Chapter 5

5.1 Project Scheduling –

Software project scheduling is an action that distributes estimated effort across the planned project duration by allocating the effort to specific software engineering tasks. It is important to note, however, that the schedule evolves over time. During early stages of project planning, a macroscopic schedule is developed. This type of schedule identifies all major process framework activities and the product functions to which they are applied. As the project gets under way, each entry on the macroscopic schedule is refined into a detailed schedule. Here, specific software actions and tasks (required to accomplish an activity) are identified and scheduled.

1. Basic principles of project scheduling –

Like all other areas of software engineering, a number of basic principles guide software project scheduling:

Compartmentalization. The project must be compartmentalized into a number of manageable activities and tasks. To accomplish compartmentalization, both the product and the process are refined.

Interdependency. The interdependency of each compartmentalized activity or task must be determined. Some tasks must occur in sequence, while others can occur in parallel. Some activities cannot commence until the work product produced by another is available. Other activities can occur independently.

Time allocation. Each task to be scheduled must be allocated some number of work units (e.g., person-days of effort). In addition, each task must be assigned a start date and a completion date that are a function of the interdependencies and whether work will be conducted on a full-time or part-time basis.

Effort validation. Every project has a defined number of people on the software team. As time allocation occurs, you must ensure that no more than the allocated number of people has been scheduled at any given time. For example, consider a project that has three assigned software engineers (e.g., three person-days are available per day of assigned effort4). On a given day, seven concurrent tasks must be accomplished. Each task requires 0.50 person-days of effort. More effort has been allocated than there are people to do the work.

Defined responsibilities. Every task that is scheduled should be assigned to a specific team member.

Defined outcomes. Every task that is scheduled should have a defined outcome. For software projects, the outcome is normally a work product (e.g., the design of a component) or a part of a work product. Work products are often combined in deliverables.

Defined milestones. Every task or group of tasks should be associated with a project milestone. A milestone is accomplished when one or more work products has been reviewed for quality (Chapter 15) and has been approved.

Each of these principles is applied as the project schedule evolves.

2. Work Breakdown Structure –

Dividing complex projects to simpler and manageable tasks is the process identified as Work Breakdown Structure (WBS).

Usually, the project managers use this method for simplifying the project execution. In WBS, much larger tasks are broken down to manageable chunks of work. These chunks can be easily supervised and estimated.

WBS is not restricted to a specific field when it comes to application. This methodology can be used for any type of project management.

Following are a few reasons for creating a WBS in a project:

- Accurate and readable project organization.
- Accurate assignment of responsibilities to the project team.
- Indicates the project milestones and control points.
- Helps to estimate the cost, time and risk.
- Illustrate the project scope, so the stakeholders can have a better understanding of the same.

Construction of a WBS

Identifying the main deliverables of a project is the starting point for deriving a work breakdown structure.

This important step is usually done by the project managers and the subject matter experts (SMEs) involved in the project. Once this step is completed, the subject matter experts start breaking down the high-level tasks into smaller chunks of work.

In the process of breaking down the tasks, one can break them down into different levels of detail. One can detail a high-level task into ten sub-tasks while another can detail the same high-level task into 20 sub-tasks.

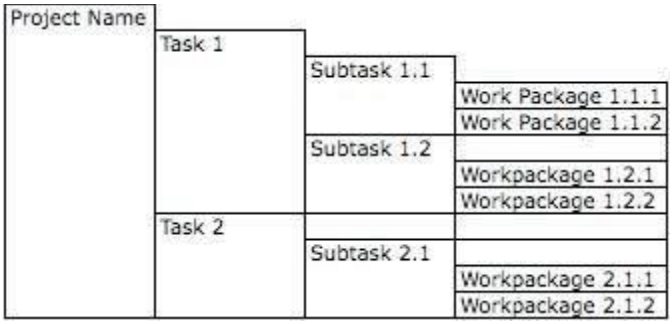
Therefore, there is no hard and fast rule on how you should breakdown a task in WBS. Rather, the level of breakdown is a matter of the project type and the management style followed for the project.

In general, there are a few "rules" used for determining the smallest task chunk. In "two weeks" rule, nothing is broken down smaller than two weeks worth of work.

This means, the smallest task of the WBS is at least two-week long. 8/80 is another rule used when creating a WBS. This rule implies that no task should be smaller than 8 hours of work and should not be larger than 80 hours of work.

One can use many forms to display their WBS. Some use tree structure to illustrate the WBS, while others use lists and tables. Outlining is one of the easiest ways of representing a WBS.

Following example is an outlined WBS:



There are many design goals for WBS. Some important goals are as follows:

- Giving visibility to important work efforts.
- Giving visibility to risky work efforts.
- Illustrate the correlation between the activities and deliverables.
- Show clear ownership by task leaders.

WBS Diagram

In a WBS diagram, the project scope is graphically expressed. Usually the diagram starts with a graphic object or a box at the top, which represents the entire project. Then, there are sub-components under the box.

These boxes represent the deliverables of the project. Under each deliverable, there are sub-elements listed. These sub-elements are the activities that should be performed in order to achieve the deliverables.

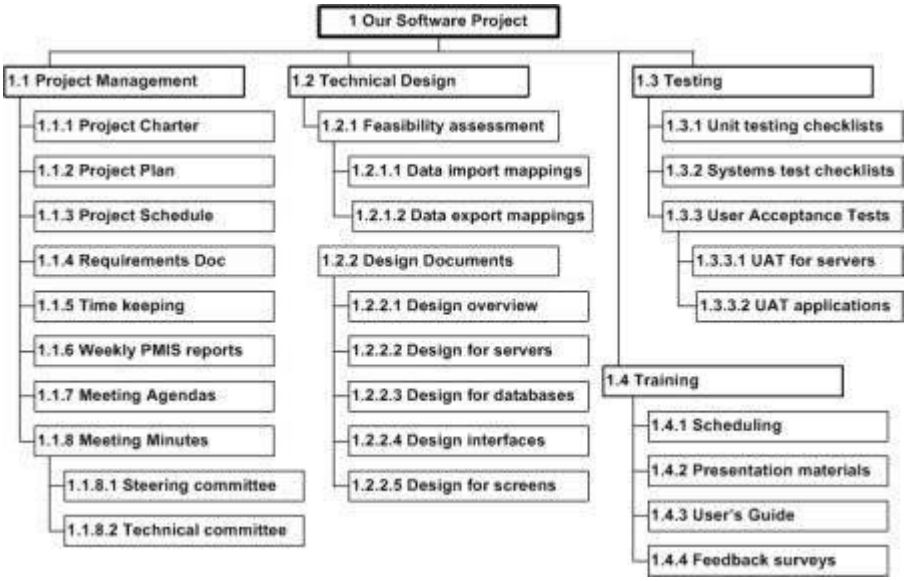
Although most of the WBS diagrams are designed based on the deliveries, some WBS are created based on the project phases. Usually, information technology projects are perfectly fit into WBS model.

Therefore, almost all information technology projects make use of WBS.

In addition to the general use of WBS, there is specific objective for deriving a WBS as well. WBS is the input for Gantt charts, a tool that is used for project management purpose.

Gantt chart is used for tracking the progression of the tasks derived by WBS.

Following is a sample WBS diagram:



The efficiency of a work breakdown structure can determine the success of a project.

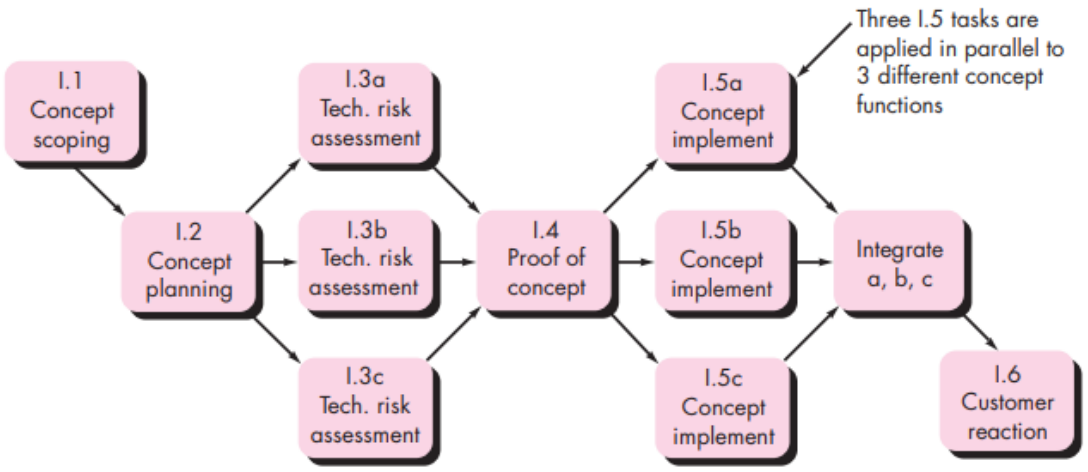
The WBS provides the foundation for all project management work, including, planning, cost and effort estimation, resource allocation, and scheduling.

Therefore, one should take creating WBS as a critical step in the process of project management.

3. Activity Network – Defining a task network

Individual tasks and subtasks have interdependencies based on their sequence. In addition, when more than one person is involved in a software engineering project, it is likely that development activities and tasks will be performed in parallel. When this occurs, concurrent tasks must be coordinated so that they will be complete when later tasks require their work product(s). A task network, also called an activity network, is a graphic representation of the task flow for a project. It is sometimes used as the mechanism through which task sequence and dependencies are input to an automated project scheduling tool. In its simplest form (used when creating a macroscopic schedule), the task network depicts major software engineering actions. Figure below shows a schematic task network for a concept development project. The concurrent nature of software engineering actions leads to a number of important scheduling requirements. Because parallel tasks occur asynchronously, you should determine intertask dependencies to ensure continuous progress toward completion. In addition, you should be aware of those tasks that lie on the critical path. That is, tasks that must be completed on schedule if the project as a whole is to be completed on schedule. These issues are discussed in more detail later in this chapter. It is important to note that the task network shown in Figure below is macroscopic.

A task network for concept development



4. Scheduling – CPM and PERT

Scheduling of a software project does not differ greatly from scheduling of any multitask engineering effort. Therefore, generalized project scheduling tools and techniques can be applied with little modification for software projects.

