Unit-1–

IntroductiontoSoftware Engineering

Definition of Software:

Software can be considered as a dual role. It is a product and, at the same time, the vehicle for delivering a product.

As a product, it delivers the computing potential embodied by computer hardware.

Ex: A network of computers that is accessible by local hardware. Whether it resides within a cellular phone or operates inside a mainframe computer,

As the 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).

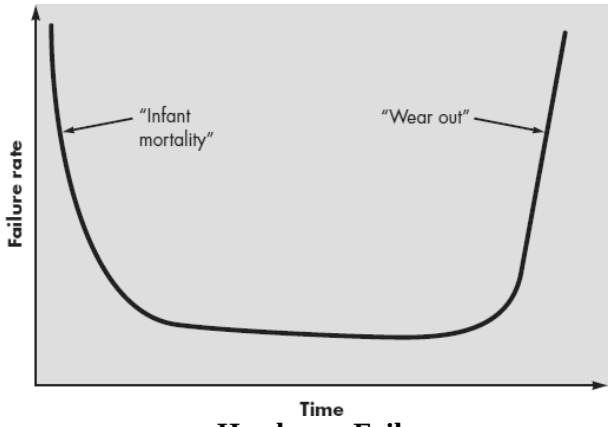
Characteristics of software:

Software is developed or engineered; it is not manufactured in the classical sense:

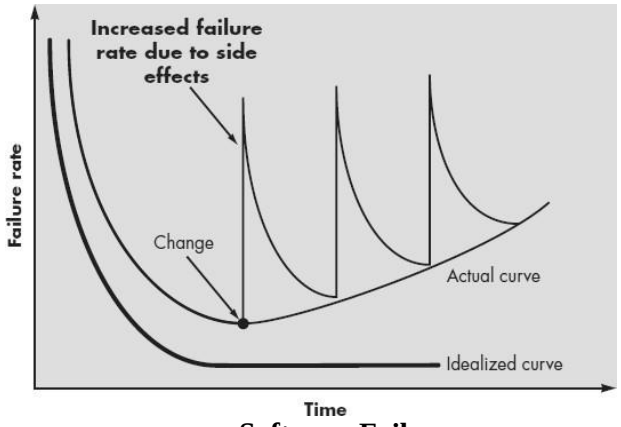
- Although some similarities exist between software development and hardware manufacturing, but few activities are fundamentally different.
- In both activities, high quality is achieved through good design, but the manufacturing phase for hardware can introduce quality problems than software.

Software doesn't "wear out."

- Hardware components suffer from the growing effects of dust, vibration, abuse, temperature extremes, and many other environmental maladies. Stated simply, the hardware begins to wear out.
- Software is not susceptible to the environmental maladies that cause hardware to wear out. In theory, therefore, the failure rate curve for software should take the form of the "idealized curve".
- 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.
- However, the implication is clear—software doesn't wear out. But it does deteriorate.



Hardware Failure curve



Software Failure curve

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.

- In the hardware world, component reuse is a natural part of the engineering process.

Application areas:

The 7 broad categories of computer software present continuing challenges for software engineers:

System software: System software is a collection of programs written to service other programs.
The systems software is characterized by

- heavy interaction with computer hardware
- heavy usage by multiple users
- concurrent operation that requires scheduling, resource sharing, and sophisticated process management
- complex data structures
- multiple external interfaces

compilers, editors and file management utilities.

Application software:

Application software consists of standalone programs that solve a specific business need.

- It facilitates business operations or management/technical decision making.
- It is used to control business functions in real-time

point-of-sale transaction processing, real-time manufacturing process control.

Engineering/Scientific software: Engineering and scientific applications range

- from astronomy to volcanology
- from automotive stress analysis to space shuttle orbital dynamics
- from molecular biology to automated manufacturing

computer aided design, system simulation and other interactive applications.

Embedded software:

- 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.
- It can perform limited and esoteric functions or provide significant function and control capability.

E.g. Digital functions in automobile, dashboard displays, braking systems etc.

Product-line software:

Designed to provide a specific capability for use by many different customers, product-line software can focus on a limited and esoteric market place or address mass consumer markets

E.g. Word processing, spreadsheets, computer graphics, multimedia, entertainment, database management, personal and business financial applications

Web-applications:

WebApps are evolving into sophisticated computing environments that not only provide standalone features, computing functions, and content to the end user, but also are integrated with corporate databases and business applications.

Artificial intelligence software:

AI software makes use of non numerical algorithms to solve complex problems that are not amenable to computation or straightforward analysis. Application within this area includes robotics, expert systems, pattern recognition, artificial neural networks, theorem proving, and game playing.

Software Myths:

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

Management Myths:

Myth:

We already have a book that's full of standards and procedures for building software - Wont 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?

Myth:

If we get behind schedule, we can add more programmers and catch up.

Reality:

Software development is not a mechanistic process like manufacturing. As new people are added, people who were working must spend time educating the new comers, thereby reducing the amount of time spend 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 built it.

Reality:

If an organization does not understand how to manage and control software projects internally, it will invariably struggle when it outsources software projects.

Customer Myths:

The customer believes myths about software because software managers and practitioners do little to correct misinformation. Myths lead to false expectations and ultimately, dissatisfaction with the developer.

Myth:

A general statement of objectives is sufficient to begin with 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 recipe for disaster.

Myth:

Project 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 and change can cause upheaval that requires additional resources and major design modification.

Practitioner's myths:

Myths that are still believed by software practitioners: during the early days of software, programming was viewed as an art from old ways and attitudes die hard.

Myth:

Once we write the program and get it to work, our jobs are 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. Documentation provides guidance for software support.

Myth:

software engineering will make us create voluminous and unnecessary documentation and will invariably slows down.

