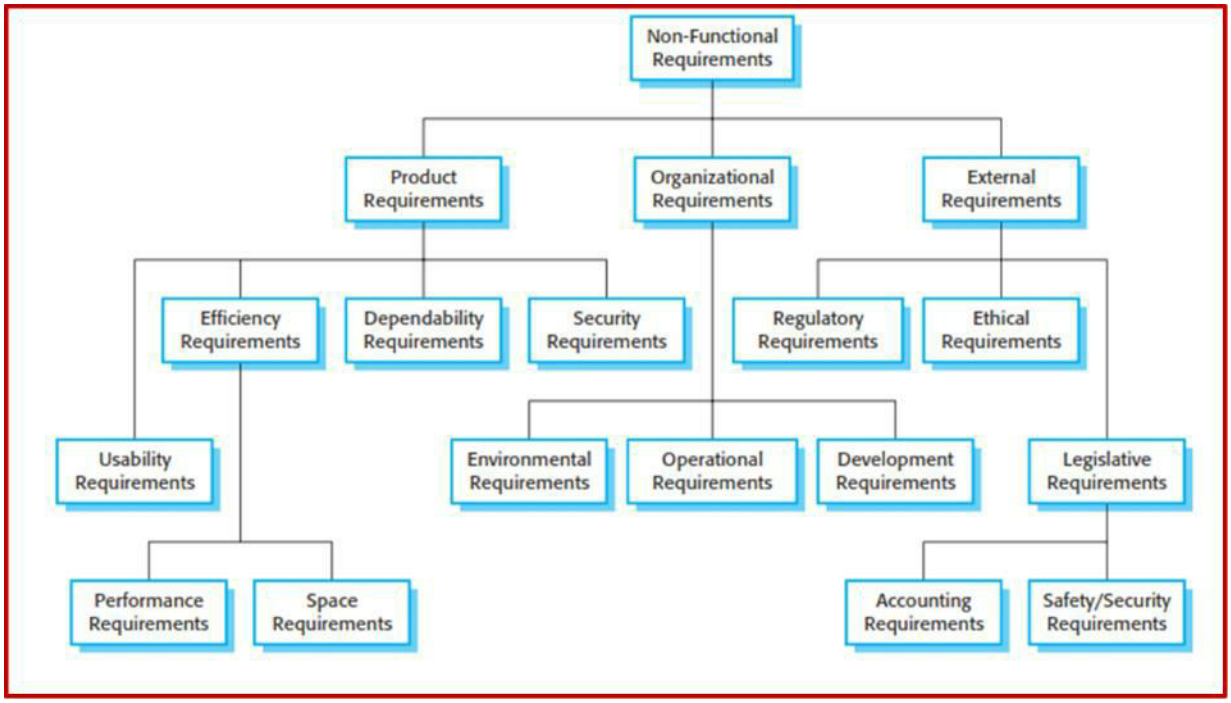
Non functional requirements

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 ME, 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 requirements implementation

- Non-functional requirements may affect the overall architecture of a system rather than the individual components.
- 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.

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.

***Examples** of **nonfunctional requirements in the** MHC-PMS*

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.

Goals and requirements

Non-functional requirements may be very difficult to state precisely and imprecise requirements may be difficult to verify.

- Goal
  - A general intention of the user such as ease of use.
- Verifiable non-functional requirement
  - A statement using some measure that can be objectively tested. Goals are helpful to developers as they convey the intentions of the system users.

Usability requirements

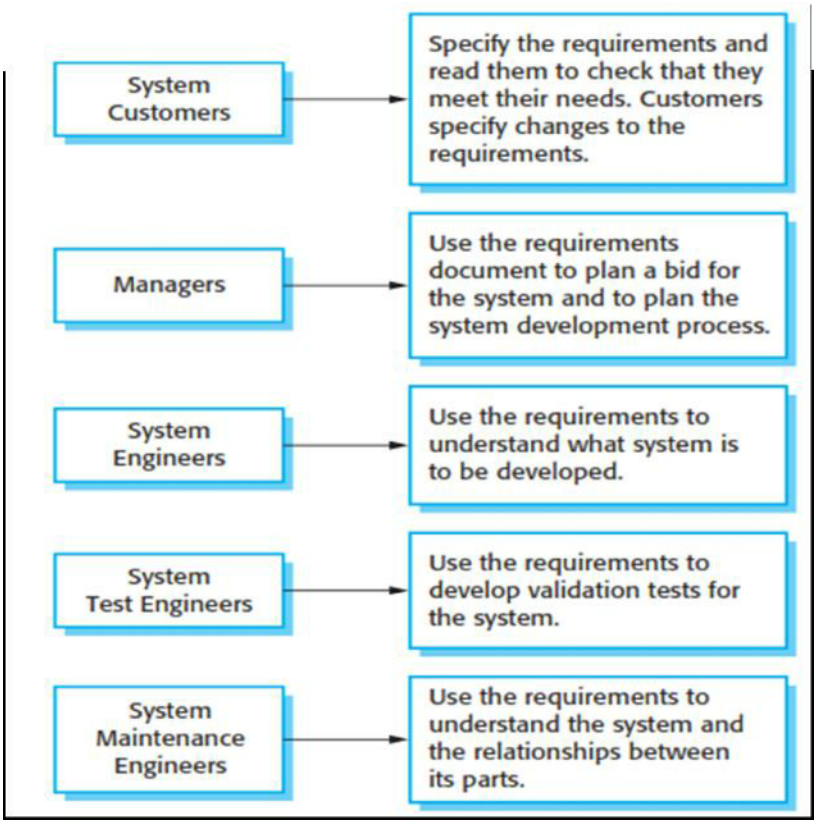
- The system should be easy to use by medical staff and should be organized in such a way that user errors are minimized. (Goal)
- Medical staff shall be able to use all the system functions after four hours of training. After this training, the average number of errors made by experienced users shall 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

