UNIT :1

Introduction To Software Engineering

Prepared by :Amit Bhaliya

BCA SEM-5

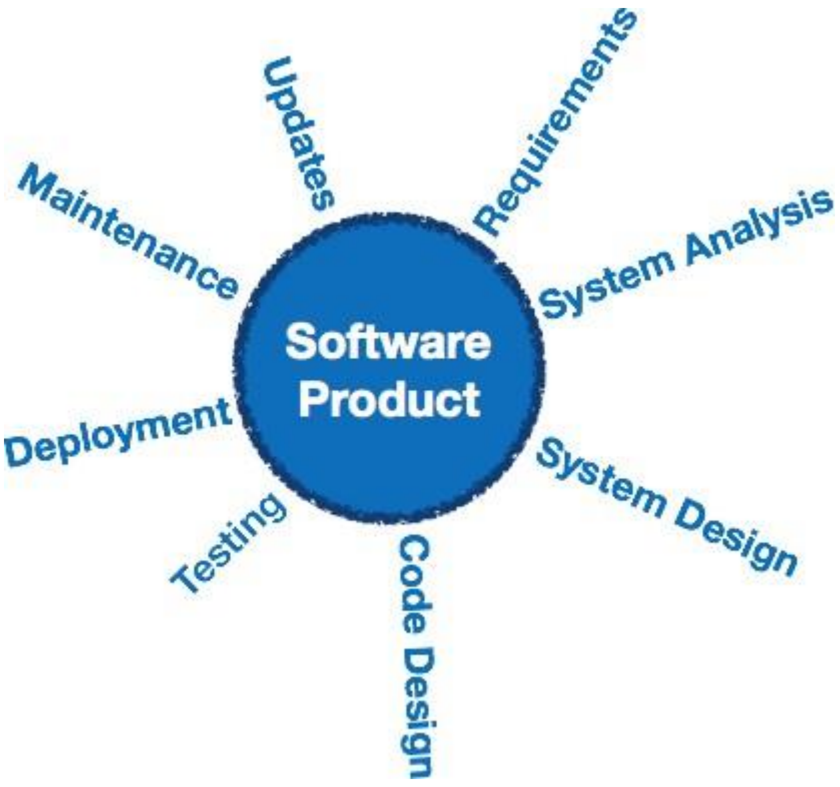
2020

P A R E K H C O L L E G E C A M P U S M A H U V A - 3 6 4 2 9 0

1.What is Software Engineering?

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

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



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

Definitions

IEEE defines software engineering as:

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

- The study of approaches as in the above statement.

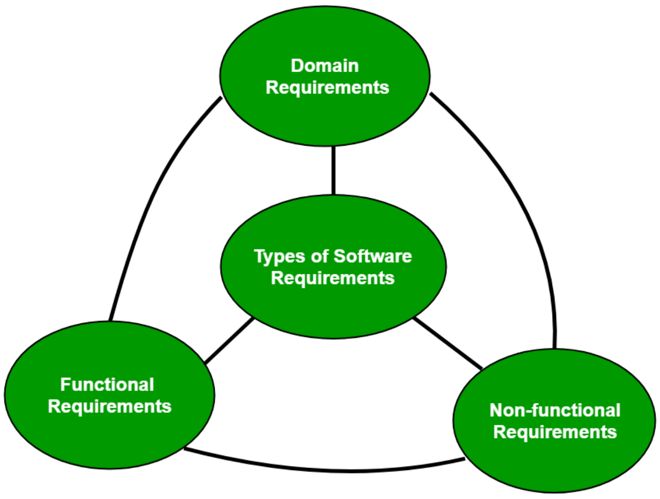
Fritz Bauer, a German computer scientist, defines software engineering as:

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

Que : 2.What is software Requirement? Explain their types.

According to IEEE standard 729, a requirement is defined as follows:
A condition or capability needed by a user to solve a problem or achieve an objective
A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification or other formally imposed documents
A documented representation of a condition or capability as in 1 and 2.

A software requirement can be of 3 types:
1.Functional requirements
2.Non-functional requirements
3.Domain requirements



1.Functional Requirements: These are the requirements that the end user specifically demands as basic facilities that the system should offer. All these functionalities need to be necessarily incorporated into the system as a part of the contract. These are represented or stated in the form of input to be given to the system, the operation performed and the output expected. They are basically the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements.
For example, in a hospital management system, a doctor should be able to retrieve the information of his patients. Each high-level functional requirement may involve several interactions or dialogues between the system and the outside world. In order to accurately describe the functional requirements, all scenarios must be enumerated.

2.Non-functional requirements: These are basically the quality constraints that the system must satisfy according to the project contract. The priority or extent to which these factors are implemented varies from one project to other. They are also called non-behavioral requirements. They basically deal with issues like:

- Portability
- Security
- Maintainability
- Reliability
- Scalability
- Performance
- Reusability
- Flexibility

NFR's are classified into following types:

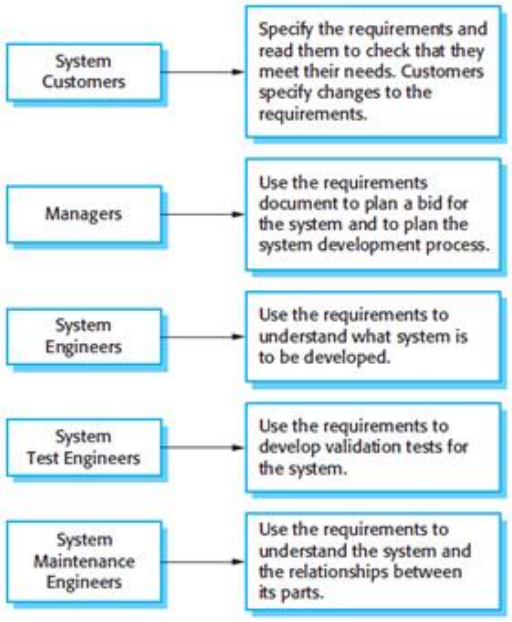
- Interface constraints
- Performance constraints: response time, security, storage space, etc.
- Operating constraints
- Life cycle constraints: maintainability, portability, etc.
- Economic constraints

3. **Domain requirements:** Domain requirements are the requirements which are characteristic of a particular category or domain of projects. The basic functions that a system of a specific domain must necessarily exhibit come under this category. For instance, in academic software that maintains records of a school or college, the functionality of being able to access the list of faculty and list of students of each grade is a domain requirement. These requirements are therefore identified from that domain model and are not user specific.
Don't stop now and take your learning to the next level. Learn all the important concepts of Data Structures and Algorithms with the help of the most trusted course: DSA Self Paced. Become industry ready at a student-friendly price.