Reality:

software engineering is not about creating documents. It is about creating quality. Better quality leads to reduced rework. And reduced rework results in faster delivery times.

Definition of Software Engineering:

IEEE defines software engineering as:

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

Fritz Bauer, a German computer scientist, defines software engineering as:

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

SOFTWARE ENGINEERING - A LAYERED TECHNOLOGY:

Software engineering is a layered technology. Any engineering approach must rest on an organizational commitment to quality. **The bedrock that supports software engineering is a quality focus.**

The foundation for software engineering is the process layer. **Process defines a framework that must be established for effective delivery of software engineering technology.**

The software forms the basis for management control of software projects and establishes the context in which

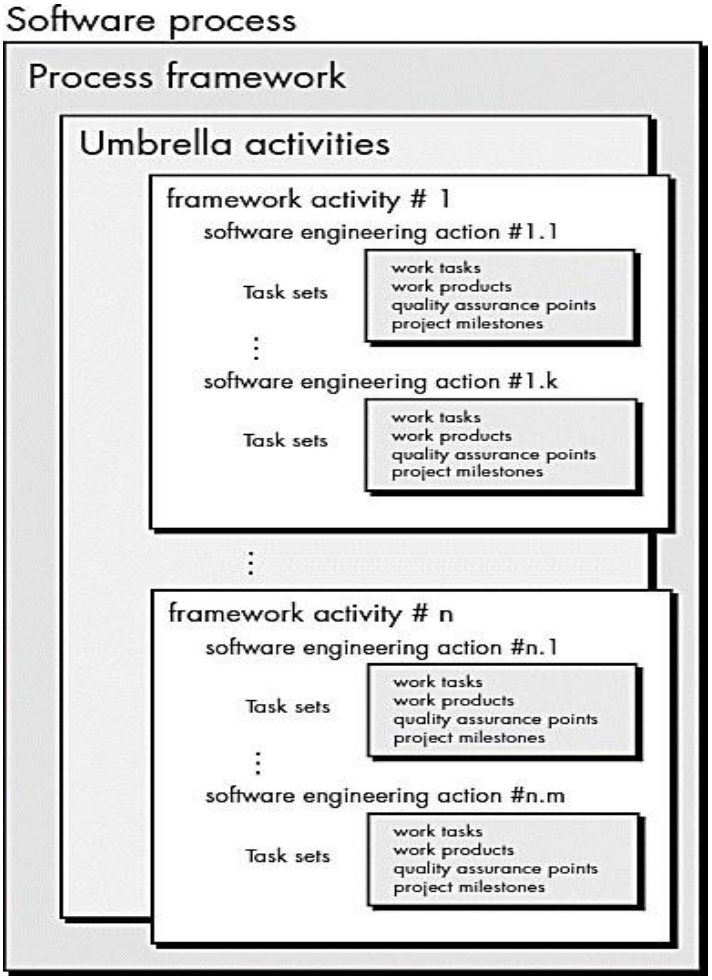
- technical methods are applied,
- work products are produced,
- milestones are established,
- quality is ensured,
- And change is properly managed.

Software engineering methods rely on a set of basic principles that govern area of the technology and include modeling activities.

Methods encompass a broad array of tasks that include

- communication,
- requirements analysis,
- design modeling,
- program construction,
- Testing and support.

Software engineering tools provide automated or semi automated support for the process and the methods. When tools are integrated so that information created by one tool can be used by another, a system for the support of software development, called computer-aided software engineering, is established.



A PROCESS FRAMEWORK:

- **Software process** must be established for effective delivery of software engineering technology.
- A **process framework** establishes the foundation for a complete software process by identifying a small number of framework activities that are applicable to all software projects, regardless of their size or complexity.
- The process framework encompasses a set of **umbrella activities** that are applicable across the entire software process.
- Each **framework activity** is populated by a set of software engineering actions
- Each **software engineering action** is represented by a number of different task sets- each a collection of software engineering work tasks, related work products, quality assurance points, and project milestones.

In brief

"A process defines who is doing what, when, and how to reach a certain goal."

A Process Framework

- establishes the foundation for a complete software process
- identifies a small number of framework activities

- applies to all s/w projects, regardless of size/complexity.
- also, set of umbrella activities
- applicable across entire s/w process.
- Each framework activity has
- set of s/w engineering actions.
- Each s/w engineering action (e.g., design) has
- collection of related tasks (called task sets): work tasks ,work products (deliverables) quality assurance points project milestones.

Generic Process Framework: A generic process framework for software engineering defines five framework activities— communication, planning, modeling, construction, and deployment. In addition, it defines a set of umbrella activities.

1**) Communication**: This framework activity involves heavy communication and collaboration with the customer and encompasses requirements gathering and other related activities.

2) Planning: This activity establishes a plan for the software engineering work that follows. It describes the technical tasks to be conducted, the risks that are likely, the resources that will be required, the work products to be produced, and a work schedule.

3) **Modeling:** This activity encompasses the creation of models that allow the developer and customer to better understand software requirements and the design that will achieve those requirements.

4**) Construction:** This activity combines core generation and the testing that is required to uncover the errors in the code.

5**) Deployment:** The software is delivered to the customer who evaluates the delivered product and provides feedback based on the evolution.

These 5 generic framework activities can be used during the development of small programs, the creation of large web applications, and for the engineering of large, complex computer-based systems.

The following are the set of Umbrella Activities.

1) **Software project tracking and control –** allows the software team to assess progress against the project plan and take necessary action to maintain schedule.

2**) Risk Management** - assesses risks that may effect the outcome of the project or the quality of the product.

3) **Software Quality Assurance** - defines and conducts the activities required to ensure software quality.