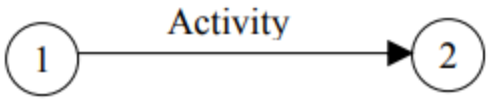
Program evaluation and review technique (PERT) and the critical path method (CPM) are two project scheduling methods that can be applied to software development. Both techniques are driven by information already developed in earlier project planning activities: estimates of effort, a decomposition of the product function, the selection of the appropriate process model and task set, and decomposition of the tasks that are selected. Interdependencies among tasks may be defined using a task network. Tasks, sometimes called the project work breakdown structure (WBS), are defined for the product as a whole or for individual functions.

Both PERT and CPM provide quantitative tools that allow you to (1) determine the critical path—the chain of tasks that determines the duration of the project, (2) establish "most likely" time estimates for individual tasks by applying statistical models, and (3) calculate "boundary times" that define a time "window" for a particular task.

- Definition
- A project is defined by a set of activities.
- Each activity is defined by its duration (time to complete the activity) and its predecessors (activities that must be completed before the activity can start).
- CPM (Critical Path Method) is used to assist the project manager in scheduling the activities (i.e., when should each activity start). It assumes that activity durations are known with certainty.
- PERT (Program Evaluation and Review Technique) is used to assist in project scheduling similar to CPM. However, PERT assumes that activity durations are random variables (i.e., probabilistic).

• Project Network

- The first step in CPM/PERT is to construct a project network.
- In the project network each activity is represented by an arc connected by two nodes. The first node represents the start of the activity and the second node represents the end of it.



- The network should reflect activities precedence relations.

- Given a list of activities and predecessors, the following rules should be followed to construct a project network:

  (1) Node 1 represents the start of the project. An arc should lead from it to represent activities with no predecessors.

(2) A unique finish node representing the completion of the project should be included in the network.

(3) Number the nodes in such a way that the node representing completion of an activity always has a larger number than the node representing beginning of the activity.

(4) An activity should not be represented by more than one arc.

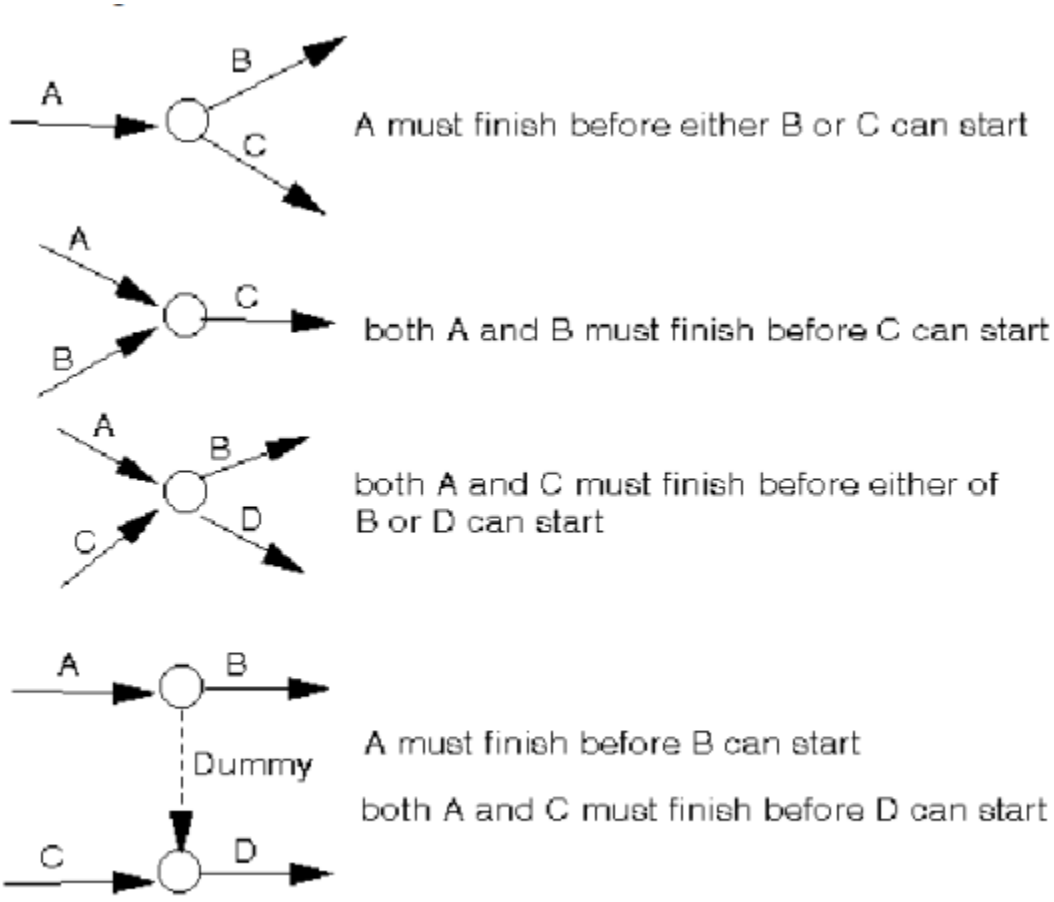
(5) Two nodes could be connected by at most one arc.

(6) Each node should have at least one entering arc and at least one leaving arc.

(7) Use the least possible number of nodes (optional).

- To avoid violation of rules (4)-(6) a dummy activity with zero duration (represented by a doted arc) may be introduced.

- Examples of network construction

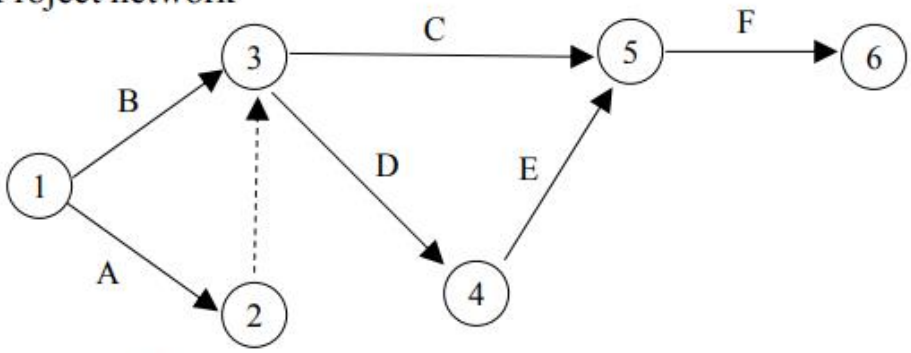


Example 1.

A project to manufacture a product is composed of the following activities:

Activity	Predecessors	Duration (days)
A = train workers	--	6
B = purchase row material	--	9
C = manufacture product 1	A, B	8
D = manufacture product 2	A, B	7
E = test product 2	D	10
F = Assemble products 1 and 2	C, E	12

Project network

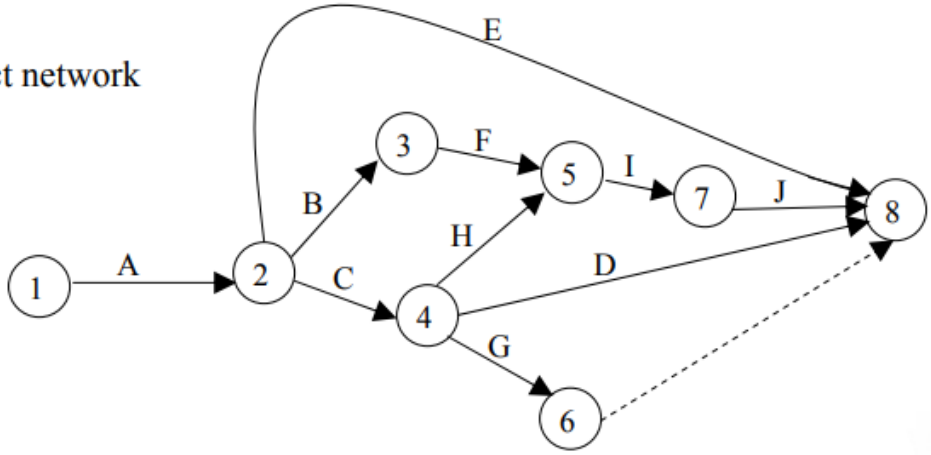


- **Example 2.**

The promoter of a rock concert must perform the following tasks before the concert can be held. Durations are in days.

Activity	Predecessors	Duration
A = find site	--	3
B = find engineers	A	2
C = hire opening act	A	6
D = set radio and TV ads	C	3
E = set up tickets agents	A	3
F = prepare electronics	B	3
G = print advertising	C	5
H = set up transportation	C	1
I = rehearsals	F, H	1.5
J = last minute details	I	2

Project network



- **Early/late event time**
  - The *early event time* for node (event) *i*, $ET(i)$ is the earliest time at which the event corresponding to node *i* can occur without violating precedence,

$$ET(i) = \max_{k \in B_i}(ET(k) + t_{ki}), \qquad (1)$$

    where B_i is the set of nodes directly preceding *i* and t_{ik} is the duration of the activity with start node *k* and end node *i*.
  - Note that for the node representing beginning of the project $ET(1) = 0$. Then, $ET(i)$ is determined from (1) recursively.
  - The *late event time* for node *i*, $LT(i)$ is the latest time at which the event corresponding to *i* can occur without delaying the completion of the project,

$$LT(i) = \min_{k \in A_i}(LT(k) - t_{ik}), \qquad (2)$$

    where A_i is the set of nodes directly succeeding *i*.
  - Note that for finish node, *n*, representing end of the project $LT(n) = ET(n)$. Then, $LT(i)$ is determined from (2) recursively.
  - Note also that $ET(n)$ represents the *minimum time required for the project completion* .

- **Early/late start/finish activity times**
  - Based on the early and late event times of start node i and end node j, we define the following useful quantities for activity (i, j).
  - The *early start time*, $ES(i, j)$, is the earliest time at which the activity could start, $ES(i, j) = ET(i)$.
  - The *early finish time*, $EF(i, j)$, is the earliest time at which the activity could be completed, $EF(i, j) = ET(i) + t_{ij}$.
  - The *late finish time*, $LF(i, j)$, is the latest time at which the activity could be completed without delaying the project, $LF(i, j) = LT(j)$.
  - The *late start time*, $LS(i, j)$, is the latest time at which the activity could be started without delaying the project, $LS(i, j) = LT(j) - t_{ij}$.

  Remark. Because nodes i and j can represent the start and end of several activities, $LT(i)$ and $ET(j)$, in general, have no direct physical interpretation related to activity (i, j).

- **Floats**
  - Floats are slack times by which an activity can be delayed.
  - The *total float* of an activity is the amount by which the start time of the activity can be delayed without delaying the completion of the project.

