Software Engineering

INTRODUCTION Module-1

- People in business write spreadsheet programs to simplify their jobs. Scientists and engineers write programs to process their experimental data. Hobbyists write programs for their own interest and enjoyment.
- Software development is professional activity where software is developed for specific business purposes.
- For inclusion in other devices, or as software products such as information systems, CAD systems for use by someone apart from its developer are usually developed by teams rather than individuals. It is maintained and changed throughout its life.

Software Crisis

Software crisis represents various problems that are faced by software developers during the development process.

Various factors that have been contributed to make software crisis are:

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 and more complex systems are required. Systems need 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. We need better software engineering education and training to address this problem

Need for Software engineering

- Economies of developed nations are dependent on software. National infrastructures and utilities are controlled by computer- based system.
- Industrial manufacturing and distribution is completely computerized, as is the financial system. Entertainment, including the music industry, computer games, and film and television, is software intensive. Therefore, software engineering is essential for the functioning of national and international societies.

- Developing an organizational information system is completely different from developing a controller for a scientific instrument. Neither of these systems has much in common with a graphics-intensive computer game. All of these applications need software engineering.

- Software engineering is responsible for producing reliable and trustworthy systems economically and efficiently.

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.

What is software?

Software is not just the programs but also all **associated 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 for users to download recent product information.

There are two fundamental types of software product:

1. ***Generic products*** These are **stand-alone systems** that are produced by a development organization and **sold on the open market to any customer** who is able to buy them. Examples of this type of product include software for PCs such as databases, word processors, drawing packages and project management tools.
2. ***Customized (or bespoke) products*** These are systems which are commissioned by a particular customer. A **software contractor develops the software especially for that customer**. Examples of this type of software include control systems for electronic devices, systems written to support a particular business process and air traffic control systems.

An important difference between these types of software is that, in generic products, the organization that develops the software controls the software specification. For custom products, the specification is usually developed and controlled by the organization that is buying the software.

What is 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: 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 the technical processes of software development software project management and with the **development of tools**, methods and theories to support software production.

Attributes of good software

Software products have a number of attributes that reflect the quality of that software. These attributes are not directly concerned with what the software does rather, they reflect its behavior.

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

Figure: Essential attributes of good software

There are three **general issues / Key Challenges** that affect many different types of software:

1. ***Heterogeneity*** Applicable to

- Distributed systems across networks that include different types of computers and mobile devices.
- All systems running on general-purpose computer

Software may also have to execute on mobile phones. Integrate new software with older legacy systems written in different programming languages.

Develop new techniques for building dependable software that is flexible enough to cope with this heterogeneity

2. ***Business and social change*** Business and society are changing incredibly quickly as emerging economies require new technologies.

They need to change their existing software and to rapidly develop new software.
Many traditional software engineering techniques are time consuming and delivery of new systems often takes longer than planned.

The engineers need to evolve so that the time required for software to deliver value to its customers is reduced.

3. ***Security and trust*** As software is intertwined with all aspects of our lives, it is essential that we can trust that software. This is especially true for remote software systems accessed through a web page or web service interface. We have to make sure that malicious users cannot attack our software and that information security is maintained.

Software Engineering diversity

Software engineering is a systematic approach to the production of software that takes into account practical cost, schedule, and dependability issues, as well as the needs of software customers and producers.
There are many different types of software engineering application including:

1. 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. Examples of such applications are office applications on a PC, CAD programs, photo manipulation software, etc.

2. Interactive transaction-based applications

These are applications that execute on a remote computer and that are accessed by users from their own PCs or terminals. Obviously, these include web applications such as e-commerce applications where you can interact with a remote system to buy goods and services. This class of application also includes business systems, where a business provides access to its systems through a web browser or special-purpose client program and cloud-based services, such as mail and photo sharing. Interactive applications often incorporate a large data store that is accessed and updated in each transaction

3. 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. Examples of embedded systems include the software in a mobile (cell) phone, software that controls anti-lock braking in a car and software in a microwave oven to control the cooking process.

4. 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. Examples of batch systems include periodic billing systems, such as phone billing systems, and salary payment systems.

5. Entertainment systems

These are systems that are primarily for personal use and which are intended to entertain the user. Most of these systems are games of one kind or another. The quality of the user interaction offered is the most important distinguishing characteristic of entertainment systems.

6. System 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. These are often computationally intensive and require high-performance parallel systems for execution.

7. 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. The software has to interact with sensors and often is installed in a hostile environment such as inside an engine or in a remote location.

8. Systems of systems

These are systems that are composed of a number of other software systems. Some of these may be generic software products, such as a spreadsheet program. Other systems in the assembly may be specially written for that environment.

Software engineering and the Web:

The development of the World Wide Web has had a profound effect on our daily lives. Initially, the Web was primarily a universally accessible information store and it had little effect on software systems.

Around 2000, the Web started to evolve and more and more functionality was added to browsers. This meant that web-based systems could be developed where, instead of a special purpose user interface, these systems could be accessed using a web browser.

The next stage in the development of web-based systems was the notion of web services. Web services are software components that deliver specific, useful functionality and which are accessed over the Web. Applications are constructed by integrating these web services, which may be provided by different companies.

In the last few years, the notion of 'software as a service' has been developed. It has been proposed that software will not normally run on local computers but will run on 'computing clouds' that are accessed over the Internet.
This radical change in software organization has, obviously, led to changes in the ways that web- based systems are engineered. For example:

1. Software reuse has become the dominant approach for constructing web-based systems.
2. When building these systems, pre-existing software components and systems are used to create new systems.
3. Web-based systems should be developed and delivered incrementally.
4. User interfaces are constrained by the capabilities of web browsers.

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 behavior 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 of 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).

Ethical principles

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 of their profession and shall promote an ethical approach to the practice of the profession.

Case Studies

To illustrate software engineering concepts, three types of systems are used as case studies:

1. **An embedded system** This is a system where the software controls a hardware device and is embedded in that device. Issues in embedded systems typically include physical size, responsiveness, power management, etc. The example of an embedded system is a software system to control a medical device.
2. **An information system** This is a system whose primary purpose is to manage and provide access to a database of information. Issues in information systems include security, usability, privacy, and maintaining data integrity. The example of an information system that is a medical records system.
3. **A sensor-based data collection system** This is a system whose primary purpose is to collect data from a set of sensors and process that data in some way. The key requirements of such systems are reliability, even in hostile environmental conditions, and maintainability. The example of a data collection system that is a wilderness weather station.

Insulin pump control system

An insulin pump is a medical system that simulates the operation of the pancreas (an internal organ). The software controlling this system is an embedded system, which collects information from a sensor and controls a pump that delivers a controlled dose of insulin to a user.

- ✧ 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.