Que: 3 What is SRS? Explain their types.

A **software requirements specification** (SRS) is a detailed description of a software system to be developed with its functional and non-functional requirements. The SRS is developed based the agreement between customer and contractors. It may include the use cases of how user is going to interact with software system. The software requirement specification document consistent of all necessary requirements required for project development. To develop the software system we should have clear understanding of Software system. To achieve this we need to continuous communication with customers to gather all requirements.

A good SRS defines the how Software System will interact with all internal modules, hardware, communication with other programs and human user interactions with wide range of real life scenarios. Using the *Software requirements specification* (SRS) document on QA lead, managers creates test plan. It is very important that testers must be cleared with every detail specified in this document in order to avoid faults in test cases and its expected results.


System Customers
Specify the requirements and read them to check that they meet their needs. Customers specify changes to the requirements.
Managers
Use the requirements document to plan a bid for the system and to plan the system development process.
System Engineers
Use the requirements to understand what system is to be developed.
System Test Engineers
Use the requirements to develop validation tests for the system.
System Maintenance Engineers
Use the requirements to understand the system and the relationships between its parts.

Characteristics of SRS:

1.**Correct**: An SRS is correct if, and only if, every requirement stated therein is one that the software shall meet. However, generally speaking there is no tool or procedure to ensure correctness.

2.**Unambiguous**: An SRS is unambiguous if, and only if, every requirement stated therein has only one interpretation. As a minimum, this requires that each characteristic of the final product be described using a single unique term.

3.**Complete:** An SRS is complete if, and only if, it includes the following elements:

a. All significant requirements, whether relating to functionality, performance, design constraints, attributes, or external interfaces. In particular, any external requirements imposed by a system specification should be acknowledged and treated.

b. Definition of the responses of the software to all realizable classes of input data in all realizable classes of situations. Note that it is important to specify the responses to both valid and invalid input values.

c. Full labels and references to all figures, tables, and diagrams in the SRS and definition of all terms and units of measure.

4.**Consistent**: Consistency refers to internal consistency. If an SRS does not agree with some higher-level document, such as a system requirements specification, then it is a violation of correctness. An SRS is internally consistent if, and only if, no subset of individual requirements described in it conflict.

5.**Ranked for importance and/or stability**: An SRS is ranked for importance and/or stability if each requirement in it has an identifier to indicate either the importance or stability of that particular requirement. Typically, all of the requirements that relate to a software product are not equally important.

6.**Verifiable**: An SRS is verifiable if, and only if, every requirement stated therein is verifiable. A requirement is verifiable if, and only if, there exists some finite cost-effective process with which a person or machine can check that the software product meets the requirement. In general any ambiguous requirement is not verifiable. Nonverifiable requirements include statements such as "works well", "good human interface", and "shall usually happen".

7.**Modifiable**: An SRS is modifiable if, and only if, its structure and style are such that any changes to the requirements can be made easily, completely, and consistently while retaining the structure and style. Modifiability generally requires an SRS to

a. Have a coherent and easy-to-use organization with a table of contents, an index, and explicit crossreferencing;

b. Not be redundant (i.e., the same requirement should not appear in more than one place in the SRS);

c. Express each requirement separately, rather than intermixed with other requirements.

8.Traceable – an SRS is traceable if the origin of each of its requirements is clear and if it makes it easy to reference each requirement in future development.

Que: 4 Explain Need for SRS.

- A good SRS defines the how Software System will interact with all internal modules, hardware, communication with other programs and human user interactions with wide range of real life scenarios.
- ◦ Using the SRS document on QA lead, managers creates test plan.
- ◦It is very important that testers must be cleared with every detail specified in this document in order to avoid faults in test cases and its expected results.
- ◦ The SRS is approved by product owner personally, the end solution won't differ a bit from an initial concept a client expected to get.
- The need for maintaining a requirements document is that the modeling activity essentially focuses on the problem structure and not its structural behavior. While in SRS, performance constraints, design constraints, and standard compliance recovery are clearly specified. This information helps in developing a proper design of the system. Various other purposes served by SRS are listed below.
- **Feedback:** Provides a feedback, which ensures to the user that the organization (which develops the software) understands the issues or problems to be solved and the software behavior necessary to address those problems.
- **Decompose problem into components:** Organizes the information and divides the problem into its component parts in an orderly manner.
- **Validation:** Uses validation strategies applied to the requirements to acknowledge that requirements are stated properly.
- **Input to design:** Contains sufficient detail in the functional system requirements to devise a design solution.
- **Basis for agreement between the** user and **the organization:** Provides a complete description of the functions to be performed by the system. In addition, it helps the users to determine whether the specified requirements are accomplished.
- **Reduce the development effort:** Enables developers to consider user requirements before the designing of the system commences. As a result, 'rework' and inconsistencies in the later stages can be reduced.
  - the developer to have a 'rough' estimate of the total cost and schedule of the project.
  - SRS is used by various individuals in the organization. System customers need SRS to specify and verify whether requirements meet the desired needs.
  - In addition, SRS enables the managers to plan for the system development processes. System engineers need a requirements document to understand what system is to be developed. These engineers also require this document to develop validation tests for the required system.
  - Lastly, requirements document is needed by system maintenance engineers to use the requirement and the relationship between its parts.

