Unit 1

Software Process

Software:

a) Instructions (Computer Programs) that when executed provide desired features, function, and performance.
b) Data structures that enable the programs to adequately manipulate information.
c) Documents that describe the operation and use of the programs.

Evolutionary Role of Software

Software performs 2 roles

1. **Product** – As a product, it delivers the computing potential embodied by computer hardware or, more broadly, a network of computers that are accessible by local hardware.
2. **Vehicle/ Platform** – As a vehicle used to deliver the product, software acts as the basis for the control of the computer (operating systems), the communication of information (networks), and the creation and control of other programs (software tools and environments).

Functions provided by Software

1. It delivers most important product i.e. Information or Data.
2. It provides the Gateway for world wide information.
3. It provides the means of exchanging information.
4. It manages the business Information.

Software Applications (Categories of Computer Software)

Software applications run on different areas, they are as follows

1. System Software

System software is a collection of programs written to service other programs.
E.g. Compilers, Editors, File Management Utilities.

2. Real-time software

Software that monitors/ analyzes/ controls real-world events as they occur is called real time Software.
E.g. Real time manufacturing process control.

3. Business Software

Business information processing is the largest single software application area. They include software that accesses one or more large databases containing business information.
E.g. Payroll, Inventory, Accounts.

4. Engineering and scientific software

Engineering & Scientific software is the software required for the engineering & scientific development purpose.
It will include all **CASE** tools & System Stimulations.
E.g. LEX, YACC, CAD, CAM.

5. Embedded Software

Embedded software resides in read-only memory and is used to control products and systems for the consumer and industrial markets.
E.g. digital functions in an automobile such as fuel control, dashboard displays, and braking systems

6. Personal Computer Software

These soft wares are used for enhancing the personal computers & to facilitate more control to the user.
E.g. Computer Graphics, Multimedia, Text Editors.

7. Web Based Software

The Web pages retrieved by a browser are software that incorporates executable instructions (e.g., CGI, HTML, Perl, or Java), and data.

8. Artificial Intelligence Software

Software makes use of non numerical algorithms to solve complex problems that are not amenable to computation or straightforward analysis.
E.g. Expert Systems, Pattern Recognition.

Software Engineering

According to Firtz Bauer – Software engineering is defined as "The establishment and use of sound engineering principles in order to obtain the software that is economical, reliable and work efficiently on real machines".

According to Boehm – "Software engineering is the application of science and mathematics by which capabilities of computer equipments are made useful to man via computer programs, procedures and related documentation".

According to IEEE – Software engineering is the systematic approach to the development, operation, maintenance and retirement of the software.

"Software engineering is the technological and managerial discipline concerned with systematic production and maintenance of software products that are developed and modified on time and within cost estimates".

Goals of Software Engineering

The primary goals of Software Engineering is

1. To improve the quality of software.
2. To increase productivity.
3. To increase the job satisfaction of software engineers.

Software engineering is based on the following disciplines

1. Computer Science.
2. Management Science.
3. Economics.
4. Communication Skills.
5. Engineering approach to problem solving.

Software Engineering is a labor intensive activity where high degree of communication is required among

1. Customers.
2. Managers.
3. Software Engineers.
4. Hardware Engineers.
5. Other related technologists.

Terms related to Software Engineering activities.

Size Factors

Project Size: - Project size is a major factor that determines the level of management control and the types of tools and techniques required on a software project. There are certain categories of projects depending on the size.

1) Trivial Projects:-

1. It involves one programmer often working part time.
2. Time spent is few days or few weeks.
3. The program will contain up to 500 lines of code.
4. The software would contain 10 to 20 subroutines
5. They are mostly personal software for exclusive use of the programmer.
6. Very little need for Formal analysis, elaborate design documentation or extensive planning.

2) Small Projects:-

1. Involves one programmer.
2. Duration of 1 to 6 months.
3. The program can contain 1000 to 2000 lines of code.
4. The total project may contain 25 to 50 routines.
5. They usually do not have interaction between other programs.
6. Generally used for small commercial applications including report generation and simple data manipulation.
7. There is negligible interaction between the programmer and customer.

3**) Medium size Projects:-**

1. They involve 2 to 5 programmers.
2. Time spent is between 1 to 2 years.
3. Lines of code range from 10,000 to 50,000 LOC.
4. Number of routines used ranges from 250 to 1000.
5. They include assemblers, compliers, process control systems, inventory systems, small MIS software etc.
6. It requires medium interaction between programmers and customers.
7. A certain degree of formality is required in Planning, Documentation and Project reviews.

4) Large Projects:-

1. Requires 5 to 20 programmers.
2. Project duration lies between 2 to 3 years.
3. It can consist of 50,000 to 100,000 LOC.
4. The entire project is a combination of several subsystems.
5. The modules in the project have significant interaction between other programs and software systems.
6. It requires multilevel management.
7. Examples are large compilers, small time sharing systems, graphics software for data acquisition and display, real time control systems etc.
8. These projects can involve extensive planning, analysis, risk estimation, standardized documentation, formal reviews etc.

5) Very Large Projects:-

1. It may require 100 to 1000 programmers.
2. Its duration may range from 4 to 5 years.
3. The number of software source instructions may reach 10 lakh LOC.
4. The entire project is divided in to several major subsystems that extensively interact with each other.
5. The subsystems are very complex and are developed separately under expert guidance.
6. The prominent feature of these kinds of projects involves real-time processing, telecommunication and multitasking.
7. Examples include large operating systems, Mega database systems, and Military command & control systems.

6) Extremely large projects:-

1. They employ 2000 to 5000 programmers.
2. Time period ranges nearly 10 years.
3. There are 1 million to 10 million LOC.
4. The entire project is an integration of several extremely large systems which further consist of several large subsystems.
5. These projects involve mainly real time processing, telecommunications, multitasking and distributed processing.
6. Examples include Air traffic control systems, Ballistic missile defense systems, Military command & control systems etc.

Quality and Productivity factors during Software Development

Following are the factors that affect the quality and productivity

1) Individual ability
2) Team communication
3) Product complexity
4) Appropriate notations
5) Systematic approaches
6) Controlling changes
7) Level of technology
8) Required reliability
9) Available time
10) Problem understanding
11) Stability of requirements
12) Required skills
13) Facilities and resources
14) Adequacy of training
15) Management skills
16) Appropriate goals
17) Rising expectations

1) Individual ability: - Production and maintenance of software is a labor intensive activity. So these factors are direct functions of individual ability and effort. There are two aspects to ability

a) The general competence of the individual and

b) Familiarity with the particular application area.

2) Team communication: - The new approach towards software development requires many individuals to be involved. Many recent innovations in software engineering, like design reviews, structured walkthroughs and code reading exercises improve communication between the programmers.

3) Product complexity: - There are three generally acknowledged levels of product complexity – a) Application programs, b) Utility programs and c) System level programs. Application programs have the highest productivity and Systems programs have the lowest productivity, measured in terms of LOC per programmer day.

4) Appropriate notations: - The representation schemes are of fundamental importance. Good and standardized notations can clarify the relationships and interactions, of any specific interest.

5) Systematic approach: - Following certain standard systematic approach in procedures and techniques makes the software development process very professional and contribute in improving the quality of the software product.

6) Controlling change: - The factors that contribute the factor of change in software are
a) Design deficiencies in hardware,
b) Change in requirements due to poor understanding of problem,
c) External economic and political factors etc.
To minimize the change the following suggestions can be utilized
a) Use of appropriate notations and standard techniques,
b) Scope of the mechanism to change ie flexibility provided to the software,
c) During planning process the strategies for controlled changes should be included, etc.

7) Level of technology: - As the level of technology is changing rapidly hence modern programming languages should be used during development. Modern programming languages provide –
a) Improved facilities for data definition & data usage,
b) User-defined exception handling,
c) Concurrent programming etc.
Also varied tools are provided such as assemblers and primitive debugging facilities integrated under a single fully integrated development environment.

8) Level of reliability: - High reliability can be achieved by taking great care during –
a) Analysis,
b) Design,
c) System testing and
d) Maintenance.
Both human and machine resources are required to increase reliability. Too much reliability causes decreased productivity when measured in LOC produced Per Programmer Month.

9) Problem understanding: - Failure to understand the problem can result in an unsuccessful project. There are some suggestions for avoiding these problems –
a) Careful planning,
b) Customer interviews,
c) Task observation,
d) Prototyping,
e) Precise product specifications etc.

10) Available time: - Programmer productivity is sensitive to the, calendar time available for project completion. It is observed that the development time can not be compressed below 75 percent of the nominal development time. Extending a project beyond some nominal duration increases the total effort required.

