PRINCIPLES OF SOFTWARE ENGINEERING

Unit-1: INTRODUCTION

The term **software engineering** is composed of two words, software and engineering.

Software is more than just a program code. A program is an executable code, which serves some computational purpose. Software is considered to be a collection of executable programming code, associated libraries and documentations. Software, when made for a specific requirement is called software product.

Engineering on the other hand, is all about developing products, using well-defined, scientific principles and methods.

Software Engineering:

So, we can define software engineering as an engineering branch associated with the procedures. The outcome of software engineering is an efficient and reliable software product.

Development of software product using well-defined scientific principles, methods and documentation.

IEEE defines software engineering as:

The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software.

The Nature of Software:

1. **Software is developed or engineered; it is not manufactured in the classical sense.**
   Although some similarities exist between software development and hard-ware manufacturing, the two activities are fundamentally different. In both activities, high quality is achieved through good design.
2. **Although the industry is moving toward component-based construction, most software continues to be custom built.**
   A software component should be designed and implemented so that it can be reused in many different programs. Modern reusable components encapsulate both data and the processing that is applied to the data, enabling the software engineer to create new applications from reusable parts.
3. **Software is easy to modify.**
   People make changes without fully understanding it.
4. **Software does not 'wear out'**
   Hardware components suffer from the cumulative effects of dust, vibration, temperature extremes, and many other environmental maladies. Stated simply, the hardware begins to wear out.

   Software deteriorates by having its design changed. When a hardware component wears out, it is replaced by a spare part. There are no software spare parts. Every software failure indicates an error in design or in the process through which design was translated into machine executable code. Therefore, the software maintenance tasks that accommodate requests for change involve considerably more complexity than hardware maintenance.

The Unique Nature of Webapps:

1. **Network intensiveness**: A WebApp resides on a network and must serve the needs of a diverse community of clients. The network may enable world-wide access and communication (i.e., the Internet).
2. **Concurrency**: A large number of users may access the WebApp at one time. In many cases, the patterns of usage among end users will vary greatly.
3. **Unpredictable load:** The number of users of the WebApp may vary by orders of magnitude from day to day. One hundred users may show up on Monday, 10,000 may use the system on Thursday.
4. **Performance:** If a WebApp user must wait too long (for access, for server-side processing, for client-side formatting and display), he or she may decide to go elsewhere.
5. **Availability:** 24*7 availability.
6. **Data driven:** The primary function of many WebApps is to use hypermedia to present text, graphics, audio, and video content to the end user.
7. **Security:** Provide high security.

Software application domains:

1. **System software**: A collection of programs written to service other programs. Some system software (e.g., compilers, editors, and file management utilities) processes complex, but determinate, information structures. Other systems applications (e.g., operating system components, drivers, networking software, telecommunications processors) process largely indeterminate data.
2. **Application software:** Application software is a set of stand-alone programs that solve a specific business need.
3. **Engineering/scientific software**: Applications range from astronomy to volcanology, from automotive stress analysis to space shuttle orbital dynamics etc.
4. **Embedded software:** resides within a product or system and is used to implement and control features and functions for the end user and for the system itself. e.g., key pad control for a microwave oven, washing machine etc.
5. **Product-line software:** designed to provide a specific capability for use by many different customers. e.g., computer graphics, multimedia, entertainment, database management, and personal and business financial applications.

The Software Process:

Process: A process is a collection of activities, actions, and tasks that are performed when some work product is to be created.

Activity: An activity strives to achieve a broad objective (e.g., communication with stakeholders) and is applied regardless of the application domain, size of the project, complexity of the effort.

Action: An action (e.g., architectural design) encompasses a set of tasks that produce a major work product (e.g., an architectural design model).

1. A process framework establishes the foundation for a complete software engineering process by identifying a small number of framework activities that are applicable to all software projects, regardless of their size or complexity.
2. A generic process framework for software engineering encompasses five activities:
   - Communication
   - Planning
   - Modelling
   - Construction
   - Deployment
3. In addition, the process framework encompasses a set of umbrella activities that are applicable across the entire software process.
4. In general, umbrella activities are applied throughout a software project and help a software team manage and control progress, quality, change, and risk. Typical umbrella activities include:
   - Software project tracking and control
   - Risk management
   - Software quality assurance
   - Technical reviews
   - Measurement
   - Software configuration management
   - Reusability management
   - Work product preparation and production

NEED OF SOFTWARE ENGINEERING

The need of software engineering arises because of higher rate of change in user requirements and environment on which the software is working.

1. **Large software**: It is easier to build a wall than to a house or building, likewise, as the size of software become large engineering has to step to give it a scientific process.
2. **Scalability:** If the software process were not based on scientific and engineering concepts, it would be easier to re-create new software than to scale an existing one.
3. **Cost:** As hardware industry has shown its skills and huge manufacturing has lower down the price of computer and electronic hardware. But the cost of software remains high if proper process is not adapted.
4. **Dynamic Nature:** The always growing and adapting nature of software hugely depends upon the environment in which the user works. If the nature of software is always changing, new enhancements need to be done in the existing one. This is where software engineering plays a good role.

5. **Quality Management:** Better process of software development provides better and quality software product.

