Unit	BCA611: Software Engineering
I	Introduction, Software Engineering, Software Process, Characteristics of Software Process, Development Process Models- waterfall, prototyping, iterative, spiral. Project Management Process, Inspection Process, Software Configuration Management process, Requirement Change Management process.
II	Software Requirement Specification (SRS)- Problem analysis, structuring information, Data flow diagram, entity relationship diagram and data dictionary, structured analysis, Characteristics and component of (SRS).
III	Planning a Software Project- Cost estimation, Single variable model, COCOMO model, software size estimation, Project scheduling and milestones, Verification & Validation. Software Architecture, Role views, Function oriented design – Top down and Bottom up strategies. Coupling, Cohesion. Concept of Object Oriented Analysis and Design
IV	Coding- Standard guideline for coding, Structured Programming, Object oriented programming, Information Hiding, Programming style, Internal Documentation. Testing- Level of testing, Unit testing, Black box & White box testing, Functional Testing, Structural Testing. Testing Process – level of testing, test plan, test case, defect logging and tracking.
V	Software Maintenance: Maintenance as part of software evaluation, reasons for maintenance, types of maintenance (Perceptive, adoptive, corrective), designing for maintainability, techniques for maintenance, case tools, Configuration Management.

Suggested Readings

- I.Sommerville, "Software Engineering", Addison Wesley,

Software engineering is an engineering branch associated with development of software product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product.

Software project management has wider scope than software engineering process as it involves communication, pre and post-delivery support etc.

Let us first understand what software engineering stands for. The term is made of two words, software and engineering.

Software is more than just a program code. A program is an executable code, which serves some computational purpose. Software is considered to be collection of executable programming code, associated libraries and documentations. Software, when made for a specific requirement is called **software product.**

Engineering on the other hand, is all about developing products, using well-defined, scientific principles and methods.

Software engineering is an engineering branch associated with development of software product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product.

Definitions

IEEE defines software engineering as:

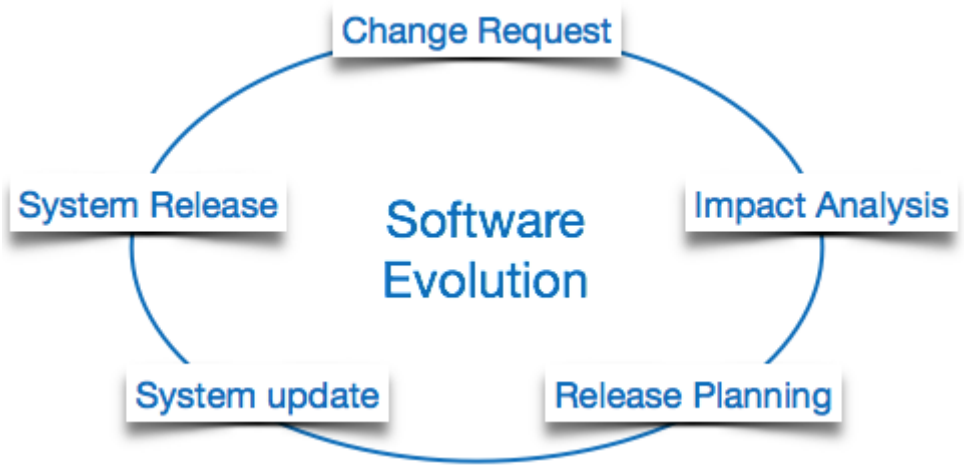
> The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is, the application of engineering to software.

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

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

Software Evolution

The process of developing a software product using software engineering principles and methods is referred to as **software evolution.** This includes the initial development of software and its maintenance and updates, till desired software product is developed, which satisfies the expected requirements.



Evolution starts from the requirement gathering process. After which developers create a prototype of the intended software and show it to the users to get their feedback at the early stage of software product development. The users suggest changes, on which several consecutive updates and maintenance keep on changing too. This process changes to the original software, till the desired software is accomplished.

Even after the user has desired software in hand, the advancing technology and the changing requirements force the software product to change accordingly. Re-creating software from scratch and to go one-on-one with requirement is not feasible. The only feasible and economical solution is to update the existing software so that it matches the latest requirements.

Software Evolution Laws

Lehman has given laws for software evolution. He divided the software into three different categories:

- **S-type (static-type) -** This is a software, which works strictly according to defined specifications and solutions. The solution and the method to achieve it, both are immediately understood before coding. The s-type software is least subjected to changes hence this is the simplest of all. For example, calculator program for mathematical computation.
- **P-type (practical-type) -** This is a software with a collection of procedures. This is defined by exactly what procedures can do. In this software, the specifications can be described but the solution is not obvious instantly. For example, gaming software.
- **E-type (embedded-type) -** This software works closely as the requirement of real-world environment. This software has a high degree of evolution as there are various changes in laws, taxes etc. in the real world situations. For example, Online trading software.

