Need of Software Engineering

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

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

[**What are Various Software Engineering Problems? Explain**](http://ecomputernotes.com/software-engineering/what-are-various-software-engineering-problems-explain)

Software engineering is the systematic approach to the development, operation, maintenance, and retirement of software. There are few fundamental problems that software engineering faces.

**The Problem of scale**:    A fundamental problem of software engineering is the problem of scale; development of a very large system requires a very different set of methods compared to developing a small system. In other words, the methods that are used for developing small systems generally do not scale up to large systems. A different set of methods has to be used for developing large software. Any large project involves the use of technology and project management.

For software projects, by technology we mean the methods, procedures, and tools that are used. In small projects, informal methods for development and management can be used. However, for large projects, both have to be much more formal.

While dealing with a small software project, the technology requirement is low and the project management requirement is also low. However, when the scale changes to large systems, to solve such problems properly, it is essential that we move in both directions-the methods used for development need to be more formal, and the project management for the development project also needs to be more formal.

**Cost, schedule and quality**: The cost of developing a system is the cost of the resources used for the system, which, in the case of software, are the manpower, hardware, software, and the other support resources. Generally, the manpower component is predominant, as software development is largely labor-intensive and the cost of the computing systems is now quite low.

Hence, the cost of software project is measured in terms of person-months, i.e. the cost is considered to be the total number of person-months spent in the project. Schedule is an important factor in many projects. Business trends are dictating that the time to market of a product should be reduced; that is, the cycle time from concept to delivery should be small. Any business with such a requirement will also require that the cycle time for building a software needed by the business be small.

One of the major factors driving any production discipline is quality. We can view quality of a software product as having three dimensions:

             Product Operation  
             Product Transition  
             Product Revision

**The Problem of consistency**: Though high quality, low cost and small cycle time are the primary objectives of any project, for an organization there is another goal: consistency. An organization involved in software development does not just want low cost and high quality for a project, but it wants these consistently.

**Software Engineering - What is Software Engineering? Write Basic Objective and Need for Software Engineering**

Over the last 50 years there has been revolutionary advancement in the field of technology, leading to improvements in hardware performance and profound changes in computing architectures. This advancement has led to the production of complex computer-based systems that are capable of providing [information](http://ecomputernotes.com/fundamental/information-technology/what-do-you-mean-by-data-and-information) in a wide variety of formats. The increase in [computer](http://ecomputernotes.com/fundamental/introduction-to-computer/what-is-computer) power has made unrealistic computer applications a feasible proposition, marking the genesis of an era where software products are far more complex as compared to their predecessors. By using software engineering practices, these complex systems can be developed in a systematic and efficient manner.

According to **IEEE**, software engineering is defined as 'the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.' In anutshell, software engineering can be defined as a systematic approach to develop software within specified time and budget.

Software engineering is a technological discipline that combines the concepts of computer science, economics, communication skills, and management science with the problem-solving approach of engineering. It also involves a standardized approach to program development, both in its managerial and technical aspects.

The profound knowledge of computer science both theoretical and practical forms the basis of software engineering. The theoretical knowledge provides an understanding of which problems are resolvable, what data structures and algorithms are appropriate, when and how they are to be used, etc. On the other hand, the practical knowledge provides an understanding of how hardware functions, how to utilize the power of programming languages and tools while developing software, etc.

One of the main objectives of software engineering is to help developers obtain high quality software. This quality is achieved through use of Total Quality Management (TQM), which enables continuous process improvement custom that leads to the development of more established approaches to software engineering.

**Basic Objective**

Software engineering is the systematic approach to the development, operation, maintenance and retirement of software. Software Engineering is the application of science and mathematics by which the capabilities of computer equipment are made useful to man via computer programs, procedures, and associated documentations.

The basic objective of software engineering is to develop methods and procedures for software development that can scale up for large systems and that can be used consistently to produce high-quality software at low cost and with a small cycle of time.

**Need for Software Engineering**

1- As Software development is expensive so proper measures are required so that the resources are used efficiently and   effectively.

2- Cost and time considerations are another factor, which arises the need for Software Engineering.

3- Reliability factors

[**What are Various Software Engineering Problems? Explain**](http://ecomputernotes.com/software-engineering/what-are-various-software-engineering-problems-explain)

Software engineering is the systematic approach to the development, operation, maintenance, and retirement of software. There are few fundamental problems that software engineering faces.

**The Problem of scale**:    A fundamental problem of software engineering is the problem of scale; development of a very large system requires a very different set of methods compared to developing a small system. In other words, the methods that are used for developing small systems generally do not scale up to large systems. A different set of methods has to be used for developing large software. Any large project involves the use of technology and project management.

For software projects, by technology we mean the methods, procedures, and tools that are used. In small projects, informal methods for development and management can be used. However, for large projects, both have to be much more formal.

While dealing with a small software project, the technology requirement is low and the project management requirement is also low. However, when the scale changes to large systems, to solve such problems properly, it is essential that we move in both directions-the methods used for development need to be more formal, and the project management for the development project also needs to be more formal.

**Cost, schedule and quality**: The cost of developing a system is the cost of the resources used for the system, which, in the case of software, are the manpower, hardware, software, and the other support resources. Generally, the manpower component is predominant, as software development is largely labor-intensive and the cost of the computing systems is now quite low.

Hence, the cost of software project is measured in terms of person-months, i.e. the cost is considered to be the total number of person-months spent in the project. Schedule is an important factor in many projects. Business trends are dictating that the time to market of a product should be reduced; that is, the cycle time from concept to delivery should be small. Any business with such a requirement will also require that the cycle time for building a software needed by the business be small.

One of the major factors driving any production discipline is quality. We can view quality of a software product as having three dimensions:

             Product Operation  
             Product Transition  
             Product Revision

**The Problem of consistency**: Though high quality, low cost and small cycle time are the primary objectives of any project, for an organization there is another goal: consistency. An organization involved in software development does not just want low cost and high quality for a project, but it wants these consistently.

[**Software Crisis in Software Engineering**](http://ecomputernotes.com/software-engineering/software-crisis)

In the late 1960s, it became clear that the development of software is different from manufacturing other products. This is because employing more manpower (programmers) later in the software development does not always help speed up the development process. Instead, sometimes it may have negative impacts like delay in achieving the scheduled targets, degradation of software quality, etc. Though software has been an important element of many systems since a long time, developing software within a certain schedule and maintaining its quality is still difficult.

History has seen that delivering software after the scheduled date or with errors has caused large scale financial losses as well as inconvenience to many. Disasters such as the Y2Kproblem affected economic, political, and administrative systems of various countries around the world. This situation, where catastrophic failures have occurred, is known as **software crisis.**The major causes of software crisis are the problems associated with poor quality software such as malfunctioning of software systems, inefficient development of software, and the most important, dissatisfaction amongst the users of the software.

The software market today has a turnover of more than millions of rupees. Out of this, approximately thirty Percent of software is used for [personal computer](http://ecomputernotes.com/fundamental/introduction-to-computer/personal-computer)s and the remaining software is developed for specific users or organizations. Application areas such as the banking sector are completely dependant on software application. Software failures in these technology-oriented areas have led to considerable loss in terms of time, money, and even human lives. History has been witness to many such failures, some of which are listed below.

1. The Northeast blackout in 2003 has been one of the major power system failures in the history of North America. This blackout involved failure of 100 power plants due to which almost 50 million customers faced power loss that resulted in financia110ss of approximately $6 billion. Later, it was determined that the major reason behind the failure was a software bug in the power monitoring and management system.
2. Year 2000 (Y2K) problem refers to the widespread snags in processing dates after the year 2000. The roots ofY2K problem can be traced back to 1960-80 when developers shortened the 4-digit date format like 1972 to a 2-digit format like 72 because of limited memory. At that time they did not realize that year 2000 will be shortened to 00 which is less than 72. In the 1990s, experts began to realize this major shortcoming in the [computer](http://ecomputernotes.com/fundamental/introduction-to-computer/what-is-computer) application and then millions were spent to handle this problem.
3. In 1996, Arian-5 space rocket, developed at the cost of $7000 million over a period of 10 years was destroyed within less than a minute after its launch. The crash occurred because there was a software bug in the rocket guidance system.
4. In 1996, one of the largest banks of US credited accounts of nearly 800 customers with approximately $9241acs. Later, it was detected that the problem occurred due to a programming bug in the banking software.
5. During the Gulf War in 1991, the United States of America used Patriot missiles as a defense against Iraqi Scud missiles. However, the Patriot failed to hit the Scud many times. As a result, 28 US soldiers were killed in Dhahran, Saudi Arabia. An inquiry into the incident concluded that a small bug had resulted in the miscalculation of missile path.

**Definition of 'Software Engineering'**

**Definition:**Software engineering is a detailed study of engineering to the design, development and maintenance of software. Software engineering was introduced to address the issues of low-quality software projects. Problems arise when a software generally exceeds timelines, budgets, and reduced levels of quality. It ensures that the application is built consistently, correctly, on time and on budget and within requirements. The demand of software engineering also emerged to cater to the immense rate of change in user requirements and environment on which application is supposed to be working.   
  
**Description:**A software product is judged by how easily it can be used by the end-user and the features it offers to the user. An application must score in the following areas:-   
  
1) Operational: -This tells how good a software works on operations like budget , usability, efficiency, correctness ,functionality , dependability , security and safety.   
  
2) Transitional: - Transitional is important when an application is shifted from one platform to another. So, portability, reusability and adaptability come in this area.   
  
3) Maintenance: - This specifies how good a software works in the changing environment. Modularity, maintainability, flexibility and scalability come in maintenance part.   
  
Software Development Lifecycle or SDLC is a series of stages in software engineering to develop proposed software application, such as:   
1) Communication   
2) Requirement Gathering   
3) Feasibility Study   
4) System Analysis   
5) Software Design   
6) Coding   
7) Testing   
8) Integration   
9) Implementation   
10) Operations and maintenance   
11) Disposition   
  
Software engineering generally begins with the first step as a user-request initiation for a specific task or an output. He submits his requirement to a service provider organization. The software development team segregates user requirement, system requirement and functional requirements. The requirement is collected by conducting interviews of a user, referring to a database, studying the existing system etc. After requirement gathering, the team analyses if the software can be made to fulfil all the requirements of the user. The developer then decides a roadmap of his plan. System analysis also includes an understanding of software product limitations. As per the requirement and analysis, a software design is made. The implementation of software design starts in terms of writing program code in a suitable programming language. Software testing is done while coding by the developers and thorough testing is conducted by testing experts at various levels of code such as module testing, program testing, product testing, in-house testing and testing the product at user’s engagement and feedback.

**What is software engineering?**

It is an **Engineering branch** associated with the development of software product using well-defined **scientific principles, method, and procedure.** The outcome of software engineering is an efficient and reliable software product.