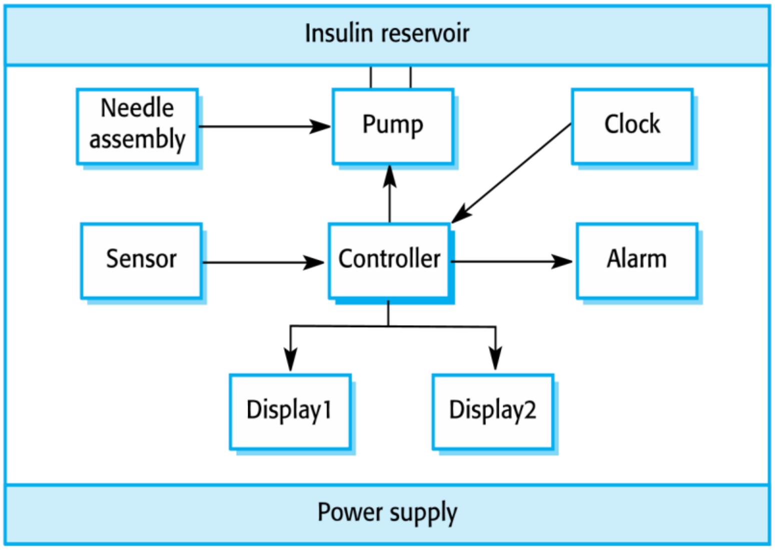


Figure: Insulin Pump Hardware

A software-controlled insulin delivery system might work by using a micro sensor embedded in the patient to measure some blood parameter that is proportional to the sugar level. This is then sent to the pump controller. This controller computes the sugar level and the amount of insulin that is needed. It then sends signals to a miniaturized pump to deliver the insulin via a permanently attached needle.

Figure shows the hardware components and organization of the insulin pump. The blood sensor measures the electrical conductivity of the blood under different conditions and that these values can be related to the blood sugar level. The insulin pump delivers one unit of insulin in response to a single pulse from a controller. Therefore, to deliver 10 units of insulin, the controller sends 10 pulses to the pump.

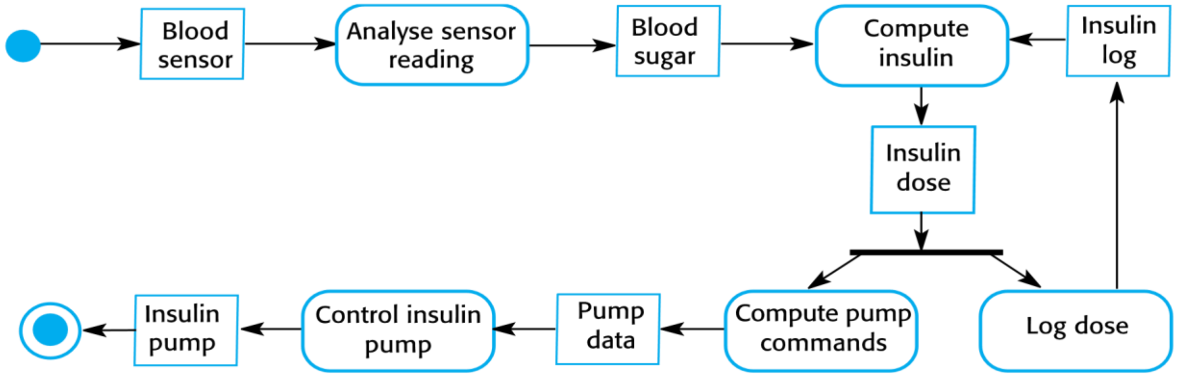


Figure: Activity Model of Insulin Pump

Figures shows, UML activity model that illustrates how the software transforms an input blood sugar level to a sequence of commands that drive the insulin pump Two essential high-level requirements that this must meet are:

- 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.

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.
- Men care 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.

Mentcare 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.

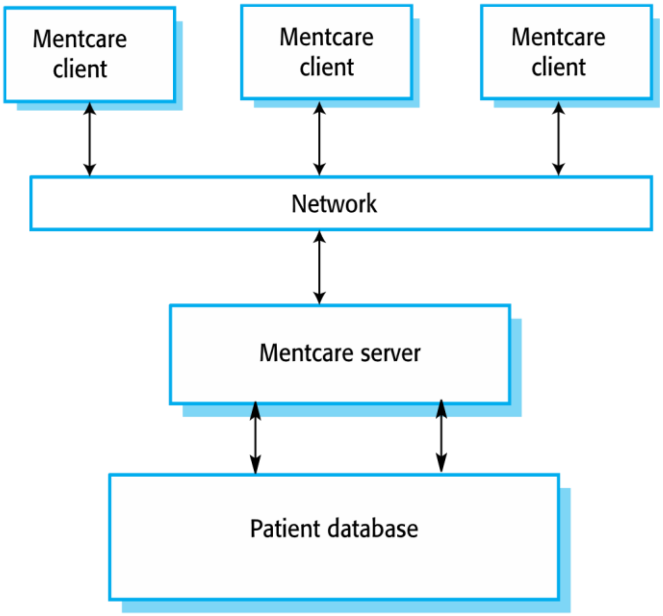


Figure: The Organization of MHC-PMS

Key features of the Mentcare system

- 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.

Mentcare system concerns:

- Privacy
  - It is essential that patient information is confidential and is never disclosed to anyone apart from authorized 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

- To help monitor climate change and to improve the accuracy of weather forecasts in remote areas, 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.

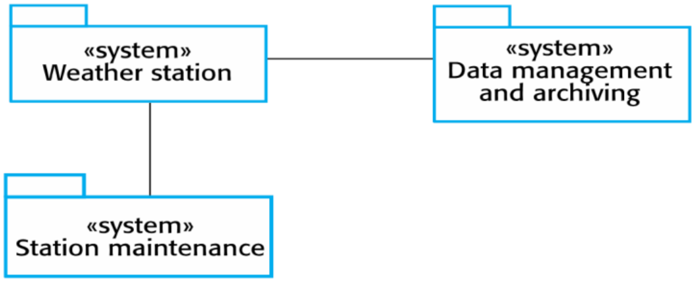


Figure: The weather station's environment

- 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.

This system must provide additional functionalities like:

- 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

A ***software process*** is a set of related activities that leads to the production of a software product. These activities may involve the development of software from scratch in a standard programming language like Java or C.

There are many different software processes but all must include four activities that are fundamental to software engineering:

1. **Software specification** The functionality of the software and constraints on its operation must be defined.
2. **Software design and implementation** The software to meet the specification must be produced.
3. **Software validation** The software must be validated to ensure that it does what the customer wants.
4. **Software evolution** The software must evolve to meet changing customer needs.

Software process models

The Waterfall model

The first published model of the software development process was derived from more general system engineering processes (Royce, 1970).

This takes the fundamental process activities of specification, development, validation, and evolution and represents them as separate process phases such as requirements specification, software design, implementation, testing, and so on.

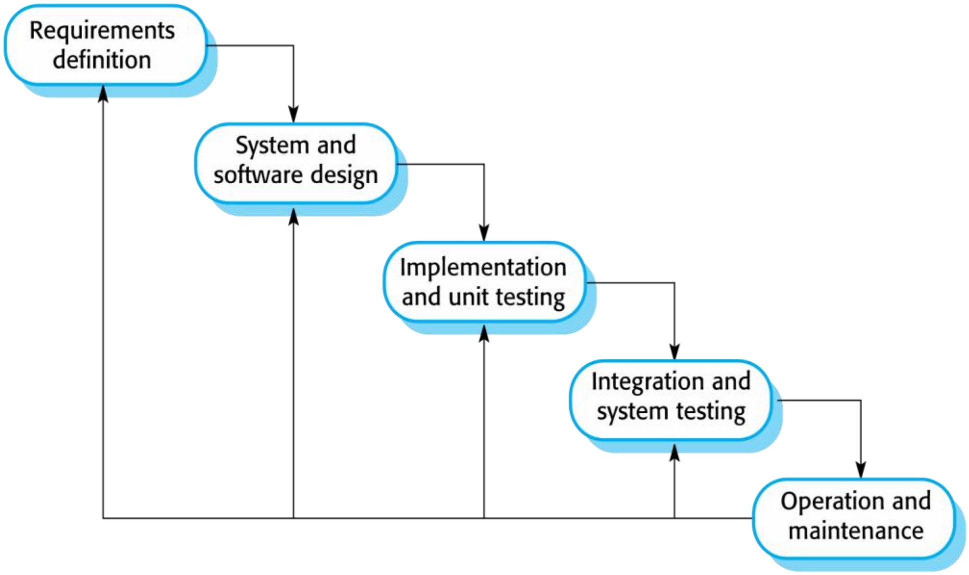

Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

Figure: Waterfall Model

1. ***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.