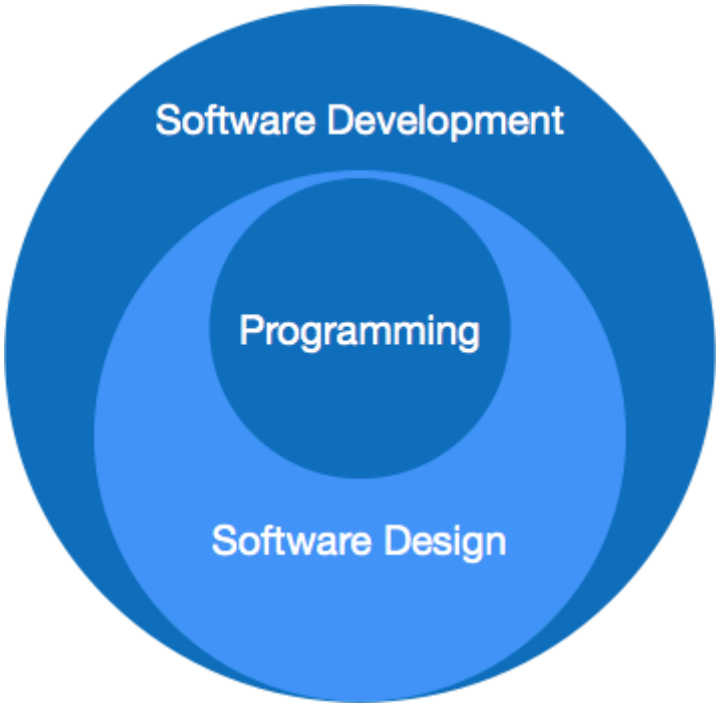
E-Type software evolution

Lehman has given eight laws for E-Type software evolution -

- **Continuing change -** An E-type software system must continue to adapt to the real world changes; else it becomes progressively less useful.
- **Increasing complexity -** As an E-type software system evolves, its complexity tends to increase unless work is done to maintain or reduce it.
- **Conservation of familiarity -** The familiarity with the software or the knowledge about how it was developed, why was it developed in that particular manner etc. must be retained at any cost, to implement the changes in the system.
- **Continuing growth-** In order for an E-type system intended to resolve some business problem, its size of implementing the changes grows according to the lifestyle changes of the business.
- **Reducing quality -** An E-type software system declines in quality unless rigorously maintained and adapted to a changing operational environment.
- **Feedback systems-** The E-type software systems constitute multi-loop, multi-level feedback systems and must be treated as such to be successfully modified or improved.
- **Self-regulation -** E-type system evolution processes are self-regulating with the distribution of product and process measures close to normal.
- **Organizational stability -** The average effective global activity rate in an evolving E-type system is invariant over the lifetime of the product.

Software Paradigms

Software paradigms refer to the methods and steps, which are taken while designing the software. There are many methods proposed and are in work today, but we need to see where in the software engineering these paradigms stand. These can be combined into various categories, though each of them is contained in one another:



Programming paradigm is a subset of Software design paradigm which is further a subset of Software development paradigm.

Software Development Paradigm

This Paradigm is known as software engineering paradigms where all the engineering concepts pertaining to the development of software are applied. It includes various researches and requirement gathering which helps the software product to build. It consists of –

- Requirement gathering
- Software design
- Programming

Software Design Paradigm

This paradigm is a part of Software Development and includes –

- Design
- Maintenance
- Programming

Programming Paradigm

This paradigm is related closely to programming aspect of software development. This includes –

- Coding

- Testing
- Integration

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 the 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.

Characteristics of good software

A software product can be judged by what it offers and how well it can be used. This software must satisfy on the following grounds:

- Operational
- Transitional
- Maintenance

Well-engineered and crafted software is expected to have the following characteristics:

Operational

This tells us how well software works in operations. It can be measured on:

- Budget
- Usability
- Efficiency
- Correctness
- Functionality
- Dependability

- Security
- Safety

Transitional

This aspect is important when the software is moved from one platform to another:

- Portability
- Interoperability
- Reusability
- Adaptability

Maintenance

This aspect briefs about how well a software has the capabilities to maintain itself in the ever-changing environment:

- Modularity
- Maintainability
- Flexibility
- Scalability

In short, Software engineering is a branch of computer science, which uses well-defined engineering concepts required to produce efficient, durable, scalable, in-budget and on-time software products.

Software Development Life Cycle, SDLC for short, is a well-defined, structured sequence of stages in software engineering to develop the intended software product.

SDLC Activities

SDLC provides a series of steps to be followed to design and develop a software product efficiently. SDLC framework includes the following steps:

Communication

This is the first step where the user initiates the request for a desired software product. He contacts the service provider and tries to negotiate the terms. He submits his request to the service providing organization in writing.

Requirement Gathering

This step onwards the software development team works to carry on the project. The team holds discussions with various stakeholders from problem domain and tries to bring out as much information as possible on their requirements. The requirements are contemplated and segregated into user requirements, system requirements and functional requirements. The requirements are collected using a number of practices as given -

- studying the existing or obsolete system and software,
- conducting interviews of users and developers,
- referring to the database or
- collecting answers from the questionnaires.

Feasibility Study

After requirement gathering, the team comes up with a rough plan of software process. At this step the team analyses if software can be made to fulfil all requirements of the user and if there is any possibility of software being no more useful. It is found out, if the project is financially, practically and technologically feasible for the organization to take up. There are many algorithms available, which help the developers to conclude the feasibility of a software project.

System Analysis

At this step the developers decide a roadmap of their plan and try to bring up the best software model suitable for the project. System analysis includes Understanding of software product limitations, learning system related problems or changes to be done in existing systems beforehand, identifying and addressing the impact of project on organization and personnel etc. The project team analyzes the scope of the project and plans the schedule and resources accordingly.

Software Design

Next step is to bring down whole knowledge of requirements and analysis on the desk and design the software product. The inputs from users and information gathered in requirement gathering phase are the inputs of this step. The output of this step comes in the form of two designs; logical design and physical design. Engineers produce meta-data and data dictionaries, logical diagrams, data-flow diagrams and in some cases pseudo codes.

