Software Engineering

Software engineering is the study of and practice of engineering to build, design, develop, maintain, and retire software. There are different areas of software engineering and it serves many functions throughout the application lifecycle. Effective software engineering requires software engineers to be educated about good software engineering best practices, disciplined and cognizant of how your company develops software, the operation it will fulfill, and how it will be maintained.

Software engineering is a new era as CIOs and Digital Leaders now understand the importance of software engineering and the impact – both good and bad – it can have on your bottom line. Vendors, IT staff, and even departments outside of IT need to be aware that software engineering is increasing in its impact – it is affecting almost all aspects of your daily business.

Importance of Software Engineering

Software engineers of all kinds, full-time staff, vendors, contracted workers, or part-time workers, are important members of the IT community.

What do software engineers do? Software engineers apply the principles of software engineering to the design, development, maintenance, testing, and evaluation of software. There is much discussion about the degree of education and or certification that should be required for software engineers.

Software engineers are well versed in the software development process, though they typically need input from IT leader regarding software requirements and what the end result needs to be. Regardless of formal education, all software engineers should work within a specific set of best practices for software engineering so that others can do some of this work at the same time.

Software engineering almost always includes a vast amount of teamwork. Designers, writers, coders, testers, various team members, and the entire IT team need to understand the code.

Software engineering is important because specific software is needed in almost every industry, in every business, and for every function. It becomes more important as time goes on – if something breaks within your application portfolio, a quick, efficient, and effective fix needs to happen as soon as possible.

Whatever you need software engineering to do – it is something that is vitally important and that importance just keeps growing. When you work with software engineers, you need to have a check and balance system to see if they are living up to their requirements and meeting KPIs.

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.

Software Development Life Cycle (SDLC) phases

There are various software development approaches defined and designed which are used/employed during development process of software, these approaches are also referred as "Software Development Process Models" (e.g.**Waterfall model,incremental model,V-model,iterative model,RAD model,Agile model,Spiral model,Prototype model** etc.). Each process model follows a particular life cycle in order to ensure success in process of software development.

Software life cycle models describe phases of the software cycle and the order in which those phases are executed. Each phase produces deliverables required by the next phase in the life cycle. Requirements are translated into design.Code is produced according to the design which is called development phase. After coding and development the testing verifies the deliverable of the implementation phase against requirements. The testing team follows Software Testing Life Cycle (STLC)which is similar to the development cycle followed by the development team.

There are following seven phases in every Software development life cycle model:

1. Requirement gathering and analysis
2. Feasibility Study
3. Design
4. Coding
5. Testing
6. Implementation
7. Maintenance

1) Requirement gathering and analysis:

Business requirements are gathered in this phase. This phase is the main focus of the project managers and stake holders. Meetings with managers, stake holders and users are held in order to determine the requirements like;

Who is going to use the system?

How will they use the system?

What data should be input into the system?

What data should be output by the system?

These are general questions that get answered during a requirements gathering phase. After requirement gathering these requirements are analyzed for their validity and the possibility of incorporating the requirements in the system to be development is also studied. Finally, a Requirement Specification document is created which serves the purpose of guideline for the next phase of the model. The testing team follows the Software Testing Life Cycle and starts the Test Planning phase after the requirements analysis is completed.

2) Feasibility Study:

In case the system proposal is acceptable to the management, the next phase is to examine the feasibility of the system. The feasibility study is basically the test of the proposed system in the light of its workability, meeting user's requirements, effective use of resources and of course, the cost effectiveness. These are categorized as technical, operational, economic, schedule and social feasibility. The main goal of feasibility study is not to solve the problem but to achieve the scope. In the process of feasibility study, the cost and benefits are estimated with greater accuracy to find the Return on Investment (ROI). This also defines the resources needed to complete the detailed investigation. The result is a feasibility report submitted to the management. This may be accepted or accepted with modifications or rejected. In short, following decision are taken in different feasibility study:

Economic feasibility -The likely benefits outweigh the cost of solving the problem which is generally demonstrated by a cost/ benefit analysis.

Operational feasibility -Whether the problem can be solved in the user's environment with existing and proposed system workings?

Organizational feasibility –Whether the proposed system is consistent with the organization's strategic objectives?

Technical feasibility -Whether the problem be solved using existing technology and resources available?

Social feasibility –Whetherthe problem be solved without causing any social issues? Whether the system will be acceptable to the society?