Software Engineering Practice:

1. **The Essence of Practice:**
   A. Understand the problem (communication and analysis).
   B. Plan a solution (modeling and software design).
   C. Carry out the plan (code generation).
   D. Examine the result for accuracy (testing and quality assurance).

A. Understand the problem:

Understand the problem contains the following questions.

- Who has a stake in the solution to the problem? That is, who are the stakeholders? (user, customer, developer, development manager)
- What are the unknowns? What data, functions, and features are required to properly solve the problem?
- Can the problem be represented graphically? Can an analysis model be created?

B. Plan the solution:

Plan the solution contains the following problems.

- Has a similar problem been solved? If so, are elements of the solution reusable?
- Can sub problems be defined? If so, are solutions readily apparent for the sub problems?
- Can you represent a solution in a manner that leads to effective implementation? Can a design model be created?

C. Carry out the plan:

The design you've created serves as a road map for the system you want to build. The "plan" will allow you to proceed without getting lost.

- Is source code traceable to the design model?
- Have the design and code been reviewed, or better, have correctness proofs been applied to the algorithm?

D. Examine The Result:

You can't be sure that your solution is perfect, but you can be sure that you've designed a sufficient number of tests to uncover as many errors as possible.

- Has a reasonable testing strategy been implemented?
- Does the solution produce results that conform to the data, functions, and features that are required? Has the software been validated against all stakeholder requirements?

A GENERIC PROCESS MODEL:

A software process model is a simplified representation of a software process. Each model represents a process from a specific perspective.

These generic models are abstractions of the process that can be used to explain different approaches to the software development. They can be adapted and extended to create more specific processes.

A generic process framework for software engineering defines five framework activities—communication, planning, modeling, construction, and deployment.

Process Flow: It describes how the frame-work activities and the actions and tasks that occur with respect to sequence and time.

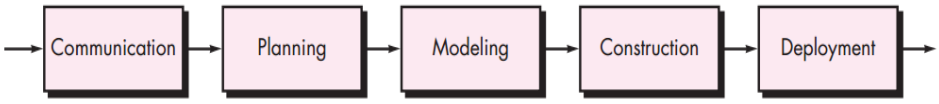
Framework Activity is populated by a set of software engineering actions.

software engineering action is defined by a task set that identifies the work tasks that are to be completed, the work products that will be produced.

Task: It consumes resources (equipment, time, persons etc) and it produces a work product (e.g. models, documents, system etc).

Different process flows listed below:

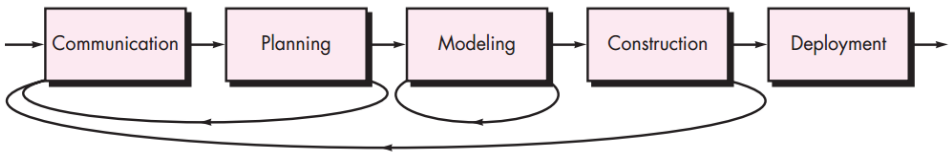
1. **Linear process flow**



(a) Linear process flow

A linear process flow executes each of the five framework activities in sequence, beginning with communication and finishing with deployment.

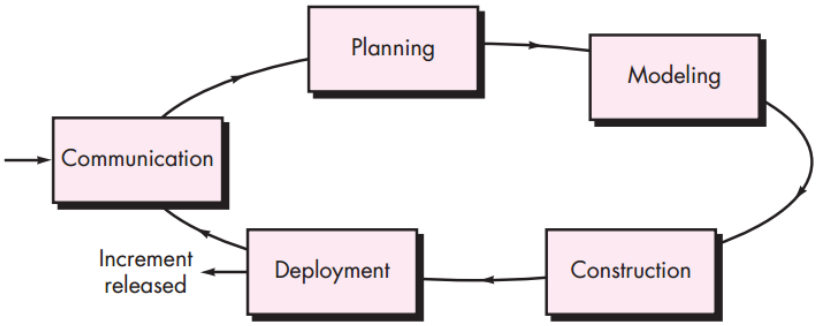
2. **Iterative process flow**



(b) Iterative process flow

An iterative process flow repeats one or more of the activities before proceeding to the next.

3. **Evolutionary process flow**

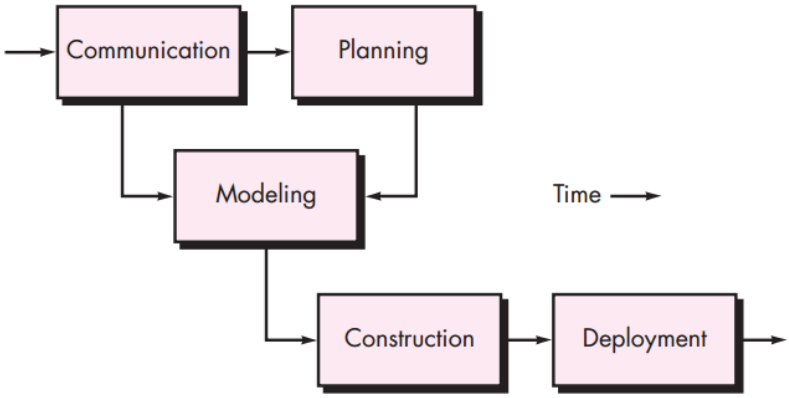


(c) Evolutionary process flow

An evolutionary process flow executes the activities in a "circular" manner.

