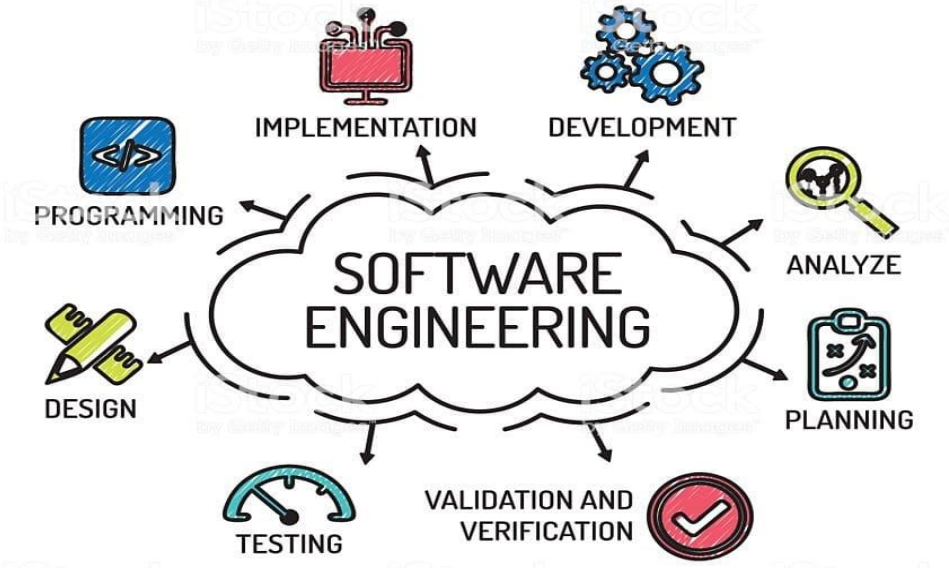
Software Engineering & Project Development

Module1 : Introduction


IMPLEMENTATION
DEVELOPMENT
PROGRAMMING
ANALYZE
SOFTWARE ENGINEERING
DESIGN
PLANNING
TESTING
VALIDATION AND VERIFICATION

Course Outcomes

- CO1: Differentiate process models to judge which process model has to be adopted for the given scenarios.
- CO2: Derive both functional and nonfunctional requirements from the case study.
- CO3: Analyze the importance of various software testing methods and agile methodology.
- CO4: Illustrate the role of project planning and quality management in software development.
- CO5: Identify appropriate techniques to enhance software quality.

Module-1

Software and Software Engineering: The nature of Software, The unique nature of WebApps, Software Engineering, The software Process, Software Engineering Practice, Software Myths.

Process Models: A generic process model, Process assessment and improvement, Prescriptive process models: Waterfall model, Incremental process models, Evolutionary process models, Concurrent models, Specialized process models. Unified Process , Personal and Team process models

The Evolving role of software

Dual role of Software

1. A Product

- Transforms information - produces, manages, acquires, modifies, displays, or transmits information
- Delivers computing potential of hardware and networks

2. A Vehicle for delivering a product

- Controls other programs (operating system)
- Effects communications (networking software)
- Helps build other software (software tools & environments)

Where can you find software?

Software is Almost Everywhere.

Software is defined as

1. Instructions - Programs that when executed provide desired function
2. Data structures -Enable the programs to adequately manipulate information
3. Documents -Describe the operation and use of the programs.

- Software is the collection of executable program codes, associated libraries and documentation.
- Software products may be developed for a particular customer or may be developed for a general market.

Software products may be

- **Generic -** developed to be sold to a range of different customers e.g. PC software such as Excel or Word.
- **Custom -** developed for a single customer according to their specification.

Software Characteristics

- Software is developed or engineered; it is not manufactured in the classical sense.
- Software does not “wear out” but it does deteriorate.
- Software continues to be custom built, as industry is moving toward component based construction.

Changing Nature of Software

The nature of software has changed a lot over the years.

1.System software: It is a collection of programs to provide service to other programs. Infrastructure software come under this category like compilers, operating systems, editors, drivers, etc.

2. Real time software: These software are used to monitor, control and analyze real world events as they occur.

3. Embedded software: This type of software is placed inside the hardware of the system and is used control the various functions of the product.

Eg: Microwave oven controls

4. Business software : This is the largest application area. The software designed to process business applications is called business software.

5. Personal computer software: The software used in personal computers are covered in this category. Examples are word processors, computer graphics, multimedia and animating tools, database management, computer games etc

6. Artificial intelligence software: Artificial Intelligence software makes use of non numerical algorithms to solve complex problems.

Examples: Robotics, Gaming etc

7. Web based software: The software related to web applications come under this category. Examples are CGI, HTML, Java, Perl, DHTML etc.

Unique Nature of Web Apps

Web-based systems and applications (WebApps) were born. Today, WebApps have evolved into sophisticated computing tools that not only provide stand-alone function to the end user, but also have been integrated with corporate databases and business applications.

WebApps are one of a number of distinct software categories. Web-based systems and applications "involve a mixture between **print publishing** and **software development**, between **marketing** and **computing**, between **internal communications and external relations,** and between **art and technology**."

The following attributes are encountered in the vast majority of WebApps.

- **Network intensiveness.** A WebApp resides on a network and must serve the needs of a diverse community of clients. The network may enable worldwide access and communication (i.e., the Internet) or more limited access and communication (e.g., a corporate Intranet).

• **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.

• **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

• **Performance**. WebApp should work effectively in terms of processing speed. 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.

• **Availability**. Although expectation of 100 percent availability is un reasonable, users of popular WebApps often demand access on a 24/7/365 basis.

• **Continuous evolution.** Unlike conventional application software that evolves over a series of planned, chronologically spaced releases, Web applications evolve continuously.

• **Immediacy**. Although immediacy—the compelling need to get software to market quickly—is a characteristic of many application domains, WebApps often exhibit a time_x0002_to-market that can be a matter of a few days or weeks.

• **Security**. Because WebApps are available via network access, it is difficult, if not impossible, to limit the population of end users who may access the application. In order to protect sensitive content and provide secure modes

Essential attributes of good software

- **Maintainability**
- Software should be written in such a way so that it can evolve to meet the changing needs of customers.
- This is a critical attribute because software change is an inevitable requirement of a changing business environment.

- **Dependability and security**
- Software dependability includes a range of characteristics including reliability, security and safety.
- Dependable software should not cause physical or economic damage in the event of system failure. Malicious users should not be able to access or damage the system

- **Efficiency**
- Software should not make wasteful use of system resources such as memory and processor cycles.
- Efficiency therefore includes responsiveness, processing time, memory utilisation, etc.

- **Acceptability**
- Software must be acceptable to the type of users for which it is designed. This means that it must be understandable, usable and compatible with other systems that they use.

- What is Software Engineering?

➢ Software Engineering is an *engineering discipline* that is concerned with *all aspects of software production.*

➢ Software engineers should adopt a systematic and organized approach to their work.

➢ use appropriate tools and techniques depending on the problem to be solved, the development constraints and the resources available

Software Engineering -A Layered Technology

In order to build software that is ready to meet the challenges of the 21st century, you must recognize a few simple realities.

- Problems should be understood before a software solution is developed.
- Design is a pivotal Software Engineering activity.
- Software should exhibit high quality.
- Software should be maintainable

These simple realities lead to one conclusion. Software in all of its forms and across all of its application domains should be engineered.

