

VBCA-Software Engineering

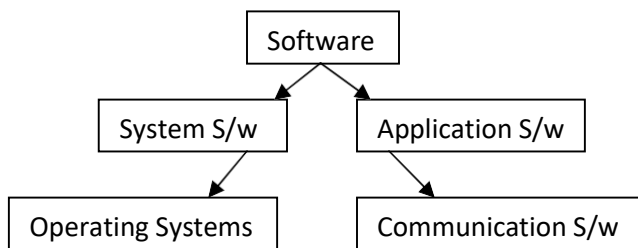
Unit I

Chapter 1

Introduction to Software Engineering

Definition: Software – A Software is a set of instructions or computer programs instructing a computer to do specific tasks. The theory of software was first proposed by Allen Turing in 1935. Software products may be developed for a particular customer or may be developed for a general market.

Classification of Software



System Software – Set of programs to control and manage the operations of computer hardware. Example: Operating Systems, device drivers (printers) etc.

Application Software – It is created to perform specific tasks for a user. Example: MS Word, Photoshop etc

Characteristics of Software

1. Software is engineered not manufactured.
2. Software is intangible – It has no mass, no volume, no color, no odor and it cannot be touched.
3. Software does not wear out, failed components must be re-engineered.
4. Maintainability – Software should meet the changing needs of customers.
5. Usability – The software should have appropriate user interface and enough number of documentations.
6. Dependability – Means reliability, it should be safe and secure to use the software without causing system failure.

What is Software Engineering?

A systematic approach to the development, operation, maintenance and retirement of software.

Definition by IEEE (Institute of Electrical and Electronics Engineers): The application of systematic, disciplined, quantifiable approach to the development, operation, maintenance of software that is the application of engineering to the software.

Goals of Software Engineering

- To improve the quality of the software product
- To increase productivity and
- To give job satisfaction to the Software engineers.

Key Challenges of Software Engineering

- i. The legacy challenge: The challenging method of maintaining and updating the software in such a way that high costs are avoided and essential business services continue to be delivered.
- ii. The Heterogeneity challenge: Systems are required to operate as distributed systems across networks. The challenge of developing techniques to build dependable software which is flexible to cope with is called heterogeneity.

- iii. The delivery challenge: A challenge of shortening delivery time for large and complex systems without compromising system quality.

Software Product

Software products are software systems that are delivered to a customer with documentation which describes how to install and use this system.

Software products may be classified into two types.

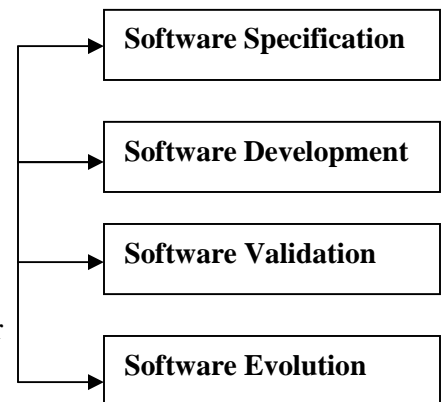
Generic Products	Bespoke (Customized) Products
These products are developed in organizations and sold in the open market to any customer who is able to buy them.	These products are developed for a single customer according to their specification.
These are stand alone systems; the target is generally how many copies are being sold.	These are customized products
These products are developed for anonymous Customers	These products are designed as per customer's requirement by software contractor.
Example: Operating Systems, Word Processors, drawing packages etc	Example : Payroll system, Inventory system, Air traffic control system etc

Software Process

The process involves translating user needs into software requirements, transforming software requirements into design, implementing design in code, testing the code and installing then checking out the software for operations.

There are 4 fundamental process activities

- Software Specification: Functions of the software and its constraints will be defined.
- Software development: The software to meet the specification must be produced.
- Software validation: The software must be validated to ensure what customer wants.
- Software Evolution: The software must evolve to meet changing customer needs.



Characteristics of Software Process

Reliability- To what extents can the process errors be avoided.

Supportability- To what extents can the process activities be supported.

Understandability – To what extent is the process easy to understand.

Maintainability- To what extent can the process help in reflecting organizational requirements.

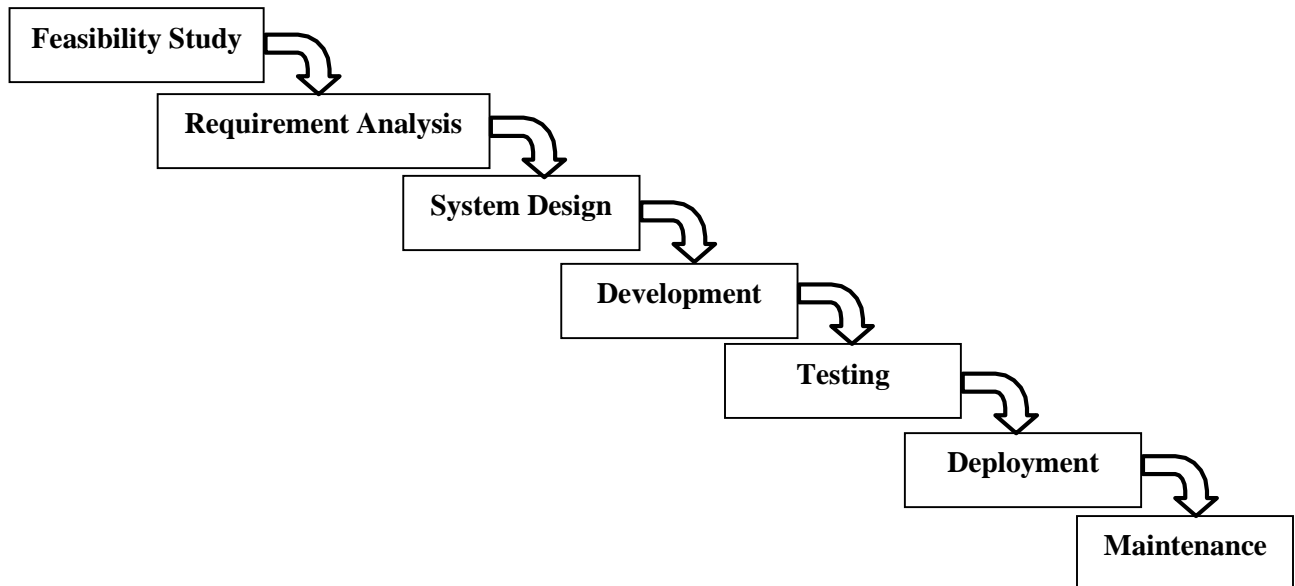
Rapidity- How fast can a process complete the task and deliver a system

SDLC (Software Development Life Cycle)

SDLC is a sequence of activities carried out by analyst, designer and user to develop and implement an information system. These activities can be carried out in different stages.