4. Parallel process flow

A parallel process flow executes one or more activities in parallel with other activities.



(d) Parallel process flow

Difference between process flow and life cycle:

- Process flow is a step by step detailed list of tasks within a framework.
- Life cycle mainly functions as a visual aid and shows the high level stages a client would go through. Each stage explains with particular interface.

Process Assessment and Improvement:

1. Standard CMMI Assessment Method for Process Improvement (SCAMPI):

The Standard CMMI Assessment Method for Process Improvement (SCAMPI) was developed to satisfy the CMMI (CMMI stands for **C**apability **M**aturity **M**odel **I**ntegration) model requirements.

SCAMPI provides a five-step process assessment model that incorporates five phases:

- Initiating
- Diagnosing
- Establishing
- Acting and
- Learning

The SCAMPI method uses the SEI CMMI (Software Engineering Institute CMMI, 2000) as the basis for assessment.

SEI: Software Engineering Institute (SEI) at Carnegie Mellon University in Pittsburgh, USA.

CMMI: CMMI is the successor of the CMM and evolved as a more matured set of guidelines and was built combining the best components of individual disciplines of

CMM (Software CMM, People CMM etc). It can be applied to product manufacturing, People management, Software development etc.

2. **CMM-Based Appraisal for Internal Process Improvement (CBA IPI):**

CBA-IPI tool is used in an organization to gain insight into the software development capability. For this, the strengths and weaknesses of the existing process are identified in order to prioritize software improvement plans and focus on software improvements, which are beneficial to the organization.

It provides a diagnostic technique for assessing the relative maturity of a software organization. It uses the SEI CMM as the basis for the assessment.

The organization's software process capability is assessed by a group of individuals known as the **assessment team,** which generates findings and provides ratings according to the **CMM** (Capability Maturity Model). These findings are collected from questionnaires, document reviews and interviews with the managers of the organization.

3. **SPICE (ISO/IEC15504):**

A standard that defines a set of requirements for software process assessment. The intent of the standard is to assist organizations in developing an objective evaluation of the efficacy of any defined software process.

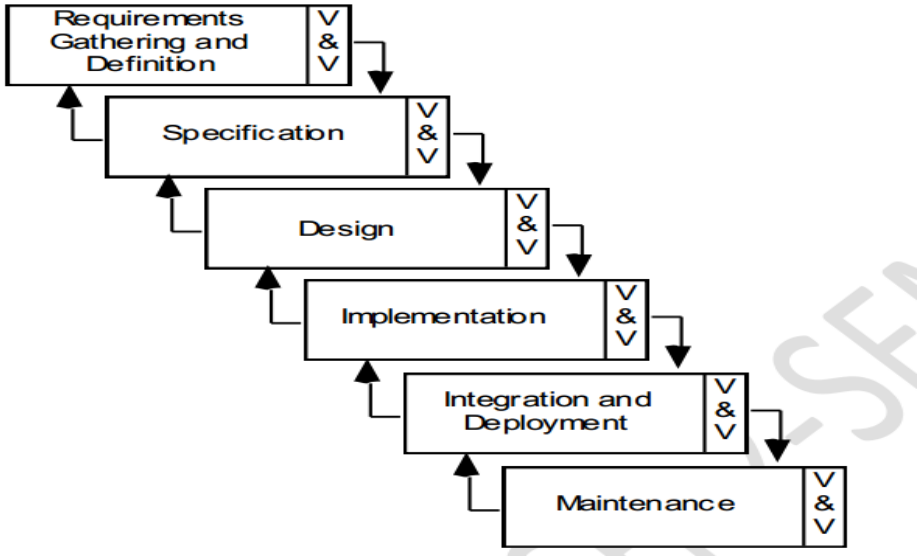
4. **ISO 9001:2000 for Software:**

A generic standard that applies to any organization that wants to improve the overall quality of the products, systems, or services that it provides. Therefore, the standard is directly applicable to software organizations and companies.

PRESCRIPTIVE PROCESS MODELS:

1. Prescriptive process models were originally proposed to bring order to the chaos (disorder) of software development.
2. The edge of chaos can be visualized as an unstable, partially structured state. It is unstable because it is constantly attracted to chaos or to absolute order.

I. The Waterfall Model:



- The model suggests that software engineers should work in a series of stages.
- Before completing each stage, they should perform quality assurance (verification and validation).
- In this Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.
- **Requirement Gathering and specification**: All possible requirements of the system to be developed are captured in the requirement gathering and analyze the requirements for clear and perfect model. The details documented in a requirement specification document.
- **System Design:** This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.
- **Implementation:** With inputs from the system design, the system is first developed in small programs called units.
- **Integration and Testing:** All the units developed in the implementation phase are integrated into a system after testing of each unit. After integration the entire system is tested for any faults and failures.
- **Deployment of system:** Once the functional and non-functional testing is done, the product is deployed in the customer environment or released into the market.
- **Maintenance:** There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released.

Applications:

1. Requirements are very well documented, clear and fixed.
2. Product definition is stable.
3. Technology is understood and is not dynamic.
4. There are no ambiguous requirements.
5. The project is short.