- Software Engineering is a layered technology. Software Engineering encompasses a Process, Methods for managing and engineering software and tools.
- Software engineering is a fully layered technology, to develop software we need to go from one layer to another. All the layers are connected and each layer demands the fulfillment of the previous layer.

THE SOFTWARE PROCESS

- A **process** is a collection of activities, actions, and tasks that are performed when some work product is to be created.
- 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, or degree of rigor with which software engineering is to be applied.
- An **action** encompasses a set of tasks that produce a major work product (e.g., an architectural design model).
- A **task** focuses on a small, but well-defined objective (e.g., conducting a unit test) that produces a tangible outcome

Process Framework

- Framework is a Standard way to build and deploy applications.
- Software Process Framework is a foundation of complete software engineering process.
- Software process framework includes all set of umbrella activities. It also includes number of framework activities that are applicable to all software projects.

Generic Process Framework Activities

- **Communication:**
  - Heavy communication and collaboration with customers, stakeholders, team
  - Encompasses requirements gathering and related activities

- **Planning:**
  - Workflow that is to follow
  - Describe technical task, likely risk, resources will require, work products to be produced and a work schedule.

Modeling:

- Help developer and customer to understand requirements (Analysis of requirements & Design of software)

Construction:

- Code generation: either manual or automated or both
- Testing – to uncover error in the code.

- **Deployment:**
  - Delivery to the customer for evaluation
  - Customer provide feedback

Umbrella Activities

1. **Software project tracking and control**
   - Assessing progress against the project plan.
   - Take adequate action to maintain schedule.

2. **Formal technical reviews**
   - Assessing software work products in an effort to uncover and remove errors before it goes into next action or activity.

3. **Software quality assurance**
   - Define and conducts the activities required to ensure software quality.

4. **Software configuration management**
   - Manages the effects of change

5. **Document preparation and production**
   - Help to create work products such as models, documents, logs, form and list.

6. **Reusability management**
   - Define criteria for work product reuse
   - Mechanisms to achieve reusable components.

7. **Measurement & Metrics**
   - All measurement related activities like cost, time man power is done.
   - Assist the team in delivering software that meets customer's needs.

8. **Risk management**
   - Assesses risks that may effect that outcome of project or quality of product (i.e. software)

General Principles of Software Engineering:

The word principle is “an important underlying law or assumption required in a system of thought.” David Hooker [Hoo96] has proposed seven principles that focus on software engineering practice as a whole.

1. The First Principle: The Reason It All Exists

A software system exists for one reason: to provide value to its users. All decisions should be made with this in mind.

2. The Second Principle: Keep It Simple, Stupid!

All design should be as simple as possible.

This facilitates having a more easily understood and easily maintained system.

3. The Third Principle: Maintain the Vision

A clear vision is essential to the success of a software project. Without one, a project almost unfailingly ends up being “of two [or more] minds” about itself.

4. The Fourth Principle: What You Produce, Others Will Consume

always specify, design, and implement knowing someone else will have to understand what you are doing. The audience for any product of software development is potentially large.

5. The Fifth Principle: Be Open to the Future

Never design yourself into a corner. Always ask “what if,” and prepare for all possible answers by creating systems that solve the general problem, not just the specific one.

6. The Sixth Principle: Plan Ahead for Reuse

Planning ahead for reuse reduces the cost and increases the value of both the reusable components and the systems into which they are incorporated.

7. The Seventh principle: Think!

Placing clear, complete thought before action almost always produces better results.

When you think about something, you are more likely to do it right. You also gain knowledge about how to do it right again.

Software Myths

Software myths are erroneous beliefs about software and the process that is used to build it.

We categorize myths from three different perspectives.

Management myths:

Myth: We already have a book that's full of standards and procedures for building software. Won't that provide my people with everything they need to know?

Reality: The book of standards may very well exist, but is it used? Are software practitioners aware of its existence? Does it reflect modern software engineering practice? Is it complete? Is it adaptable? Is it streamlined to improve time-to-delivery while still maintaining a focus on

quality? In many cases, the answer to all of these questions is "no."

Myth: If we get behind schedule, we can add more programmers and catch up (sometimes called the “Mongolian horde” concept).

Reality: Software development is not a mechanistic process like manufacturing. As new people

are added, people who are working must spend time educating the newcomers, thereby reducing the amount of time spent on productive development effort. People can be added but only in a planned and well coordinated manner.

Myth: If I decide to outsource the software project to a third party, I can just relax and let that firm build it.

Reality: If an organization does not understand how to manage and control software projects internally, it will invariably struggle when it out-sources software projects.

Customer myths.

Myth: A general statement of objectives is sufficient to begin writing programs—we can fill in the details later.

Reality: Although a comprehensive and stable statement of requirements is not always possible, an ambiguous "statement of objectives" is a recipe for disaster.

Myth: Software requirements continually change, but change can be easily accommodated because software is flexible.

Reality: It is true that software requirements change, but the impact of change varies with the time at which it is introduced. When requirements changes are requested early (before design or code has been started), the cost impact is relatively small. However, as time passes, the cost impact grows rapidly

Customer myths.

Myth: A general statement of objectives is sufficient to begin writing programs—we can fill in the details later.

Reality: Although a comprehensive and stable statement of requirements is not always possible, an ambiguous "statement of objectives" is a recipe for disaster.

Myth: Software requirements continually change, but change can be easily accommodated because software is flexible.

Reality: It is true that software requirements change, but the impact of change varies with the time at which it is introduced. When requirements changes are requested early (before design or code has been started), the cost impact is relatively small. However, as time passes, the cost impact grows rapidly

Practitioner's myths:

Myth: Once we write the program and get it to work, our job is done.

Reality: Someone once said that "the sooner you begin 'writing code,' the longer it'll take you to get done." Industry data indicate that between 60 and 80 percent of all effort expended on software will be expended after it is delivered to the customer for the first time.

Myth: The only deliverable work product for a successful project is the working program.

Reality: A working program is only one part of a software configuration that includes many elements.

Myth: Software engineering will make us create voluminous and unnecessary documentation and will invariably slow us down.

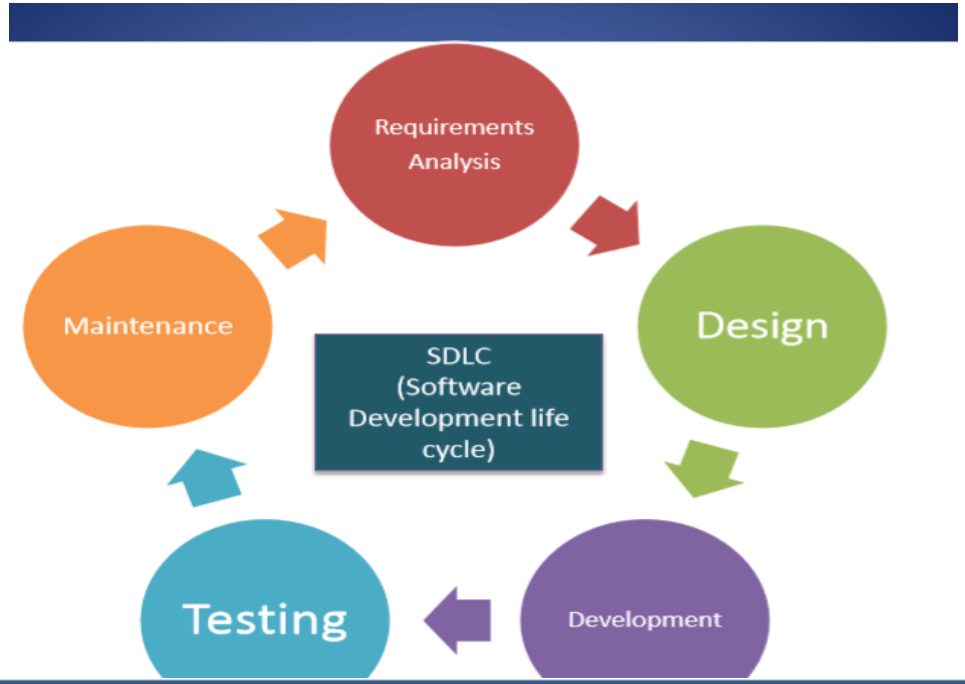
Reality: Software engineering is not about creating documents. It is about creating a quality product. Better quality leads to reduced rework.

