Concept of system:

What is a System?

The word System is derived from Greek word Systema, which means an organized relationship between any set of components to achieve some common cause or objective.

A system is "an orderly grouping of interdependent components linked together according to a plan to achieve a specific goal."

Constraints of a System

A system must have three basic constraints −

- A system must have some **structure and behavior** which is designed to achieve a predefined objective.
- **Interconnectivity** and **interdependence** must exist among the system components.
- The **objectives of the organization** have a **higher priority** than the objectives of its subsystems.

For example, traffic management system, payroll system, automatic library system, human resources information system.

BASIS OF COMPARISON	PROGRAM	SOFTWARE
Description	Program is a set of instructions written in a programming language used to execute for a specific task or particular function.	Program is a set of instructions written in a programming language used to execute for a specific task or particular function.
Categories	A program does not have further categorization.	Software can be categorized into two categories: application software and system software.
Flexibility	A program cannot be software.	Software can be a program.
Consists Of	A program consists of a set of instructions which are coded in a programming language like c, C++, PHP, Java etc.	Software consists of bundles of programs and data files. Programs in specific software use these data files to perform a dedicated type of tasks.
User Interface	Programs do not have a user interface.	Every software has a dedicated user interface.
Development	A program is developed and used by either a single programmer or a group of programmers.	Software is developed by either a single programmer or a group of programmers.
Compilation	A program is compiled every time when we need to generate some output from it.	Whole software is compiled, tested and debugged during the development process.

Functionality & Features	Program has limited functionality and less features.	Software has lots of functionality and features such as GUI, input/output data, process etc.
Dependability	Program functionality is dependent on compiler.	Software functionality is dependent on the operating system.
Creation Time	A program takes less time to build/make.	Software takes relatively more time to build/make when compared to program.
Development Approach	Program development approach is un-procedural, un-organized and unplanned.	Software development approach is systematic, organized and very well planned.
Size	The size of a program ranges from kilobytes (Kb) to megabytes (Mb).	The size of a software ranges from megabytes (Mb) to Gigabytes (Gb).
Examples	Operating system, office suite, video games, malware, a web browser like Mozilla Firefox and Apple Safari.	Microsoft Word, Microsoft Excel, VLC media player, Firefox, Adobe Reader, Windows, Linux, Unix, Mac etc.

Emergence of Software Engineering

Software engineering discipline is the result of advancement in the field of technology. In this section, we will discuss various innovations and technologies that led to the emergence of software engineering discipline.

Early Computer Programming

As we know that in the early 1950s, computers were slow and expensive. Though the programs at that time were very small in size, these computers took considerable time to process them. They relied on assembly language which was specific to computer architecture. Thus, developing a program required lot of effort. Every programmer used his own style to develop the programs.

High Level Language Programming

With the introduction of semiconductor technology, the computers became smaller, faster, cheaper, and reliable than their predecessors. One of the major developments includes the progress from assembly language to high-level languages. Early high level programming languages such as COBOL and FORTRAN came into existence. As a result, the programming became easier and thus, increased the productivity of the programmers. However, still the programs were limited in size and the programmers developed programs using their own style and experience.

Control Flow Based Design

With the advent of powerful machines and high level languages, the usage of computers grew rapidly: In addition, the nature of programs also changed from simple to complex. The increased size and the complexity could not be managed by individual style. It was analyzed that clarity of control flow (the sequence in which the program's instructions are executed) is of great importance. To help the programmer to design programs having good control flow structure, **flowcharting technique** was developed. In flowcharting technique, the algorithm is represented using flowcharts. A **flowchart** is a graphical representation that depicts the sequence of operations to be carried out to solve a given problem.

Note that having more GOTO constructs in the flowchart makes the control flow messy, which makes it difficult to understand and debug. In order to provide clarity of control flow, the use of

GOTO constructs in flowcharts should be avoided and **structured constructs-decision,** sequence, and loop-should be used to develop **structured flowcharts.** The decision structures are used for conditional execution of statements (for example, if statement). The sequence structures are used for the sequentially executed statements. The loop structures are used for performing some repetitive tasks in the program. The use of structured constructs formed the basis of the **structured programming** methodology.

Structured programming became a powerful tool that allowed programmers to write moderately complex programs easily. It forces a logical structure in the program to be written in an efficient and understandable manner. The purpose of structured programming is to make the software code easy to modify when required. Some languages such as Ada, Pascal, and dBase are designed with features that implement the logical program structure in the software code.

Data-Flow Oriented Design

With the introduction of very Large Scale Integrated circuits (VLSI), the computers became more powerful and faster. As a result, various significant developments like networking and GUIs came into being. Clearly, the complexity of software could not be dealt using control flow based design. Thus, a new technique, namely, **data-flow-oriented** technique came into existence. In this technique, the flow of data through business functions or processes is represented using **Data-flow Diagram (DFD). IEEE** defines a data-flow diagram (also known as **bubble chart** and **work-flow diagram)** as 'a diagram that depicts data sources, data sinks, data storage, and processes performed on data as nodes, and logical flow of data as links between the nodes.'

Object Oriented Design

Object-oriented design technique has revolutionized the process of software development. It not only includes the best features of structured programming but also some new and powerful features such as encapsulation, abstraction, inheritance, and polymorphism. These new features have tremendously helped in the development of well-designed and high-quality software. Object-oriented techniques are widely used these days as they allow reusability of the code. They lead to faster software development and high-quality programs. Moreover, they are easier to adapt and scale, that is, large systems can be created by assembling reusable subsystems.

Unit-2

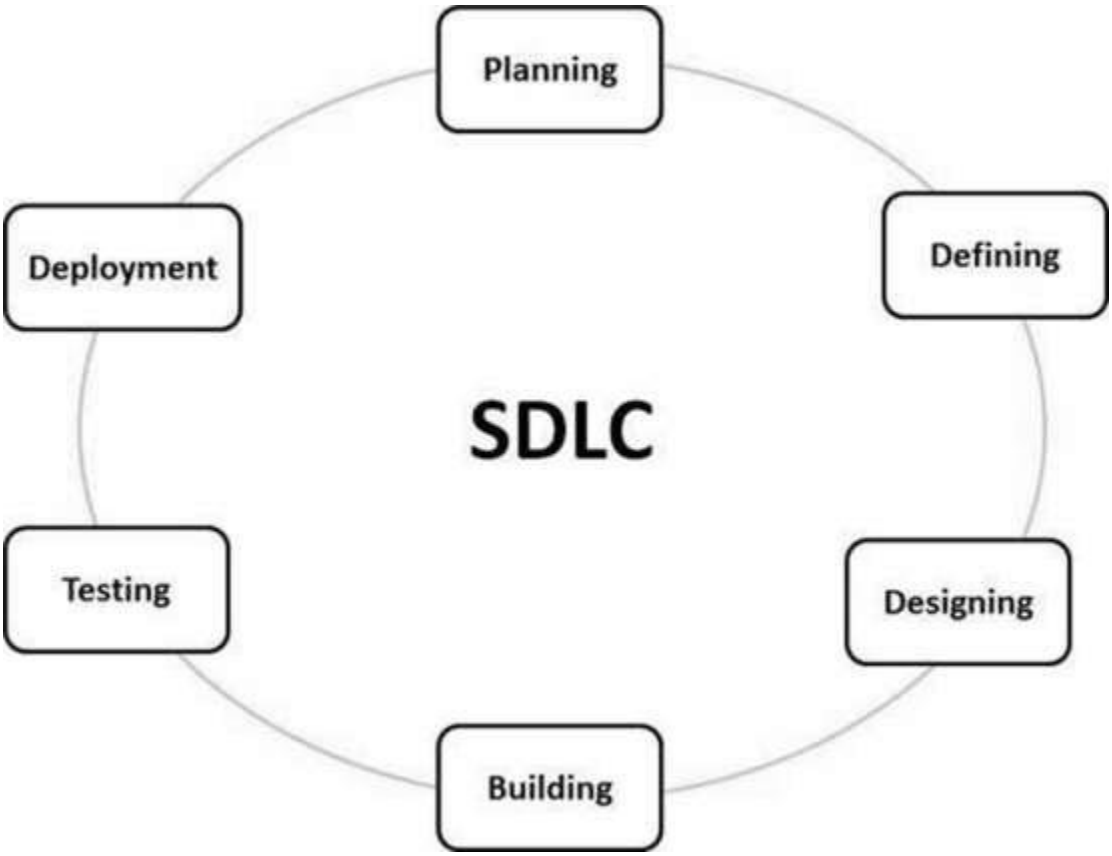
Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality softwares. The SDLC aims to produce a high-quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates.

- SDLC is the acronym of Software Development Life Cycle.
- It is also called as Software Development Process.
- SDLC is a framework defining tasks performed at each step in the software development process.
- ISO/IEC 12207 is an international standard for software life-cycle processes. It aims to be the standard that defines all the tasks required for developing and maintaining software.

What is SDLC?

SDLC is a process followed for a software project, within a software organization. It consists of a detailed plan describing how to develop, maintain, replace and alter or enhance specific software. The life cycle defines a methodology for improving the quality of software and the overall development process.

The following figure is a graphical representation of the various stages of a typical SDLC.



A typical Software Development Life Cycle consists of the following stages −

Stage 1: Planning and Requirement Analysis

Requirement analysis is the most important and fundamental stage in SDLC. It is performed by the senior members of the team with inputs from the customer, the sales department, market surveys and domain experts in the industry. This information is then used to plan the basic project approach and to conduct product feasibility study in the economical, operational and technical areas.