SDLC can be broadly classified into 7 phases.

1. Feasibility Study: The main aim is to determine whether the product is financially worthwhile and technically feasible.
2. Requirement analysis: In this phase the aim is to find exact requirement of the customers, Requirements are classified into a) Functional Requirements (Input /Output needs) and b) Non-functional requirements (Constraints like time, cost etc). Finally a SRS document is prepared as an output.

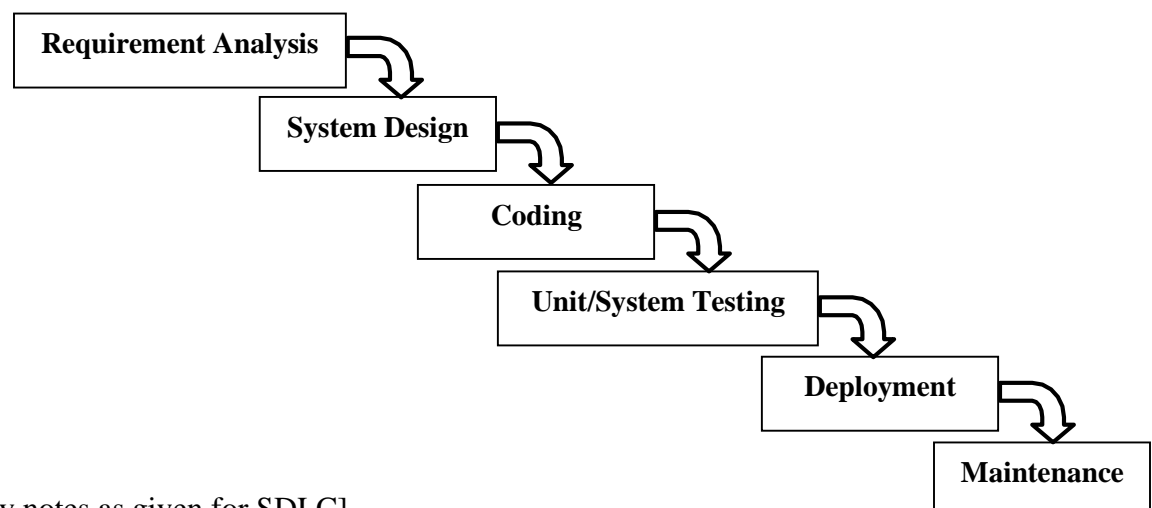


3. System Design: Software architecture is derived from SRS document. A new system is designed according to the needs of the user.
4. Development: This is the actual phase where the system is developed. The whole design is built and implemented.
5. Testing: During implementation phase each module of the design is coded and each module is unit tested individually. This is to check if each individual module works correctly. This is the most critical phase.
6. Deployment: The developed system is handed over to the client. The old system is dispensed and the new system is put to operations and used.
7. Software Maintenance: In this phase adding enhancements, improvements and updates to the new versions are done.

Different types of SDLC Models are: 1) Waterfall Model, 2) Prototype model, 3) Prototyping model, 4) Iterative enhancement model

Waterfall Model

This model is a software life cycle where the stages are depicted as cascading from one to another. It was described by W.W. Royce in 1970. As the figure implies one development stage should be completed before the next begins.



[Follow the same theory notes as given for SDLC]

Advantages of Waterfall Model

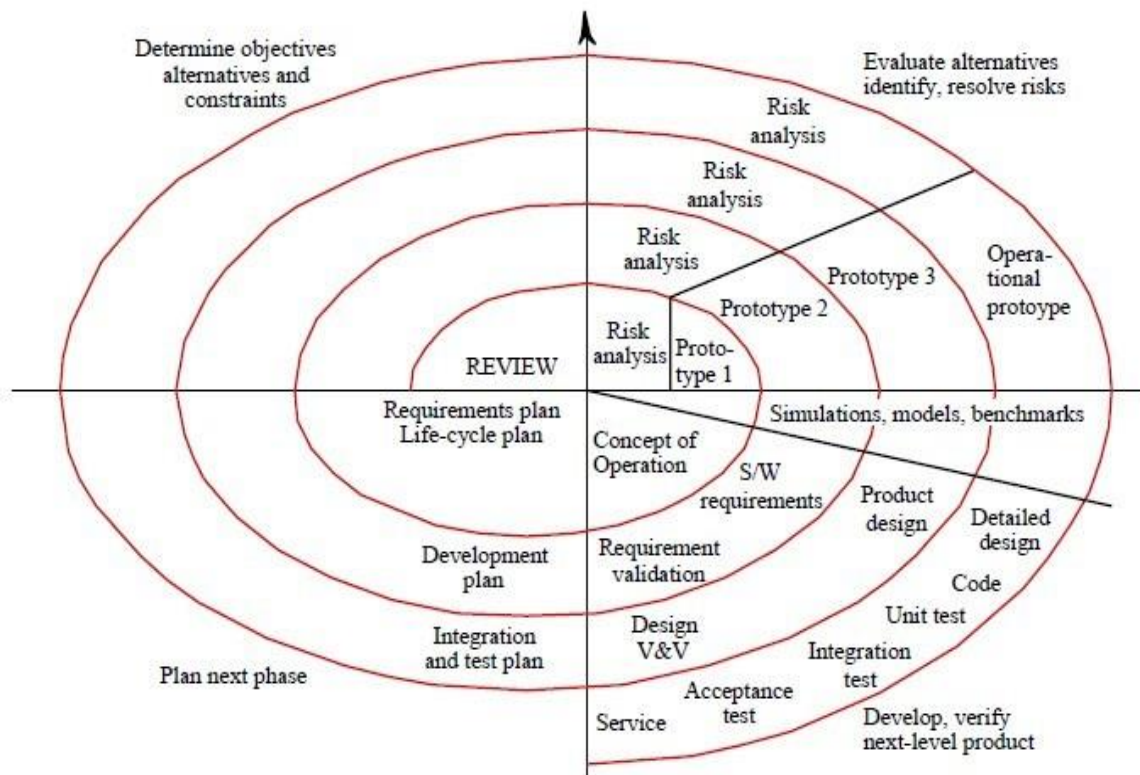
- i. Easy to explain to the user.
- ii. Stages and activities are well defined
- iii. Verification at each stage ensures early detection of errors
- iv. Widely used to identify and meet the milestones
- v. Establishes communication between customer and developer to meet the specifications.