11) Required Skills: - The practice of SE required variety of skills. The requirement definition and design activity are conceptual in nature hence require good creative problem solving skills. Debugging requires deductive or detective kind of skills. Preparations of external documents require good writing and expressive skills. Software Engineers should have good social skills to interact properly with the managers, customers and other engineers.

12) Facilities and resources: - Through various studies it had been found that work-related factors such as –
a) Good machine,
b) Quite place of work,
c) Plenty of access to the machine
d) Creative challenges,
e) Variety of tasks,
f) Opportunities for professional advancement etc, creates more satisfaction among the programmers rather than status factors.

13) Adequacy of training: - It is seen that the freshers from college do not have the following skills, which are very necessary for becoming a successful software engineer –
a) Express one self clearly in English
b) Develop & validate software requirements and design specifications.
c) Work with in application area.
d) Perform software maintenance.
e) Perform economic analysis.
f) Work with project management techniques.
g) Work in groups.
Hence adequate of training is required for entry-level programmers.

14) Management Skills: - As the concept of SE is new hence the managers who work in traditional styles are unable to properly manage the SE activities. Also it is seen that during giving promotions if the competency of the candidate is only measured in the technical point of view then there may arise the same kind of problem. So the

management skills in a candidate should also include technical and managerial competency.

15) Appropriate goals: - Setting of appropriate goals is a major contributing factor in successful execution of a project. Generally the most common goals include
a) Generality,
b) Efficiency and
c) Reliability.

High productivity and quality factors can be achieved by adhering to the goals and requirements established for the software during project planning.

16) Rising expectations: - Progress is constantly being made in the development of tools and techniques to improve software quality and programmer productivity, equally diversity, size and complexity of software applications are growing at a fast rate but also the expectations and increasing demands. Hence it's necessary to adhere to the latest that is happening.

17) Stability of requirements: - The requirements of a software project includes
a) Manpower,
b) Software and
c) Hardware.

The requirements definition is done during problem analysis phase where the amount and type of resources are predicted. In this point if the analysis and prediction is not based on facts, expert guidance and experience from previous projects, then the requirements would change through out the development process and may extend to maintenance phase. Un-stability of requirements increases the cost, degrades overall quality and reduces the productivity. Hence great care should be taken during the initial phase of problem identification & problem analysis.

Other factors: - There are several other factors that influence the productivity –
a) Familiarity to the programming environment,
b) Amount and type of access to the computing system,
c) Stability of the computing system,
d) Memory & timing constrains,
e) Experience in that particular area of interest,
f) Data-base size etc.

Managerial Issues

Ideal or expected activities of Managers in a firm –

1. Managers should control the resources and the environment in which technical activities occurs.
2. They also have ultimate responsibility for ensuring that software products are delivered on time and with in cost estimates.
3. They have the responsibility of ensuring that the software products exhibit the functional and quality attributes desired by the customer.

4. The managers are responsible for project management which includes methods for organizing and monitoring the project progress.
5. Organizing and monitoring a project includes –

a) Cost estimation,
b) Resource allocation policies,
d) Budgetary control,
e) Setting project milestones,
f) Making schedule adjustments,
g) Establishing quality assurance procedures,
h) Establishing effective communication between project members,
i) Customer communications,
j) Development of contractual agreements with customers taking in to view legal obligations and responsibilities.

Software Life Cycle Models (Software Engineering Paradigm)

1. Water Fall Model (Linear Sequential Model)

1) It is a simplest model, which states that the phases are organized in a linear order.
2) The model was originally proposed by Royce.
3) The various phases in this model are

a) **Feasibility Study** – The main aim of the feasibility study activity is to determine whether it would be financially and technically feasible to develop the product. The feasibility study activity involves the analysis of the problem and collection of all relevant information relating to the product such as the different data items which would be input to the system, the processing required to be carried out on these data, the output data required to be produced by the system as well as various constraints on the behavior of the system.

b) **Requirement Analysis** – The aim of the requirement analysis is to understand the exact requirements of the customer and to document them properly. The requirement analysis activity is begun by collecting all relevant data regarding the product to be developed from the users of the product and from the customer through interviews and discussions. During this activity, the user requirements are systematically organized into a software requirement specification (SRS) document. The important components of this document are the functional requirements, the nonfunctional requirements, and the goals of implementation. The SRS document is written in end-user terminology. This makes the SRS document understandable by the customer. After all, it is important that the SRS document be reviewed and approved by the customer.

c) **Design** – This phase is concerned with

1) Identifying software components like functions, data streams and data stores.
2) Specifying software structure.

3) Maintaining a record of design decisions and providing blue prints for the implementation phase
4) The are two categories of Design

a) **Architectural Design** – It involves
   1. Identifying the software components.
   2. Decoupling and decomposing the software components into modules and conceptual data structures.
   3. Specifying the interconnection between the various components.

b) **Detailed Design** – It is concerned with details of the implementation procedures to process the algorithms, data structures and interaction between the modules and data structures. The various activities that this phase includes are
   1. Adaptation of existing code.
   2. Modification of existing algorithms.
   3. Design of data representation and
   4. Packaging of the software product.

   This process is highly influenced by the programming language under the implementation would be carried out. This stage is different from implementation.

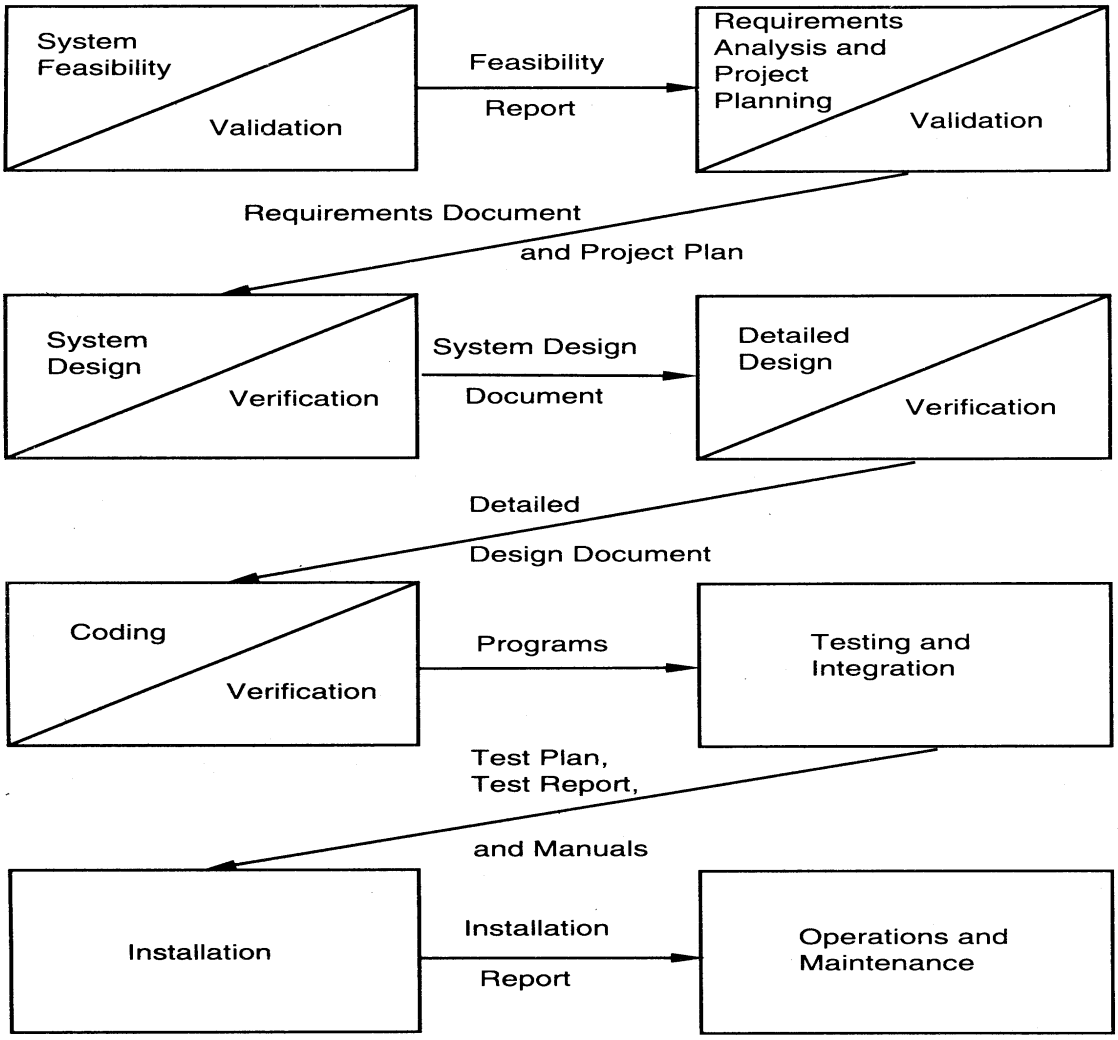
d) **Implementation** – It involves the translation of the design specifications into source code. It also involves activities like debugging, documentation and unit testing of the source code. In this stage various styles of programming can be followed like built-in and user defined data types, secure type checking, flexible scope rules, exception handling, concurrency control etc.