- For an activity represented by an arc connecting nodes i and j, the total float is

$$TF(i,j) = LT(j) - ET(i) - t_{ij}. \qquad (3)$$

- The *free float* of an activity is the amount by which the start time of the activity can be delayed without delaying the start of any later activity beyond its earliest starting time.
- For an activity represented by an arc connecting nodes i and j, the free float is

$$FF(i,j) = ET(j) - ET(i) - t_{ij}.$$

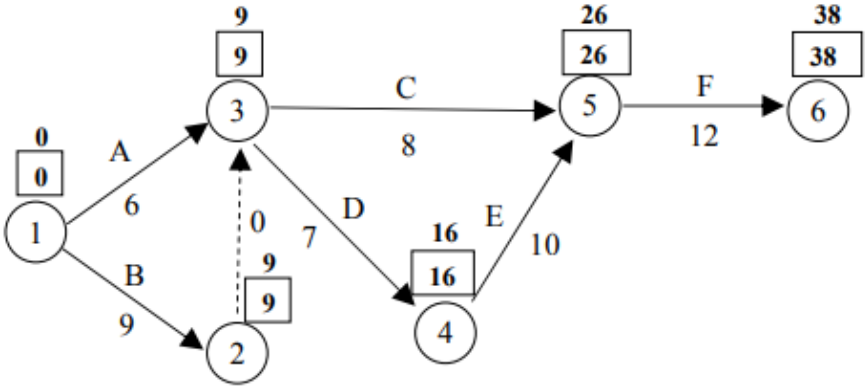
- If $FF(i,j) < TF(i,j)$, and activity (i,j) starts at time t, such that $FF(i,j) < t < TF(i,j)$, then activity (j,k) succeeding (i,j) cannot start before $ES(j) + t - FF(i,j)$.

- **Critical path**
  - A *critical activity* is an activity that cannot be delayed without delaying the completion of the project.
  - That is, a delay of Δ days on a critical activity will increase the length of the project by Δ days.
  - Critical activity should be monitored carefully to avoid delays.
  - A critical activity has a total float of zero.
  - A path from the start node to the finish node that consists entirely of critical nodes is a *critical path*.
  - A critical path is the longest path from start node to finish node.
  - The length of the critical path is the minimum time required for

project completion. It is equal to $LT(n) = ET(n)$, where n is the finish node.

- **Example 3.**

  Critical path for the network in Example 1.



Starting with $ET(1) = 0$, utilizing (1) the early event times, $ET(i)$, $i = 2, \ldots, 6$, are obtained as shown in the rectangles above each node. To find the late event times, $LT(i)$, start with node 6 where $LT(6) = ET(6) = 38$. Then, utilizing (2), $LT(i)$, $i = 5, \ldots, 1$, are obtained as shown above the little squares on top of each node.

Next, the total floats for each activity, $TF(i, j)$, are evaluated from (3).

Activity (i, j)	**Duration (t_{ij})**	**$TF(i, j)$**
A (1,3)	6	$9 - 0 - 6 = 3$
B (1, 2)	9	$9 - 0 - 9 = 0$
Dummy (2, 3)	0	$9 - 9 - 0 = 0$
C (3, 5)	8	$26 - 9 - 8 = 9$
D (3, 4)	7	$16 - 9 - 7 = 0$
E (4, 5)	10	$26 - 16 - 10 = 0$
F (5, 6)	12	$36 - 26 - 12 = 0$

Critical activities are those with $TF(i, j) = 0$. Then, the critical path is $B \rightarrow D \rightarrow E \rightarrow F$.

PERT

Before any activity begins related to the work of a project, every project requires an advanced, accurate time estimate. Without an accurate estimate, no project can be completed within the budget and the target completion date.

Developing an estimate is a complex task. If the project is large and has many stakeholders, things can be more complex.

Therefore, there have been many initiatives to come up with different techniques for estimation phase of the project in order to make the estimation more accurate.

PERT (Program Evaluation and Review Technique) is one of the successful and proven methods among the many other techniques, such as, CPM, Function Point Counting, Top-Down Estimating, WAVE, etc.

PERT was initially created by the US Navy in the late 1950s. The pilot project was for developing Ballistic Missiles and there have been thousands of contractors involved.

After PERT methodology was employed for this project, it actually ended two years ahead of its initial schedule.

The PERT Basics

At the core, PERT is all about management probabilities. Therefore, PERT involves in many simple statistical methods as well.

Sometimes, people categorize and put PERT and CPM together. Although CPM (Critical Path Method) shares some characteristics with PERT, PERT has a different focus.

Same as most of other estimation techniques, PERT also breaks down the tasks into detailed activities.

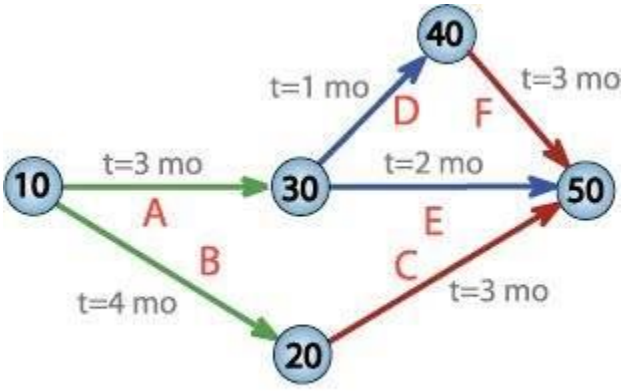
Then, a Gantt chart will be prepared illustrating the interdependencies among the activities. Then, a *network* of activities and their interdependencies are drawn in an illustrative manner.

In this map, a *node* represents each event. The activities are represented as arrows and they are drawn from one event to another, based on the sequence.

Next, the Earliest Time (TE) and the Latest Time (TL) are figured for each activity and identify the slack time for each activity.

When it comes to deriving the estimates, the PERT model takes a statistical route to do that. We will cover more on this in the next two sections.

Following is an example PERT chart:



PERT network chart for a seven-month project with five milestones (10 through 50) and six activities (A through F)

The Three Chances

There are three estimation times involved in PERT; Optimistic Time Estimate (TOPT), Most Likely Time Estimate (TLIKELY), and Pessimistic Time Estimate (TPESS).

In PERT, these three estimate times are derived for each activity. This way, a range of time is given for each activity with the most probable value, TLIKELY.

Following are further details on each estimate:

1. TOPT

This is the fastest time an activity can be completed. For this, the assumption is made that all the necessary resources are available and all predecessor activities are completed as planned.

2. TLIKELY

Most of the times, project managers are asked only to submit one estimate. In that case, this is the estimate that goes to the upper management.

3. TPESS

This is the maximum time required to complete an activity. In this case, it is assumed that many things go wrong related to the activity. A lot of rework and resource unavailability are assumed when this estimation is derived.

The PERT Mathematics

BETA probability distribution is what works behind PERT. The expected completion time (E) is calculated as below:

```
E = (TOPT + 4 x TLIEKLY + TPESS) / 6
```

At the same time, the possible variance (V) of the estimate is calculated as below:

```
V = (TPESS - TOPT)^2 / 6^2
```

Now, following is the process we follow with the two values:

- For every activity in the critical path, E and V are calculated.
- Then, the total of all Es are taken. This is the overall expected completion time for the project.
- Now, the corresponding V is added to each activity of the critical path. This is the variance for the entire project. This is done only for the activities in the critical path as only the critical path activities can accelerate or delay the project duration.
- Then, standard deviation of the project is calculated. This equals to the square root of the variance (V).
- Now, the normal probability distribution is used for calculating the project completion time with the desired probability.
- The best thing about PERT is its ability to integrate the uncertainty in project times estimations into its methodology.

- It also makes use of many assumption that can accelerate or delay the project progress. Using PERT, project managers can have an idea of the possible time variation for the deliveries and offer delivery dates to the client in a safer manner.

DIFFERENCE BETWEEN PERT AND CPM:

1. PERT is event oriented whereas CPM is activity oriented. In simple words, in PERT network interest is focused upon start or completion of events and not on activities themselves.

2. In CPM network no allowance is made for uncertainties in the duration of time involved whereas in PERT network uncertainty is considered.

3. In PERT, time is not related to cost whereas in CPM the object is to develop an optimum time cost relationship. However, PERT has since been extended in this direction and the line dividing PERT/CPM is gradually fading out.

4. In CPM duration of activity is estimated with a fair degree of accuracy. In PERT duration of activities are not so accurate and definite.

5. In CPM both time and cost can be controlled during planning. Pert is basically a tool for planning.

6. PERT is used in research and development project, basically for non-repetitive type projects.

CPM is widely used in construction projects.

5.2 – Project Tracking –

a. Time-line charts or GANTT charts

When creating a software project schedule, you begin with a set of tasks (the work breakdown structure). If automated tools are used, the work breakdown is input as a task network or task outline. Effort, duration, and start date are then input for each task. In addition, tasks may be assigned to specific individuals.

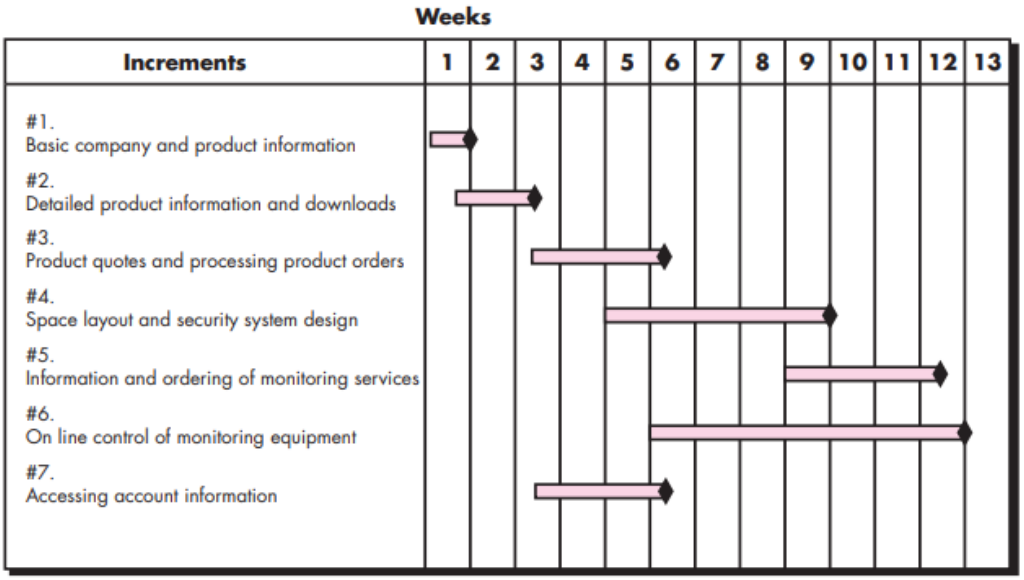
As a consequence of this input, a time-line chart, also called a Gantt chart, is generated. A time-line chart can be developed for the entire project. Alternatively, separate charts can be developed for each project function or for each individual working on the project.