PROCESS MODELS
Module 1- Chapter 2

Software Process Model

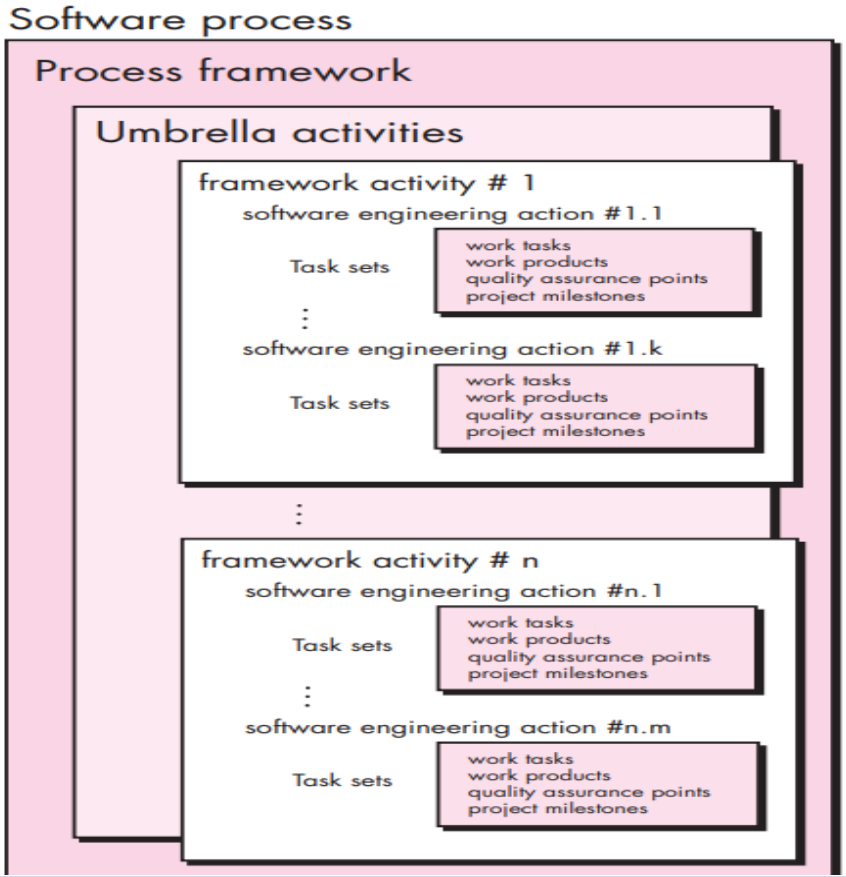
- A software process model is a road map that helps you create a timely, high-quality software.
- Aim to produce a high quality software that meets customer expectation
- Each process model followed the SDLC, “Describing how to develop, maintain, replace and alter or enhance specific software”

SDLC


Requirements Analysis
Design
Development
Testing
Maintenance
SDLC (Software Development life cycle)

Generic Process Model

- A process was defined as a collection of work activities, actions, and tasks that are performed when some work product is to be created.
- Each of these activities, actions, and tasks reside within a framework or model that defines their relationship with the process and with one another.
- The software process is represented schematically in Figure next. Referring to the figure, each framework activity is populated by a set of software engineering actions.
- Each 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, the quality assurance points that will be required, and the milestones that will be used to indicate progress.

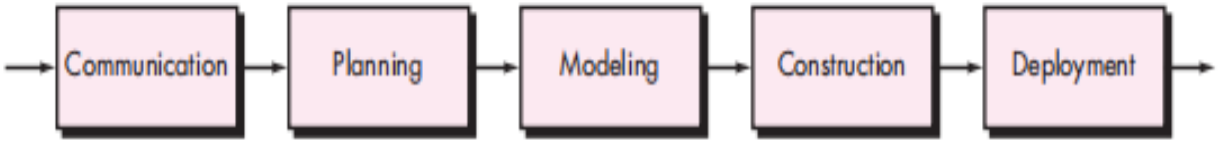

Software process
Process framework
Umbrella activities
framework activity # 1
software engineering action #1.1
Task sets
work tasks
work products
quality assurance points
project milestones
software engineering action #1.k
Task sets
work tasks
work products
quality assurance points
project milestones
framework activity # n
software engineering action #n.1
Task sets
work tasks
work products
quality assurance points
project milestones
software engineering action #n.m
Task sets
work tasks
work products
quality assurance points
project milestones

- A generic process framework for software engineering defines five framework activities— communication, planning, modeling, construction, and deployment.
- In addition, a set of umbrella activities—project tracking and control, risk management, quality assurance, configuration management, technical reviews, and others—are applied throughout the process.
- One important aspect of the software process that is to be discussed is called **process flow.**
- **It** describes how the framework activities and the actions and tasks that occur within each framework activity are organized with respect to sequence and time and is illustrated in Figure

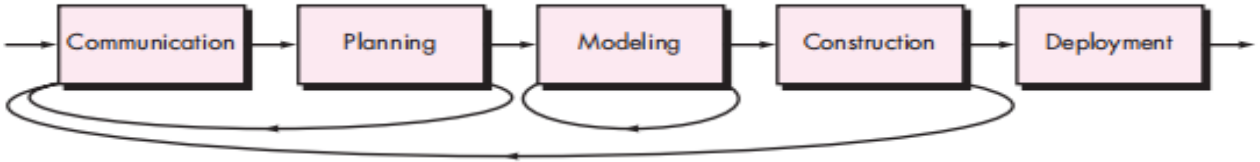
Defining a Framework Activity:

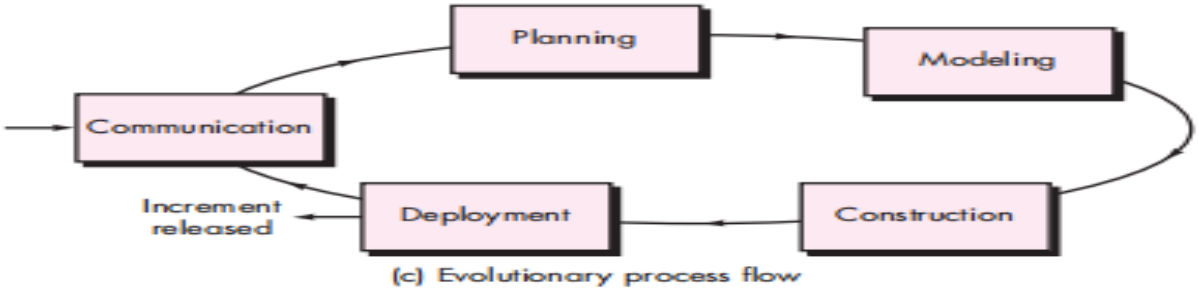
To properly execute any one of these activities as part of the software process, a key question that the software team needs to ask is:

What actions are appropriate for a framework activity, given the nature of the problem to be solved.