e) **Testing** – It involves two kinds of activities

1. **Integration Testing** – It involves testing of integrating various modules and testing there overall performance due to their integration.
2. **Acceptance Testing** – It involves planning and execution of various types of tests in order to demonstrate that the implemented software system satisfies the requirements stated in the requirements document.

f) **Maintenance** – In this phase the activities include

1. **Corrective Maintenance** – Correcting errors that were not discovered during the product development phase.
2. **Perfective Maintenance** – Improving the implementation of the system, and enhancing the functionalities of the system according to customer's requirements.
3. **Adaptive Maintenance** – Adaptation of software to new processing environments.



Advantages of Water Fall model

1. All phases are clearly defined.
2. One of the most systematic methods for software development.
3. Being oldest, this is one of the time tested models.
4. It is simple and easy to use.

Disadvantages of Water Fall Model

1. Real Projects rarely follow sequential model.
2. It is often difficult for the customer to state all requirements explicitly.
3. Poor model for complex and object oriented projects.
4. Poor model for long and ongoing projects.
5. High amount of risk and uncertainty.
6. Poor model where requirements are at a moderate to high risk of changing.

7. This model is suited to automate the existing manual system for which all requirements are known before the design starts. But for new system, having such unchanging requirements is not possible.

2. Incremental Model

1. The *incremental model* combines elements of the linear sequential model (applied repetitively) with the iterative philosophy of prototyping.
2. It is iterative in nature.
3. It focuses on the delivery of the operational product with each increment
4. The process is repeated until the complete product is produced.
5. For example, word-processing software developed using the incremental paradigm might deliver basic file management, editing, and document production functions in the first increment; more sophisticated editing and document production capabilities in the second increment; spelling and grammar checking in the third increment; and advanced page layout capability in the fourth increment. It should be noted that the process flow for any increment can incorporate the prototyping paradigm.
6. When an incremental model is used, the first increment is often a core product. That is, basic requirements are addressed, but many supplementary features (some known, others unknown) remain undelivered.
7. The core product is used by the customer (or undergoes detailed review). As a result of use and/or evaluation, a plan is developed for the next increment. The plan addresses the modification of the core product to better meet the needs of the customer and the delivery of additional features and functionality.
8. This process is repeated following the delivery of each increment, until the complete product is produced.
9. Incremental development is particularly useful when staffing is unavailable for a complete implementation by the business deadline that has been established for the project.

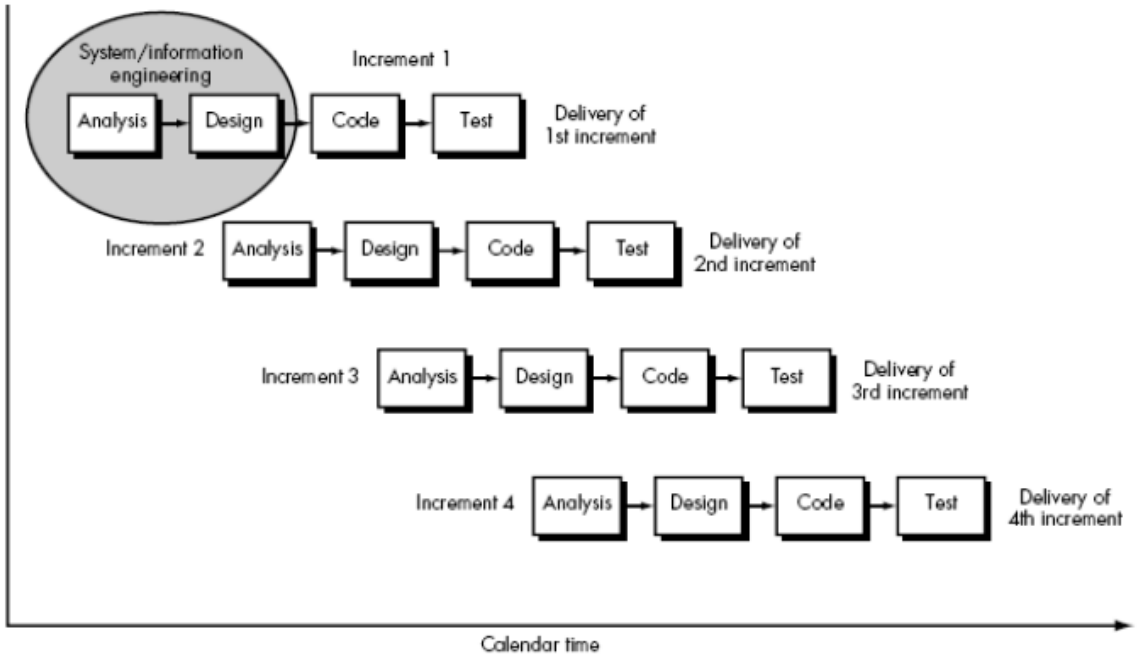
Advantages

1. It provides on the rigid nature of sequential approach.
2. This method is of great help when organization is low on staffing.
3. Generates working software quickly and early during the software life cycle.
4. More flexible – less costly to change scope and requirements.
5. Easier to test and debug during a smaller iteration.
6. Easier to manage risk because risky pieces are identified and handled during its iteration.
7. Each iteration is an easily managed milestone.

Disadvantages

1. This model could be time consuming.
2. Each phase of an iteration is rigid and do not overlap each other.

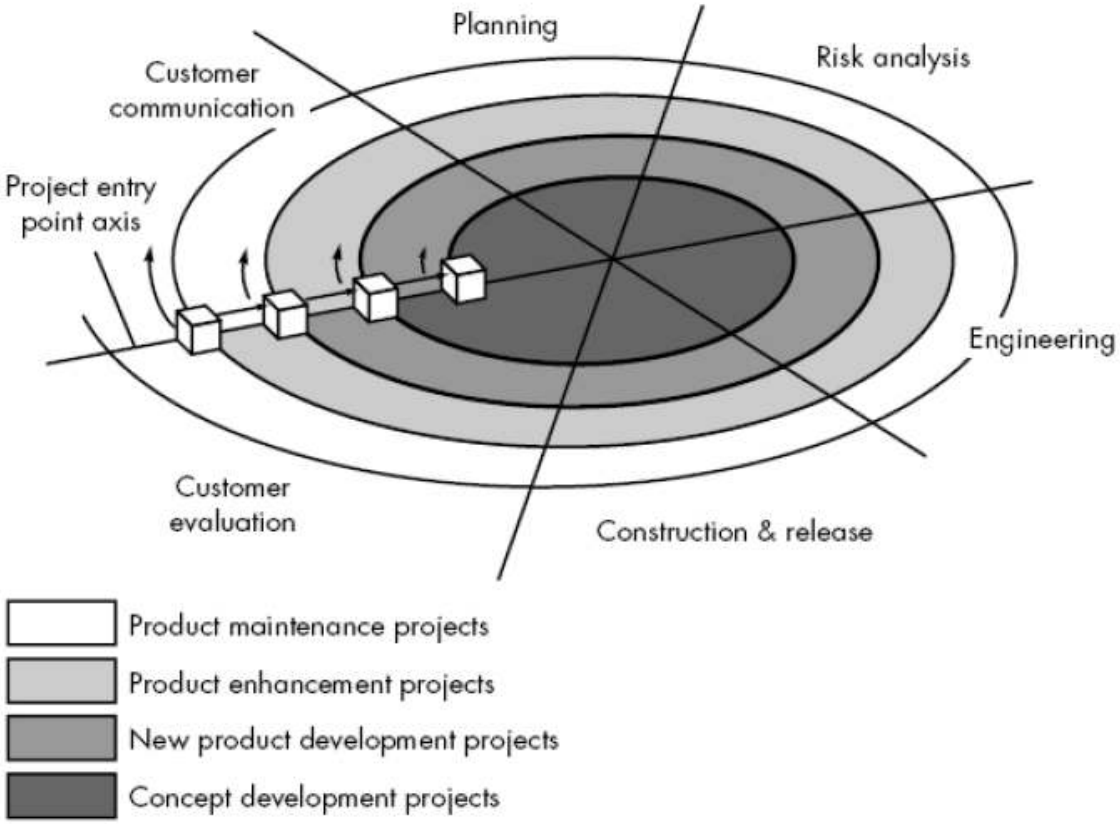
3. Problems may arise pertaining to system architecture because not all requirements are gathered up front for the entire software life cycle.



