VARDHAMAN COLLEGE OF ENGINEERING

(AUTONOMOUS)

Affiliated to **JNTUH,** Approved by **AICTE,** Accredited by **NAAC** with **A++** Grade, **ISO 9001:2015** Certified

Kacharam, Shamshabad, Hyderabad - 501218, Telangana, India

Department of Information Technology

UNITWISE NOTES Of SOFTWARE ENGINEERING

(II B. Tech - II SEMESTER)

(A6603_VCE-R20)

Hours Per Week			Hours Per Semester			Credits	Assessment Marks		
L	**T**	**P**	**L**	**T**	**P**	**C**	**CIE**	**SEE**	**Total**
2	0	2	28	0	28	3	30	70	100

B. Tech

(For batches admitted from the A.Y 2021 - 2022)

April 2022

LECTURE NOTES UNIT WISE

UNIT -1

Introduction to Software Engineering

Software and Software Engineering: The Nature of Software, Defining the Discipline, The Software Process, Software Engineering Practice. Process Models: A Generic Process Model, Defining a Framework Activity, Identifying a Task Set, Process Assessment and Improvement, Prescriptive Process Models, Product and process.

****

1.1 Nature of Software:

1.1.1 Defining the Discipline

* **Software** is a program or set of programs containing instructions that provide desired functionality. And

* **Engineering** is the process of designing and building something that serves a particular purpose and finds a cost-effective solution to problems.

* ***Software Engineering***

According to IEEE,

Software engineering is a systematic, disciplined, quantifiable study and approach to the design, development, operation, and maintenance of a software system.

According to Pressman,

- Instructions that mean computer programs that when executed provide desired features, function and performance.
- Data structures that enables the program to adequately manipulate information and,
- Descriptive information in both hardcopy and virtual forms that describes the operations and use of the programs.

* The characteristics of software are different from characteristics of hardware:

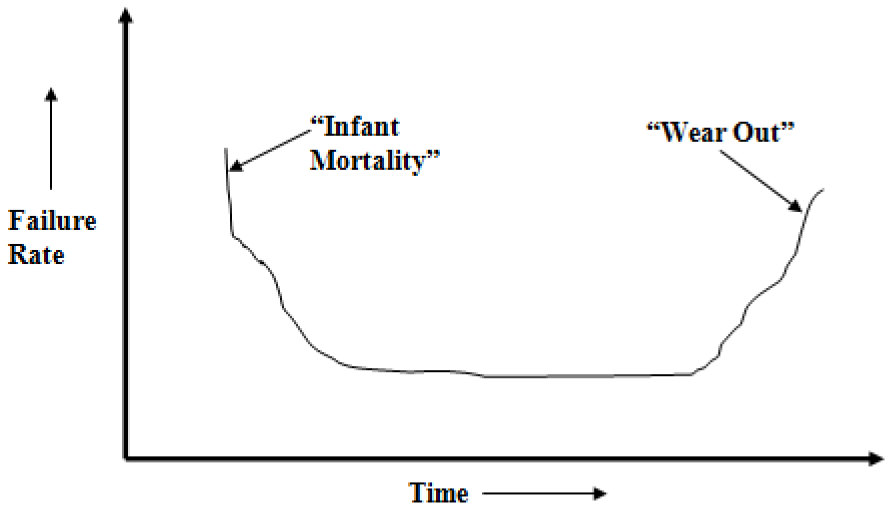
(1) Software is developed (Or) engineered; it is not manufactured in the classical sense

- ✓ In both software development and hardware manufacturing **high quality** is achieved using good design
- ✓ The manufacturing phase for hardware can introduce quality problems, which are **non-existent** (Or easily corrected) for software

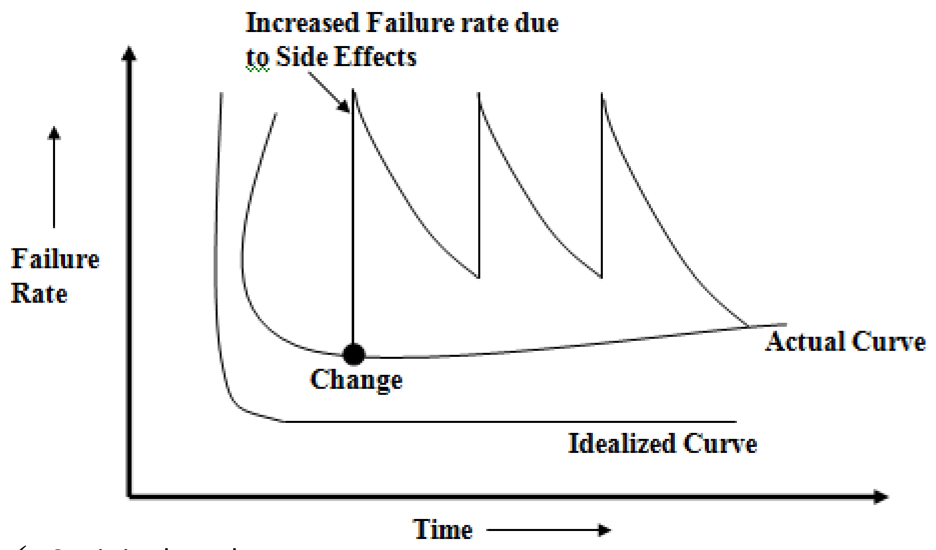
- ✓ Both the activities **dependent on people**, but the relationship between people applied and work accomplished is entirely different
- ✓ Both activities require the construction of a **"Product"**, but the approaches are different
- ✓ Software engineering costs are concentrated in engineering.

(2) Software doesn't "wear out"

- ✓ The hardware exhibits relatively **high failure rates** early in its life, these failures are caused due to design (Or) manufacturing defects
- ✓ Once defects are corrected the failure rate drops to a steady-state level, for some period of time
- ✓ As time passes, hardware components suffer from the cumulative effects of Dust, Vibration, abuse, temperature extremes and other environmental maladies
- ✓ The failure rate rises again, stated simply the hardware begins to **"wear out"**. This is shown below using **"Bath Tub Curve".**



- ✓ Software is not susceptible to the environmental maladies
- ✓ The failure rate curve for software, should take the form of "Idealized Curve" .
- ✓ Early in the life of a program, undiscovered errors causes high failure rates
- ✓ Once these errors are corrected (without introducing other errors), the curve
    - o Flattens



- ✓ So, it is clear that;
    - Software **doesn't wear out**, but it does **deteriorate**
- ✓ Software will undergoes changes during its life time
- ✓ The errors will be introduced as changes are made. This causes the failure rate to spike
- ✓ Before the curve can return to the original steady state failure, another changes is requested causing the curve to spike again
- ✓ So due to changes the software is deteriorating
- ✓ The software maintenance involves more complexity than hardware maintenance because,

=> When hardware components wear out, it is replaced by spare part, but there are no software parts

(3) Although the industry is moving toward component – based construction, most software continues to be custom built

- ✓ In hardware world, component reuse is a natural part of the engineering process
- ✓ But in software world it has to be achieved on a broad scale
- ✓ A software component should be designed and implemented that can be reused in different programs

<u>Example:</u>

- ✓ Today's user interfaces built with reusable components that enable;

=> Creation of windows
=> Pull-down Menus &
=> Wide variety of interaction mechanism.

