What is Software (CO1)

Software is

- An instruction that when executed provide desired feature, function and performance
- Data structure that enable the programs to adequately manipulate information.
- Documents that describe the operation and use of the programs

What is Software?

IEEE definition of Software:-

Software is the collection of computer programs procedures, rules and associated documentation and data.

To be precise

Software is the collection of computer programs whose objective is to enhance the capabilities of hardware

Importance of Software

- It has become a driving force.
- It is engine that drive business decision making.
- It serve as the basis for modern scientific investigation and engineering problem solving.
- It is embedded in all kind of systems like transportation , medical, telecommunications, military, industrial process, entertainment, office products

Importance of Software

- It affects nearly every aspect of our lives and has become pervasive in our commerce, our culture and our every day activities software impact on our society and culture is significant .
- As software importance grows, the software community continually attempts to develop technologies that will make it easier faster and less expensive to build high quality computer programs

Introduction to Software Engineering

- IEEE Definition:-

  Software Engg. Is the application of systematic, disciplined, quantifiable approach to the development, operation and maintenance of software.

or

Software Engineering Is an Engineering discipline which is concerned with all aspects of software production

Introduction to Software Engineering

- According to fritz bauer:-

  Software Engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and work efficiently on real machine

Software Components (CO1)

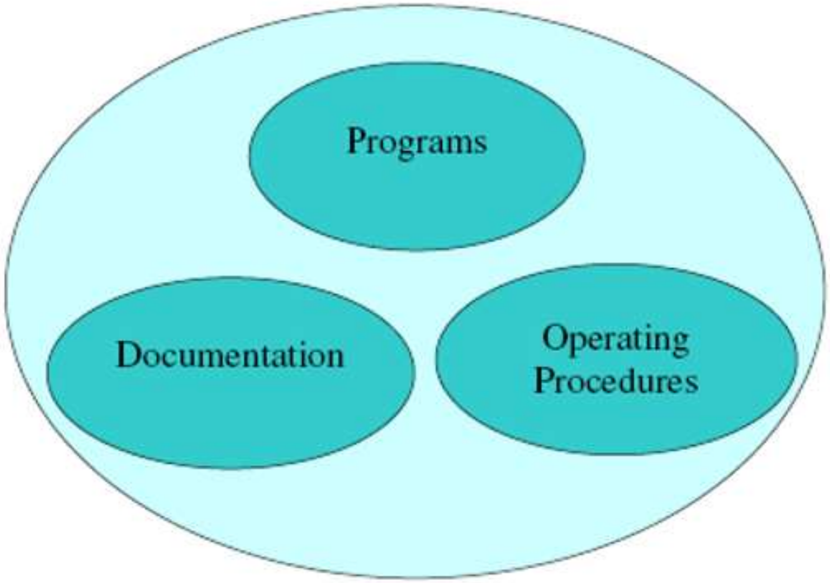
- Software components is a system elements offering predefined service and able to communicate with other components

It should fulfill following five criteria

1. Multiple use
2. Non context specific
3. Composable with other components
4. Encapsulated i.e. non investigable through its interfaces
5. A unit of independent deployment and versioning

Software Components

- A simpler definition can be – A component is an object written to a specification. It does not matter what the specification is :com, java beans
- Software components often take the form of objects or collection of objects (from object oriented programming). In some binary or textual form adhering to some interface description language (IDL) so that the component may exist autonomously from other component i



Software Components

The components needs

1. To be fully documented
2. More through testing
3. Robust input validity checking
4. To pass back useful error messages as appropriate
5. To be built with an awareness that it will be put to unforeseen uses
6. A mechanism for compensating developers who invest the (substantial) effort implied above

Objective of software Engineering

- The basic objective of software engineering is to develop methods and procedures for software development that can scale up for large systems and that can be used consistently to produce high-quality software at low cost and with a small cycle of time

Software Characteristics(CO1)