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.
- 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.



Requirements document variability

- Information in requirements document depends on type of system and the approach to development used.
- Systems developed incrementally will, typically, have less detail in the requirements document.
- Requirements documents standards have been designed e.g. IEEE standard. These are mostly applicable to the requirements for large systems engineering projects.

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.

Requirements and design

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
  - A system architecture may be designed to structure the requirements;
  - 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.

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

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. 4 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.

Example requirements for the insulin pump software system

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.

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.

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.
- Information about the information needed for the computation and other entities used.
- Description of the action to be taken.
- Pre and post conditions (if appropriate).
- The side effects (if any) of the function.

A structured specification of a requirement for an insulin pump

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.

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 system 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.

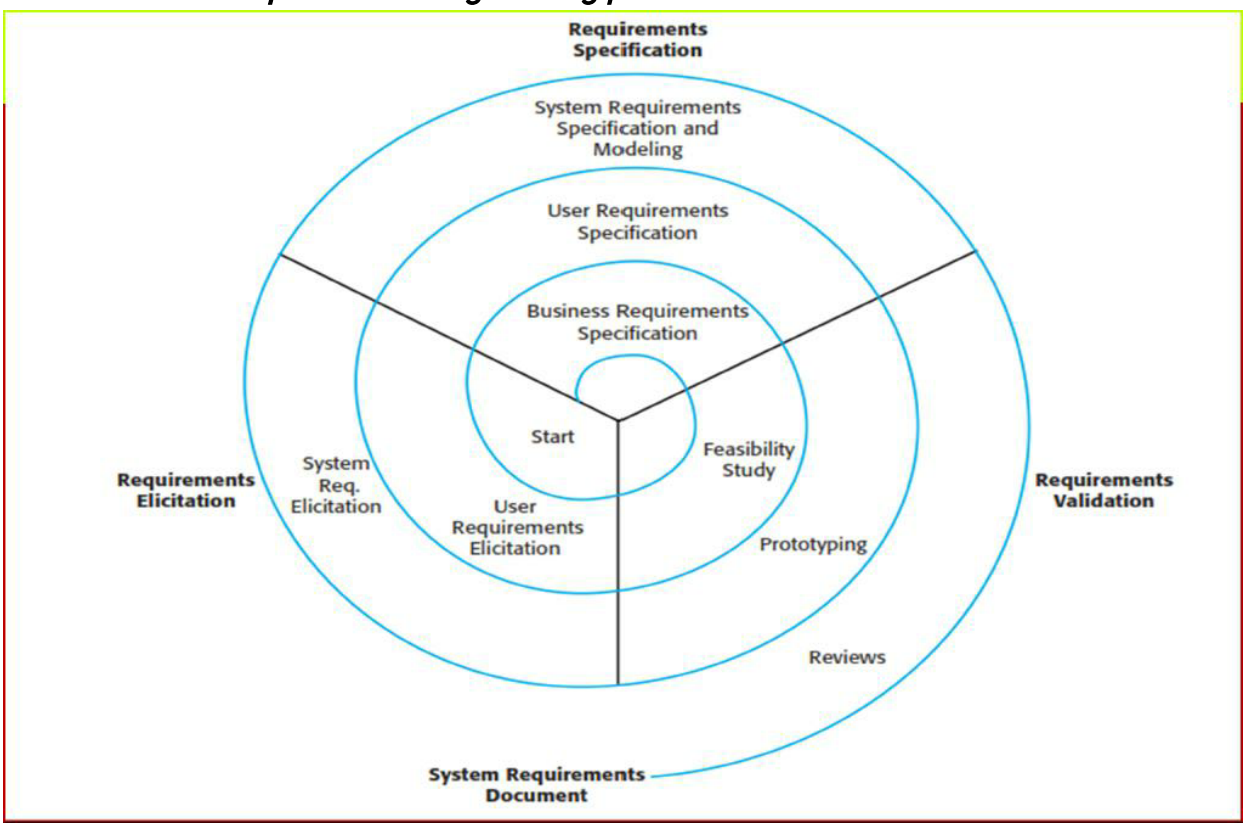
Tabular specification of computation for an insulin pump

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

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



Requirements elicitation and analysis

Sometimes called requirements elicitation or requirements discovery.