Planning for the quality assurance requirements and identification of the risks associated with the project is also done in the planning stage. The outcome of the technical feasibility study is to define the various technical approaches that can be followed to implement the project successfully with minimum risks.

Stage 2: Defining Requirements

Once the requirement analysis is done the next step is to clearly define and document the product requirements and get them approved from the customer or the market analysts. This is done

through an **SRS (Software Requirement Specification)** document which consists of all the product requirements to be designed and developed during the project life cycle.

Stage 3: Designing the Product Architecture

SRS is the reference for product architects to come out with the best architecture for the product to be developed. Based on the requirements specified in SRS, usually more than one design approach for the product architecture is proposed and documented in a DDS - Design Document Specification.

This DDS is reviewed by all the important stakeholders and based on various parameters as risk assessment, product robustness, design modularity, budget and time constraints, the best design approach is selected for the product.

A design approach clearly defines all the architectural modules of the product along with its communication and data flow representation with the external and third party modules (if any). The internal design of all the modules of the proposed architecture should be clearly defined with the minutest of the details in DDS.

Stage 4: Building or Developing the Product

In this stage of SDLC the actual development starts and the product is built. The programming code is generated as per DDS during this stage. If the design is performed in a detailed and organized manner, code generation can be accomplished without much hassle.

Developers must follow the coding guidelines defined by their organization and programming tools like compilers, interpreters, debuggers, etc. are used to generate the code. Different high level programming languages such as C, C++, Pascal, Java and PHP are used for coding. The programming language is chosen with respect to the type of software being developed.

Stage 5: Testing the Product

This stage is usually a subset of all the stages as in the modern SDLC models, the testing activities are mostly involved in all the stages of SDLC. However, this stage refers to the testing only stage of the product where product defects are reported, tracked, fixed and retested, until the product reaches the quality standards defined in the SRS.

Stage 6: Deployment in the Market and Maintenance

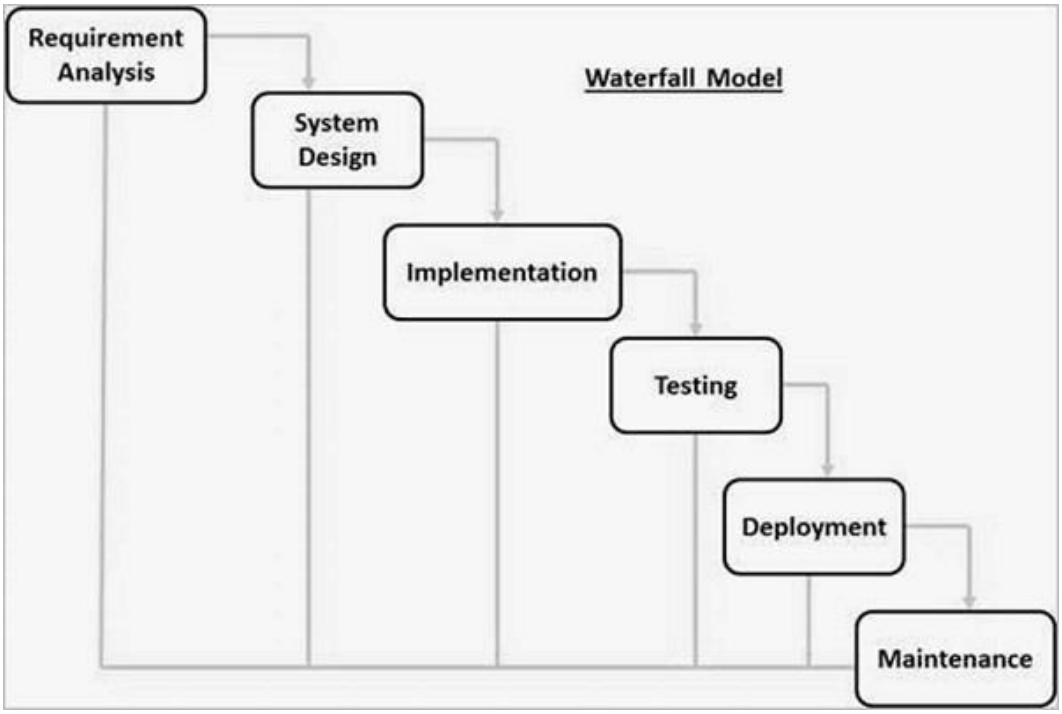
Once the product is tested and ready to be deployed it is released formally in the appropriate market. Sometimes product deployment happens in stages as per the business strategy of that organization. The product may first be released in a limited segment and tested in the real business environment (UAT- User acceptance testing).

Then based on the feedback, the product may be released as it is or with suggested enhancements in the targeting market segment. After the product is released in the market, its maintenance is done for the existing customer base.

Waterfall Model - Design

Waterfall approach was first SDLC Model to be used widely in Software Engineering to ensure success of the project. In "The Waterfall" approach, the whole process of software development is divided into separate phases. In this Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.

The following illustration is a representation of the different phases of the Waterfall Model.



The sequential phases in Waterfall model are −

- **Requirement Gathering and analysis** − All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.
- **System Design** − The requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.
- **Implementation** − With inputs from the system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.
- **Integration and Testing** − All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.
- **Deployment of system** − Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.
- **Maintenance** − There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

All these phases are cascaded to each other in which progress is seen as flowing steadily downwards (like a waterfall) through the phases. The next phase is started only after the defined

set of goals are achieved for previous phase and it is signed off, so the name "Waterfall Model". In this model, phases do not overlap.

Waterfall Model - Application

Every software developed is different and requires a suitable SDLC approach to be followed based on the internal and external factors. Some situations where the use of Waterfall model is most appropriate are −

- Requirements are very well documented, clear and fixed.
- Product definition is stable.
- Technology is understood and is not dynamic.
- There are no ambiguous requirements.
- Ample resources with required expertise are available to support the product.
- The project is short.

Waterfall Model - Advantages

The advantages of waterfall development are that it allows for departmentalization and control. A schedule can be set with deadlines for each stage of development and a product can proceed through the development process model phases one by one.

Development moves from concept, through design, implementation, testing, installation, troubleshooting, and ends up at operation and maintenance. Each phase of development proceeds in strict order.

Some of the major advantages of the Waterfall Model are as follows −

- Simple and easy to understand and use
- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Clearly defined stages.
- Well understood milestones.
- Easy to arrange tasks.
- Process and results are well documented.

Waterfall Model - Disadvantages

The disadvantage of waterfall development is that it does not allow much reflection or revision. Once an application is in the testing stage, it is very difficult to go back and change something that was not well-documented or thought upon in the concept stage.

The major disadvantages of the Waterfall Model are as follows −

- 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. So, risk and uncertainty is high with this process model.
- It is difficult to measure progress within stages.
- Cannot accommodate changing requirements.
- Adjusting scope during the life cycle can end a project.

Spiral Model - Design

The spiral model has four phases. A software project repeatedly passes through these phases in iterations called Spirals.

Identification

This phase starts with gathering the business requirements in the baseline spiral. In the subsequent spirals as the product matures, identification of system requirements, subsystem requirements and unit requirements are all done in this phase.

This phase also includes understanding the system requirements by continuous communication between the customer and the system analyst. At the end of the spiral, the product is deployed in the identified market.

Design

The Design phase starts with the conceptual design in the baseline spiral and involves architectural design, logical design of modules, physical product design and the final design in the subsequent spirals.

Construct or Build

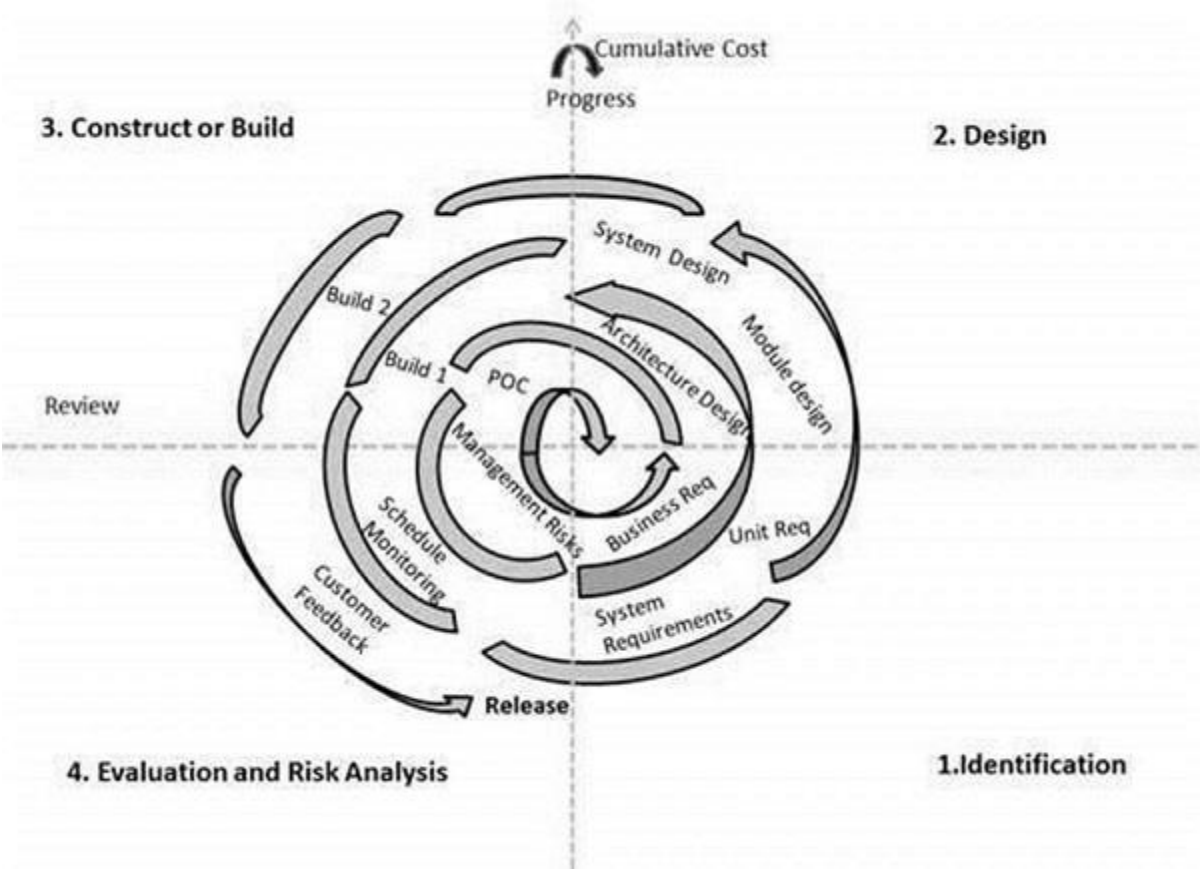
The Construct phase refers to production of the actual software product at every spiral. In the baseline spiral, when the product is just thought of and the design is being developed a POC (Proof of Concept) is developed in this phase to get customer feedback.