The key characteristics of software are

1. Most software is custom built, rather than being assembled from existing components.

2. Software is developed or engineered; It is not manufactured in the classical sense.

3. *Software is flexible.*

4. *Software doesn't wear out.*

Software Characteristics

Software does not have wear out phase

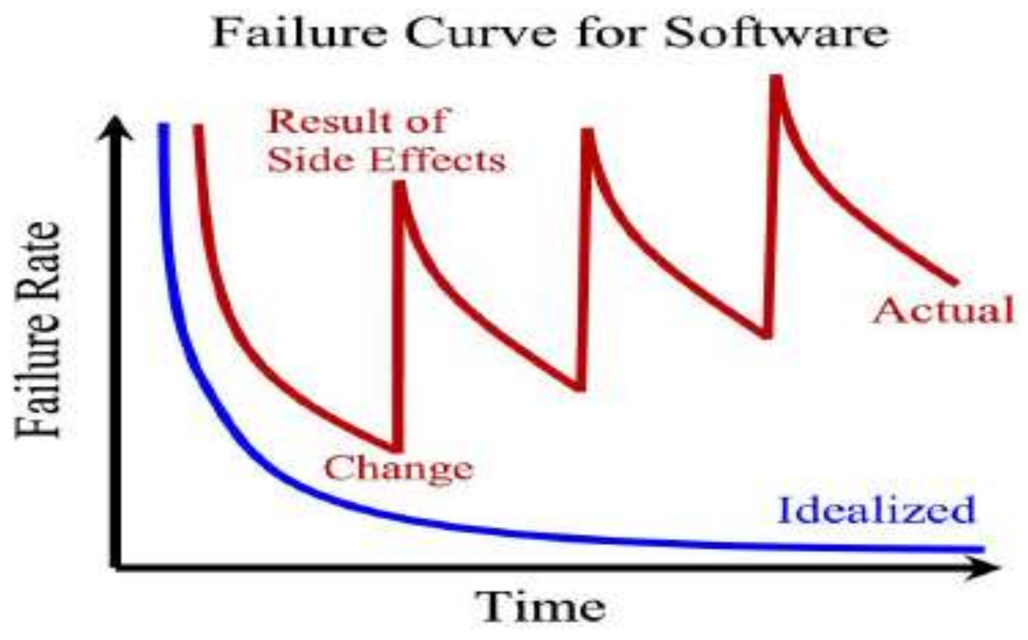


Figure: Software does not have wear out

Software Characteristics

Failure rate of hardware

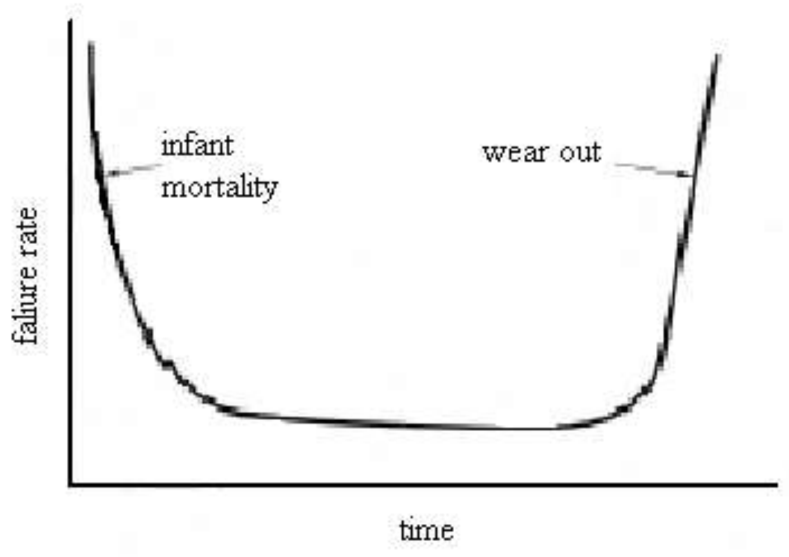


Figure: Failure rate of hardware

Software Characteristics(CO1)

- We don’t have this phase for the software as it does not wear out.
- Important point is software becomes reliable overtime instead of wearing out
- It becomes obsolete.
- Software may be retired due to environmental changes , new requirement and new expectation

Software Crisis (CO1)

History:
The name arose in the 1960's with a '*software crisis*'

As hardware costs decreased the demand for software increased

Today:

Industrialized economies have become increasingly dependent on effective software – now a huge software industry
Software is an important part of most aspects of human activity:
Embedded in systems such as transportation, medical, military, telecommunications, entertainment, industrial, office machines …

Software Crisis

According to latest IBM report 31% of the projects get cancelled before they are completed,53% over run their cost estimates by an average of 189% and for every 100 projects there are 94 restarts.

During development phase many problems are encountered. These set of problem is known as software crisis.

Software Crisis

The problem and causes of software crisis encountered on different stage of software development are

1. <u>Problems</u> : So, what are the problems?

(i)Demand still outstrips supply

(ii) ~70% of software effort is maintenance because software:

(iii) Software is not produced on time and within budget

(iv) Software is becoming larger and more complex

(v) Many software products have poor quality

(vi) Software development is costly

(vii)Serious schedule overruns are common

2. Problems:

Schedule and cost estimate are often grossly inaccurate. Communication between customer and software developer is often poor. The quality of software is sometimes less than adequate.

Software Crisis

3. <u>SOFTWARE CRISIS IN THE PROGRAMMER'S POINT OF VIEW:-</u>

Problem of compatibility

Problem of portability

Problem in documentation

Problem of piracy of software

Problem in coordination of work of different people

Problem of maintenance in proper manner

Software Crisis

4. SOFTWARE CRISIS IN THE USER'S POINT OF VIEW:-

Software cost is very high

Customers are moody or choosy

Hardware goes very down

Lack of specialization in development

Problem of different version of software

Problem of views

Problem of bugs

Software Process(CO1)

Software process is the related set of activities and process that are involved in developing and evolving a software system.

or

A set of activities whose goal is the development or evolution of software.

or

A software process is a set of activities and associated results which produce a software product.

or

– A set of interrelated activities, which transform inputs into outputs (*ISO 12207/8402*)

used by an organization or project to plan, manage, execute, monitor, control and improve any software related activity

Software Process

There are four fundamental process activities which are common to all software processes. These activities are

1. **Software specification:-** the functionality of the software and constraints on its operation must be defined.
2. **Software development:-** 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 the changing customer needs.

Different software processes organize these activities in different ways and are described at different levels of detail. The timing of the activities varies, as does the result of each activity. Different organizations may use different process to produce the same type of product. However some process are more suitable than other for same types of application.

Software Process

- The software industry considers software development as a process. According to Booch and Rambough A process defines *who is doing what, when and how to reach a certain goal ?*

  Software Engineering is a field, which combines process, methods and tools for the development of software.

  The concept of process is the main step in the software engineering approach. When these activities are performed in specific sequence in accordance with ordering constraints, the desired results are produced.

Evolution Of Software (CO1)

Software Engineering principles have evolved over the past more than fifty years from art to an engineering discipline.

Development in the field of hardware and software computing make a significant change in the twentieth century. We can decide software development process in to four eras.

Early Era: During the early years general-purpose hardware became common place. Software, on the other hand, was custom-designed for each application and had a relatively limited distribution. Most software was developed and ultimately used by the same person or organization.