1.1.2 Software Domains

There are seven broad categories of computer software, that continuing challenges for software engineers are:

(1) System Software:

It is a collection of programs **written to service other programs**

=> Some system software processes complex, but determinate information structure and some other processes largely indeterminate data

Example:

=> Compilers, editors and file management utilities

Characteristics:

=> Heavy interaction with computer hardware

=> Heavy usage by multiple users

=> Complex data structures

=> Multiple external interfaces

=> Concurrent operation.

(2) Application Software:

- ✓ It is a **standalone program**, that solve a specific business need
- ✓ This software process business (Or) Technical decision making. In addition it is used to control business functions in real time

Example:

=> Point – of – sale transaction processing

=> Real – time manufacturing process control

(3) Engineering / Scientific Software:

- ✓ This software used in various applications such as;

  => Astronomy

  => Molecular biology

  => Automated Manufacturing etc.
- ✓ Modern application within the scientific / engineering area is moving away from conventional numerical algorithm
- ✓ Computer aided design, system simulation and other interactive application, begun to take on real – time

(4) Embedded Software:

- ✓ This Software resides **within a product (Or) System**
- ✓ It is used to **implement and control** features and functions for the end user and for the system itself

Example:

=> Keypad control for a microwave oven

=> Digital Functions in an automobile such as fuel control, dash board

Displays and braking system etc.,

(5) <u>Product-line Software:</u>

- ✓ It is designed to **provide a specific capability**, for use by many different customers
- ✓ This software **focuses on esoteric market place** and address mass consumer market

<u>Example:</u>

=> Word processing
=> Spread sheets
=> Computer Graphics
=> Multimedia and entertainment etc.,

(6) <u>Web Applications:</u>

- ✓ This Software span a wide array of applications
- ✓ Web applications are evolving into sophisticated computing environment, that not only provide stand alone features, computing functions and content to end users, but also integrate corporate database and business application

(7) <u>Artificial Intelligence Software:</u>

- ✓ This software makes use of **non numerical algorithms**, to solve complex problems
- ✓ Applications with in this area include;

=> Robotics
=> Expert Systems
=> Pattern recognition
=>Theorem proving and game playing.

****

1.1.4 Legacy Software's:

* This software system developed decades ago [i.e., **older programs**] and it have been continually modified to meet changes in business requirements and computing platforms

* The Proliferation of such systems is causes headache for large organizations who find them costly to maintain and risky to evolve.

Legacy systems often evolve for one or more of the following reasons:

• The software must be adapted to meet the needs of new computing environments or technology.

• The software must be enhanced to implement new business requirements.

• The software must be extended to make it interoperable with other more modern systems or databases.

• The software must be re-architected to make it viable within a network environment.

The Evolving Role of Software

* Today software takes a **DUAL ROLE**

=> **As a Product**

=> A vehicle for delivering a product

1. As a product –

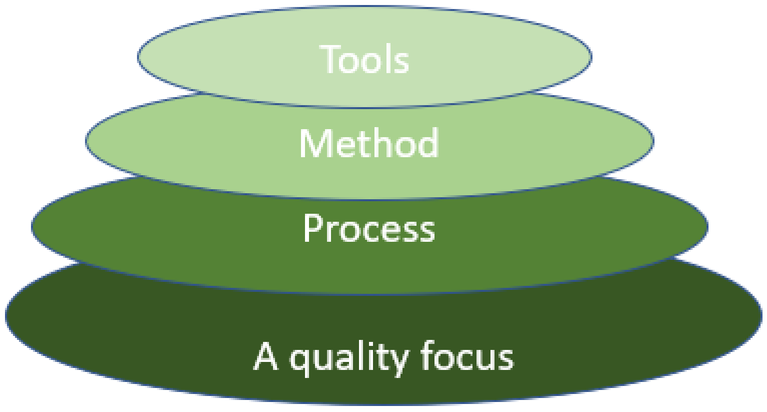
- ✓ It delivers the computing potential across networks of Hardware.
- ✓ It enables the Hardware to deliver the expected functionality.
- ✓ It acts as an information transformer because it produces, manages, acquires, modifies, displays, or transmits information.

2. As a vehicle for delivering a product –

- ✓ It provides system functionality (e.g., payroll system)
- ✓ It controls other software (e.g., an operating system)
- ✓ It helps build other software (e.g., software tools)

Layered Technology in Software Engineering software Tools:

Software engineering is fully a 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.



Layered technology is divided into four parts:

1. A quality focus: It defines the continuous process improvement principles of software. It provides integrity that means providing security to the software so that data can be accessed by only an authorized person, no outsider can access the data. It also focuses on maintainability and usability.

2. Process: It is the foundation or base layer of software engineering. It is key that binds all the layers together which enables the development of software before the deadline or on time. Process defines a framework that must be established for the effective delivery of software engineering technology. The software process covers all the activities, actions, and tasks required to be carried out for software development.

Process activities are listed below:-

- **Communication:** It is the first and foremost thing for the development of software. Communication is necessary to know the actual demand of the client.
- **Planning:** It basically means drawing a map for reduced the complication of development.
- **Modeling:** In this process, a model is created according to the client for better understanding.
- **Construction:** It includes the coding and testing of the problem.
- **Deployment:-** It includes the delivery of software to the client for evaluation and feedback.

3. Method: During the process of software development the answers to all "how-to-do" questions are given by method. It has the information of all the tasks which includes communication, requirement analysis, design modeling, program construction, testing, and support.

4. Tools: Software engineering tools provide a self-operating system for processes and methods. Tools are integrated which means information created by one tool can be used by another.

1.5 Software Engineering Practice:

Software Engineering Practice Generic framework activities—communication, planning, modeling, construction, and deployment—and umbrella activities establish a skeleton architecture for software engineering work. Practices are as follows

1. Understand the problem (communication and analysis).
2. Plan a solution (modeling and software design).

3. Carry out the plan (code generation).
4. Examine the result for accuracy (testing and quality assurance).

Understand the problem: It's worth spending a little time to understand, answering a few simple questions:
• Who has a stake in the solution to the problem? That is, who are the stakeholders?
• What are the unknowns? What data, functions, and features are required to properly solve the problem?
• Can the problem be compartmentalized? Is it possible to represent smaller problems that may be easier to understand?
• Can the problem be represented graphically? Can an analysis model be created?
Plan the solution: Now you understand the problem and you can't wait to begin coding. Before you do, slow down just a bit and do a little design:
• Have you seen similar problems before? Are there patterns that are recognizable in a potential solution? Is there existing software that implements the data, functions, and features that are required? • Has a similar problem been solved? If so, are elements of the solution reusable?
• Can sub problems be defined? If so, are solutions readily apparent for the sub problems?
• Can you represent a solution in a manner that leads to effective implementation? Can a design model be created?
Examine the result: You can't be sure that your solution is perfect, but you can be sure that you've designed a sufficient number of tests to uncover as many errors as possible.
• Is it possible to test each component part of the solution? Has a reasonable testing strategy been implemented?
• Does the solution produce results that conform to the data, functions, and features that are required? Has the software been validated against all stakeholder requirements?