Then in the subsequent spirals with higher clarity on requirements and design details a working model of the software called build is produced with a version number. These builds are sent to the customer for feedback.

Evaluation and Risk Analysis

Risk Analysis includes identifying, estimating and monitoring the technical feasibility and management risks, such as schedule slippage and cost overrun. After testing the build, at the end of first iteration, the customer evaluates the software and provides feedback.

The following illustration is a representation of the Spiral Model, listing the activities in each phase.



Based on the customer evaluation, the software development process enters the next iteration and subsequently follows the linear approach to implement the feedback suggested by the customer. The process of iterations along the spiral continues throughout the life of the software.

Spiral Model Application

The Spiral Model is widely used in the software industry as it is in sync with the natural development process of any product, i.e. learning with maturity which involves minimum risk for the customer as well as the development firms.

The following pointers explain the typical uses of a Spiral Model −

- When there is a budget constraint and risk evaluation is important.
- For medium to high-risk projects.
- Long-term project commitment because of potential changes to economic priorities as the requirements change with time.
- Customer is not sure of their requirements which is usually the case.
- Requirements are complex and need evaluation to get clarity.

- New product line which should be released in phases to get enough customer feedback.
- Significant changes are expected in the product during the development cycle.

Spiral Model - Pros and Cons

The advantage of spiral lifecycle model is that it allows elements of the product to be added in, when they become available or known. This assures that there is no conflict with previous requirements and design.

This method is consistent with approaches that have multiple software builds and releases which allows making an orderly transition to a maintenance activity. Another positive aspect of this method is that the spiral model forces an early user involvement in the system development effort.

On the other side, it takes a very strict management to complete such products and there is a risk of running the spiral in an indefinite loop. So, the discipline of change and the extent of taking change requests is very important to develop and deploy the product successfully.

The advantages of the Spiral SDLC Model are as follows −

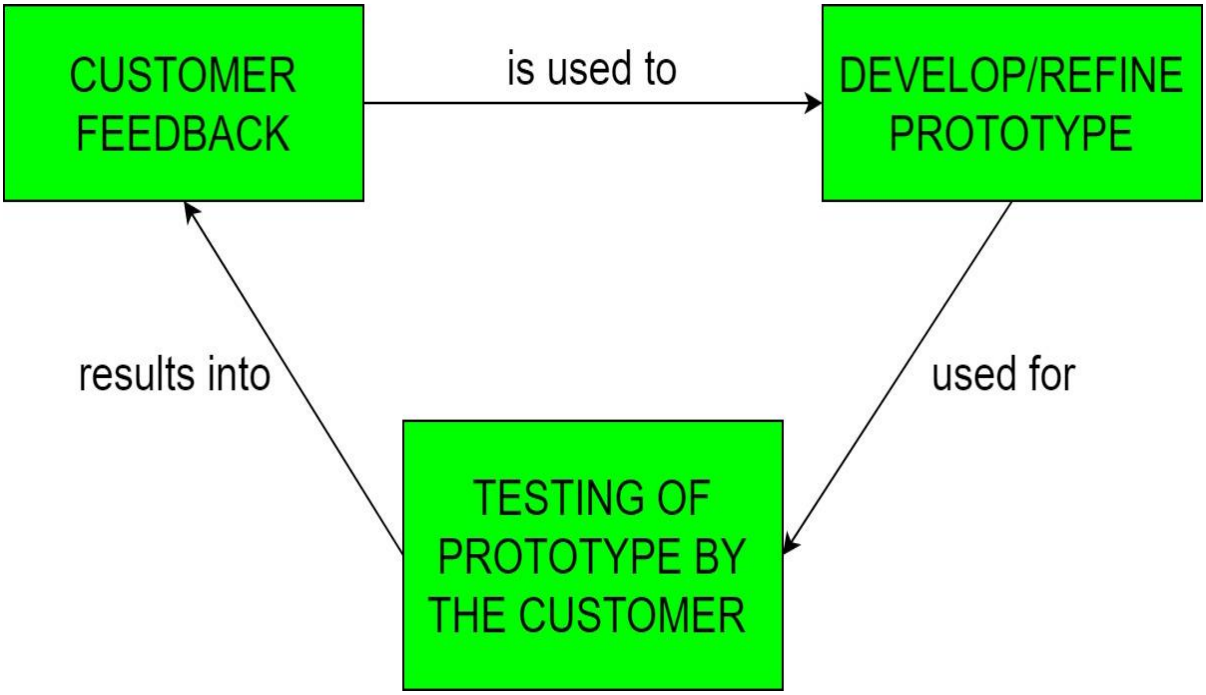
- Changing requirements can be accommodated.
- Allows extensive use of prototypes.
- Requirements can be captured more accurately.
- Users see the system early.
- Development can be divided into smaller parts and the risky parts can be developed earlier which helps in better risk management.

The disadvantages of the Spiral SDLC Model are as follows −

- Management is more complex.
- End of the project may not be known early.
- Not suitable for small or low risk projects and could be expensive for small projects.
- Process is complex
- Spiral may go on indefinitely.
- Large number of intermediate stages requires excessive documentation.

Prototyping Model

Prototyping is defined as the process of developing a working replication of a product or system that has to be engineered. It offers a small scale facsimile of the end product and is used for obtaining customer feedback as described below:



The Prototyping Model is one of the most popularly used Software Development Life Cycle Models (SDLC models). This model is used when the customers do not know the exact project requirements beforehand. In this model, a prototype of the end product is first developed, tested and refined as per customer feedback repeatedly till a final acceptable prototype is achieved which forms the basis for developing the final product.

In this process model, the system is partially implemented before or during the analysis phase thereby giving the customers an opportunity to see the product early in the life cycle. The process starts by interviewing the customers and developing the incomplete high-level paper model. This document is used to build the initial prototype supporting only the basic functionality as desired by the customer. Once the customer figures out the problems, the prototype is further refined to eliminate them. The process continues till the user approves the prototype and finds the working model to be satisfactory.

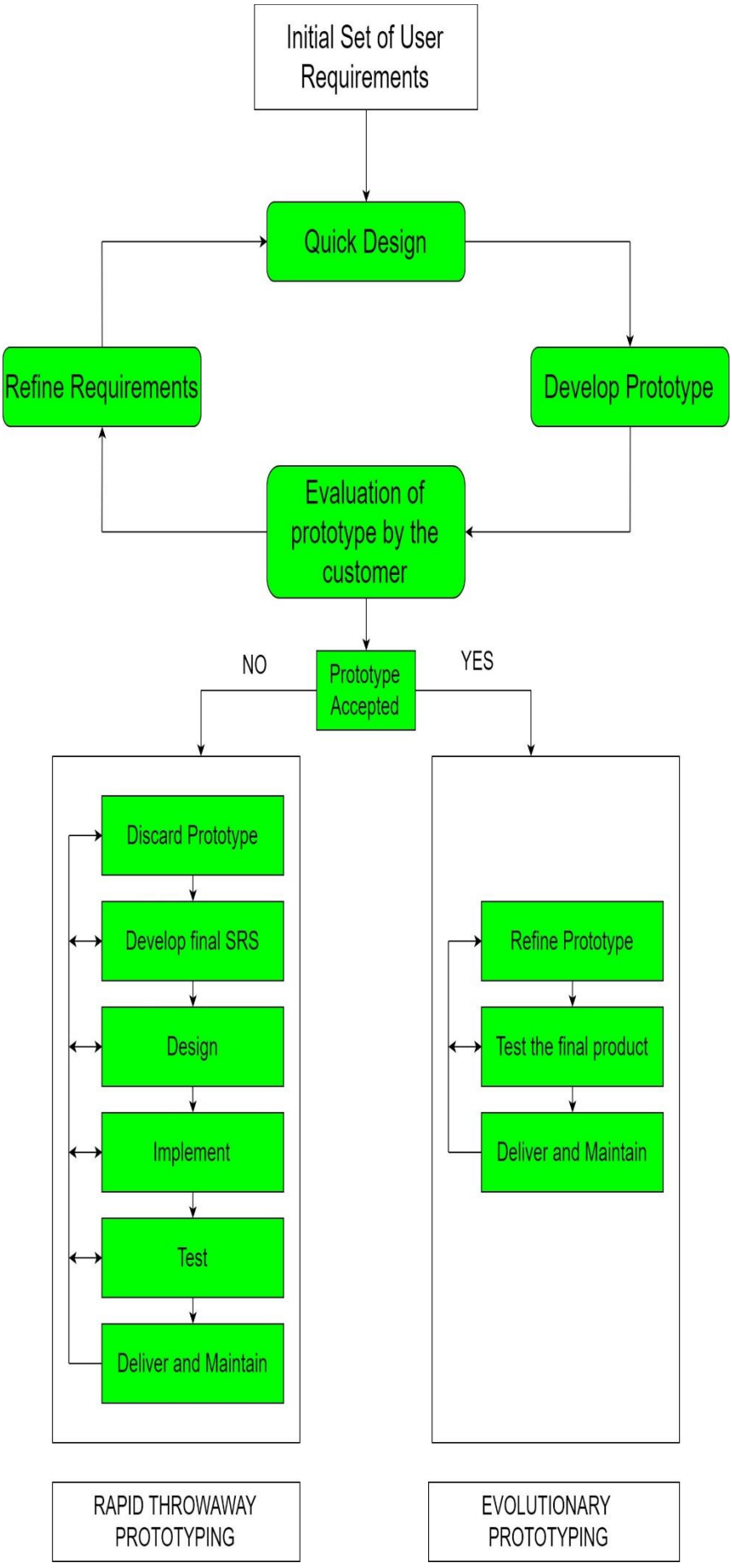
There are 2 approaches for this model:

1. **Rapid Throwaway Prototyping –**

   This technique offers a useful method of exploring ideas and getting customer feedback for each of them. In this method, a developed prototype need not necessarily be a part of the ultimately accepted prototype. Customer feedback helps in preventing unnecessary design faults and hence, the final prototype developed is of a better quality.

2. **Evolutionary Prototyping –**

   In this method, the prototype developed initially is incrementally refined on the basis of customer feedback till it finally gets accepted. In comparison to Rapid Throwaway Prototyping, it offers a better approach which saves time as well as effort. This is because developing a prototype from scratch for every iteration of the process can sometimes be very frustrating for the developers.


Initial Set of User Requirements
Quick Design
Refine Requirements
Develop Prototype
Evaluation of prototype by the customer
NO
Prototype Accepted
YES
Discard Prototype
Develop final SRS
Design
Implement
Test
Deliver and Maintain
Refine Prototype
Test the final product
Deliver and Maintain
RAPID THROWAWAY PROTOTYPING
EVOLUTIONARY PROTOTYPING

Advantages –

- The customers get to see the partial product early in the life cycle. This ensures a greater level of customer satisfaction and comfort.
- New requirements can be easily accommodated as there is scope for refinement.
- Missing functionalities can be easily figured out.
- Errors can be detected much earlier thereby saving a lot of effort and cost, besides enhancing the quality of the software.
- The developed prototype can be reused by the developer for more complicated projects in the future.
- Flexibility in design.

Disadvantages –

- Costly w.r.t time as well as money.
- There may be too much variation in requirements each time the prototype is evaluated by the customer.
- Poor Documentation due to continuously changing customer requirements.
- It is very difficult for the developers to accommodate all the changes demanded by the customer.
- There is uncertainty in determining the number of iterations that would be required before the prototype is finally accepted by the customer.
- After seeing an early prototype, the customers sometimes demand the actual product to be delivered soon.
- Developers in a hurry to build prototypes may end up with sub-optimal solutions.
- The customer might lose interest in the product if he/she is not satisfied with the initial prototype.

Use –

The Prototyping Model should be used when the requirements of the product are not clearly understood or are unstable. It can also be used if requirements are changing quickly. This model can be successfully used for developing user interfaces, high technology software-intensive systems, and systems with complex algorithms and interfaces. It is also a very good choice to demonstrate the technical feasibility of the product.

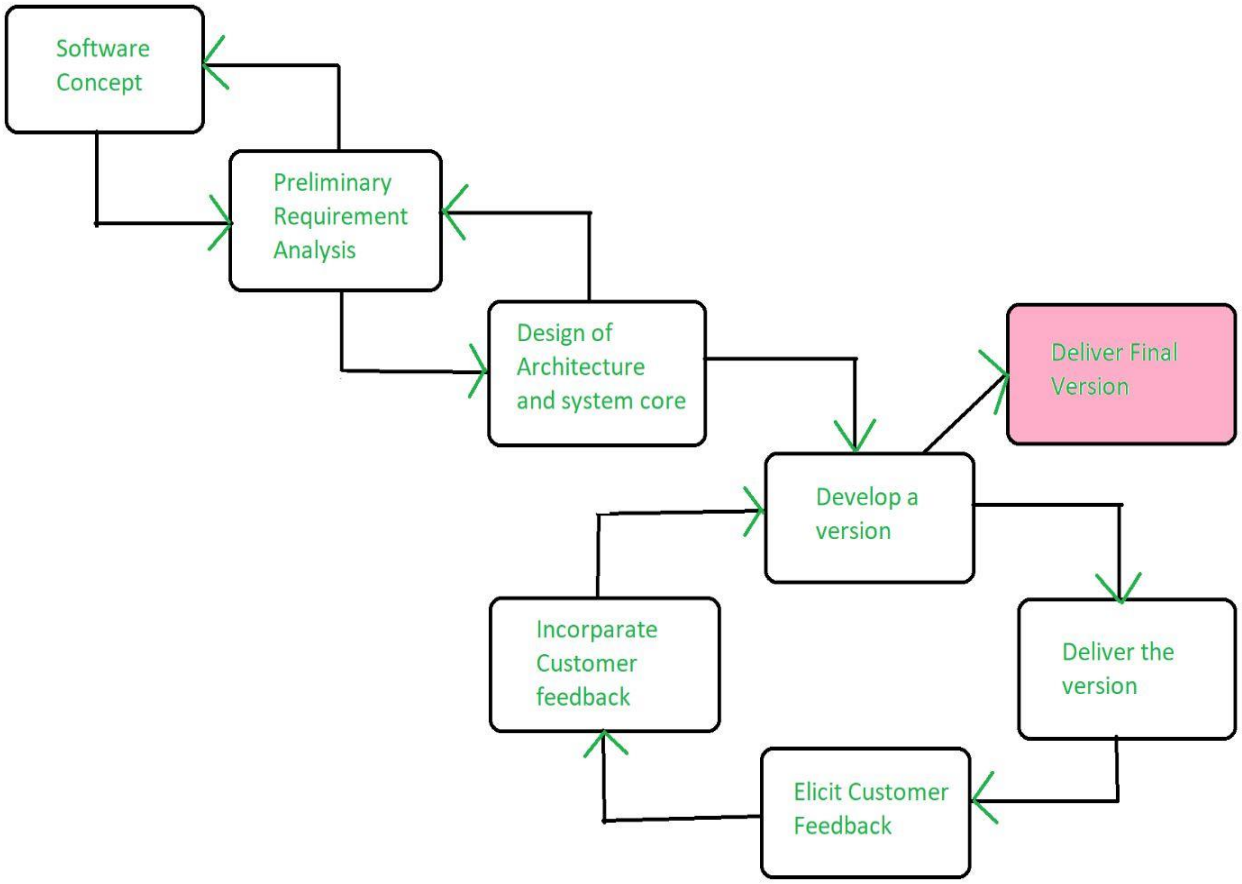
Evolutionary Model

Evolutionary model is a combination of Iterative and Incremental model of software development life cycle. Delivering your system in a big bang release, delivering it in incremental process over time is the action done in this model. Some initial requirements and architecture envisioning need to be done.

It is better for software products that have their feature sets redefined during development because of user feedback and other factors. The Evolutionary development model divides the development cycle into smaller, incremental waterfall models in which users are able to get access to the product at the end of each cycle.

Feedback is provided by the users on the product for the planning stage of the next cycle and the development team responds, often by changing the product, plan or process. Therefore, the software product evolves with time.

All the models have the disadvantage that the duration of time from start of the project to the delivery time of a solution is very high. Evolutionary model solves this problem in a different approach.



Evolutionary model suggests breaking down of work into smaller chunks, prioritizing them and then delivering those chunks to the customer one by one. The number of chunks is huge and is the number of deliveries made to the customer. The main advantage is that the customer's confidence increases as he constantly gets quantifiable goods or services from the beginning of the project to verify and validate his requirements. The model allows for changing requirements as well as all work in broken down into maintainable work chunks.

Application of Evolutionary Model:

1. It is used in large projects where you can easily find modules for incremental implementation. Evolutionary model is commonly used when the customer wants to start using the core features instead of waiting for the full software.
2. Evolutionary model is also used in object oriented software development because the system can be easily portioned into units in terms of objects.

Advantages:

- In evolutionary model, a user gets a chance to experiment partially developed system.
- It reduces the error because the core modules get tested thoroughly.

Disadvantages:

- Sometimes it is hard to divide the problem into several versions that would be acceptable to the customer which can be incrementally implemented and delivered.

Agile Software Development

Agile is a time-bound, iterative approach to software delivery that builds software incrementally from the start of the project, instead of trying to deliver all at once.

Why Agile?

Technology in this current era is progressing faster than ever, enforcing the global software companies to work in a fast-paced changing environment. Because these businesses are operating in an ever-changing environment, it is impossible to gather a complete and exhaustive set of software requirements. Without these requirements, it becomes practically hard for any conventional software model to work.