Coding

This step is also known as programming phase. The implementation of software design starts in terms of writing program code in the suitable programming language and developing error-free executable programs efficiently.

Testing

An estimate says that 50% of whole software development process should be tested. Errors may ruin the software from critical level to its own removal. 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 end. Early discovery of errors and their remedy is the key to reliable software.

Integration

Software may need to be integrated with the libraries, databases and other program(s). This stage of SDLC is involved in the integration of software with outer world entities.

Implementation

This means installing the software on user machines. At times, software needs post-installation configurations at user end. Software is tested for portability and adaptability and integration related issues are solved during implementation.

Operation and Maintenance

This phase confirms the software operation in terms of more efficiency and less errors. If required, the users are trained on, or aided with the documentation on how to operate the software and how to keep the software operational. The software is maintained timely by updating the code according to the changes taking place in user end environment or technology. This phase may face challenges from hidden bugs and real-world unidentified problems.

Disposition

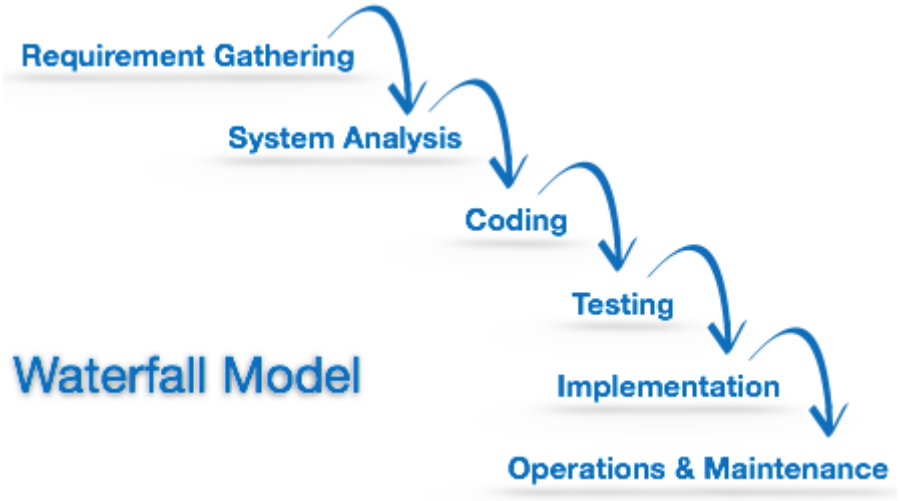
As time elapses, the software may decline on the performance front. It may go completely obsolete or may need intense up gradation. Hence a pressing need to eliminate a major portion of the system arises. This phase includes archiving data and required software components, closing down the system, planning disposition activity and terminating system at appropriate end-of-system time.

Software Development Paradigm

The software development paradigm helps developer to select a strategy to develop the software. A software development paradigm has its own set of tools, methods and procedures, which are expressed clearly and defines software development life cycle. A few of software development paradigms or process models are defined as follows:

Waterfall Model

Waterfall model is the simplest model of software development paradigm. It says the all the phases of SDLC will function one after another in linear manner. That is, when the first phase is finished then only the second phase will start and so on.

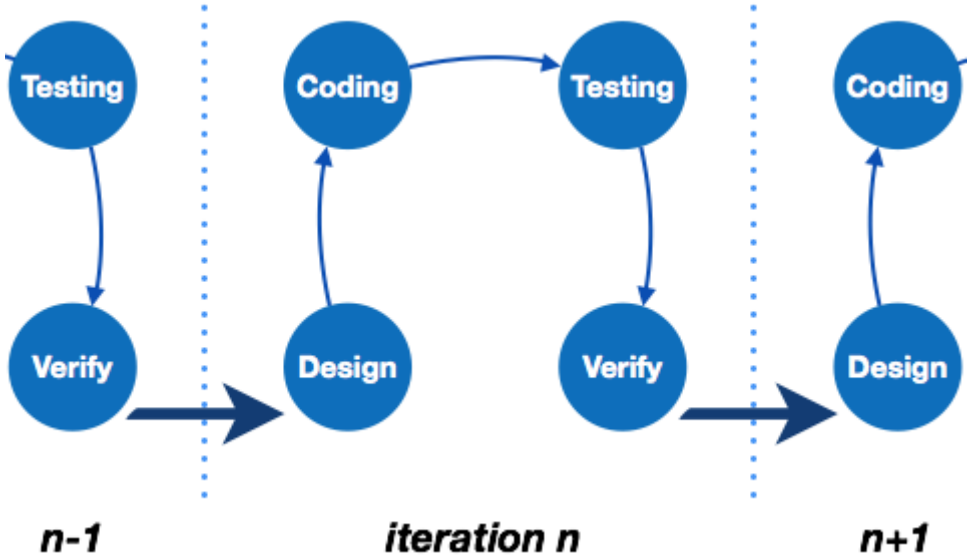


This model assumes that everything is carried out and taken place perfectly as planned in the previous stage and there is no need to think about the past issues that may arise in the next phase. This model does not work smoothly if there are some issues left at the previous step. The sequential nature of model does not allow us go back and undo or redo our actions.

This model is best suited when developers already have designed and developed similar software in the past and are aware of all its domains.

Iterative Model

This model leads the software development process in iterations. It projects the process of development in cyclic manner repeating every step after every cycle of SDLC process.