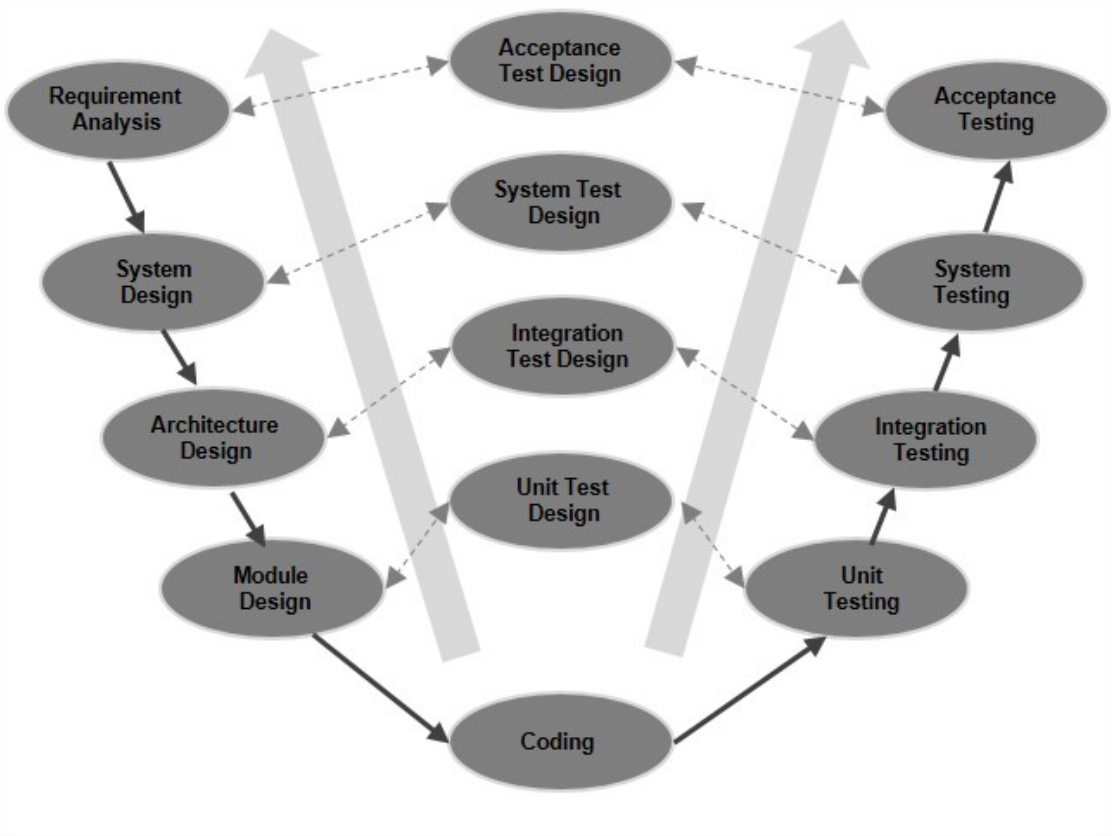
Advantages:

1. Simple and easy to understand and use.
2. Phases are processed and completed one at a time.
3. Works well for smaller projects where requirements are very well understood.

Disadvantages:

1. No working software is produced until late during the life cycle.
2. Not a good model for complex and object-oriented projects.
3. Poor model for long and ongoing projects.
4. It is difficult to measure progress within stages.

II. V-MODEL:



- The V-model is an SDLC model where execution of processes happens in a sequential manner in a V-shape.
- It is also known as **Verification and Validation model**.
- The V-Model is an extension of the waterfall model and is based on the association of a testing phase for each corresponding development stage.
- This means that for every single phase in the development cycle, there is a directly associated testing phase.
- This is a highly-disciplined model and the next phase starts only after completion of the previous phase.
- Under the V-Model, the corresponding testing phase of the development phase is planned in parallel. So, there are Verification phases on one side of the 'V' and Validation phases on the other side. The Coding Phase joins the two sides of the V-Model.
- A software team moves down the left side of the V, basic problem requirements are refined into progressively more detailed and technical representations of the problem and its solution.
- Once code has been generated, the team moves up the right side of the V, essentially performing a series of tests (quality assurance actions) that validate each of the models created as the team moved down the left side.
- In reality, there is no fundamental difference between the classic water fall model and the V-model.

Validation Phases: The different Validation Phases in a V-Model are explained in detail below.

- **Unit Testing:** Unit testing is the testing at code level and helps eliminate bugs at an early stage, though all defects cannot be uncovered by unit testing.
- **Integration Testing:** Integration testing is associated with the architectural design phase. Integration tests are performed to test the coexistence and communication of the internal modules within the system.
- **System Testing:** System testing is directly associated with the system design phase. System tests check the entire system functionality.
- **Acceptance Testing:** Acceptance testing is associated with the business requirement analysis phase and involves testing the product in user environment.