In this era the software are mainly based on (1950-1960)

- Limited Distribution
- Custom Software
- Batch Orientation

Evolution Of Software

Second Era: The second era to computer system evolution introduced new concepts of human machine interaction. Interactive techniques opened a new world of application and new levels of hardware and software sophistication. Real time software deals with changing environment and one other is multi-user in which many user can perform or work on a software at a time.

In this era the software are mainly based on (1960-1972)

- Multi-user
- Data base
- Real time
- Product software
- Multiprogramming

Evolution Of Software

Third Era: In the earlier age the software was custom designed and limited distribution but in this era the software was consumer designed and the distribution is also not limited. The cost of the hardware is also very low in this era.

In this era the software are mainly based on(1973-1985)

- Embedded intelligence
- Consumer impact
- Distributed systems
- Low cost hardware

Fourth Era:- The fourth era of computer system evolution moves us away from individual computers and computer programs and towards the collective impact of computers and software. As the fourth era progresses, new technologies have begun to emerge.

Evolution Of Software

In this era the software are mainly based on(1985-)

- Powerful desktop system
- Expert system
- Artificial intelligence
- Network computers
- Parallel computing
- Object oriented technology

  at this time the concept of software making is object oriented technology or network computing etc.

Definition of Software Engineering Process and Conventional Engineering Process

- **1. Software Engineering Process** :
  It is a engineering process which is mainly related to computers and programming and developing different kinds of applications through the use of information technology.
- **2. Conventional Engineering Process :**
  It is a engineering process which is highly based on empirical knowledge and is about building cars, machines and hardware.

Similarity with conventional Engineering Process

- **Conventional Engg.(steps to solve problem)**
  - Problem formulation and analysis.
  - Search for alternatives.
  - Decision and specification
  - Implementation
  - Testing
- **Software Engg.(steps to solve problem)**
  - Understand the problem(analysis and specification)
  - Alternative solution
  - Plan a solution(modeling and s/w design)
  - Carry out the plan(coding)
  - Examine the result for accuracy(testing and quality assurance)

Both develop tool and technique for high quality useful product

Similarity with conventional Engg. Process

s/w Engg. Process	Conventional Engg. Process
• Based on computer sci., information science and discrete mathematics.	• Based science, mathematics and empirical knowledge
• Construct non real(abstract) artifact.	• Construct real artifact.
• Main concern is cost of development and reliability is measured by no. of error per thousands lines of source code .	• Main concern is cost of production and reliability measure by time of failure.
• 50 years old.	• 1000 years old.
• s/w engineering are not get able to precisely describe and measure(a) material they used and (2)result of their work.	• Able.
• LOC, function point or complexity measure only capture the amount of s/w created.	• Use meter, volt and other units of measurement.
• Maintenance apply new and untested elements in s/w project.	• Maintenance try to apply known and tested principle

Software Quality Attributes

Software Quality Attributes are:

- Correctness, Reliability, Adequacy, Learnability, Robustness, Maintainability, Readability, Extensibility, Testability, Efficiency, Portability.

Correctness:

- The correctness of a software system refers to:
- – Agreement of program code with specifications
  – Independence of the actual application of the software system.

Software Quality Attributes

Reliability: Reliability of a software system derives from

- – Correctness
  – Availability
- The behavior over time for the fulfillment of a given specification depends on the reliability of the software system.

Software Quality Attributes

Adequacy:

The input required of the user should be limited to only what is necessary. The software system should expect information only if it is necessary for the functions that the user wishes to carry out.

Learnability:

Learnability of a software system depends on:

– The design of user interfaces

– The clarity and the simplicity of the user instructions

Software Quality Attributes

Robustness: Robustness reduces the impact of operational mistakes, erroneous input data, and hardware errors.

Maintainability: Maintainability = suitability for debugging (localization and correction of errors) and for modification and extension of functionality.

- The **maintainability** of a software system depends on its:
- – Readability
  – Extensibility
  – Testability

Software Quality Attributes

Readability: Readability of a software system depends on its:

- – Form of representation
  – Programming style
  – Consistency
  – Readability of the implementation programming languages
  – Structuredness of the system
  – Quality of the documentation
  – Tools available for inspection

Extensibility: Extensibility allows required modifications at the appropriate locations to be made without undesirable side effects.

Software Quality Attributes

Testability: suitability for allowing the programmer to follow program execution (runtime behavior under given conditions) and for debugging. The testability of a software system depends on its:

- – Modularity
  – Structuredness
- Modular, well-structured programs prove more suitable for systematic, stepwise testing than monolithic, unstructured programs.
- Testing tools and the possibility of formulating consistency conditions (assertions) in the source code reduce the testing effort and provide important prerequisites for the extensive, systematic testing of all system components.

Software Quality Attributes

Efficiency: ability of a software system to fulfill its purpose with the best possible utilization of all necessary resources (time, storage, transmission channels, and peripherals).

Portability: the ease with which a software system can be adapted to run on computers other than the one for which it was designed.

- – Structuredness: System-dependent elements are collected in easily interchangeable program components.

Objective of QA

In a nutshell the role of QA is to provide information about the quality of the product to the relevant people. If you release software without any kind of QA, then you are not assured of anything, and are essentially pushing the responsibility of finding issues in your software to your users. There is a lot of disagreement about the best way to perform this function, but few would argue it must be performed at some point, to some extent, by someone.