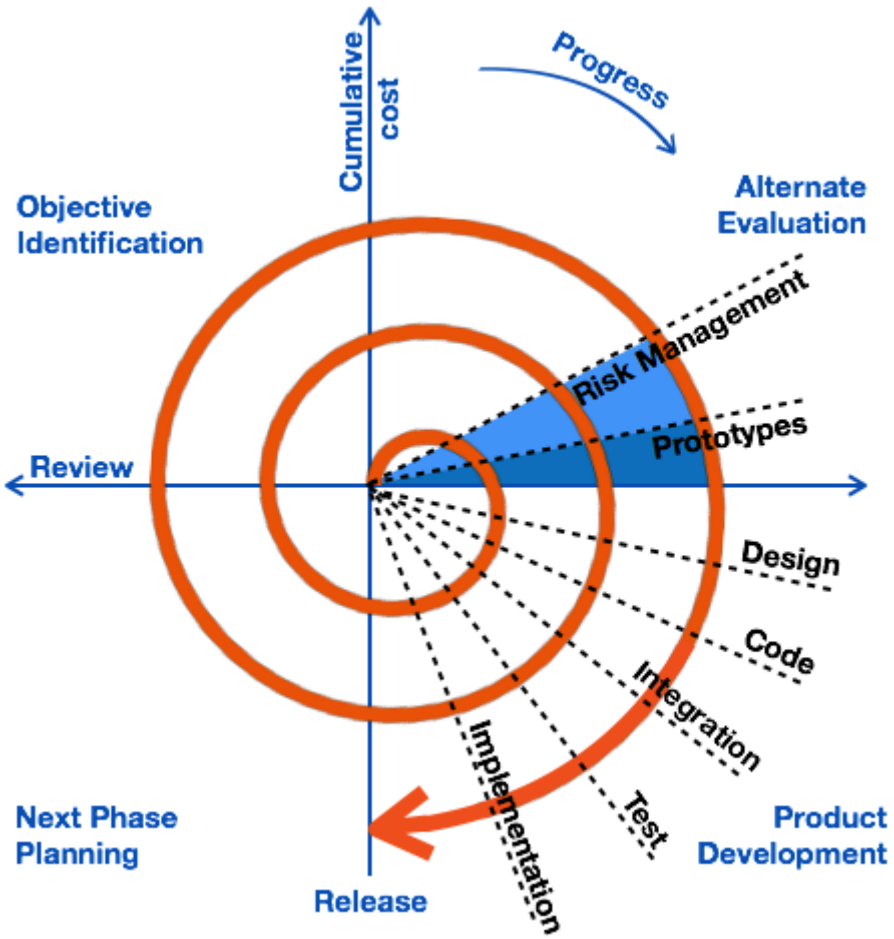


The software is first developed on very small scale and all the steps are followed which are taken into consideration. Then, on every next iteration, more features and modules are designed, coded, tested and added to the software. Every cycle produces a software, which is complete in itself and has more features and capabilities than that of the previous one.

After each iteration, the management team can do work on risk management and prepare for the next iteration. Because a cycle includes small portion of whole software process, it is easier to manage the development process but it consumes more resources.

Spiral Model

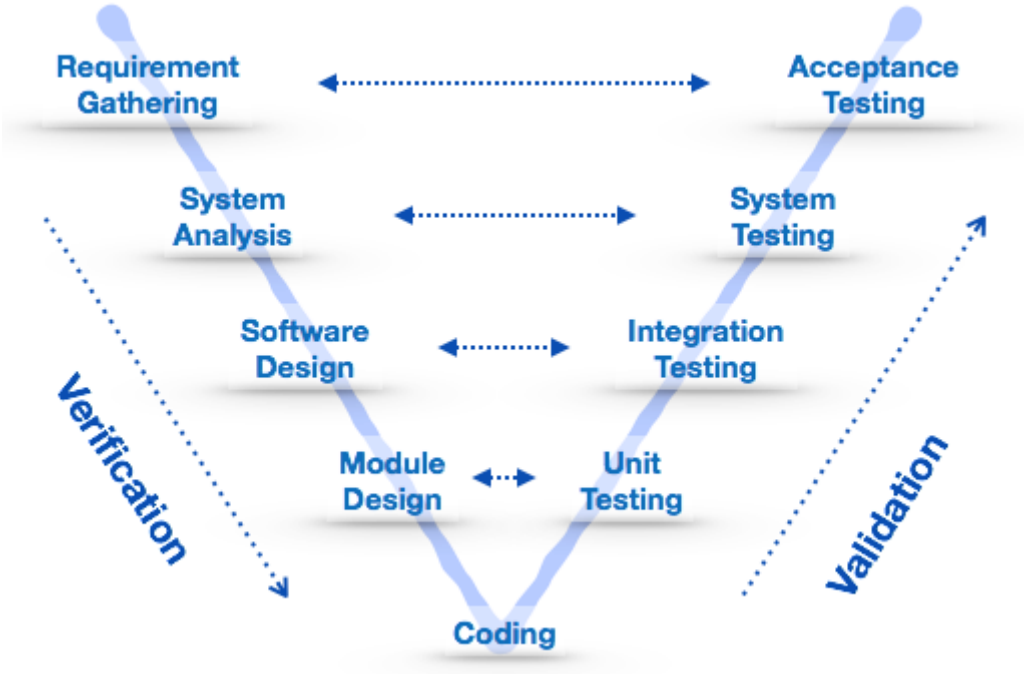
Spiral model is a combination of both, iterative model and one of the SDLC model. It can be seen as if you choose one SDLC model and combine it with cyclic process (iterative model).



This model considers risk, which often goes un-noticed by most other models. The model starts with determining objectives and constraints of the software at the start of one iteration. Next phase is of prototyping the software. This includes risk analysis. Then one standard SDLC model is used to build the software. In the fourth phase of the plan of next iteration is prepared.