1.6 Software Myths

- ✓ It is beliefs about software
- ✓ The process used to built it, can be traced to the earliest days of computing
- ✓ The myth – have a number of attributes that have made them insidious [i.e. proceeding inconspicuously but harmfully]
- ✓ For instance myths appear to be reasonable statements of facts [Sometimes containing elements of truth]

1.5.1 Management Myths:

* Managers in most disciplines are often under pressure to maintain budget, keep schedules from slipping and improve quality
* A software manager often grasp at belief in a software myth, if that belief will lessen the pressure

Myth 1:

We already have a book that is 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 well exists……But is it used?
=> Are software practitioners aware of its existence?
=> Does it reflect modern software engineering practices?
=> 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 2:

If we get behind schedule, we can add more programmers and catch up [Sometimes called 'Mongolian horde Concept"]

Reality:

* Adding people to a late software project makes it later
* As new people are added, people who were working must spend time in educating the new comers, 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 3:

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

1.5.2 Customer Myths:

* The customer who requests computer software may be,
=> a person
=> a technical group
=> a marketing / sales department (Or)
=> an outside company
* In many cases customer believes myths about software
* Myths lead to **false expectations** (by the customer) and ultimately, **dissatisfaction** with the developer

Myth 1:

A general statement of objectives is sufficient to begin writing programs – We can fill in the detail later

Reality:

* An ambiguous statement {i.e. having two meanings} leads to a serious of problems
* But Unambiguous statements are developed only through effective and continuous communication between customer and developer
* So comprehensive and stable statements of requirements is not always possible

Myth 2:

Project requirements continually change, but changes can be easily accommodated because software is flexible

Reality:

* When requirements changes are requested early [before design (Or) code has been started] cost impact is relatively small
* When requirement changes are requested after design (Or) code has been started cost impact is too high
* The change can cause upheaval [i.e. Violent change (Or) disturbance] that require additional resources and major design modification

1.5.3 <u>Practitioner's Myths:</u>

Myth 1:

Once we write the program and get it to work our job is done

Reality:

* Industry data indicate that between 60 and 80 percent of all efforts expended on software will be expanded after it is delivered to the customer for the first time

Myth 2:

Until I get the program running – I have no way of assessing its quality

Reality:

* Apply any one of the effective software quality mechanism, from the beginning of the project
* Software quality reviews are more effective than testing for finding certain classes of software errors

Myth 3:

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

Reality:

* A work program is part of software configuration, that includes many elements
* But documentation only provides a foundation for successful engineering and support for software

Myth 4:
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 about creating quality
* So better quality leads to reduced rework
* Reduced rework leads to faster delivery times
Conclusion:
* Many software professionals recognizes the fallacy [**i.e. mistaken belief**] of software myths, this will indirectly promote the poor management and technical practices
* But recognition of software realities is the first step toward formulation of practical solutions for software engineering.

1.3 Software Process

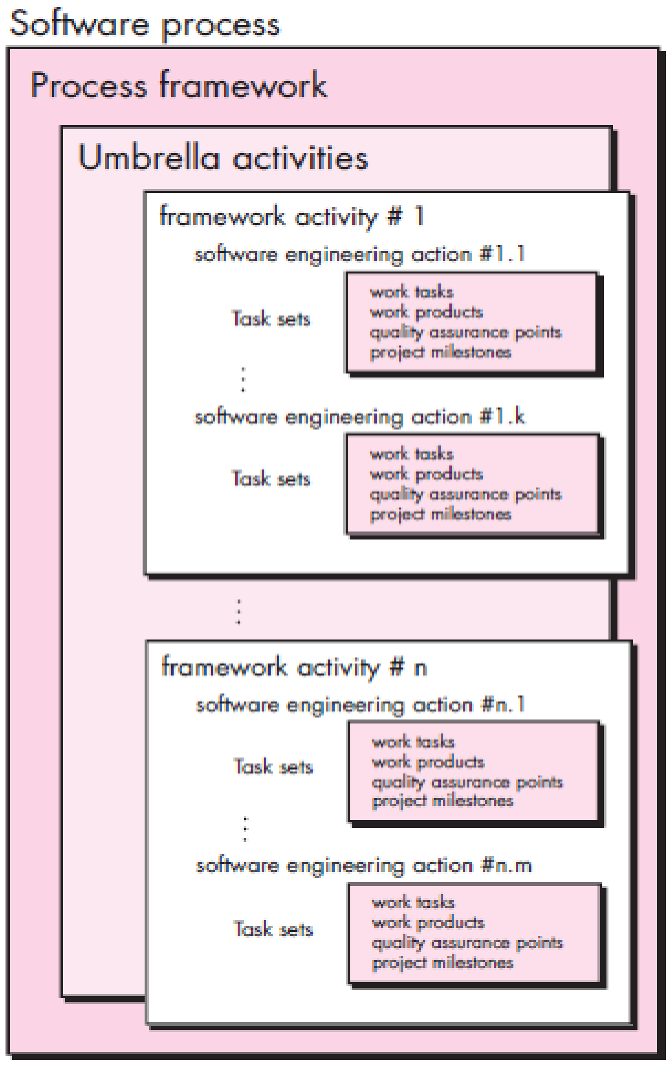
Software Process is a process defines who is doing what when and how to reach a certain goal". It is a Collection of **Activities, Actions**, and **Tasks** are performed to when the product is to be created'.

- **An activity** strives to achieve broad objective and is applied regardless of the application, domain, and 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.
- **A task** focuses on a small, but well defined objective that produces a tangible outcome.

1.3.1 Generic Process Model:

A Process Framework:
* It identifying a small number of framework activities that are applicable to all software projects, regardless of their size (Or) complexity
* The process framework include a set of umbrella activities, that are applicable across the entire software process
Framework Activity:
* It contains a set of software engineering actions [A collection of related tasks that produces a major software engineering work product]
Example:
Design is a software engineering action

* Each action contain individual work tasks
* The work tasks accomplish some part of the work implied by the action.



(1) Communication:

* It involves heavy communication and collaboration with the customer
* Also it includes requirement gathering and related activities.

(2) Planning:

* It describes the

=> Technical tasks to be conducted
=> The risks that are expected
=> Resources that will be required
=> Work products to be produced
=> Work Schedule

(3) Modeling:

* It describes the creation of models – that allow the developer and the customer to understand software requirements and design for those requirements

(4) Construction:

* It combines

=> Code generation [either manual (Or) automated]

=> Testing [Required uncovering errors in the code]

(5) Deployment:

* The software is delivered to customer
* Customer evaluates the delivered product
* Customer provides feedback based on the evaluation
* These generic framework activities can be used during the;

=> Development of small programs

=> Creation of large web applications

=> Engineering of large complex computer based systems

* The software process is different in each case, but framework activities remain same.

1.3.2 Umbrella Activities

In general, umbrella activities are applied throughout a software project and help a software team manage and control progress, quality, change, and risk. Typical umbrella activities include:

Software project tracking and control — allows the software team to assess progress against the project plan and take any necessary action to maintain the schedule.

Risk management — assesses risks that may affect the outcome of the project or the quality of the product.

Software quality assurance — defines and conducts the activities required to ensure software quality.

A technical review — assesses software engineering work products in an effort to uncover and remove errors before they are propagated to the next activity.

Measurement — defines and collects process, project, and product measures that assist the team in delivering software that meets stakeholders' needs; can be used in conjunction with all other framework and umbrella activities.

Software configuration management—manages the effects of change throughout the software process.

Reusability management — defines criteria for work product reuse (including software components) and establishes mechanisms to achieve reusable components.

Work product preparation and production — encompasses the activities required to create work products such as models, documents, logs, forms, and lists.

1.3.3 Task Set:

It defines the actual work to be done, to accomplish the objectives of a software engineering action. Requirement gathering is an important software engineering action, that occurs during the communication activity.