Disadvantages of Waterfall Model

- i. The next stage begins only after the previous stage is complete, making it rigid.
- ii. User training is not given much importance.
- iii. Interaction with the user takes place right at the beginning and then at the time of deployment, which creates a gap between the phases.
- iv. Due to its cascading flow there is very little interaction from the user.

Boehmia's Spiral Model

- Spiral model was proposed by Boehm in 1988.
- Each loop in the spiral represents a phase of the software process.
- Innermost loop is concerned with system feasibility, next loop system requirement, followed by system design and so on



Each loop is split into 4 sectors:

1. Objective setting – Project risks are identified, alternative strategies depending on these risks may be planned.
2. Risk assessment and reduction - Project risks are identified · detailed analysis carried · steps taken to reduce risks.

Ex: A prototype system (Toy like implementation with limited functional capabilities and low reliability just for the purpose of examining)

3. Development and Validation – Choosing the most appropriate development model.
4. Planning – Project is reviewed and decisions are made whether to continue with further loop of the spiral.

Advantages of Spiral Model

- i. High amount of risk analysis
- ii. Supports large and high risk projects
- iii. Spiral model is one of the most flexible SDLC models.
- iv. Changes can be introduced later in the life cycle as well
- v. Project monitoring is easy as each loop requires a review from concerned people.

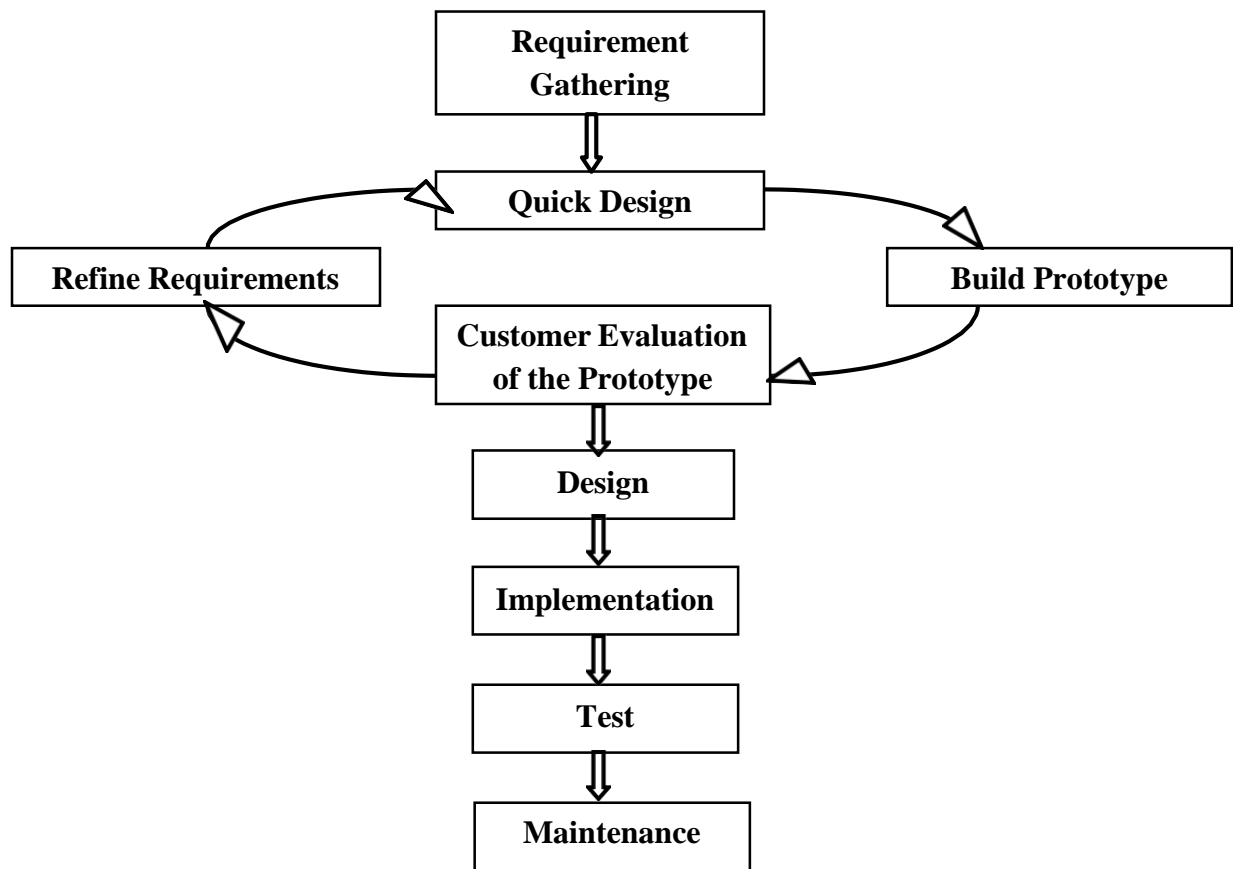
Disadvantages of Spiral model

- i. When to stop the spiral process is not clear.
- ii. Cost involved in this model is usually high.
- iii. Does not work well for smaller projects.
- iv. Project's success is highly dependent on the risk analysis phase.

Prototype Model

Prototype is a **partially developed product /dummy model** that allow customers and developers to analyze if the proposed system is suitable for the finished product.

- A prototype is a **toy implementation** which is built before starting actual development.
- The reason for developing a prototype is it is impossible to “get it right” the first time; we must plan to throw away the first product in order to develop a good product.
- The developed prototype is submitted to the customer/user for evaluation, based on the customer feedback the model is modified/refined. The cycle continues until the customer approves the prototype.



Advantages of Prototype Model

- i. Modification in prototype is faster.
- ii. Helps determine feasibility of the system.
- iii. Software Developers commitment is higher.