Que: 5 Explain Role for SRS.

- The users and the client get a brief idea about the software while in the initial stages.
- The purposes and the intentions as well as the expected results are properly defined. It hence lays the outline for software design.
- The desired goals are defined thereby easing off the efforts of the developers in terms of time and cost.
- It forms a basis for the agreement between the client and the developer.
- It becomes easier while transferring and using the solution elsewhere or with new customers as the basis of functioning of the software is mentioned.
- It acts as a material for reference at a later stage.
- It acts as the basis for reviews.

Que:6 Explain Software Process.

The term **software** specifies to the set of computer programs, procedures and associated documents (Flowcharts, manuals, etc.) that describe the program and how they are to be used. A software process is the set of activities and associated outcome that produce a software product. Software engineers mostly carry out these activities. These are four key process activities, which are common to all software processes. These activities are:

1. **Software specifications:** The functionality of the software and constraints on its operation must be defined.
2. **Software development:** The software to meet the requirement must be produced.
3. **Software validation:** The software must be validated to ensure that it does what the customer wants.
4. **Software evolution:** The software must evolve to meet changing client needs.

Que:7 Explain Software Characteristics.

As we know that software is any computer program which can also be defined as a set of instructions which are responsible for guiding the computer to perform certain tasks. The following are the **characteristics of software**:

1. Software does not wear out
2. Software is not manufacture
3. Usability of Software
4. Reusability of components
5. Flexibility of software
6. Maintainability of software
7. Portability of software
8. Reliability of Software

Now let us elaborate each of them...

1. Software does not wear out:
Different things like clothes, shoes, ornaments do wear out after some time. But, software once created never wears out. It can be used for as long as needed and in case of need for any updating, required changes can be made in the same software and then it can be used further with updated features.

2.Software is not manufactured:
Software is not manufactured but is developed. So, it does not require any raw material for its development.

3.Usability of Software:
The usability of the software is the simplicity of the software in terms of the user. The easier the software is to use for the user, the more is the usability of the software as more number of people will now be able to use it and also due to the ease will use it more willingly.

4.Reusability of components:
As the software never wears out, neither do its components, i.e. code segments. So, if any particular segment of code is required in some other software, we can reuse the existing code form the software in which it is already present. This reduced our work and also saves time and money.

5.Flexibility of software:
A software is flexible. What this means is that we can make necessary changes in our software in the future according to the need of that time and then can use the same software then also.

6.Maintainability of software:
Every software is maintainable. This means that if any errors or bugs appear in the software, then they can be fixed.

7.Portability of software:
Portability of the software means that we can transfer our software from one platform to another that too with ease. Due to this, the sharing of the software among the developers and other members can be done flexibly.

8.Reliability of Software:
This is the ability of the software to provide the desired functionalities under every condition. This means that our software should work properly in each condition.

Que: 8 Explain Proejct Management.

Software project management is an art and discipline of planning and supervising software projects. It is a sub-discipline of software project management in which software projects planned, implemented, monitored and controlled.
It is a procedure of managing, allocating and timing resources to develop computer software that fulfills requirements.
In software Project Management, the client and the developers need to know the length, period and cost of the project.

Prerequisite of software project management?

There are three needs for software project management. These are:
1. Time
2.Cost
3.Quality

It is an essential part of the software organization to deliver a quality product, keeping the cost within the client?s budget and deliver the project as per schedule. There are various factors, both external and internal, which may impact this triple factor. Any of three-factor can severely affect the other two.

Need of Software Project Management:

Software is an non-physical product. Software development is a new stream in business and there is very little experience in building software products. Most of the software products are made to fit client's requirements. The most important is that the basic technology changes and advances so frequently and rapidly that experience of one product may not be applied to the other one. Such type of business and environmental constraints increase risk in software development hence it is essential to manage software projects efficiently.It is necessary for an organization to deliver quality product, keeping the cost within client's budget constrain and deliver the project as per scheduled. Hence in order, software project management is necessary to incorporate user requirements along with budget and time constraints.

Software Project Management consists of several different type of managements:

1. Conflict Management:

Conflict management is the process to restrict the negative features of conflict while increasing the positive features of conflict. The goal of conflict management is to improve learning and group results including efficacy or performance in an organizational setting. Properly managed conflict can enhance group results.

2. Risk Management:

Risk management is the analysis and identification of risks that is followed by synchronized and economical implementation of resources to minimize, operate and control the possibility or effect of unfortunate events or to maximize the realization of opportunities.

3. Requirement Management:

It is the process of analyzing, prioritizing, tracing and documenting on requirements and then supervising change and communicating to pertinent stakeholders. It is a continuous process during a project.

4. Change Management:

Change management is a systematic approach for dealing with the transition or transformation of an organization's goals, processes or technologies. The purpose of change management is to execute strategies for effecting change, controlling change and helping people to adapt to change.

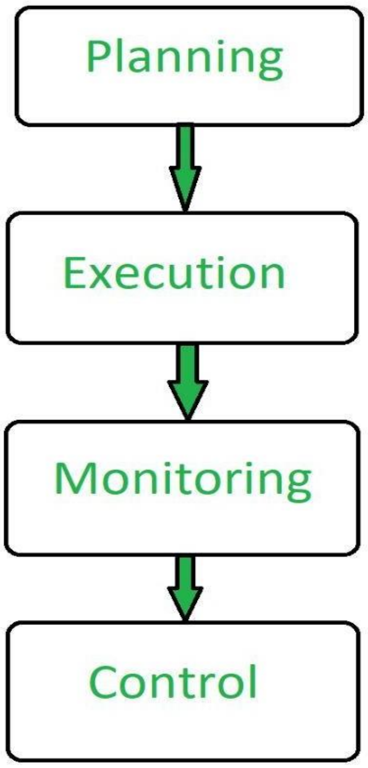
5. Software Configuration Management:

Software configuration management is the process of controlling and tracing changes in the software, part of the larger cross-disciplinary field of configuration management. Software configuration management include revision control and the inauguration of baselines.