3) System Design:

Based on the user requirements and the detailed analysis of a new system, the new system must be designed. This is the phase of system designing. It is the most crucial phase inthe development of a system. The logical system design arrived at as a result of system analysis and is converted into physical system design. In the design phase the SDLC process continues to move from the what questions of the analysis phase to the how.The logical design produced during the analysis is turned into a physical design-a detailed description of what is needed to solve original problem. Input, output, databases, forms, codification schemes and processing specifications are drawn up in detail. In the design stage, the programming language and the hardware and software platform in which the new system will run are also decided. Data structure, control process, equipment source, work load and limitation of the system, Interface, documentation, training, procedures of using the system, taking backups and staffing requirement are decided at this stage.

There are several tools and techniques used for describing the system design of the system. These tools and techniques are: Flowchart, Data flow diagram (DFD), Data dictionary, Structured English, Decision table and Decision tree.

4) Coding:

The system design needs to be implemented to make it a workable system. This demands the coding of design into computer language, i.e., programming language. This is also called the programming phase in which the programmer converts the program specifications into computer instructions, which we refer to as programs. It is an important stage where the defined procedures are transformed into control specifications by the help of a computer language. The programs coordinate the data movements and control the entire process in a system. A well written code reduces the testing and maintenance effort. It is generally felt that the programs must be modular in nature. This helps in fast development, maintenance and future changes, if required. Programming tools like compilers, interpreters and language like c, c++, and java etc., are used for coding with respect to the type of application. The right programming language should be chosen.

5) Testing:

Before actually implementing the new system into operations, a test run of the system is done removing all the bugs, if any. It is an important phase of a successful system. After coding the whole programs of the system, a test plan should be developed and run on a given set of test data. The output of the test run should match the expected results. Sometimes, system testing is considered as a part of implementation process.

Using the test data following test run are carried out

- Program test
- System test

Program test: When the programs have been coded and compiled and brought to working conditions, they must be individually tested with the prepared test data. All verification and validation be checked and any undesirable happening must be noted and debugged (error corrected).

System Test: After carrying out the program test for each of the programs of the system and errors removed, then system test is done. At this stage the test is done on actual data. The complete system is executed on the actual data. At each stage of the execution, the results or output of the system is analyzed.

During the result analysis, it may be found that the outputs are not matching the expected output of the system. In such case, the errors in the particular programs are identified and are fixed and further tested for the expected output. All independent modules be brought together and all the interfaces to be tested between multiple modules, the whole set of software is tested to establish that all modules work together correctly as an application or system or package.

When it is ensured that the system is running error-free, the users are called with their own actual data so that the system could be shown running as per their requirements

6)Implementation:

After having the user acceptance of the new system developed, the implementation phase begins. Implementation is the stage of a project during which theory is turned into practice. The major steps involved in this phase are;

- Acquisition and Installation of Hardware and Software
- Conversion
- User Training
- Documentation

The hardware and the relevant software required for running the system must be made fully operational before implementation. The conversion is also one of the most critical and expensive activities in the system development life cycle. The data from the old system needs to be converted to operate in the new format of the new system. The database needs to be setup with security and recovery procedures fully defined.During this phase, all the programs of the system are loaded onto the user's computer. After loading the system, training of the user starts.

After the users are trained about the computerized system, working has to shift from manual to computerized working. The process is calledChangeover. The following strategies are followed for changeover of the system.

1.Direct Changeover:This is the complete replacement of the old system by the new system. It is a risky approach and requires comprehensive system testing and training.

2.Parallel run: In parallel run both the systems, i.e., computerized and manual, are executed simultaneously for certaindefined period. The same data is processed by both the systems. This strategy is less risky but more expensive because of the following facts:

Manual results can be compared with the results of the computerized system.

1. The operational work is doubled.
2. Failure of the computerised system at the early stage does not affect the working of the organization, because the manual system continues to work, as it used to do.

3. Pilot run: In this type of run, the new system is run with the data from one or more ofthe previous periods for the whole or part of the system. The results are compared with the oldsystem results. It is less expensive and risky than parallel run approach. This strategy builds the confidence and the errors are traced easily without affecting the operations.

The documentation of the system is also one of the most important activity in the system development life cycle. This ensures the continuity of the system. Generally following two types of documentations are prepared for any system.