V – model

The major drawback of waterfall model is we move to the next stage only when the previous one is finished and there was no chance to go back if something is found wrong in later stages. V-Model provides means of testing of software at each stage in reverse manner.

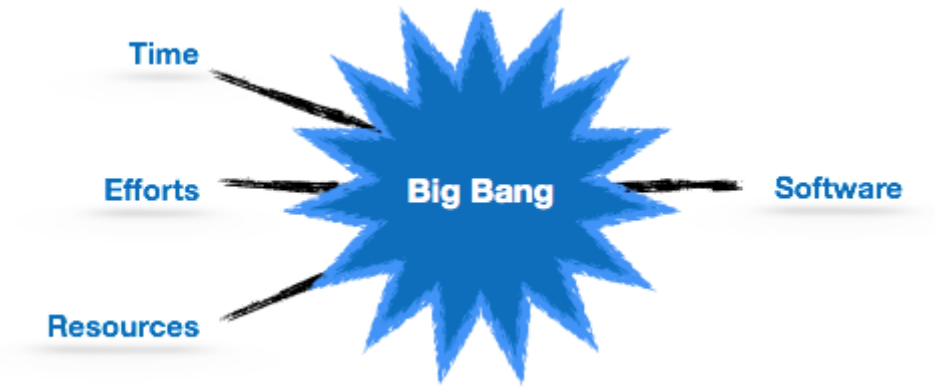


At every stage, test plans and test cases are created to verify and validate the product according to the requirement of that stage. For example, in requirement gathering stage the test team prepares all the test cases in correspondence to the requirements. Later, when the product is developed and is ready for testing, test cases of this stage verify the software against its validity towards requirements at this stage.

This makes both verification and validation go in parallel. This model is also known as verification and validation model.

Big Bang Model

This model is the simplest model in its form. It requires little planning, lots of programming and lots of funds. This model is conceptualized around the big bang of universe. As scientists say that after big bang lots of galaxies, planets and stars evolved just as an event. Likewise, if we put together lots of programming and funds, you may achieve the best software product.



For this model, very small amount of planning is required. It does not follow any process, or at times the customer is not sure about the requirements and future needs. So the input requirements are arbitrary.

This model is not suitable for large software projects but good one for learning and experimenting.

Software Project Management

The job pattern of an IT company engaged in software development can be seen split in two parts:

- Software Creation
- Software Project Management

A project is well-defined task, which is a collection of several operations done in order to achieve a goal (for example, software development and delivery). A Project can be characterized as:

- Every project may has a unique and distinct goal.
- Project is not routine activity or day-to-day operations.
- Project comes with a start time and end time.
- Project ends when its goal is achieved hence it is a temporary phase in the lifetime of an organization.
- Project needs adequate resources in terms of time, manpower, finance, material and knowledge-bank.

Software Project

A Software Project is the complete procedure of software development from requirement gathering to testing and maintenance, carried out according to the execution methodologies, in a specified period of time to achieve intended software product.

Need of software project management

Software is said to be an intangible product. Software development is a kind of all new stream in world business and there’s very little experience in building software products. Most software products are tailor made to fit client’s requirements. The most important is that the underlying technology changes and advances so frequently and rapidly that experience of one product may not be applied to the other one. All such business and environmental constraints bring risk in software development hence it is essential to manage software projects efficiently.

The image above shows triple constraints for software projects. It is an essential part of software organization to deliver quality product, keeping the cost within client's budget constrain and deliver the project as per scheduled. There are several factors, both internal and external, which may impact this triple constrain triangle. Any of three factor can severely impact the other two.

Therefore, software project management is essential to incorporate user requirements along with budget and time constraints.

Software Project Manager

A software project manager is a person who undertakes the responsibility of executing the software project. Software project manager is thoroughly aware of all the phases of SDLC that the software would go through. Project manager may never directly involve in producing the end product but he controls and manages the activities involved in production.

A project manager closely monitors the development process, prepares and executes various plans, arranges necessary and adequate resources, maintains communication among all team members in order to address issues of cost, budget, resources, time, quality and customer satisfaction.

Let us see few responsibilities that a project manager shoulders -

Managing People

- Act as project leader
- Liaison with stakeholders
- Managing human resources
- Setting up reporting hierarchy etc.

Managing Project

- Defining and setting up project scope
- Managing project management activities
- Monitoring progress and performance
- Risk analysis at every phase
- Take necessary step to avoid or come out of problems

- Act as project spokesperson

Software Management Activities

Software project management comprises of a number of activities, which contains planning of project, deciding scope of software product, estimation of cost in various terms, scheduling of tasks and events, and resource management. Project management activities may include:

- **Project Planning**
- **Scope Management**
- **Project Estimation**

Project Planning

Software project planning is task, which is performed before the production of software actually starts. It is there for the software production but involves no concrete activity that has any direction connection with software production; rather it is a set of multiple processes, which facilitates software production. Project planning may include the following:

Scope Management

It defines the scope of project; this includes all the activities, process need to be done in order to make a deliverable software product. Scope management is essential because it creates boundaries of the project by clearly defining what would be done in the project and what would not be done. This makes project to contain limited and quantifiable tasks, which can easily be documented and in turn avoids cost and time overrun.

During Project Scope management, it is necessary to -

- Define the scope
- Decide its verification and control
- Divide the project into various smaller parts for ease of management.
- Verify the scope
- Control the scope by incorporating changes to the scope

Project Estimation

For an effective management accurate estimation of various measures is a must. With correct estimation managers can manage and control the project more efficiently and effectively.