Figure below illustrates the format of a time-line chart. It depicts a part of a software project schedule that emphasizes the concept scoping task for a word-processing (WP) software product. All project tasks (for concept scoping) are listed in the left-hand column. The horizontal bars indicate the duration of each task. When multiple bars occur at the same time on the calendar, task concurrency is implied. The diamonds indicate milestones.

An example time-line chart

Work tasks	Week 1	Week 2	Week 3	Week 4	Week 5
I.1.1 Identify needs and benefits					
Meet with customers					
Identify needs and project constraints					
Establish product statement					
Milestone: Product statement defined					
I.1.2 Define desired output/control/input (OCI)					
Scope keyboard functions					
Scope voice input functions					
Scope modes of interaction					
Scope document diagnosis					
Scope other WP functions					
Document OCI					
FTR: Review OCI with customer					
Revise OCI as required					
Milestone: OCI defined					
I.1.3 Define the function/behavior					
Define keyboard functions					
Define voice input functions					
Describe modes of interaction					
Describe spell/grammar check					
Describe other WP functions					
FTR: Review OCI definition with customer					
Revise as required					
Milestone: OCI definition complete					
I.1.4 Isolation software elements					
Milestone: Software elements defined					
I.1.5 Research availability of existing software					
Research text editing components					
Research voice input components					
Research file management components					
Research spell/grammar check components					
Milestone: Reusable components identified					
I.1.6 Define technical feasibility					
Evaluate voice input					
Evaluate grammar checking					
Milestone: Technical feasibility assessed					
I.1.7 Make quick estimate of size					
I.1.8 Create a scope definition					
Review scope document with customer					
Revise document as required					
Milestone: Scope document complete					

Time line for macroscopic project schedule



Once the information necessary for the generation of a time-line chart has been input, the majority of software project scheduling tools produce project tables—a tabular listing of all project tasks, their planned and actual start and end dates, and a variety of related information (Figure below). Used in conjunction with the time-line chart, project tables enable you to track progress.

An example project table

Work tasks	Planned start	Actual start	Planned complete	Actual complete	Assigned person	Effort allocated	Notes
I.1.1 Identify needs and benefits							Scoping will require more effort/time
Meet with customers	wk1, d1	wk1, d1	wk1, d2	wk1, d2	BLS	2 p-d	
Identify needs and project constraints	wk1, d2	wk1, d2	wk1, d2	wk1, d2	JPP	1 p-d	
Establish product statement	wk1, d3	wk1, d3	wk1, d3	wk1, d3	BLS/JPP	1 p-d	
Milestone: Product statement defined	wk1, d3	wk1, d3	wk1, d3	wk1, d3			
I.1.2 Define desired output/control/input (OCI)							
Scope keyboard functions	wk1, d4	wk1, d4	wk2, d2		BLS	1.5 p-d	
Scope voice input functions	wk1, d3	wk1, d3	wk2, d2		JPP	2 p-d	
Scope modes of interaction	wk2, d1		wk2, d3		MLL	1 p-d	
Scope document diagnostics	wk2, d1		wk2, d2		BLS	1.5 p-d	
Scope other WP functions	wk1, d4	wk1, d4	wk2, d3		JPP	2 p-d	
Document OCI	wk2, d1		wk2, d3		MLL	3 p-d	
FTR: Review OCI with customer	wk2, d3		wk2, d3		all	3 p-d	
Revise OCI as required	wk2, d4		wk2, d4		all	3 p-d	
Milestone: OCI defined	wk2, d5		wk2, d5				
I.1.3 Define the function/behavior							

b. Tracking the schedule –

If it has been properly developed, the project schedule becomes a road map that defines the tasks and milestones to be tracked and controlled as the project proceeds. Tracking can be accomplished in a number of different ways:

• Conducting periodic project status meetings in which each team member reports progress and problems

• Evaluating the results of all reviews conducted throughout the software engineering process

• Determining whether formal project milestones (the diamonds shown in Figure 27.3) have been accomplished by the scheduled date

• Comparing the actual start date to the planned start date for each project task listed in the resource table

• Meeting informally with practitioners to obtain their subjective assessment of progress to date and problems on the horizon

• Using earned value analysis to assess progress quantitatively

c. Earned value analysis –

A technique for performing quantitative analysis of progress does exist. It is called earned value analysis (EVA). The earned value system provides a common value scale for every [software project] task, regardless of the type of work being performed. The total hours to do the whole project are estimated, and every task is given an earned value based on its estimated percentage of the total.

Stated even more simply, earned value is a measure of progress. It enables you to assess the "percent of completeness" of a project using quantitative analysis rather than rely on a gut feeling.

To determine the earned value, the following steps are performed:

1. The budgeted cost of work scheduled (BCWS) is determined for each work task represented in the schedule. During estimation, the work (in person-hours or person-days) of each software engineering task is planned. Hence, $BCWS_i$ is the effort planned for work task i. To determine progress at a given point along the project schedule, the value of BCWS is the sum of the $BCWS_i$ values for all work tasks that should have been completed by that point in time on the project schedule.

2. The BCWS values for all work tasks are summed to derive the budget at completion (BAC).

Hence,

$BAC = \Sigma(BCWS_k)$ for all tasks k

3. Next, the value for budgeted cost of work performed (BCWP) is computed. The value for BCWP is the sum of the BCWS values for all work tasks that have actually been completed by a point in time on the project schedule.

The distinction between the BCWS and the BCWP is that the former represents the budget of the activities that were planned to be completed and the latter represents the budget of the activities that actually were completed."

Given values for BCWS, BAC, and BCWP, important progress indicators can be computed:

Schedule performance index, SPI = BCWP/BCWS

Schedule variance, SV BCWP - BCWS

SPI is an indication of the efficiency with which the project is utilizing scheduled resources. An SPI value close to 1.0 indicates efficient execution of the project schedule. SV is simply an absolute indication of variance from the planned schedule.

Percent scheduled for completion = BCWS/BAC

provides an indication of the percentage of work that should have been completed by time t.

Percent complete = BCWP/BAC

provides a quantitative indication of the percent of completeness of the project at a given point in time t. It is also possible to compute the actual cost of work performed (ACWP). The value for ACWP is the sum of the effort actually expended on work tasks that have been completed by a point in time on the project schedule. It is then possible to compute

Cost performance index, CPI = BCWP/ACWP

Cost variance, CV BCWP – ACWP

A CPI value close to 1.0 provides a strong indication that the project is within its defined budget. CV is an absolute indication of cost savings (against planned costs) or shortfall at a particular stage of a project. Like over-the-horizon radar, earned value analysis illuminates scheduling difficulties before they might otherwise be apparent. This enables you to take corrective action before a project crisis develops.

5.3 –

A. Software Quality Management –

Software Quality Management is a process that ensures the required level of software quality is achieved when it reaches the users, so that they are satisfied by its performance. The process involves quality assurance, quality planning, and quality control. This tutorial provides a complete overview of Software Quality Management and describes the various steps involved in the process. The entire content is divided into sections for easy understanding.

Quality software refers to a software which is reasonably bug or defect free, is delivered in time and within the specified budget, meets the requirements and/or expectations, and is maintainable. In the software engineering context, software quality reflects both **functional quality** as well as **structural quality**.

- **Software Functional Quality** – It reflects how well it satisfies a given design, based on the functional requirements or specifications.
- **Software Structural Quality** – It deals with the handling of non-functional requirements that support the delivery of the functional requirements, such as robustness or maintainability, and the degree to which the software was produced correctly.
- **Software Quality Assurance** – Software Quality Assurance (SQA) is a set of activities to ensure the quality in software engineering processes that ultimately

result in quality software products. The activities establish and evaluate the processes that produce products. It involves process-focused action.

- **Software Quality Control** – Software Quality Control (SQC) is a set of activities to ensure the quality in software products. These activities focus on determining the defects in the actual products produced. It involves product-focused action.

The Software Quality Challenge

In the software industry, the developers will never declare that the software is free of defects, unlike other industrial product manufacturers usually do. This difference is due to the following reasons.

Product Complexity

It is the number of operational modes the product permits. Normally, an industrial product allows only less than a few thousand modes of operation with different combinations of its machine settings. However, software packages allow millions of operational possibilities. Hence, assuring of all these operational possibilities correctly is a major challenge to the software industry.

Product Visibility

Since the industrial products are visible, most of its defects can be detected during the manufacturing process. Also the absence of a part in an industrial product can be easily detected in the product. However, the defects in software products which are stored on diskettes or CDs are invisible.

Product Development and Production Process

In an industrial product, defects can be detected during the following phases –

- **Product development** – In this phase, the designers and Quality Assurance (QA) staff checks and tests the product prototype to detect its defects.
- **Product production planning** – During this phase, the production process and tools are designed and prepared. This phase also provides opportunities to inspect the product to detect the defects that went unnoticed during the development phase.

- **Manufacturing** – In this phase, QA procedures are applied to detect failures of products themselves. Defects in the product detected in the first period of manufacturing can usually be corrected by a change in the product's design or materials or in the production tools, in a way that eliminates such defects in products manufactured in future.

However, in the case of software, the only phase where defects can be detected is the development phase. In case of software, product production planning and manufacturing phases are not required as the manufacturing of software copies and the printing of software manuals are conducted automatically.

The factors affecting the detection of defects in software products versus other industrial products are shown in the following table.

Characteristic	Software Products	Other Industrial Products
Complexity	Millions of operational options	thousand operational options
visibility of product	Invisible Product Difficult to detect defects by sight	Visible Product Effective detection of defects by sight
Nature of development and production process	can defect defects in only one phase	can detect defects in all of the following phases • Product development • Product production planning • Manufacturing

These characteristics of software such as complexity and invisibility make the development of software quality assurance methodology and its successful implementation a highly professional challenge.

B. Software Quality Assurance -

Software Quality Assurance (SQA) is a set of activities for ensuring quality in software engineering processes. It ensures that developed software meets and complies with the defined or standardized quality specifications. SQA is an ongoing process within the Software Development Life Cycle (SDLC) that routinely checks the developed software to ensure it meets the desired quality measures.