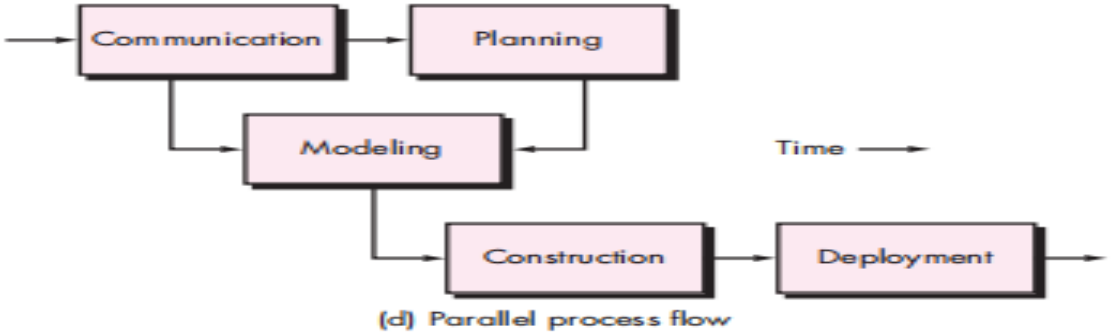


(a) Linear process flow





(c) Evolutionary process flow



(d) Parallel process flow

For a small software project requested by one person (at a remote location) with simple, straightforward requirements, the communication activity might encompass little more than a phone call with the appropriate stakeholder.

The work tasks that this action encompasses are:

1. Make contact with stakeholder via telephone.
2. Discuss requirements and take notes
3. Organize notes into a brief written statement of requirements.
4. E-mail to stakeholder for review and approval.

If the project was considerably more complex with many stakeholders, each with a different set of requirements, the communication activity might have six distinct actions:

inception, elicitation, elaboration, negotiation, specification, and validation.

Identifying a Task Set

You should choose a task set that best accommodates the needs of the project and the characteristics of your team.

A task set defines the actual work to be done to accomplish the objectives of a software engineering action.

- ➢ A list of the task to be accomplished
- ➢ A list of the work products to be produced
- ➢ A list of the quality assurance filters to be applied

Process Pattern

A process pattern

- describes a process-related problem that is encountered during software engineering work.
- identifies the environment in which the problem has been encountered.
- suggests one or more proven solutions to the problem.

Ambler has proposed a template for describing a process pattern:

1. Pattern Name

The pattern is given a meaningful name describing it within the context of the software process (e.g., Technical Reviews).

2. Forces

The environment in which the pattern is encountered and the issues that make the problem visible and may affect its solution

3. Types

- Stage patterns— defines a problem associated with a framework activity for the process. It includes multiple task patterns as well.

  For example: Establishing Communication would incorporate the task pattern Requirements Gathering and others.

- Task patterns— defines a problem associated with a software engineering action or work task and relevant to successful software engineering practice
- Phase patterns— define the sequence of framework activities that occur with the process, even when the overall flow of activities is iterative in nature.

  Example includes Spiral Model or Prototyping.

4. Initial context

Describes the conditions under which the pattern applies, prior to the initiation of the pattern:

(1) What organizational or team-related activities have already occurred?

(2) What is the entry state for the process?

(3) What software engineering information or project information already exists?

5. Problem : The specific problem to be solved by the pattern.

6. Solution: Describes how to implement the pattern successfully.

7. Resulting Context. Describes the conditions that will result once the pattern has been successfully implemented. Upon completion of the pattern:

(1) What organizational or team-related activities must have occurred?

(2) What is the exit state for the process?

(3) What software engineering information or project information has been developed?

8. Related Patterns. Provide a list of all process patterns that are directly related to this one. This may be represented as a hierarchy or in some other diagrammatic form.

9. Known Uses and Examples. Indicate the specific instances in which the pattern is applicable.

Conclusion on Process patterns

➢ Provide an effective mechanism for addressing problems associated with any software process.

➢ The patterns enable you to develop a hierarchical process description that begins at a high level of abstraction (a phase pattern).

➢ The description is then refined into a set of stage patterns that describe framework activities.

➢ Once process patterns have been developed, they can be reused for the definition of process variants.

Process Assessment and Improvement

Process patterns must be coupled with solid software engineering practice.

Assessment attempts to understand the current state of the software process with the intent on improving it.

A number of different approaches to software process assessment and improvement have been proposed over the past few decades.

- **Standard CMMI assessment method for process improvement(SCAMPI)**
  - provides a five step process assessment model that incorporates initiating, diagnosing, establishing, acting and learning

- **CMM based appraisal for internal process improvement (CBA IPI)**

  - provides a diagnostic technique for assessing the relative maturity of a software organization using the SEI CMM.

- **SPICE(ISO/IEC 15504)**

  - standard 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.

- **ISO 9001:2000 for software**

  - is 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 Model

- ***Prescriptive process models advocate an orderly approach to software engineering.***

i.e Defines a distinct set of activities, actions, tasks, milestones, and work products that are required to engineer high-quality software

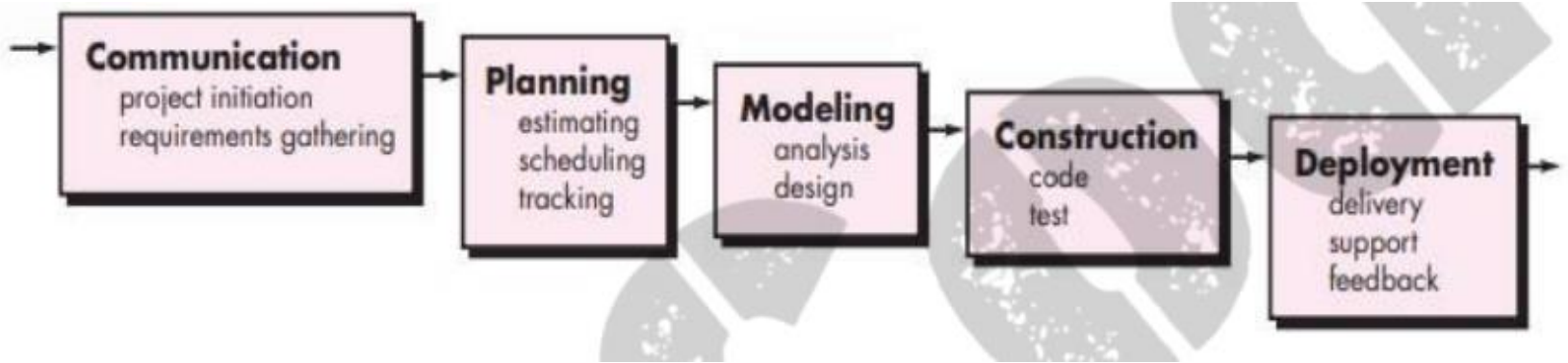
- The activities may be linear, incremental, or evolutionary

- Waterfall Model –represents elements of a linear process flow
- Incremental Model – combines elements of linear and parallel process flows
- Evolutionary Model – follows the evolutionary process flow that combines elements of linear and iterative process flows
- Prototyping
- Spiral
- Concurrent Model – combines elements of iterative and parallel process flows