- User or Operator Documentation

- System Documentation

User Documentation: The user documentation is a complete description of the system from the user's point of view detailing how to use or operate the system. It also includes the major error messages likely to be encountered by the user.

System Documentation: The system documentation contains the details of system design, programs, their coding, system flow, data dictionary, process description, etc. This helps to understand the system and permit changes tobe made in the existing system to satisfy new user needs.

7)Maintenance :

Maintenance is necessary to eliminate errors in the system during its working life and to tune the system to any variations in its working environments. It must meet the scope of any future enhancement, future functionality and any other added functional features to cope up with the latest future needs. It has been seen that there are always some errors found in the systems that must be noted and corrected. It also means the review of the system from time to time. The review of the system is done for:

- knowing the full capabilities of the system
- knowing the required changes or the additional requirements
- studying the performance.

Systems Development Life Cycle (SDLC) puts emphasis on decision making processes that affect system cost and usefulness. These decisions must be based on full consideration of business processes, functional requirements, and economic and technical feasibility. The primary objectives of any SDLC is to deliver quality system which meets or exceed customer expectations and within cost estimates, work effectively and efficiently within the current and planned infrastructure, and is an inexpensive to maintain. SDLC establishes a logical order of events for conducting system development that is controlled, measured, documented, and ultimately improved

SDLC -- WATERFALL MODEL

The Waterfall Model was first Process Model to be introduced. It is also referred to as a linear sequential life cycle model. It is very simple to understand and use. In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases.

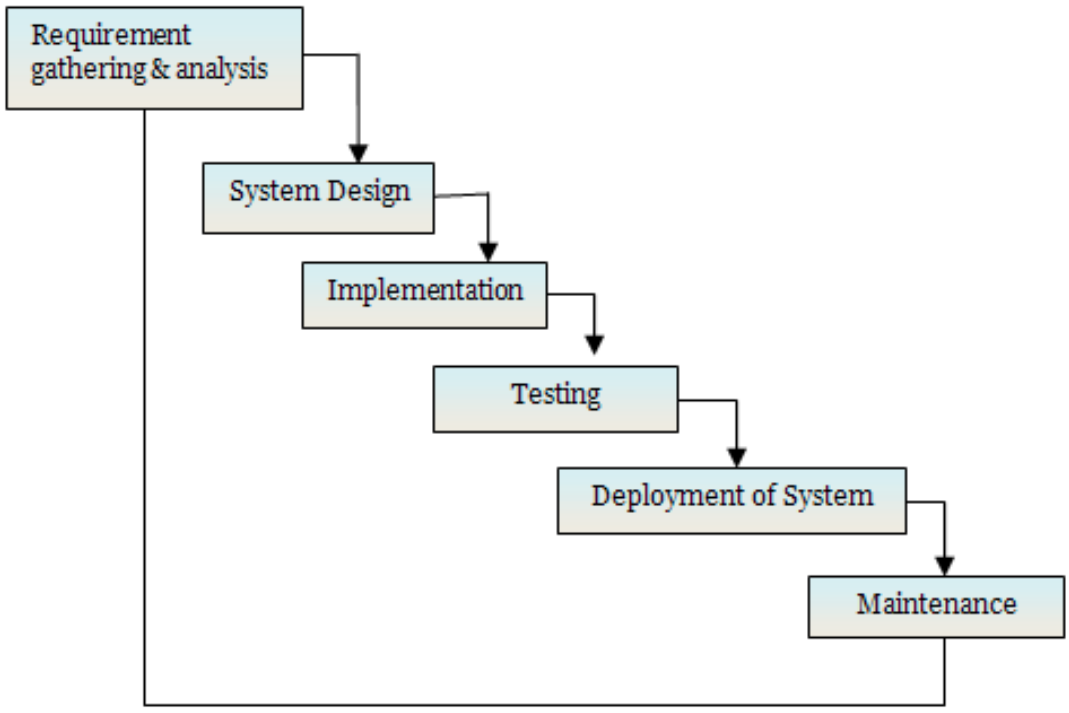
Waterfall model is the earliest SDLC approach that was used for software development .

The waterfall Model illustrates the software development process in a linear sequential flow; hence it is also referred to as a linear-sequential life cycle model. This means that any phase in the development process begins only if the previous phase is complete. In waterfall model phases do not overlap.

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 Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.