Disadvantages of Prototype Model

- i. Prototyping tools are expensive.
- ii. Design and code for the prototype is usually thrown away.
- iii. In order to get the prototype work quickly the quality is compromised.

Risk Management

What is a Risk?

“Risk is the potential future harm that may arise from some present action”.

What is Risk management?

Identifying risks and drawing up plans to minimize their effect on the project is called Risk management.

Risk management plays an important role in software development. There are several **types of risks**:

- 1) Project risks –Affects project resources/schedule.
- 2) Product risks- Affects quality or performance of software being developed.
- 3) Business risks- Affects organization development.
- 4) Generic risks- Affects overall project. Example: Loss of team members, loss of funding.

Process of risk management

1. Risk Identification: Project, product and business risks are identified.
2. Risk analysis: Consequences of risks are assessed.
3. Risk planning: Addresses the risk either by avoiding it or minimizing its effects.
4. Risk monitoring: Risk is constantly assessed and information about the risk becomes available.

CASE (Computer aided software engineering)

Computer-aided software engineering (CASE) is software to support software development. It provides automated support for software process activities. Ex: Automated translators to generate new versions of a program

Important Questions [2 Marks]

1. Define Software Engineering.
2. Write the goals of Software Engineering.
3. Define Software. Write its classification.
4. What is Software product?
5. Differentiate Generic and Bespoke products.
6. Mention any 4 characteristics of Software.
7. What is Prototype model?
8. What is Risk? Mention the various types.
9. What is feasibility study?
10. Define CASE.

Important Questions

1. Explain different phases of SDLC with a neat diagram.
2. Explain Waterfall model with Spiral model with its advantages and disadvantages.
3. What is Software Process? What are the activities involved in software process?
4. What are the key challenges facing Software Engineering?
5. Write short notes on Risk management.

Chapter 2

System Engineering

Definition: System – A System is a collection of inter related components that work together to achieve some objective.

A system may include software, mechanical, electrical and electronic hardware and be operated by people. Example – Security Camera.

Definition: System Engineering – The activity of designing, implementing, validating, installing and maintaining systems as a whole is known as System Engineering.

Emergent properties

Properties of the system as a whole rather than properties that can be derived from the properties of components of a system. Some examples of emergent properties are:

The overall weight of the system

- This is an example of an emergent property that can be computed from individual component properties.

The reliability of the system

- This depends on the reliability of system components and the relationships between the components.

The usability of a system

- This is a complex property which is not simply dependent on the system hardware and software but also depends on the system operators and the environment where it is used.

System and their Environment

- * Systems are not independent but exist in an environment
- * System's function may be to change its environment.
- * Environment affects the functioning of the system.
- * When a System is a part of another system, it is called the sub-system.

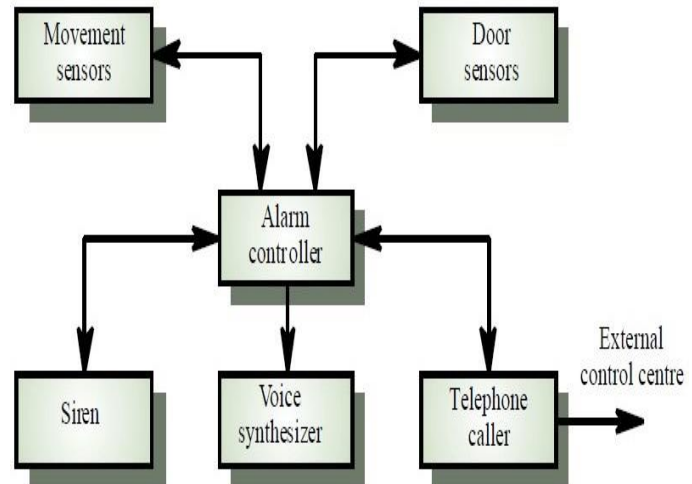
Example of System that includes Subsystem
(System Hierarchies)

- The heating System, Power System, Water System so on is all Sub-Systems within the building.
- The building is located in a street, which is in a town and so on.
- The system engineer should not only consider the system as complete entity but must also have knowledge of the Environment where the system needs to be installed.

A Simple Intruder Alarm System

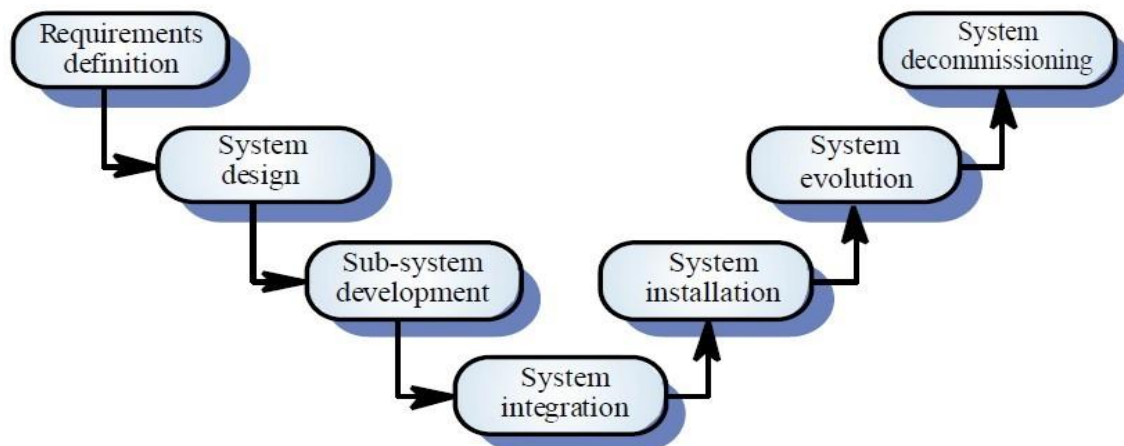
Sub system functionality in the intruder alarm.

- Sensor
- Movement sensor, door sensor
- Actuator
- Siren (Emits audible warning)
- Communication
- Telephone caller (makes external calls)
- Co-ordination
- Alarm controller (Controls system operations)
- Interface
- Voice synthesizer. (Notifies location of the suspect intruder)



System Engineering Process

It is a process that ensures that the customer's needs are satisfied throughout a system's entire life cycle.