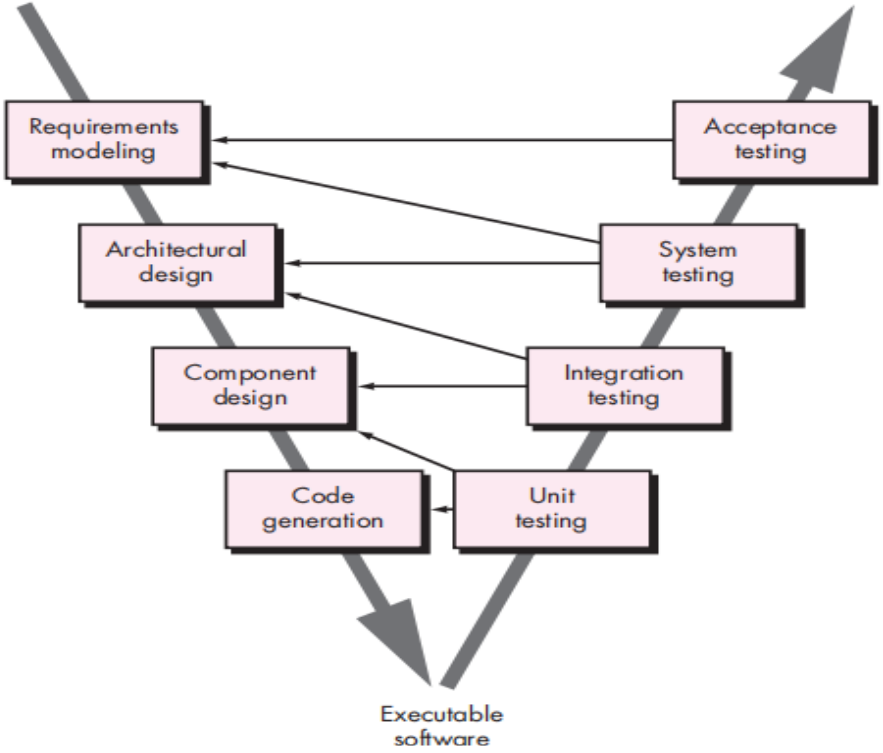
Waterfall Model

- Oldest software lifecycle model
- Proposed by Winston Royce in 1970
- Used when requirements are well understood and risk is low
- Work flow is in a linear (i.e., sequential) fashion.
- In the waterfall model, software activities proceed logically through a sequence of steps.
- Each step bases its work on the activities of the previous step. The design phase logically follows the requirements phase, the implementation follows the design & so on.
- The next phase should not start until the current phase is completed.

Waterfall Model



- A variation in the representation of the waterfall model is called the V-model.



Requirement gathering and Analysis.

- This is the first phase of waterfall model which includes a **meeting with the customer to understand his requirements.**
- This is the most crucial phase as any ***misinterpretation*** at this stage may give rise to ***validation issues*** later.
- The software definition must be detailed and accurate with *no ambiguities.*
- It is very important to understand the customer requirements and expectations so that the *end product meets his specifications.*
- All this requirements are then well documented and discussed further with the customer for reviewing.

DESIGN

- The customer requirements are broken down into logical modules for the ease of implementation. Hardware and software requirements for every module are Identified and designed accordingly.
- Also the inter relation between the various logical modules is established at this stage. Algorithms and diagrams defining the scope and objective of each logical model are developed.
- In short, this phase lays a fundamental for actual programming and implementation
- It is an intermediate step between requirements analysis and coding.

Design focuses on program attribute such as-

- Data Structure.
- Software Architecture.
- Algorithm Details etc…….
- The requirements are translated in some easy to represent form using which coding can be done effectively and efficiently.
- The design needs to be documented for further use.

Coding

- Coding is a step in which design is translated into machine-readable form.
- If design is done in sufficient detail then coding can be done effectively. Programs are created in this phase.
- In this phase all software divided into small module then do coding for that small module rather than coding for whole software.
- According to design programmers do code and make class and structure of whole software

Testing

- In this stage, both individual components and the integrated components are methodically verified to ensure that they are error-free and fully meet the requirements outlined in the first step.
- In this phase testing is divided into two parts
  1) HARDWARE
  2) SOFTWARE.

Maintenance

- This is the final phase of the waterfall model, in which the completed software product is handed over to the client after alpha, beta testing.
- After the software has been deployed on the client site, it is the duty of the software development team to undertake routine maintenance activities by visiting the client site.
- If the customer suggests changes or enhancements the software process has to be followed all over again right from the first phase i.e. requirement analysis.
- The usually the longest stage of the software. In this phase the software is updated to.

When to use the Waterfall Model

- Requirements are **very well known**
- Product definition is **stable**
- Technology is **understood**
- Porting an existing product to a **new platform.**

Waterfall Model (Problems)

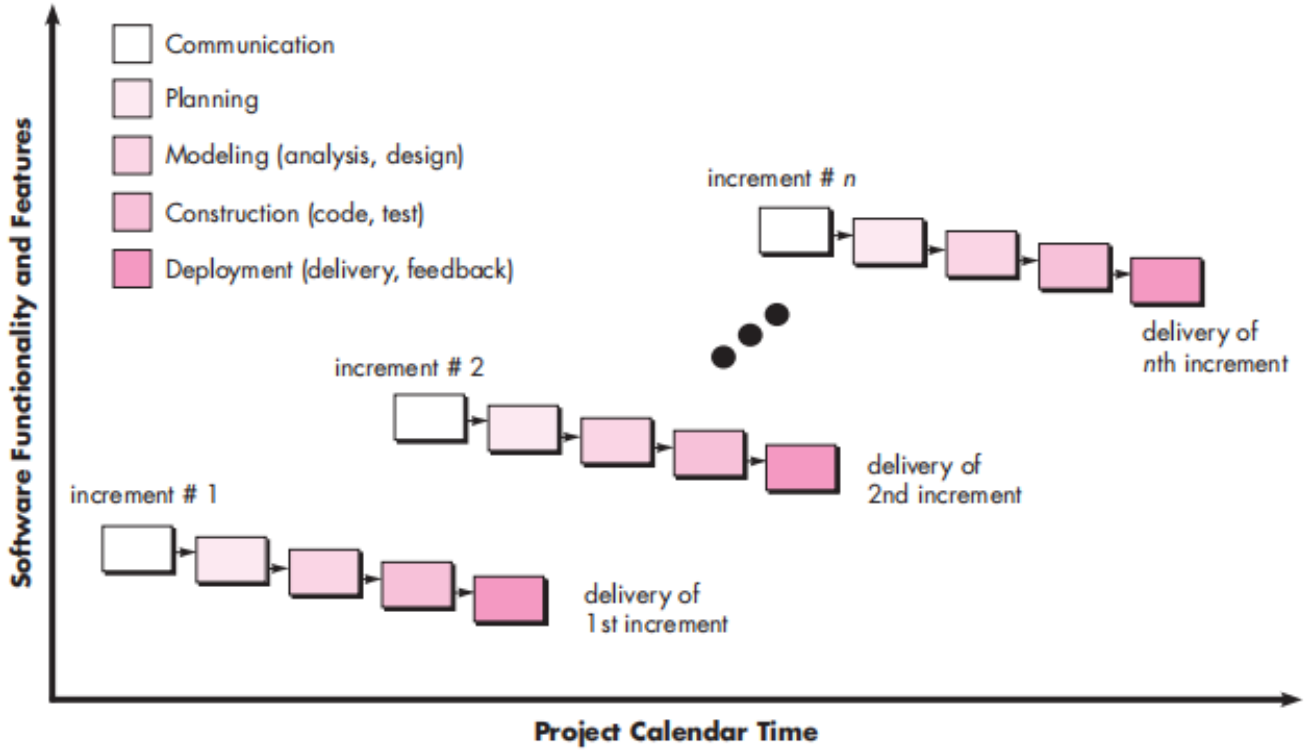
- Doesn't support iteration, so changes can cause confusion
- Difficult for customers to state all requirements explicitly and up front
- Requires customer patience because a working version of the program doesn't occur until the final phase.