Software Development Life Cycle

- It is used to facilitate the development of large s/w product in a systematic, well defined and cost effective manner.
- It is the period of time that start when a software product is conceived and end when the product is no longer available for use.
- **Purpose of SDLC**
  1. Help to understand entire process.
  2. Enforces a structured approach to development.
  3. Enables planning of resources in advance.
  4. Enables subsequent controls of them.
  5. Aids mgmt. to track progress of system.
- SDLC can be decided min. 5 and max. 9 phases. Each phase identified along with Entry and Exit Criteria.

SDLC

- **SDLC includes the following phases**
  1. Project initiation and planning/ recognition of need/ preliminary investigation
  2. Project identification and selection/ feasibility study.
     - I. Organizational feasibility.
     - II. Economic feasibility.
     - III. Technical feasibility.
     - IV. Operational feasibility.
  3. Project analysis.
  4. System design.
     - I. Output design.
     - II. Input design.
     - III. File design.
     - IV. Processing design.
     - V. Detailed system documentation.
  5. Coding.

SDLC

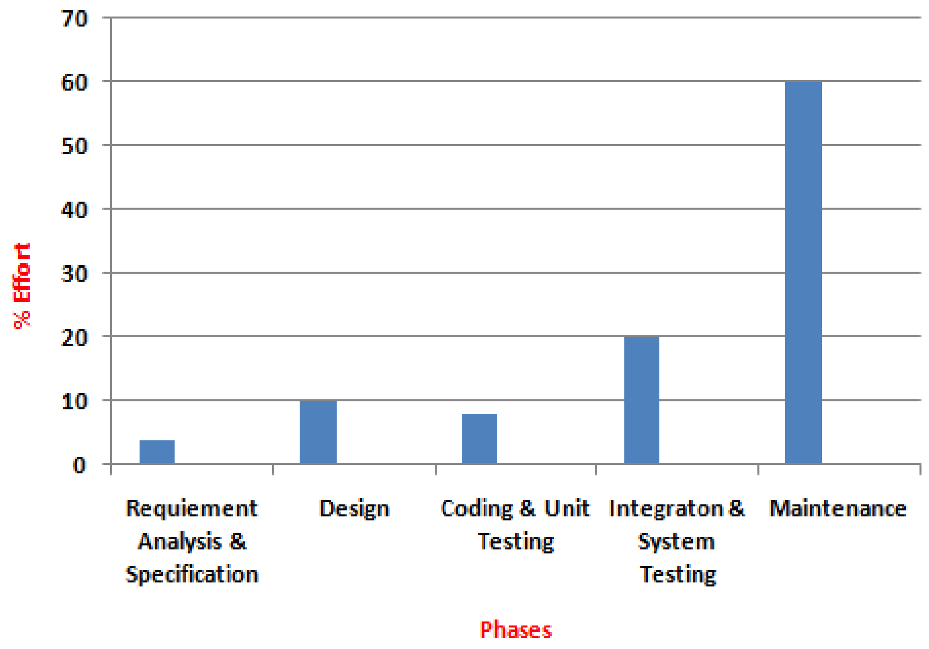
6. Testing.
    - I. Unit Testing (testing each module isolation from other modules)
    - II. System Testing
        - ➢ Alpha Testing (performed by development Team)
        - ➢ Beta Testing (performed by friendly set of customers)
        - ➢ Acceptance Testing (performed by customer himself)

7. Implementation

8. Maintenance.
    - I. Corrective maintenance.
    - II. Adaptive maintenance.
    - III. Perfective maintenance.
    - IV. Preventive maintenance.

% Effort Involved in SDLC Phases

- Maintenance Phase requires max.(60%) effort.
- In development phases Integration and System Testing requires max. effort



Source: Software Engg. Book by Rajeev Mall

Waterfall Model (CO1)

- Origins
  - Proposed in the early 70s & widely used even today
  - Based strongly on the way hardware systems are designed & implemented