2. ***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.**

3. ***Implementation and unit testing***

- The software design is realized as a **set of programs or program units**.
- Unit testing involves verifying that each **unit meets its specification.**

4. ***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.**

5. ***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.

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.

Incremental development

Incremental development is based on the idea of developing an initial implementation, exposing this to user comment and evolving it through several versions until an adequate system has been developed.

1. In an incremental development process, identify which of the services are most important and which are least important to them.

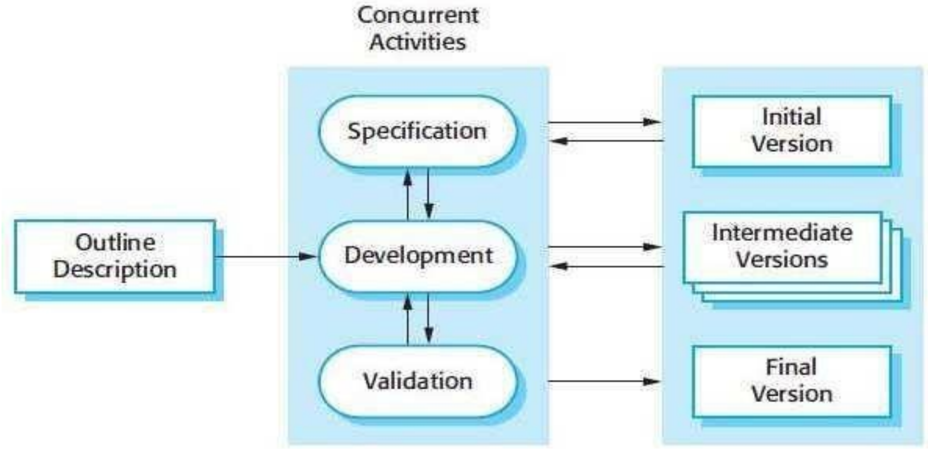
2. A number of delivery increments are then defined, with each increment providing a sub- set of the system functionality.

3. The allocation of services to increments depends on the service priority with the highest priority services delivered first.

4. During development, further requirements analysis for later increments can take place(Evolutionary model) but requirements changes for the current increment are not accepted(Waterfall model).

5. They can experiment with the system that helps them clarify their requirements for later increments and for later versions of the current increment.

6. As new increments are completed, they are integrated with existing increments so that the system functionality improves with each delivered increment.



Advantages:

1. Customers do not have to wait until the entire system is delivered before they can gain value from it. The first increment satisfies their most critical requirements so they can use the software immediately.

2. Customers can use the early increments as prototypes and gain experience that informs their requirements for later system increments.

3. There is a lower risk of overall project failure.
4. As the highest priority services are delivered first, and later increments are integrated with them, it is inevitable that the most important system services receive the most testing.

Disadvantages:

1. Increments should be relatively small and each increment should deliver some system functionality.

2. It can be difficult to map the **customer's requirements** onto increments of the right size.

3. As requirements are not defined in detail until an increment is to be implemented, it can be hard to identify common facilities that are needed by all increments.

Boehm's Spiral Model

A risk-driven software process framework (the spiral model) was proposed by Boehm. Software process is represented as a spiral. Each loop in the spiral represents a phase of the software process. It combines change avoidance with change tolerance. It assumes that changes are a result of project risks and includes explicit risk management activities to reduce these risks.

Each loop is split into four sectors:

1. **Objective setting**

   - To determine specific objectives, alternatives and constraints
   - Constraints on the process and the product are identified
   - A detailed management plan is drawn up
   - Project risks are identified

- Alternative strategies may be planned

2. **Risk assessment and reduction**

- A detailed analysis for each identified project risk is carried out
- Measurements to reduce these risks

3. **Development and validation**

- A development model for the system is chosen
- Throwaway prototyping: user interface risks are dominant
- Development based on formal transformations: safety risks are the main consideration
- Waterfall model: main identified risk is a sub-system integration

4. **Planning**

- Project is reviewed
- Decision: continue with a further loop of the spiral? If so, 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.

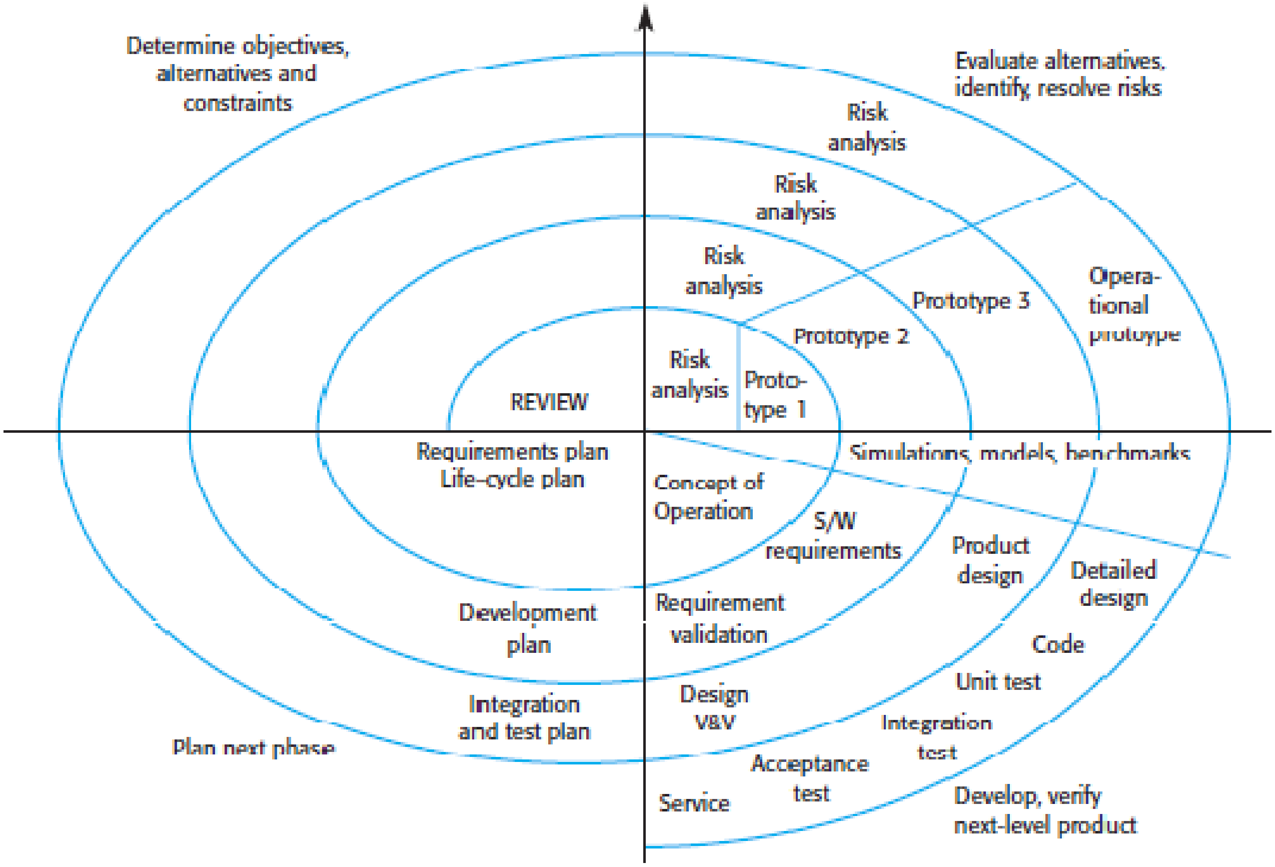


Fig: Boehm's Spiral Model of Software Process

Process activities

1. Software specification (Requirements engineering process)

Definition: the process of understanding and defining what services are required from the system and identifying the constraints on the system's operation and development