6. Release Management:

Release Management is the task of planning, controlling and scheduling the build in deploying releases. Release management ensures that organization delivers new and enhanced services required by the customer, while protecting the integrity of existing services.

Aspects of Software Project Management:



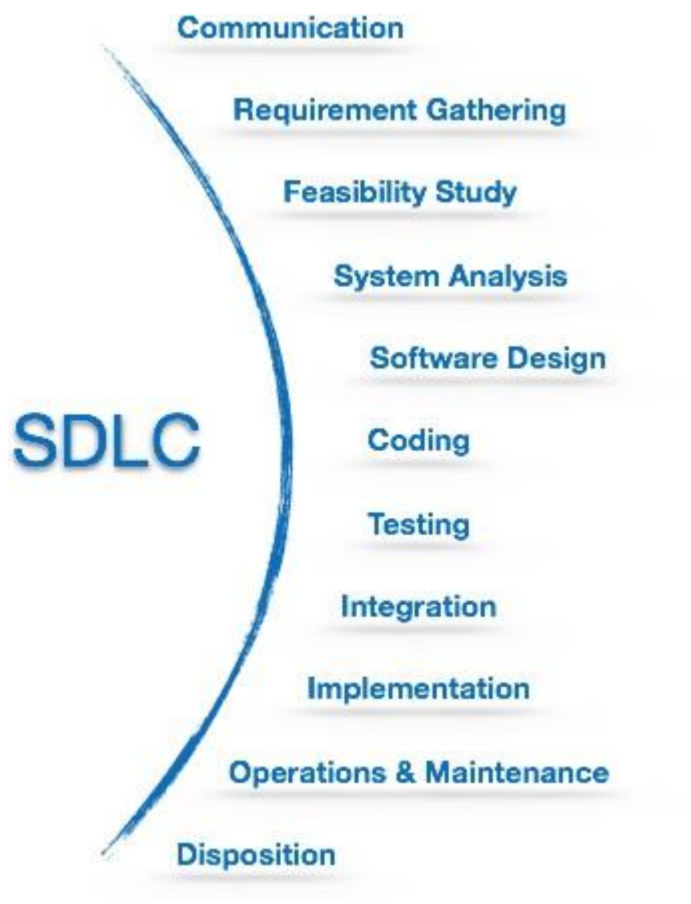
Advantages of Software Project Management:
-It helps in planning of software development.
-Implementation of software development is made easy.
-Monitoring and controlling are aspects of software project management.
-It overall manages to save time and cost for software development.

Que: 9. What is SDLC? Explain its various stages or Phase Development Process.

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

SDLCActivities

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



Communication

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

Requirement Gathering

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

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

Feasibility Study

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

System Analysis

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

Software Design

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

Coding

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

Testing

An estimate says that 50% of whole software development process should be tested. Errors may ruin the software from critical level to its own removal. Software testing is done while coding by the developers and thorough testing is conducted by testing experts at various levels of code such as module testing, program testing, product testing, in-house testing and testing the product at user's end. Early discovery of errors and their remedy is the key to reliable software.

Integration

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

Implementation

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

Operation and Maintenance

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

Disposition

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

Que :10.What is process model? Explain different types of process model.

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.

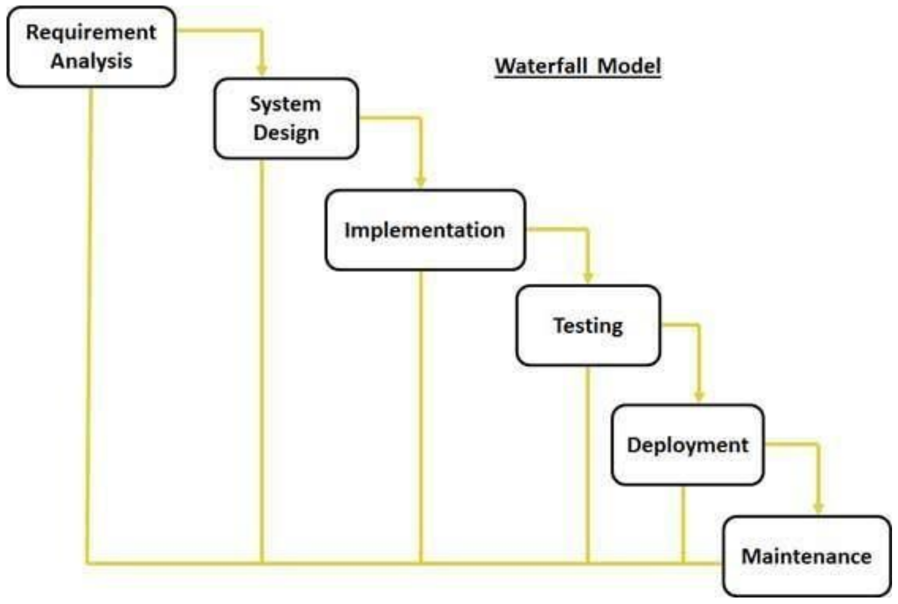
Waterfall model is the earliest SDLC approach that was used for software development .