3. Spiral Model

1. Evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the linear sequential model.
2. It provides the potential for rapid development of incremental versions of the software.
3. Using the spiral model, software is developed in a series of incremental releases. During early iterations, the incremental release might be a paper model or prototype. During later iterations, increasingly more complete versions of the engineered system are produced.
4. A spiral model is divided into a number of framework activities, also called *task regions*.
5. There are six task regions.
   1. Customer communication—tasks required to establish effective communication between developer and customer.
   2. Planning—tasks required to define resources, timelines, and other project related information.
   3. Risk analysis—tasks required to assess both technical and management risks.
   4. Engineering—tasks required to build one or more representations of the application.
   5. Construction and release—tasks required to construct, test, install, and provide user support (e.g., documentation and training).

6. Customer evaluation—tasks required to obtain customer feedback based on evaluation of the software representations created during the engineering stage and implemented during the installation stage.



Advantages:

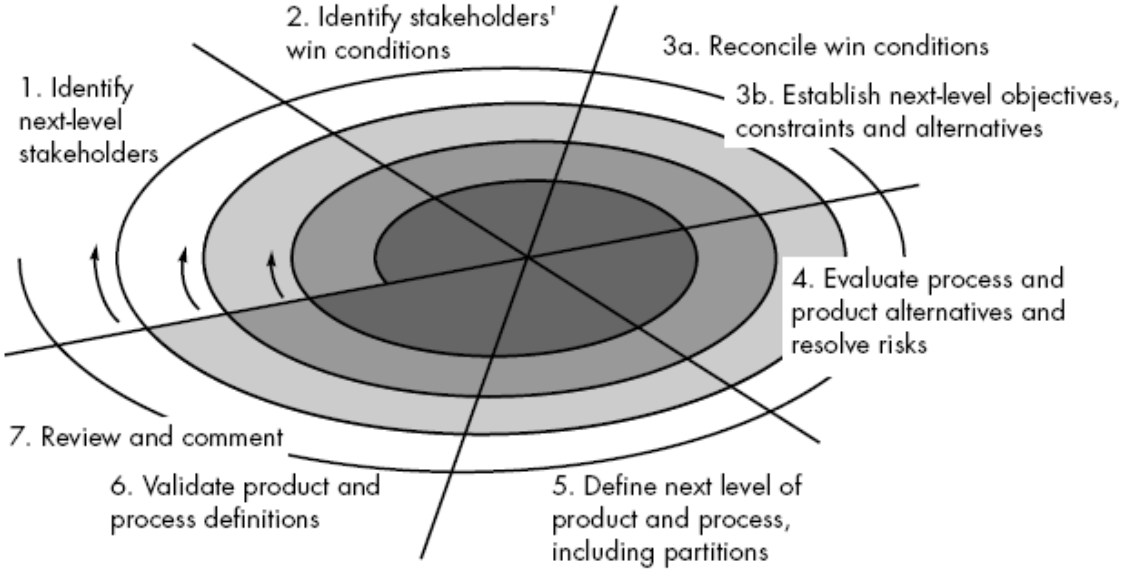
1. It is a realistic approach for development of large scale system.
2. High amount of risk analysis
3. Good for large and mission-critical projects.
4. Software is produced early in the software life cycle.

Disadvantages:

1. It is not widely used.
2. It may be difficult to convince customers (particularly in contract situations) that the evolutionary approach is controllable.
3. It demands considerable risk assessment expertise and relies on this expertise for success. If a major risk is not uncovered and managed, problems will undoubtedly occur.
4. Can be a costly model to use.
5. Risk an alysis requires highly specific expertise.
6. Project's success is highly dependent on the risk analysis phase.
7. Doesn't work well for smaller projects.

4. WINWIN Spiral Model

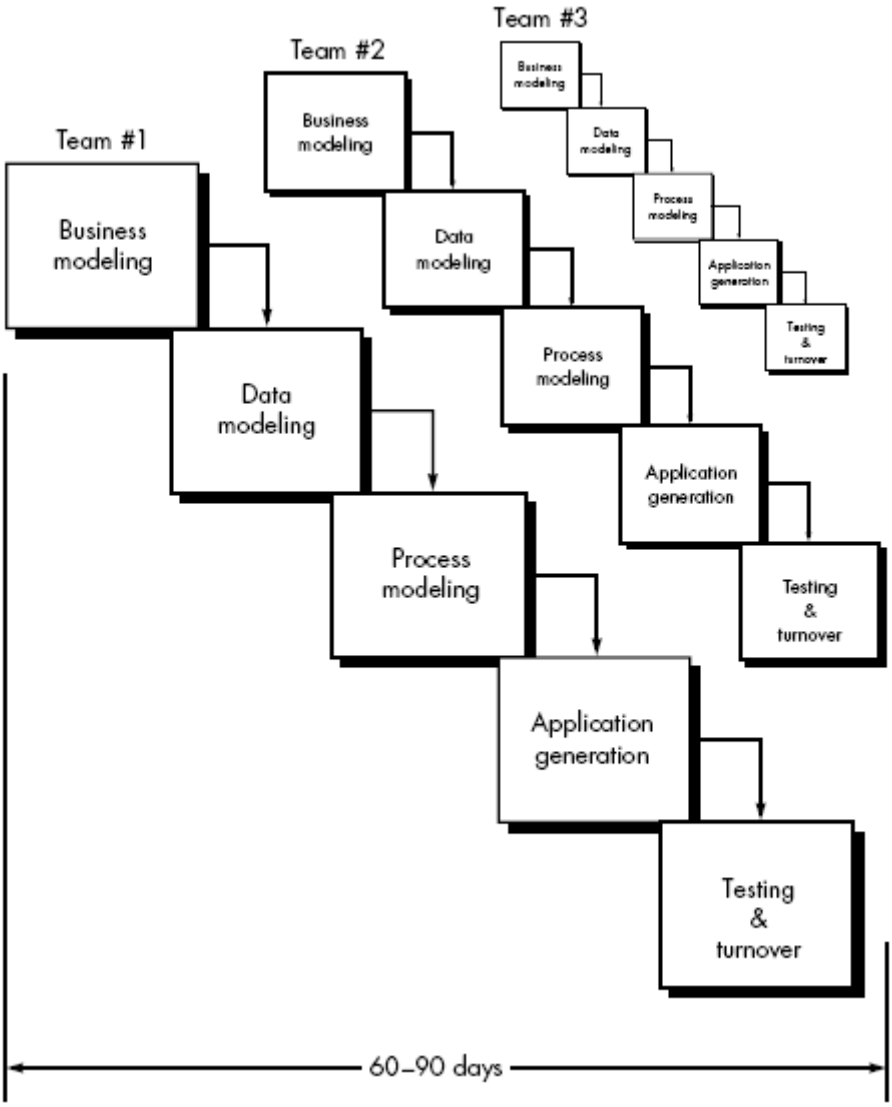
1. This is an adaptation of the spiral model which emphasis is explicitly placed on the involvement of the client in a negotiation process at the genesis of the product development. Ideally, the developer would simply ask the customer what is required and the customer would provide sufficient detail to proceed. Unfortunately this rarely happens and significant negotiations between both parties are required to balance functionality, performance, etc… with cost and time-to-market considerations.
2. First WIN is for customer and second one is for developer.
3. The best negotiations strive for a "WIN-WIN" result. That is, the customer wins by getting the system or product that satisfies the majority of the customer's needs and the developer wins by working to realistic and achievable budgets and deadlines.
4. WINWIN spiral model defines a set of negotiation activities at the beginning of each pass around the spiral. Rather than a single customer communication activity, the following activities are defined.
    a) Identification of the system or subsystem's key "Stake holders".
    b) Determinations of the Stake holder's "WIN conditions".
    c) Negotiations of the Stake holder's WIN conditions to reconcile them into a set of "WIN-WIN" conditions for all concerned (including the software project team).
5. In addition to the emphasis placed on early negotiations, the WINWIN spiral model introduces three process milestones, called anchor points that help establish the completion of one cycle around the spiral and provide decision milestones before the software project proceeds.
6. In essence, the anchor points represent three different views of progress as the project traverses the spiral.
7. The first anchor point, life cycle objectives (LCO), defines a set of objectives for each major software engineering activity. For example, as part of LCO, a set of objectives establish the definition of top-level system/product requirements.
8. The second anchor point, life cycle architecture (LCA), establishes objectives that must be met as the system and software architecture is defined. For example, as part of LCA, the software project team must demonstrate that it has evaluated the applicability of off-the-shelf and reusable software components and considered their impact on architectural decisions.
9. Initial operational capability (IOC) is the third anchor point and represents a set of objectives associated with the preparation of the software for installation/distribution, site preparation prior to installation and assistance required by all parties that will use or support the software.