Requirements engineering: critical stage of the software process as errors at this stage inevitably lead to later problems in the system design and implementation

i Feasibility study

Estimate whether the identified user needs may be satisfied using current software and hardware technologies. Considerations: cost-effective, development within existing budgetary constraints. Should be relatively cheap and quick

ii Requirements elicitation and analysis

Derive the system requirements through observation of existing systems, discussions with potential users and procurers, task analysis. Development of one or more system models and prototypes possible. Help to understand the system to be specified.

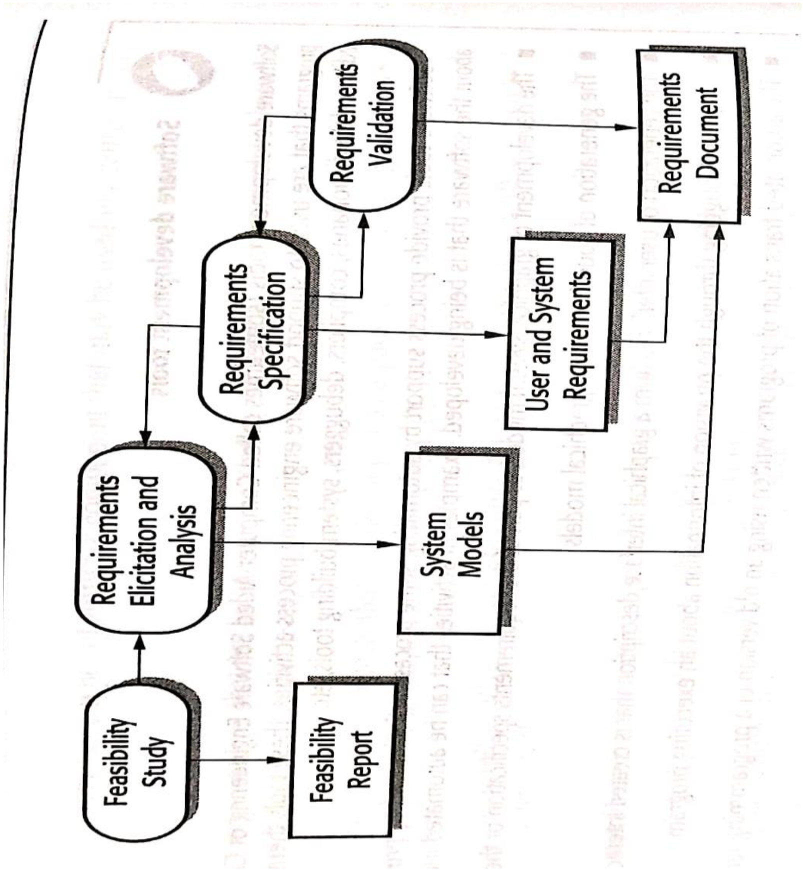
iii Requirements specification

Activity of translating the information gathered during the analysis activity into a document that defines a set of requirements. User requirements: abstract statements of the system requirements for the customer and end-user of the system. System requirements: more detailed description of the functionality to be provided

iv Requirements validation

Checks the requirements for realism, consistency and completeness. Errors are discovered to correct the specification.

Fig: The requirements engineering process



2. 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.

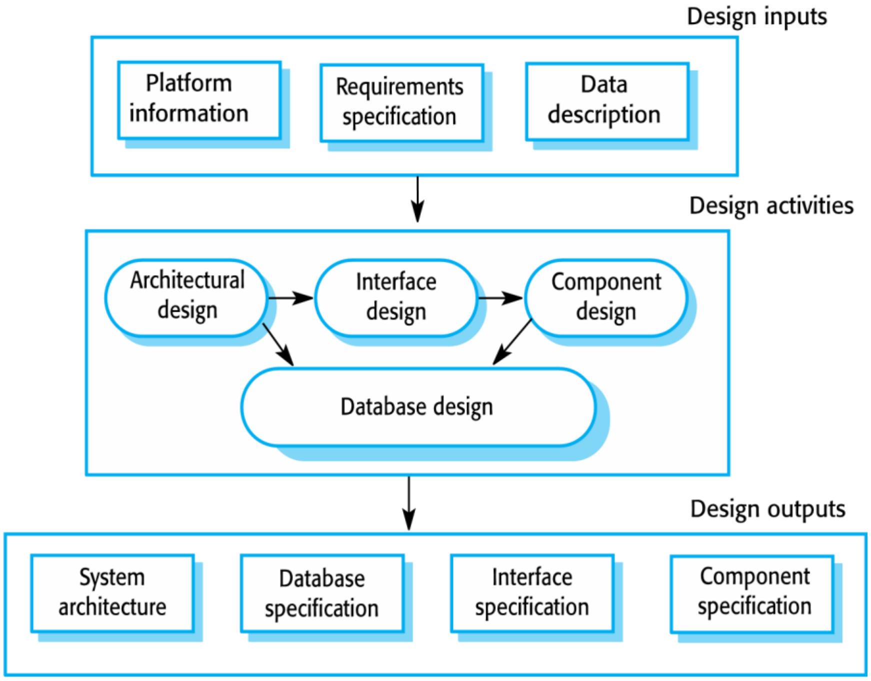


Fig: A general model of the design process

Figure 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.

A **specification for the next stage** is the **output of each design activity**.
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. ***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.

3. ***Component design*** Services are allocated to components and the interfaces of these components are designed.

4. ***Data design*** The data structures used in the system implementation are designed in detail and specified.

3. 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 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.

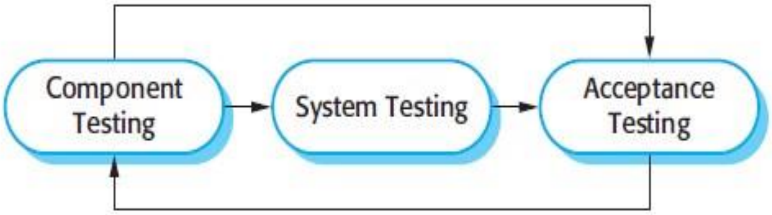


Figure 2.6 Stages of testing

The stages in the testing process are:

1. ***Component (or 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.