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 • 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.	• 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. • 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

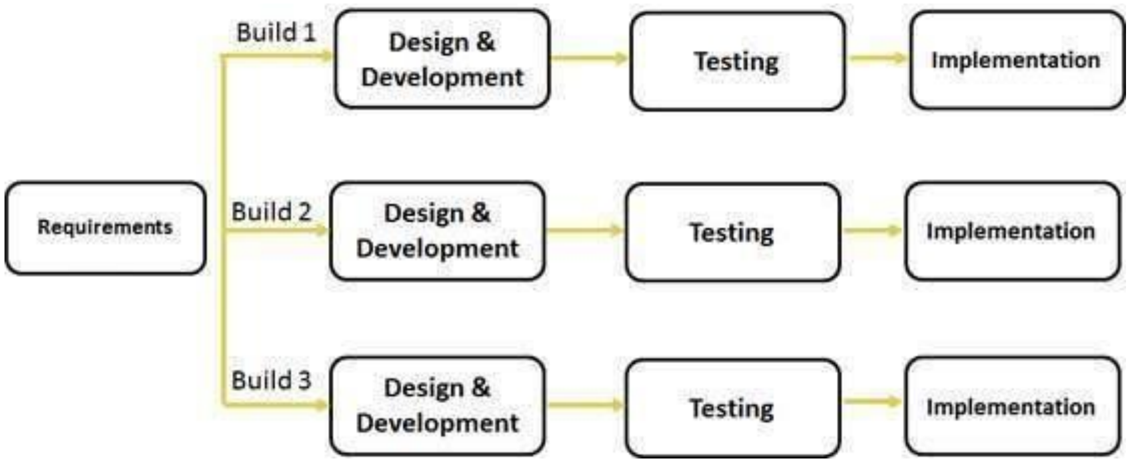
In Iterative model, iterative process starts with a simple implementation of a small set of the software requirements and iteratively enhances the evolving versions until the complete system is implemented and ready to be deployed.

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 is then reviewed in order to identify further requirements. This process is then repeated, producing a new 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 and iteratively 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 behind this method is to develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental).

Following is the pictorial representation of Iterative and Incremental model:



Iterative and Incremental development is a combination of both iterative design or iterative method and incremental build model for development. "During software development, more than one iteration of the software development cycle may be in progress at the same time." and "This process 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 testing phases. Each subsequent release of the module adds function to the previous release. The process continues till the complete system is ready as per the requirement.

The key to successful use of an iterative software development lifecycle is rigorous validation of requirements, and verification & testing of each version of the software against those requirements 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 in the 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 contract basis for specific iterations.
- There are some high risk features and goals which may change in the future.

Iterative Model Prosand Cons

The advantage of this model is that there is a working model of the system at a very early stage of development which makes it easier to find functional or design flaws. Finding issues at an early stage 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 software development 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:

Pros	Cons
• Some working functionality can be developed quickly and early in the life cycle. • Results are obtained early and periodically. • Parallel development can be planned. • Progress can be measured. • Less costly to change the scope/requirements. • Testing and debugging during smaller iteration is easy. • Risks are identified and resolved during iteration; and each iteration is an easily managed milestone. • Easier to manage risk - High risk part is done first. • With every increment operational product is delivered. • Issues, challenges & risks identified from each increment can be utilized/applied to the next increment. • Risk analysis is better.	• More resources may be required. • Although cost of change is lesser but it is not very suitable for changing requirements. • More management attention is required. • System architecture or design issues may arise because not all requirements are gathered in the beginning of the entire life cycle. • Defining increments may require definition of the complete system. • Not suitable for smaller projects. • Management complexity is more. • End of project may not be known which is a risk. • Highly skilled resources are required for risk analysis. • Project.s progress is highly dependent upon the risk analysis phase.

• Itsupports changing requirements. • Initial Operating time is less. • Better suited for large and mission-critical projects. • During life cycle software is produced early which facilitates customer evaluation and feedback.	

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 linear development 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 iteration around the spiral.

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 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.

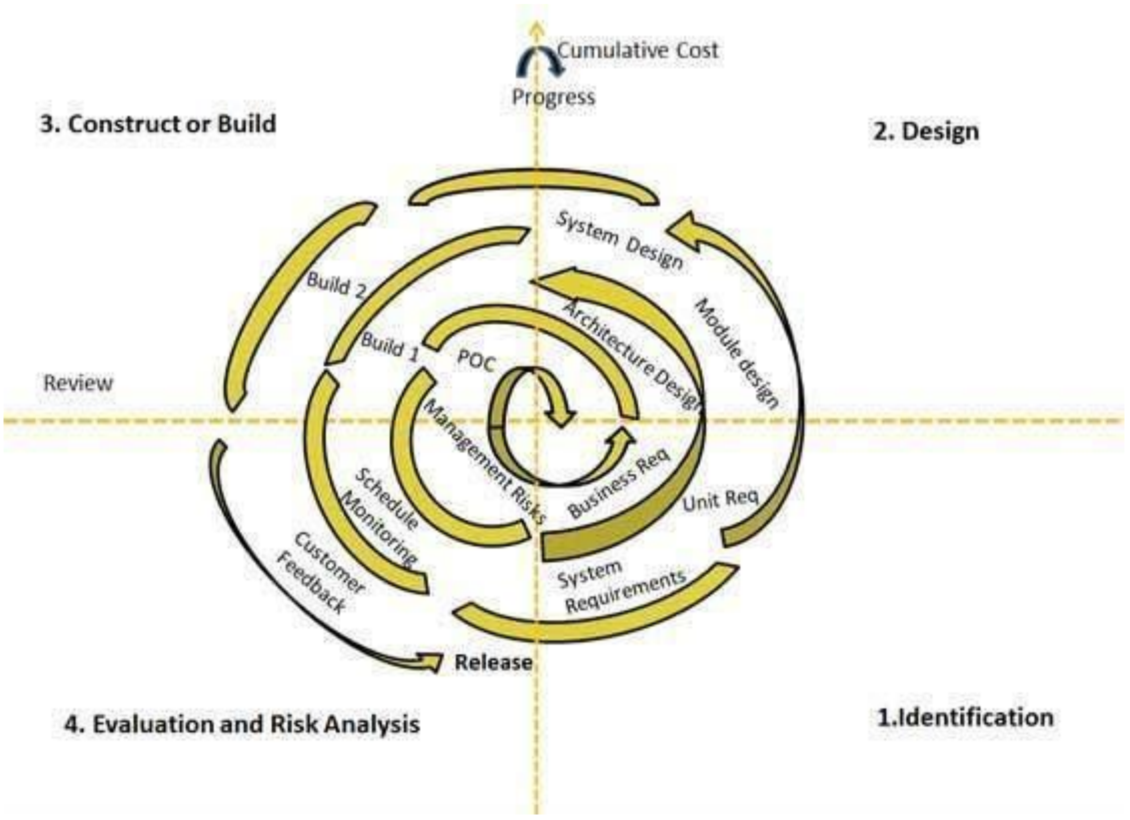
- **Design:**Design phase starts with the conceptual design in the baseline spiral and involves architectural design, logical design of modules, physical product design and final design in the subsequent spirals.