4) **Formal Technical Reviews -** assesses software engineering work products in an effort to uncover and remove errors before they are propagated to the next action or activity.

5) **Measurement** - define and collects process, project and product measures that assist the team in delivering software that needs customer's needs, can be used in conjunction with all other framework and umbrella activities.

6) **Software configuration management** - manages the effects of change throughout the software process.

7) **Reusability management -** defines criteria for work product reuse and establishes mechanisms to achieve reusable components.

8**) Work Product preparation and production -** encompasses the activities required to create work products such as models, document, logs, forms and lists.

PROCESS PATTERNS

The software process can be defined as a collection patterns that define a set of activities, actions, work tasks, work products and/or related behaviors required to develop computer software.

Type: The pattern type is specified. There are three types

1**. Task patterns** define a software engineering action or work task that is part of the process and relevant to successful software engineering practice.

Example: Requirement Gathering

2. **Stage Patterns** define a framework activity for the process. This pattern incorporates multiple task patterns that are relevant to the stage.

Example: Communication

3. **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: Spiral model or prototyping.

Process assessment

The existence of a software process is no guarantee that software will be delivered on time, that it will

meet the customer's needs, or that it will exhibit the technical characteristics that will lead to long-term quality characteristics. In addition, the process itself should be assessed to be essential to ensure that it meets a set of basic process criteria that have been shown to be essential for a successful software engineering.

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

Standards CMMI Assessment Method for Process Improvement (SCAMPI) provides a five step process assessment model that incorporates initiating, diagnosing, establishing, acting & learning. The SCAMPI method uses the SEI CMMI as the basis for assessment.

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 as the basis for the assessment.

SPICE (ISO/IEC15504) standard defines a set of requirements for software process assessments. 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, system, or services that it provides. Therefore, the standard is directly applicable to software organizations &companies.

Process Models

Prescriptive process models define a set of activities, actions, tasks, milestones, and work products that are required to engineer high-quality software. These process models are not perfect, but they do provide a useful roadmap for software engineering work.

Following are prescriptive process models:

Waterfall model

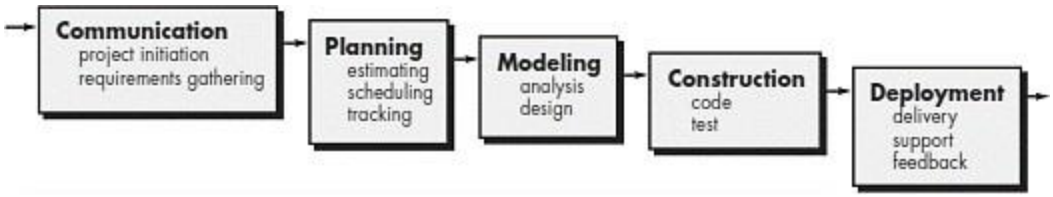
Incremental Models

- Incremental Model
- RAD Model

Evolutionary Process Model

- Prototype Model
- Spiral Model
- Concurrent Process Model

The waterfall model:



The waterfall model, sometimes called the classic life cycle/Linear, suggests a systematic sequential approach to software development that begins with customer specification of requirements and progresses through planning, modeling, construction, and deployment.

Advantage:

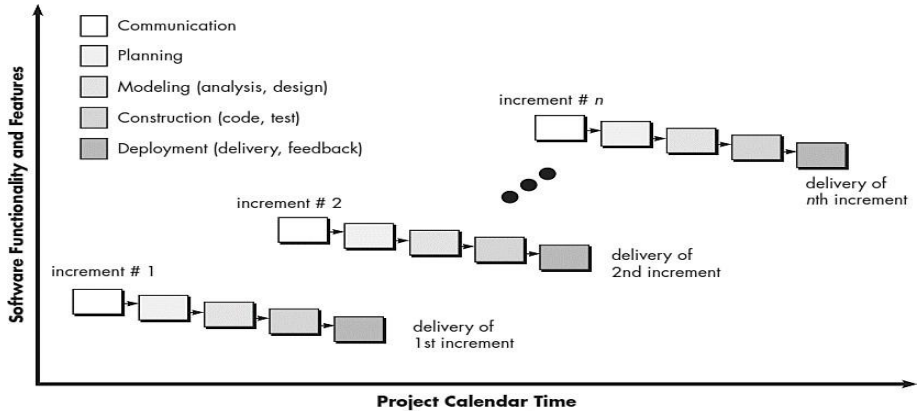
It can serve as a useful process model in situations where requirements are fixed and work is to proceed to complete in a linear manner.

The problems that are sometimes encountered when the waterfall model is applied are:

1. Real projects rarely follow the sequential flow that the model proposes. Although the linear model can accommodate iteration, it does so indirectly. As a result, changes can cause confusion as the project team proceeds.
2. It is often difficult for the customer to state all requirements explicitly. The waterfall model requires this and has difficulty accommodating the natural uncertainty that exist at the beginning of many projects.
3. The customer must have patience. A working version of the programs will not be available until late in the project time-span. If a major blunder is undetected then it can be disastrous until the program is reviewed.

The Incremental Model:

Incremental development is particularly useful when staffing is unavailable for a complete implementation by the business deadline that has been established for the project. Early increments can be implemented with fewer people. If the core product is well received, additional staff can be added to implement the next increment. In addition, increments can be planned to manage technical risks.



The incremental model combines elements of the waterfall model applied in an iterative fashion.

The incremental model delivers a series of releases called increments that provide progressively more functionality for the customer as each increment is delivered.

When an incremental model is used, the first increment is often a core product. That is, basic requirements are addressed. The core product is used by the customer. As a result, a plan is developed for the next increment.