SQA practices are implemented in most types of software development, regardless of the underlying software development model being used. SQA incorporates and implements software testing methodologies to test the software. Rather than checking for quality after completion, SQA processes test for quality in each phase of development, until the software is complete. With SQA, the software development process moves into the next phase only once the current/previous phase complies with the required quality standards. SQA generally works on one or more industry standards that help in building software quality guidelines and implementation strategies.

It includes the following activities –

- Process definition and implementation
- Auditing
- Training

Processes could be –

- Software Development Methodology
- Project Management
- Configuration Management
- Requirements Development/Management
- Estimation
- Software Design
- Testing, etc.

Once the processes have been defined and implemented, Quality Assurance has the following responsibilities –

- Identify the weaknesses in the processes
- Correct those weaknesses to continually improve the process

Components of SQA System

An SQA system always combines a wide range of SQA components. These components can be classified into the following six classes –

Pre-project components

This assures that the project commitments have been clearly defined considering the resources required, the schedule and budget; and the development and quality plans have been correctly determined.

Components of project life cycle activities assessment

The project life cycle is composed of two stages: the development life cycle stage and the operation–maintenance stage.

The development life cycle stage components detect design and programming errors. Its components are divided into the following sub-classes: Reviews, Expert opinions, and Software testing.

The SQA components used during the operation–maintenance phase include specialized maintenance components as well as development life cycle components, which are applied mainly for functionality to improve the maintenance tasks.

Components of infrastructure error prevention and improvement

The main objective of these components, which is applied throughout the entire organization, is to eliminate or at least reduce the rate of errors, based on the organization's accumulated SQA experience.

Components of software quality management

This class of components deal with several goals, such as the control of development and maintenance activities, and the introduction of early managerial support actions that mainly prevent or minimize schedule and budget failures and their outcomes.

Components of standardization, certification, and SQA system assessment

These components implement international professional and managerial standards within the organization. The main objectives of this class are utilization of international professional knowledge, improvement of coordination of the organizational quality systems with other organizations, and assessment of the achievements of quality systems according to a common scale. The various standards may be classified into two main groups: quality management standards and project process standards.

Organizing for SQA – the human components

The SQA organizational base includes managers, testing personnel, the SQA unit and the persons interested in software quality such as SQA trustees, SQA committee members, and SQA forum members. Their main objectives are to initiate and support the implementation of SQA components, detect deviations from SQA procedures and methodology, and suggest improvements.

Pre-project Software Quality Components

These components help to improve the preliminary steps taken before starting a project. It includes –

- Contract Review
- Development and Quality Plans

Contract Review

Normally, a software is developed for a contract negotiated with a customer or for an internal order to develop a firmware to be embedded within a hardware product. In all these cases, the development unit is committed to an agreed-upon functional specification, budget and schedule. Hence, contract review activities must include a detailed examination of the project proposal draft and the contract drafts.

Specifically, contract review activities include –

- Clarification of the customer’s requirements
- Review of the project’s schedule and resource requirement estimates

- Evaluation of the professional staff's capacity to carry out the proposed project
- Evaluation of the customer's capacity to fulfil his obligations
- Evaluation of development risks

Development and Quality Plans

After signing the software development contract with an organization or an internal department of the same organization, a development plan of the project and its integrated quality assurance activities are prepared. These plans include additional details and needed revisions based on prior plans that provided the basis for the current proposal and contract.

Most of the time, it takes several months between the tender submission and the signing of the contract. During these period, resources such as staff availability, professional capabilities may get changed. The plans are then revised to reflect the changes that occurred in the interim.

The main issues treated in the project development plan are –

- Schedules
- Required manpower and hardware resources
- Risk evaluations
- Organizational issues: team members, subcontractors and partnerships
- Project methodology, development tools, etc.
- Software reuse plans

The main issues treated in the project's quality plan are –

- Quality goals, expressed in the appropriate measurable terms
- Criteria for starting and ending each project stage
- Lists of reviews, tests, and other scheduled verification and validation activities

C. Elements of SQA –

Software quality assurance encompasses a broad range of concerns and activities that focus on the management of software quality. These can be summarized in the following manner :

Standards. The IEEE, ISO, and other standards organizations have produced a broad array of software engineering standards and related documents. Standards may be adopted voluntarily by a software engineering organization or imposed by the customer or other stakeholders. The job of SQA is to ensure that standards that have been adopted are followed and that all work products conform to them.

Reviews and audits. Technical reviews are a quality control activity performed by software engineers for software engineers. Their intent is to uncover errors. Audits are a type of review performed by SQA personnel with the intent of ensuring that quality guidelines are being followed for software engineering work. For example, an audit of the review process might be conducted to ensure that reviews are being performed in a manner that will lead to the highest likelihood of uncovering errors.

Testing. Software testing is a quality control function that has one primary goal—to find errors. The job of SQA is to ensure that testing is properly planned and efficiently conducted so that it has the highest likelihood of achieving its primary goal.

Error/defect collection and analysis. The only way to improve is to measure how you're doing. SQA collects and analyzes error and defect data to better understand how errors are introduced and what software engineering activities are best suited to eliminating them.

Change management. Change is one of the most disruptive aspects of any software project. If it is not properly managed, change can lead to confusion, and confusion almost always leads to poor quality. SQA ensures that adequate change management practices have been instituted.

Education. Every software organization wants to improve its software engineering practices. A key contributor to improvement is education of software engineers, their managers, and other stakeholders. The SQA organization takes the lead in software process improvement and is a key proponent and sponsor of educational programs.

Vendor management. Three categories of software are acquired from external software vendors—shrink-wrapped packages (e.g., Microsoft Office), a tailored shell that provides a basic skeletal structure that is custom tailored to the needs of a purchaser, and contracted software that is custom designed and constructed from specifications provided by the customer organization. The job of the SQA organization is to ensure that high-quality software results by suggesting specific quality practices that the vendor should follow (when possible), and incorporating quality mandates as part of any contract with an external vendor.

Security management. With the increase in cyber crime and new government regulations regarding privacy, every software organization should institute policies that protect data at all levels, establish firewall protection for WebApps, and ensure that software has not been tampered with internally. SQA ensures that appropriate process and technology are used to achieve software security.

Safety. Because software is almost always a pivotal component of humanrated systems (e.g., automotive or aircraft applications), the impact of hidden defects can be catastrophic. SQA may be

responsible for assessing the impact of software failure and for initiating those steps required to reduce risk.

Risk management. Although the analysis and mitigation of risk is the concern of software engineers, the SQA organization ensures that risk management activities are properly conducted and that risk-related contingency plans have been established.

In addition to each of these concerns and activities, SQA works to ensure that software support activities (e.g., maintenance, help lines, documentation, and manuals) are conducted or produced with quality as a dominant concern.

5.4 Quality Evaluation Standards –

A. Six Sigma –

Six Sigma is the most widely used strategy for statistical quality assurance in industry today. Originally popularized by Motorola in the 1980s, the Six Sigma strategy "is a rigorous and disciplined methodology that uses data and statistical analysis to measure and improve a company's operational performance by identifying and eliminating defects' in manufacturing and service-related processes".

The term Six Sigma is derived from six standard deviations—3.4 instances (defects) per million occurrences—implying an extremely high quality standard. The Six Sigma methodology defines three core steps:

• Define customer requirements and deliverables and project goals via well-defined methods of customer communication.

• Measure the existing process and its output to determine current quality performance (collect defect metrics).

• Analyze defect metrics and determine the vital few causes.

If an existing software process is in place, but improvement is required, Six Sigma suggests two additional steps:

• Improve the process by eliminating the root causes of defects.

• Control the process to ensure that future work does not reintroduce the causes of defects.

These core and additional steps are sometimes referred to as the DMAIC (define, measure, analyze, improve, and control) method. If an organization is developing a software process (rather than improving an existing process), the core steps are augmented as follows:

• Design the process to (1) avoid the root causes of defects and (2) to meet customer requirements.

• Verify that the process model will, in fact, avoid defects and meet customer requirements.

This variation is sometimes called the DMADV (define, measure, analyze, design, and verify) method. A comprehensive discussion of Six Sigma is best left to resources dedicated to the subject.

B. ISO for software –

ISO 9000-3, the Guidelines offered by the International Organization for Standardization (ISO), represent implementation of the general methodology of quality management ISO 9000 Standards to the special case of software development and maintenance.

Eight principles guide the new ISO 9000-3 standard; these were originally set down in the ISO 9000:2000 standard (ISO, 2000b), as follows:

(1) Customer focus. Organizations depend on their customers and therefore should understand current and future customer needs.

(2) Leadership. Leaders establish the organization's vision. They should create and maintain an internal environment in which people can become fully involved in achieving the organization's objectives via the designated route.

(3) Involvement of people. People are the essence of an organization; their full involvement, at all levels of the organization, enables their abilities to be applied for the organization's benefit.

(4) Process approach. A desired result is achieved more efficiently when activities and resources are managed as a process.

(5) System approach to management. Identifying, understanding and managing processes, if viewed as a system, contributes to the organization's effectiveness and efficiency.

(6) Continual improvement. Ongoing improvement of overall performance should be high on the organization's agenda.

(7) Factual approach to decision making. Effective decisions are based on the analysis of information.

(8) Mutually supportive supplier relationships. An organization and its suppliers are interdependent; a mutually supportive relationship enhances the ability of both to create added value.

C. CMMI Levels, Process areas –

CMMI is a process improvement approach that provides organizations with the essential elements of effective processes. CMMI can help you make decisions about your process improvement plans.

CMMI can help you make decisions about your process improvement plans.

Process improvement is continuous improvement. We can never reach perfection. In this tutorial, we will learn CMM that is a continuously evolving and improving model where the focus is always on doing better. Our reach should always exceed our grasp.

➢ What is CMM?

- CMM stands for **C**apability **M**aturity **M**odel.
- Focuses on elements of essential practices and processes from various bodies of knowledge.
- Describes common sense, efficient, proven ways of doing business (which you should already be doing) - not a radical new approach.
- CMM is a method to evaluate and measure the maturity of the software development process of an organization.
- CMM measures the maturity of the software development process on a scale of 1 to 5.

CMM was originally developed for Software Development and Maintenance but later it was developed for –

- Systems Engineering
- Supplier Sourcing
- Integrated Product and Process Development
- People CMM
- Software Acquisition

➢ What is CMMI?

CMM Integration project was formed to sort out the problem of using multiple CMMs. CMMI product team's mission was to combine three **Source Models**into a single improvement framework for the organizations pursuing enterprise-wide process improvement. These three Source Models are –

- Capability Maturity Model for Software (SW-CMM) - v2.0 Draft C.
- Electronic Industries Alliance Interim Standard (EIA/IS) - 731 Systems Engineering.