Advantages

- Easy to understand, easy to use
- Provides structure
- Milestones are clear
- Good for management control (plan, staff, track)
- Works well when quality is more important than cost or schedule

Incremental Process Model

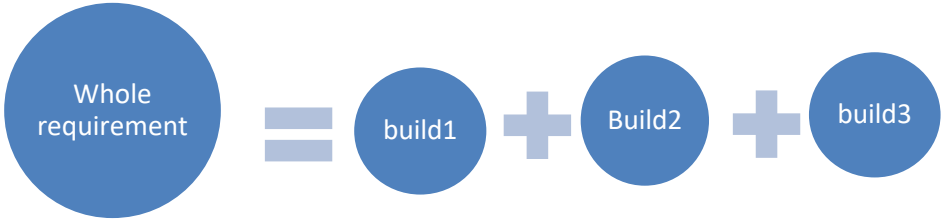
- The incremental model combines elements of **linear and parallel** process flows.
- The incremental model applies linear sequences in a staggered fashion as calendar time progresses.
- Each linear sequence produces deliverable “increments” of the software.


Communication
Planning
Modeling (analysis, design)
Construction (code, test)
Deployment (delivery, feedback)
increment # n
delivery of nth increment
increment # 2
delivery of 2nd increment
increment # 1
delivery of 1st increment
Software Functionality and Features
Project Calendar Time

- 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.
- The core product is used by the customer (or undergoes detailed evaluation). 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.

It basically:

- Divides the overall project into a number of increments.
- Then it applies the waterfall model to each increment.
- The system is put into production when the first increment is delivered.
- As time passes additional increments are completed and added to the working system



Advantages:

- Generates working software quickly and early during the SLC
- This model is more flexible – less costly to change scope and requirements.
- It is easier to test and debug during a smaller iteration.
- In this model customer can respond to each built.
- Lowers initial delivery cost.
- Easier to manage risk because risks are identified and handled during it's iteration.

Disadvantages:

- Needs good planning and design.
- Needs a clear and complete definition of the whole system before it can be broken down and built incrementally.
- *Total cost is higher than waterfall.*

When to use the Incremental model?

- When the requirements of the complete system are clearly defined and understood.
- Major requirements must be defined; however, some details can evolve with time.
- There is a need to get a product to the market early.
- A new technology is being used.
- Resources with needed skill set are not available

Problem :

- User inconvenience

Evolutionary Process Model

- Combination of iterative and incremental model
- We break our work into smaller parts.
- Priotize those parts and delive to customer one by one

- Iterative is of two types
  - Prototype Model
  - Spiral Model

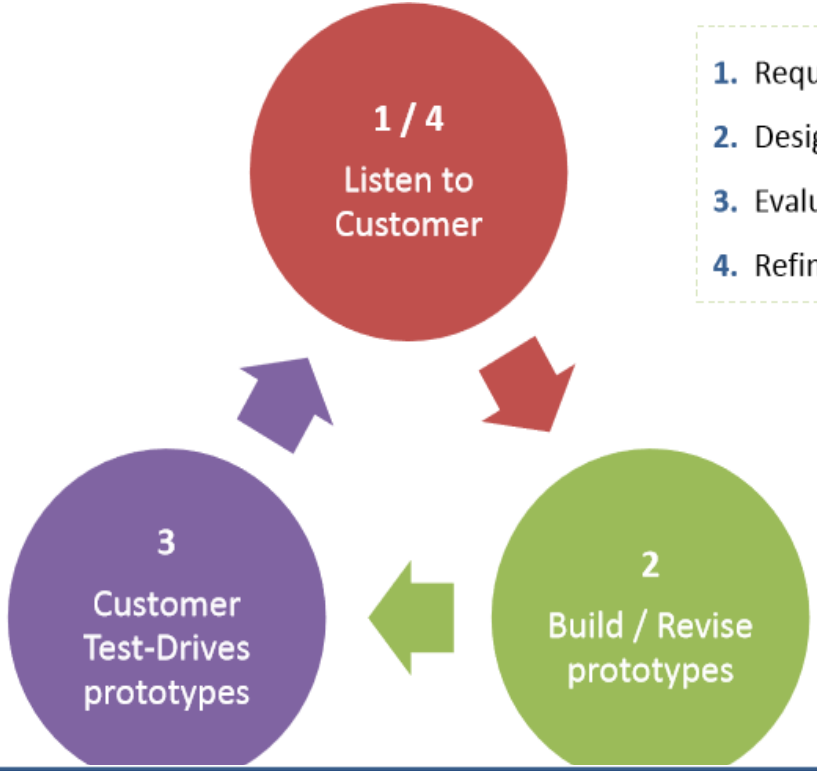
Evolutionary Process Model

Prototyping Model

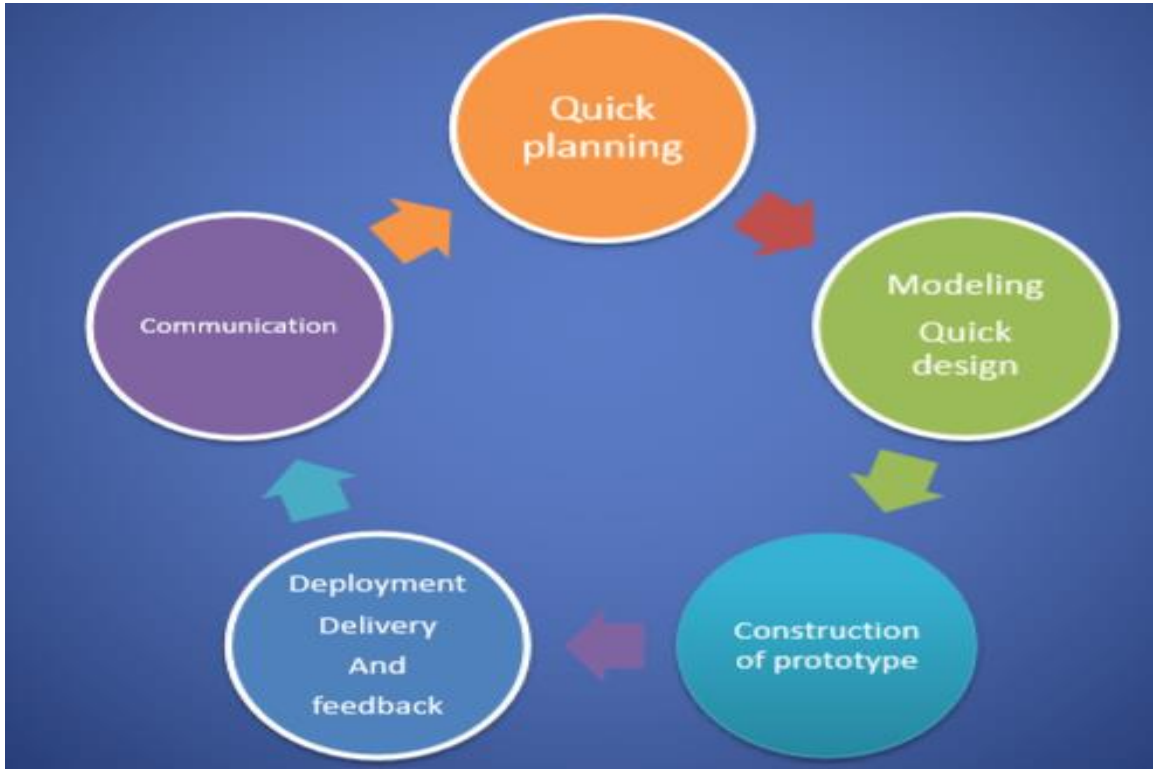
- When a customer defines a set of general objectives for a software but does not identify detailed I/O or processing requirements.
- A prototype is built to understand the requirements.
- By using this prototype, the client can get an "actual feel" of the system
- The interactions with prototype can enable the client to better understand the requirements of the desired system.
- Prototyping is an attractive idea for complicated and large systems

Consists of 4 iterating phases:

- Requirements gathering.
- Design and build SW prototype.
- Evaluate prototype with customer.
- Refine requirements.