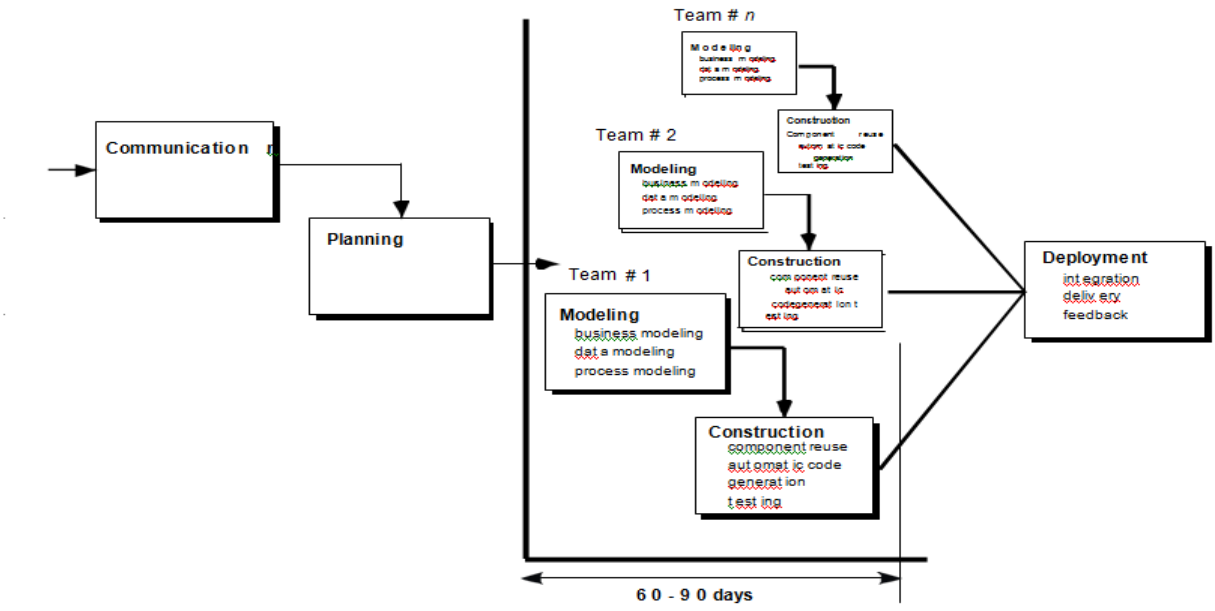
The plan addresses the modification of the core product to better meet the needs of the customer and the delivery of additional features and functionality.

This process is repeated following the delivery of each increment, until the complete product is produced.

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.

The Rad Model:

Rapid Application Development (RAD) is an incremental software process model that emphasizes a short development cycle. The RAD model is a “high-speed” adaption of the waterfall model, in which rapid development is achieved by using a component base construction approach.
The RAD approach maps into the generic framework activities.



Communication works to understand the business problem and the information characteristics that the software must accommodate.

Planning is essential because multiple software teams works in parallel on different system functions.

Modeling addresses three major phases- business modeling, data modeling and process modeling- and establishes design representation that serve the application of automatic code generation.

Deployment establishes a basis for subsequent iterations.

The RAD approach has drawbacks:

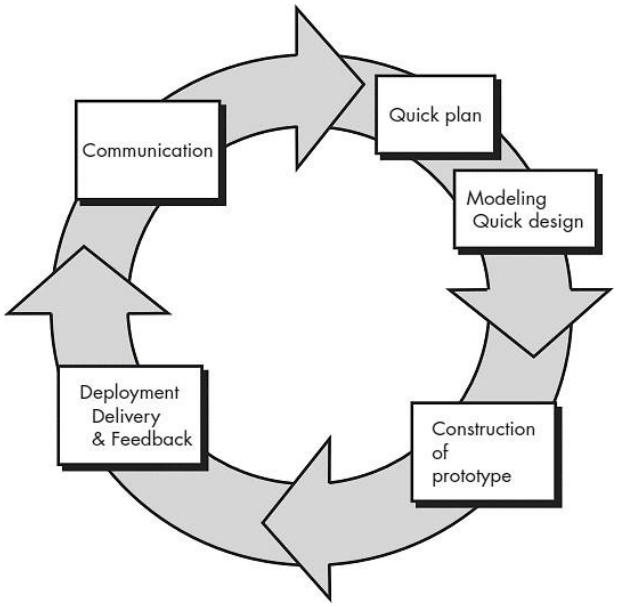
For large, but scalable projects, RAD requires sufficient human resources to create the right number of RAD teams.
If developers and customers are not committed to the rapid-fire activities necessary to complete the system in a much abbreviated time frame, RAD projects will fail
If a system cannot be 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; and
RAD may not be appropriate when technical risks are high.

Evolutionary Process Models:

Evolutionary models are iterative. They are characterized in a manner that enables software engineers to develop increasingly more complete versions of the software.

Prototyping:

- Prototyping is more commonly used as a technique that can be implemented within the context of anyone of the process model.
- The prototyping paradigm begins with communication. The software engineer and customer meet and 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. The quick design leads to the construction of a prototype. The prototype is deployed and then evaluated by the customer/user.
- Iteration occurs as the prototype is tuned to satisfy the needs of the customer, while at the same time enabling the developer to better understand what needs to be done.