- Integrated Product Development Capability Maturity Model (IPD-CMM) v0.98.

CMM Integration

- Builds an initial set of integrated models.
- Improves best practices from source models based on lessons learned.
- Establishes a framework to enable integration of future models.

The CMMI is structured as follows –

- Maturity Levels (staged representation) or Capability Levels (continuous representation)
- Process Areas
- Goals: Generic and Specific
- Common Features
- Practices: Generic and Specific

➢ CMMI Representation

This chapter will discuss about two CMMI representations and rest of the subjects will be covered in subsequent chapters.

A representation allows an organization to pursue different improvement objectives. An organization can go for one of the following two improvement paths.

Staged Representation

The staged representation is the approach used in the Software CMM. It is an approach that uses predefined sets of process areas to define an improvement path for an organization. This improvement path is described by a model component called a Maturity Level. A maturity level is a well-defined evolutionary plateau towards achieving improved organizational processes.

CMMI Staged Representation

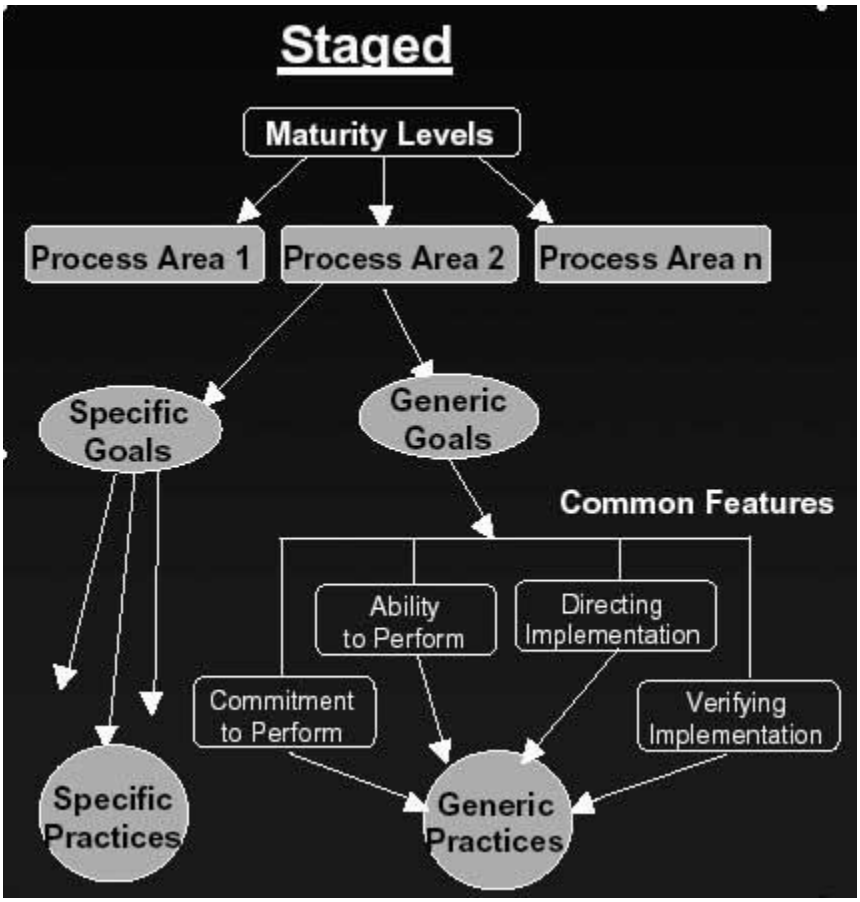
- Provides a proven sequence of improvements, each serving as a foundation for the next.

- Permits comparisons across and among organizations by the use of maturity levels.
- Provides an easy migration from the SW-CMM to CMMI.
- Provides a single rating that summarizes appraisal results and allows comparisons among organizations.

Thus Staged Representation provides a pre-defined roadmap for organizational improvement based on proven grouping and ordering of processes and associated organizational relationships. You cannot divert from the sequence of steps.

CMMI Staged Structure

Following picture illustrates CMMI Staged Model Structure.



Continuous Representation

Continuous representation is the approach used in the SECM and the IPD-CMM. This approach allows an organization to select a specific process area

and make improvements based on it. The continuous representation uses Capability Levels to characterize improvement relative to an individual process area.

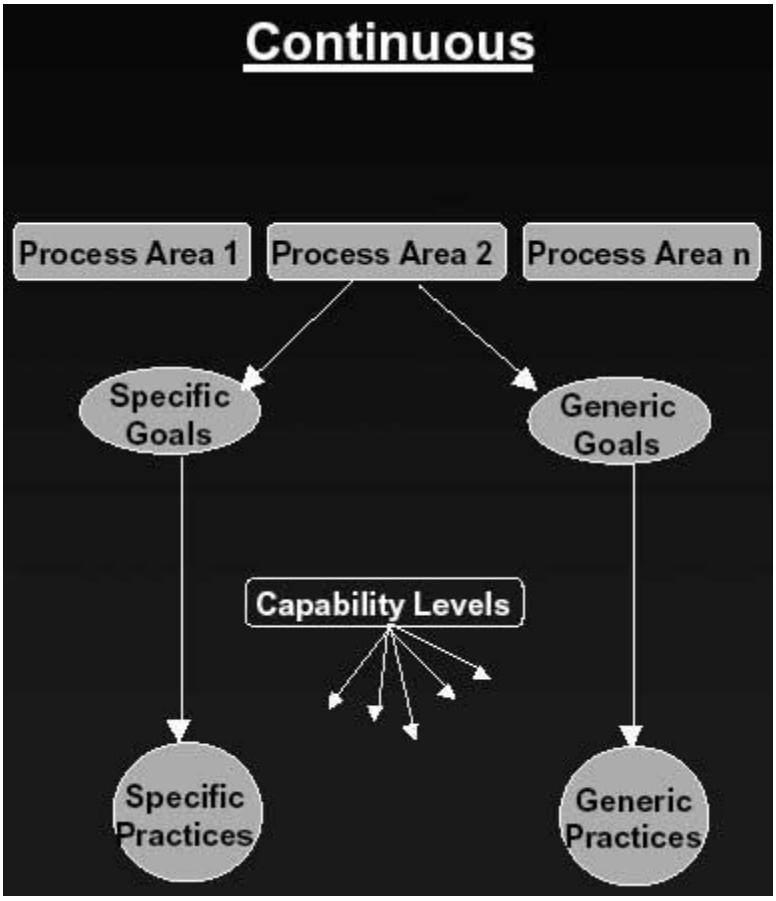
CMMI Continuous Representation

- Allows you to select the order of improvement that best meets your organization's business objectives and mitigates your organization's areas of risk.
- Enables comparisons across and among organizations on a process-area-by-process-area basis.
- Provides an easy migration from EIA 731 (and other models with a continuous representation) to CMMI.

Thus Continuous Representation provides flexibility to organizations to choose the processes for improvement, as well as the amount of improvement required.

CMMI Continuous Structure

The following picture illustrates the CMMI Continuous Model Structure.



- CMMI - Maturity Levels

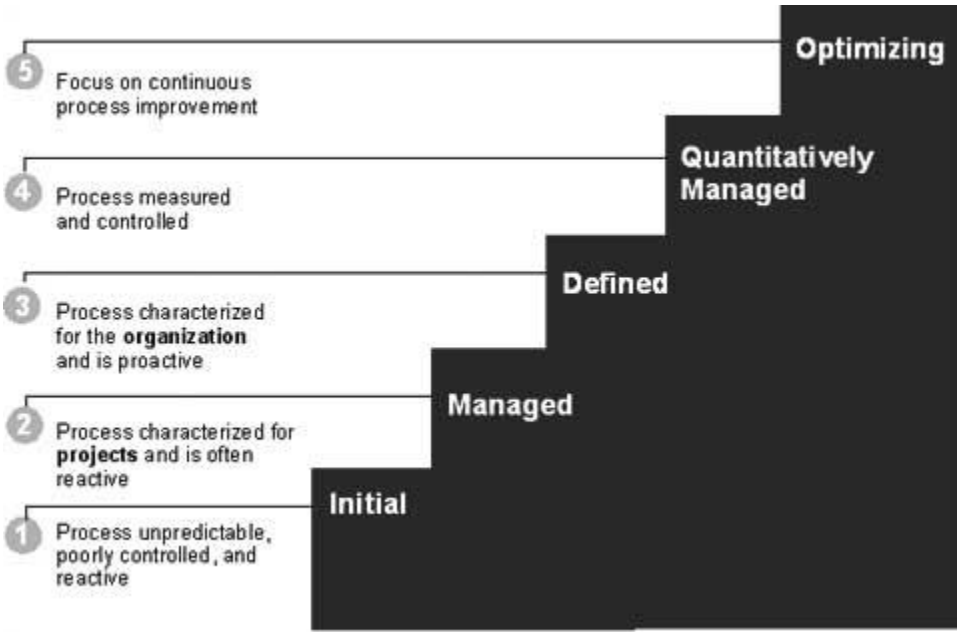
A maturity level is a well-defined evolutionary plateau toward achieving a mature software process. Each maturity level provides a layer in the foundation for continuous process improvement.

CMMI models with staged representation, have five maturity levels designated by the numbers 1 through 5. They are –

- Initial
- Managed
- Defined
- Quantitatively Managed
- Optimizing

CMMI Staged Representation Maturity Levels

The following image shows the maturity levels in a CMMI staged representation.



Now we will learn the details about each maturity level. Next section will list down all the process areas related to these maturity levels.

Maturity Level Details

Maturity levels consist of a predefined set of process areas. The maturity levels are measured by the achievement of the **specific** and **generic goals**that apply to each predefined set of process areas. The following sections describe the characteristics of each maturity level in detail.

Maturity Level 1 Initial

At maturity level 1, processes are usually ad hoc and chaotic. The organization usually does not provide a stable environment. Success in these organizations depend on the competence and heroics of the people in the organization and not on the use of proven processes.

Maturity level 1 organizations often produce products and services that work; however, they frequently exceed the budget and schedule of their projects.

Maturity level 1 organizations are characterized by a tendency to over commit, abandon processes in the time of crisis, and not be able to repeat their past successes.

Maturity Level 2 Managed

At maturity level 2, an organization has achieved all the **specific** and **generic goals** of the maturity level 2 process areas. In other words, the projects of the organization have ensured that requirements are managed and that processes are planned, performed, measured, and controlled.