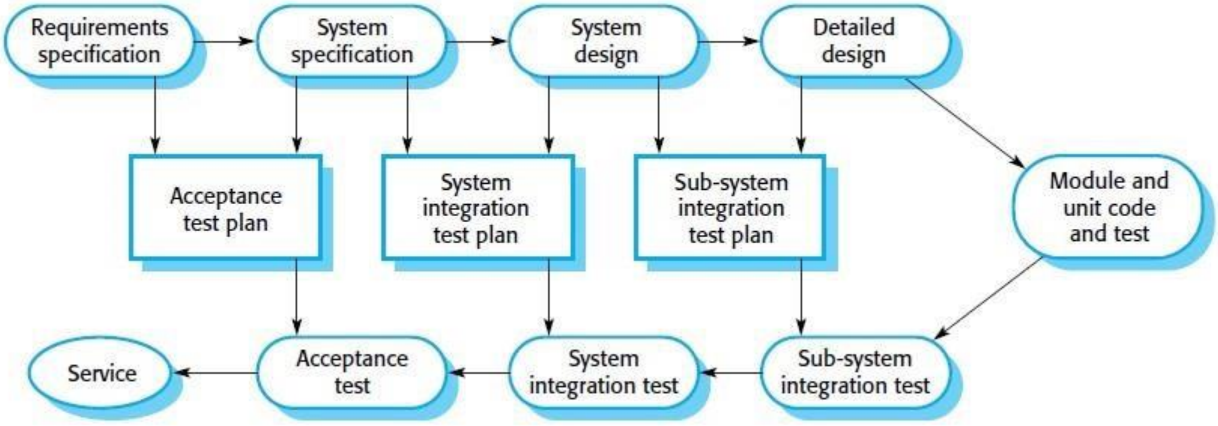

Requirements specification
System specification
System design
Detailed design
Acceptance test plan
System integration test plan
Sub-system integration test plan
Module and unit code and test
Service
Acceptance test
System integration test
Sub-system integration test

Figure: Testing phases in the software process

- **Component development and testing are interleaved**. Programmers make up their own test data and incrementally test the code as it is developed. The programmer knows the component best and is therefore the best person to generate test cases.
- If an incremental approach to development is used, each increment should be tested as it is developed, with these tests based on the requirements for that increment

- Later stages of testing involve integrating work from a number of programmers and must be planned in advance. Figure 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.

4. 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) where software is continually changed over its lifetime in response to changing requirements and customer needs.

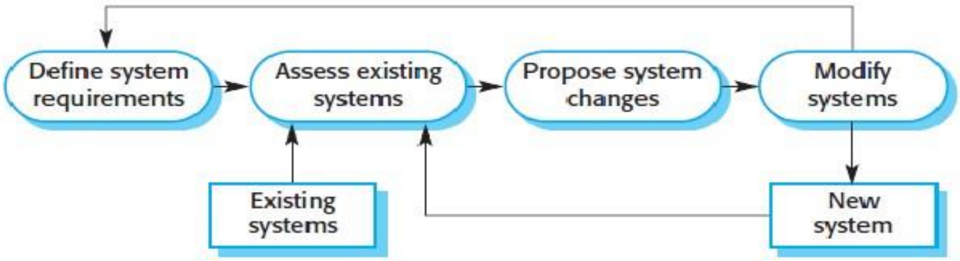


Figure: System evolution

Requirements Engineering

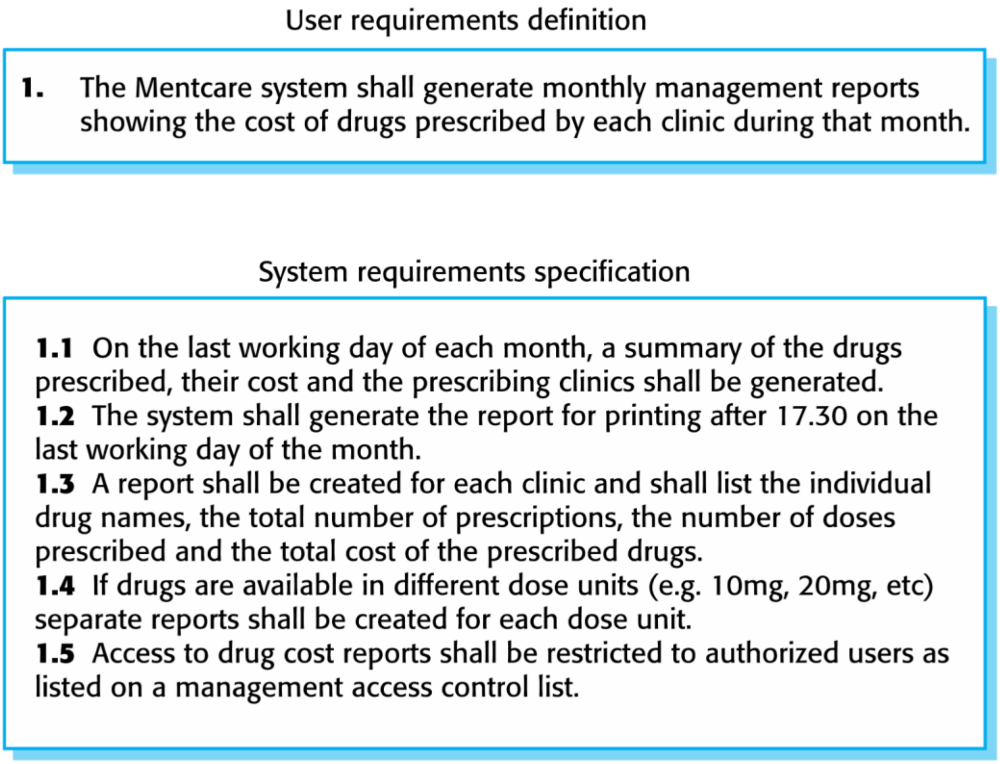
Software requirements

The requirements for a system are the descriptions of the services provided by the system and its operational constraints.

The process of finding out, analyzing, documenting and checking these services and constraints is called *requirements engineering* (RE).

Requirements document for the system have ***user requirements*** to mean the **high-level abstract requirements** and ***system requirements*** to mean the **detailed description** of what the system should do. User requirements and system requirements may be defined as follows:

1. ***User requirements*** are **statements**, in a natural language plus **diagrams**, of what services the system is expected to provide and the constraints under which it must operate.
2. ***System requirements*** set out the **system's functions, services and operational constraints** in detail. The system requirements document (sometimes called a functional specification) should be precise. It should define exactly what is to be implemented. It may be part of the contract between the system buyer and the software developers.



User requirements definition

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

Figure: User and system requirements

Functional and Non-functional Requirements

1. **Functional requirements:**

- Statements of services the system should provide
- How the system should react to particular inputs
- How the system should behave in particular situations
- Optional: what the system should do
- Examples: security (user authentication facilities in the system)
- Vary from general to very specific requirements (what the system should do vs. local ways of working/organization of an existing system)
- May by expressed as user or system requirements or both
- Imprecision's may cause software engineering problems (see ambiguous user requirements)
- Function requirements specification should be complete and consistent (no contradictory definitions)
- Completeness and consistency practically impossible for large complex systems
- Easy to make mistakes and omissions when writing specifications for complex systems
- Many stakeholders in a large system (they often have inconsistent needs)

2. **Non-functional requirements:**

- Constraints on the services or functions offered by the system

- Include constraints concerning timing, development process and standards
- Often apply to the system as a whole (unlike individual features or services)
- Example: security (limiting access to authorized users)
- May relate to emergent system properties: reliability, response time and store occupancy
- May define constraints on the system implementation: I/O device capabilities, data representations used in interfaces
- Specify or constrain characteristics of the system as a whole: performance security, availability
- Often more critical than individual function requirements
- Whole system may be unusable when failing to meet a non-functional requirement
- Often more difficult to relate components to nonfunctional requirements
- Implementation of these requirements may be diffused
- May affect the overall architecture of a system (rather than the individual components)
- May generate a number of related functional requirements that define new system services that are required (e.g. security)
- May also generate requirements that restrict existing requirements
- Arise through user needs: budget constraints, organizational policies, need for interoperability with other software/hardware, external factors (safety regulations, privacy legislation)
- May come from

  Classification of Non functional Requirements :

  - **Product requirements:** specify/constrain the behavior of the software

    Usability requirements
    Efficiency requirements (performance, space)
    Dependability requirements
    Security requirements

  - **Organizational requirements:** broad system req. derived from policies and procedures in the customer's and developer's organization.