Project estimation may involve the following:

- **Software size estimation**

Software size may be estimated either in terms of KLOC (Kilo Line of Code) or by calculating number of function points in the software. Lines of code depend upon coding practices and Function points vary according to the user or software requirement.

- **Effort estimation**

  The managers estimate efforts in terms of personnel requirement and man-hour required to produce the software. For effort estimation software size should be known. This can either be derived by managers' experience, organization's historical data or software size can be converted into efforts by using some standard formulae.

- **Time estimation**

  Once size and efforts are estimated, the time required to produce the software can be estimated. Efforts required is segregated into sub categories as per the requirement specifications and interdependency of various components of software. Software tasks are divided into smaller tasks, activities or events by Work Breakthrough Structure (WBS). The tasks are scheduled on day-to-day basis or in calendar months.

  The sum of time required to complete all tasks in hours or days is the total time invested to complete the project.

- **Cost estimation**

  This might be considered as the most difficult of all because it depends on more elements than any of the previous ones. For estimating project cost, it is required to consider -

  - Size of software
  - Software quality
  - Hardware
  - Additional software or tools, licenses etc.
  - Skilled personnel with task-specific skills
  - Travel involved
  - Communication
  - Training and support

Project Estimation Techniques

We discussed various parameters involving project estimation such as size, effort, time and cost.

Project manager can estimate the listed factors using two broadly recognized techniques –

Decomposition Technique

This technique assumes the software as a product of various compositions.

There are two main models -

- **Line of Code** Estimation is done on behalf of number of line of codes in the software product.
- **Function Points** Estimation is done on behalf of number of function points in the software product.

Empirical Estimation Technique

This technique uses empirically derived formulae to make estimation. These formulae are based on LOC or FPs.

- **Putnam Model**

  This model is made by Lawrence H. Putnam, which is based on Norden's frequency distribution (Rayleigh curve). Putnam model maps time and efforts required with software size.

- **COCOMO**

  COCOMO stands for COnstructive COst MOdel, developed by Barry W. Boehm. It divides the software product into three categories of software: organic, semi-detached and embedded.

Project Scheduling

Project Scheduling in a project refers to roadmap of all activities to be done with specified order and within time slot allotted to each activity. Project managers tend to define various tasks, and project milestones and arrange them keeping various factors in mind. They look for tasks lie in critical path in the schedule, which are necessary to complete in specific manner (because of task interdependency) and strictly within the time allocated. Arrangement of tasks which lies out of critical path are less likely to impact over all schedule of the project.

For scheduling a project, it is necessary to -

- Break down the project tasks into smaller, manageable form
- Find out various tasks and correlate them
- Estimate time frame required for each task
- Divide time into work-units
- Assign adequate number of work-units for each task
- Calculate total time required for the project from start to finish

Resource management

All elements used to develop a software product may be assumed as resource for that project. This may include human resource, productive tools and software libraries.

The resources are available in limited quantity and stay in the organization as a pool of assets. The shortage of resources hampers the development of project and it can lag behind the schedule. Allocating extra resources increases development cost in the end. It is therefore necessary to estimate and allocate adequate resources for the project.

Resource management includes -

- Defining proper organization project by creating a project team and allocating responsibilities to each team member
- Determining resources required at a particular stage and their availability
- Manage Resources by generating resource request when they are required and de-allocating them when they are no more needed.

Project Risk Management

Risk management involves all activities pertaining to identification, analyzing and making provision for predictable and non-predictable risks in the project. Risk may include the following:

- Experienced staff leaving the project and new staff coming in.
- Change in organizational management.
- Requirement change or misinterpreting requirement.
- Under-estimation of required time and resources.
- Technological changes, environmental changes, business competition.

Risk Management Process

There are following activities involved in risk management process:

- **Identification -** Make note of all possible risks, which may occur in the project.
- **Categorize -** Categorize known risks into high, medium and low risk intensity as per their possible impact on the project.
- **Manage -** Analyze the probability of occurrence of risks at various phases. Make plan to avoid or face risks. Attempt to minimize their side-effects.
- **Monitor -** Closely monitor the potential risks and their early symptoms. Also monitor the effects of steps taken to mitigate or avoid them.

Project Execution & Monitoring

In this phase, the tasks described in project plans are executed according to their schedules.

Execution needs monitoring in order to check whether everything is going according to the plan. Monitoring is observing to check the probability of risk and taking measures to address the risk or report the status of various tasks.

These measures include -

- **Activity Monitoring -** All activities scheduled within some task can be monitored on day-to-day basis. When all activities in a task are completed, it is considered as complete.
- **Status Reports -** The reports contain status of activities and tasks completed within a given time frame, generally a week. Status can be marked as finished, pending or work-in-progress etc.
- **Milestones Checklist -** Every project is divided into multiple phases where major tasks are performed (milestones) based on the phases of SDLC. This milestone checklist is prepared once every few weeks and reports the status of milestones.

Project Communication Management

Effective communication plays vital role in the success of a project. It bridges gaps between client and the organization, among the team members as well as other stake holders in the project such as hardware suppliers.