- Involves technical staff working with customers to fmd 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.

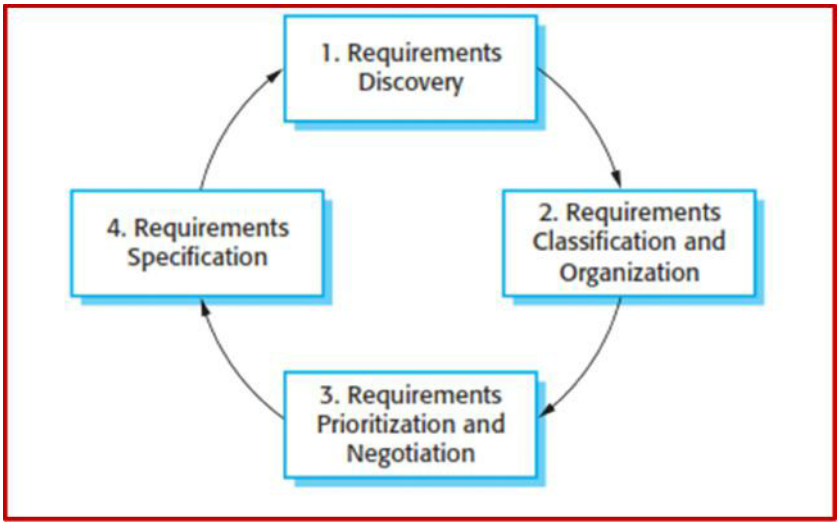
Problems of requirements analysis

- 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,
    - Requirements classification and organization,
    - Requirements prioritization and negotiation,
    - Requirements specification.

The requirements elicitation and analysis process



Process activities

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

  Requirements classification and organisation
  - 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.

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.

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.

Stakeholders in the MHC-PMS

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.
- 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 staff that are responsible for ensuring that system information can be maintained and preserved, and that record keeping procedures have been properly implemented.

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 fmd it hard to articulate or think that it isn't worth articulating.

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 fmishes.

Scenario for collecting medical history in MHC-PMS

INITIAL ASSUMPTION:
The patient has seen a medical receptionist who has created a record in the system and collected the patient's personal information (name, address, age, etc.). A nurse is logged on to the system and is collecting medical history.

NORMAL:
The nurse searches for the patient by family name. If there is more than one patient with the same surname, the given name (first name in English) and date of birth are used to identify the patient.

The nurse chooses the menu option to add medical history.

The nurse then follows a series of prompts from the system to enter information about consultations elsewhere on mental health problems (free text input), existing medical conditions (nurse selects conditions from menu), medication currently taken (selected from menu), allergies (free text), and home life (form).

WHAT CAN GO WRONG:
The patient's record does not exist or cannot be found. The nurse should create a new record and record personal information.

Patient conditions or medication are not entered in the menu. The nurse should choose the 'other' option and enter free text describing the condition/medication.

Patient cannot/will not provide information on medical history. The nurse should enter free text recording the patient's inability/unwillingness to provide information. The system should print the standard exclusion form stating that the lack of information may mean that treatment will be limited or delayed. This should be signed and handed to the patient.

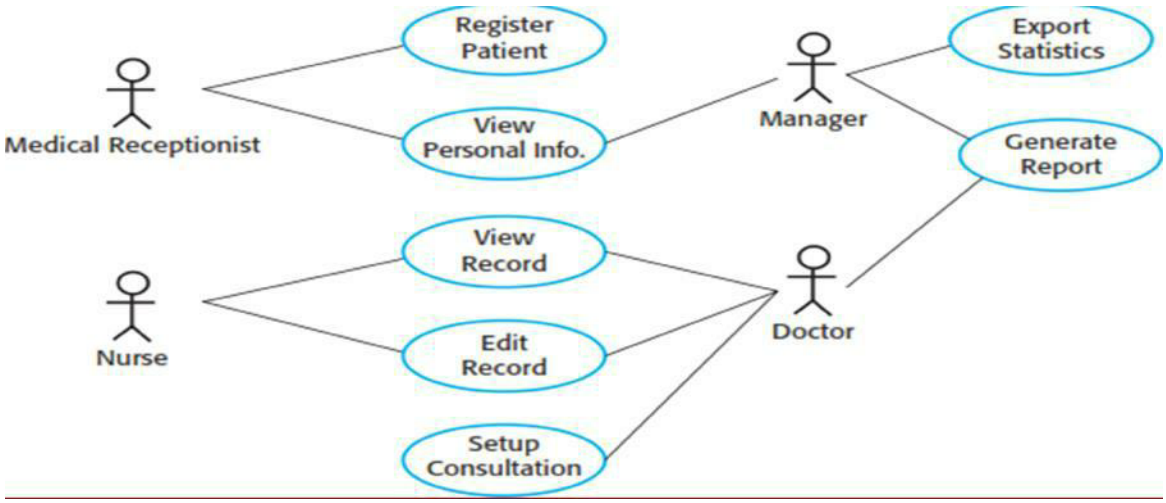
OTHER ACTIVITIES:
Record may be consulted but not edited by other staff while information is being entered.