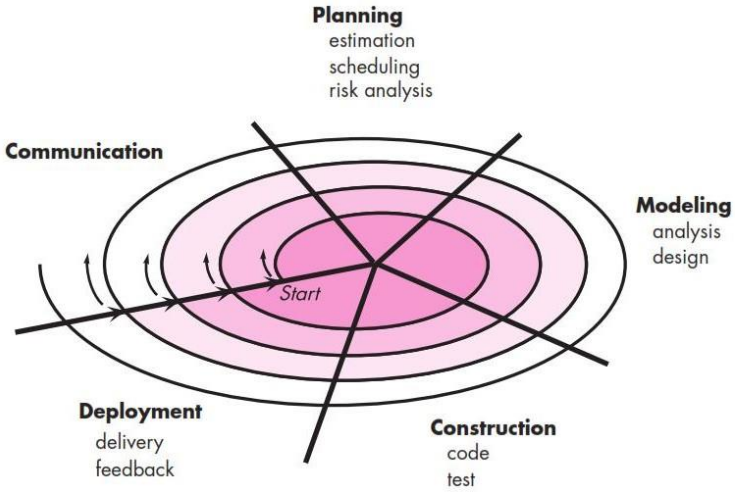
If a customer defines a set of general objectives for software, but does not identify detailed input, processing, or output requirements, in such situation prototyping paradigm is best approach.
If a developer may be unsure of the efficiency of an algorithm, the adaptability of an operating system then he can go for this prototyping method.

Advantages:

- The prototyping paradigm assists the software engineer and the customer to better understand what is to be built when requirements are fuzzy.
- The prototype serves as a mechanism for identifying software requirements. If a working prototype is built, the developer attempts to make use of existing program fragments or applies tools.

The Spiral Model

- The spiral model, originally proposed by Boehm, is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model.
- The spiral model can be adapted to apply throughout the entire life cycle of an application, from concept development to maintenance.
- Using the spiral model, software is developed in a series of evolutionary releases. During early iterations, the release might be a paper model or prototype. During later iterations, increasingly more complete versions of the engineered system are planning estimation scheduling risk analysis



- The first circuit around the spiral might result in the development of product specification; subsequent passes around the spiral might be used to develop a prototype and then progressively more sophisticated versions of the software.
- Each pass through the planning region results in adjustments to the project plan. Cost and schedule are adjusted based on feedback derived from the customer after delivery. In addition, the project manager adjusts the planned number of iterations required to complete the software.
- It maintains the systematic stepwise approach suggested by the classic life cycle but incorporates it into an iterative framework that more realistically reflects the real world.
- The first circuit around the spiral might represent a "concept development project" which starts at the core of the spiral and continues for multiple iterations until concept development is complete.
- If the concept is to be developed into an actual product, the process proceeds outward on the spiral and a "new product development project" commences.
- Later, a circuit around the spiral might be used to represent a "product enhancement project." In essence, the spiral, when characterized in this way, remains operative until the software is retired.

Advantages:

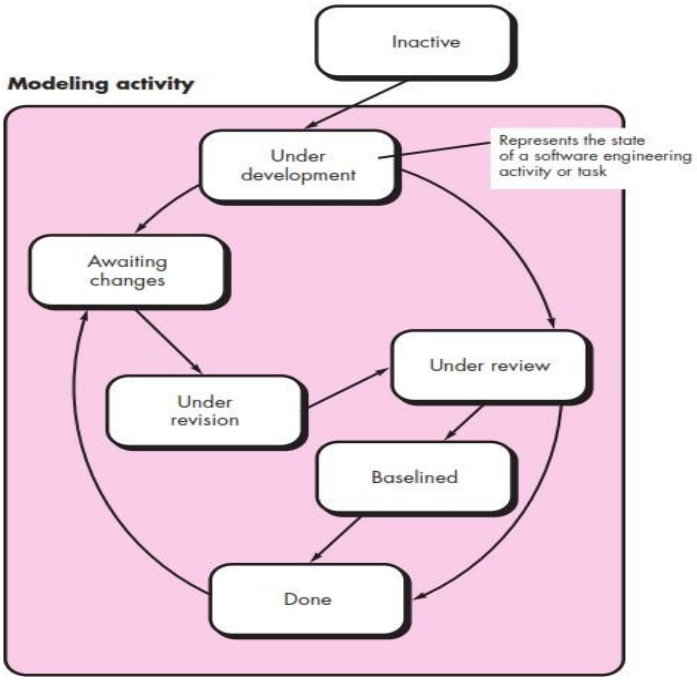
It provides the potential for rapid development of increasingly more complete versions of the software.

Draw Backs:

It may be difficult to convince customers that the evolutionary approach is controllable. 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.

The Concurrent Development Model:

The concurrent development model, sometimes called concurrent engineering, can be represented schematically as a series of framework activities, software engineering actions and tasks, and their associated states.



The activity modeling may be in anyone of the states noted at any given time. Similarly, other activities or tasks can be represented in an analogous manner. All activities exist concurrently but reside in different states.

Any of the activities of a project may be in a particular state at any one time:

- under development
- awaiting changes
- under revision
- under review

In a project the communication activity has completed its first iteration and exists in the awaiting changes state. The modeling activity which existed in the none state while initial communication was completed, now makes a transition into the under development state. If, however, the customer indicates that changes in requirements must be made, the modeling activity moves from the under development state into the awaiting changes state.

The concurrent process model defines a series of events that will trigger transitions from state to state for each of the software engineering activities, actions, or tasks.

The event analysis model correction which will trigger the analysis action from the done state into the awaiting changes state.

Advantages:

- The concurrent process model is applicable to all types of software development and provides an accurate picture of the current state of a project.

- It defines a network of activities rather than each activity, action, or task on the network exists simultaneously with other activities, action and tasks.

The Unified Process:

The Unified process recognizes the importance of customer communication and streamlined methods for describing the customer's view of a system. It emphasizes the important role of software architecture and "helps the architect focus on the right goals, such as understandability, reliance to future changes, and reuse".