**So now in brief:**

**Engineering forces us to focus on Systematic, scientific and well defined processes to produce "A Good Quality Product."**

The application of a **Systematic, Disciplined, Quantifiable** approach, to the development, operation and maintenance of software, and the study of these approaches, that is the application of Engineering to software.

# What is software process? What are the 4 activities of software process framework?

### Software Process:

Software Process is a set of required activities and the outcome of the activities with a target to produce a software product. A software process is a flowchart of developing a software product, which includes fathering requirements, analyzing those requirements, scheduling development phases, checking the developments, implementing changes etc. and this can be till the delivery of the final software product to the after delivery service methods and more.

## Following are 4 activities which are common to all software processes.

### Software Specification:

The customer (if any) and the engineers gathers and analyze the features, workflow, operational constraints or limitations of the final software product. This part is common in every all software process despite of how big or small, simple or complex the software product is.   
  
Software Design & Implementation:

After all the specifications, goals for the software product are fixed, engineers starts developing the software which not only includes coding but also gathering required artworks, audio and visual elements for the software product.

**Software Validation:**

Software product must be checked for existing bugs, incomplete or unavailable features etc. However, software validation can be done after software development in case of small software, but can also done multiple times (milestones) during the software development phase.

### Software Evolution:

A software product can take years to being developed. In the meantime, specifications, feature requirements etc. can be changed or added or can be required to remove from the software product. Thus this is where the software will be modified to accept those changes.

There exists various software process model which presents different approaches to software product development which depends on the software that has to be developed.

[**Explain Software Process & Characteristics**](http://ecomputernotes.com/software-engineering/explain-software-process-characteristics)

The process that deals with the technical and management issues of software development is called a software process. A software development project must have at least development activities and project management activities. The fundamental objectives of a process are the same as that of software engineering (after all, the process is the main vehicle of satisfying the software engineering objectives), viz. optimality and scalability.

Optimality means that the process should be able to produce high-quality software at low cost, and scalability means that it should also be applicable for large software projects. To achieve these objectives, a process should have some properties. Predictability of a process determines how accurately the outcome of following a process in a project can be predicted before the project is completed. Predictability can be considered a fundamental property of any process, In fact, if a process is not predictable, it is of limited use.

One of the important objectives of the development project should be to produce software that is easy to maintain. And the process should be such that it ensures this maintainability. Testing consumes the most resources during development. Underestimating the testing effort often causes the planners to allocate insufficient resources for testing, which, in turn, results in unreliable software or schedule slippage.

The goal of the process should not be to reduce the effort of design and coding, but to reduce the cost of maintenance. Both testing and maintenance depend heavily on the design and coding of software, and these costs can be considerably reduced if the software is designed and coded to make testing and maintenance easier. Hence, during the early phases of the development process the prime issues should be "can it be easily tested" and "can it be easily modified". Errors can occur at any stage during development.

However error detection and correction should be a continuous process that is done throughout software development. Detecting errors soon after they have been introduced is clearly an objective that should be supported by the process. A process is also not a static entity.

As the productivity (and hence the cost of a project) and quality are determined largely by the process to satisfy the engineering objectives of quality improvement and cost reduction, the software process must be improved. Having process improvement as a basic goal of the software process implies that the software process used is such that is supports its improvement.

[**What is SRS?List & Describe Various Characteristics of an SRS**](http://ecomputernotes.com/software-engineering/what-is-srslist-describe-various-characteristics-of-an-srs)

Software requirement specification (SRS) is a document that completely describes what the proposed software should do without describing how software will do it. The basic goal of the requirement phase is to produce the SRS, Which describes the complete behavior of the proposed software. SRS is also helping the clients to understand their own needs.

**Advantages**

Software SRS establishes the basic for agreement between the client and the supplier on what the software product will do.

• A SRS provides a reference for validation of the final product.  
• A high-quality SRS is a prerequisite to high-quality software.  
• A high-quality SRS reduces the development cost.

**Characteristics of an SRS**

• Correct  
• Complete  
• Unambiguous  
• Verifiable  
• Consistent  
• Ranked for importance and/or stability  
• Modifiable  
• Traceable

An SRS is **correct** if every requirement included in the SRS represents something required in the final system. An SRS is complete, if everything the software is supposed to do and the responses of the software to all classes of input data are specified in the SRS. Correctness ensures that what is specified is done correctly, **completeness** ensures that everything is indeed specified.

An SRS is **unambiguous** if and only if every requirement stated has one and only one interpretation. Requirements are often written in natural language, which are inherently ambiguous.

An SRS is **verifiable** if and only if every stated requirement is verifiable. A requirement is verifiable if there exists some cost-effective process that can check whether the final software meets that requirement. An SRS is consistent if there is no requirement that conflicts with another.

Terminology can cause **inconsistencies**; for example, different requirements may use different terms to refer to the same object. All the requirements for software are not of equal importance. Some are critical, others are important but not critical, and there are some, which are desirable, but not very important. An SRS is ranked for importance and the stability of the requirement are indicated. Stability of requirement reflects the chances of it changing in future. An SRS is traceable if the origin of each of its requirements is clear and if it facilitates the referencing of each requirement in future development. Forward traceability means that each requirement should be traceable to some design and code elements. Backward traceability requires that it be possible to trace design and code elements to the requirements they support. Traceability aids verification and validation.

[**List & Explain Various Components of an SRS**](http://ecomputernotes.com/software-engineering/list-explain-various-components-of-an-srs)

Completeness of specifications is difficult to achieve and even more difficult to verify. Having guidelines about what different things an SRS should specify will help in completely specifying the requirements. Here we describe some of the system properties than an SRS should specify.

**The basic issues an SRS must address**

1.       Functionality

2.       Performance

3.       Design constraints imposed on an implementation

4.       External interfaces

**Functional Requirements**

1.       Which outputs should be produced from the given inputs?

2.       Relationship between the input and output.

3.       A detailed description of all the data inputs and their source, the units of measure.

4.       The range of valid inputs.

**Design Constraints**

1.       Standards that must be followed.

2.       Resource limits & operating environment.

3.       Reliability

4.       Security requirement

5.       Policies that may have an impact on the design of the system.

**Standards Compliance:**

This specifies the requirements for the standards that the system must follow.

**Hardware Limitations:**

 The software may have to operate on some existing or predetermined hardware thus imposing restrictions on the design.

Reliability and Fault Tolerance:

Fault tolerance requirements can place a major constraint on how the system is to be designed. Fault tolerance requirements often make the system more complex and expensive.

**Security:**

Security requirements are particularly significant in defense systems and many [database](http://ecomputernotes.com/fundamental/what-is-a-database/advantages-and-disadvantages-of-dbms) systems. Security requirements place restrictions on the use of certain commands, control access to data, provide different kinds of access requirements for different people require the use of passwords and cryptography techniques and maintain a log of activities in the system.

Since the requirements document serves as a foundation for subsequent software development phases, it is important to develop the document in the prescribed manner. For this, certain guidelines are followed while preparing SRS. These guidelines are listed below.

1. **Functionality:**It should be separate from implementation.
2. **Analysis model:**It should be developed according to the desired behavior of a system. This should include data and functional response of a system to various inputs given to it.
3. **Cognitive model:**It should be developed independently of design or implementation model. This model expresses a system as perceived by the users.
4. **The content and structure**of the **specification:**It should be flexible enough to accommodate changes.
5. **Specification:**It should be robust. That is, it should be tolerant towards incompleteness and complexity.

**External Interface Requirements:**

All the possible interactions of the software with people, hardware and other software should be clearly specified. For the user interface, the characteristics of each user interface of the software product should be specified. User interface is becoming increasingly important and must be given proper attention. A preliminary user manual should be created with all use commands, screen formats and explanation of how the system will appear to the user, and feedback and error message.

Like other specifications these requirements should be precise and verifiable. So a statement likes “the system should be no longer than six characters” or command names should reflect the function they perform used. If the software is to execute on existing hardware or on predetermined hardware, all the characteristics of the hardware, including memory restrictions, should be specified. In addition, the current use and load characteristics of the hardware should be given.

A Software Requirements Specification (SRS) is a document that describes the nature of a project, software or application. In simple words, SRS document is a manual of a project provided it is prepared before you kick-start a project/application. This document is also known by the names SRS report, software document. A software document is primarily prepared for a project, software or any kind of application.

There are a set of guidelines to be followed while preparing the software requirement specification document. This includes the purpose, scope, functional and nonfunctional requirements, software and hardware requirements of the project. In addition to this, it also contains the information about environmental conditions required, safety and security requirements, software quality attributes of the project etc.

**What is a Software Requirements Specification document?**

A Software requirements specification document describes the intended purpose, requirements and nature of a software to be developed. It also includes the yield and cost of the software.

In this document, flight management project is used as an example to explain few points.

[**Explain Software Configuration Management Process**](http://ecomputernotes.com/software-engineering/explain-software-configuration-management-process)

Throughout development, software consists of a collection of items (such as programs, data and documents) that can easily be changed. During software development, the design, code, and even requirements are often changed, and the changes occur at any time during the development. This easily changeable nature of software and the fact that changes often take place require that changes be done in a controlled manner.

Software configuration management (SCM) is the discipline for systematically controlling the changes that take place during development. Software configuration management is a process independent of the development process largely because most development models cannot accommodate change at any time during development. SCM can be considered as having three major components:

    Software configuration identification  
    Change control  
    Status accounting and auditing

**Configuration identification:**

 The first requirement for any change management is to have clearly agreed-on basis for change. That is, when a change is done, it should be clear to what changes has been applied. This requires baselines to be established. A baseline change is the changing of the established baseline, which is controlled by SCM.

After baseline changes the state of the software is defined by the most recent baseline and the changes that were made. Some of the common baselines are functional or requirements baseline, design baseline, and product or system baseline. Functional or requirement baseline is generally the requirements document that specifies the functional requirements for the software. Design baseline consists of the different components in the software and their designs. Product or system baseline represents the developed system.

It should be clear that a baseline is established only after the product is relatively stable. Though the goal of SCM is to control the establishment and changes to these baselines, treating each baseline as a single unit for the purpose of change is undesirable, as the change may be limited to a very small portion of the baseline.

**Change control:**

Most of the decisions regarding the change are generally taken by the configuration control board (CCB), which is a group of people responsible for configuration management, headed by the configuration manager. For smaller projects, the CCB might consist of just one person. A change is initiated by a change request.

The reason for change can be anything. However, the most common reasons are requirement changes, changes due to bugs, platform changes, and enhancement changes. The CR for change generally consists of three parts. The first part describes the change, reason for change, the SCIs that are affected, the priority of the change, etc.