Following is a diagrammatic representation of different phases of 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 doc.

- **System Design:** The requirement specifications from first phase are studied in this phase and system design is prepared. System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture.
- **Implementation:** With inputs from 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 Pros & Cons

Advantage

The advantage of waterfall development is 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.

Disadvantage

The disadvantage of waterfall development is that it does not allow for 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 following table lists out the pros and cons of Waterfall model:

Pros	Cons
Simple and easy to understand and use.	No working software is produced until late during the life cycle.
Easy to manage due to the rigidity of the model . ach phase has specific deliverables and a review process.	High amounts of risk and uncertainty.
Phases are processed and completed one at a time.	Not a good model for complex and object-oriented projects.
Works well for smaller projects where requirements are very well understood.	Poor model for long and ongoing projects.
Clearly defined stages.	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.
Well understood milestones.	It is difficult to measure progress within stages.
Easy to arrange tasks.	Cannot accommodate changing requirements.
Process and results are well documented.	No working software is produced until late in the life cycle.
	Adjusting scope during the life cycle can end a project.
	Integration is done as a "big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early.

ITERATIVE MODEL

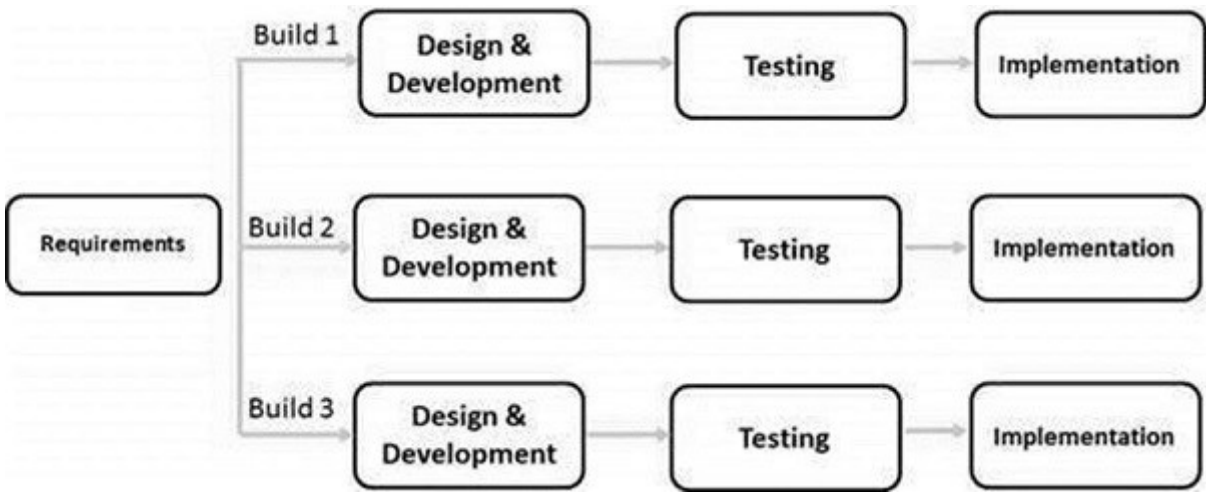
In Iterative model, iterative process starts with a simple implementation of a small set of thesoftware requirements and iteratively enhances the evolving versions until the complete system isimplemented and ready to be deployed.

An iterative life cycle model does not attempt to start with a full specification of requirements.Instead, development begins by specifying and implementing just part of the software, which isthen reviewed in order to identify further requirements. This process is then repeated, producing anew version of the software at the end of each iteration of the model.

Iterative Model design

Iterative process starts with a simple implementation of a subset of the software requirements anditeratively enhances the evolving versions until the full system is implemented. At each iteration,design modifications are made and new functional capabilities are added. The basic idea behindthis method is to develop a system through repeated cycles iterative and in smaller portions at atime incremental.

Following is the pictorial representation of Iterative and Incremental model;



Iterative and Incremental development is a combination of both iterative design or iterativemethod and incremental build model for development. "During software development, more thanone iteration of the software development cycle may be in progress at the same time." and "Thisprocess may be described as an "evolutionary acquisition" or "incremental build" approach."

In incremental model the whole requirement is divided into various builds. During each iteration,the development module goes through the requirements, design, implementation and testingphases. Each subsequent release of the module adds function to the previous release. The processcontinues till the complete system is ready as per the requirement.