Example:

* For simple projects task set for requirement gathering might be look like this;

=> Make a list of stake holders for a project
=> Invite all stake holders to an informal meeting
=> Ask each stake holders to make a list of features and functions required
=> Discuss requirements and build a final list
=> Prioritize requirements
=> Note areas of Uncertainty

Task Set for Software Project

A task set is a collection of software engineering work tasks, milestones, and deliverables that must be accomplished to complete a particular project.

IDENTIFYING A TASK SET

Task set is the actual work to be done to achieve an objective of engineering action. For small project, consider **elicitation action in communication activity**, this may include :

- Prepare a list of stakeholders of the project.
- Organize a meeting for stakeholders.
- Discuss requirements.
- Finalize requirements list.
- Make a list of issues raised.

For large projects extraneous steps may be added to elicitation such as interviewing each stakeholder separately before the group meeting, more techniques are applied in discussing requirements…etc.

1.4 PROCESS MODELS

Process Models – Definition

* It is a distinct 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 provide a useful roadmap for software engineering work

1.4.1 THE WATER FALL MODEL

* It is sometimes called the classic life cycle

* It suggests a systematic, sequential approach to software development that begins with customer specification of requirements and progress through

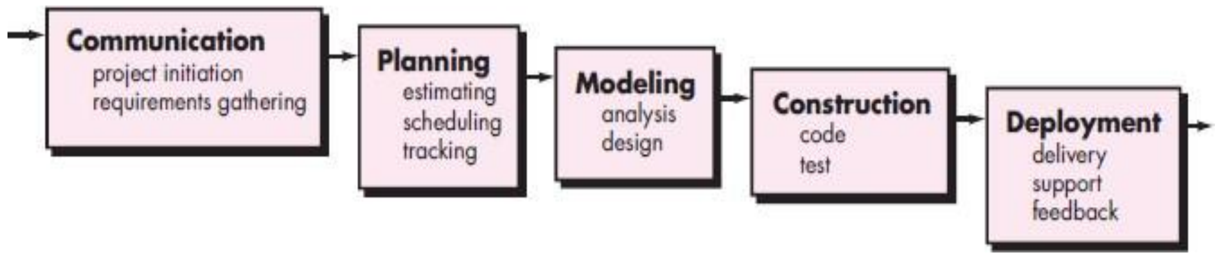
=> Planning
=> Modeling
=> Construction and
=> Deployment

Problems encountered in waterfall model:

(1) Real projects rarely follow the sequential flow. As a result changes cause confusion as the project team proceeds

(2) It is difficult for the customer to state all requirements explicitly

(3) The customer must have patience

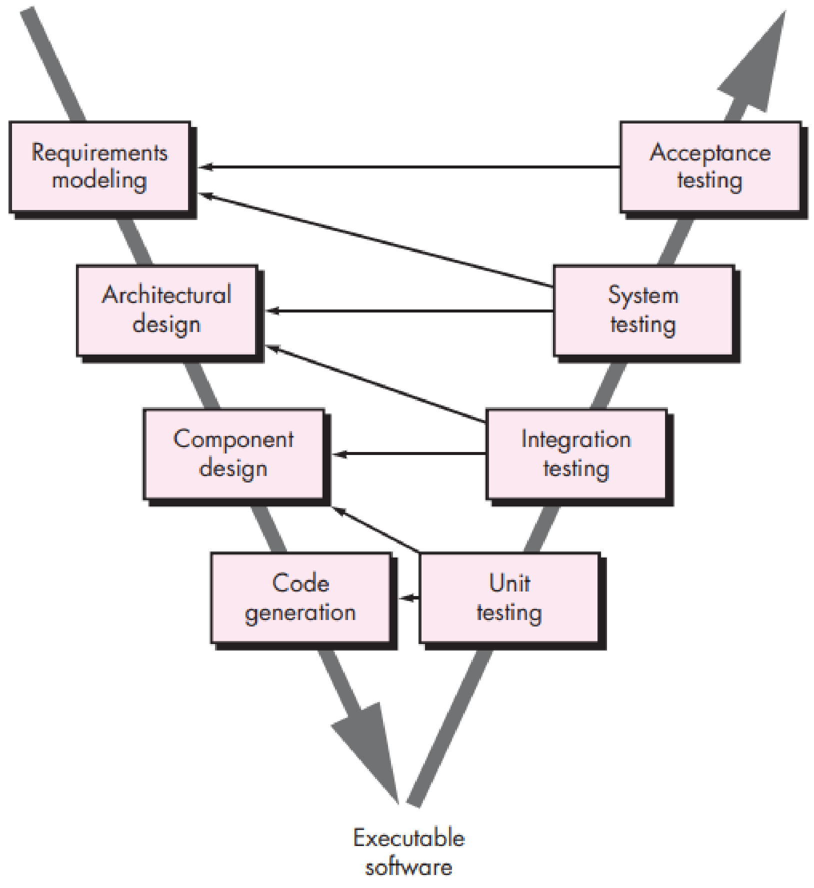


* The linear nature of the water fall model leads to " Blocking State" in which some project team members must wait for other members of the team to complete dependent task

* The water fall model can serve as a useful process model in situations where

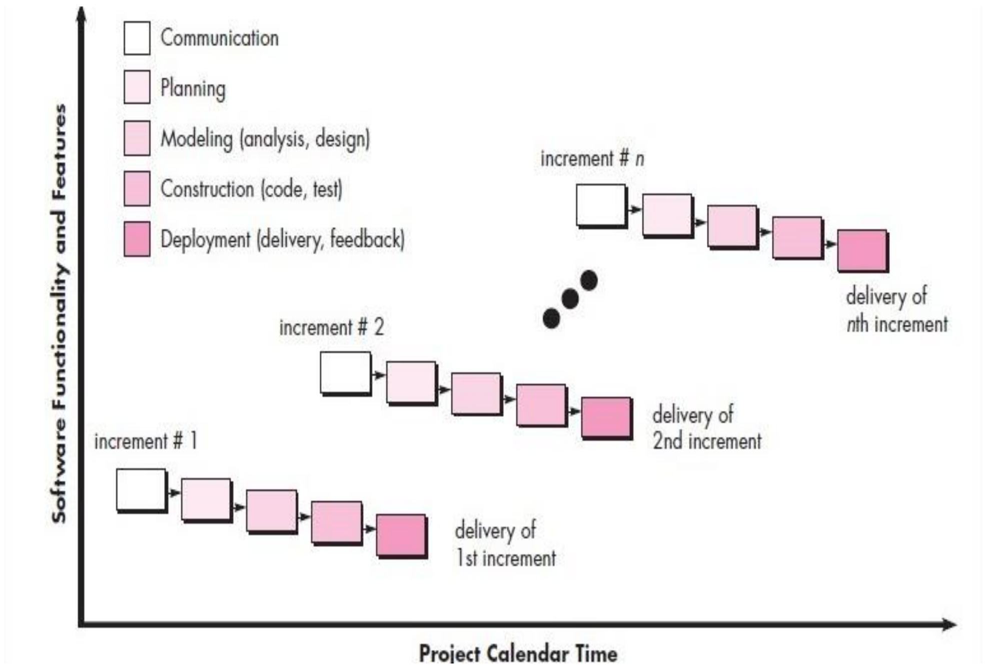
=> Requirements are fixed and work is to proceed to completion in a linear manner.

A variation in the representation of the waterfall model is called the V-model. Represented in Figure 2.4, the V-model depicts the relationship of quality assurance actions to the actions associated with communication, modeling, and early construction activities. As software team moves down the left side of the V, basic problem requirements are refined into progressively more detailed and technical representations of the problem and its solution. Once code has been generated, the team moves up the right side of the V, essentially performing a series of tests (quality assurance actions) that validate each of the models created as the team moved down the left side. In reality, there is no fundamental difference between the classic life cycle and the Vmodel. The V-model provides a way of visualizing how verification and validation actions are applied to earlier engineering work.