The second part, filled by the CM, describes the decision taken by the CCB on this CR, the action the CM feels need to be done to implement this change and any other comments the CM may have. The third part is filled by the implementer, which later implements the change.

**Status accounting and auditing:**

For status accounting, the main source of [information](http://ecomputernotes.com/fundamental/information-technology/what-do-you-mean-by-data-and-information) is the CRs and FRs themselves. Generally, a field in the CR/FR is added that specifies its current status. The status could be active, complete, or not scheduled. Information about dates and efforts can also be added to the CR, the information from the CRs/FRs can be used to prepare a summary, which can be used by the project manager and the CCB to track all the changes.

[**What is Risk Management? Give Brief Ideas for Risk Assessment and Control**](http://ecomputernotes.com/software-engineering/what-is-risk-management-give-brief-ideas-for-risk-assessment-and-control)

Any large project involves certain risks, and that is true for software projects. Risk management is an emerging area that aims to address the problem of identifying and managing the risks associated with a software project.

 Risk is a project of the possibility that the defined goals are not met. The basic motivation of having risk management is to avoid disasters and heavy losses. The current interest in risk management is due to the fact that the history of software development projects is full of major and minor failures. A large percentage of projects have run considerably over budget and behind schedule, and many of these have been abandoned midway. It is now argued that many of these failures were due to the fact that the risks were not identified and managed properly.

Risk management is an important area, particularly for large projects. Like any management activity, proper planning of that activity is central to success. Here we discuss various aspects of risk management and planning.

**Risk Management Overview**

Risk is defined as an exposure to the chance of injury of loss (Kon94]. That is, risk implies that there is a possibility that negative may happen. In the context of software projects, negative implies that here is an adverse effect on cost, quality, or schedule. Risk management is the area that tries to ensure that the impact of risks on cost, quality, and schedule is minimum.

Like configuration management, which minimizes the impact of change, risk management minimizes the impact of risks. However, risk management is generally done by the project management. For this reason we have not considered risk management as a separate process   (through it can validly be considered one) but have considered such activities as part of project management.

Risk management can be considered as dealing with the possibility and actual occurrence of those events that are not “regular” or commonly expected. Normally project management handles the commonly expected events, such as people going on leave or some requirements changing. It deals with events that are infrequent, somewhat out of the control of the project management, and are large enough (i.e. can have a major impact on the project) to justify special attention.

# [When is Cost Estimation Done? Discuss the COCOMO Model along with the Parameters Defined in it](http://ecomputernotes.com/software-engineering/when-is-cost-estimation-done-discuss-the-cocomo-model-along-with-the-parameters-defined-in-it)

Any cost estimation model can be viewed as a function that outputs the cost estimate. The basic idea of having a model or procedure for cost estimation is that it reduces the problem of estimation of determining the value of he “key parameters” that characterize the project, based on which the cost can be estimated.

 The primary factor that controls the cost is the size of the project. That is, the larger the project, the greater the cost & resource requirement. Other factors that affect the cost include programmer ability, experience of developers, complexity of the project, & reliability requirements.

The goal of a cost model is to determine which of these many parameters have significant effect on cost & then to discover the relationships between the cost. The most common approach for estimating effort is to make a function of a single variable. Often this variable is the project size, & the equation of efforts is:

EFFORT = a x size b

Where a & b are constants.

If the size estimate is in KDLOC, the total effort, E, in person-months can be given by the equation.

          E = 5.2 (KDLOC) 91

## On Size Estimation

Though the single variable cost models with size as the independent variable result in simple models that can be easily obtained, applying them for estimation is not simple. The reason is that these models now require size as the input, & size of the project is not known early in development & has to be estimated.

For estimating the size, the system is generally partitioned into components it is likely to have. Once size estimates for components are available, to get the overall size estimate for the system, the estimates for all the components can be added up. Similar property does not hold for cost estimation, as cost of developing a system is not the sum of costs of developing the components. With the size-based models, if the size estimate is inaccurate, the cost estimates produced by the models will also be inaccurate.

## COCOMO Model

The Constructive cost model (COCOMO) was developed by Boehm. This model also estimates the total effort in terms of person-months of the technical project staff. The effort estimate includes development, management, and support tasks but does not include the cost of the secretarial and other staff that might be needed in an organization. The basic steps in this model are: -

1. Obtain an initial estimate of the development effort from the estimate of thousands of delivered lines of source code (KDLOC).

2. Determine a set of 15 multiplying factors from different attributes of the project.

3. Adjust the effort estimate by multiplying the initial estimate with all the multiplying factors.

The initial estimate is determined by an equation of the form used in the static single – variable models, using KDLOC as the measures of size. To determine the initial effort Ei in person-months the equation used is of the type

Ei = a \* (KDLOC)b.  
The value of the constants a and b depend on the project type. In COCOMO, projects are categorized into three types – organic, semidetached, and embedded.

Organic projects are in an area in which the organization has considerable experience and requirements are less stringent. A small team usually develops such systems. Examples of this type of project are simple business systems, simple inventory management systems, and data processing systems.

**Estimation techniques**

There is no simple way to make an accurate estimate of the effort required to develop a software system. You may have to make initial estimates on the basis of a high level user requirements definition. The software may have to run on unfamiliar computers or use new development technology. The people involved in the project and their skills will probably not be known. All of these mean that it is impossible to estimate system development costs accurately at an early stage in a project.

Furthermore, there is a fundamental difficulty in assessing the accuracy of different approaches to cost-estimation techniques. Project cost estimates are often self fulfilling. The estimate is used to define the project budget, and the product is adjusted so that the budget figure is realised. I do not know of any controlled experiments with project costing where the estimated costs were not used to bias the experiment. A controlled experiment would not reveal the cost estimate to the project manager. The actual costs would then be compared with the estimated project costs. However, such an experiment is probably impossible because of the high costs involved and the number of variables that cannot be controlled.

Nevertheless, organisations need to make software effort and cost estimates. To do so, one or more of the techniques described in Figure 26.4 may be used (Boehm, 1981). All of these techniques rely on experience-based judgements by project managers who use their knowledge of previous projects to arrive at an estimate of the resources required for the project. However, there may be important differences between past and future projects. Many new development methods and techniques have been introduced in the last 10 years. Some examples of the changes that may affect estimates based on experience include:

1. Distributed object systems rather than mainframe-based systems

2. Use of web services

3. Use of ERP or database-centred systems

4. Use of off-the-shelf software rather than original system development

5. Development for and with reuse rather than new development of all parts of a system

|  |  |
| --- | --- |
|  |  |
| Algorithmic cost.  modelling | A model is developed using historical cost information that relates some software metric (usually its size) to the project cost. An estimate is made of that metric and the model predicts the effort required |
| Expert judgement | Several experts on the proposed software development techniques and the application domain are consulted. They each estimate the project cost. These estimates are compared and discussed. The estimation process iterates until an agreed estimate is reached. |
| Estimation by.  analogy | This technique is applicable when other projects in the same application domain have been completed. The cost of a new project is estimated by analogy with these completed projects. Myers (Myers, 1989) gives a very clear description of this approach |
| Parkinson’s Law | Parkinson’s Law states that work expands to fill the time available. The cost is determined by available resources rather than by objective assessment. If the software has to be delivered in 12 months and 5 people are available, the effort required is estimated to be 60 person-months. |
| Pricing to win | The software cost is estimated to be whatever the customer has available to spend on the project. The estimated effort depends on the customer’s budget and not on the software functionality. |

6. Development using scripting languages such as TCL or Perl (Ousterhout, 1998)

7. The use of CASE tools and program generators rather than unsupported software development.

If project managers have not worked with these techniques, their previous experience may not help them estimate software project costs. This makes it more difficult for them to produce accurate costs and schedule estimates.

You can tackle the approaches to cost estimation shown in Figure 26.4 using either a top-down or a bottom-up approach. A top-down approach starts at the system level. You start by examining the overall functionality of the product and how that functionality is provided by interacting sub-functions. The costs of system-level activities such as integration, configuration management and documentation are taken into account.

The bottom-up approach, by contrast, starts at the component level. The system is decomposed into components, and you estimate the effort required to develop each of these components. You then add these component costs to compute the effort required for the whole system development.

The disadvantages of the top-down approach are the advantages of the bottom-up approach and vice versa. Top-down estimation can underestimate the costs of solving difficult technical problems associated with specific components such as interfaces to nonstandard hardware. There is no detailed justification of the estimate that is produced. By contrast, bottom-up estimation produces such a justification and considers each component. However, this approach is more likely to underestimate the costs of system activities such as integration. Bottom-up estimation is also more expensive. There must be an initial system design to identify the components to be costed.

Each estimation technique has its own strengths and weaknesses. Each uses different information about the project and the development team, so if you use a single model and this information is not accurate, your final estimate will be wrong. For large projects, therefore, you should use several cost estimation techniques and compare their results. If these predict radically different costs, you probably do not have enough information about the product or the development process. You should look for more information about the product, process or team and repeat the costing process until the estimates converge.

These estimation techniques are applicable where a requirements document for the system has been produced. This should define all users and system requirements. You can therefore make a reasonable estimate of the system functionality that is to be developed. In general, large systems engineering projects will have such a requirements document.

However, in many cases, the costs of many projects must be estimated using only incomplete user requirements for the system. This means that the estimators have very little information with which to work. Requirements analysis and specification is expensive, and the managers in a company may need an initial cost estimate for the system before they can have a budget approved to develop more detailed requirements or a system prototype.

Under these circumstances, “pricing to win” is a commonly used strategy. The notion of pricing to win may seem unethical and unbusinesslike. However, it does have some advantages. A project cost is agreed on the basis of an outline proposal. Negotiations then take place between client and customer to establish the detailed project specification. This specification is constrained by the agreed cost. The buyer and seller must agree on what is acceptable system functionality. The fixed factor in many projects is not the project requirements but the cost. The requirements may be changed so that the cost is not exceeded.

For example, say a company is bidding for a contract to develop a new fuel delivery system for an oil company that schedules deliveries of fuel to its service stations. There is no detailed requirements document for this system so the developers estimate that a price of $900,000 is likely to be competitive and within the oil company’s budget. After they are granted the contract, they negotiate the detailed requirements of the system so that basic functionality is delivered; then they estimate the additional costs for other requirements. The oil company does not necessarily lose here because it has awarded the contract to a company that it can trust. The additional requirements may be funded from a future budget, so that the oil company’s budgeting is not disrupted by a very high initial software cost.

## ****COCOMO Models****

**It is**a heuristic estimation technique. This is also known as constructive Got model. Software development project can be classified into one of the three categories based on the development complexity i.e., organic, semidetached and embedded.

### ****Organic****

We can consider a development project to be of organic type, if the project deals with developing a well understood’ application program. The size of the development team is reasonably small and the team members are experienced in developing similar types of project.