The key to successful use of an iterative software development lifecycle is rigorous validation ofrequirements, and verification & testing of each version of the software against thoserequirements within each cycle of the model. As the software evolves through successive cycles,tests have to be repeated and extended to verify each version of the software.

Iterative Model Application

Like other SDLC models, Iterative and incremental development has some specific applications inthe software industry. This model is most often used in the following scenarios:

- Requirements of the complete system are clearly defined and understood.
- Major requirements must be defined; however, some functionalities or requested enhancements may evolve with time.
- There is a time to the market constraint.
- A new technology is being used and is being learnt by the development team while working on the project.
- Resources with needed skill set are not available and are planned to be used on contractbasis for specific iterations.
- There are some high risk features and goals which may change in the future.

Iterative Model Pros and Cons

The advantage of this model is that there is a working model of the system at a very early stage ofdevelopment which makes it easier to find functional or design flaws. Finding issues at an earlystage of development enables to take corrective measures in a limited budget.

The disadvantage with this SDLC model is that it is applicable only to large and bulky softwaredevelopment projects. This is because it is hard to break a small software system into further small serviceable increments/modules.

The following table lists out the pros and cons of Iterative and Incremental SDLC Model;

Some working functionality can bedeveloped quickly and early in the lifecycle.	More resources may be required.Not suitable for smaller projects.
Results are obtained early andperiodically.Parallel development can be planned.	Although cost of change is lesser but it isnot very suitable for changingrequirements.
Progress can be measured.	More management attention is required.
Less costly to change thescope/requirements.	System architecture or design issuesmay arise because not all requirementsare gathered in the beginning of theentire life cycle.

Testing and debugging during smaller iteration is easy.	Defining increments may requiredefinition of the complete system.
Risks are identified and resolved duringiteration; and each iteration is an easilymanaged milestone.	Management complexity is more.End of project may not be known which isa risk.
Easier to manage risk - High risk part isdone first.	Highly skilled resources are required forrisk analysis.
With every increment operationalproduct is delivered.	Project's progress is highly dependentupon the risk analysis phase.
Issues, challenges & risks identified fromeach increment can be utilized/appliedto the next increment.	
Risk analysis is better.It supports changing requirements.	
Initial Operating time is less.	
Better suited for large and mission-critical projects.	
During life cycle software is producedearly which facilitates customer evaluation and feedback	

SOFTWARE PROTOTYPE MODEL

The Software Prototyping refers to building software application prototypes which display the functionality of the product under development but may not actually hold the exact logic of the original software.

Software prototyping is becoming very popular as a software development model, as it enables to understand customer requirements at an early stage of development. It helps get valuable feedback from the customer and helps software designers and developers understand about what exactly is expected from the product under development.

What is Software Prototyping?

- Prototype is a working model of software with some limited functionality.
- The prototype does not always hold the exact logic used in the actual software application and is an extra effort to be considered under effort estimation.
- Prototyping is used to allow the users evaluate developer proposals and try them out before implementation.
- It also helps understand the requirements which are user specific and may not have been considered by the developer during product design.

Following is the stepwise approach to design a software prototype:

Basic Requirement Identification: This step involves understanding the very basics product requirements especially in terms of user interface. The more intricate details of the internal design and external aspects like performance and security can be ignored at this stage.

Developing the initial Prototype: The initial Prototype is developed in this stage, where the very basic requirements are showcased and user interfaces are provided. These features may not exactly work in the same manner internally in the actual software developed and the workarounds are used to give the same look and feel to the customer in the prototype developed.

Review of the Prototype: The prototype developed is then presented to the customer and the other important stakeholders in the project. The feedback is collected in an organized manner and used for further enhancements in the product under development.

Revise and enhance the Prototype: The feedback and the review comments are discussed during this stage and some negotiations happen with the customer based on factors like ,time and budget constraints and technical feasibility of actual implementation. The changes accepted are again incorporated in the new Prototype developed and the cycle repeats until customer expectations are met.

Prototypes can have horizontal or vertical dimensions. Horizontal prototype displays the user interface for the product and gives a broader view of the entire system, without concentrating on internal functions. A vertical prototype on the other side is a detailed elaboration of a specific function or a sub system in the product.