- **Construct or Build:**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 customer for feedback.

- **Evaluation and Risk Analysis:**Risk Analysis includes identifying, estimating, and monitoring 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.

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 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

Spiral Model is very widely used in the software industry as it is in synch with the natural development process of any product i.e. learning with maturity and also involves minimum risk for the 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 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 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 and allows for making an orderly transition to a maintenance activity. Another positive aspect is that the 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 of running the spiral in 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 following table lists out the pros and cons of Spiral SDLC Model:

Pros	Cons
• Changing requirements can be accommodated. • Allows for extensive use of prototypes • Requirements can be captured more accurately. • Users see the system early. • Development can be divided into smaller parts and more risky parts can be developed earlier which helps better risk management.	• Management is more complex. • End of 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 indefinitely. • Large number of intermediate stages requires excessive documentation.

v-model

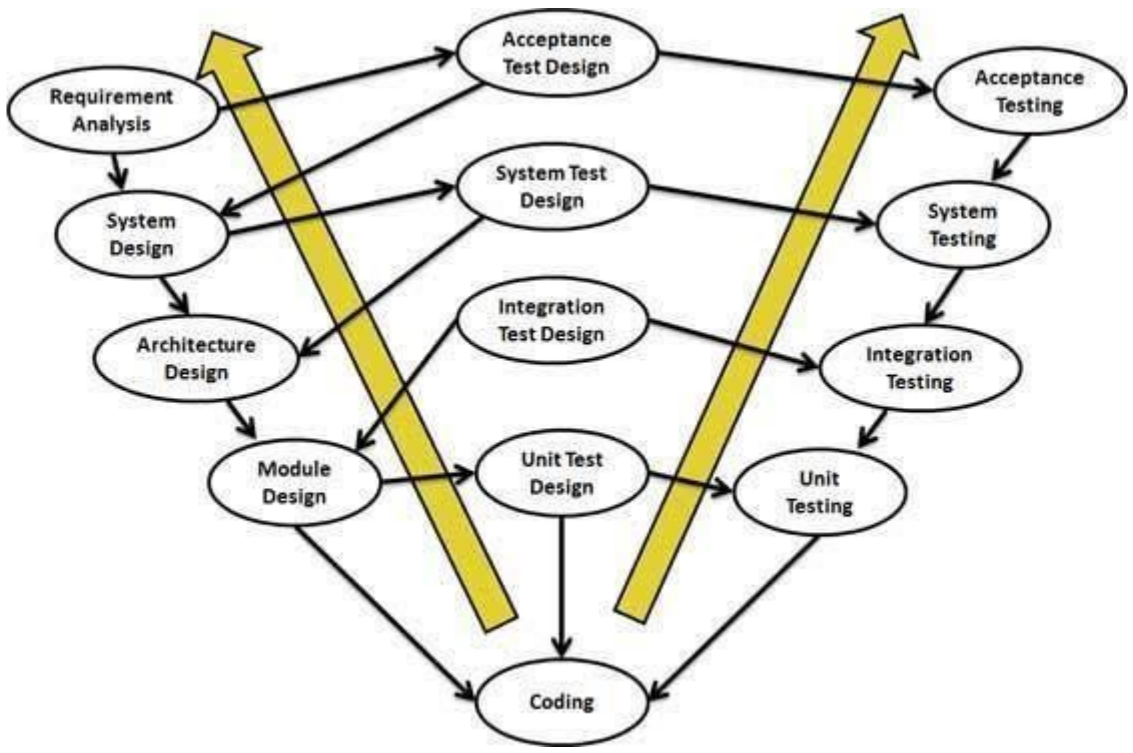
The V - model is SDLC model where execution of processes happens in a sequential manner in V-shape. It is also known as Verification and Validation model.

V - Model is an extension of the waterfall model and is based on association of a testing phase for each corresponding development stage. This means that for every single phase in the development cycle there is a directly associated testing phase. This is a highly disciplined model and next phase starts only after completion of the previous phase.

V- Model design

Under V-Model, the corresponding testing phase of the development phase is planned in parallel. So there are Verification phases on one side of the .V. and Validation phases on the other side. Coding phase joins the two sides of the V-Model.

The below figure illustrates the different phases in V-Model of SDLC.



Verification Phases

Following are the Verification phases in V-Model:

- **Business Requirement Analysis:** This is the first phase in the development cycle where the product requirements are understood from the customer perspective. This phase involves detailed communication with the customer to understand his expectations and exact requirement. This is a very important activity and need to be managed well, as most of the customers are not sure about what exactly they need. The acceptance test design planning is done at this stage as business requirements can be used as an input for acceptance testing.

- **System Design:** Once you have the clear and detailed product requirements, it.s time to design the complete system. System design would comprise of understanding and detailing the complete hardware and communication setup for the product under development. System test plan is developed based on the system design. Doing this at an earlier stage leaves more time for actual test execution later.

- **Architectural Design:** Architectural specifications are understood and designed in this phase. Usually more than one technical approach is proposed and based on the technical and financial feasibility the final decision is taken. System design is broken down further into modules taking up different functionality. This is also referred to as High Level Design (HLD).