1.4.2 INCREMENTAL PROCESS MODELS

* The incremental model combines elements of the water fall model applied in an iterative fashion
* This 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

▧ => Communication
▥ => Planning
≋ => Modeling {Analysis, Design}
▦ => Construction {Code, Test}
▩ => Deployment {Delivery, Feedback}

Main Idea:
* When an incremental model is used the first increment is called “ CORE PRODUCT” i.e. the basic requirements are addressed but main supplementary features remain undelivered
* The core product is used by customer (Or) undergoes detailed evaluation.
* As a result of evaluation a plan is developed for next increment.

≋ => Modeling {Analysis, Design}

▦ => Construction {Code, Test}

▩ => Deployment {Delivery, Feedback}

* The plan addresses the modification of the core product to better meet the needs of the customer and the delivery of the additional features and functionality
* This process is repeated for each increment delivery, until the complete product is developed
* Unlike prototyping model, the incremental model focuses on the delivery of an operational product with each increment
* This model 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 core product is well received additional staff can be added to implement the next increment
* Increments can be planned to manage technical risks
* For example a major availability of new hardware is under development, whose delivery date is uncertain
* So plan early increments in a way that avoids the use of this hardware, thereby enabling partial functionality to be delivered to end-users without inordinate delay

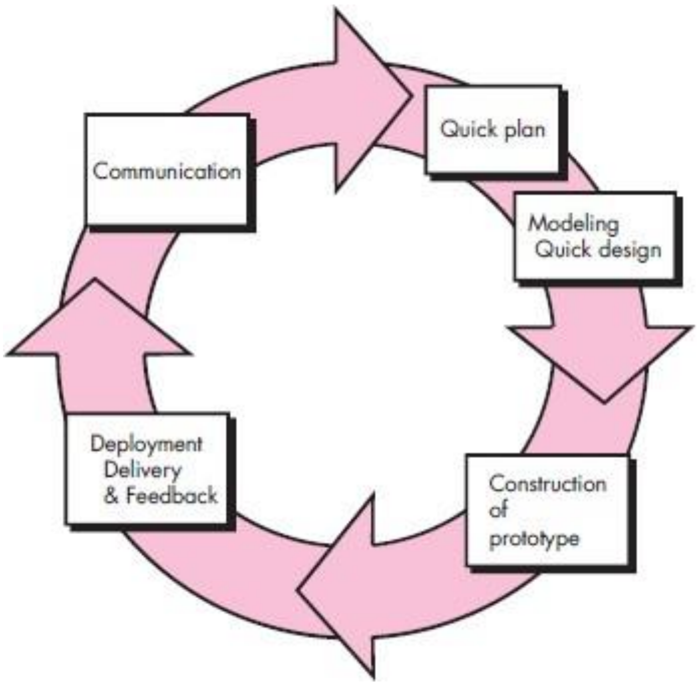
1.4.3 EVOLUTIONARY PROCESS MODELS

* Like all computer systems software also evolve over a period of time
* Making a straight line path to an end-product is unrealistic, if business and product requirements often change as development proceeds
* Suppose if a set of core product (Or) system requirement is well understood, but the details of product (Or) system extensions have not yet to be defined.
* These case, a software engineer needs a process model that has been designed to accommodate a product that evolves over time.
* The evolutionary process modes are more suitable for the above case.

<u>Types:</u>

(1) Prototyping Model
(2) Spiral Model

1.4.3.1 PROTOTYPE MODEL



* The prototype model may offer the best approach, if

=> Customer defines a set of objectives for software, but does not identify detailed in put, processing (Or) output requirements

=> In other cases, the developer may be unsure of the efficiency of an algorithm (Or) the form that human-machine interaction should take

* The prototyping model assists the software engineer and the customer to better understand

=> What is to be built, when requirements are fuzzy?

<u>Communication:</u>

* The prototype model begins with communication

* The software engineer and customer meet and define the overall objectives for the software

=> Identify whatever requirements are known

=> Outline areas where further definition is mandatory

<u>Quick design:</u>

* The quick design focuses on a representation of those aspects of the software that will be visible to the customer / end user

<u>Example:</u>

* Human interface Layout (Or0 Output display formats.

<u>Construction of prototype:</u>

* Ideally the prototype serves as a mechanism for identifying software requirements

* If the working prototype is built, the developer uses the existing programs fragments that ensure working programs to be generated quickly

<u>Deployment</u>:

* The prototype is deployed and evaluated by the customer

* Feedback is used to refine requirements for the software

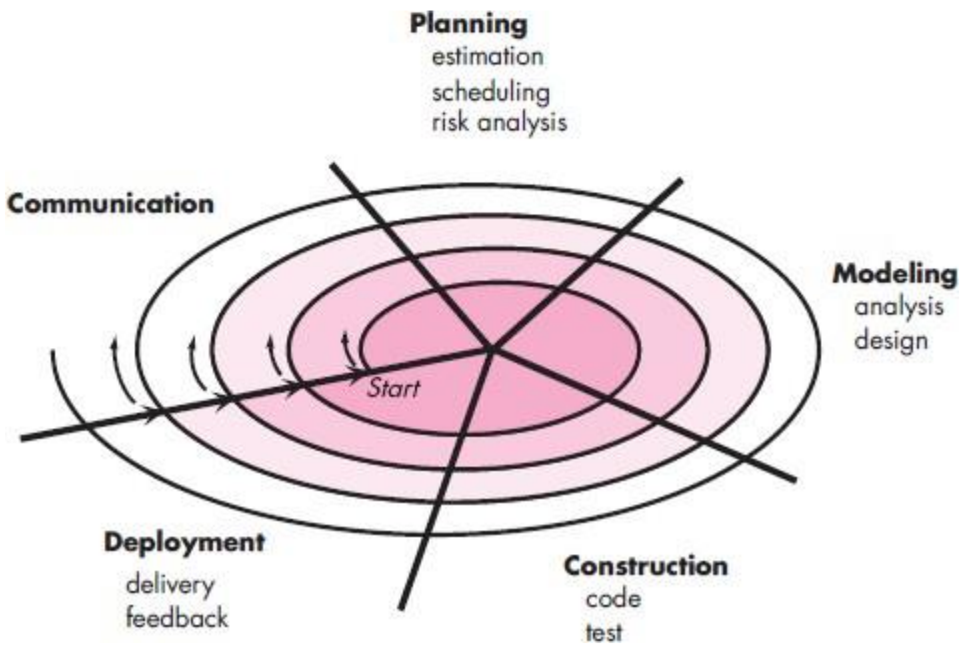
<u>Drawbacks:</u>

(i) The customer sees what appears to be a working version of the software, unaware that the prototype is held together

(ii) The developer makes the implementation compromises in order to get a prototype working quickly

(iii) An inefficient algorithm may be implemented simply to demonstrate capability

1.4.3.2 SPIRAL MODEL:



Definition:

* It is evolutionary software process model that combines the iterative nature of prototyping with the controlled and systematic aspect of the water fall model

Two main features

(i) A **cyclic approach** for incrementally growing a system's degree of definition and implementation while decreasing its degree of risk

(ii) An **anchor point** mile stones for ensuring stake holders commitment to feasible and mutually satisfactory system solutions

Main idea:

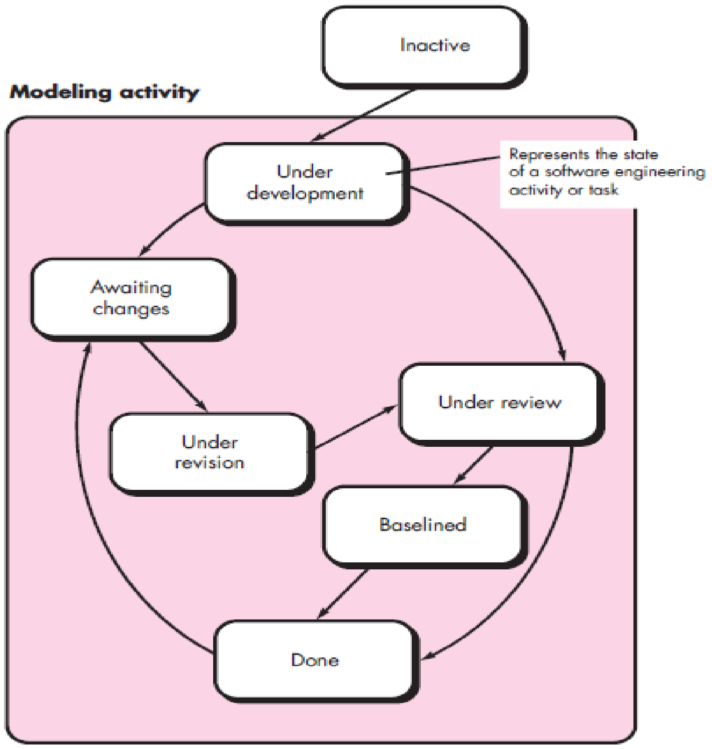
* A spiral model is divided into a set of frame work activities, defined by the software engineering team
* Each frame work activities represent one segment of spiral path
* As evolutionary process begins, the software team performs activities that are implied by a circuit around the spiral in a clockwise direction beginning at the centre
* The first circuit around the spiral might result in development of a product specification
* Subsequent passes might be used to develop a prototype and 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
* Using spiral model software is developed in a series of evolutionary release
* During each iteration the release might me a paper model (Or) prototype
* During later iterations, increasingly more complete versions of the engineered system are produced
* Unlike other process model that end when software is delivered, the spiral model can be adapted to apply throughout the life of the software
* In the above fig first circuit represents concept development project
* Once concept is developed into actual product, process proceeds outwards on spiral, A new product development project commences
* Later circuit around the spiral might be used to represent "product enhancement project"
* Whenever a change is initiated the process starts at the appropriate entry point

Advantages:

(i) It is a realistic approach to the development of large-scale system and software
(ii) It enables the developer to apply the prototyping approach at any stage in th evolution of the product
(iii) It maintains the systematic stepwise approach suggested by classic life cycle and incorporates it into iterative framework

1.4.4 CONCURRENT DEVELOPMENT MODEL

* It is also called as "**Concurrent Engineering**"
* It can be represented schematically as a series of framework activities, software engineering actions and tasks and their associated states
* At any given time the modeling activity may be in any one of the states



* All activities exist concurrently, but reside in different states

<u>Example:</u>

* Initially in a project, the communication activity has completed its first iteration and exists in awaiting change state

* When initial communication was completed, the modeling activity exists in none state, makes a transition from none state into the under development state

* If customer indicates changes in requirements, the modeling activity moves from under development into awaiting change state

* A series of events will trigger transition from state to state for each of the software engineering activities, actions (or) tasks

* During early stages of design [a software engineering action that occurs during the modeling activity] an inconsistency in the analysis model will trigger the analysis action from the done state into the awaiting change state.

Advantages:

(1) The concurrent process model provides an accurate picture of the current state of a project

(2) It defines the software engineering activities, actions and tasks as a network, instead of sequence of events

(3) Each activity, action (or) tasks on network exists simultaneously with other activities.

1.5 Product and Process:

Process: Process is how we go from the beginning to the end of a project. All projects use a process. Many project managers, however, do not choose a process based on the people and product at hand. They simply use the same process they've always used or misused. Let's focus on two points regarding process:

(1) Process improvement and
(2) Using the right process for the people and product at hand.

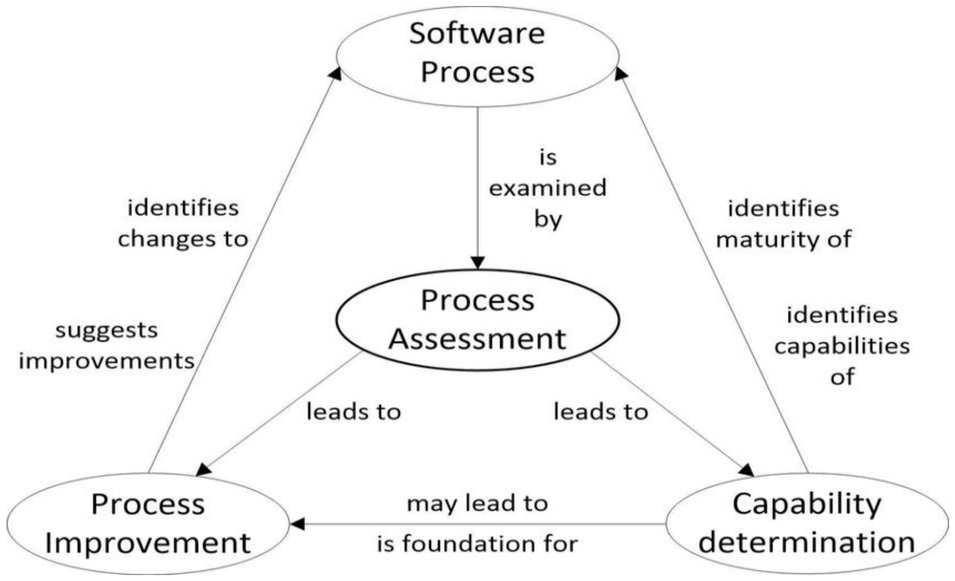
Product: The product is the result of a project. The desired product satisfies the customers and keeps them coming back for more. Sometimes, however, the actual product is something less. The product pays the bills and ultimately allows people to work together in a process and build software. Always keep the product in focus. Our current emphasis on process sometimes causes us to forget the product. This results in a poor product, no money, no more business, and no more need for people and process.

1.6 Process Assessment:

* The software process gives no guarantee that;
=> Software will be delivered on time
=> it will meet the customer needs

* In addition the process should be accessed to ensure that it meets a set of basic process criteria that have been essential for a software engineering

* The relationship between the software process and methods applied for assessment and improvement are shown below



<u>Software process assessment</u> – <u>Different approaches</u>

(1) Standard CMMI Assessment Methods for Process Improvement [SCAMPI]

* It provides a five step process assessment model. They are
(i) Initiating
(ii) Diagnosing
(iii) Establishing
(iv) Acting

(v) Learning

(2) CMM – Based Appraisal for Internal Process Improvement [CBAIPI]

* It provides a diagnosing techniques for assessing the relatively maturity of a software organization

(3) SPICE [ISO / IEC 15504]

* It defines a set of requirements for software process assessment
* This standard helps the organization in developing an objective evaluation of any defined software process

(4) ISO 9001: 2000 for Software

* This standard is applied to any organization that wants to improve;

=> the over all quality of the products, systems, services it provides

* This standard is directly applicable to software organizations and companies
* This standard uses a **"Plan – do – Check – act"** cycle that is applied to quality management elements of software projects

Plan => It establishes the process objectives, activities and tasks necessary to achieve high quality software and resultant customer satisfaction

Do => It implements the software products [including both framework and umbrella activities]

Check => It monitors are measures the process to ensure that all requirements established for quality management have been achieved

Act => It initiates the software process improvement activities that continually work to improve the process

UNIT-2

AGILE DEVELOPMENT:

What are Agility, Agility and the Cost of Change, Agile Process, Scrum, **Other Agile Process Models:** Adaptive Software Development (ASD), Dynamic System Development Method (DSDM), Crystal.