The purpose of both horizontal and vertical prototype is different. Horizontal prototypes are used to get more information on the user interface level and the business requirements. It can even be presented in the sales demos to get business in the market. Vertical prototypes are technical in nature and are used to get details of the exact functioning of the sub systems. For example, database requirements, interaction and data processing loads in a given sub system.

Software Prototyping Types

There are different types of software prototypes used in the industry. Following are the major software prototyping types used widely:

Throwaway/Rapid Prototyping: Throwaway prototyping is also called as rapid or close ended prototyping. This type of prototyping uses very little efforts with minimum requirement analysis to build a prototype. Once the actual requirements are understood, the prototype is discarded and the actual system is developed with a much clear understanding of user requirements.

Evolutionary Prototyping: Evolutionary prototyping also called as breadboard prototyping is based on building actual functional prototypes with minimal functionality in the beginning. The prototype developed forms the heart of the future prototypes on top of which the entire system is built. Using evolutionary prototyping only well understood

requirements are included in the prototype and the requirements are added as and when they are understood.

Incremental Prototyping: Incremental prototyping refers to building multiple functional prototypes of the various sub systems and then integrating all the available prototypes to form a complete system.

Extreme Prototyping : Extreme prototyping is used in the web development domain. It consists of three sequential phases. First, a basic prototype with all the existing pages is presented in the html format. Then the data processing is simulated using a prototype services layer. Finally the services are implemented and integrated to the final prototype. This process is called Extreme Prototyping used to draw attention to the second phase of the process, where a fully functional UI is developed with very little regard to the actual services.

Software Prototyping Application

Software Prototyping is most useful in development of systems having high level of user interactions such as online systems. Systems which need users to fill out forms or go through various screens before data is processed can use prototyping very effectively to give the exact look and feel even before the actual software is developed.

Software that involves too much of data processing and most of the functionality is internal with very little user interface does not usually benefit from prototyping. Prototype development could be an extra overhead in such projects and may need lot of extra efforts.

Software Prototyping Pros and Cons

Software prototyping is used in typical cases and the decision should be taken very carefully so that the efforts spent in building the prototype add considerable value to the final software developed. The model has its own pros and cons discussed as below.

Following table lists out the pros and cons of Big Bang Model:

Pros	Cons
Increased user involvement in the product even before implementation	Risk of insufficient requirement analysis owing to too much dependency on prototype
Since a working model of the system is displayed, the users get a better understanding of the system being developed.	Users may get confused in the prototypes and actual systems.
Reduces time and cost as the defects can be detected much earlier.	Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans.
Quicker user feedback is available leading to better solutions.	Developers may try to reuse the existing prototypes to build the actual system, even when its not technically feasible

Missing functionality can be identified easily	The effort invested in building prototypes may be too much if not monitored properly
Confusing or difficult functions can be identified	

SDLC - SPIRAL MODEL

The spiral model combines the idea of iterative development with the systematic, controlled aspects of the waterfall model.