The conventional software models such as Waterfall Model that depends on completely specifying the requirements, designing, and testing the system are not geared towards rapid software development. As a consequence, a conventional software development model fails to deliver the required product.

This is where the agile software development comes to the rescue. It was specially designed to curate the needs of the rapidly changing environment by embracing the idea of incremental development and develop the actual final product.

Let's now read about the on which the Agile has laid its foundation:

Principles:

1. Highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. It welcomes changing requirements, even late in development.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shortest timescale.
4. Build projects around motivated individuals. Give them the environment and the support they need, and trust them to get the job done.
5. Working software is the primary measure of progress.
6. Simplicity the art of maximizing the amount of work not done is essential.
7. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Development in Agile: Let's see a brief overview of how development occurs in Agile philosophy.

- In Agile development, Design and Implementation are considered to be the central activities in the software process.
- Design and Implementation phase also incorporate other activities such as requirements elicitation and testing into it.
- In an agile approach, iteration occurs across activities. Therefore, the requirements and the design are developed together, rather than separately.
- The allocation of requirements and the design planning and development as executed in a series of increments. In contrast with the conventional model, where requirements gathering needs to be completed in order to proceed to the design and development phase, it gives Agile development an extra level of flexibility.
- An agile process focuses more on code development rather than documentation.

Example: Let's go through an example to understand clearly about how agile actually works. A Software company named **ABC** wants to make a new web browser for the latest release of its operating system. The deadline for the task is 10 months. The company's head assigned two teams named **Team A** and **Team B** for this task. In order to motivate the teams, the company head says that the first team to develop the browser would be given a salary hike and a one week full sponsored travel plan. With the dreams of their wild travel fantasies, the two teams set out on the journey of the web browser. The team A decided to play by the book and decided to choose the Waterfall model for the development. Team B after a heavy discussion decided to take a leap of faith and choose Agile as their development model.

The Development plan of the Team A is as follows:

- Requirement analysis and Gathering – 1.5 Months
- Design of System – 2 Months
- Coding phase – 4 Months
- System Integration and Testing – 2 Months
- User Acceptance Testing – 5 Weeks

The Development plan for the Team B is as follows:

- Since this was an Agile, the project was broken up into several iterations.
- The iterations are all of the same time duration.
- At the end of each iteration, a working product with a new feature has to be delivered.

- Instead of Spending 1.5 months on requirements gathering, They will decide the core features that are required in the product and decide which of these features can be developed in the first iteration.
- Any remaining features that cannot be delivered in the first iteration will be delivered in the next subsequent iteration, based in the priority
- At the end of the first iterations, the team will deliver a working software with the core basic features.

Both the team have put their best efforts to get the product to a complete stage. But then out of blue due to the rapidly changing environment, the company's head come up with an entirely new set of features and want to be implemented as quickly as possible and wanted to push out a working model in 2 days. Team A was now in a fix, they were still in their design phase and did not yet started coding and they had no working model to display. And moreover, it was practically impossible for them to implement new features since waterfall model there is not reverting back to the old phase once you proceed to the next stage, that means they would have to start from the square one again. That would incur them heavy cost and a lot of overtime. Team B was ahead of Team A in a lot of aspects, all thanks to Agile Development. They also had the working product with most of the core requirement since the first increment. And it was a piece of cake for them to add the new requirements. All they had to do is schedule these requirements for the next increment and then implement them.

Advantages:

- Deployment of software is quicker and thus helps in increasing the trust of the customer.
- Can better adapt to rapidly changing requirements and respond faster.
- Helps in getting immediate feedback which can be used to improve the software in the next increment.
- People – Not Process. People and interactions are given a higher priority rather than process and tools.
- Continuous attention to technical excellence and good design.

Disadvantages:

- In case of large software projects, it is difficult to assess the effort required at the initial stages of the software development life cycle.
- The Agile Development is more code focused and produces less documentation.

- Agile development is heavily depended on the inputs of the customer. If the customer has ambiguity in his vision of the final outcome, it is highly likely for the project to get off track.
- Face to Face communication is harder in large-scale organizations.
- Only senior programmers are capable of taking the kind of decisions required during the development process. Hence it's a difficult situation for new programmers to adapt to the environment.

Agile is a framework which defines how the software development needs to be carried on. Agile is not a single method, it represents the various collection of methods and practices that follow the value statements provided in the manifesto. Agile methods and practices do not promise to solve every problem present in the software industry (No Software model ever can). But they sure help to establish a culture and environment where solutions emerge.

4. Comparative Analysis of SDLC Models

Features	**Waterfall Model**	**Iterative Model**	**Prototyping Model**	**Spiral Model**
Requirements Specification	Beginning	Beginning	Frequently Changed	Beginning
Cost	Low	Low	High	Expensive
Simplicity	Simple	Intermediate	Simple	Intermediate
Expertise Required	High	High	Medium	High
Risk Involvement	High	Easily Manage	Low	Low
Overlapping Phases	No	No	Yes	Yes
Flexibility	Rigid	Less Flexible	Highly Flexible	Flexible
Maintenance	Least Glamorous	Promoted Maintainability	Routine Maintenance	Typical
Reusability	Limited	Yes	Weak	High
Documentation Required	Vital	Yes	Weak	Yes
User Involvement	Only At Beginning	Intermediate	High	High
Cost Control	Yes	No	No	Yes
Resource Control	Yes	Yes	No	Yes
Guarantee of success	Less	High	Good	High

Software planning

Responsibilities of Software Project Manager

A software project manager is the most important person inside a team who takes the overall responsibilities to manage the software projects and play an important role in the successful completion of the projects. A project manager has to face many difficult situations to accomplish these works. In fact, the job responsibilities of a project manager range from invisible activities like building up team morale to highly visible customer presentations. Most of the managers take responsibility for writing the project proposal, project cost estimation, scheduling, project staffing, software process tailoring, project monitoring and control, software configuration management, risk management, managerial report writing and presentation and interfacing with clients. The task of a project manager are classified into two major types:

1. Project planning
2. Project monitoring and control

Project planning

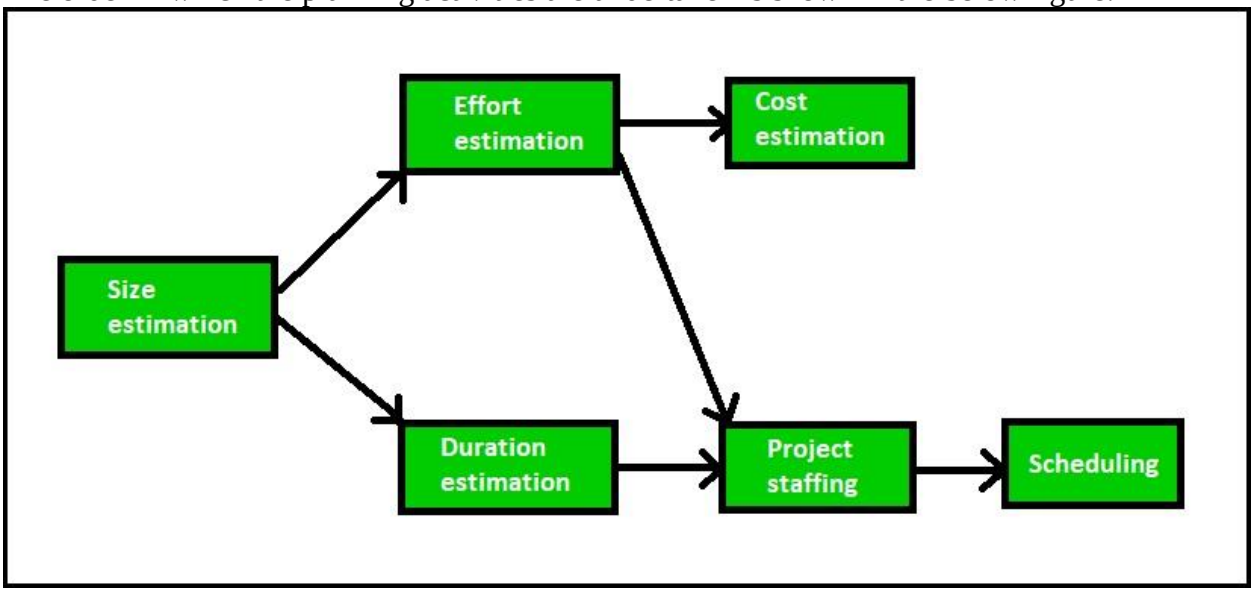
Project planning is undertaken immediately after the feasibility study phase and before the starting of the requirement analysis and specification phase. Once a project has been found to be feasible, Software project managers started project planning. Project planning is completed before any development phase starts. Project planning involves estimating several characteristics of a project and then plan the project activities based on these estimations. Project planning is done with most care and attention. A wrong estimation can result in schedule slippage. Schedule delay can cause customer dissatisfaction, which may lead to a project failure. For effective project planning, in addition to a very good knowledge of various estimation techniques, past experience is also very important. During the project planning the project manager performs the following activities:

1. **Project Estimation:** Project Size Estimation is the most important parameter based on which all other estimations like cost, duration and effort are made.
    - **Cost Estimation:** Total expenses to develop the software product is estimated.
    - **Time Estimation:** The total time required to complete the project.
    - **Effort Estimation:** The effort needed to complete the project is estimated.

    The effectiveness of all later planning activities is dependent on the accuracy of these three estimations.