Phases of The Unified Process:

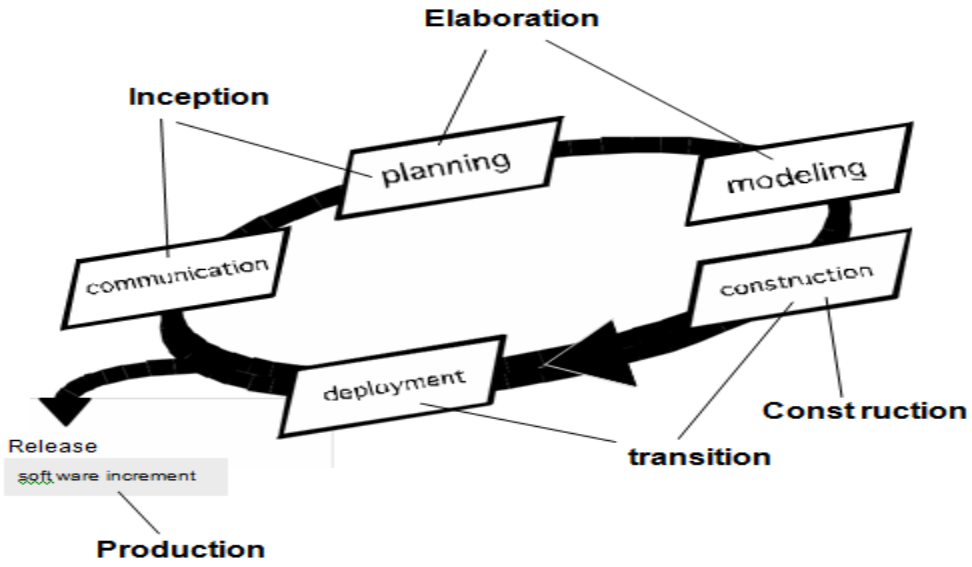
The inception phase of the UP encompasses both customer communication and planning activities. By collaborating with the customer and end-users, business requirements for the software are identified, a rough architecture for the system is proposed and a plan for the iterative, incremental nature of the ensuing project is developed.

The elaboration phase encompasses the customer communication and modeling activities of the generic process model. Elaboration refines and expands the preliminary use-cases that were developed as part of the inception phase and expands the architectural representation.

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 or acquires the software components that will make each use-case operational for end-users.

The transition phase of the UP encompasses the latter stages of the generic construction activity and the first part of the generic deployment activity. Software given to end-users for user feedback reports both defects and necessary changes.

The production phase of the UP coincides with the deployment activity of the generic process. During this phase, the on-going use of the software is monitored, support for the operating environment is provided, and defect reports and requests for changes are submitted and evaluated.



Agility and Agile Process Model

Agility:

- Agility is dynamic, content specific, aggressively change embracing, and growth oriented
- It encourages team structures and attitudes that make communication (among team members, between technologists and business people, between software engineers and their managers) more simplistic.
- It emphasizes rapid delivery of operational software and de-emphasizes the importance of intermediate work products.
- It recognizes that planning in an uncertain world has its limits and that a project plan must be flexible.
- Agility can be applied to any software process.

Agility Principles:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be

able to maintain a constant pace indefinitely.

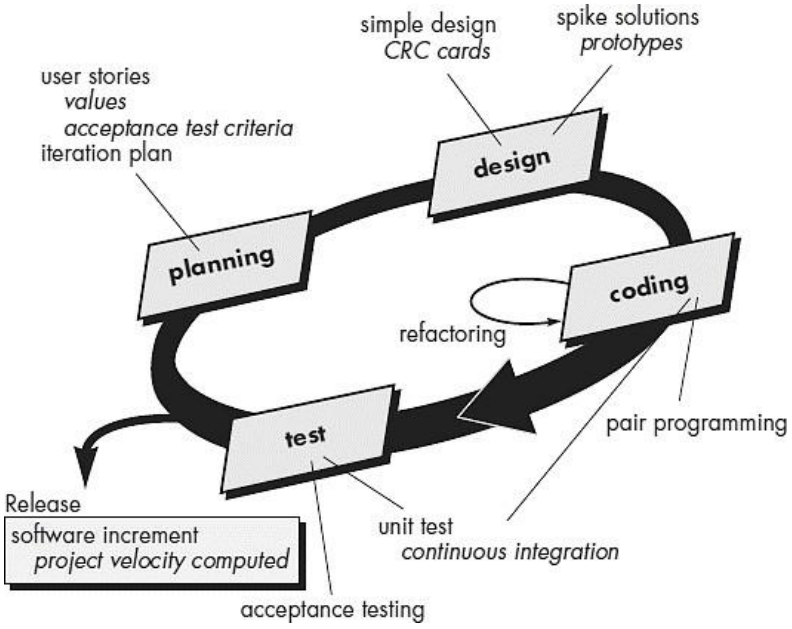
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity—the art of maximizing the amount of work not done—is essential.
11. The best architectures, requirements, and designs emerge from self– organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agile Process Models:

- Extreme Programming (XP)
- Adaptive Software Development (ASD)
- Scrum
- Crystal
- Feature Driven Development (FDD)
- Agile Modeling (AM)

Extreme Programming

- It is most widely used agile process model.
- XP uses an object-oriented approach as its preferred development paradigm.
- It defines four (4) framework activities Planning, Design, Coding, and Testing.