2.1 What is "Agility"?

- Effective (rapid and adaptive) response to change
- Effective communication among all stakeholders
- Drawing the customer onto the team
- Organizing a team so that it is in control of the work performed

Agility in context of software engineering

- Agility means effective (rapid and adaptive) response to change, effective communication among all stockholder.
- Drawing the customer onto team and organizing a team so that it is in control of work performed. -The Agile process, light-weight methods are People-based rather than plan-based methods.
- The agile process forces the development team to focus on software itself rather than design and documentation.
- The agile process believes in iterative method.
- The aim of agile process is to deliver the working model of software quickly to the customer For example: Extreme programming is the best known of agile process.

Agile Model

Agile SDLC model is a combination of iterative and incremental process models with focus on process adaptability and customer satisfaction by rapid delivery of working software product. Agile Methods break the product into small incremental builds. These builds are provided in iterations. Each iteration typically lasts from about one to three weeks. Every iteration involves cross functional teams working simultaneously on various areas like –

- Planning
- Requirements Analysis
- Design
- Coding
- Unit Testing and
- Acceptance Testing.

At the end of the iteration, a working product is displayed to the customer and important stakeholders.

What is Agile?

Agile model believes that every project needs to be handled differently and the existing methods need to be personalized to best suit the project requirements. In Agile, the tasks are divided to time boxes (small time frames) to deliver specific features for a release.

Iterative approach is taken and working software build is delivered after each iteration. Each build is incremental in terms of features; the final build holds all the features required by the customer.

The Agile thought process had started early in the software development and started becoming popular with time due to its flexibility and adaptability.

The most popular agile methods include Rational Unified Process (1994), Scrum (1995), Crystal Clear, Extreme Programming (1996), Adaptive Software Development, Feature Driven Development, and Dynamic Systems Development Method (DSDM) (1995). These are now collectively referred to as **Agile Methodologies**, after the Agile Manifesto was published in 2001.

Following are the Agile Manifesto principles −

- **Individuals and interactions** − In Agile development, self-organization and motivation are important, as are interactions like co-location and pair programming.
- **Working software** − Demo working software is considered the best means of communication with the customers to understand their requirements, instead of just depending on documentation.
- **Customer collaboration** − As the requirements cannot be gathered completely in the beginning of the project due to various factors, continuous customer interaction is very important to get proper product requirements.
- **Responding to change** − Agile Development is focused on quick responses to change and continuous development.

Agile Vs Traditional SDLC Models

Agile is based on the **adaptive software development methods**, whereas the traditional SDLC models like the waterfall model is based on a predictive approach. Predictive teams in the traditional SDLC models usually work with detailed planning and have a complete forecast of the exact tasks and features to be delivered in the next few months or during the product life cycle.

Predictive methods entirely depend on the **requirement analysis and planning** done in the beginning of cycle. Any changes to be incorporated go through a strict change control management and prioritization. Agile uses an **adaptive approach** where there is no detailed planning and there is clarity on future tasks only in respect of what features need to be developed. There is feature driven development and the team adapts to the changing product requirements dynamically. The product is tested very frequently, through the release iterations, minimizing the risk of any major failures in future.

Customer Interaction is the backbone of this Agile methodology, and open communication with minimum documentation are the typical features of Agile development environment. The agile teams work in close collaboration with each other and are most often located in the same geographical location.

Agile Model - Pros and Cons

Agile methods are being widely accepted in the software world recently. However, this method may not always be suitable for all products. Here are some pros and cons of the agile model.

Advantages of the Agile Model:

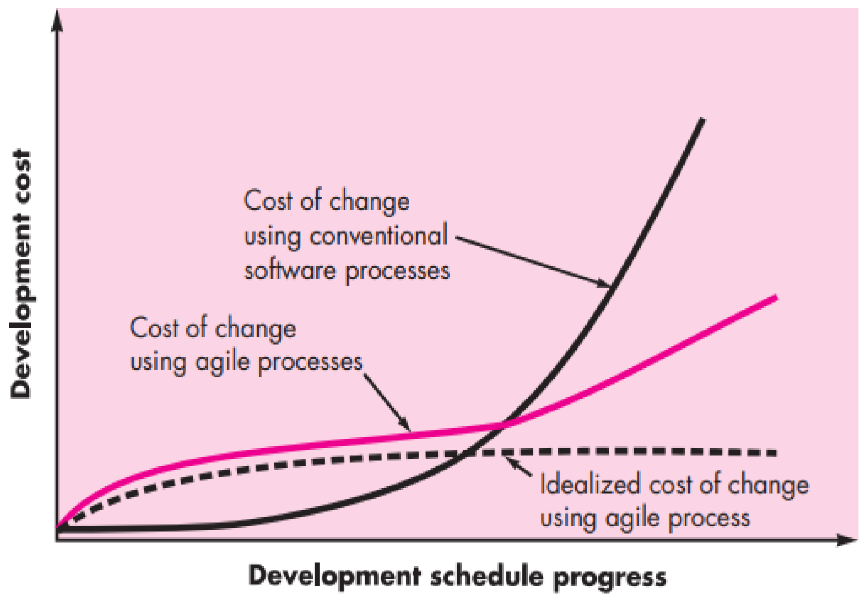
- Is a very realistic approach to software development?
- Promotes teamwork and cross training.
- Functionality can be developed rapidly and demonstrated.
- Resource requirements are minimum.
- Suitable for fixed or changing requirements
- Delivers early partial working solutions.
- Good model for environments that change steadily.
- Minimal rules, documentation easily employed.
- Enables concurrent development and delivery within an overall planned context.
- Little or no planning required.
- Easy to manage.
- Gives flexibility to developers.

Disadvantages of the Agile Model:

- Not suitable for handling complex dependencies.
- More risk of sustainability, maintainability and extensibility.
- An overall plan, an agile leader and agile PM practice is a must without which it will not work.
- Strict delivery management dictates the scope, functionality to be delivered, and adjustments to meet the deadlines.
- Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction.

- There is a very high individual dependency, since there is minimum documentation generated.
- Transfer of technology to new team members may be quite challenging due to lack of documentation.

2.2 Agility and the Cost of Change



An agile process reduces the cost of change because software is released in increments and change can be better controlled within an increment.

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. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference for 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, then trust them to get the job done.
6. The most efficient and effective method of conveying information to and within the 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.