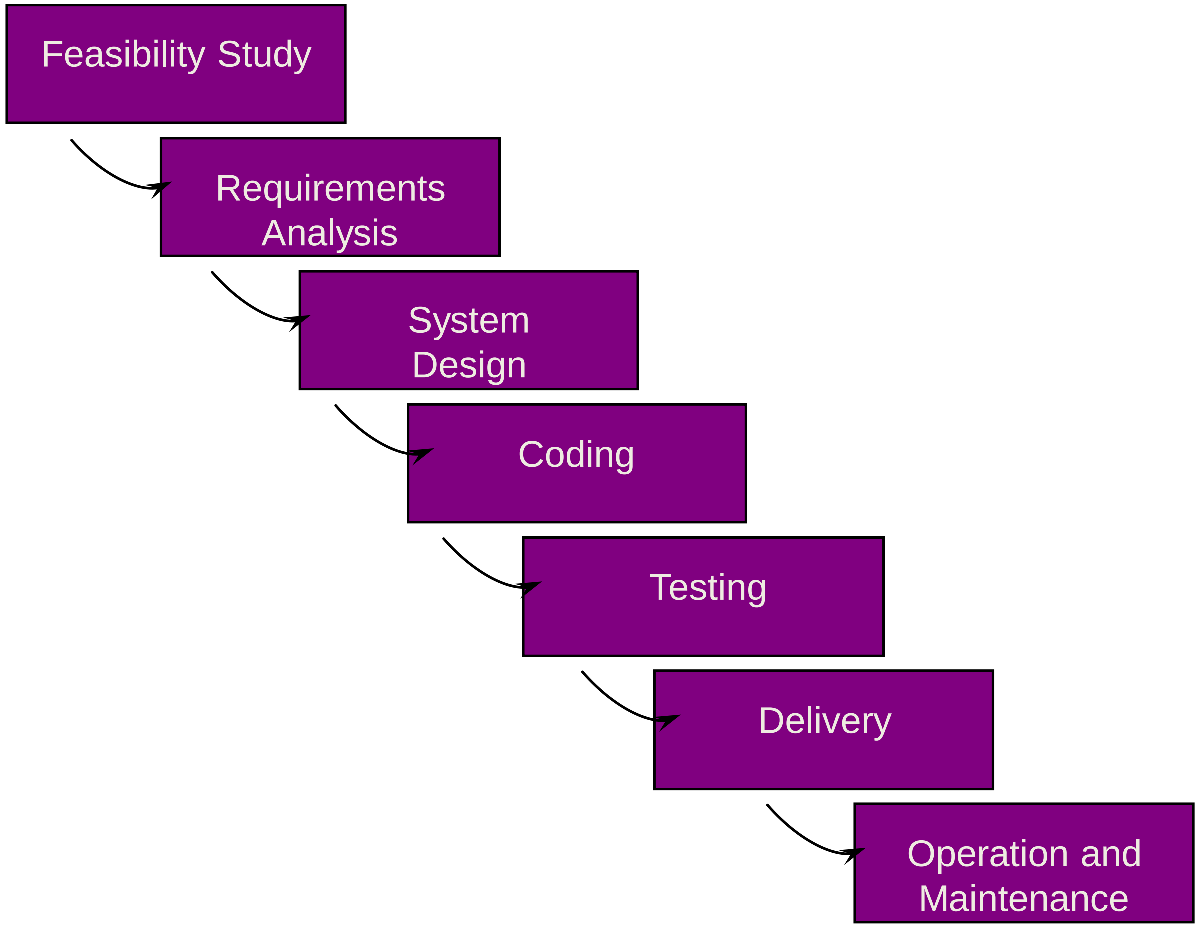
- A model of the software development process in which the constituent activities - concept, requirements, design, development, testing & installation- are performed in a linear order.

- There is a single pass through each phase. Overlap between these phases is neither precluded nor mandated.

- There is little or no iterative development

- Sometimes called systematic sequential approach

Waterfall Model



Waterfall Model

The phases shown in the figure are the following:

- Feasibility Study
- Requirements Analysis
- System Design
- Coding
- Testing
- Delivery
- Operation And Maintenance

Waterfall Model

Advantages:

- Easy to understand even by non technical persons i.e., customers
- Each phase has well defined inputs and outputs e.g., input to system design stage is Requirement Specification Documents (RSD) and output is the design document
- Easy to use as software development proceeds.
- Each stage has well defined manager deliverables or milestones
- Helps the project in proper planning of the project.

Waterfall Model

Disadvantages:

- It is difficult to define all requirements at the beginning of a project.
- This model is not suitable for accommodating any change.
- It involves heavy documentation.
- We can not go in the backward direction while SDLC performs
- The model is not suitable for new projects because of uncertainty in the specifications

Prototype Model (CO1)

- A prototype is a partially developed product. It is a process of developing working *replica* of a system.

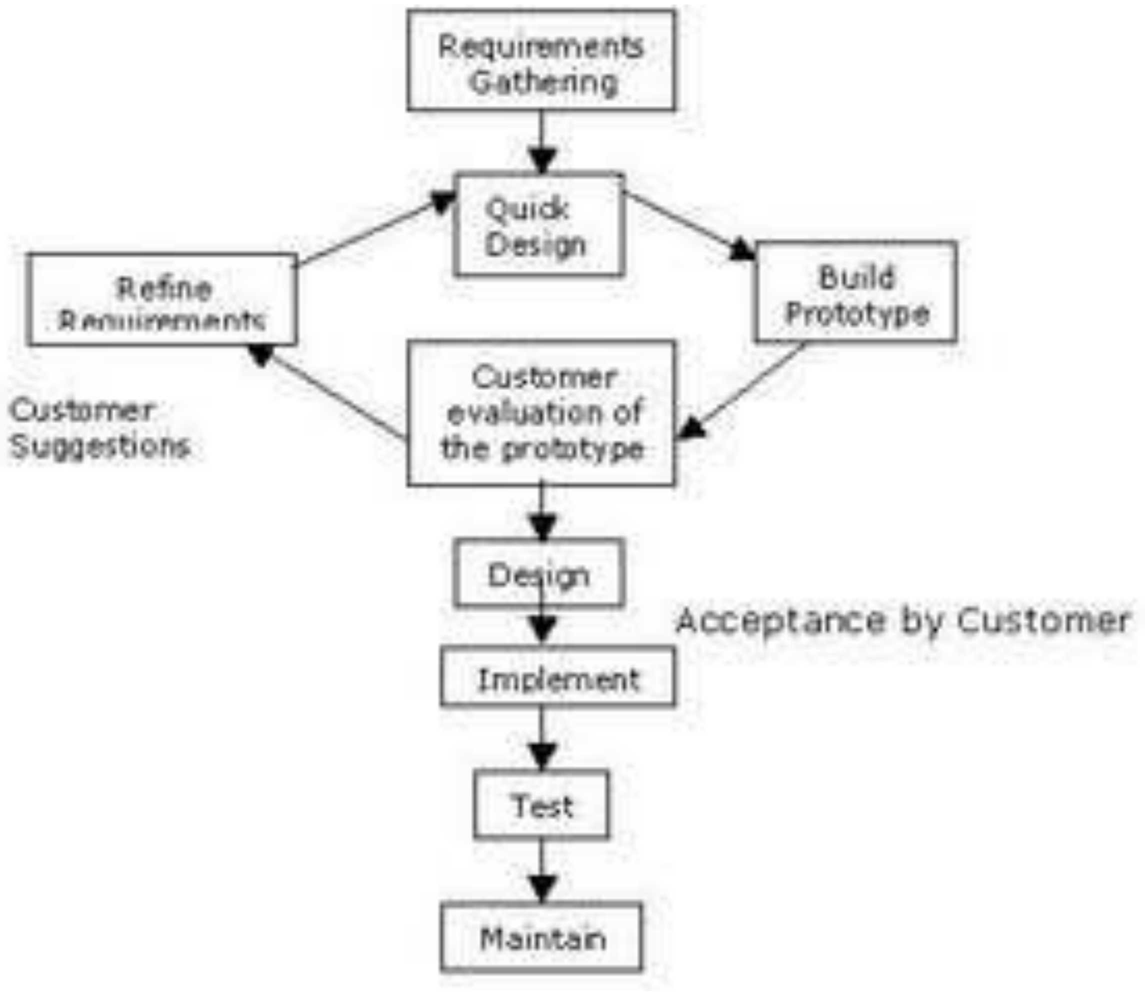
- It always happen that a customer defines a set of general objectives for software but does not identify detailed input, processing, or output requirements. In other words the developer may be unsure of the efficiency of an algorithm, the adaptability of an operating system or the form the human/machine interaction should take.