SYSTEM STATE ON COMPLETION:
User is logged on. The patient record including medical history is entered in the database, a record is added to the system log showing the start and end time of the session and the nurse involved.

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 (see Chapter 5).
- Sequence diagrams may be used to add detail to use-cases by showing the sequence of event processing in the system.



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.

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.

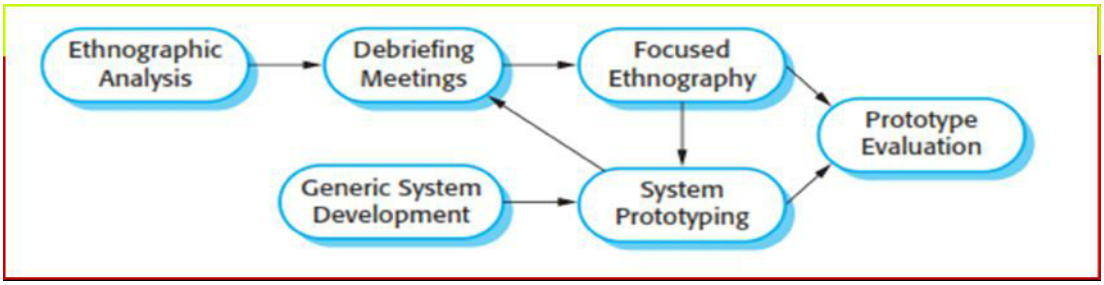
Focused ethnography

Developed in a project studying the air traffic control process

- Combines ethnography with prototyping
- Prototype development results in unanswered questions which focus the ethnographic analysis.

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



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.

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.

- Prototyping
  - Using an executable model of the system to check requirements.
  - Developing tests for requirements to check testability.

Requirements reviews

- Regular reviews should be held while the requirements defmition 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
  - Is the requirement realistically testable?
- Comprehensibility
  - Is the requirement properly understood?
- Traceability
  - Is the origin of the requirement clearly stated?
- Adaptability
  - Can the requirement be changed without a large impact on other requirements?

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.

Changing requirements

The business and technical environment of the system always changes after installation.

- New hardware may be introduced, it may be necessary to interface the system with other systems, business priorities may change (with consequent changes in the system support required), and new legislation and regulations may be introduced that the system must necessarily abide by.

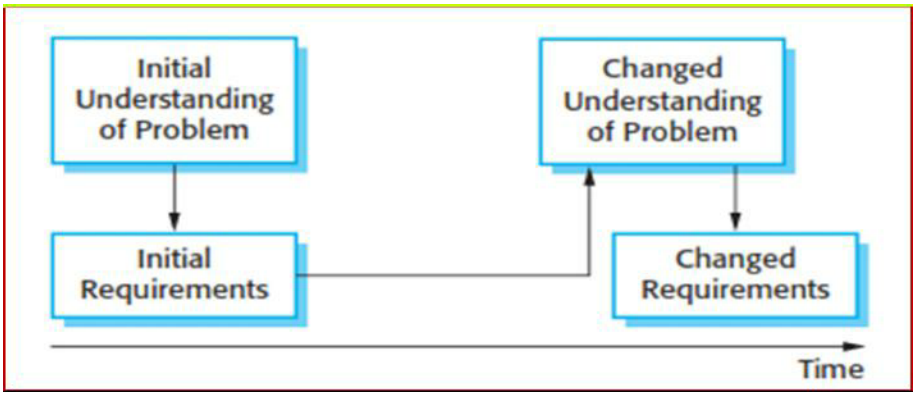
- The people who pay for a system and the users of that system are rarely the same people.
  - System customers impose requirements because of organizational and budgetary constraints. These may conflict with end-user requirements and, after delivery, new features may have to be added for user support 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.

- The final system requirements are inevitably a compromise between them and, with experience, it is often discovered that the balance of support given to different users has to be changed.

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
  - ✓ 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.
- 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.