Planning:

- Begins with the creation of a set of stories (also called user stories)
- Each story is written by the customer and is placed on an index card
- The customer assigns a value (i.e. a priority) to the story
- Agile team assesses each story and assigns a cost
- Stories are grouped to for a deliverable increment
- A commitment is made on delivery date

- After the first increment “project velocity” is used to help define subsequent delivery dates for other increments

Design:

- Follows the keep it simple principle
- Encourage the use of CRC (class-responsibility-collaborator) cards
- For difficult design problems, suggests the creation of “spike solutions”—a design prototype
- Encourages “refactoring”—an iterative refinement of the internal program design
- Design occurs both before and after coding commences

Coding:

Encourages “pair programming”

- Developers work in pairs, checking each other's work and providing the support to always do a good job.
- Mechanism for real-time problem solving and real-time quality assurance
- Keeps the developers focused on the problem at hand

Needs continuous integration with other portions (stories) of the s/w, which provides a “smoke testing” environment

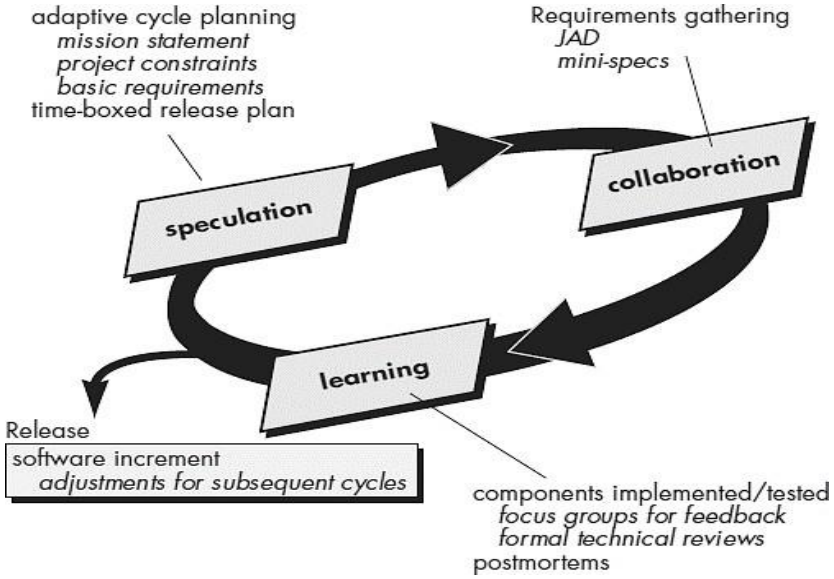
Testing:

Tests should be implemented using a framework to make testing automated.

- Acceptance tests, also called customer tests, are specified by the customer and executed to assess customer visible functionality
- Acceptance tests are derived from user stories

Adaptive Software Development (ASD)

Adaptive Software Development (ASD) is a technique for building complex software and systems. ASD focus on human collaboration and team self-organization. ASD incorporates three phases Speculation, Collaboration, and Learning.



Speculation:

- The big idea behind speculate is when we plan a product to its smallest detail as in a requirements

up front Waterfall variant, we produce the product we intend and not the product the customer needs.

- In the ASD mindset, planning is to speculation as intention is to need.

Collaboration:

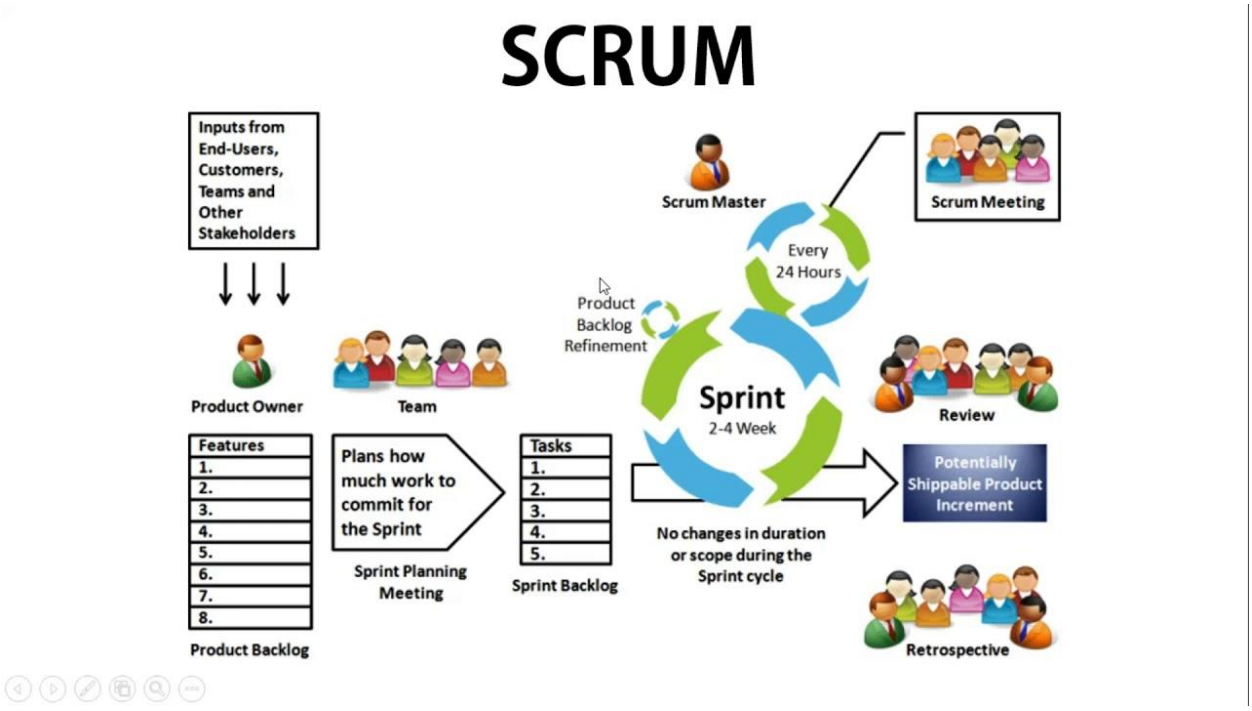
- Collaboration represents a balance between managing the doing and creating and maintaining the collaborative environment."
- Speculation says we can't predict outcomes. If we can't predict outcomes, we can't plan. If we can't plan, traditional project management theory suffers.
- Collaboration weights speculation in that a project manager plans the work between the predictable parts of the environment and adapts to the uncertainties of various factors—stakeholders, requirements, software vendors, technology, etc.