Advantages

1. Flexibility – The model let the teams adapt to accompanying risks and uncertainties, such as a rapid project schedule and changing team composition.
2. Discipline – The modeling framework was sufficiently formal to maintain focus on achieving three main, or "anchor-point", milestones: The life-cycle objectives, the life cycle architecture, and the initial operational capability.
3. Trust Enhancement – The model provided a means for growing trust among the project stakeholders, enabling them to evolve from adversarial, contact-oriented system development approaches.

5. RAD Model



1. Rapid application development (RAD) is an incremental software development process model that emphasizes an extremely short development cycle.
2. The RAD model is a "high-speed" adaptation of the linear sequential model in which rapid development is achieved by using component-based construction.
3. If requirements are well understood and project scope is constrained, the RAD process enables a development team to create a "fully functional system" within very short time periods (e.g., 60 to 90 days).
4. Used primarily for information systems applications, the RAD approach encompasses the following phases

    a) **Business Modeling: -** This phase involves identification of various information flows between various business functions that act as the driving force in the development process. It's a kind of problem identification task.

b) **Data Modeling: -** The relationships between various data elements are identified. It's like the planning phase where the data which drives the business functions are identified and planned out to establish synchronization so that the designing could be done.
c) **Process Modeling: -** It's like the design phase. Here the properly modeled data are put together such that actual coding to implement the functionalities for data manipulation could be done.
d) **Application generation: -** It's like the coding and implementation phase. Here the planned out processes are actually coded and executed. In this phase it's taken in to consideration that the code should be made is such a manner that they become reusable.
e) **Testing and turnover: -** Here the final code is repeatedly tested for reliability, robustness and maintainability. If some short comings are encountered then again the initial stages are repeated for identification and eradication of errors.

5. In this type of development process a problem is initially analyzed in details such that the problem could be divided or decomposed into manageable smaller problems. The analysis is done in such a way that the decomposed problem could be solved parallel and the solutions could be integrated to solve the entire problem.

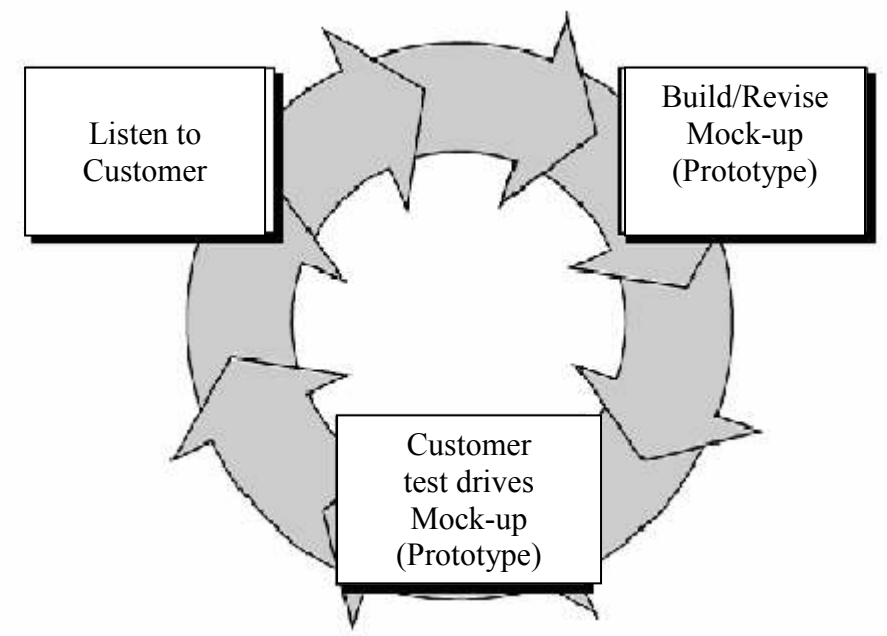
Advantages

1. As RAD stresses on reuse of components testing time may be very less in certain cases.
2. RAD model can be most efficient when it comes to overall development time.

Disadvantages

1. For large but scalable projects, RAD requires sufficient human resources to create the right number of RAD teams.
2. RAD requires developers and customers who are committed to the rapid-fire activities necessary to get a system complete in a much abbreviated time frame. If commitment is lacking from either constituency, RAD projects will fail.
3. Not all types of applications are appropriate for RAD. If a system cannot be properly modularized, building the components necessary for RAD will be problematic. If high performance is an issue and performance is to be achieved through tuning the interfaces to system components, the RAD approach may not work.
4. RAD is not appropriate when technical risks are high. This occurs when a new application makes heavy use of new technology or when the new software requires a high degree of interoperability with existing computer programs.

6. Prototyping Model



1. Often, a customer defines a set of general objectives for software, but does not identify detailed input, processing, or output requirements. In other cases, the developer may be unsure of the efficiency of an algorithm, the adaptability of an operating system, or the form that human-machine interaction should take. In these and many other situations, a prototyping paradigm may offer the best approach.
2. The various phases of this model are
    a) **Listen to Customer: -** This is the starting step, where the developer and customer together
        - Define the overall objectives for the software,
        - Identify the known requirements and
        - The analyst then outlines those factors and requirements that are not visible normally but are mandatory from development point of view.
    b) **Build prototype: -** After the identification of the problem a quick design is done which will cause the software show the output that the customer wants. This quick design leads to the development of a prototype (a temporary working model).
    c) **Customer test drives the prototype: -** The customer runs and checks the prototype for its perfection. The prototype is evaluated by the customer and further improvements are made to the prototype unless the customer is satisfied.

3. All the stages are repeated until the customer gets satisfied. When the final prototype is fully accepted by the customer then final development processes like proper coding for attaining maintainability, proper documentation, testing

for robustness etc. are carried out. And finally the software is delivered to the customer.

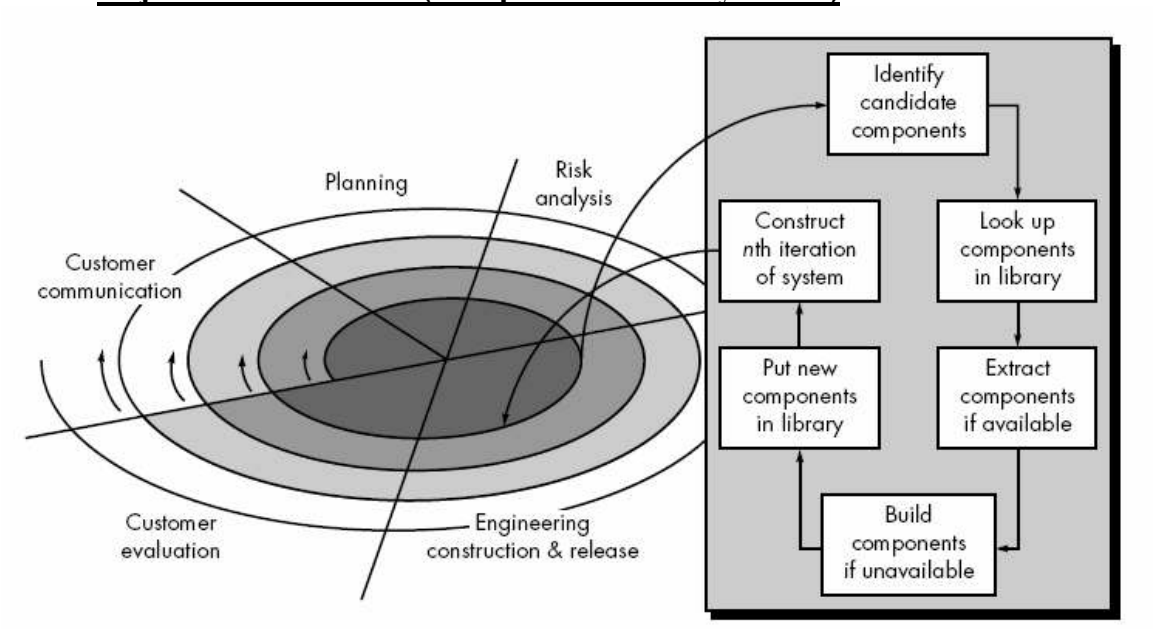
Advantages

1. It serves as the mechanism for identifying the requirements.
2. The developer use the existing program fragment means it helps to generate the software quickly.
3. Continuous developer – Consumer communication is available.