III. Incremental Model:

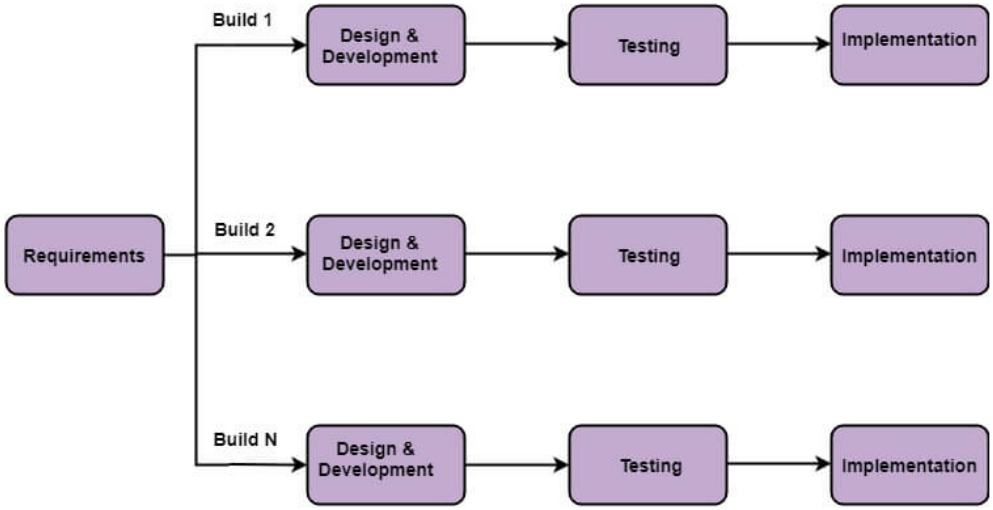


Fig: Incremental Model

- Incremental Model is a process of software development where requirements divided into multiple standalone modules of the software development cycle.
- In this model, each module goes through the requirements, design, implementation and testing phases.
- When an incremental model is used, the first increment is often a core product. Core system implementing only a few basic features is built and then that is delivered to the customer. The core product is used by the customer.
- 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. This process is repeated following the delivery of each increment, until the complete product is produced.
- Every subsequent release of the module adds function to the previous release. The process continues until the complete system achieved.

Example:

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.

Advantages of Incremental Model:

1. Errors are easy to be recognized.
2. Easier to test and debug
3. More flexible.
4. Simple to manage risk because it handled during its iteration.
5. The Client gets important functionality early.

Disadvantage of Incremental Model:

1. Need for good planning
2. Total Cost is high.
3. Well defined module interfaces are needed.

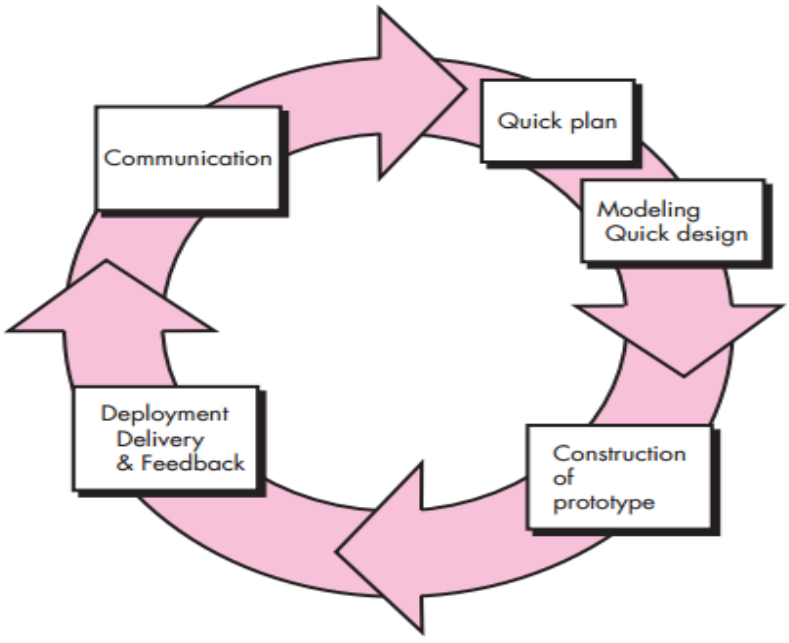
IV. Evolutionary Process Models

- Evolutionary models are iterative type models.
- They allow to develop more complete versions of the software.

Following are the evolutionary process models.

1. The prototyping model
2. The spiral model
3. Concurrent development model

1. The Prototyping Model:



- The Prototyping Model is one of the most popularly used Software Development Life Cycle Models (SDLC models).
- This model is used when the customers do not know the exact project requirements beforehand.
- In this model, a prototype of the end product is first developed, tested and refined as per customer feedback repeatedly till a final acceptable prototype is achieved which forms the basis for developing the final product.
- The process starts by interviewing the customers and developing the incomplete high-level paper model.
- This document is used to build the initial prototype supporting only the basic functionality as desired by the customer.
- Once the customer figures out the problems, the prototype is further refined to eliminate them.
- The process continues till the user approves the prototype and finds the working model to be satisfactory.

The prototyping paradigm begins with communication. You meet with other stakeholders to define the overall objectives for the software, identify whatever requirements are known, and outline areas where further definition is mandatory. Prototyping iteration is planned quickly, and modeling occurs. A quick design focuses on a representation of those aspects of the software that will be visible to end users.

The quick design leads to the construction of a prototype. The prototype is deployed and evaluated by stakeholders, who provide feedback that is used to further refine requirements. Iteration occurs as the prototype is tuned to satisfy the needs of various stakeholders.

Advantages:

- New requirements can be easily accommodated as there is scope for refinement.
- Missing functionalities can be easily figured out.
- The developed prototype can be reused by the developer for more complicated projects in the future.
- Flexibility in design.