### ****Semidetached****

If the development team consists of a mixture of experienced and unexperienced staff. Team members may have limited experience on related system but may be unfamiliar with some aspects of the system being developed.

### ****Embedded****

We apply this approach, if the software being developed is strongly coupled to complex hardware or if stringent regulations on the operational procedure exist.

According to Boehm, software cost estimation should be done through three   
stages i.e., basic COCOMO, intermediate COCOMO and complete COCOMO.

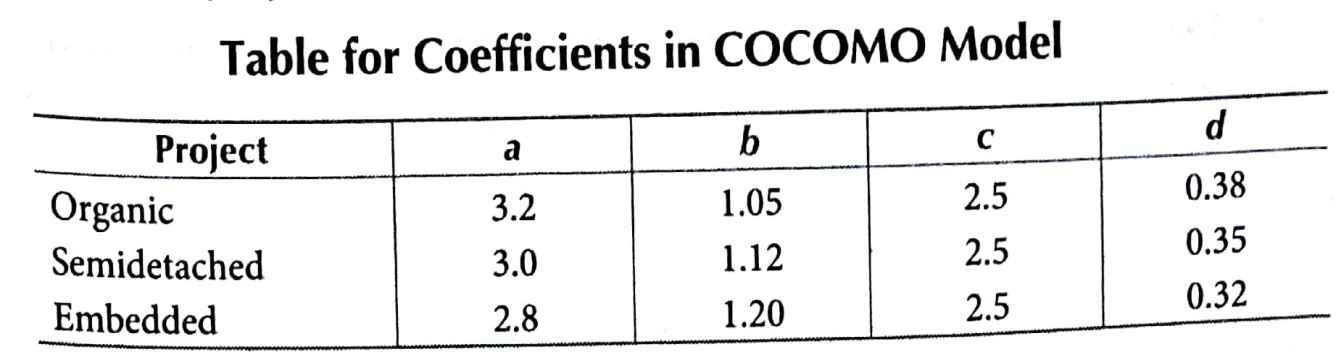
## ****Basic COCOMO Model****

Barry Boehm introduced a hierarchy of software estimation models named Constructive Cost Model. Basic equations of COCOMO model are

**Efforts in person-months (E) = a (KLOC)b**

**Development time in months (D) = c (E)d**

Where a, b, c and d are coefficients that have fixed values for different classes of projects.



**Constructive Cost Model (COCOMO)**

* COCOMO is one of the most widely used software estimation models in the world.
* This model is developed in 1981 by Barry Boehm to give estimation of number of man-months it will take to develop a software product.
* COCOMO predicts the efforts and schedule of software product based on size of software.
* COCOMO has three different models that reflect complexity

1. Basic Model
2. Intermediate Model
3. Detailed Model

* Similarly, there are three classes of software projects.

1. Organic mode In this mode, relatively simple, small software projects with a small team are handled. Such team should have good application experience to less rigid requirements.
2. Semi-detached projects In this class intermediate project in which team with mixed experience level are handled. Such project may have mix of rigid and less than rigid requirements.
3. Embedded projects In this class, project with tight hardware, software and operational constraints are handled.

* Each Model in detail

**1. Basic Model**

The basic COCOMO model estimate the software development effort using only Lines of code

Various equations in this model are

E=abKLOCbb

D=cbEdb

Where, E is the effort applied in person-months,

D is the development time in chronological months and

KLOC is the estimated number of delivered lines of code for the project

**2. Intermediate Model**

This is extension of COCOMO model.

This estimation model makes use of set of “Cost Driver Attributes” to compute the cost of software.

**I. Product attributes**

a. required software reliability

b. size of application data base

c. complexity of the product

**II. Hardware attributes**

a. run-time performance constraints

b. memory constraints

c. volatility of the virtual machine environment

d. required turnaround time

**III. Personnel attributes**

a. analyst capability

b. software engineer capability

c. applications experience

d. virtual machine experience

e. programming language experience

**IV. Project attributes**

a. use of software tools

b. application of software engineering methods

c. required development schedule

Each of the 15 attributes is rated on a 6 point scale that ranges from "very low" to "extra high" (in importance or value).

The intermediate COCOMO model takes the form

E=aiKLOCbi×EAF

Where, E is the effort applied in person-months and

KLOC is the estimated number of delivered lines of code for the project

**3. Detailed COCOMO Model**

The detailed model uses the same equation for estimation as the intermediate Model.

But detailed model can estimate the effort (E), duration (D), and person (P) of each of development phases, subsystem and models.

II copy

### What is COCOMO? Explain COCOMO model in detail?

The Constructive Cost Model (COCOMO) is an algorithmic software cost estimation model developed by Barry Boehm. The model uses a basic regression formula, with parameters that are derived from historical project data and current project characteristics.

COCOMO was first published in 1981 Barry W. Boehm's Book Software engineering economics[1] as a model for estimating effort, cost, and schedule for software projects. It drew on a study of 63 projects at TRW Aerospace where Barry Boehm was Director of Software Research and Technology in 1981. The study examined projects ranging in size from 2,000 to 100,000 lines of code, and programming languages ranging from assembly to PL/I. These projects were based on the waterfall model of software development which was the prevalent software development process in 1981.

References to this model typically call it COCOMO 81. In 1997 COCOMO II was developed and finally published in 2000 in the book Software Cost Estimation with COCOMO II[2]. COCOMO II is the successor of COCOMO 81 and is better suited for estimating modern software development projects. It provides more support for modern software development processes and an updated project database. The need for the new model came as software development technology moved from mainframe and overnight batch processing to desktop development, code reusability and the use of off-the-shelf software components. This article refers to COCOMO 81.

 COCOMO consists of a hierarchy of three increasingly detailed and accurate forms. The first level, Basic COCOMO is good for quick, early, rough order of magnitude estimates of software costs, but its accuracy is limited due to its lack of factors to account for difference in project attributes (Cost Drivers). Intermediate COCOMO takes these Cost Drivers into account and Detailed COCOMO additionally accounts for the influence of individual project phases.

The Constructive Cost Model (COCOMO) is an algorithmic software cost estimation model developed by Barry Boehm. The model uses a basic regression formula, with parameters that are derived from historical project data and current project characteristics.

 COCOMO was first published in 1981 Barry W. Boehm's Book Software engineering economics[1] as a model for estimating effort, cost, and schedule for software projects. It drew on a study of 63 projects at TRW Aerospace where Barry Boehm was Director of Software Research and Technology in 1981. The study examined projects ranging in size from 2,000 to 100,000 lines of code, and programming languages ranging from assembly to PL/I. These projects were based on the waterfall model of software development which was the prevalent software development process in 1981.

 References to this model typically call it COCOMO 81. In 1997 COCOMO II was developed and finally published in 2000 in the book Software Cost Estimation with COCOMO II[2]. COCOMO II is the successor of COCOMO 81 and is better suited for estimating modern software development projects. It provides more support for modern software development processes and an updated project database. The need for the new model came as software development technology moved from mainframe and overnight batch processing to desktop development, code reusability and the use of off-the-shelf software components. This article refers to COCOMO 81.

 COCOMO consists of a hierarchy of three increasingly detailed and accurate forms. The first level, Basic COCOMO is good for quick, early, rough order of magnitude estimates of software costs, but its accuracy is limited due to its lack of factors to account for difference in project attributes (Cost Drivers). Intermediate COCOMO takes these Cost Drivers into account and Detailed COCOMO additionally accounts for the influence of individual project phases.Basic COCOMO computes software development effort (and cost) as a function of program size. Program size is expressed in estimated thousands of lines of code (KLOC).

COCOMO applies to three classes of software projects:

**\* Organic projects - "**small" teams with "good" experience working with "less than rigid" requirements

    \* **Semi-detached projects** - "medium" teams with mixed experience working with a mix of rigid and less than rigid requirements

    \* **Embedded projects** - developed within a set of "tight" constraints (hardware, software, operational, ...)

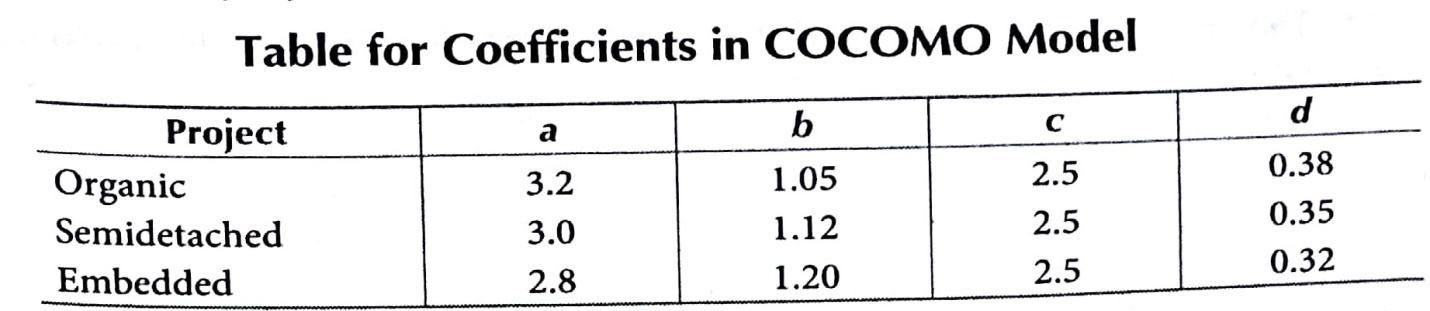
The basic COCOMO equations take the form

     Effort Applied = ab(KLOC)bb [ man-months ]

    Development Time = cb(Effort Applied)db [months]

    People required = Effort Applied / Development Time [count]

The coefficients ab, bb, cb and db are given in the following table.



Basic COCOMO is good for quick estimate of software costs. However it does not account for differences in hardware constraints, personnel quality and experience, use of modern tools and techniques, and so on.

Intermediate COCOMO computes software development effort as function of program size and a set of "cost drivers" that include subjective assessment of product, hardware, personnel and project attributes. This extension considers a set of four "cost drivers",each with a number of subsidiary attributes:-

    \* Product attributes

           Required software reliability

           Size of application database

           Complexity of the product

    \* Hardware attributes

          Run-time performance constraints

          Memory constraints

          Volatility of the virtual machine environment

           Required turnabout time

    \* Personnel attributes

           Analyst capability

          Software engineering capability

           Applications experience

           Virtual machine experience

           Programming language experience

    \* Project attributes

           Use of software tools

           Application of software engineering methods

           Required development schedule

# What is Waterfall model- advantages, disadvantages and when to use it?

FaceokLinkedInTwitterGoogle+Email

The Waterfall Model was first Process Model to be introduced. It is also referred to as a **linear-sequential life cycle model**.  It is very simple to understand and use.  In a waterfall model, each phase must be completed fully before the next phase can begin. This type of [**software development model**](http://tryqa.com/what-are-the-software-development-models/) is basically used for the for the project which is small and there are no uncertain requirements. At the end of each phase, a review takes place to determine if the project is on the right path and whether or not to continue or discard the project. In this model [**software testing**](http://tryqa.com/what-is-a-software-testing/) starts only after the development is complete. In **waterfall model phases** do not overlap.