    Environmental requirement (operating environment of the system)
    Operational process requirement (how the system will be used)
    Development process requirement (programming language, development environment, process standards)

  - **External requirements:** from factor external to the system and its development process
    Regulatory requirements (what must be done for the system to be used by a regulator)
    Ethical requirements (ensure the system will be acceptable to its user and the general public)
    Legislative requirements (ensure the system operates within the law)

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

Product requirement

The Mentcare system shall be available to all clinics during normal working hours (Mon–Fri, 0830–17.30). Downtime within normal working hours shall not exceed five seconds in any one day.

Organizational requirement

Users of the Mentcare 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.

Metrics for specifying non-functional 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

Synonyms: Software Requirements Specification, SRS

Definition: official statement of what the system developers should implement. It should include user and system requirements.

Diverse set of users:

- System customers who specify the req. and read them to check that they meet their needs
  (they also may make changes)
- Managers use the document to plan a bid for the system plan system development process
- System Engineers responsible for developing the software
- System test engineers to develop validation tests for the system
- System maintenance engineers to understand the system and the relationships between its parts

Structure of a requirements document: (IEEE format):

1. Preface

- Define the expected readership of the document
- Describe its version history
- A rational for the creation of a new version
- A summary of the changes made in each version

2. Introduction

- Describe the need for the system
- Describe the system's functions
- Explain how it will work with other systems
- Describe how the system fits into the overall business or strategic objectives of the organization commissioning the software

3. Glossary

- Define the technical terms used in the document
- Should not make assumptions about the experience or expertise of the reader

4. User requirements definition

- Describe the services provided for the user
- Non-functional system requirements should also be described
- May use natural language, diagrams or other notations that are understandable to customers
- Product and process standards should be specified (if applicable)

5. System architecture

- A high-level overview of the anticipated system architecture
- Showing the distribution of functions across system modules
- Architectural components should be highlighted (that are reused)

6. System requirements specification

- Describe the functional and non-functional requirements in more detail
- Optional: further detail to the non-functional requirements
- Optional: interfaces to other systems

7. System models

- Include graphical system models showing the relationships between the systems components, the system and its environment. Examples: object models, data-flow models, semantic data models

8. System evolution

- Describe the fundamental assumptions on which the system is based
- Any anticipated changes due to hardware evolution, changing user needs etc.
- Useful for system designers: may help to avoid design decisions

9. Appendices

- Provide detailed, specific information that is related to the application being developed
- Example: hardware and database descriptions
- Hardware requirements: minimal and optional configurations for the system
- Database requirements: logical organization of the data used by the system and the relationships between data

10. Index

- A normal alphabetic index
- An index of diagrams
- An index of functions etc.

Requirements Specification

- Should be clear, unambiguous, easy to understand, complete, consistent
- Use of natural language, simple tables, forms and intuitive diagrams
- A complete and detailed specification of the whole system

How to write a System Requirement Specification:

- Natural language
- Structured natural language
- Design description languages
- Graphical notations
- Mathematical specifications

Natural Language Specification

Guidelines:

- Is expressive, intuitive and universal
- Use a standard format
- Use language consistently to distinguish between mandatory and desirable requirements
- Use text highlighting
- Do not assume that readers understand technical software engineering language
- Try to associate a rationale with each user requirement

Structured specifications:

- A way of writing system requirements where the freedom of the requirements writer is limited and all requirements are written in a standard way
- Maintains most of the expressiveness and understandability of natural language
- Ensures that some uniformity is imposed on the specification
- Define one or more standard templates for requirements and represent these templates as structured forms

Information in a standard form for specifying functional requirements:

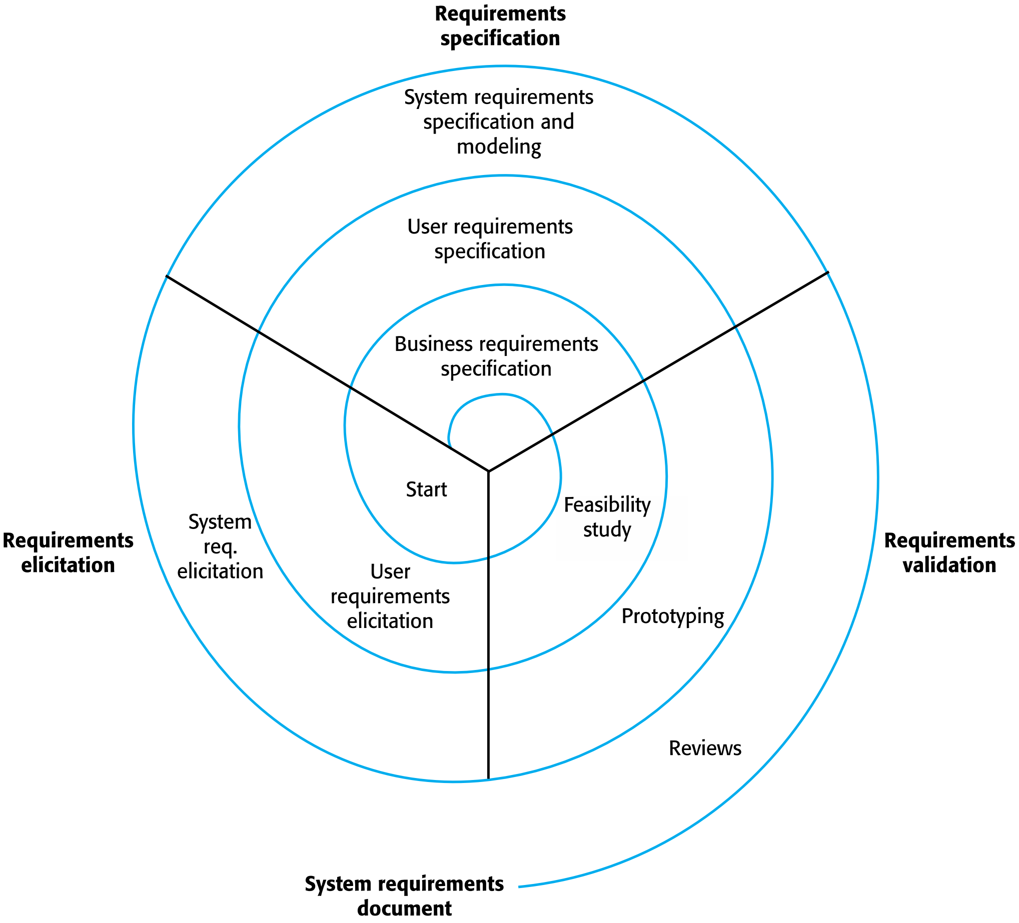
- Description of the function or entity being specified
- Inputs and where these come from
- Outputs and where these go to
- Information about the information that is needed for the computation or other entities in the system that are used
- Pre- and post-condition before and after the function is called
- Side effects of the operation

Requirements Engineering Processes

1. High-level activities:

- Feasibility study
- Elicitation and analysis
- Specification
- Validation

2. Characteristics of a spiral view: A spiral view of the requirements engineering process



Requirements specification
System requirements specification and modeling
User requirements specification
Business requirements specification
Start
Feasibility study
Requirements elicitation
System req. elicitation
User requirements elicitation
Prototyping
Requirements validation
Reviews
System requirements document

- Activities are organized as an iterative process around a spiral
- Output is the system requirements document
- Amount of time and effort for each activity depends on current stage and type of system to be developed

Requirements engineering: the process of applying a structured analysis method (e.g. object oriented analysis).