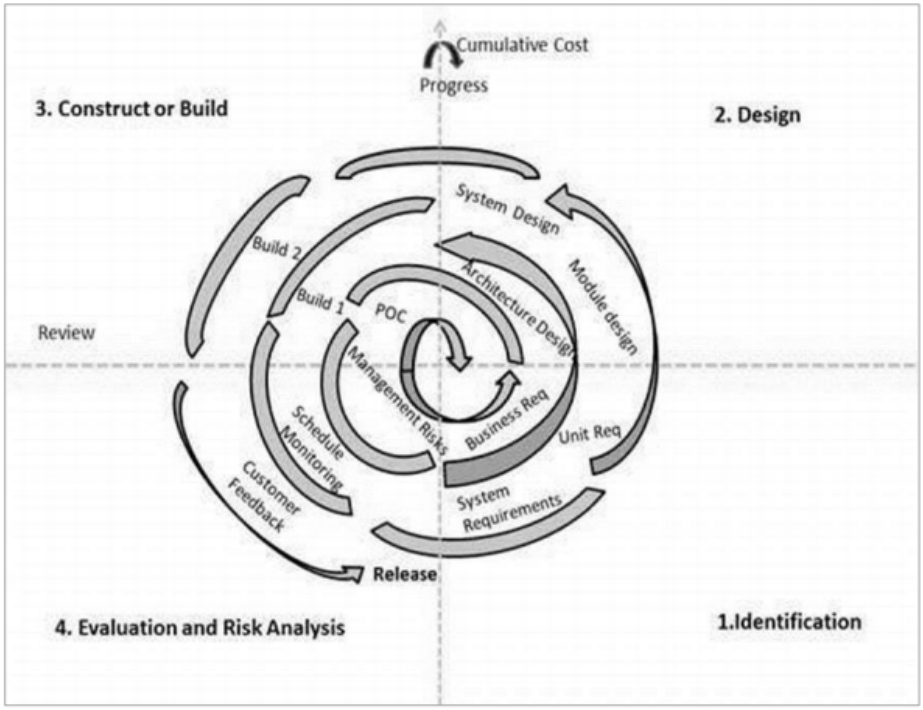
Disadvantages:

- Costly with respect to time as well as money.
- Poor Documentation due to continuously changing customer requirements.
- It is very difficult for the developers to accommodate all the changes demanded by the customer.
- The customer might lose interest in the product if he/she is not satisfied with the initial prototype.

2. The Spiral Model:

- The spiral model combines the idea of iterative development with the systematic, controlled aspects of the waterfall model.
- This Spiral model is a combination of iterative development process model and sequential linear development model i.e. the waterfall model with a very high emphasis on risk analysis.

- It allows incremental releases of the product or incremental refinement through each iteration around the spiral.



The spiral model has four phases. A software project repeatedly passes through these phases in iterations called **Spirals**.

i. Identification
ii. Design
iii. Construct or Build
iv. Evaluation and Risk Analysis

i. Identification:

a) This phase starts with gathering the business requirements in the baseline spiral.
b) In the subsequent spirals as the product matures, identification of system requirements, subsystem requirements and unit requirements are all done in this phase.
c) This phase also includes understanding the system requirements by continuous communication between the customer and the system analyst.

ii. Design:

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

iii. Construct or Build:

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

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

iv. Evaluation and Risk Analysis:

a) Risk Analysis includes identifying, estimating and monitoring the technical feasibility and management risks, such as cost overrun etc.

b) After testing the build, at the end of first iteration, the customer evaluates the software and provides feedback.

Uses:

1. For medium to high-risk projects.
2. When there is a budget constraint and risk evaluation is important.
3. Requirements are complex and need evaluation to get clarity.
4. Significant changes are expected in the product during the development cycle.

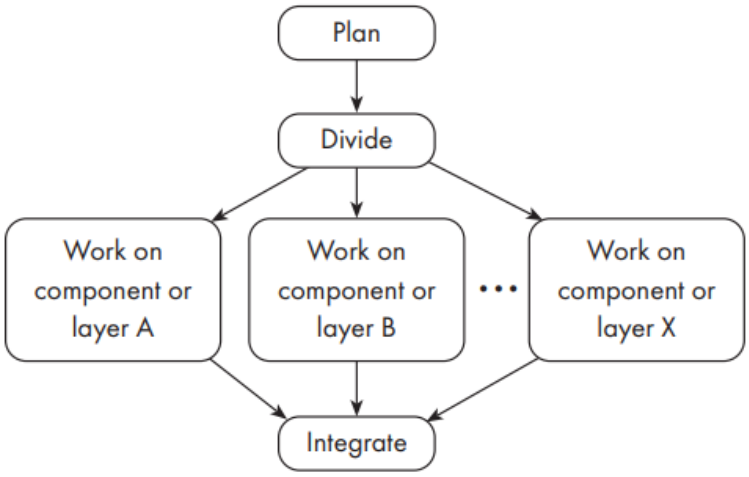
Advantages:

1. Changing requirements can be accommodated.
2. Requirements can be captured more accurately.
3. Development can be divided into smaller parts and the risky parts can be developed earlier which helps in better risk management.

Disadvantages:

1. Management is more complex.
2. Not suitable for small or low risk projects and could be expensive for small projects.
3. Process is complex.
4. Spiral may go on indefinitely.
5. Large number of intermediate stages requires excessive documentation.