The process discipline reflected by maturity level 2 helps to ensure that existing practices are retained during times of stress. When these practices are in place, projects are performed and managed according to their documented plans.

At maturity level 2, requirements, processes, work products, and services are managed. The status of the work products and the delivery of services are visible to management at defined points.

Commitments are established among relevant stakeholders and are revised as needed. Work products are reviewed with stakeholders and are controlled.

The work products and services satisfy their specified requirements, standards, and objectives.

Maturity Level 3 Defined

At maturity level 3, an organization has achieved all the **specific** and **generic goals** of the process areas assigned to maturity levels 2 and 3.

At maturity level 3, processes are well characterized and understood, and are described in standards, procedures, tools, and methods.

A critical distinction between maturity level 2 and maturity level 3 is the scope of standards, process descriptions, and procedures. At maturity level 2, the standards, process descriptions, and procedures may be quite different in each specific instance of the process (for example, on a particular project).

At maturity level 3, the standards, process descriptions, and procedures for a project are tailored from the organization's set of standard processes to suit a particular project or organizational unit. The organization's set of standard processes includes the processes addressed at maturity level 2 and maturity level 3. As a result, the processes that are performed across the organization are consistent except for the differences allowed by the tailoring guidelines.

Another critical distinction is that at maturity level 3, processes are typically described in more detail and more rigorously than at maturity level 2. At maturity level 3, processes are managed more proactively using an understanding of the interrelationships of the process activities and detailed measures of the process, its work products, and its services.

Maturity Level 4 Quantitatively Managed

At maturity level 4, an organization has achieved all the **specific goals** of the process areas assigned to maturity levels 2, 3, and 4 and the **generic goals**assigned to maturity levels 2 and 3.

At maturity level 4, sub-processes are selected that significantly contribute to the overall process performance. These selected sub-processes are controlled using statistical and other quantitative techniques.

Quantitative objectives for quality and process performance are established and used as criteria in managing the processes. Quantitative objectives are based on the needs of the customer, end users, organization, and process implementers. Quality and process performance are understood in statistical terms and are managed throughout the life of the processes.

For these processes, detailed measures of process performance are collected and statistically analyzed. Special causes of process variation are identified and, where appropriate, the sources of special causes are corrected to prevent future occurrences.

Quality and process performance measures are incorporated into the organization's measurement repository to support fact-based decision making in the future.

A critical distinction between maturity level 3 and maturity level 4 is the predictability of process performance. At maturity level 4, the performance

of processes is controlled using statistical and other quantitative techniques, and is quantitatively predictable. At maturity level 3, processes are only qualitatively predictable.

Maturity Level 5 Optimizing

At maturity level 5, an organization has achieved all the **specific goals**of the process areas assigned to maturity levels 2, 3, 4, and 5 and the **generic goals** assigned to maturity levels 2 and 3.

Processes are continually improved based on a quantitative understanding of the common causes of variation inherent in processes.

This level focuses on continually improving process performance through both incremental and innovative technological improvements.

Maturity Levels and Process Areas

Here is a list of all the corresponding process areas defined for a software organization. These process areas may be different for different organization.

This section provides the names of the related process areas. For more details about these Process Areas go through the CMMI Process Areas Chapter.

Level	Focus	Key Process Area	Result
5 Optimizing	Continuous Process Improvement	Organizational Innovation and Deployment Causal Analysis and Resolution	Highest Quality / Lowest Risk
4 Quantitatively Managed	Quantitatively Managed	Organizational Process Performance Quantitative Project Management	Higher Quality / Lower Risk

3 Defined	Process Standardization	Requirements Development Technical Solution Product Integration Verification Validation Organizational Process Focus Organizational Process Definition Organizational Training Integrated Project Mgmt (with IPPD extras) Risk Management Decision Analysis and Resolution Integrated Teaming (IPPD only) Org. Environment for Integration (IPPD only) Integrated Supplier Management (SS only)	Medium Quality / Medium Risk
2 Managed	Basic Project Management	Requirements Management Project Planning Project Monitoring and Control Supplier Agreement Management	Low Quality / High Risk

		Measurement and Analysis Process and Product Quality Assurance Configuration Management	
1 Initial	Process is informal and Adhoc		Lowest Quality / Highest Risk

➢ CMMI - Capability Levels

A capability level is a well-defined evolutionary plateau describing the organization's capability relative to a process area. A capability level consists of related specific and generic practices for a process area that can improve the organization's processes associated with that process area. Each level is a layer in the foundation for continuous process improvement.

Thus, capability levels are cumulative, i.e., a higher capability level includes the attributes of the lower levels.

In CMMI models with a continuous representation, there are six capability levels designated by the numbers 0 through 5.

- 0 – Incomplete
- 1 – Performed
- 2 – Managed
- 3 – Defined
- 4 – Quantitatively Managed
- 5 – Optimizing

A short description of each capability level is as follows –

Capability Level 0: Incomplete

An "incomplete process" is a process that is either not performed or partially performed. One or more of the specific goals of the process area are not satisfied and no generic goals exist for this level since there is no reason to institutionalize a partially performed process.

This is tantamount to Maturity Level 1 in the staged representation.

Capability Level 1: Performed

A Capability Level 1 process is a process that is expected to perform all of the Capability Level 1 specific and generic practices. Performance may not be stable and may not meet specific objectives such as quality, cost, and schedule, but useful work can be done. This is only a start, or baby-step, in process improvement. It means that you are doing something but you cannot prove that it is really working for you.

Capability Level 2: Managed

A managed process is planned, performed, monitored, and controlled for individual projects, groups, or stand-alone processes to achieve a given purpose. Managing the process achieves both the model objectives for the process as well as other objectives, such as cost, schedule, and quality. As the title of this level indicates, you are actively managing the way things are done in your organization. You have some metrics that are consistently collected and applied to your management approach.

Note – metrics are collected and used at all levels of the CMMI, in both the staged and continuous representations. It is a bitter fallacy to think that an organization can wait until Capability Level 4 to use the metrics.

Capability Level 3: Defined

A capability level 3 process is characterized as a "defined process." A defined process is a managed (capability level 2) process that is tailored from the organization's set of standard processes according to the organization's tailoring guidelines, and contributes work products, measures, and other process-improvement information to the organizational process assets.

Capability Level 4: Quantitatively Managed

A capability level 4 process is characterized as a "quantitatively managed process." A quantitatively managed process is a defined (capability level 3) process that is controlled using statistical and other quantitative techniques. Quantitative objectives for quality and process performance are established and used as criteria in managing the process. Quality and process performance is understood in statistical terms and is managed throughout the life of the process.

Capability Level 5: Optimizing

An optimizing process is a quantitatively managed process that is improved, based on an understanding of the common causes of process variation inherent to the process. It focuses on continually improving process performance through both incremental and innovative improvements. Both the defined processes and the organization's set of standard processes are the targets of improvement activities.

Capability Level 4 focuses on establishing baselines, models, and measurements for process performance. Capability Level 5 focuses on studying performance results across the organization or entire enterprise, finding common causes of problems in how the work is done (the process[es] used), and fixing the problems in the process. The fix would include updating the process documentation and training involved where the errors were injected.

Organization of Process Areas in Continuous Representation

Category	Process Area
Project Management	• Project Planning • Project Monitoring and Control • Supplier Agreement Management • Integrated Project Management(IPPD) • Integrated Supplier Management (SS) • Integrated Teaming (IPPD)

	• Risk Management Quantitative Project Management
Support	• Configuration Management • Process and Product Quality Assurance • Measurement and Analysis Causal Analysis and Resolution • Decision Analysis and Resolution • Organizational Environment for Integration (IPPD)
Engineering	• Requirements Management • Requirements Development • Technical Solution • Product Integration • Verification • Validation
Process Management	• Organizational Process Focus • Organizational Process Definition • Organizational Training • Organizational Process Performance • Organizational Innovation and Deployment

➢ CMMI - Key Process Areas

A Process Area is a cluster of related practices in an area that, when implemented collectively, satisfy a set of goals considered important for making significant improvement in that area. All CMMI process areas are common to both continuous and staged representations.

5.5

A. Software security -

Security testing is very important to keep the system protected from malicious activities on the web.

What is Security Testing?

Security testing is a testing technique to determine if an information system protects data and maintains functionality as intended. Security testing does not guarantee complete security of the system, but it is important to include security testing as a part of the testing process.

Security testing takes the following six measures to provide a secured environment –

- **Confidentiality** – It protects against disclosure of information to unintended recipients.
- **Integrity** – It allows transferring accurate and correct desired information from senders to intended receivers.
- **Authentication** – It verifies and confirms the identity of the user.
- **Authorization** – It specifies access rights to the users and resources.
- **Availability** – It ensures readiness of the information on requirement.
- **Non-repudiation** – It ensures there is no denial from the sender or the receiver for having sent or received the message.

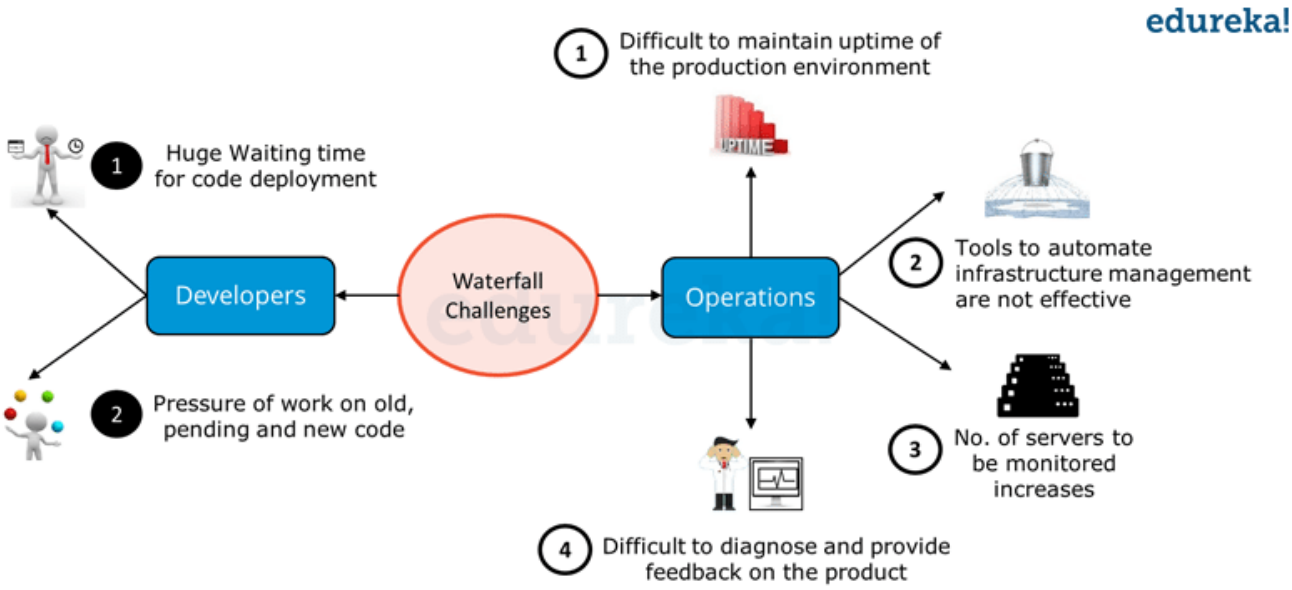
Example

Spotting a security flaw in a web-based application involves complex steps and creative thinking. At times, a simple test can expose the most severe security risk. You can try this very basic test on any web application –

- Log into the web application using valid credentials.
- Log out of the web application.
- Click the BACK button of the browser.
- Verify if you are asked to log in again or if you are able go back to the logged in page again.