**Diagram of Waterfall-model:**



**Advantages of waterfall model:**

* This model is simple and easy to understand and use.
* It is easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process.
* In this model phases are processed and completed one at a time. Phases do not overlap.
* Waterfall model works well for smaller projects where requirements are very well understood.

**Disadvantages of waterfall model:**

* Once an application is in the [**testing**](http://tryqa.com/what-is-a-software-testing/) stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.
* No working software is produced until late during the life cycle.
* High amounts of risk and uncertainty.
* Not a good model for complex and object-oriented projects.
* Poor model for long and ongoing projects.
* Not suitable for the projects where requirements are at a moderate to high risk of changing.

**When to use the waterfall model:**

* This model is used only when the requirements are very well known, clear and fixed.
* Product definition is stable.
* Technology is understood.
* There are no ambiguous requirements
* Ample resources with required expertise are available freely
* The project is short.

Very less customer interaction is involved during the development of the product. Once the product is ready then only it can be demoed to the end users. Once the product is developed and if any failure occurs then the cost of fixing such issues are very high, because we need to update everywhere from document till the logic.

Or

### Advantages of Waterfall Iterative Model

1)  Waterfall model is simple to implement and also the amount of resources required for it are minimal.  
2)   In this model, output is generated after each stage (as seen before), therefore it has high visibility. The client and project manager gets a feel that there is considerable progress. Here it is important to note that in any project psychological factors also play an important role.  
3)  Project management, both at internal level and client's level, is easy again because of visible outputs after each phase. Deadlines can be set for the completion of each phase and evaluation can be done from time to time, to check if project is going as per milestones.  
4)  This methodology is significantly better than the haphazard approach to develop software. It provides a template into which methods of analysis, design, coding, testing and maintenance can be placed.  
5)  This methodology is preferred in projects where quality is more important as compared to schedule or cost.

### Disadvantages of Waterfall Iterative Model of SDLC

1)  Real projects rarely follow the sequential flow and iterations in this model are handled indirectly. These changes can cause confusion as the project proceeds.   
2)  It is often difficult to get customer requirements explicitly. Thus specifications can't be freezed. If that case arises baseline approach is followed, wherein output of one phase is carried forward to next phase. For example, even if SRS is not well defined and requirements can't be freezed, still design starts. Now if any changes are made in SRS then formal procedure is followed to put those changes in baseline document.  
3)  In this model we freeze software and hardware. But as technology changes at a rapid pace,such freezing is not advisable especially in long-term projects.   
4)  This method is especially bad in case client is not IT-literate as getting specifications from such a person is tough.  
5)  Even a small change in any previous stage can cause big problem for subsequent phases as all phases are dependent on each-other.  
6)  Going back a phase or two can be a costly affair.  
  
Projects where Waterfall Method is suitable for SDLC:-

1)  In development of database-related software, eg commercial projects.  
2)  In development of E-commerce website or portal.  
3)  In Development of network protocol software.

# What is Spiral model- advantages, disadvantages and when to use it?

The spiral model is similar to the [**incremental model**](http://tryqa.com/what-is-incremental-model-advantages-disadvantages-and-when-to-use-it/), with more emphasis placed on risk analysis. The spiral model has four phases: Planning, Risk Analysis, Engineering and Evaluation. A software project repeatedly passes through these phases in iterations (called Spirals in this model). The baseline spiral, starting in the planning phase, requirements are gathered and risk is assessed. Each subsequent spirals builds on the baseline spiral. Its one of the [**software development models**](http://tryqa.com/what-are-the-software-development-models/) like [**Waterfall**](http://tryqa.com/what-is-waterfall-model-advantages-disadvantages-and-when-to-use-it/), [**Agile**](http://tryqa.com/what-is-agile-model-advantages-disadvantages-and-when-to-use-it/), [**V-Model**](http://tryqa.com/what-is-v-model-advantages-disadvantages-and-when-to-use-it/).

**Planning Phase:**Requirements are gathered during the planning phase. Requirements like ‘BRS’ that is ‘Bussiness Requirement Specifications’ and ‘SRS’ that is ‘System Requirement specifications’.

**Risk Analysis:** In the**risk analysis phase**, a process is undertaken to identify risk and alternate solutions.  A prototype is produced at the end of the risk analysis phase. If any risk is found during the risk analysis then alternate solutions are suggested and implemented.

**Engineering Phase:** In this phase software is **developed**, along with [**testing**](http://tryqa.com/what-is-a-software-testing/) at the end of the phase. Hence in this phase the development and testing is done.

E**valuation phase:**This phase allows the customer to evaluate the output of the project to date before the project continues to the next spiral.

**Diagram of Spiral model:**

|  |  |
| --- | --- |
| **Advantages of Spiral model:**   * High amount of risk analysis hence, avoidance of Risk is enhanced. * Good for large and mission-critical projects. * Strong approval and documentation control. * Additional Functionality can be added at a later date.   Software is produced early in the [**software life cycle**](http://tryqa.com/what-are-the-software-development-life-cycle-phases/) | **[Spiral model](http://tryqa.com/wp-content/uploads/2012/01/Spiral-model.jpg)** |

**Disadvantages of Spiral model:**

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

**When to use Spiral model:**

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

**What is v model- advantages, disadvantages and when to use it?**

**V- model** means Verification and Validation model. Just like the [**waterfall model**](http://tryqa.com/what-is-waterfall-model-advantages-disadvantages-and-when-to-use-it/), the V-Shaped life cycle is a sequential path of execution of processes. Each phase must be completed before the next phase begins. **V-Model** is one of the [**many software development models**](http://tryqa.com/what-are-the-software-development-models/).Testing of the product is planned in parallel with a corresponding phase of development in **V-model**.

**Diagram of V-model:**



The various phases of the V-model are as follows:

**Requirements** like BRS and SRS begin the life cycle model just like the waterfall model. But, in this model before development is started, a [**system test**](http://tryqa.com/what-is-system-testing/) plan is created.  The [**test plan**](http://tryqa.com/what-is-the-purpose-and-importance-of-test-plans/) focuses on meeting the functionality specified in the requirements gathering.

**The high-level design (HLD)** phase focuses on system architecture and design. It provide overview of solution, platform, system, product and service/process. An [**integration test**](http://tryqa.com/what-is-integration-testing/) plan is created in this phase as well in order to test the pieces of the software systems ability to work together.

**The low-level design** **(LLD)** phase is where the actual software components are designed. It defines the actual logic for each and every component of the system. Class diagram with all the methods and relation between classes comes under LLD. [**Component tests**](http://tryqa.com/what-is-component-testing/) are created in this phase as well.

**The implementation** phase is, again, where all coding takes place. Once coding is complete, the path of execution continues up the right side of the V where the test plans developed earlier are now put to use.

**Coding:** This is at the bottom of the V-Shape model. Module design is converted into code by developers. [**Unit Testing**](http://tryqa.com/what-is-unit-testing/) is performed by the developers on the code written by them.

**Advantages of V-model:**

* Simple and easy to use.
* Testing activities like planning, [**test designing**](http://tryqa.com/what-is-test-design-or-how-to-specify-test-cases/) happens well before coding. This saves a lot of time. Hence higher chance of success over the waterfall model.
* Proactive defect tracking – that is defects are found at early stage.
* Avoids the downward flow of the defects.
* Works well for small projects where requirements are easily understood.

**Disadvantages of V-model:**

* Very rigid and least flexible.
* Software is developed during the implementation phase, so no early prototypes of the software are produced.
* If any changes happen in midway, then the test documents along with requirement documents has to be updated.

**When to use the V-model:**

* The V-shaped model should be used for small to medium sized projects where requirements are clearly defined and fixed.
* The V-Shaped model should be chosen when ample technical resources are available with needed technical expertise.

High confidence of customer is required for choosing the V-Shaped model approach. Since, no prototypes are produced, there is a very high risk involved in meeting customer expectations.

**What is Incremental model- advantages, disadvantages and when to use it?**

In **incremental model** the whole requirement is divided into various builds. Multiple development cycles take place here, making the life cycle a [**“multi-waterfall” cycle**](http://tryqa.com/what-is-waterfall-model-advantages-disadvantages-and-when-to-use-it/).  Cycles are divided up into smaller, more easily managed modules. Incremental model is a type of software development model like [**V-model**](http://tryqa.com/what-is-v-model-advantages-disadvantages-and-when-to-use-it/), [**Agile model**](http://tryqa.com/what-is-agile-model-advantages-disadvantages-and-when-to-use-it/) etc.

In this model, each module passes through the requirements, design, implementation and [**testing**](http://tryqa.com/what-is-a-software-testing/) phases. A working version of software is produced during the first module, so you have working software early on during the [**software life cycle**](http://tryqa.com/what-are-the-software-development-life-cycle-phases/). Each subsequent release of the module adds function to the previous release. The process continues till the complete system is achieved.

For example:

In the diagram above when we work **incrementally**we are adding piece by piece but expect that each piece is fully finished. Thus keep on adding the pieces until it’s complete. As in the image above a person has thought of the application. Then he started building it and in the first iteration the first module of the application or product is totally ready and can be demoed to the customers. Likewise in the second iteration the other module is ready and integrated with the first module. Similarly, in the third iteration the whole product is ready and integrated. Hence, the product got ready step by step.

**Diagram of Incremental model:**



**Advantages of Incremental model:**

* Generates working software quickly and early during the software life cycle.
* This model is more flexible – less costly to change scope and requirements.
* It is easier to test and debug during a smaller iteration.
* In this model customer can respond to each built.
* Lowers initial delivery cost.
* Easier to manage risk because risky pieces are identified and handled during it’d iteration.

**Disadvantages of Incremental model:**

* Needs good planning and design.
* Needs a clear and complete definition of the whole system before it can be broken down and built incrementally.
* Total cost is higher than [**waterfall**](http://tryqa.com/what-is-waterfall-model-advantages-disadvantages-and-when-to-use-it/).

**When to use the Incremental model:**

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

# What is Prototype model- advantages, disadvantages and when to use it?

The basic idea in **Prototype model** is that instead of freezing the requirements before a design or coding can proceed, a throwaway prototype is built to understand the requirements. This prototype is developed based on the currently known requirements. Prototype model is a [**software development model**](http://tryqa.com/what-are-the-software-development-models/). By using this prototype, the client can get an “actual feel” of the system, since the interactions with prototype can enable the client to better understand the requirements of the desired system.  Prototyping is an attractive idea for complicated and large systems for which there is no manual process or existing system to help determining the requirements.