Communication can be oral or written. Communication management process may have the following steps:

- **Planning** - This step includes the identifications of all the stakeholders in the project and the mode of communication among them. It also considers if any additional communication facilities are required.
- **Sharing** - After determining various aspects of planning, manager focuses on sharing correct information with the correct person on correct time. This keeps every one involved the project up to date with project progress and its status.
- **Feedback** - Project managers use various measures and feedback mechanism and create status and performance reports. This mechanism ensures that input from various stakeholders is coming to the project manager as their feedback.
- **Closure** - At the end of each major event, end of a phase of SDLC or end of the project itself, administrative closure is formally announced to update every stakeholder by sending email, by distributing a hardcopy of document or by other mean of effective communication.

After closure, the team moves to next phase or project.

Configuration Management

Configuration management is a process of tracking and controlling the changes in software in terms of the requirements, design, functions and development of the product.

IEEE defines it as “the process of identifying and defining the items in the system, controlling the change of these items throughout their life cycle, recording and reporting the status of items and change requests, and verifying the completeness and correctness of items”.

Generally, once the SRS is finalized there is less chance of requirement of changes from user. If they occur, the changes are addressed only with prior approval of higher management, as there is a possibility of cost and time overrun.

Baseline

A phase of SDLC is assumed over if it baselined, i.e. baseline is a measurement that defines completeness of a phase. A phase is baselined when all activities pertaining to it are finished and well documented. If it was not the final phase, its output would be used in next immediate phase.

Configuration management is a discipline of organization administration, which takes care of occurrence of any change (process, requirement, technological, strategical etc.) after a phase is baselined. CM keeps check on any changes done in software.

Change Control

Change control is function of configuration management, which ensures that all changes made to software system are consistent and made as per organizational rules and regulations.

A change in the configuration of product goes through following steps -

- **Identification** - A change request arrives from either internal or external source. When change request is identified formally, it is properly documented.
- **Validation** - Validity of the change request is checked and its handling procedure is confirmed.
- **Analysis** - The impact of change request is analyzed in terms of schedule, cost and required efforts. Overall impact of the prospective change on system is analyzed.
- **Control** - If the prospective change either impacts too many entities in the system or it is unavoidable, it is mandatory to take approval of high authorities before change is incorporated into the system. It is decided if the change is worth incorporation or not. If it is not, change request is refused formally.
- **Execution** - If the previous phase determines to execute the change request, this phase take appropriate actions to execute the change, does a thorough revision if necessary.
- **Close request** - The change is verified for correct implementation and merging with the rest of the system. This newly incorporated change in the software is documented properly and the request is formally is closed.

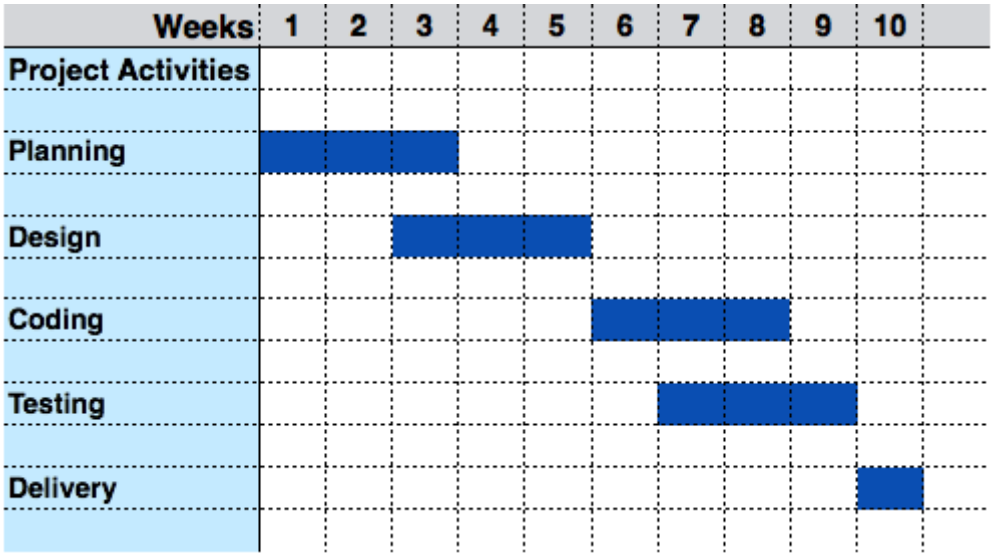
Project Management Tools

The risk and uncertainty rises multifold with respect to the size of the project, even when the project is developed according to set methodologies.

There are tools available, which aid for effective project management. A few are described -

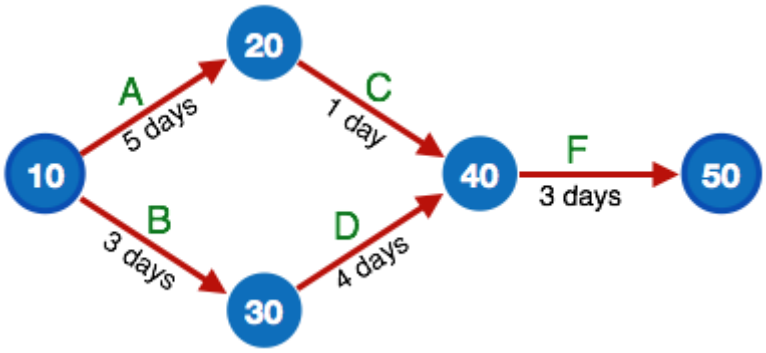
Gantt Chart

Gantt charts was devised by Henry Gantt (1917). It represents project schedule with respect to time periods. It is a horizontal bar chart with bars representing activities and time scheduled for the project activities.



PERT Chart

PERT (Program Evaluation & Review Technique) chart is a tool that depicts project as network diagram. It is capable of graphically representing main events of project in both parallel and consecutive way. Events, which occur one after another, show dependency of the later event over the previous one.