Learning:

- Based on short iterations of design, build and testing, knowledge accumulates from the small mistakes we make due to false assumptions, poorly stated or ambiguous requirements or misunderstanding the stakeholders' needs.
- Correcting those mistakes through shared learning cycles leads to greater positive experience and eventual mastery of the problem domain.

Scrum:

Scrum is an innovative approach to getting work done in efficient way. It is iterative & incremental agile software development method. These iterations are time boxed with various iterations & each iteration is called Sprint. The Sprint is basically 2-4 week long & each sprint requires sprint planning estimation. Scrum is the most popular agile project management methodology in software development. The term Scrum is formed from Rugby.



Each **Agile Development Scrum** team having three core scrum roles: Product Owner, Scrum Master & The Team.

1) Product Owner: The Product Owner is the person who represents the stakeholders and is the voice of the customer. Product owner writes the User Stories, ordered priorities and add in the **Product Backlog.**

2) Scrum Master: The **Scrum Master** is a facilitative team leader who ensures that the team adheres to its chosen process and removes blocking issues to deliver the sprint deliverable/goal. Scrum Master is not a team leader but act as a shield for the team from external interference's & also removes barriers. ScrumMaster facilitates the daily scrums.

3) The Team: The *scrum development* team is generally size of 5-9 peoples with self-organizing and cross-functional skills who do actual work like Analysis, Design, Development, Testing, Documentation etc.

Scrum methodology advocates for a planning meeting at the start of the sprint, where team members figure out how many items they can commit to, and then create a sprint backlog – a list of the tasks to perform during the sprint.

During an agile Scrum sprint, the Scrum team takes a small set of features from idea to coded and tested functionality. At the end, these features are done, meaning coded, tested and integrated into the evolving product or system.

On each day of the sprint, all team members should attend a daily Scrum meeting, including the ScrumMaster and the product owner. This meeting is timeboxed to no more than 15 minutes. During that time, team members share what they worked on the prior day, will work on that day, and identify any impediments to progress.

The Scrum model sees daily scrums as a way to synchronize the work of team members as they discuss the work of the sprint.

At the end of a sprint, the team conducts a sprint review during which the team demonstrates the new functionality to the PO or any other stakeholder who wishes to provide feedback that could influence the next sprint.

This feedback loop within Scrum software development may result in changes to the freshly delivered functionality, but it may just as likely result in revising or adding items to the product backlog.

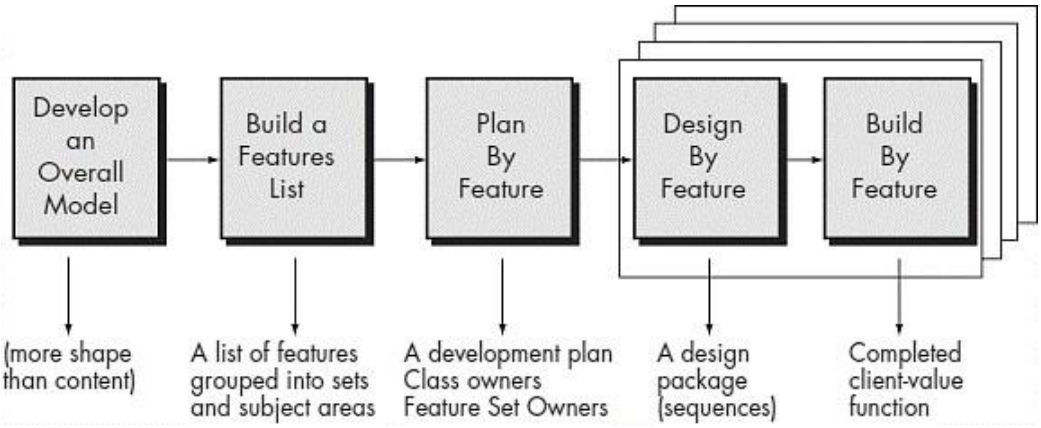
Another activity in Scrum project management is the sprint retrospective at the end of each sprint. The whole team participates in this meeting, including the ScrumMaster and PO. The meeting is an opportunity to reflect on the sprint that has ended, and identify opportunities to improve.

Crystal

Crystal method is an agile software development approach that focuses primarily on people and their interactions when working on a project rather than on processes and tools. Alistair believed that the people's skills and talents as well as the way they communicate has the biggest impact on the outcome of the project.

Crystal is actually comprised of a family of agile methodologies such as Crystal Clear, Crystal Yellow, Crystal Orange and others, whose unique characteristics are driven by several factors such as team size, system criticality, and project priorities.

Feature Driven Development(FDD)



FDD is a model-driven, short-iteration process.

It begins with establishing an overall model shape.

Then it continues with a series of two-week "design by feature, build by feature" iterations.

The features are small, "useful in the eyes of the client" results.

FDD designs the rest of the development process around feature delivery using the following eight practices:

- Domain Object Modeling
- Developing by Feature
- Component/Class Ownership
- Feature Teams
- Inspections
- Configuration Management
- Regular Builds
- Visibility of progress and results

Unlike other agile methods, FDD describes specific, very short phases of work, which are to be accomplished separately per feature.

These include Domain Walkthrough, Design, Design Inspection, Code, Code Inspection, and Promote to Build.