The prototype are usually not complete systems and many of the details are not built in the prototype. The goal is to provide a system with overall functionality.

**Diagram of Prototype model:**



**Advantages of Prototype model:**

* Users are actively involved in the development
* Since in this methodology a working model of the system is provided, the users get a better understanding of the system being developed.
* Errors can be detected much earlier.
* Quicker user feedback is available leading to better solutions.
* Missing functionality can be identified easily
* Confusing or difficult functions can be identified  
  Requirements validation, Quick implementation of, incomplete, but  
  functional, application.

**Disadvantages of Prototype model:**

* Leads to implementing and then repairing way of building systems.
* Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans.
* Incomplete application may cause application not to be used as the  
  full system was designed  
  Incomplete or inadequate problem analysis.

**When to use Prototype model:**

* Prototype model should be used when the desired system needs to have a lot of interaction with the end users.
* Typically, online systems, web interfaces have a very high amount of interaction with end users, are best suited for Prototype model. It might take a while for a system to be built that allows ease of use and needs minimal training for the end user.
* Prototyping ensures that the end users constantly work with the system and provide a feedback which is incorporated in the prototype to result in a useable system. They are excellent for designing good human computer interface systems.

# What is Agile model – advantages, disadvantages and when to use it?

**Agile development model** is also a type of [**Incremental model**](http://tryqa.com/what-is-incremental-model-advantages-disadvantages-and-when-to-use-it/). Software is developed in incremental, rapid cycles. This results in small incremental releases with each release building on previous functionality. Each release is thoroughly [**tested**](http://tryqa.com/why-is-testing-necessary/) to ensure [**software quality**](http://tryqa.com/what-is-software-quality/) is maintained. It is used for time critical applications.  Extreme Programming (XP) is currently one of the most well known agile [**development life cycle model**](http://tryqa.com/what-are-the-software-development-models/).

**Diagram of Agile model:**



**Advantages of Agile model:**

* Customer satisfaction by rapid, continuous delivery of useful software.
* People and interactions are emphasized rather than process and tools. Customers, developers and testers constantly interact with each other.
* Working software is delivered frequently (weeks rather than months).
* Face-to-face conversation is the best form of communication.
* Close, daily cooperation between business people and developers.
* Continuous attention to technical excellence and good design.
* Regular adaptation to changing circumstances.
* Even late changes in requirements are welcomed

**Disadvantages of Agile model:**

* In case of some software deliverables, especially the large ones, it is difficult to assess the effort required at the beginning of the software development life cycle.
* There is lack of emphasis on necessary designing and documentation.
* The project can easily get taken off track if the customer representative is not clear what final outcome that they want.
* Only senior programmers are capable of taking the kind of decisions required during the development process. Hence it has no place for newbie programmers, unless combined with experienced resources.

**When to use Agile model:**

* When new changes are needed to be implemented. The freedom agile gives to change is very important. New changes can be implemented at very little cost because of the frequency of new increments that are produced.
* To implement a new feature the developers need to lose only the work of a few days, or even only hours, to roll back and implement it.
* Unlike the [**waterfall model**](http://tryqa.com/what-is-waterfall-model-advantages-disadvantages-and-when-to-use-it/) in agile model very limited [**planning**](http://tryqa.com/what-is-the-purpose-and-importance-of-test-plans/) is required to get started with the project. Agile assumes that the end users’ needs are ever changing in a dynamic business and IT world. Changes can be discussed and features can be newly effected or removed based on feedback. This effectively gives the customer the finished system they want or need.
* Both system developers and stakeholders alike, find they also get more freedom of time and options than if the software was developed in a more rigid sequential way. Having options gives them the ability to leave important decisions until more or better data or even entire hosting programs are available; meaning the project can continue to move forward without fear of reaching a sudden standstill

# What is Iterative model- advantages, disadvantages and when to use it?

FacebookLinkedInTwitterGoogle+Email

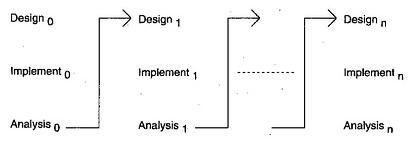
An iterative [**life cycle model**](http://tryqa.com/what-are-the-software-development-models/) does not attempt to start with a full specification of requirements. Instead, development begins by specifying and implementing just part of the software, which can then be reviewed in order to identify further requirements. This process is then repeated, producing a new version of the software for each cycle of the model.

For example:



In the diagram above when we work **iteratively**we create rough product or product piece in one iteration, then review it and improve it in next iteration and so on until it’s finished. As shown in the image above, in the first iteration the whole painting is sketched roughly, then in the second iteration colors are filled and in the third iteration finishing is done. Hence, in iterative model the whole product is developed step by step.

**Diagram of Iterative model:**



**Advantages of Iterative model:**

* In iterative model we can only create a high-level design of the application before we actually begin to build the product and define the design solution for the entire product. Later on we can design and built a skeleton version of that, and then evolved the design based on what had been built.
* In iterative model we are building and improving the product step by step. Hence we can track the defects at early stages. This avoids the downward flow of the defects.
* In iterative model we can get the reliable user feedback. When presenting sketches and blueprints of the product to users for their feedback, we are effectively asking them to imagine how the product will work.
* In iterative model less time is spent on documenting and more time is given for designing.

**Disadvantages of Iterative model:**

* Each phase of an iteration is rigid with no overlaps
* Costly system architecture or design issues may arise because not all requirements are gathered up front for the entire lifecycle

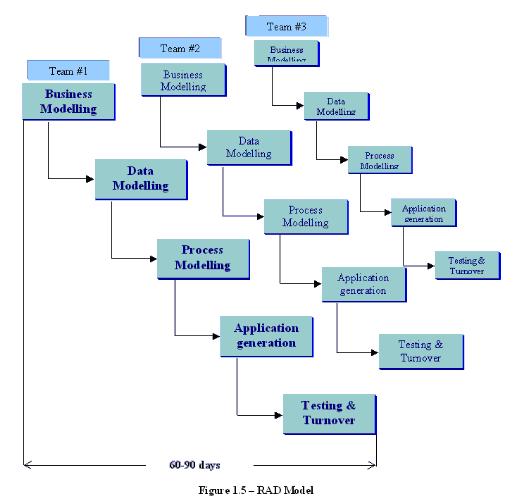
**When to use iterative model:**

* Requirements of the complete system are clearly defined and understood.
* When the project is big.
* Major requirements must be defined; however, some details can evolve with time.

# What is RAD model- advantages, disadvantages and when to use it?

RAD model is Rapid Application Development model. It is a type of [**incremental model**](http://tryqa.com/what-is-incremental-model-advantages-disadvantages-and-when-to-use-it/). In RAD model the components or functions are developed in parallel as if they were mini projects. The developments are time boxed, delivered and then assembled into a working prototype.  This can quickly give the customer something to see and use and to provide feedback regarding the delivery and their requirements.

**Diagram of RAD-Model:**



The phases in the rapid application development (RAD) model are:

**Business modeling:** The information flow is identified between various business functions.  
**Data modeling:** Information gathered from business modeling is used to define data objects that are needed for the business.  
**Process modeling:** Data objects defined in data modeling are converted to achieve the business information flow to achieve some specific business objective. Description are identified and created for CRUD of data objects.  
**Application generation:** Automated tools are used to convert process models into code and the actual system.  
**Testing and turnover:** Test new components and all the interfaces.

**Advantages of the RAD model:**

* Reduced development time.
* Increases reusability of components
* Quick initial reviews occur
* Encourages customer feedback
* Integration from very beginning solves a lot of [**integration issues**](http://tryqa.com/what-is-system-integration-testing/).

**Disadvantages of RAD model:**

* Depends on strong team and individual performances for identifying business requirements.
* Only system that can be modularized can be built using RAD
* Requires highly skilled developers/designers.
* High dependency on modeling skills
* Inapplicable to cheaper projects as cost of modeling and automated code generation is very high.

**When to use RAD model:**

* RAD should be used when there is a need to create a system that can be modularized in 2-3 months of time.
* It should be used if there’s high availability of designers for modeling and the budget is high enough to afford their cost along with the cost of automated code generating tools.
* RAD [**SDLC model**](http://tryqa.com/what-are-the-software-development-models/) should be chosen only if resources with high business knowledge are available and there is a need to produce the system in a short span of time (2-3 months).

‘’’

### Problem Analysis

* **Informal Approach :**

1. The informal approach to analysis is one where no defined methodology is used.
2. The information about the system is obtained by interaction with the client, end users, questionnaires, study of existing documents, brainstorming, etc.
3. The informal approach to analysis is used widely and can be quite useful because conceptual modeling-based approaches frequently do not model all aspects of the problem and are not always well suited for all the problems.
4. as the SRS is to be validated and the feedback from the validation activity may require further analysis or specification.
5. choosing an informal approach to analysis is not very risky—-the errors that may be introduced are not necessarily going to slip by the requirements phase. Hence such approaches may be the most practical approach to analysis in some situations.

* Various fact finding methods are used to collect detailed information about every aspect of an existing system.
* **Shadowing :**

1. Shadowing is a technique in which you observe a user performing the tasks in the actual work environment and ask the user any questions related to the task.
2. You typically follow the user as the user performs tasks.
3. The information obtained by using this technique was firsthand and in context.

* **Interviews:**

1. In interview is a one-on-one meeting between a member of the project team and a user.
2. The quality of the information a team gathers depends on the skills of both the interviewer and the interviewee.
3. An interviewer can learn a great deal about the difficulties and limitations of the current solution.
4. Interviews provide the opportunity to ask a wide range of questions about topics that you cannot observe by means of shadowing.

* **Some of the questions we covered were:**

1. What do you look from the system?
2. What are the problems you face while performing your task?
3. What are the details that need to be maintained?
4. Who provides the information needed to perform tasks?
5. What information do you need to maintain for future use?
6. What changes would make their experience more enjoyable?

* **Hence with this we were able to achieve the following:**

1. Identify the types of information that is gathered and processed and maintained.
2. Identify the sources for information.
3. Identify techniques required for data processing and the business rules to keep in mind to do the same.
4. Identify end user requirements from the system.
5. Identify the changes to be made to the system to make the experience more user-friendly.

**Software Metrics and Measures**

Software Measures can be understood as a process of quantifying and symbolizing various attributes and aspects of software.

Software Metrics provide measures for various aspects of software process and software product.

Software measures are fundamental requirement of software engineering. They not only help to control the software development process but also aid to keep quality of ultimate product excellent.