- The prototyping begins with communication.

Prototype Model (CO1)

- Developer and customer meet and define overall objective for the software, identify whatever requirements are known, and outline area where further definition is mandatory. A quick design than occurs. The quick design focuses on the representation of those aspects of the software that will visible to customer/user (e.g., input approaches and output format).

Prototype Model (CO1)



Prototype Model (CO1)

- Reasons for using Prototyping Model

  An important purpose is to illustrate the input data formats, messages, reports, and the interactive dialogues to the customer. This is valuable for gaining better understanding of the customer's needs. The prototype model is very useful in developing the GUI part of the system. The prototype model can be used when the technical solutions are unclear to the development team. Often major design decisions depend on a issues like the response time of a hardware controller or the efficiency of a sorting algorithm.

Prototype Model (CO1)

- The third reason for developing a prototype is that it is impossible to “get in right” the first time and one must plant to throw away the first product in order to develop a good product later.

Advantage of Prototyping Model

- suitable for large system for which there is no manual process to define the requirements.
- User training to use the system
- User service determination
- System training
- Requirement are not freezed

Prototype Model (CO1)

Limitation of Prototyping Model

- It was difficult to find all the requirements of the software initially
- It was very difficult to predict how the system will be after development.

Spiral Model(CO1)

- The Spiral Model The spiral model is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model. It provides the potential for rapid development of increasingly more complete versions of the software. The spiral model is a realistic approach to the development of large scale systems and software. The spiral development model is a risk-driven process model generator that is used to guide multi stakeholder concurrent engineering of software intensive systems.

Spiral Model

- Task set: In this model each of the regions with the set of work tasks is called as task set. The size of the task set will vary according to the project.

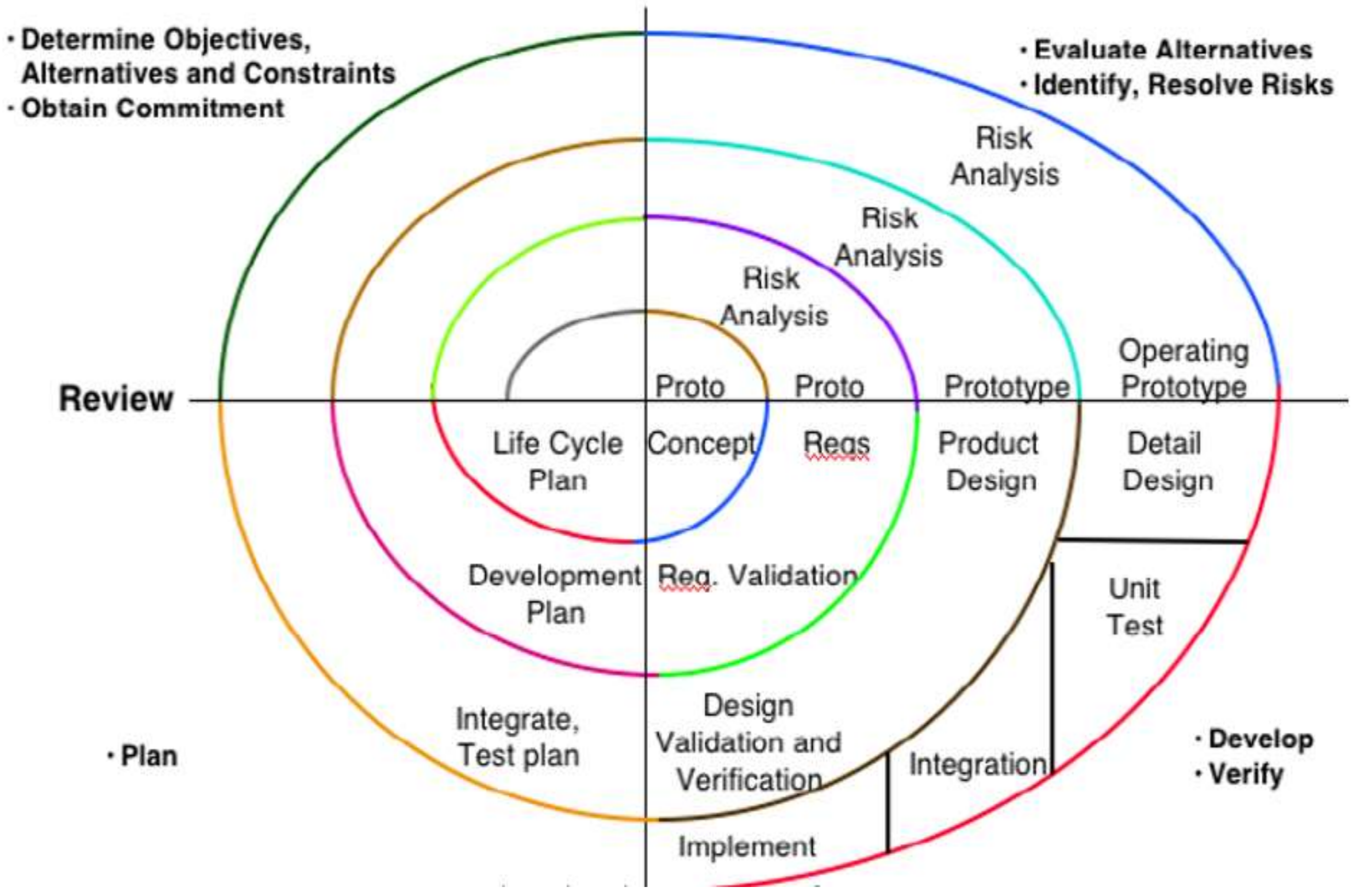
- Task region: Task region is a region where set of task is achieved.