3. Concurrent Development Model:



The concurrent engineering model

- The concurrent engineering model explicitly accounts for the divide and conquer principle. Each team works on its own component.

- In the concurrent engineering model, there has to be some initial planning, and periodic integration. The diagram therefore only shows a picture of a single iteration.
- Concurrent modeling is applicable to all types of software development and provides an accurate picture of the current state of a project.
- It defines a process network. Each activity, action, or task on the network exists simultaneously with other activities, actions, or tasks. Events generated at one point in the process network trigger transitions among the states.

SPECIALIZED PROCESS MODELS:

Specialized process models take on many of the characteristics of one or more of the traditional models presented in the preceding sections.

Types in specialized process models:

1. Component-Based Development (Promotes reusable components)
2. The formal methods model (Mathematical formal methods are backbone here)
3. Aspect Oriented Software Development (Uses Cross Cutting Technology)

1. Component-Based Development:

- Commercial off-the-shelf (COTS) software components, developed by vendors who offer them as products, provide targeted functionality with well-defined interfaces.
- The component-based development model incorporates many of the characteristics of the spiral model.
- Modeling and construction activities begin with the identification of candidate components. These components can be designed as either conventional software modules or object-oriented classes or packages of classes.
- The component-based development model leads to software reuse, and reusability provides software engineers with a number of measurable benefits.
- The component-based development model incorporates the following steps (implemented using an evolutionary approach):
  i) Available component-based products are researched and evaluated for the application domain in question.
  ii) Component integration issues are considered.
  iii) A software architecture is designed to accommodate the components.
  iv) Components are integrated into the architecture.
  v) Comprehensive testing is conducted to ensure proper functionality.

2. The formal methods model:

- The formal methods model encompasses a set of activities that leads to formal mathematical specification of computer software.
- Formal methods enable you to specify, develop, and verify a computer-based system by applying a rigorous (perfect), mathematical notation.

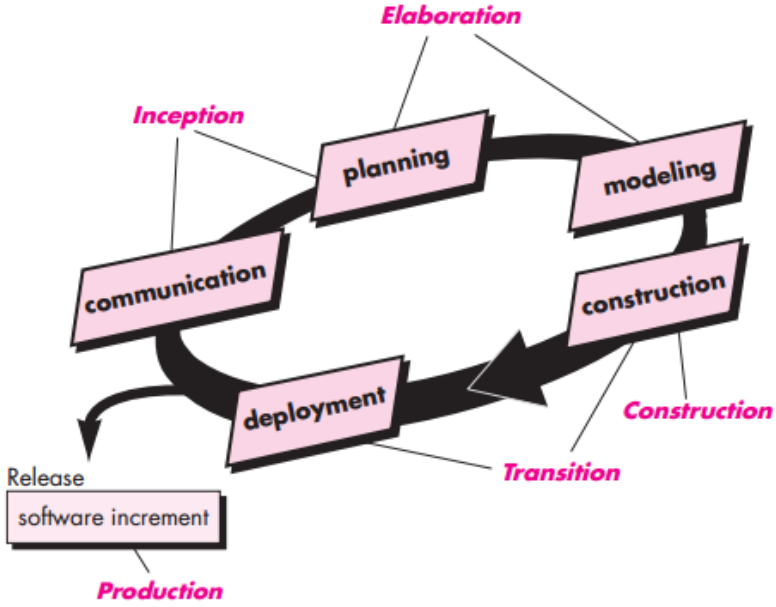
- When formal methods are used during development, they provide a mechanism for eliminating many of the problems that are difficult to overcome using other software engineering paradigms. Ambiguity, incompleteness, and inconsistency can be discovered and corrected more easily, not through ad hoc review, but through the application of mathematical analysis.
- Although not a mainstream approach, the formal methods model offers the promise of defect-free software. Which includes the following steps:
    i) The development of formal models is currently quite time consuming and expensive.
    ii) Because few software developers have the necessary background to apply formal methods, extensive training is required.
    iii) It is difficult to use the models as a communication mechanism for technically unsophisticated customers.
- The formal methods approach has gained adherents (followers or supporters) among software developers who must build safety-critical software (e.g., developers of aircraft avionics and medical devices) and among developers that would suffer severe economic hardship should software errors occur.

3. <u>Aspect Oriented Software Development:</u>

- Modern computer-based systems become more sophisticated (and complex), which contain certain concerns.
- Concern is a particular set of information that has an effect on code. Concerns are not program issues but reflect the system requirements and priorities of the system stakeholders.
- **Examples of concerns**: Performance, security, profiling, logging, transaction management, authentication, error-checking, memory management etc.
- When concerns cut across multiple system functions, features, and information, they are often referred to as crosscutting concerns.
- Often cannot be cleanly decomposed from the rest of the system which results in scattering (disorganise) or tangling (confuse).
- Aspect oriented Programming is designed to handle cross cutting concerns by providing a mechanism known as aspect. Aspect is the key unit.
- Aspect-oriented software development (AOSD), often referred to as aspect oriented programming (AOP), is a relatively new software engineering paradigm that provides a process and methodological approach for defining, specifying, designing, and constructing aspects.
- Cross cutting concerns allow us to reuse the components. Advantages by using cross cutting concerns are quick development, specialization, enabling/disabling aspects at runtime.
- A distinct aspect-oriented process has not yet matured. However, it is likely that such a process will adopt characteristics of both evolutionary and concurrent process models.