According to Tom DeMarco, a (Software Engineer), “You cannot control what you cannot measure.” By his saying, it is very clear how important software measures are.

Let us see some software metrics:

* **Size Metrics -**LOC (Lines of Code), mostly calculated in thousands of delivered source code lines, denoted as KLOC.

Function Point Count is measure of the functionality provided by the software. Function Point count defines the size of functional aspect of software.

* **Complexity Metrics -**McCabe’s Cyclomatic complexity quantifies the upper bound of the number of independent paths in a program, which is perceived as complexity of the program or its modules. It is represented in terms of graph theory concepts by using control flow graph.
* **Quality Metrics -**Defects, their types and causes, consequence, intensity of severity and their implications define the quality of product.

The number of defects found in development process and number of defects reported by the client after the product is installed or delivered at client-end, define quality of product.

* **Process Metrics -**In various phases of SDLC, the methods and tools used, the company standards and the performance of development are software process metrics.
* **Resource Metrics -**Effort, time and various resources used, represents metrics for resource measurement.

### Role of Metrics and Measurement in software development

* The terms measure, measurement, and metrics are often used interchangeably, it is important to note the subtle differences between them. Because measure can be used either as a noun or a verb, definitions of the term can become confusing.
* When a single data point has been collected (e.g., the number of errors uncovered in the review of a single module), a **measure** has been established.
* **Measurement** occurs as the result of the collection of one or more data points (e.g., a number of module reviews are investigated to collect measures of the number of errors for each).
* A software **metric** relates the individual measures in some way (e.g., the average number of errors found per review or the average number of errors found per person-hour expended on reviews).
* Measurements in the physical world can be categorized in two ways: direct measures and indirect measures.
* Direct measures of the product include lines of code (LOC) produced, execution speed, memory size, and defects reported over some set period of time. Indirect measures of the product include functionality, quality, complexity, efficiency, reliability, maintainability.
* The basic purpose of metrics at any point during a development project is to provide quantitative information to the management process so that the information can be used to effectively control the development process. Unless the metric is useful in some form to monitor or control the cost, schedule, or quality of the project, it is of little use for a project.
* There are very few metrics that have been defined for requirements.

**Project Scheduling**

Project scheduling is concerned with the techniques that can be employed to manage the activities that need to be undertaken during the development of a project.

Scheduling is carried out in advance of the project commencing and involves:

•    identifying the tasks that need to be carried out;

•    estimating how long they will take;

•    allocating resources (mainly personnel);

•    scheduling when the tasks will occur.

Once the project is underway control needs to be exerted to ensure that the plan continues to represent the best prediction of what will occur in the future:

•    based on what occurs during the development;

•    often necessitates revision of the plan.

Effective project planning will help to ensure that the systems are delivered:

•    within cost;

•    within the time constraint;

•    to a specific standard of quality.

**Two project scheduling techniques** will be presented, the Milestone Chart (or Gantt Chart) and the Activity Network.

Milestone Charts

Milestones mark significant events in the life of a project, usually critical activities which must be achieved on time to avoid delay in the project.

Milestones should be truely significant and be reasonable in terms of deadlines (avoid using intermediate stages).

Examples include:

•    installation of equipment;

•    completion of phases;

•    file conversion;

•    cutover to the new system

Gantt Charts

A Gantt chart is a horizontal bar or line chart which will commonly include the following features:

•    activities identified on the left hand side;

•    time scale is drawn on the top (or bottom) of the chart;

•    a horizontal open oblong or a line is drawn against each activity indicating estimated duration;

•    dependencies between activities are shown;

•    at a review point the oblongs are shaded to represent the actual time spent (an alternative is to represent actual and estimated by 2 separate lines);

•    a vertical cursor (such as a transparent ruler) placed at the review point makes it possible to establish activities which are behind or ahead of schedule.

Activity Networks

The foundation of the approach came from the Special Projects Office of the US Navy in 1958. It developed a technique for evaluating the performance of large development projects, which became known as PERT - Project Evaluation and Review Technique. Other variations of the same approach are known as the critical path method (CPM) or critical path analysis (CPA).

The heart of any PERT chart is a network of tasks needed to complete a project, showing the order in which the tasks need to be completed and the dependencies between them. This is represented graphically:

EXAMPLE OF ACTIVITY NETWORK

The diagram consists of a number of circles, representing events within the development lifecycle, such as the start or completion of a task, and lines, which represent the tasks themselves. Each task is additionally labelled by its time duration. Thus the task between events 4 & 5 is planned to take 3 time units. The primary benefit is the identification of the critical path.

The critical path = total time for activities on this path is greater than any other path through the network (delay in any task on the critical path leads to a delay in the project).

Tasks on the critical path therefore need to be monitored carefully.

The technique can be broken down into 3 stages:

1. Planning:

•    identify tasks and estimate duration of times;

•    arrange in feasible sequence;

•    draw diagram.

2. Scheduling:

•    establish timetable of start and finish times.

3. Analysis:

•    establish float;

•     evaluate and revise as necessary

**Basic Principles for Project Scheduling:-**

**Compartmentalization**

– The project must be compartmentalized into a number of manageable activities, actions, and tasks; both the product and the process are decomposed

• **Interdependency**

– The interdependency of each compartmentalized activity, action, or task must be determined

– Some tasks must occur in sequence while others can occur in parallel

– Some actions or activities cannot commence until the work product produced by another is available

**Time allocation**

– Each task to be scheduled must be allocated some number of work units

– In addition, each task must be assigned a start date and a completion date that are a function of the interdependencies

– Start and stop dates are also established based on whether work will be conducted on a full-time or parttime basis.

• **Effort validation**

– Every project has a defined number of people on the team

– As time allocation occurs, the project manager must ensure that no more than the allocated number of people have been scheduled at any given time.

**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 such as a work product or 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 and has been approved

# [Project Planning in Software Engineering](http://ecomputernotes.com/software-engineering/project-planning)

Before starting a software project, it is essential to determine the tasks to be performed and properly manage allocation of tasks among individuals involved in the software development. Hence, planning is important as it results in effective software development.

Project planning is an organized and integrated management process, which focuses on activities required for successful completion of the project. It prevents obstacles that arise in the project such as changes in projects or organization's objectives, non-availability of resources, and so on. Project planning also helps in better utilization of resources and optimal usage of the allotted time for a project. The other objectives of project planning are listed below.

* It defines the roles and responsibilities of the project management team members.
* It ensures that the project management team works according to the business objectives.
* It checks feasibility of the schedule and user requirements.
* It determines project constraints.

Several individuals help in planning the project. These include senior management and project management team. Senior management is responsible for employing team members and providing resources required for the project. The project management team, which generally includes project managers and developers, is responsible for planning, determining, and tracking the activities of the project. Table lists the tasks performed by individuals involved in the software project.

**Tasks of Individuals involved in Software Project**

|  |  |
| --- | --- |
| **Senior Management** | **Project Management Team** |
| * Approves the project, employ personnel, and provides resources required for the project. * Reviews project plan to ensure that it accomplishes the business objectives. * Resolves conflicts among the team members. * Considers risks that may affect the project so that appropriate measures can be taken to avoid them. | * Reviews the project plan and implements procedures for completing the project. * Manages all project activities. * Prepares budget and resource allocation plans. * Helps in resource distribution, project management, issue resolution, and so on. * Understands project objectives and finds ways to accomplish the objectives. * Devotes appropriate time and effort to achieve the expected results. * Selects methods and tools for the project. |

Project planning should be effective so that the project begins with well-defined tasks. Effective project planning helps to minimize the additional costs incurred on the project while it is in progress. For effective project planning, some principles are followed. These principles are listed below.

* **Planning is necessary:** Planning should be done before a project begins. For effective planning, objectives and schedules should be clear and understandable.
* **Risk analysis:** Before starting the project, senior management and the project management team should consider the risks that may affect the project. For example, the user may desire changes in requirements while the project is in progress. In such a case, the estimation of time and cost should be done according to those requirements (new requirements).
* **Tracking of project plan:** Once the project plan is prepared, it should be tracked and modified accordingly.
* **Meet quality standards and produce quality deliverables:**The project plan should identify processes by which the project management team can ensure quality in software. Based on the process selected for ensuring quality, the time and cost for the project is estimated.
* **Description of flexibility to accommodate changes:** The result of project planning is recorded in the form of a project plan, which should allow new changes to be accommodated when the project is in progress.

Project planning comprises project purpose, project scope, project planning process, and project plan. This [information](http://ecomputernotes.com/fundamental/information-technology/what-do-you-mean-by-data-and-information) is essential for effective project planning and to assist project management team in accomplishing user requirements.

## Project Purpose

Software project is carried out to accomplish a specific purpose, which is classified into two categories, namely, project objectives and business objectives. The commonly followed project objectives are listed below.

* **Meet user requirements:** Develop the project according to the user requirements after understanding them.
* **Meet schedule deadlines:** Complete the project milestones as described in the project plan on time in order to complete the project according to the schedule.
* **Be within budget:** Manage the overall project cost so that the project is within the allocated budget.
* **Produce quality deliverables:** Ensure that quality is considered for accuracy and overall performance of the project.

## Business Software Engineering

Business objectives ensure that the organizational objectives and requirements are accomplished in the project. Generally, these objectives are related to business process improvements, customer satisfaction, and quality improvements. The commonly followed business objectives are listed below.

* **Evaluate processes:** Evaluate the business processes and make changes when and where required as the project progresses.
* **Renew policies and processes:** Provide flexibility to renew the policies and processes of the organization in order to perform the tasks effectively.
* **Keep the project on schedule:**Reduce the downtime (period when no work is done) factors such as unavailability of resources during software development.
* **Improve software:** Use suitable processes in order to develop software that meets organizational requirements and provides competitive advantage to the organization.

## Project Scope

With the help of user requirements, the project management team determines the scope of the project before the project begins. This scope provides a detailed description of functions, features, constraints, and interfaces of the software that are to be considered. Functions describe the tasks that the software is expected to perform. Features describe the attributes required in the software as per the user requirements. Constraints describe the limitations imposed on software by hardware, memory, and so on. Interfaces describe the interaction of software components (like modules and functions) with each other. Project scope also considers software performance, which in turn depends on its processing capability and response time required to produce the output.

Once the project scope is determined, it is important to properly understand it in order to develop software according to the user requirements. After this, project cost and duration are estimated. Lf the project scope is not determined on time, the project may not be completed within the specified schedule. Project scope describes the following information.

* The elements included and excluded in the project
* The processes and entities
* The functions and features required in software according to the user requirements.