- Principle of spiral model: Elaborate software entity object and find out constraints and alternatives. Elaborate definition of the software entity for the project. Spiral model terminate a project, if it is too risky

Spiral Model

- Customer communication: Task requires establishing effective communication between developer and customer.
- Planning: Task requires defining resources, time& other related information.
- Risk information: Task requires to access both technical and management risk.
- Engineering: Task required building one or more representation of the application.
- Construction &release: The task requires constructing the project & releasing the project to the customer.
- Customer evaluation: Task required obtaining customer feedback based on evolution of the software representation created during the installation stage

Spiral Model



Spiral Model

Advantage:

- User will be able to see the project development cycle.
- Risk analysis which resolves higher priority error.
- Project is very much refined.
- Reusability of the software.

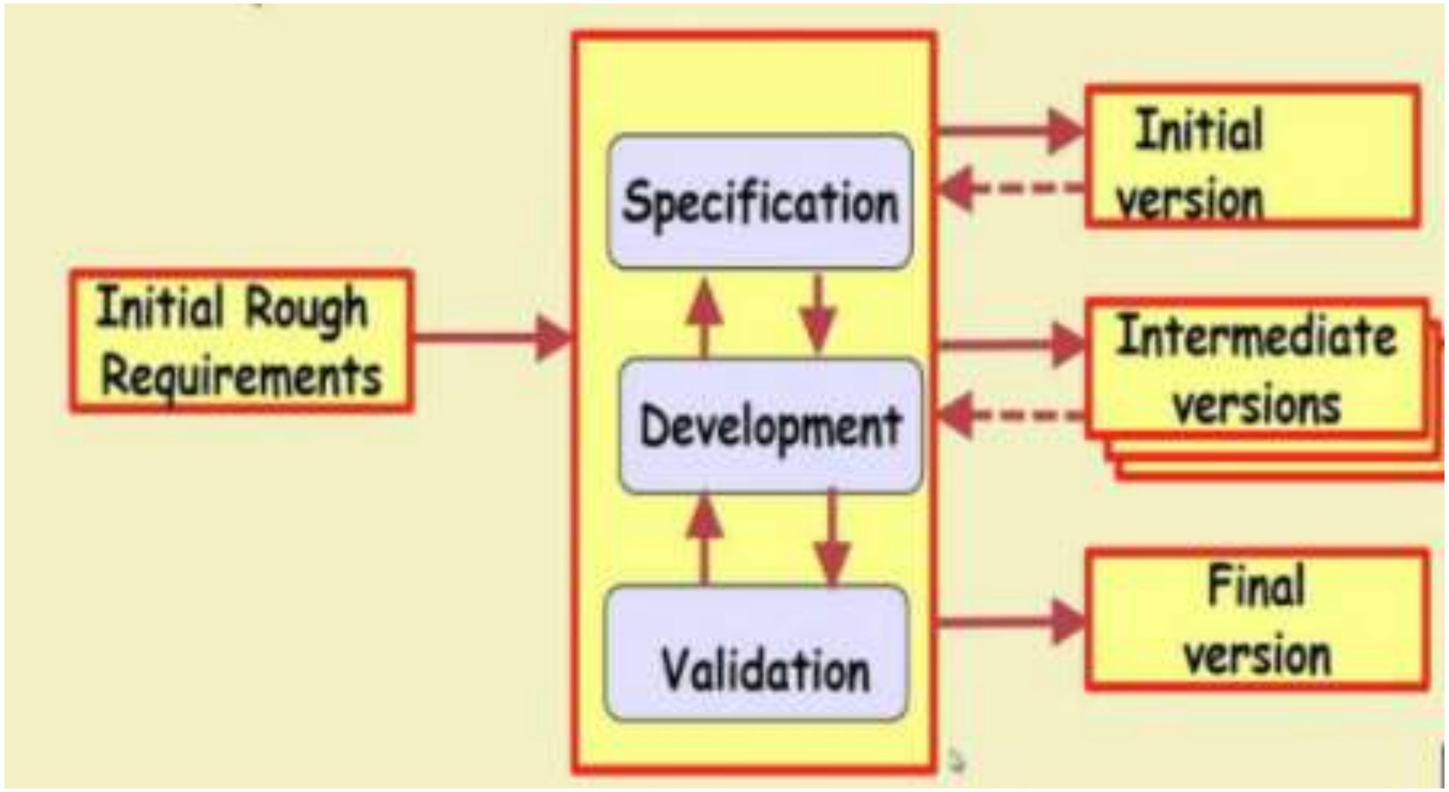
Disadvantages:

- It is only suitable for large size project.
- Model is more complex to use.
- Management skill is necessary so as to analyze the risk factor

Evolutionary development Model

- It is designed to iterative development of s/w project.
- Requirements may be changed in subsequent iterations.
- Waterfall model occur here in cyclical fashion.
- Unlike iterative model it does not require a useable product at the end of each cycle.
- Development, requirement are implemented by category rather then priority.
- **Ex:** in simple database application to implement
    - Cycle1: GUI
    - Cycle2: file Manipulation
    - Cycle3: queries
    - Cycle4: updates

Evolutionary development Model



Evolutionary development Model

- Advantage:
  - Used when it is not necessary to provide minimal version of the system quickly.
  - Model are useful for project using new technology that is not well understood.
  - Used for complex project where all functionality must be delivered at one time, but requirements are unstable or not well understood at the beginning.
- Disadvantage:
  - Not possible to develop fixed cost estimates and static schedule due to uncertainty.
  - Requiring persons whop have the sufficient knowledge to understand and implement the s/w.
  - It is hard to split the problem into several versions that would be acceptable to the customer and which can be incrementally implemented and delivered.

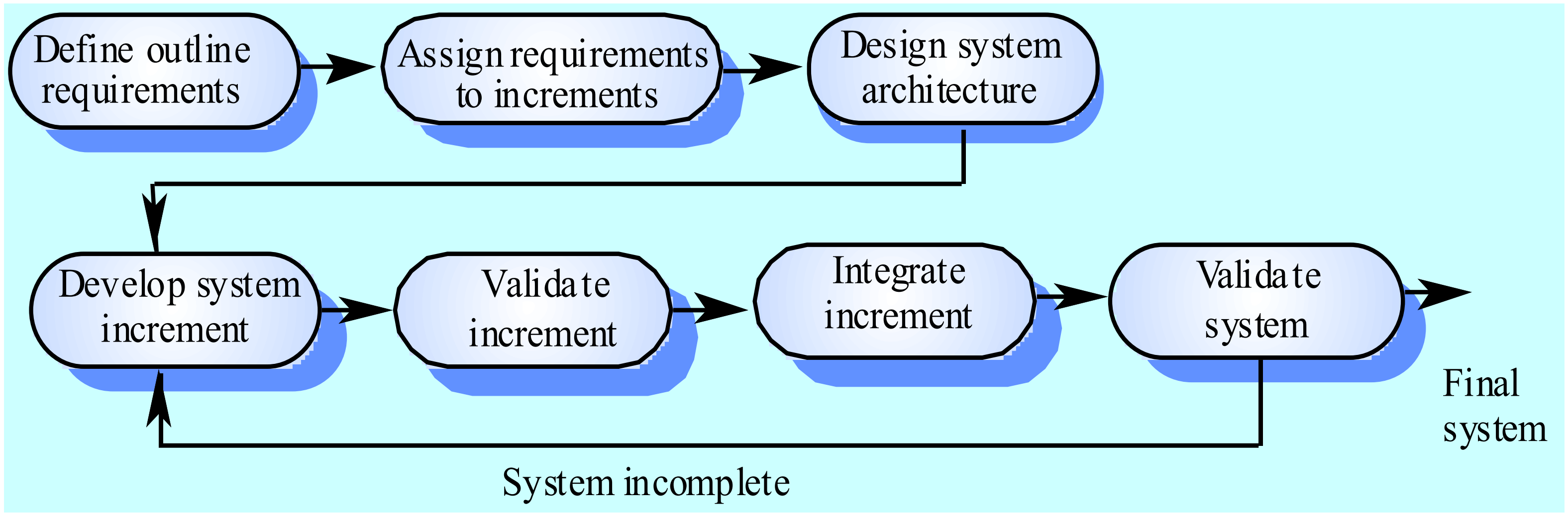
Iterative Enhancement Model(CO1)

The Incremental Model applies linear sequence in a staggered fashion as calendar time progresses. Each linear sequence produces deliverable increments of the software.

For example word processing software developed using the incremental paradigm might deliver basic file management editing, and document production function in first increment; more sophisticated editing, and document production capabilities in the second increment; spelling and grammar checking in the third increment; and advance page layout capabilities in the forth increment.

It should be noted that process flow for any increment may in corporate the prototyping paradigm

Iterative Enhancement Model



Iterative Enhancement Model

- When an incremental model is used, the first increment is often a core product. That is, basic requirements are addressed, but many supplementary feature (some known, other unknown) remain undelivered. The core product is used by the customer.

- As a result of use and/or evaluation, a plan is developed for next increment. The plan addresses the modification of the core product to better meet the need of the customer and the delivery of the additional feature and functionality. This process is repeated following the delivery of each increment ,until the complete product is produced.

Iterative Enhancement Model

- Advantages of Incremental model
  - The feedbacks from early increments improve the later stages
  - User get benefit earlier than with a conventional approach
  - Early delivery of some useful component improves cash flow, because you get
  - some return on investment early on.
  - Smaller sub projects are easier to control and manage

Iterative Enhancement Model

- Disadvantages of Incremental model
  - Software breakage, i.e. later increments may require modification to earlier increments
  - Programmer may be more productive on one large system than on a series of smaller one
  - Some problems are difficult to divide into functional units (modules), which can be incrementally developed and delivered