- Involves analyzing the system and developing a set of graphical system models (case models for system specification)

- Set of models describes the behavior of the system (annotated with additional information; e.g. system's required performance or reliability)

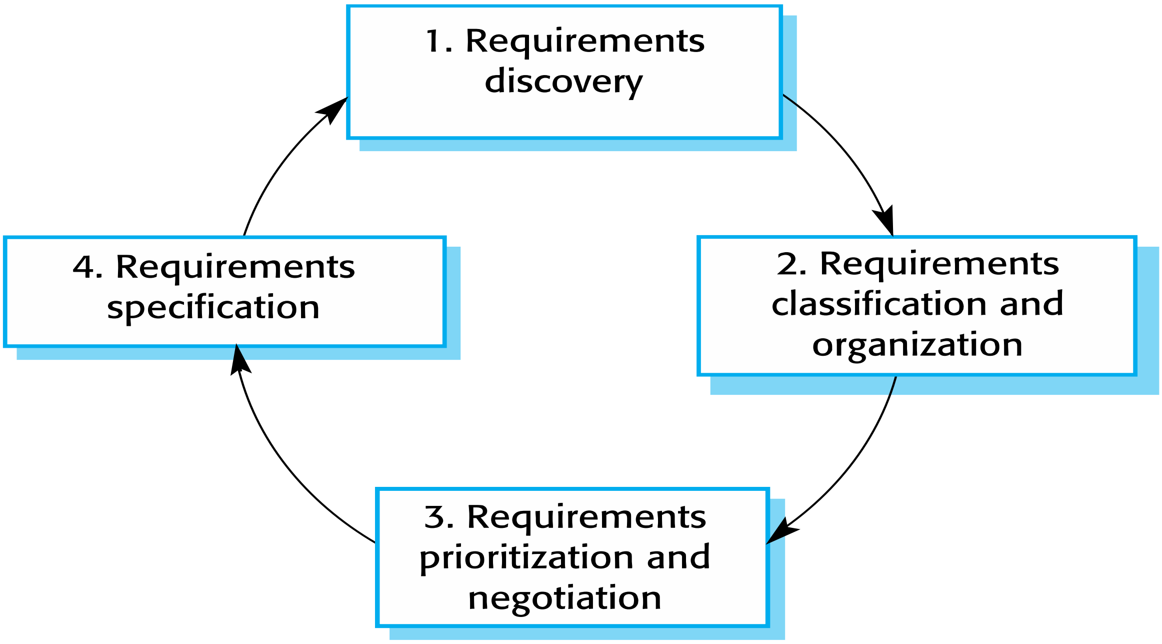
Requirements elicitation: a human-centered activity (unlike constraints by rigid system models)

Requirements management: process of managing changing requirements, e.g. modifications to the system's hardware, software or organizational environment.

Requirements Elicitation and Analysis :

- Software engineers work with customers and system end-users to find out about the application domain
- what services the system should provide
- the required performance of the system hardware constraints

The requirements elicitation and analysis process:


1. Requirements discovery
2. Requirements classification and organization
3. Requirements prioritization and negotiation
4. Requirements specification

1. Requirements discovery

- Interact with stakeholders of the system to discover their requirements (domain requirements and documentation)
- Several complementary techniques available

2. Requirements classification and organization

- Takes unstructured collection of requirements
- Groups related requirements
- Organizes them into coherent clusters
- Most common way: model of the system architecture to identify sub-systems and to associate requirements with ach sub-system
- Practice: requirements engineering and architectural design hard to separate

3. Requirements prioritization and negotiation

- Conflict of requirements with several stakeholders
- Prioritize requirements
- Find and resolve requirements conflicts through negotiation
- Compromises inevitable

4. Requirements specification

- Document requirements (formal or informal)

Eliciting and understanding requirements from stakeholders is difficult:

- Difficult to articulate
- Possibly unrealistic demands (stakeholders may don’t know what is and isn’t feasible)
- Stakeholders might not know what they want from a computer system (expect in most
- general terms)
- Requirements engineers may not understand the naturally expressed requirements of
- stakeholders (in their own terms and with implicit knowledge)
- Different stakeholders have different requirements
- Political factors
- Dynamic economic and business environment (where the analysis takes place)

Requirements discovery:

- Synonym: requirements elicitation
- The process of gathering information about the required system and existing systems
- Distilling the user and system requirements from this information
- Sources of information: documentation, system stakeholders, specifications of similar systems (plus interviews, observations)

Interviewing

- Formal and informal interviews
- Closed interviews: stakeholder answers a pre-defined set of questions
- Open interviews
- No pre-defined agenda
- Requirements engineering team explores a range of issues with system stakeholders
- Develop a better understanding of their needs
- Practice: mixture of open and closed interviews
- Part of most requirements engineering processes

- Questions to stakeholders about the system that they currently use and the system to be developed
- Difficult to elicit domain knowledge through interviews (possibly)
- Impossible to discuss domain requirements without using terminology. Inadequate knowledge leads to misunderstandings
- ‘basis’/fundamental (but professional) knowledge might be difficult to explain to nonprofessionals (in terms of software engineering or computer science)
- Ineffective technique for eliciting knowledge about organizational requirements and Constraints
-

  Characteristics of effective interviews:

  Open-minded: avoid pre-conceived ideas about the requirements (surprising requirements)

  Prompt the interviewee to get discussions (using a springboard questions, requirements proposal, b working together on a prototype system)

  Should be used with other requirements elicitation techniques (not to miss essential information)

Scenarios

- Often easier to relate to real-life examples rather than abstract descriptions
- Useful for adding detail to an outline requirements description
- Covers one or a small number of possible interactions
- Starts with an outline of the interaction May include

  What the system and users expect when the scenario start
  Normal flow of events in the scenario
  What can go wrong and how this is handled
  Information about other activities going on at the same time

- System state when the scenario finishes
- Involves working with stakeholders to identify scenarios and to capture details to be included in these scenarios
- May be written as texts, supplemented by diagrams, screen shots etc.

Use cases:

- A requirements discovery technique that were first introduced in the Object method
- A fundamental feature of the unified modeling language
- Identifies the actors involved in an interaction and names the type of interaction (supplemented by additional information describing the interaction with the system)
- Documented using a high-level use case diagram
- No hard and fast distinction between scenarios and use cases
- Identify the individual interactions between the system and its users or other systems

- Should be documented with a textual description (may be linked to other models in the
UML)