System Requirements Definition

Three types of requirement defined at this stage

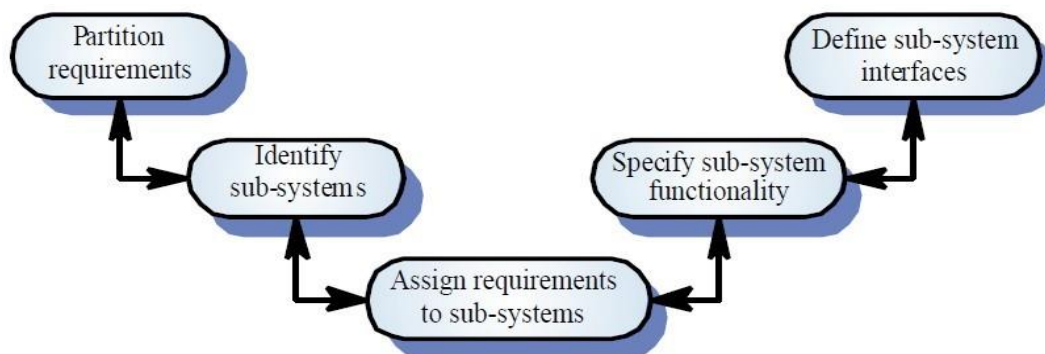
- Abstract functional requirements- System functions are defined in an abstract level
- System properties - Non-functional emergent requirements for the system such as reliability, usability etc in general are defined
- System behavior- It is important to specify what the system must not do and what it should do.

System Design Process

The Activities involved in this process are:

- Partition requirements
- Identify sub-systems
- Assign Requirements to sub system

- Specify sub system functionality
- Define sub-system interfaces



1. **Partition Requirements:** Analyze the requirements and organize them into related groups.
2. **Identify sub-systems:** Identify sub-systems that can individually or collectively meet the requirements.
3. **Assign Requirements to subsystem:** Assign the requirements to each identified sub-system.
4. **Specify sub-system functionality:** Relationship between the sub-systems should be identified at this stage which can collectively meet the system requirements.
5. **Define sub-system interface:** Once defining the sub-system interface have been agreed, parallel development of the sub-system becomes possible.

Sub-system development

- Typically parallel projects developing the hardware, software and communications
- It may involve some COTS (Commercial Off-the-Shelf) components.
- COTS systems may not meet the requirements exactly but it is worth to modify and use.
- It is cheaper to buy existing products rather than develop special purpose components.
- The sub-system development activity involves developing each of the sub-systems identified during system design.

System Integration

The process of putting hardware, software and people together to make a system.

Integration can be done in 2 ways:

1. **Big Bang Method-** All the sub-systems are integrated in the same way.
2. **Incremental Integration-** Sub-systems are integrated one at a time; it reduces the cost of Error location.

Once the components have been integrated, System Testing takes place.

System Installation

It is the activity of installing the system in the environment in which it is intended to operate.

Some of the problems that can arise during installations are:

- Environmental assumptions may be incorrect.
- May be human resistance to the introduction of a new system.
- System may have to coexist with alternative systems for some time.
- May be physical installation problems (e.g. cabling problems).
- Operator training has to be identified.

System Evolution

- Large systems have a long lifetime. They must evolve to meet changing requirements

Evolution is inherently costly for the following reasons:

- Changes must be analyzed from a technical and business perspective
- Sub-systems interact so unanticipated problems can arise
- System structure is corrupted as changes are made to it, hence its cost changes.
- Existing systems which must be maintained are sometimes called legacy systems.

System Decommissioning

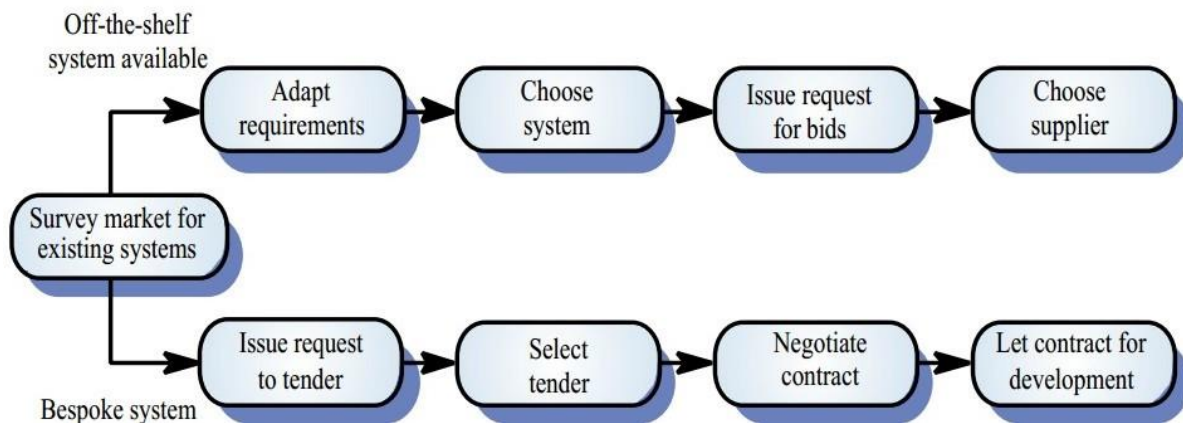
- System Decommissioning means taking the system out of service after the end of its useful operational lifetime.
- Example : For hardware systems this may involve de-assembling and recycling materials.
- May require data to be restructured and converted so that it can be used in some other system.

System Procurement

Definition: Acquiring a system for an organization to meet some need/requirement is called System Procurement.

Some System specification and architectural design is usually necessary before procurement

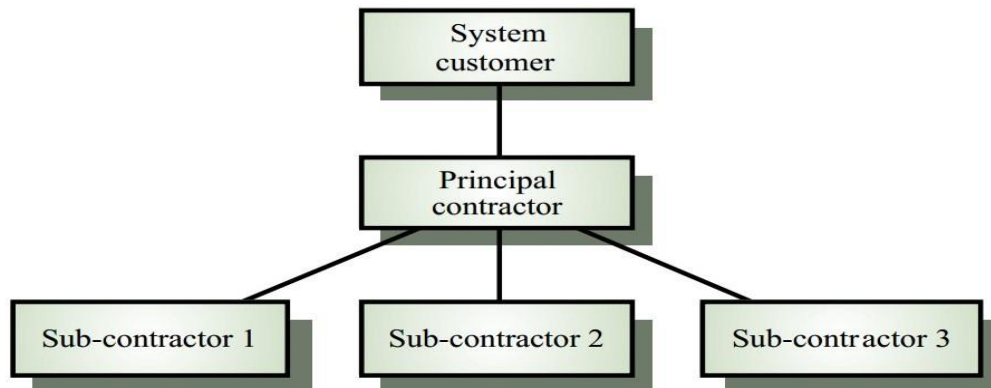
- You need a specification to let a contract for system development
- The specification may allow you to buy a commercial off-the-shelf (COTS) system, always cheaper than developing a system from scratch.