2.4 Agile Process Models

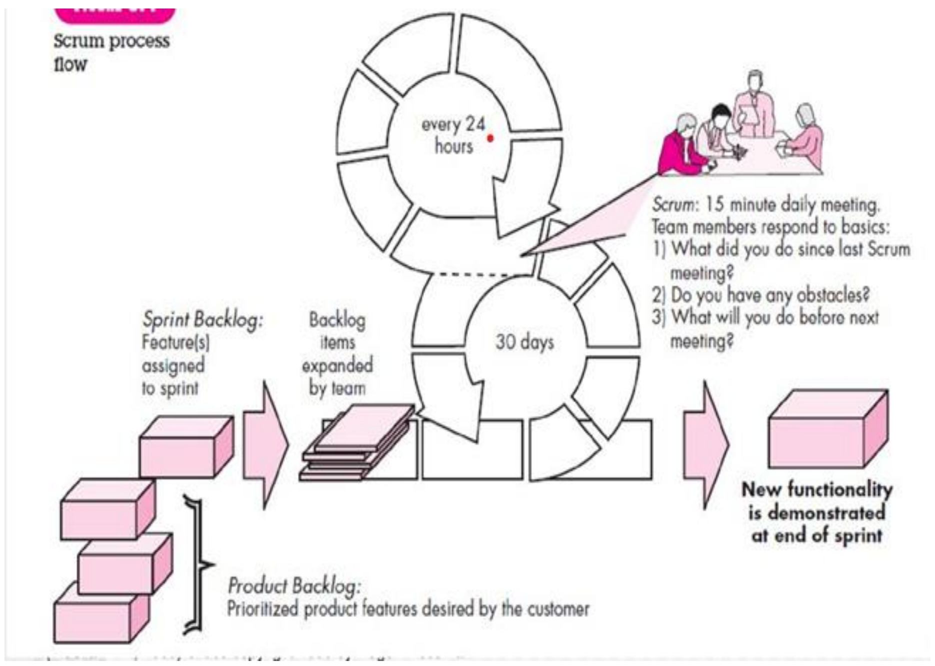
- **Scrum**
- **Adaptive Software Development (ASD)**
- **Dynamic Systems Development Method (DSDM)**
- **Crystal**
- **Extreme Programming (XP)**
- **kanban**
- **Feature Driven Development (FDD)**
- **Agile Modeling (AM)**

2.4.1 SCRUM

What is 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. According to latest surveys Scrum is the most popular agile project management methodology in software development. The term Scrum is formed from Rugby.

Scrum is ideally used where highly emergent or rapidly changing requirements. Scrum is basically worked on a self-organizing, cross-functional team. In the overall scrum team there is no team leader who assign the task to team rather whole scrum members work as a team & they decides the task on which they will work on. Also the problem will be resolve by team.


Scrum process flow
every 24 hours
30 days
Scrum: 15 minute daily meeting. Team members respond to basics:
1) What did you do since last Scrum meeting?
2) Do you have any obstacles?
3) What will you do before next meeting?
Sprint Backlog: Feature(s) assigned to sprint
Backlog items expanded by team
New functionality is demonstrated at end of sprint
Product Backlog: Prioritized product features desired by the customer

Process flow of Scrum Methodologies:

Process flow of scrum testing is as follows:

- Each iteration of a scrum is known as Sprint
- Product backlog is a list where all details are entered to get end product
- During each Sprint, top items of Product backlog are selected and turned into Sprint backlog
- Team works on the defined sprint backlog
- Team checks for the daily work
- At the end of the sprint, team delivers product functionality

It consists of three roles, and their responsibilities are explained as follows:

- **Scrum Master**
  - Master is responsible for setting up the team, sprint meeting and removes obstacles to progress
- **Product owner**
  - The Product Owner creates product backlog, prioritizes the backlog and is responsible for the delivery of the functionality at each iteration
- **Scrum Team**
  - Team manages its own work and organizes the work to complete the sprint or cycle.

Advantages of Scrum Development:

- In this methodology, decision-making is entirely in the hands of the teams

- This methodology enables project's where the business requirements documentation is not considered very significant for the successful development
- It is a lightly controlled method which totally empathizes on frequent updating of the progress, therefore, project development steps is visible in this method
- A daily meeting easily helps the developer to make it possible to measure individual productivity. This leads to the improvement in the productivity of each of the team members.

Disadvantages of Scrum Development:

- This kind of development model is suffered if the estimating project costs and time will not be accurate
- It is good for small, fast moving projects but not suitable for large size projects
- This methodology needs experienced team members only. If the team consists of people who are novices, the project cannot be completed within exact time frame.

2.4.2 Adaptive Software Development (ASD)

Adaptive Software Development is a method to build complex software and system. ASD focuses on human collaboration and self-organization. ASD **"life cycle"** incorporates three phases namely:

1. Speculation
2. Collaboration
3. Learning

These are explained as following below.

1. **Speculation:**
   During this phase project is initiated and planning is conducted. The project plan uses project initiation information like project requirements, user needs, customer mission statement, etc, to define set of release cycles that the project wants.
2. **Collaboration:**
   It is the difficult part of ASD as it needs the workers to be motivated. It collaborates communication and teamwork but emphasizes individualism as individual creativity plays a major role in creative thinking. People working together must trust each others to
   - Criticize without animosity,
   - Assist without resentment,
   - Work as hard as possible,
   - Possession of skill set,
   - Communicate problems to find effective solution.
3. **Learning:**
   The workers may have a overestimate of their own understanding of the technology which may not lead to the desired result. Learning helps the workers to increase their level of understanding over the project.

Learning process is of 3 ways:

1. Focus groups
2. Technical reviews
3. Project postmortem

ASD's overall emphasis on the dynamics of self-organizing teams, interpersonal collaboration, and individual and team learning yield software project teams that have a much higher likelihood of success.

2.4.3 Dynamic Systems Development Method (DSDM)

DSDM is an agile software development methodology. It is an iterative, incremental approach that is largely based on the Rapid Application Development (RAD) methodology. The method provides a four-phase framework consisting of:

- Feasibility and business study
- Functional model / prototype iteration
- Design and build iteration
- Implementation

Within each phase, DSDM relies on several different activities and techniques based on these principles:

- Projects evolve best through direct and co-located collaboration between the developers and the users.
- Self-managed and empowered teams must have the authority to make time sensitive and critical project-level decisions.
- Design and development is incremental and evolutionary in nature and is largely driven by regular, iterative user feedback.
- Working software deliverables are defined as systems that address the critical, current business needs versus systems that address less critical future needs.
- Frequent and incremental delivery of working software is valued over infrequent delivery of perfectly working software.
- All changes introduced during development must be reversible.
- Continuous integration and quality assurance testing is conducted in-line, throughout the project lifecycle.
- Visibility and transparency is encouraged through regular communication and collaboration amongst all project stakeholders.

Why use DSDM?

- Results of development are directly and promptly visible
- Since the users are actively involved in the development of the system, they are more likely to embrace it and take it on.
- Basic functionality is delivered quickly, with more functionality being delivered at regular intervals.

- Eliminates bureaucracy and breaks down the communication barrier between interested parties.
- Because of constant feedback from the users, the system being developed is more likely to meet the need it was commissioned for.
- Early indicators of whether project will work or not, rather than a nasty surprise halfway through the development
- System is delivered on time and on budget.
- Ability of the users to affect the project's direction.

2.4.4 Crystal Methods

Crystal Methods are a family of software development 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. It was developed by Alistair Cockburn and Jim High smith from his study and interviews of teams.

The methods are color-coded to signify the risk to human life. For example, projects that may involve risk to human life will use Crystal Sapphire while projects that do not have such risks will use Crystal Clear. Crystal focuses on six primary aspects: people, interaction, community, communication, skills, and talents. Process is considered secondary. There are also seven common properties in Crystal that indicate higher possibility of success and they include frequent delivery, reflective improvement, osmotic communication, and easy access to expert users. The methods are very flexible and avoid rigid processes because of its human-powered or people-centric focus.