1. Requirements gathering.
2. Design and build SW prototype.
3. Evaluate prototype with customer.
4. Refine requirements.


Quick planning
Modeling
Quick design
Construction of prototype
Deployment
Delivery
And
feedback
Communication

- Follows an evolutionary and iterative approach
- Used when requirements are not well understood
- Serves as a mechanism for identifying software requirements
- Focuses on those aspects of the software that are visible to the customer/user
- Feedback is used to refine the prototype

Advantages:

- Users are actively involved in the development
- Users get a better understanding of the system being developed.
- Errors can be detected much earlier.
- Quicker user feedback is available leading to better solutions.

Disadvantages:

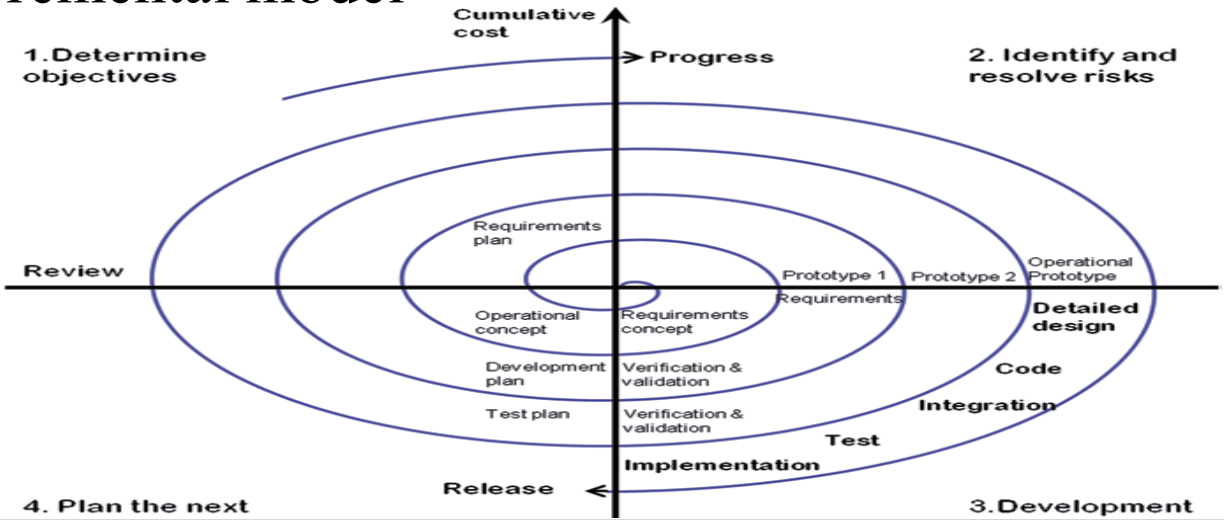
- Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans.
- Incomplete application may cause application not to be used as the full system was designed.
- Incomplete problem analysis.

When to use Prototype Model

- Prototype model should be used when the desired system needs to have a lot of interaction with the end users.
- Typically, online systems, web interfaces have a very high amount of interaction with end users, are best suited for Prototype model.
- They are excellent for designing good human computer interface systems

Spiral Model

- Invented by Dr. Barry Boehm in 1988
- Follows an evolutionary approach
- similar to the incremental model



- The Spiral Model is a combination of the waterfall model and the iterative model.
- It provides support for Risk Handling.
- In its diagrammatic representation, looks like a spiral with many loops.
- The exact number of loops of the spiral is unknown and can vary from project to project.
- Each loop of the spiral is called a phase of the software development process.

Two main distinguishing features:

- one is **cyclic approach** for incrementally growing a system's degree of definition and implementation while decreasing its degree of risk.
- The other is a set of **anchor point milestones** for ensuring stakeholder commitment to feasible and mutually satisfactory system solutions.

- Using the spiral model, software is developed in a series of evolutionary releases.
- During early iterations, the release might be a model or prototype. During later iterations, increasingly more complete versions of the engineered system are produced.
- Risk is considered as each revolution is made. Anchor point milestones are a combination of work products and conditions that are attained along the path of the spiral are noted for each evolutionary pass.
- The spiral model is a realistic approach to the development of large-scale systems and software. Because software evolves as the process progresses, the developer and customer better understand and react to risks at each evolutionary level.

Planning Phase

- Requirements are gathered(BRS &SRS)

Risk Analysis:

- a process is undertaken to identify risk and alternate solutions.
- A prototype is produced at the end of the risk analysis phase.

Engineering Phase :

- software is developed,
- Testing at the end of the phase

Evaluation phase:

- This phase allows the customer to evaluate the output of the project to date before the project continues to the next spiral.

Advantages :

- High amount of risk analysis hence, avoidance of Risk is enhanced.
- Good for large and mission-critical projects.
- Strong approval and documentation control.
- Additional Functionality can be added at a later date.
- Software is produced early in the software life.

Disadvantages :

- Can be a costly model to use.
- Risk analysis requires highly specific expertise.
- Project's success is highly dependent on the risk analysis phase.
- Doesn't work well for smaller projects.

When to use Spiral model:

- When costs and risk evaluation is important
- For medium to high-risk projects
- Long-term project commitment unwise because of potential changes to economic priorities
- Users are unsure of their needs
- Requirements are complex
- New product line
- Significant changes are expected

A FINAL COMMENT ON EVOLUTIONARY PROCESS

- Difficult in project planning
- Speed of evolution is not known
- Does not focus on flexibility and extensibility (more emphasis on high quality)
- Requirement is balance between high quality and flexibility and extensibility

Waterfall vs Spiral:

- The sequential nature of the waterfall model if a bug is found or an error is incurred for a preliminary reason, we need to start from the scratch again.
- Whereas, under spiral model every prototype is tried and tested and hence the chances of find errors at later stages are very rare.
- In spiral, we can easily adjust the software development with the required changes.
- The prototypes which are created in every stage, enables us to roll back only a few steps.
- Waterfall model the stages are executed under a sequential flow. Every new phase is processed only after completing the previous phase.

Incremental Vs Spiral:

- Incremental Development is a practice where the system functionalities are sliced into increments (small portions).
- In each increment, a vertical slice of functionality is delivered by going through all the activities of the software development process, from the requirements to the deployment.
- Incremental Development (adding) is often used together with Iterative Development (redo) in software development. This is referred to as Iterative and Incremental Development (IID).
- The spiral model is similar to the incremental model, with more emphases placed on risk analysis.
- A software project repeatedly passes through these phases in iterations (called Spirals in this model). The baseline spiral, starting in the planning phase, requirements are gathered and risk is assessed.