Disadvantages

1. Customer considers the prototype as an original working version of the software.
2. Developer makes implementation compromise in order to get prototype working quickly.
3. This model is time consuming.

7. Object Oriented Model (Component Assembly Model)



1) This kind of model is used for object based (oriented) development process.

2) Here the emphasis is on the creation of reusable classes that encapsulate both data and functions, where the functions can to manipulate the data.

3) This model incorporates many of the characteristics of spiral model and iterative approach towards software development.

4) During the Engineering phase and Construction & release phase, the data is examined and accordingly algorithms applied for data manipulation is also decided. Corresponding data and algorithms are encapsulated into a Class.

5) Classes created in past are stored in a class library. Once the required classes are identified, the classes libraries are searched, if they are found then they are reused. In case where the required classes are not found in the library then they are engineered using object oriented methods.

6) When the process of recognition and usage of classes is done then the development process returns back to the spiral path and subsequently if required reenter in the component assembly process during successive iterations, unless fully acceptable product is made.

Difference between WaterFall model and Spiral model

No.	WaterFall Model	Spiral Model
1	It is a Systematic, sequential approach to software development that begins at system level and progress through analysis, design, coding, testing, and maintenance.	In this model software is developed in a series of incremental versions by the process of iteration through a number of framework activities like Customer communication, planning, risk analysis, engineering, construction engineers and customer evaluation.
2	It is the oldest and the most widely used paradigm in software engineering.	It is an evolutionary approach to software engineering and is currently gaining foothold.
3	It does not include provisions for risk assessment.	Risk assessment is one of the major activities of the spiral model.
4	It is not convenient to model real life large scale projects since most of them do not follow a sequential flow that the model proposes.	Spiral model incorporates the steps of the waterfall model in an interactive framework and hence can more realistically reflect the real world system.
5	This model requires the specification of all requirements at a very early stage in the development, which is often difficult to do.	The Spiral model iterates through the stages of development several times, each time providing a more improved version of the previous stage, additional requirements can be incorporated at later stages of development.
6	The effects of major risks taken can be gauges only at a very later stage of development and by then it would be too terminating the project.	Spiral model requires considerations of risks at all stages of development and if risk are too great, the process can be terminated at a much earlier stage.
7	A working version of the program is available at a much later stage of the project.	A skeleton working model of the system can be developed at earlier stage, which is the refined in successive interaction.

8	No planning is done.	Every pass through planning results in the adjustment of large scale systems of the customer.
9	Entry point is not present.	Entry points are specified.

Q. Explain why the waterfall model of the software process is not an accurate reflection of the software development activities.

Ans – The waterfall model of the software process is not accurate because

1. The linear nature of the classic life cycle leads to "blocking states" in which some project team members must wait for other members of the team to complete development tasks.
2. In fact, the time spent waiting can exceed the time spent on procedure work. The blocking state tends to be more prevalent at the beginning and end of a linear sequential process.
3. Each of these problems is real.
4. However, the classic life cycle paradigm has a definite and important place in software engineering work.
5. It provides a template into which methods for analysis, design, coding, testing, and support can be placed.
6. The classic life cycle remains a widely used procedural model for software engineering.
7. While it does have weakness, it is significantly better than a haphazard approach to software development.
8. The waterfall approach assumes that a complete system will be delivered after the linear sequence is completed.
9. The evolutionary nature of software is not considered in either of these classic software engineering paradigms.

Q. Which of the development process models would you follows for the following projects. Give justifications.

a) A simple data processing project.

Ans: For above project I would like to go for **waterfall model.**

Justification

In the given project, the requirement will be fixed and there is no near chance of changing it. Also for processing data, those basic operations will be fixed and we have delivered all the operations at a time.

In waterfall model, we can very well implement all specified requirements and deliver the whole product at a time.

b) A new system comparing finger prints.

Ans: For above project I would like to go for **spiral model**.

Justification

In the given project, requirement i.e. finger print data will be added continuously so we need to consider it. Even customer feedback i.e. usefulness of the project has to be checked every time. In spiral model, for each set of requirement we can follow up six activities and in turn that spiral way will be continuing till the project is delivered.

c) An online inventory management system for an automobile industry.

Ans: For above project I would like to go for **incremental model**.

Justification

In this project, we will be delivering new services in every increment. Hence, in online MIS system at a time, it is not possible to deliver the entire module and we can not, even delay the delivery. So for this reason, we can consider some features and deliver first increment. Later on, in each increment we can add new features and deliver the entire module increment by increment.

d) A new missile tracking system.

Ans: For this project I would like to go for **waterfall model**.

Justification

Here for missile tracking system the frequency range for tracking missiles will be fixed. All data will be provided at start only. Using this data the entire system has to deliver at a time only so waterfall model is useful here.

e) A satellite launching system. It is not known if the current H/W and S/W technology is nature enough to achieve the goals.

Ans: For this project I would like to go for **prototype model**.

Justification

Satellite launching system is very costly system and current hardware and software are not matching. The basic idea here is that instead of freezing is the requirement before any design or coding can proceed. Throwaway prototype is build to help understand the requirements. Development of the prototype obviously undergoes design, coding and testing but each of these phases is not done very formally or thoroughly.

f) An on-line inventory management for peripheral device manufacturing company.

Ans: For this project I would like to go for **waterfall model**.

Justification

This software is not so much costly. The software team gets all the information from the user and then analysis will be frozen. After the analysis frozen system has developed.

g) A data entry system for office staff that have never used computers before. The user interface and user friendliness are extremely important.

Ans: for this project I would like to go for **increment model**.

Justification

This software is also not so much costly. The basic idea is such that the software is developed in increment, each increment adding some functional capability to the system until the system is implemented. At each step, extensions and design modifications can be made. An advantage of this approach is that it can result in better testing because testing each increment is likely to be easier then testing the entire system as in the waterfall model.

Software Development Process

In the software development process, we have to focus on the activities directly related to production of the software, for example, design, coding, and testing. A development process model specifies some activities that, according to the model, should be performed, and the order in which they should be performed. For cost, quality, and project management reasons, development processes are generally phased.

As the development process specifies the major development and quality assurance activities that need to be performed in the project, the development process really forms the core of the software process. The management process is decided, based on the development process. Due to the importance of development process, various models have been proposed. As processes consist of a sequence of steps, let us first discuss what should be specified for a step.

A Process Step Specification

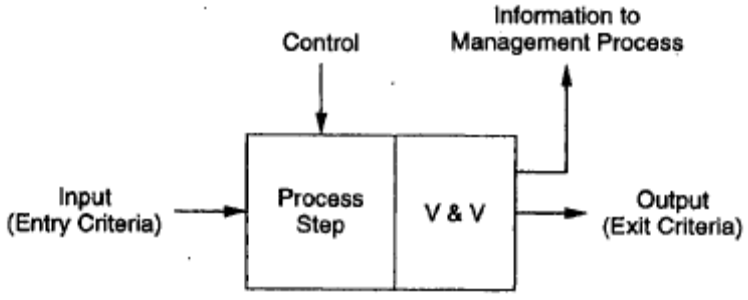
A production process is a sequence of steps. Each step performs a well-defined activity leading towards the satisfaction of the project goals, with the output of one step forming the input of the next one. Most process models specify the steps that need to be performed and the order in which they need to be performed. However, when implementing a process model, there are some practical issues, like when to initiate a step and when to terminate a step, that need to be addressed. Here we discuss some of these issues.

A process should aim to detect defects in the phase in which they are introduced. This requires that there be some verification and validation (V & V) at the end of each step.

(In verification, consistency with the inputs of the phase is checked, while in validation the consistency with the needs of user is checked.) This implies that there is a clearly defined output of a phase, which can be verified by some means and can form input to the next phase (which may be performed by other people). In other words, it is not acceptable to say that the output of a phase is an idea or a thought in the mind of someone; the output must be a formal and tangible entity. Such outputs of a development process, which are not the final output, are frequently called the work products. In software, a work product can be the requirements document, design document, code, prototype, etc. This restriction that the output of each step should be some work product, that can be verified, suggests that the process should have a small number of steps. Having too many steps results in too many work products or documents, each requiring V & V, and can be very expensive. Due to this, at the top level, a development process typically consists of a few steps, each satisfying a clear objective and producing a document used for V & V. How to perform the activity of the particular step or phase is generally not an issue of the development process.