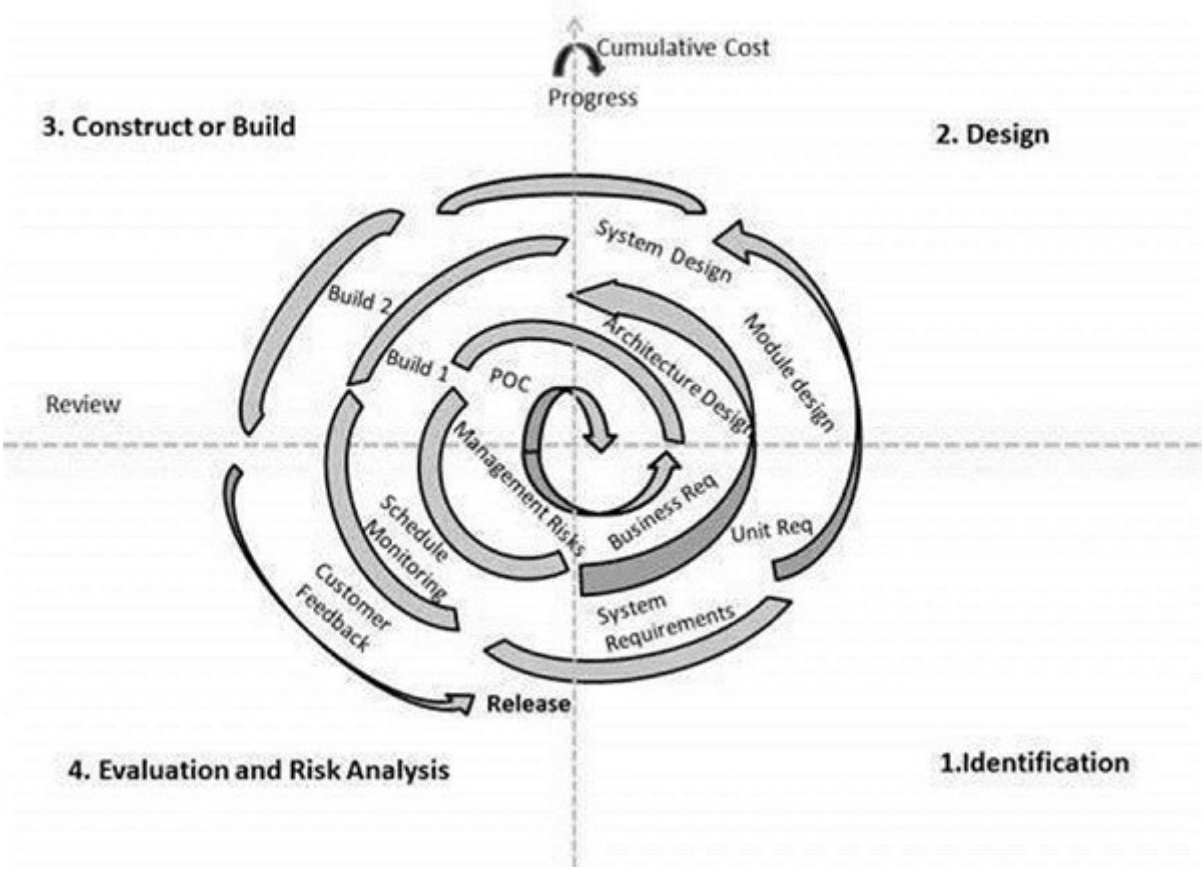
Spiral model is a combination of iterative development process model and sequential lineardevelopment model i.e. waterfall model with very high emphasis on risk analysis.It allows for incremental releases of the product, or incremental refinement through each iterationaround the spiral.

Spiral Model design

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

- **Identification:**This phase starts with gathering the business requirements in the baselinespiral. In the subsequent spirals as the product matures, identification of systemrequirements, subsystem requirements and unit requirements are all done in this phase.This also includes understanding the system requirements by continuous communicationbetween the customer and the system analyst. At the end of the spiral the product isdeployed in the identified market.
- **Design:**Design phase starts with the conceptual design in the baseline spiral and involvesarchitectural design, logical design of modules, physical product design and final design inthe subsequent spirals.
- **Construct or Build**:Construct phase refers to production of the actual software product atevery spiral. In the baseline spiral when the product is just thought of and the design is beingdeveloped a POC *ProofofConcept* is developed in this phase to get customer feedback.Then in the subsequent spirals with higher clarity on requirements and design details aworking model of the software called build is produced with a version number. These buildsare sent to customer for feedback.
- **Evaluation and Risk Analysis:**Risk Analysis includes identifying, estimating, andmonitoring technical feasibility and management risks, such as schedule slippage and costoverrun. After testing the build, at the end of first iteration, the customer evaluates thesoftware and provides feedback.

  Following is a diagrammatic representation of spiral model listing the activities in each phase:



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

Spiral Model Application

Spiral Model is very widely used in the software industry as it is in synch with the naturaldevelopment process of any product i.e. learning with maturity and also involves minimum risk forthe customer as well as the development firms. Following are the typical uses of Spiral model:

- When costs 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 therequirements 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 for 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 andallows for making an orderly transition to a maintenance activity. Another positive aspect is thatthe spiral model forces early user involvement in the system development effort.

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

The following table lists out the pros and cons of Spiral SDLC Model;

Pro	Cons
• Changing requirements can beaccommodated. • Allows for extensive use of prototypes • Requirements can be captured moreaccurately. • Users see the system early. • Development can be divided into smallerparts and more risky parts can bedeveloped earlier which helps better riskmanagement.	• Management is more complex. • End of project may not be known early. • Not suitable for small or low risk projectsand could be expensive for smallprojects. • Process is complexSpiral may go indefinitely. • Large number of intermediate stagesrequires excessive documentation.