Procurement issues

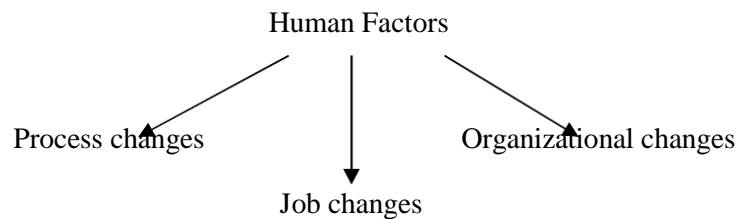
- Requirements may have to be modified to match the capabilities of off-the-shelf components.
- There is usually a contract negotiation period to agree changes after the contractor to build a system has been selected

Contractor/Sub-contractor model



- The procurement of large hardware/software systems is usually based around some principal contractor.
- Sub-contracts are issued to other suppliers to supply parts of the system.
- Customer leases with the principal contractor and does not deal directly with sub-contractors.

Human Factors



Process changes: In a organization a change in the process can result in job loss of the workers, as they lack training in the new system which becomes difficult for them to cope.

Job changes: As new and faster systems are introduced, the workers may have to change the way they work.

Organizational changes: If a organization is dependent on a complex system, those who know how to operate the system have a great deal of power in the organization.

.....

[Note: System Architectural model of a ship to be written in notes. Pg 2.10-2.11]

Chapter 3

Software Requirement Analysis and Specification

Definition: Software Requirement- The process of establishing the services that the customer requires from a system and the constraints under which it operates and is developed.

Requirements may serve a dual function

- May be the basis for a bid for a contract - therefore must be open to interpretation.
- May be the basis for the contract itself - therefore must be defined in detail.

Both these statements may be called requirements

Definition: Software Requirement Specification – A structured document setting out detailed descriptions of the system services written as a contract between client and contractor.

Classification of Software Requirements

1. 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.

2. Non-Functional Requirements

Constraints on the services or functions offered by the system such as timing constraints, cost constraints, constraints on the development process, standards etc.

3. System requirements

A structured document setting out detailed descriptions of the system services written as a contract between client and contractor

4. User Requirements

Statements in natural language plus diagrams of the services the system provides and its operational constraints. It is written for customers.

Functional Requirements 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.

*The Functional requirements of the system describe what the system should do.

*It depends on the type of software, expected users and the type of system where the software is used.

*Functional user requirements may be high-level statements of what the system should do but it should describe the system services in detail.

Non - Functional Requirements:

*Non-functional requirements may be more critical than functional requirements. If these are not met, the system is useless.

*Constraints on the services or functions offered by the system such as timing constraints, cost constraints, storage constraints, constraints on the development process, standards etc.

Classification of Software Requirements

Product requirements

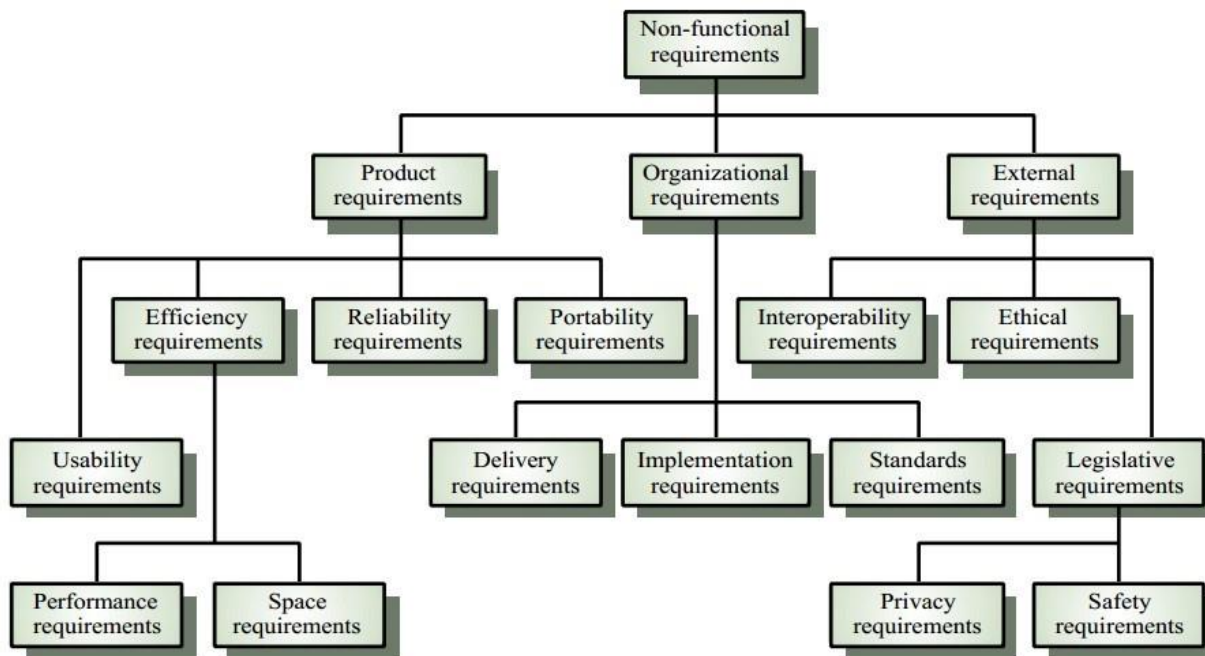
- Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability etc.

Organizational requirements

- Requirements which are a consequence of organizational 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.



Software Requirement Specification

Outline Structure of Software Requirement Specification

IEEE Standards suggests the following structure for SRS document

1. Introduction

- 1.1 Purpose of the requirement's document.
- 1.2 Scope of the product.
- 1.3 Definitions, acronyms and abbreviations.
- 1.4 References
- 1.5 Overview of remainder of the document.