Waterfall Model Challenges

The Water-fall model worked fine and served well for many years however it had some challenges. In the following diagram the challenges of Waterfall Model are highlighted.



In the above diagram you can see that both Development and Operations had challenges in the Waterfall Model. From Developers point of view there were majorly two challenges:

1 After Development, the code deployment time was huge.

2 Pressure of work on old, pending and new code was high because development and deployment time was high.

On the other hand, Operations was also not completely satisfied. There were four major challenges they faced as per the above diagram:

1 It was difficult to maintain ~100% uptime of the production environment.

2 Infrastructure Automation tools were not very affective.

3 Number of severs to be monitored keeps on increasing with time and hence the complexity.

4 It was very difficult to provide feedback and diagnose issue in the product.

DevOps integrates developers and operations team to improve collaboration and productivity.

According to the DevOps culture, a single group of Engineers (developers, system admins, QA's. Testers etc turned into DevOps Engineers) has end to end responsibility of the Application (Software) right from gathering the requirement to development, to testing, to infrastructure deployment, to application deployment and finally monitoring & gathering feedback from the end users, then again implementing the changes.

This is a never ending cycle and the logo of DevOps makes perfect sense to me. Just look at the above diagram – What could have been a better symbol than infinity to symbolize DevOps?

Now let us see how DevOps takes care of the challenges faced by Development and Operations. Below table describes how DevOps addresses Dev Challenges.

edureka!	Dev Challenges	DevOps Solution
	Waiting time for code deployment	• Continuous Integration ensures there is quick deployment of code, faster testing and speedy feedback mechanism • Thus there is no waiting time to deploy the code. Hence the developer focuses on building the current code
	Pressure of work on old, pending and new code	

DevOps Tutorial Table 1 – Above table states how DevOps solves Dev Challenges

Going further, below table describes how DevOps addresses Ops Challenges.

	Ops Challenges	DevOps Solution
	Difficult to maintain uptime of the production environment	Containerization / Virtualization ensures there is a simulated environment created to run the software as containers offer great reliability for service uptime
	Tools to automate infrastructure management are not effective	Configuration Management helps you to organize and execute configuration plans, consistently provision the system, and proactively manage their infrastructure
	No. of servers to be monitored increases	Continuous Monitoring Effective monitoring and feedbacks system is established through Nagios Thus effective administration is assured edureka!
	Difficult to diagnose and provide feedback on the product	

DevOps Tutorial Table 2 – Above table states how DevOps solves Ops Challenges

However, you would still be wondering, how to implement DevOps. To expedite and actualize DevOps process apart from culturally accepting it, one also needs various DevOps tools like Puppet, Jenkins, GIT, Chef, Docker, Selenium, AWS etc to achieve automation at various stages which helps in achieving Continuous Development, Continuous Integration, Continuous Testing, Continuous Deployment, Continuous Monitoring to deliver a quality software to the customer at a very fast pace.

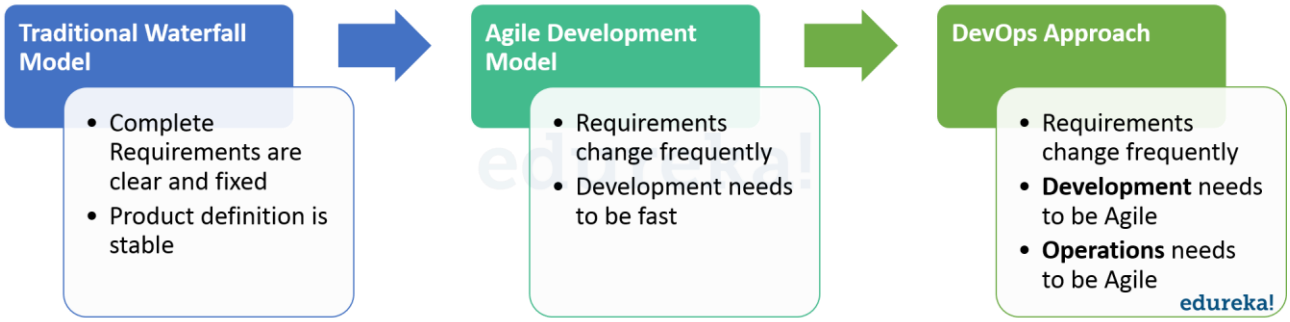
DevOps Lifecycle can be broadly broken down into the below DevOps Stages:

- Continuous Development
- Continuous Integration
- Continuous Testing
- Continuous Monitoring
- Virtualization and Containerization

These stages are the building blocks to achieve DevOps as a whole.

Evolution of Software Development

DevOps evolved from existing software development strategies/ methodologies over the years in response to business needs. Let us briefly look at how these models evolved and in which scenarios they would work best.



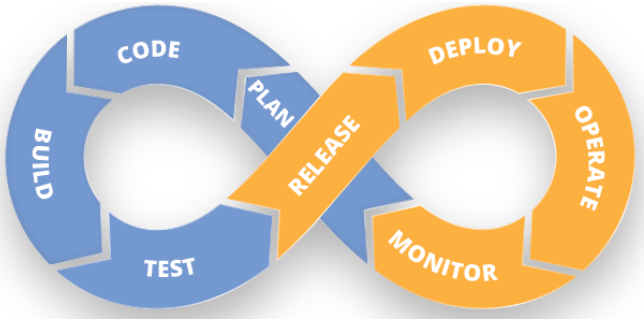
The slow and cumbersome Waterfall model evolved into Agile which saw development teams working on the software in short sprints lasting not more than two weeks. Having such a short release cycle helped the development team work on client feedback and incorporate it along with bug fixes in the next release. While this Agile SCRUM approach brought agility to development, it was lost on Operations which did not come up to speed with Agile practices. Lack of collaboration between Developers and Operations Engineers still slowed down the development process and releases. DevOps methodology was born out of this need for better collaboration and faster delivery. DevOps enables continuous software delivery with less complex problems to fix and faster resolution of problems.

Now that we have understood the evolution of DevOps, let us look at what is DevOps in detail.

What is DevOps?

DevOps is a software development approach which involves Continuous Development, Continuous Testing, Continuous Integration, Continuous Deployment and Continuous Monitoring of the software throughout its development life cycle. These activities are possible only in DevOps, not Agile or waterfall, and this is why Facebook and other top companies have chosen DevOps as the way forward for their business goals. DevOps is the preferred approach to develop high quality software in shorter development cycles which results in greater customer satisfaction. Check out the below video on What is DevOps before you go ahead.

DevOps life cycle



Continuous Development:

This is the stage in the DevOps life cycle where the Software is developed continuously. Unlike the Waterfall model, the software deliverables are broken down into multiple sprints of short development cycles, developed and then delivered in a very short time. This stage involves the Coding and Building phases and makes use of tools such as **Git** and **SVN** for maintaining the different versions of the code, and tools like **Ant**, **Maven**, **Gradle** for building/ packaging the code into an executable file that can be forwarded to the QAs for testing.

Continuous Testing:

This is the stage where the developed software is continuously tested for bugs. For Continuous testing, automation testing tools like **Selenium**, **TestNG**, **JUnit**, etc are used. These tools allow the QAs to test multiple code-bases thoroughly in parallel to ensure that there are no flaws in the functionality. In this phase, use of **Docker containers** for simulating 'test environment' on the fly, is also a preferred choice. Once the code is tested, it is continuously integrated with the existing code.

Continuous Integration:

This is the stage where the code supporting new functionality is integrated with the existing code. Since there is continuous development of software, the updated code needs to be integrated continuously as well as smoothly with the systems to reflect changes to the end users. The changed code, should also ensure that there are no errors in the runtime environment, allowing us to test the changes and check how it reacts with other changes.

Jenkins is a very popular tool used for Continuous Integration. Using Jenkins, one can pull the latest code revision from GIT repository and produce a build which can finally be deployed to test or production server. It can be set to trigger a new build automatically as soon as there is a change in the GIT repository or can be triggered manually on click of a button.

Continuous Deployment:

It is the stage where the code is deployed to the production environment. Here we ensure that the code is correctly deployed on all the servers. If there is any addition of functionality or a new feature is introduced then one should be ready to welcome greater website traffic. So it is also the responsibility of the SysAdmin to scale up the servers to host more users.

Since the new code is deployed on a continuous basis, configuration management tools play an important role for executing tasks quickly and frequently. **Puppet**, **Chef**, **SaltStack** and **Ansible** are some popular tools that are used in this stage.

Containerization tools also play an important role in the deployment stage. **Docker** and **Vagrant** are the popular tools which help produce consistency across Development, Test, Staging and Production environments. Besides this, they also help in scaling-up and scaling-down of instances easily.

Continuous Monitoring:

This is a very crucial stage in the DevOps life cycle which is aimed at improving the quality of the software by monitoring its performance. This practice involves the participation of the Operations team who will monitor the user activity for bugs / any improper behavior of the system. This can also be achieved by making use of dedicated monitoring tools which will continuously monitor the application performance and highlight issues.

Some popular tools used are **Splunk**, **ELK Stack**, **Nagios**, **NewRelic** and **Sensu**. These tools help you monitor the application and the servers closely to check the health of the system proactively. They can also improve productivity and increase the reliability of the systems, reducing IT support costs. Any major issues found could be reported to the development team so that it can be fixed in the continuous development phase.

These DevOps stages are carried out on loop continuously until the desired product quality is achieved.