As a development process, typically, contains a sequence of steps, the next issue that comes is when a phase should be initiated and terminated. This is frequently done by specifying the entry criteria and exit criteria for a phase. The entry criteria of a phase specify the conditions that the input to the phase should satisfy in order to initiate the activities of that phase. The output criteria specify the conditions that the work product of this phase should satisfy in order to terminate the activities of the phase. The entry and exit criteria specify constraints of when to start and stop an activity. It should be clear that the entry criteria of a phase should be consistent with the exit criteria of the previous phase.

The entry and exit criteria for a phase in a process depend largely on the implementation of the process. For example, for the same process, one organization may have the entry criteria for the design phase as "requirements document signed by the client" and another may have "no more than X errors detected per page in the requirement review." As each phase ends with some V & V activity, a common exit criteria for a phase is "V & V of the phase completed satisfactorily," where satisfactorily is defined by the organization based on the objectives of the project and its experience in using the process. The specification of a step with its input, output, and entry exit criteria is shown in Figure below.



A step in the development process

Verification & Validation (V&V)

The goals of verification and validation activities are to asses and improve the quality of the work products generated during development and modification of software. The quality attributes include – a) Correctness, b) Completeness, c) Consistency, d) Reliability, e) Usefulness, f) Conformance to standards and g) overall cost effectiveness.

Verification: - It's the process of determining whether the product is built in a right manner or not. There are two categories of verification: -
a) Life-cycle verification and
b) Formal verification.

a) Life-Cycle verification: - It's the process of determining the degree to which the work products of a given phase of the development cycle fulfill the specifications established during prior phases.
b) Formal verification: - It's a rigorous mathematical demonstration that whether the source code conforms to the specification.

Validation: - It's the process of evaluation of the software at the end of the software development process to determine compliance with the requirements. In other words it's the process of determining whether the correct product is built or not.

Verification and validation involve the assessment of work products to determine conformance to the specifications. Specifications include –

a) Requirements specification
b) The design documentation
c) Various stylistic guidelines
d) Implementation language standards
e) Project standards
f) Organizational standards
g) User expectations.

Verification

1. Verification is done to ensure that the work product of a given phase of the development cycle fulfill the specifications established during prior phase.
2. According to Boehm :

   Verification: Are we producing the product right?

3. Verification ensures the product is designed to deliver all functionality to the customer; it typically involves reviews and meetings to evaluate documents, plans, code, requirements and specifications; this can be done with checklists, issues lists, and walkthroughs and inspection meetings.

4. In verification uncovering of defects will be done in primary ways. Here no code will be executed. Before building actual system this checking will be done. This process will be called Quality Assurance.
5. Verification is nothing but the Static Testing.
6. Inputs to the verification are check list, issue list, walkthroughs and inspection meetings, reviews and meetings.
7. The output of the verification is a nearly a perfect set of documents, plans, specifications and requirements document.
8. According to the CMM, Validation - The process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements.

Validation

1. Validation is the process of evaluating software at the end of the software development process to determine compliance with the requirements.
2. According to Boehm:

   Validation: Are we producing the right product?

3. Validation ensures that functionality, as defined in requirements, is the intended behavior of the product; validation typically involves actual testing and takes place after verifications are completed.
4. Validation concern, checking will be done by executing code for errors (defects.). This can be called as Quality Control.
5. Validation is nothing but the Dynamic Testing.
6. The input on the validation on the other hand is the actual testing of an actual product.
7. The output of the validation on the other is a nearly perfect, actual product.
8. According to CMM, Verification- The process of evaluating software to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.

Difference between Verification and Validation

No.	Verification	Validation
1	Verification is done to ensure that the work product of a given phase of the development cycle fulfill the specifications established during prior phase.	Validation is the process of evaluating software at the end of the software development process to determine compliance with the requirements.
2	Verification: Are we producing the product right?	Validation: Are we producing the right product?
3	Verification ensures the product is designed to deliver all functionality to	Validation ensures that functionality, as defined in requirements, is the intended

	the customer; it typically involves reviews and meetings to evaluate documents, plans, code, requirements and specifications; this can be done with checklists, issues lists, and walkthroughs and inspection meetings.	behavior of the product; validation typically involves actual testing and takes place after verifications are completed.
4	In verification uncovering of defects will be done in primary ways. Here no code will be executed. Before building actual system this checking will be done. This process will be called Quality Assurance.	Validation concern, checking will be done by executing code for errors (defects). This also can be called as Quality Control.
5	Verification is nothing but the Static Testing.	Validation is nothing but the Dynamic Testing.
6	Verification is done before validation.	Validation cannot be done before verification.
7	Inputs to the verification are check list, issue list, walkthroughs and inspection meetings, reviews and meetings.	The input on the validation on the other hand is the actual testing of an actual product.
8	The output of the verification is a nearly a perfect set of documents, plans, specifications and requirements document.	The output of the validation on the other is a nearly perfect, actual product.

System Engineering

System

1. A set or arrangement of things so related as to form a unity or organic whole.
2. A set of facts, principles, rules, etc., classified and arranged in an orderly form so as to show a logical plan linking the various parts.
3. A method or plan of classification or arrangement.
4. An established way of doing something, method, procedure….

Computer Based System

"A set of arrangements of elements that are organized to accomplish some predefined goal by processing information."

The goal may be to support some business function or to develop a product that can be said to generate business revenue. To accomplish the goal, a computer based system makes use of a variety of system elements:

Software – Computer programs, data structures, and related work products that serve to affect the logical method, procedure, or control that is required.

Hardware – Electronic devices that provide computing capability, the interconnectivity devices (e.g., network switches, telecommunications devices) that enable the flow of data, and electromechanical devices (e.g., sensors, motors, pumps) that provide external world function.

People – Users and operators of hardware and software.

Database – A large, organized collection of information that is accessed via software and persists over time.

Documentation – Descriptive information (e.g., models, specifications, hardcopy manuals, online help files, web sites) that portrays the use and/or operation of the system.

Procedures – The steps that define the specific use of each system element or the procedural context in which the system resides.

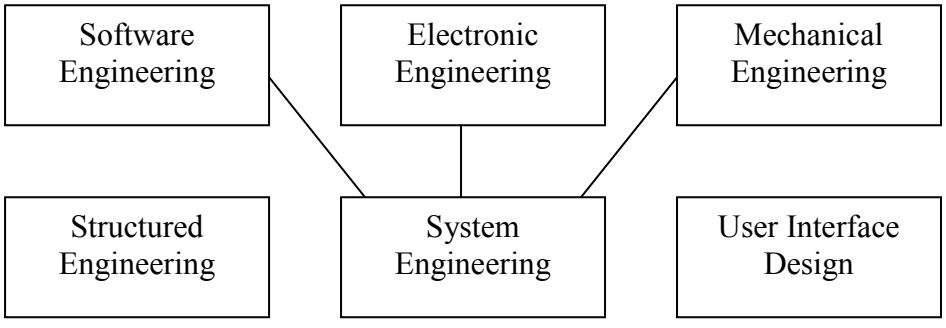
These elements combine in a variety of ways to transform information. For example, a marketing department transforms raw sales data into a profile of the typical purchaser of a product; a robot transforms a command file containing specific instructions into a set of control signals that cause some specific physical action. Creating an information system to assist the marketing department and control software to support the robot both require system engineering.

System Engineering

"System engineering is an activity of specifying, designing, implementing, validating, deploying and maintaining technical system."

System engineers are not just concerned with software but also with hardware and the system's interactions with users and its environment. They must think about the services that the system provides the constraints under which the system must be built and operated and the ways in which the system is used to fulfill its purpose.

Difference between system Engineering and software Engineering



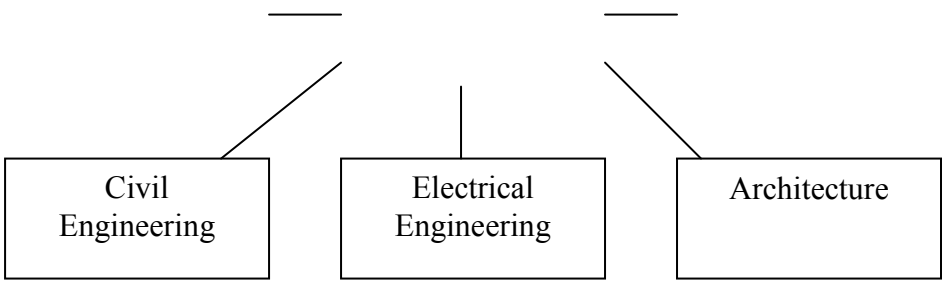


Fig Disciplines involved in systems engineering

There are important distinctions between the system engineering process and software development process

1. **Limited scope of rework during system development** – Once some system engineering decisions have been made, they are very expensive to change. Reworking the system design to solve these problems is rarely possible. One reason software has become so important in systems is that it allows changes to be made during system development, in response to new requirements.