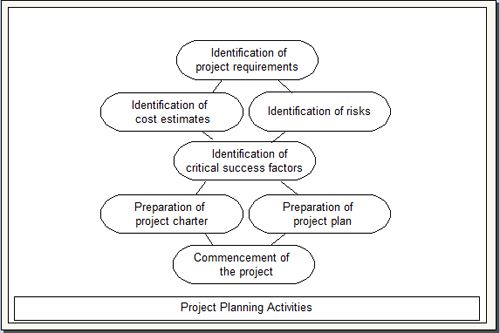
Note that the project management and senior management team should communicate with the users to understand their requirements and develop software according to those requirements and expected functionalities.

## Project Planning Process

The project planning process involves a set of interrelated activities followed in an orderly manner to implement user requirements in software and includes the description of a series of project planning activities and individual(s) responsible for performing these activities. In addition, the project planning process comprises the following.

1. Objectives and scope of the project
2. Techniques used to perform project planning
3. Effort (in time) of individuals involved in project
4. Project schedule and milestones
5. Resources required for the project
6. Risks associated with the project.

Project planning process comprises several activities, which are essential for carrying out a project systematically. These activities refer to the series of tasks performed over a period of time for developing the software. These activities include estimation of time, effort, and resources required and risks associated with the project.

[](http://ecomputernotes.com/images/Project-Planning-Activities.jpg)

Project planning process consists of the following activities.

* **Identification of project requirements:** Before starting a project, it is essential to identify the project requirements as identification of project requirements helps in performing the activities in a systematic manner. These requirements comprise information such as project scope, data and functionality required in the software, and roles of the project management team members.
* **Identification of cost estimates:** Along with the estimation of effort and time, it is necessary to estimate the cost that is to be incurred on a project. The cost estimation includes the cost of hardware, network connections, and the cost required for the maintenance of hardware components. In addition, cost is estimated for the individuals involved in the project.
* **Identification of risks:** Risks are unexpected events that have an adverse effect on the project. Software project involves several risks (like technical risks and business risks) that affect the project schedule and increase the cost of the project. Identifying risks before a project begins helps in understanding their probable extent of impact on the project.
* **Identification of critical success factors:** For making a project successful, critical success factors are followed. These factors refer to the conditions that ensure greater chances of success of a project. Generally, these factors include support from management, appropriate budget, appropriate schedule, and skilled software engineers.
* **Preparation of project charter:** A project charter provides a brief description of the project scope, quality, time, cost, and resource constraints as described during project planning. It is prepared by the management for approval from the sponsor of the project.
* **Preparation of project plan:** A project plan provides information about the resources that are available for the project, individuals involved in the project, and the schedule according to which the project is to be carried out.
* **Commencement of the project:** Once the project planning is complete and resources are assigned to team members, the software project commences.

Once the project objectives and business objectives are determined, the project end date is fixed. The project management team prepares the project plan and schedule according to the end date of the project. After analyzing the project plan, the project manager communicates the project plan and end date to the senior management. The progress of the project is reported to the management from time to time. Similarly, when the project is complete, senior management is informed about it. In case of delay in completing the project, the project plan is re-analyzed and corrective actions are taken to complete the project. The project is tracked regularly and when the project plan is modified, the senior management is informed.

## Project Plan

As stated earlier, a project plan stores the outcome of project planning. It provides information about the end date, milestones, activities, and deliverables of the project. In addition, it describes the responsibilities of the project management team and the resources required for the project. It also includes the description of hardware and software (such as compilers and interfaces) and lists the methods and standards to be used. These methods and standards include algorithms, tools, review techniques, design language, programming language, and testing techniques.

A project plan helps a project manager to understand, monitor, and control the development of software project. This plan is used as a means of communication between the users and project management team. There are various advantages associated with a project plan, some of which are listed below.

* It ensures that software is developed according to the user requirements, objectives, and scope of the project.
* It identifies the role of each project management team member involved in the project.
* It monitors the progress of the project according to the project plan.
* It determines the available resources and the activities to be performed during software development.
* It provides an overview to management about the costs of the software project, which are estimated during project planning.

Note that there are differences in the contents of two project plans depending on the kind of project and user requirements. Atypical project plan is divided into the following sections.

1. **Introduction:** Describes the objectives of the project and provides information about the constraints that affect the software project.
2. **Project organization:** Describes the responsibilities assigned to the project management team members for completing the project.
3. **Risk analysis:** Describes the risks that can possibly arise during software development as well as explains how to assess and reduce the effect of risks.
4. **Resource requirements:** Specifies the hardware and software required to carry out the software project. Cost estimation is done according to these resource requirements.
5. **Workbreakdown:** Describes the activities into which the project is divided. It also describes the milestones and deliverables of the project activities.
6. **Project schedule:** Specifies the dependencies of activities on each other. Based on this, the time required by the project management team members to complete the project activities is estimated.

In addition to these sections, there are several plans that may be a part of or 'linked to a project plan. These plans include quality assurance plan, verification and validation plan, configuration management plan, maintenance plan, and staffing plan.

**Quality Assurance Plan**

The quality assurance plan describes the strategies and methods that are to be followed to accomplish the following objectives.

1. Ensure that the project is managed, developed, and implemented in an organized way.
2. Ensure that project deliverables are of acceptable quality before they are delivered to the user.

**Verification and Validation Plan**

The verification and validation plan describes the approach, resources and schedule used for system validation. The verification and validation plan, which comprises the following sections.

$11.      **General information:**Provides description of the purpose, scope, system overview, project references, acronyms and abbreviations, and points of contact. Purpose describes the procedure to verify and validate the components of the system. Scope provides information about the procedures to verify and validate as they relate to the project. System overview provides information about the organization responsible for the project and other information such as system name, system category, operational status of the system, and system environment. Project references provide the list of references used for the preparation of the verification and validation plan. Acronyms and abbreviations provide a list of terms used in the document. Points of contact provide information to users when they require assistance from organization for problems such as troubleshooting and so on.

$12.      **Reviews and walkthroughs:** Provides information about the schedule and procedures. Schedule describes the end date of milestones of the project. Procedures describe the tasks associated with reviews and walkthroughs. Each team member reviews the document for errors and consistency with the project requirements. For walkthroughs, the project management team checks the project for correctness according to software requirements specification (SRS).

$13.      **System test plan and procedures:** Provides information about the system test strategy, [database](http://ecomputernotes.com/fundamental/what-is-a-database/advantages-and-disadvantages-of-dbms) integration, and platform system integration. System test strategy provides an overview of the components required for integration of the database and ensures that the application runs on at least two specific platforms. Database integration procedure describes how database is connected to the Graphical User Interface (GUI).Platform system integration procedure is performed on different [operating system](http://ecomputernotes.com/fundamental/disk-operating-system/what-is-operating-system)s to test the platform.

$14.      **Acceptance test and preparation for delivery:** Provides information about procedure, acceptance criteria, and installation procedure. Procedure describes how acceptance testing is to be performed on the software to verify its usability as required. Acceptance criteria describes that software will be accepted only if all the components, features and functions are tested including the system integration testing. In addition, acceptance criteria checks whether the software accomplishes user expectations such as its ability to operate on several platforms. Installation procedure describes the steps of how to install the software according to the operating system being used.

**Configuration Management Plan**

The configuration management plan defines the process, which is used for making changes to the project scope. Generally, the configuration management plan is concerned with redefining the existing objectives of the project and deliverables (software products that are delivered to the user after completion of software development).

**Maintenance Plan**

The maintenance plan specifies the resources and processes required for making the software operational after its installation. Sometimes, the project management team (or software development team) does not carry out the task of maintenance. In such a case, a separate team known as software maintenance team performs the task of software maintenance.

The maintenance plan, which comprises the sections listed below.

$11.      **Introduction and background:**Provides a description of software to be maintained and the services required for it. It also specifies the scope of maintenance activities that are to be performed.

$12.      **Budget:** Specifies the budget required for carrying out software maintenance and operational activities.

$13.      **Roles and responsibilities:** Specifies the roles and responsibilities of the team members associated with the software maintenance and operation. It also describes the skills required to perform maintenance and operational activities. In addition to software maintenance team, software maintenance comprises user support, user training, and support staff.

$14.      **Performance measures and reporting:** Identifies the performance measures required for carrying out software maintenance. It also describes how measures required for enhancing the performance of services (for the software) are recorded and reported.

$15.      **Management approach:** Identifies the methodologies that are required for establishing maintenance priorities of the projects. For this purpose, the management either refers to the existing methodologies or identifies new methodologies. Management approach also describes how users are involved in software maintenance and operations activities as well as how users and project management team communicate with each other.

$16.      **Documentation strategies:** Provides a description of the documentation that is prepared for user reference. Generally, documentation includes reports, information about problems occurring in software, error messages, and the system documentation.

$17.      **Training:** Provides information about the training activities.

$18.      **Acceptance:** Defines a point of agreement between the project management team and software maintenance team after the completion of implementation and transition activities. Once the agreement has been made, the software maintenance begins.

**Staffing Plan**

The staffing plan describes the number of individuals required for a project. It includes selecting and assigning tasks to the project management team members. It provides information about appropriate skills required to perform the tasks to produce the project deliverables and manage the project. In addition, it provides information of resources such as tools, equipment, and processes used by the project management team.

Staff planning is performed by a staff planner, who is responsible for determining the individuals available for the project. Other responsibilities of a staff planner are listed below.

11.      The staff planner determines individuals, who can be from existing staff, staff on contract, or newly employed staff. It is important for the staff planner to know the structure of the organization to determine the availability of staff.

12.      The staff planner determines the skills required to execute the tasks mentioned in the project schedule and task plan. In case staff with required skills is not available, staff planner informs the project manager about the requirements.

13.      The staff planner ensures that the required staff with required skills is available at the right time. For this purpose, the staff planner plans the availability of staff after the project schedule is fixed. For example, at the initial stage of a project, staff may consist of a project manager and a few software engineers whereas during software development, staff consists of software designers as well as the software developers.

14.      The staff planner defines roles and responsibilities of the project management team members so that they can communicate and coordinate with each other according to the tasks assigned to them. Note that the project management team can be further broken down into sub-teams depending on the size and complexity of the project.

The staffing plan comprises the following sections.

11.      **General information:** Provides information such as name of the project and project manager who is responsible for the project. In addition, it specifies the start and end dates of the project.

12.      **Skills assessment:** Provides information, which is required for assessment of skills. This information includes the knowledge, skill, and ability of team members who are required to achieve the objectives of the project. In addition, it specifies the number of team members required for the project.

13.      **Staffing profile:** Describes the profile of the staff required for the project. The profile includes calendar time, individuals involved, and level of commitment. Calendar time specifies the period of time such as month or quarter for which individuals are required to complete the project. Individuals who are involved in the project have specific designations such as project manager and the developer. Level of commitment is the utilization rate of individuals such as work performed on full-time and part-time basis.

14.      **Organization chart:** Describes the organization of project management team members. In addition, it includes information such as name, designation, and role of each team member.