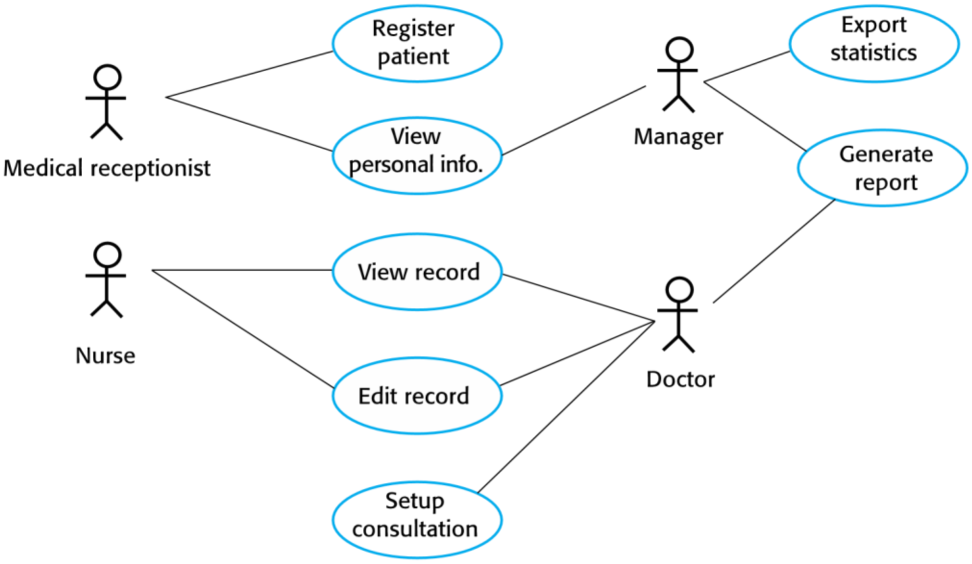
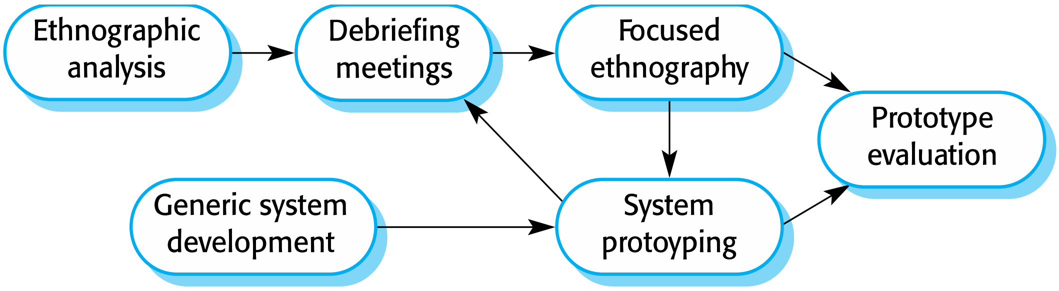


Fig: Use cases for the Mentcare system

- Effective technique (along with scenarios) for eliciting requirements form stakeholders who interact directly with the system
- UML is a de facto standard for object-oriented modeling

Ethnography:

- Software systems are used in social and organizational contexts (may constrain or influence software system requirements)
- Successful systems: satisfy social and organizational requirements
- Definition: an observational technique that can be used to understand operational processes and help derive support requirements for these processes
- Effective for discovering 2 types of requirements:
- Requirements that are derived from the way in which people actually work (rather than the way in which process definitions say they ought to work)
- Requirements that are derived from cooperation and awareness of other people’s activities
- Can be combined with prototyping
- Ethnographic studies can reveal critical process details that are often missed by other requirements elicitation techniques)



Requirements Validation

- The process of checking that requirements actually define the system that the customer really wants
- Overlaps with analysis: it is concerned with finding problems with the requirements
- Important because errors in a requirements document can lead to extensive rework costs
- Process should not be underestimated

Checks during the validation process:

1. Validity checks

- Validity check is a verification performed, either through software or manually, to verify that no errors are present or that it adheres to a standard.

2. Consistency checks

- Requirements in the document should not conflict

3. Completeness checks

- Requirements that define all functions and constraints intended by the system user

4. Realism checks

- Check to ensure that requirements can actually be implemented

5. Verifiability

- System requirements should always be written so that they are verifiable
- Reduces the potential for dispute between customer and contract

Requirements techniques to be used individually or in conjunction:

1. Requirements reviews

- Systematic analysis by a team of requires to check for errors and inconsistencies

2. Prototyping

- Demonstration of an executable model of the system
- Personal experimentation to check if it meets their real needs

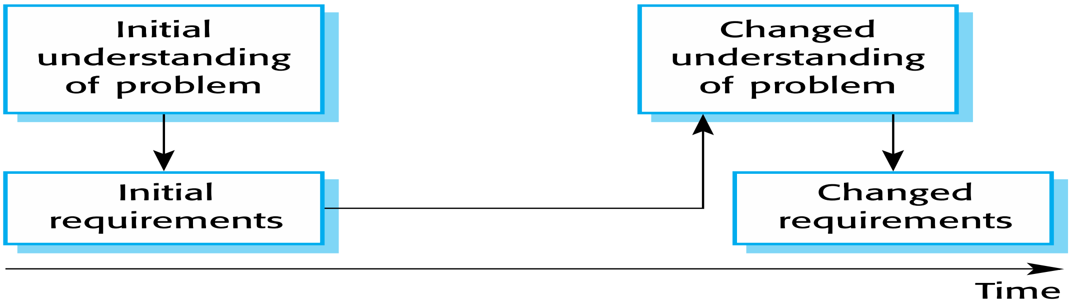
3. Test-case generation

- Requirements should be testable
- Difficult or impossible design often means difficult implementation

Requirements Management

Definition: the process of understanding and controlling changes to system requirements

- Requirements for large systems are always changing. Reasons:
  These systems are usually developed to address 'wicked' problems (software requirements are bound to be incomplete)
- Keep track of individual requirements
- Maintain links between dependent requirements
- Establish a formal process for making change proposals and linking these to system requirements
- Start planning how to manage changing requirements during the requirements elicitation Process


Initial understanding of problem
Initial requirements
Changed understanding of problem
Changed requirements
Time

Reasons why change is inevitable:

- Business and technical environment of the system always changes after installation
- People how pay for a system and the users of that system are rarely the same people
- Large systems usually have a diverse user community
- ○ Many users have different requirements and priorities that may be conflicting or contradictory
- ○ Final system requirements: a compromise

Requirements management planning:

- Establishes the level of requirements management detail that is required and Required decisions to be made
  - ○ Requirements identification: each requirements must be uniquely identified
  - ○ A change management process: set of activities that assess the impact and cost of changes
  - ○ Traceability policies: define the relationships between each requirement and between the requirements and the system design that should be recorded
  - ○ Tool support: process large amounts of information about the requirements (specialist requirements management systems, spreadsheets, simple database systems)

•Required tool support for

- Requirements storage: requirements should be maintained in a secure, managed data store that is accessible to everyone involved in the requirements engineering process
- Change management: process of change management is simplified if active tool support is available
- Traceability management: some tools help discover possible relationships between requirements

Requirements change management:

- Should be applied to all proposed changes to a system's requirements (after they have been approved)
- Essential because Need to decide if the benefits of implementing new requirements are justified by the costs of implementation

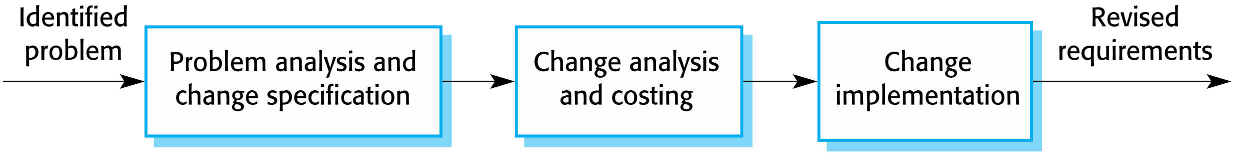
- Advantage of a formal change management

  Change proposals are treated consistently

  Changes to the requirements document are made in a controlled way

Principal stages to a change management process:

- Problem analysis and change specification
- Change analysis and costing
- Change implementation


Identified problem
Problem analysis and change specification
Change analysis and costing
Change implementation
Revised requirements