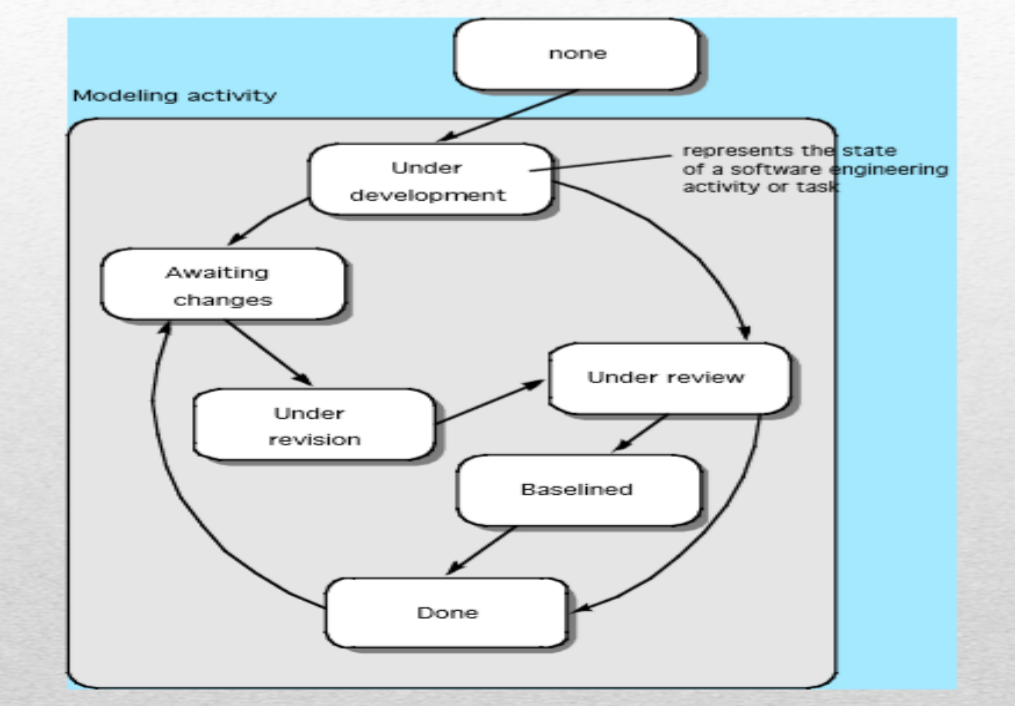
Prototype Model Vs Spiral Model:

- Prototype model is suitable when the requirement of the client is not clear and it is supposed to be changed. It doesn't cover any risk management. While Spiral model is an enhancement of the prototyping model with so many extra features.

- Spiral model is called a meta model. Spiral model is made with the features of Prototype model and Waterfall model. Spiral model takes special care about Risk Analysis. Where as it is not given importance in Prototype model.

- Prototype model that end when software is delivered after software deliver is not responsible for any problem of software.
- The spiral model can be adapt to apply throughout the life of computer software.

Concurrent Models



- The concurrent process model sometimes called concurrent engineering allows to a software team to represent iterative and concurrent elements of any process model.
- This model is suited to all types of software but is generally used for client server applications
- In real life the software development activities do not take place in sequence
- Most activities will be going on concurrently but reside in different states.
- The states will change when some event occurs. So in this model all the activities are shown along with their states at any point of time.
- As time goes on the states of the activities will change.
- It provides an accurate picture of the current state of a project.

- It defines software engineering as a network of activities each in a different state, each dependent on one another and each going on concurrently.
- Awaiting changes state – after the completion of the 1st iteration (communication activity)
- None state – while initial communication is carried out (modeling activity)
- Under development state – after the communication is completed.
- Awaiting changes state – if any changes in the requirement must be made

Specialized process models

Specialized process models take on many of the characteristics of one or more of the traditional models.

However, these models tend to be applied when a specialized or narrowly defined software engineering approach is chosen.

There are 3 types of specialized process models:

1. Component Based Development
2. Formal Methods Model
3. Aspect Oriented Software development

- **Component based development** - the process to apply when reuse is a development objective.The component based development specialized process model incorporates the following steps:
- Available component based products are researched and evaluated for the application domain in question.
- Component integration issues are considered.
- A software architecture is designed to accommodate the components.
- Components are integrated into the architecture.
- Comprehensive testing is conducted to ensure proper functionality.

- **Formal methods** - emphasizes the mathematical specification of requirements
- 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, mathematical notation.
- A variation on this approach, called **clean room software engineering.**
- Ambiguity, incompleteness, and inconsistency can be discovered and corrected more easily, but through the application of mathematical analysis.

- **Aspects Oriented Software Development** - provides a process and methodological approach for defining, specifying, designing, and constructing *aspects*
- AOSD defines “aspects” that express customer concerns that cut across multiple system functions, features, and information.
- 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.”
- Grundy provides further discussion of aspects in the context of what he calls aspect oriented component engineering (AOCE).

Personal and Team Process Models

- The best software process is one that is close to the **people who will be doing the work**.
- Each software engineer would **create a process that best fits his or her needs,** and at the same time **meets the broader needs of the team and the organization**.
- Alternatively, the team itself would create its own process, and at the same time meet the narrower needs of individuals and the broader needs of the organization.

Personal and Team Process Models

- The quality of a software system is determined by the **quality of its worst components**.
- The quality of a software component is governed by the **individual who developed it**.
- The quality of a software component is governed by the **quality of the process used** to develop it.
- The key to quality is the individual **developer's skill, commitment**, and personal process **discipline**.

- As a software professional, you are responsible for your personal process.
- You should measure, track, and analyze your work and learn from your performance variations. Also incorporate into personal practices.

Personal Software Process (PSP)

- The personal software process (PSP) emphasizes **personal measurement** of both the **work product** that is produced and the **resultant quality** of the work product.
- The PSP process model defines five framework activities:
  - planning,
  - high-level design,
  - high level design review,
  - Development and
  - postmortem.

Personal Software Process (PSP)

- **Planning:** This activity isolates **requirements** and, base on these develops both size and resource estimates.
  In addition, a **defect estimate** is made. All metrics are recorded on worksheets or templates. Finally, development tasks are identified and a project schedule is created.
- **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.
- **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.

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

Team Software Process (TSP)

- The goal of TSP is to build a "**self-directed project team**" to produce high-quality software.
- The following are the objectives for TSP:
  - **Build self-directed teams** that **plan and track** their work, **establish goals**, and **own their processes and plans**. These can be pure software teams or integrated product teams(IPT) of 3 to about 20 engineers.
  - Show managers how to **coach and motivate** their teams and how to help them **sustain peak performance**.
  - Accelerate software process
  - Provide improvement guidance to high-maturity organizations.

- A self-directed team defines
  - **roles and responsibilities** for each team member
  - **tracks quantitative project data**
  - identifies a team process appropriate for project
  - a **strategy** for implementing the process
  - defines **local standards** that are applicable to the teams software engineering work;
  - **continually assesses risk** and reacts to it
  - **Tracks, manages, and reports project status**.

- Each project is “launched” using a sequence of tasks.
- The following **launch script** is recommended
  - Review project objectives with management and agree on and document team goals.
  - Establish team roles
  - Define the teams development process
  - Make a quality plan and set quality targets
  - Plan for the needed support facilities