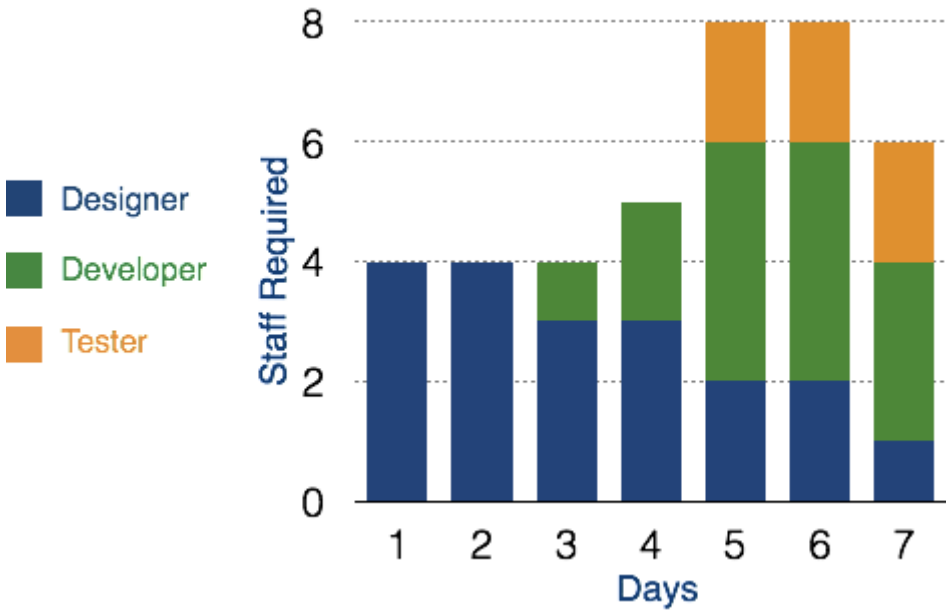


Events are shown as numbered nodes. They are connected by labeled arrows depicting sequence of tasks in the project.

Resource Histogram

This is a graphical tool that contains bar or chart representing number of resources (usually skilled staff) required over time for a project event (or phase). Resource Histogram is an effective tool for staff planning and coordination.

Staff	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7
Designer	4	4	3	3	2	2	1
Developer	0	0	1	2	4	4	3
Tester	0	0	0	0	2	2	2
Total	4	4	4	5	8	8	6



Critical Path Analysis

This tools is useful in recognizing interdependent tasks in the project. It also helps to find out the shortest path or critical path to complete the project successfully. Like PERT diagram, each event is allotted a specific time frame. This tool shows dependency of event assuming an event can proceed to next only if the previous one is completed.

The events are arranged according to their earliest possible start time. Path between start and end node is critical path which cannot be further reduced and all events require to be executed in same order.

- Credible source

Software Requirements

We should try to understand what sort of requirements may arise in the requirement elicitation phase and what kinds of requirements are expected from the software system.

Broadly software requirements should be categorized in two categories:

Functional Requirements

Requirements, which are related to functional aspect of software fall into this category.

They define functions and functionality within and from the software system.

Examples -

- Search option given to user to search from various invoices.
- User should be able to mail any report to management.
- Users can be divided into groups and groups can be given separate rights.
- Should comply business rules and administrative functions.
- Software is developed keeping downward compatibility intact.

Non-Functional Requirements

Requirements, which are not related to functional aspect of software, fall into this category. They are implicit or expected characteristics of software, which users make assumption of.

Non-functional requirements include -

- Security
- Logging
- Storage
- Configuration
- Performance
- Cost
- Interoperability
- Flexibility
- Disaster recovery
- Accessibility

Requirements are categorized logically as

- **Must Have** : Software cannot be said operational without them.
- **Should have** : Enhancing the functionality of software.
- **Could have** : Software can still properly function with these requirements.
- **Wish list** : These requirements do not map to any objectives of software.

While developing software, ‘Must have’ must be implemented, ‘Should have’ is a matter of debate with stakeholders and negation, whereas ‘could have’ and ‘wish list’ can be kept for software updates.

User Interface requirements

UI is an important part of any software or hardware or hybrid system. A software is widely accepted if it is -

- easy to operate
- quick in response
- effectively handling operational errors
- providing simple yet consistent user interface

User acceptance majorly depends upon how user can use the software. UI is the only way for users to perceive the system. A well performing software system must also be equipped with attractive, clear, consistent and responsive user interface. Otherwise the functionalities of software system cannot be used in convenient way. A system is said be good if it provides means to use it efficiently. User interface requirements are briefly mentioned below -

- Content presentation
- Easy Navigation
- Simple interface
- Responsive
- Consistent UI elements
- Feedback mechanism
- Default settings
- Purposeful layout
- Strategically use of color and texture.
- Provide help information
- User centric approach
- Group based view settings.

Software System Analyst

System analyst in an IT organization is a person, who analyses the requirement of proposed system and ensures that requirements are conceived and documented properly & correctly. Role of an analyst starts during Software Analysis Phase of SDLC. It is the responsibility of analyst to make sure that the developed software meets the requirements of the client.

System Analysts have the following responsibilities:

- Analysing and understanding requirements of intended software
- Understanding how the project will contribute in the organization objectives
- Identify sources of requirement

- Validation of requirement
- Develop and implement requirement management plan
- Documentation of business, technical, process and product requirements
- Coordination with clients to prioritize requirements and remove and ambiguity
- Finalizing acceptance criteria with client and other stakeholders

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.