The data transfer and communication between the internal modules and with the outside world (other systems) is clearly understood and defined in this stage. With this information, integration tests can be designed and documented during this stage.

- **Module Design:**In this phase the detailed internal design for all the system modules is specified, referred to as Low Level Design (LLD). It is important that the design is compatible with the other modules in the system architecture and the other external systems. Unit tests are an essential part of any development process and helps eliminate the maximum faults and errors at a very early stage. Unit tests can be designed at this stage based on the internal module designs.

Coding Phase

The actual coding of the system modules designed in the design phase is taken up in the Coding phase. The best suitable programming language is decided based on the system and architectural requirements. The coding is performed based on the coding guidelines and standards. The code goes through numerous code reviews and is optimized for best performance before the final build is checked into the repository.

Validation Phases

Following are the Validation phases in V-Model:

- **Unit Testing:** Unit tests designed in the module design phase are executed on the code during this validation phase. Unit testing is the testing at code level and helps eliminate bugs at an early stage, though all defects cannot be uncovered by unit testing.

- **Integration Testing:** Integration testing is associated with the architectural design phase. Integration tests are performed to test the coexistence and communication of the internal modules within the system.

- **System Testing:** System testing is directly associated with the System design phase. System tests check the entire system functionality and the communication of the system under development with external systems. Most of the software and hardware compatibility issues can be uncovered during system test execution.

- **Acceptance Testing:** Acceptance testing is associated with the business requirement analysis phase and involves testing the product in user environment. Acceptance tests uncover the compatibility issues with the other systems available in the user environment. It also discovers the non functional issues such as load and performance defects in the actual user environment.

V- Model Application

V- Model application is almost same as waterfall model, as both the models are of sequential type. Requirements have to be very clear before the project starts, because it is usually expensive to go back and make changes. This model is used in the medical development field, as it is strictly disciplined domain. Following are the suitable scenarios to use V-Model:

- Requirements are well defined, clearly documented and fixed.
- Product definition is stable.
- Technology is not dynamic and is well understood by the project team.
- There are no ambiguous or undefined requirements.
- The project is short.

V- Model Prosand Cons

The advantage of V-Model is that it.s very easy to understand and apply. The simplicity of this model also makes it easier to manage. The disadvantage is that the model is not flexible to changes and just in case there is a requirement change, which is very common in today.s dynamic world, it becomes very expensive to make the change.

The following table lists out the pros and cons of V-Model:

Pros	**Cons**
• This is a highly disciplined model and Phases are completed one at a time. • Works well for smaller projects where requirements are very well understood. • 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.	• High 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. • Once an application is in the testing stage, it is difficult to go back and

	change a functionality • No working software is produced until late during the life cycle.

Big bang model

The Big Bang model is SDLC model where we do not follow any specific process. The development just starts with the required money and efforts as the input, and the output is the software developed which may or may not be as per customer requirement.

B ig Bang Model is SDLC model where there is no formal development followed and very little planning is required. Even the customer is not sure about what exactly he wants and the requirements are implemented on the fly without much analysis.

Usually this model is followed for small projects where the development teams are very small.

Big Bang Modeldesignand Application

Big bang model comprises of focusing all the possible resources in software development and coding, with very little or no planning. The requirements are understood and implemented as they come. Any changes required may or may not need to revamp the complete software.

This model is ideal for small projects with one or two developers working together and is also useful for academic or practice projects. It.s an ideal model for the product where requirements are not well understood and the final release date is not given.

Big Bang Model Prosand Cons

The advantage of Big Bang is that its very simple and requires very little or no planning. Easy to mange and no formal procedure are required.

However the Big Bang model is a very high risk model and changes in the requirements or misunderstood requirements may even lead to complete reversal or scraping of the project. It is ideal for repetitive or small projects with minimum risks.

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

Pros	Cons
• This is a very simple model • Little or no planning required • Easy to manage • Very few resources required • Gives flexibility to developers • Is a good learning aid for new comers or students	• Very High risk and uncertainty. • Not a good model for complex and object-oriented projects. • Poor model for long and ongoing projects. • Can turn out to be very expensive if requirements are misunderstood

Agile model

Agile SDLC model is a combination of iterative and incremental process models with focus on process adaptability and customer satisfaction by rapid delivery of working software product.

Agile Methods break the product into small incremental builds. These builds are provided in iterations. Each iteration typically lasts from about one to three weeks. Every iteration involves cross functional teams working simultaneously on various areas like planning, requirements analysis, design, coding, unit testing, and acceptance testing.