2. General Description

- 2.1 Product perspective.
- 2.2 Product functions.
- 2.3 User characteristics.
- 2.4 General constraints.
- 2.5 Assumptions and dependencies.

3. Specific Requirements

- 3.1 Functional requirements
- 3.2 Non Functional requirements
- 3.3 Interface requirements

4. Appendices

5. Index

Definition: SRS is a formal document, which acts as a representation of software that enables the users to review whether it (SRS) is according to their requirements.

A structured document setting out detailed descriptions of the system services written as a contract between client and contractor.

Various other **Purposes** served by SRS are listed below.

1. **Feedback:** Provides a feedback, which ensures to the user that the organization (which develops the software) understands the issues or problems to be solved and the software behavior necessary to address those problems.
2. **Decompose problem into components:** Organizes the information and divides the problem into its component parts in an orderly manner.
3. **Input to design:** Contains sufficient detail in the functional system requirements to devise a design solution.
4. **Basis for agreement between the user and the organization:** Provides a complete description of the functions to be performed by the system.
5. **Reduce the development effort:** Enables developers to consider user requirements before the designing of the system commences. As a result, 'rework' and inconsistencies in the later stages can be reduced.
6. **Estimating costs and schedules:** Determines the requirements of the system and thus enables the developer to have a 'rough' estimate of the total cost and schedule of the project.

Characteristics of SRS

1. **Correct:** Correctness ensures that all specified requirements are performed correctly to meet the software.
2. **Unambiguous:** SRS is unambiguous when every stated requirement has only one interpretation. This implies that each requirement is uniquely interpreted.
3. **Complete:** SRS is complete when the requirements clearly define what the software is required to do. This includes all the requirements related to performance, design and functionality.

Components of SRS

Functionality

Environment and System Objectives

System Delivery and Installation

Design constraints

External interface requirements

Conceptually, any SRS should have these components. Now we will discuss them one by one.

1. Functionality:

- Procedures for starting up and closing down the system.
- Operations on Normal condition.
- Operations on abnormal conditions.

2. Environment and System Objectives

- Physical Attributes of the environment : size, shape and locality
- Safety/Security/Hazards

3. System Delivery and Installation

- Examples of these requirements are: Deadlines/ Quality assurance/ document structure/ standards/training/manuals/ support and maintenance.

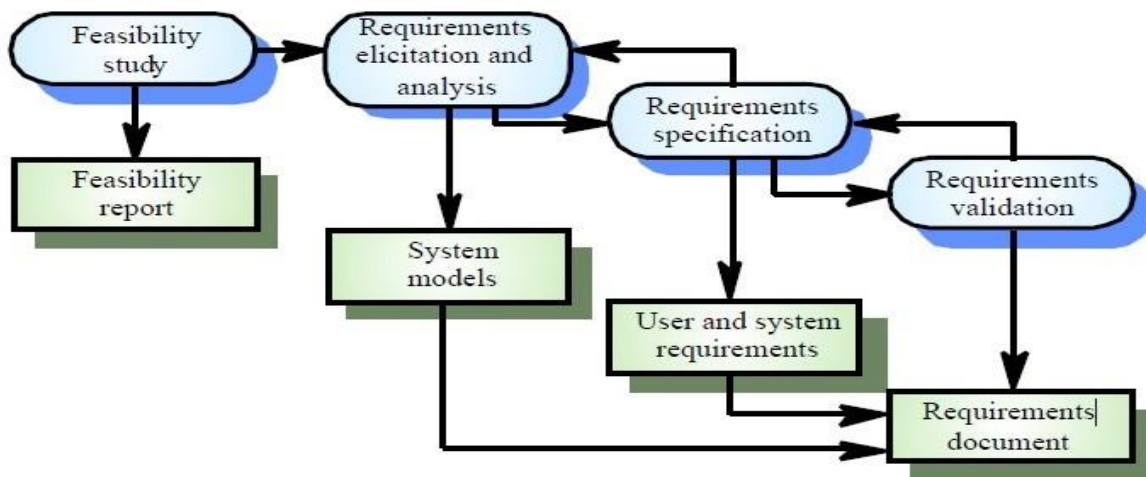
4. Design Constraints

- Hardware/Software standards, particular libraries, operating systems to be used and compatibility issues.

5. Functional Constraints

- Properties are: Performance, efficiency, response times, safety, security, reliability, quality and dependability.

The requirements engineering process



The process of establishing the services that the customer requires from a system and the constraints under which it operates is called **Requirement Engineering process**.

1. Feasibility study: The study determines whether or not a system is financially worthwhile and technically feasible.

It is the study that checks A) If the system contributes to organizational objectives. B) If the system can be engineered using current technology and within budget. C) Based on information assessment (what is required), information collected a brief report is written.

A feasibility study is conducted to answer many questions like

- What are current process problems?
- How will the proposed system help?
- What will be the integration problems?
- Is new technology needed? What skills?
- What facilities must be supported by the proposed system?

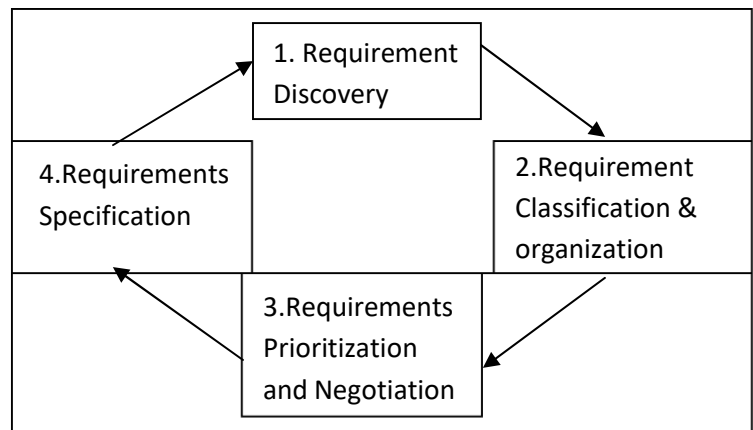
2. Requirement Elicitation and Analysis:

Activities involved in Elicitation and Analysis are

- Requirement Discovery
- Requirement classification and organization
- Requirement prioritization and negotiation
- Requirement Specification

The process involves technical staff working with customers to find out about the application domain, the services that the system should provide and the system's operational constraints.

It may involve end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc. These are called **stakeholders**.