2. **Interdisciplinary involvement** – Many engineering disciplines may be involved in system engineering. There is a lot of scope for misunderstanding because different engineers terminology and conventions.

System engineering hierarchy

System engineering encompasses a collection of top-down and bottom-up methods to navigate the hierarchy illustrated in fig (a). The system engineering process usually begins with a "world view". That is, the entire business or product domain is examined to ensure that the proper business or technology context can be established. The world view is refined to focus more fully on a specific domain of interest. Within a specific domain, the need for targeted system elements (e.g., data, software, hardware, and people) is analyzed. Finally, the analysis, design, and construction of a targeted system element is initiated. At the top of the hierarchy, very broad contexts are established and, at the bottom, detailed technical activities, performed by the relevant engineering discipline (e.g., hardware or software engineering), are conducted.

Stated in a slightly more formal manner, the world view (WV) is composed of a set of domain (D_j), which can each be a system or system of systems in its own right.

WV = {D1, D2, D3… Dn)

Each domain is composed of specific elements (Ej) each of which serves some role in accomplishing the objective and goals of the domain or component:

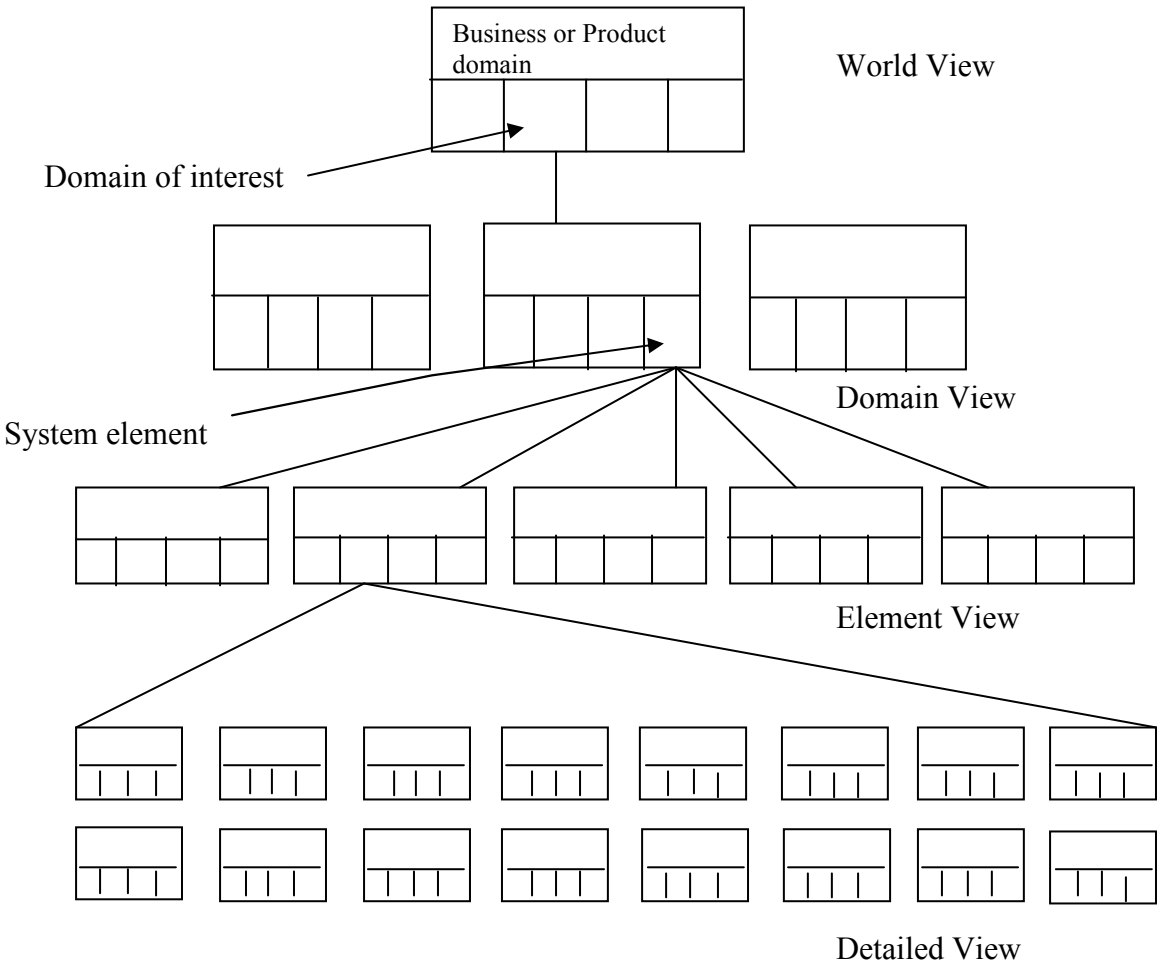
Di = {E1, E2, E3… En}

Finally, each element is implemented by specifying the technical components (Ck) that achieve the necessary function for an element:

Ej = {C1, C2, C3… Cn}

In the software context, a component could be a computer program, a reusable program component, a module, a class or object, or even a programming language statement.

It is important to note that the system engineer narrows the focus of work as she moves downward in the hierarchy. However, the world view portrays a clear definition of overall functionality that will enable the engineer to understand the domain, and ultimately the system or product, in the proper context.



Fig(a) System Engineering Hierarchy

Q. Explain the approach used by SEI (Software Engineering Institute) to determine the current state of process maturity of an organization.

Ans: The SEI approach provides a measure of the global effectiveness of a company's software engineering practices and establishes five process maturity levels that are defined in the following manner:

Level 1: Initial.
The software process is characterized as ad hoc and occasionally even chaotic. Few processes are defined, and success depends on individual effort.

Level 2: Repeatable.
Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.

Level 3: Defined.
The software process for both management and engineering activities is documented, standardized, and integrated into an organization wide software process. All projects use a documented and approved version of the organization's process for developing and supporting software. This level includes all characteristics defined for level 2.

Level 4: Managed.
Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled using detailed measures. This level includes all characteristics defined for level 3.

Level 5: Optimizing.
Continuous process improvement is enabled by quantitative feedback from the process and from testing innovative ideas and technologies. This level includes all characteristics defined for level 4.

The five levels defined by the SEI were derived as a consequence of evaluating responses to the SEI assessment questionnaire that is based on the CMM. The results of the questionnaire are distilled to a single numerical grade that provides an indication of an organization's process maturity.

The SEI has associated key process areas (KPAs) with each of the maturity levels. The KPAs describe those software engineering functions (e.g., software project planning, requirements management) that must be present to satisfy good practice at a particular level. Each KPA is described by identifying the following characteristics:

1. ***Goals***—the overall objectives that the KPA must achieve.
2. ***Commitments***—requirements (imposed on the organization) that must be met to achieve the goals or provide proof of intent to comply with the goals.

3. ***Abilities***—those things that must be in place (organizationally and technically) to enable the organization to meet the commitments.
4. ***Activities***—the specific tasks required to achieve the KPA function.
5. ***Methods for monitoring implementation***—the manner in which the activities are monitored as they are put into place.
6. ***Methods for verifying implementation***—the manner in which proper practice for the KPA can be verified.

Eighteen KPAs (each described using these characteristics) are defined across the maturity model and mapped into different levels of process maturity. The following KPAs should be achieved at each process maturity level:

1. **Process maturity level 2**
    a) Software configuration management
    b) Software quality assurance
    c) Software subcontract management
    d) Software project tracking and oversight
    e) Software project planning
    f) Requirements management

2. **Process maturity level 3**
    a) Peer reviews
    b) Intergroup coordination
    c) Software product engineering
    d) Integrated software management
    e) Training program
    f) Organization process definition
    g) Organization process focus

3. **Process maturity level 4**
    a) Software quality management
    b) Quantitative process management

4. **Process maturity level 5**
    a) Process change management
    b) Technology change management
    c) Defect prevention

Each of the KPAs is defined by a set of *key practices* that contribute to satisfying its goals. The key practices are policies, procedures, and activities that must occur before a key process area has been fully instituted. The SEI defines *key indicators* as "those key practices or components of key practices that offer the greatest insight into whether the goals of a key process area have been achieved."