- **AOSD Tools:** Spring, AspectJ, XWeaver, Aspect#, AspectC++ etc.

The Unified Process:



- The Unified Process recognizes the importance of customer communication and streamlined methods for describing the customer's view of a system (usecase view).
- It emphasizes the important role of software architecture.
- It helps the architect focus on the right goals, such as understandability, reliance to future changes, and reuse.
- It suggests a process flow that is iterative and incremental, providing the evolutionary feel that is essential in modern software development.

Phases of the Unified Process: It contains 5 phases.

1. Inception
2. Elaboration
3. Construction
4. Transition
5. Production

1. Inception Phase:

- The inception phase of the UP includes both customer communication and planning activities.
- A vision of the product is created.
- By collaborating with stakeholders, business requirements for the software are identified and a rough architecture for the system is proposed.
- Fundamental business requirements are described through a set of preliminary usecases (describes preliminary functions).
- This phase contains vision document, initial usecase model, initial architecture, project plan, initial risk assessment, prototypes etc.

2. Elaboration phase:

- The elaboration phase includes the planning and modeling activities of the generic process model.

- Elaboration improves and expands the preliminary use cases that were developed as part of the inception phase.
- Expands the architectural representation to include five different views of the software such as the usecase model, the requirements model, the design model, the implementation model, and the deployment model.
- This phase includes usecase model, non functional requirements, analysis model, software architecture description, executable architectural prototype, revised risk list, workflows etc.

3. **Construction phase:**
   - The construction phase of the UP is identical to the construction activity defined for the generic software process.
   - Using the architectural model as input, the construction phase develops the software components.
   - All necessary and required features and functions for the software increment (i.e., the release) are implemented in source code.
   - As components are being implemented, unit tests are designed and executed for each. In addition, integration activities (component assembly and integration testing) are conducted.
   - This phase includes design model, software components, test plan, test cases, and support documentation etc.
4. **Transition phase:**
   - The transition phase of the UP includes the final stages of the generic construction activity and the first part of the generic deployment (delivery and feedback) activity.
   - Software is given to end users for beta testing and user feedback reports both defects and necessary changes.
   - The software team creates the necessary support information such as user manuals, troubleshooting guides, installation procedures required for the release.
5. **Production phase:**
   - The production phase of the UP coincides with the deployment activity of the generic process.
   - During this phase, the ongoing use of the software is monitored, support for the operating environment (infrastructure) is provided, and defect reports and requests for changes are submitted and evaluated.

PERSONAL AND TEAM PROCESS MODELS:

1. **Personal Software Process (PSP):**
   - The Personal Software Process (PSP) is a structured software development process that is designed to help software engineers to better understand and improve their performance

.

- The PSP model defines five framework activities:
    1. Planning
    2. High Level Design
    3. High Level Design Review
    4. Development
    5. Postmortem

1. Planning:

- This activity isolates requirements and develops both size and resource estimates.
- In addition, a defect estimate (the number of defects projected for the work) is made.
- All metrics are recorded on worksheets or templates.
- Finally, development tasks are identified and a project schedule is created.

2. High Level Design:

- External specifications for each component to be constructed are developed and a component design is created.
- Prototypes are built when uncertainty exists.
- All issues are recorded and tracked.

3. High Level Design Review:

- Formal verification methods are applied to uncover errors in the design.
- Metrics are maintained for all important tasks and work results.

4. Development:

- The component-level design is refined and reviewed.
- Code is generated, reviewed, compiled, and tested.
- Metrics are maintained for all important tasks and work results.

5. Postmortem:

- The effectiveness of the process is determined by using measures and metrics.
- Measures and metrics should provide guidance for modifying the process to improve its effectiveness.

2. Team Software Process (TSP):

- TSP is defined as operational process framework.
- The goal of TSP is to build a "self-directed" project team that organizes itself to produce high-quality software.
- The following are the objectives for TSP:
    i. Ensures quality software product.
    ii. Create secure software product.
    iii. Improve process management in an organization.
- TSP defines the following framework activities:
    i. Project launch.

ii. High-level design.
iii. Implementation
iv. Integration and test
v. Postmortem.

- TSP makes use of a wide variety of scripts, forms, and standards that serve to guide team members in their work.
- "Scripts" define specific process activities (i.e., project launch, design, implementation, integration and system testing, postmortem)

PROCESS TECHNOLOGY:

- Process technology tools have been developed to help software organizations analyze their current process, organize work tasks, control and monitor progress, and manage technical quality.
- Process technology tools allow a software organization to build an automated model of the process framework, task sets, and umbrella activities.
- The model is normally represented as a network, which can be analyzed to determine typical workflow and examine alternative process structures that might lead to reduced development time or cost.
- Once an acceptable process has been created, other process technology tools can be used to allocate, monitor, and even control all software engineering activities, actions, and tasks.
- Each member of a software team can use such tools to develop a checklist of work tasks to be performed, work products to be produced, and quality assurance activities to be conducted.
- Process Technology tools include diagram tools, process modeling tools, project management tools, documentation tools, analysis tools, design tools, configuration tools, programming tools, prototyping tools, quality assurance tools, maintenance tools etc.