2. **Scheduling:** After completion of estimation of all the project parameters, scheduling for manpower and other resources are done.
3. **Staffing:** Team structure and staffing plans are made.
4. **Risk Management:** The project manager should identify the unanticipated risks that may occur during project development risk, analysis the damage might cause these risks and take risk reduction plan to cope up with these risks.
5. **Miscellaneous plans:** This includes making several other plans such as quality assurance plan, configuration management plan, etc.

The order in which the planning activities are undertaken is shown in the below figure:



Project monitoring and control

Project monitoring and control activities are undertaken once the development activities start. The main focus of project monitoring and control activities is to ensure that the software development proceeds as per plan. This includes checking whether the project is going on as per plan or not if any problem created then the project manager must take necessary action to solve the problem.

Role of a software project manager: There are many roles of a project manager in the development of software.

- **Lead the team:** The project manager must be a good leader who makes a team of different members of various skills and can complete their individual task.
- **Motivate the team-member:** One of the key roles of a software project manager is to encourage team member to work properly for the successful completion of the project.

- **Tracking the progress:** The project manager should keep an eye on the progress of the project. A project manager must track whether the project is going as per plan or not. If any problem arises, then take necessary action to solve the problem. Moreover, check whether the product is developed by maintaining correct coding standards or not.
- **Liaison:** Project manager is the link between the development team and the customer. Project manager analysis the customer requirements and convey it to the development team and keep telling the progress of the project to the customer. Moreover, the project manager checks whether the project is fulfilling the customer requirements or not.
- **Documenting project report:** The project manager prepares the documentation of the project for future purpose. The reports contain detailed features of the product and various techniques. These reports help to maintain and enhance the quality of the project in the future.

Necessary skills of software project manager: A good theoretical knowledge of various project management technique is needed to become a successful project manager, but only theoretical knowledge is not enough. Moreover, a project manager must have good decision-making abilities, good communication skills and the ability to control the team members with keeping a good rapport with them and the ability to get the work done by them. Some skills such as tracking and controlling the progress of the project, customer interaction, good knowledge of estimation techniques and previous experience are needed.

Skills that are the most important to become a successful project manager are given below:

- Knowledge of project estimation techniques
- Good decision-making abilities at the right time
- Previous experience of managing a similar type of projects
- Good communication skill to meet the customer satisfaction
- A project manager must encourage all the team members to successfully develop the product
- He must know the various type of risks that may occur and the solution for these problems.

LINES OF CODE (LOC) LOC is the simplest among all metrics available to estimate project size. This metric is very popular because it is the simplest to use. Using this metric, the project size is

estimated by counting the number of source instructions in the developed program. Obviously, while counting the number of source instructions, lines used for commenting the code and the header lines should be ignored. Determining the LOC count at the end of a project is a very simple job. However, accurate estimation of the LOC count at the beginning of a project is very difficult. In order to estimate the LOC count at the beginning of a project, project managers usually divide the problem into modules and each module into sub modules and so on, until the sizes of the different leaf-level modules can be approximately predicted. To be able to do this, past experience in developing similar products is helpful. By using the estimation of the lowest level modules, project managers arrive at the total size estimation. "A line of code is any line of program text that is not a comment or blank line, regardless of the number of statements or fragments of statements on the line. This specifically includes all lines containing program header, declarations, and executable and non-executable statements." For Example: Fig. 1 shows the C Program of calculating the Average of three numbers. This C program is written in 9 lines including comments that is begins with "/*". Remember that C comments are not executed.

```
Line No.
1 Main()
2 {
3 int a, b, c, avg;
4 /*enter value of a,b,c*/
5 scanf("%d %d %d", &a, &b,
&c
6 /*calculate average*/
7 avg = (a+b+c)/3;
8 printf("avg = %d", avg);
9 }
```

Fig. 1: According to the definition of the LOC, above Program (shown in Fig. 1) has a 7 LOC

Advantage of LOC:

- Counting LOC of any Program text is a very simplest method. Disadvantage of LOC:
- LOC is language dependent. For example a line of assembler is not the same as a line of COBOL.

- They also reflect what the system is rather than what it does.

- Counting LOC is similar to counting bricks in a building. Measuring systems by the LOC is rather like measuring a building by the number of bricks which is used to create the building while Building are normally described in terms of number and size of the rooms, their total areas in square feet.

- Difficult to estimate LOC from problem description and therefore it is not very useful for project planning

COCOMO Model

Boehm proposed COCOMO (Constructive Cost Estimation Model) in 1981.COCOMO is one of the most generally used software estimation models in the world. COCOMO predicts the efforts and schedule of a software product based on the size of the software.

The necessary steps in this model are:

1. Get an initial estimate of the development effort from evaluation of thousands of delivered lines of source code (KDLOC).
2. Determine a set of 15 multiplying factors from various attributes of the project.
3. Calculate the effort estimate by multiplying the initial estimate with all the multiplying factors i.e., multiply the values in step1 and step2.

The initial estimate (also called nominal estimate) is determined by an equation of the form used in the static single variable models, using KDLOC as the measure of the size. To determine the initial effort E_i in person-months the equation used is of the type is shown below

$$\mathbf{E_i=a*(KDLOC)b}$$

The value of the constant a and b are depends on the project type.

In COCOMO, projects are categorized into three types:

1. Organic
2. Semidetached
3. Embedded

1.Organic: A development project can be treated of the 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 methods of projects. **Examples of this type of projects are simple business systems, simple inventory management systems, and data processing systems.**

2. Semidetached: A development project can be treated with semidetached type if the development consists of a mixture of experienced and inexperienced staff. Team members may have finite experience in related systems but may be unfamiliar with some aspects of the order being developed. **Example of Semidetached system includes developing a new operating system (OS), a Database Management System (DBMS), and complex inventory management system.**

3. Embedded: A development project is treated to be of an embedded type, if the software being developed is strongly coupled to complex hardware, or if the stringent regulations on the operational method exist. **For Example:** ATM, Air Traffic control.

For three product categories, Bohem provides a different set of expression to predict effort (in a unit of person month)and development time from the size of estimation in KLOC(Kilo Line of code) efforts estimation takes into account the productivity loss due to holidays, weekly off, coffee breaks, etc.

According to Boehm, software cost estimation should be done through three stages:

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

1. Basic COCOMO Model: The basic COCOMO model provide an accurate size of the project parameters. The following expressions give the basic COCOMO estimation model:

$$\textbf{Effort} = a_1 * (KLOC)^{a_2} \ PM$$

$$\textbf{Tdev} = b_1 * (efforts)^{b_2} \ Months$$

Where

KLOC is the estimated size of the software product indicate in Kilo Lines of Code,

a_1, a_2, b_1, b_2 are constants for each group of software products,

Tdev is the estimated time to develop the software, expressed in months,

Effort is the total effort required to develop the software product, expressed in **person months (PMs)**.

Estimation of development effort

For the three classes of software products, the formulas for estimating the effort based on the code size are shown below:

Organic: Effort = 2.4(KLOC) 1.05 PM

Semi-detached: Effort = 3.0(KLOC) 1.12 PM

Embedded: Effort = 3.6(KLOC) 1.20 PM

Estimation of development time

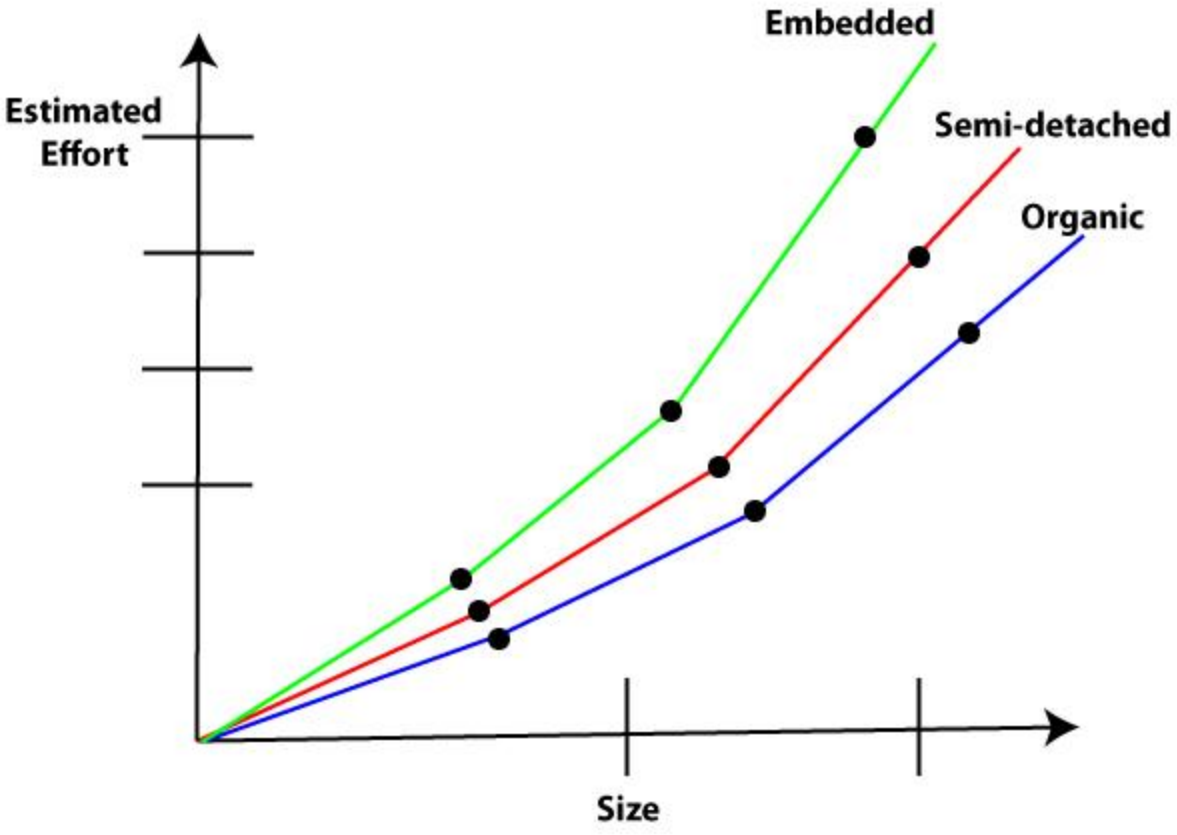
For the three classes of software products, the formulas for estimating the development time based on the effort are given below:

Organic: Tdev = 2.5(Effort) 0.38 Months

Semi-detached: Tdev = 2.5(Effort) 0.35 Months

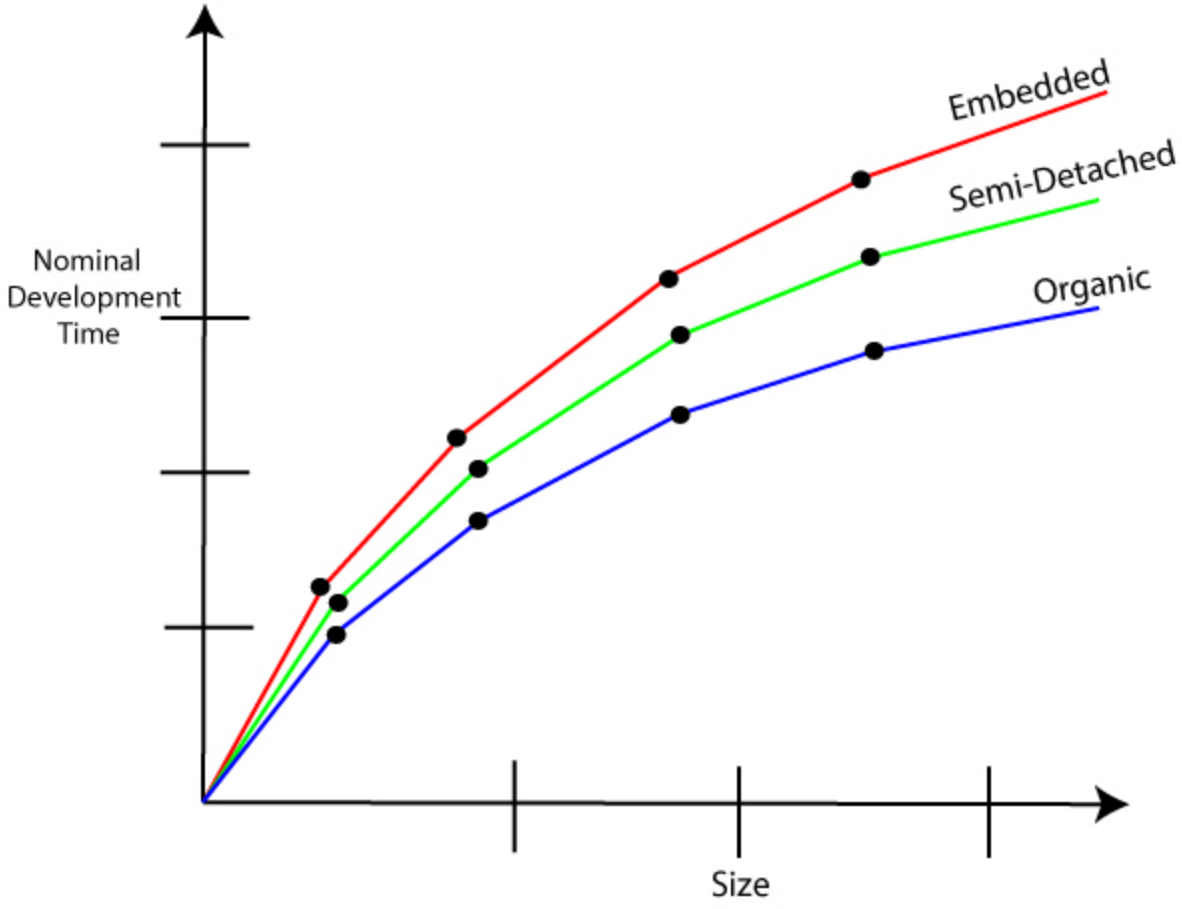
Embedded: Tdev = 2.5(Effort) 0.32 Months

Some insight into the basic COCOMO model can be obtained by plotting the estimated characteristics for different software sizes. Fig shows a plot of estimated effort versus product size. From fig, we can observe that the effort is somewhat superliner in the size of the software product. Thus, the effort required to develop a product increases very rapidly with project size.



Effort versus product size

The development time versus the product size in KLOC is plotted in fig. From fig it can be observed that the development time is a sub linear function of the size of the product, i.e. when the size of the product increases by two times, the time to develop the product does not double but rises moderately. This can be explained by the fact that for larger products, a larger number of activities which can be carried out concurrently can be identified. The parallel activities can be carried out simultaneously by the engineers. This reduces the time to complete the project. Further, from fig, it can be observed that the development time is roughly the same for all three categories of products. For example, a 60 KLOC program can be developed in approximately 18 months, regardless of whether it is of organic, semidetached, or embedded type.



Development time versus size

From the effort estimation, the project cost can be obtained by multiplying the required effort by the manpower cost per month. But, implicit in this project cost computation is the assumption that the entire project cost is incurred on account of the manpower cost alone. In addition to manpower cost, a project would incur costs due to hardware and software required for the project and the company overheads for administration, office space, etc.

It is important to note that the effort and the duration estimations obtained using the COCOMO model are called a nominal effort estimate and nominal duration estimate. The term nominal implies that if anyone tries to complete the project in a time shorter than the estimated duration, then the cost will increase drastically. But, if anyone completes the project over a longer period of time than the estimated, then there is almost no decrease in the estimated cost value.

Example1: Suppose a project was estimated to be 400 KLOC. Calculate the effort and development time for each of the three model i.e., organic, semi-detached & embedded.

Solution: The basic COCOMO equation takes the form:

Effort=a_1*(KLOC) a_2 PM

Tdev=b_1*(efforts)b_2 Months

Estimated Size of project= 400 KLOC

(i)Organic Mode

E = 2.4 * (400)1.05 = 1295.31 PM

D = 2.5 * (1295.31)0.38=38.07 PM

(ii)Semidetached Mode

E = 3.0 * (400)1.12=2462.79 PM

D = 2.5 * (2462.79)0.35=38.45 PM

(iii) Embedded Mode

E = 3.6 * (400)1.20 = 4772.81 PM

D = 2.5 * (4772.8)0.32 = 38 PM

Example2: A project size of 200 KLOC is to be developed. Software development team has average experience on similar type of projects. The project schedule is not very tight. Calculate the Effort, development time, average staff size, and productivity of the project.

Solution: The semidetached mode is the most appropriate mode, keeping in view the size, schedule and experience of development time.

Hence E=3.0(200)1.12=1133.12PM

D=2.5(1133.12)0.35=29.3PM

$$\text{Average Staff Size (SS)} = \frac{E}{D} \text{ Persons}$$

$$= \frac{1133.12}{29.3} = 38.67 \text{ Persons}$$

$$\text{Productivity} = \frac{KLOC}{E} = \frac{200}{1133.12} = 0.1765 \text{ KLOC/PM}$$

P = 176 LOC/PM

2. Intermediate Model: The basic Cocomo model considers that the effort is only a function of the number of lines of code and some constants calculated according to the various software systems. The intermediate COCOMO model recognizes these facts and refines the

initial estimates obtained through the basic COCOMO model by using a set of 15 cost drivers based on various attributes of software engineering.

Classification of Cost Drivers and their attributes:

(i) Product attributes -

- Required software reliability extent
- Size of the application database
- The complexity of the product

Hardware attributes -

- Run-time performance constraints
- Memory constraints
- The 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

The cost drivers are divided into four categories:

Cost Drivers	RATINGS					
	Very low	Low	Nominal	High	Very High	Extra High
Product Attributes						
RELY	0.75	0.88	1.00	1.15	1.40	..
DATA	..	0.94	1.00	1.08	1.16	..
CPLX	0.70	0.85	1.00	1.15	1.30	1.65
Computer Attributes						
TIME	..	..	1.00	1.11	1.30	1.66
STOR	..	..	1.00	1.06	1.21	1.56
VIRT	..	0.87	1.00	1.15	1.30	..
TURN	..	0.87	1.00	1.07	1.15	..

Cost Drivers	RATINGS					
	Very low	Low	Nominal	High	Very high	Extra high
Personnel Attributes						
ACAP	1.46	1.19	1.00	0.86	0.71	..
AEXP	1.29	1.13	1.00	0.91	0.82	..
PCAP	1.42	1.17	1.00	0.86	0.70	..
VEXP	1.21	1.10	1.00	0.90	..	..
LEXP	1.14	1.07	1.00	0.95	..	..
Project Attributes						
MODP	1.24	1.10	1.00	0.91	0.82	..
TOOL	1.24	1.10	1.00	0.91	0.83	..
SCED	1.23	1.08	1.00	1.04	1.10	..