The various activities involved in Requirement Analysis and Elicitation are :

- Domain Understanding : The analyst must understand the Background Knowledge of the application.
- Requirement Collection : Gathering the requirement by involving the users and stakeholders.
- Requirement classification : Organizing the requirements gathered from different sources.
- Requirement conflicts : Involves stakeholders and engineers to solve the problems that contradict the organization and business rules.
- Requirement prioritization : Discovering the most important requirements by interacting with stakeholders and organizing them into most priority order.
- Requirement validation : Checks stakeholder's expectation on the system with gathered requirements.

Techniques for Requirement Elicitation and Analysis

1. View point oriented elicitation
2. Scenarios
3. Ethnography

1. View point oriented elicitation

Stakeholders represent different ways of looking at a problem or problem viewpoints.

There are 3 types of viewpoints:

1. Interactive View points : Represents people or other systems that interact directly with the system. EX : In a bank ATM , the banks customer and banks account database.
2. Indirect view points : Represents stakeholders who do not use the system directly but influence the system in some way. EX : Management of the bank and bank security staff.
3. Domain view points : Represents domain characteristics and constraints that influence system requirements. EX: Standards that have been developed for inter banking communication like ATM.

The example used here is an auto-teller system which provides some automated banking services. Services include cash withdrawal, message passing (send a message to request a service), ordering a statement and transferring funds.

Auto- teller Viewpoints

Bank customers
Representatives of other banks
Hardware and software maintenance engineers
Marketing department
Bank managers and counter staff
Database administrators and security staff
Communications engineers
Personnel department

2. Scenarios

Scenarios are descriptions of how a system is used in practice.

They are helpful in requirements elicitation as people can relate to these more readily than abstract statement of what they require from a system.

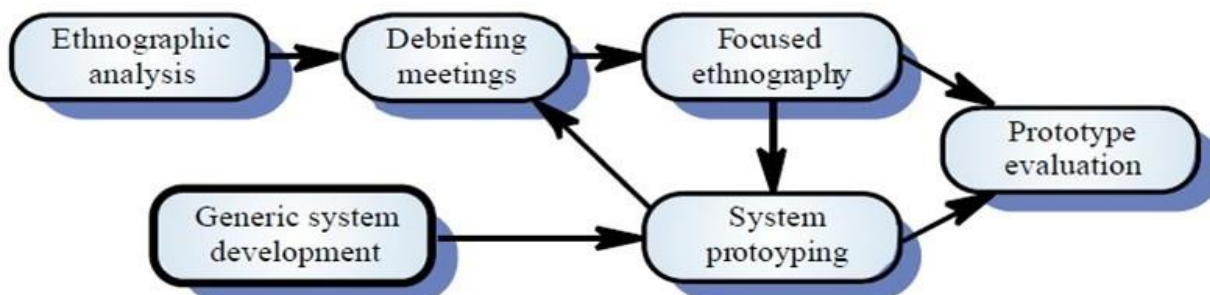
Scenarios are particularly useful for adding detail to an outline requirements description.

Scenario Descriptions

- System state at the beginning of the scenario
- Normal flow of events in the scenario
- What can go wrong and how this is handled
- Other concurrent activities
- System state on completion of the scenario

3. Ethnography

Definition: "Ethnography is an observational technique that is used to understand social and organizational requirement."



Ethnographic studies have shown that work is usually richer and more complex than suggested by simple system models. These are the Requirements that are derived from the way that people actually work. Ethnographic requirements are derived from cooperation and awareness of other people's activities.

Requirements Specification Methods

Specification Method	Description
Structured Natural language	This approach depends on defining standard forms or templates to express the requirements specification in natural language.
Design description language	This approach uses a language like a programming language but with more abstract features to specify the requirements by defining an operational model of the system.
Graphical notations	It has much complex graphical vocabulary and is most widely used by specialists.
Mathematical specifications	These are notations based on mathematical concepts such as finite-state machines or sets.

Requirement Validation

Requirement Validation is concerned with demonstrating the requirements that define the system which customer really wants. Requirements error costs are high so validation is very important i.e. fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error.

Requirement Validity Checks

Requirement Validity Checks	Description
1.Validity Checks	Aims to ensure that the system meets all functional, behavioral and performance requirements.
2.Consistency Checks	Requirements collected must be consistent and should not lead to conflict.
3.Realism Checks	These checks take into account the budget and schedule for the system development.
4.Verifiability	At the completion of the system, it must be possible to demonstrate that the delivered system meets all the requirements.

Requirement Validation techniques

1. Requirement Reviews

2.Prototyping

3.Test Case Generation

Requirements reviews

- Systematic manual analysis of the requirements
- Reviews here can also check for a)Verifiability, b)Comprehensibility, c)Traceability and d)Adaptability

Prototyping

- Using an executable model of the system to check requirements.

Test-case generation

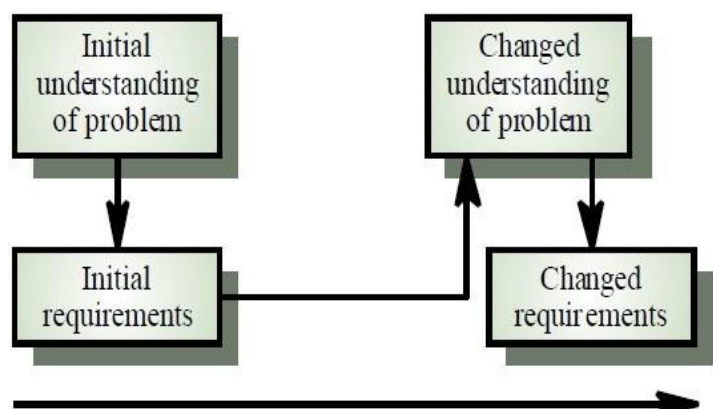
- Developing tests for requirements to check testability, if a test is difficult or impossible to design that means requirements are unrealistic.

Requirement Management

Requirement Evolution

Based on the Requirement Evolution, requirements are classified into 2 types :

- 1) Enduring Requirements
- 2) Volatile Requirements



Enduring requirements- Stable requirements derived from the core activity of the customer organization. E.g. a hospital will always have doctors, nurses, etc. May be derived from domain models.

Volatile requirements - Requirements which change during development or when the system is in use. In a hospital, requirements derived from health-care policy etc