Intermediate COCOMO equation:

$$E=a_i (KLOC)\, b_i * EAF$$

$$D=c_i (E) d_i$$

Coefficients for intermediate COCOMO

Project	a_i	b_i	c_i	d_i
Organic	2.4	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

3. Detailed COCOMO Model: Detailed COCOMO incorporates all qualities of the standard version with an assessment of the cost driver?s effect on each method of the software engineering process. The detailed model uses various effort multipliers for each cost driver property. In detailed cocomo, the whole software is differentiated into multiple modules, and then we apply COCOMO in various modules to estimate effort and then sum the effort.

The Six phases of detailed COCOMO are:

1. Planning and requirements
2. System structure
3. Complete structure
4. Module code and test
5. Integration and test
6. Cost Constructive model

Unit-4

Requirement Engineering

The process to gather the software requirements from client, analyze and document them is known as requirement engineering.

The goal of requirement engineering is to develop and maintain sophisticated and descriptive 'System Requirements Specification' document.

Requirement Engineering Process

It is a four step process, which includes –

- Feasibility Study
- Requirement Gathering
- Software Requirement Specification
- Software Requirement Validation

Let us see the process briefly -

Feasibility study

When the client approaches the organization for getting the desired product developed, it comes up with rough idea about what all functions the software must perform and which all features are expected from the software.

Referencing to this information, the analysts does a detailed study about whether the desired system and its functionality are feasible to develop.

This feasibility study is focused towards goal of the organization. This study analyzes whether the software product can be practically materialized in terms of implementation, contribution of project to organization, cost constraints and as per values and objectives of the organization. It explores technical aspects of the project and product such as usability, maintainability, productivity and integration ability.

The output of this phase should be a feasibility study report that should contain adequate comments and recommendations for management about whether or not the project should be undertaken.

Requirement Gathering

If the feasibility report is positive towards undertaking the project, next phase starts with gathering requirements from the user. Analysts and engineers communicate with the client and end-users to know their ideas on what the software should provide and which features they want the software to include.

Software Requirement Specification

SRS is a document created by system analyst after the requirements are collected from various stakeholders.

SRS defines how the intended software will interact with hardware, external interfaces, speed of operation, response time of system, portability of software across various platforms, maintainability, speed of recovery after crashing, Security, Quality, Limitations etc.

The requirements received from client are written in natural language. It is the responsibility of system analyst to document the requirements in technical language so that they can be comprehended and useful by the software development team.

SRS should come up with following features:

- User Requirements are expressed in natural language.
- Technical requirements are expressed in structured language, which is used inside the organization.
- Design description should be written in Pseudo code.
- Format of Forms and GUI screen prints.
- Conditional and mathematical notations for DFDs etc.

Software Requirement Validation

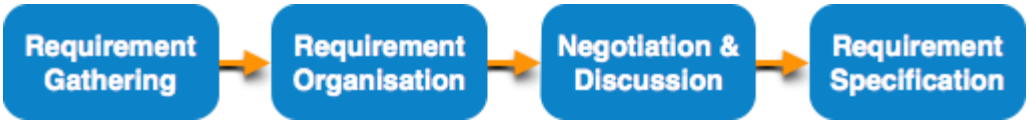
After requirement specifications are developed, the requirements mentioned in this document are validated. User might ask for illegal, impractical solution or experts may interpret the requirements incorrectly. This results in huge increase in cost if not nipped in the bud. Requirements can be checked against following conditions -

- If they can be practically implemented
- If they are valid and as per functionality and domain of software
- If there are any ambiguities
- If they are complete

- If they can be demonstrated

Requirement Elicitation Process

Requirement elicitation process can be depicted using the folloiwng diagram:



- **Requirements gathering -** The developers discuss with the client and end users and know their expectations from the software.
- **Organizing Requirements -** The developers prioritize and arrange the requirements in order of importance, urgency and convenience.
- **Negotiation & discussion -** If requirements are ambiguous or there are some conflicts in requirements of various stakeholders, if they are, it is then negotiated and discussed with stakeholders. Requirements may then be prioritized and reasonably compromised.

  The requirements come from various stakeholders. To remove the ambiguity and conflicts, they are discussed for clarity and correctness. Unrealistic requirements are compromised reasonably.

- **Documentation -** All formal & informal, functional and non-functional requirements are documented and made available for next phase processing.

Requirement Elicitation Techniques

Requirements Elicitation is the process to find out the requirements for an intended software system by communicating with client, end users, system users and others who have a stake in the software system development.

There are various ways to discover requirements

Interviews

Interviews are strong medium to collect requirements. Organization may conduct several types of interviews such as:

- Structured (closed) interviews, where every single information to gather is decided in advance, they follow pattern and matter of discussion firmly.
- Non-structured (open) interviews, where information to gather is not decided in advance, more flexible and less biased.
- Oral interviews
- Written interviews
- One-to-one interviews which are held between two persons across the table.
- Group interviews which are held between groups of participants. They help to uncover any missing requirement as numerous people are involved.

Surveys

Organization may conduct surveys among various stakeholders by querying about their expectation and requirements from the upcoming system.

Questionnaires

A document with pre-defined set of objective questions and respective options is handed over to all stakeholders to answer, which are collected and compiled.

A shortcoming of this technique is, if an option for some issue is not mentioned in the questionnaire, the issue might be left unattended.

Task analysis

Team of engineers and developers may analyze the operation for which the new system is required. If the client already has some software to perform certain operation, it is studied and requirements of proposed system are collected.

Domain Analysis

Every software falls into some domain category. The expert people in the domain can be a great help to analyze general and specific requirements.

Brainstorming

An informal debate is held among various stakeholders and all their inputs are recorded for further requirements analysis.

Prototyping

Prototyping is building user interface without adding detail functionality for user to interpret the features of intended software product. It helps giving better idea of requirements. If there is no software installed at client's end for developer's reference and the client is not aware of its own requirements, the developer creates a prototype based on initially mentioned requirements. The prototype is shown to the client and the feedback is noted. The client feedback serves as an input for requirement gathering.

Observation

Team of experts visit the client's organization or workplace. They observe the actual working of the existing installed systems. They observe the workflow at client's end and how execution problems are dealt. The team itself draws some conclusions which aid to form requirements expected from the software.

CHARACTERISTICS OF THE SRS

Introduction to Characteristics of the SRS

To properly satisfy the basic goals an SRS should have certain properties and should contain different types of requirements. A good SRS document should have the following characteristics:

1. Completeness

2. Clarity

3. Correctness

4. Consistency

5. Verifiability

6. Ranking

7. Modifiability

8. Traceability

9. Feasibility

1. Completeness: A SRS is complete if everything the software the software is supposed to do and the responses of the software to all class of input data are specified in SRS. It ensures that everything is indeed specified. It is one of the most difficult properties to spot. To ensure completeness, one has to detect the absence of specification which is much harder to determine.

2. Clarity: The documented requirement should lead to only a single interpretation, independent of the person or the time when the interpretation is done. The SRS needs to be unambiguous to the authors, the users, other reviewers as well as the developers and testers who will use the document. So SRS writer should be careful about ambiguity. One way to avoid ambiguity is to use some formal requirements specification language, but it is the major drawback. The formal languages require more effort to write an SRS more cost and the increased difficulty in reading and understanding formally stated requirements.

3. Correctness: The SRS can be considered as correct if every requirements stated in the SRS is required in the proposed system. Correctness of an SRS:

Ensures that what is specified is done correctly.

Is an easier property to establish as it basically involves examining each requirement to make sure that it represents the use requirements?

There are no real tools or procedures that ensure correctness. If there are any preceding documents then the requirements from those earlier documents need to be traced to the SRS:

4. Consistency: Requirements at all levels must be consistent with each other .any conflict between requirements within the SRS must be identified and resolved. The types of conflicts that generally occur are:

Characteristics (format details) of real word entity interfacing with the system maybe conflicting. For an example, one requirements states that an individual can work up o 6 hours whereas another requirement state is as 8 hours.

The terminology used for some entities events may be different for example different requirements may use different terms to refer to the same objects.

5. Verifiability: A SRS is verifiable if and only if every stated requirement is verifiability. A requirement is verifiable if there exists some cost effective process that can check whether the final software meets that requirements un ambiguity is essential for verifiability of requirements is often done through reviews it also implies that SRS in understandable, at least by the developer the client and the users.

6.Ranking : Generally, the requirements are stated according to their priorities are critical, other are important but not critical, and there are some which are desirable but not very important. Similarly some requirements are core requirements which are not likely to change as time passes, while others are more dependent on time. A SRS is ranked for importance and or stability if for each requirement the importance and the stability of the requirements are indicated.

7. Modifiability: The SRS needs to be documented in such a manner that a history of changes can be contained in the document. It will also necessary to be able to highlight and tr5ace the changes of the requirements as we progress through the project. Certain good practices (that can lead to high modifiability are minimal redundancy and the numbering of the requirements. According to IEEE, standard 830-1993 (recommended practice for software requirements specification) SRS is modifiable if its structure and style are such that any necessary changes to the requirements can be made easily while preserving completeness an consistency while retaining its structure and style.

8. Traceability: As SRS is traceable if the origin of each its requirements is clear and if it facilitates the referencing of each requirements in future development. There are two types of traceability.

Backward traceability: to the documentation and other work products created prior to the requirements phase. This will depend upon how well referencing and labelling has been provided in the previous documents work products,

Forward traceability: will depend upon how each requirement in the SRS is labelled numbered. This traceability is extremely important, as this is one of the ways of tracing requirements to its implementation. Traceability aids verification and validation.

9. Feasibility: Though it may not be possible to confirm the feasibility of implementation of all the requirements any requirement which is apparent infeasible, should be eliminated from the SRS.