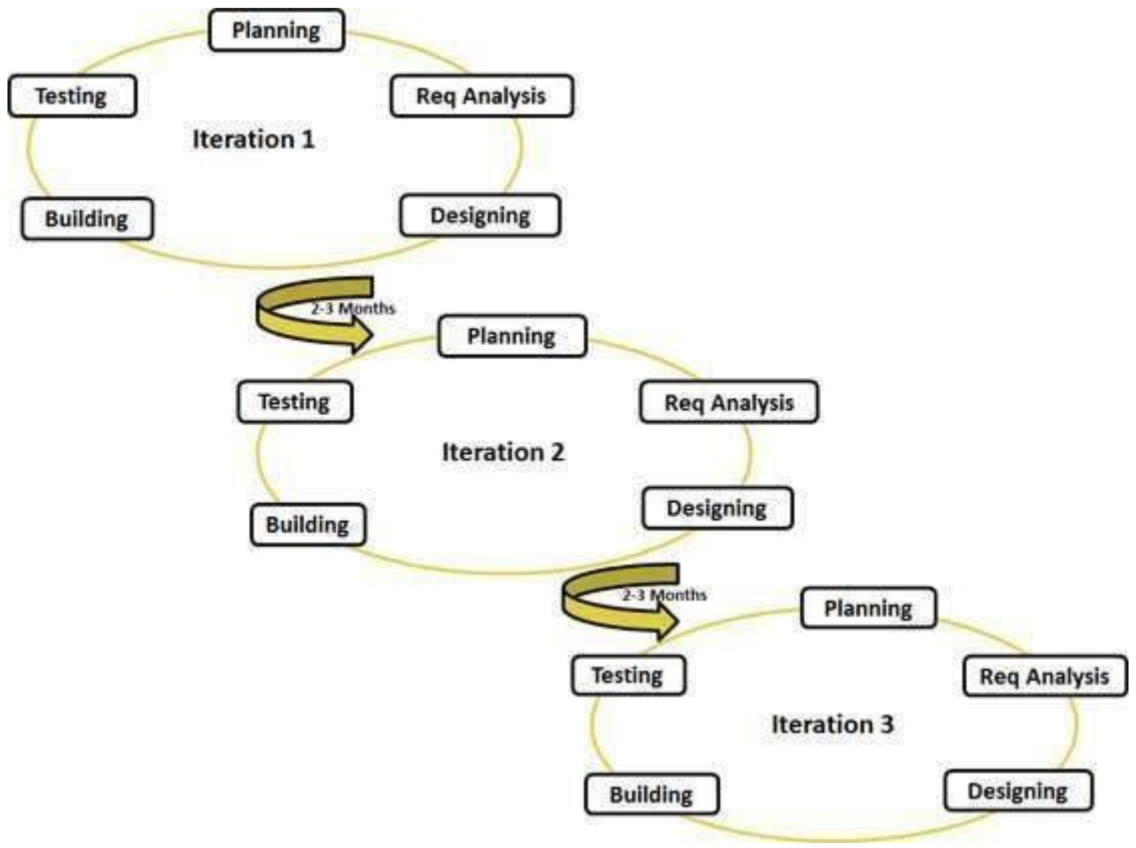
At the end of the iteration a working product is displayed to the customer and important stakeholders.

Whatis Agile?

Agile model believes that every project needs to be handled differently and the existing methods need to be tailored to best suit the project requirements. In agile the tasks are divided to time boxes (small time frames) to deliver specific features for a release.

Iterative approach is taken and working software build is delivered after each iteration. Each build is incremental in terms of features; the final build holds all the features required by the customer.

Here is a graphical illustration of the Agile Model:



Agile thought process had started early in the software development and started becoming popular with time due to its flexibility and adaptability.

The most popular agile methods include Rational Unified Process (1994), Scrum (1995), Crystal Clear, Extreme Programming (1996), Adaptive Software Development, Feature Driven Development, and Dynamic Systems Development Method (DSDM) (1995). These are now collectively referred to as agile methodologies, after the Agile Manifesto was published in 2001.

Following are the Agile Manifesto principles

- **Individuals and interactions** - in agile development, self-organization and motivation are important, as are interactions like co-location and pair programming.
- **Working software** - Demo working software is considered the best means of communication with the customer to understand their requirement, instead of just depending on documentation.
- **Customer collaboration** - As the requirements cannot be gathered completely in the beginning of the project due to various factors, continuous customer interaction is very important to get proper product requirements.
- **Responding to change** - agile development is focused on quick responses to change and continuous development.

Agile Vs Traditional SDLCModels

Agile is based on the adaptive software development methods where as the traditional SDLC models like waterfall model is based on predictive approach.

Predictive teams in the traditional SDLC models usually work with detailed planning and have a complete forecast of the exact tasks and features to be delivered in the next few months or during the product life cycle. Predictive methods entirely depend on the requirement analysis and planning done in the beginning of cycle. Any changes to be incorporated go through a strict change control management and prioritization.

Agile uses adaptive approach where there is no detailed planning and there is clarity on future tasks only in respect of what features need to be developed. There is feature driven development and the team adapts to the changing product requirements dynamically. The product is tested very frequently, through the release iterations, minimizing the risk of any major failures in future.

Customer interaction is the backbone of Agile methodology, and open communication with minimum documentation are the typical features of Agile development environment. The agile teams work in close collaboration with each other and are most often located in the same geographical location.

Agile Model Pros and Cons

Agile methods are being widely accepted in the software world recently, however, this method may not always be suitable for all products. Here are some pros and cons of the agile model.

Following table lists out the pros and cons of Agile Model:

Pros	Cons
• Is a very realistic approach to software development • Promotes teamwork and cross training. • Functionality can be developed rapidly and demonstrated. • Resource requirements are minimum.	• Not suitable for handling complex dependencies. • More risk of sustainability, maintainability and extensibility. • An overall plan, an agile leader and agile PM practice is a must without which it will not work. • Strict delivery management dictates the scope, functionality to be

• Suitable for fixed or changing requirements • Delivers early partial working solutions. • Good model for environments that change steadily. • Minimal rules, documentation easily employed. • Enables concurrent development and delivery within an overall planned context. • Little or no planning required • Easy to manage • Gives flexibility to developers	delivered, and adjustments to meet the deadlines. • Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction. • There is very high individual dependency, since there is minimum documentation generated. • Transfer of technology to new team members may be quite challenging due to lack of documentation.

Rad model

The RAD (Rapid Application Development) model is based on prototyping and iterative development with no specific planning involved. The process of writing the software itself involves the planning required for developing the product.

Rapid Application development focuses on gathering customer requirements through workshops or focus groups, early testing of the prototypes by the customer using iterative concept, reuse of the existing prototypes (components), continuous integration and rapid delivery.

Whatis RAD?

Rapid application development (RAD) is a software development methodology that uses minimal planning in favor of rapid prototyping. A prototype is a working model that is functionally equivalent to a component of the product.

In RAD model the functional modules are developed in parallel as prototypes and are integrated to make the complete product for faster product delivery.

Since there is no detailed preplanning, it makes it easier to incorporate the changes within the development process. RAD projects follow iterative and incremental model and have small teams comprising of developers, domain experts, customer representatives and other IT resources working progressively on their component or prototype.

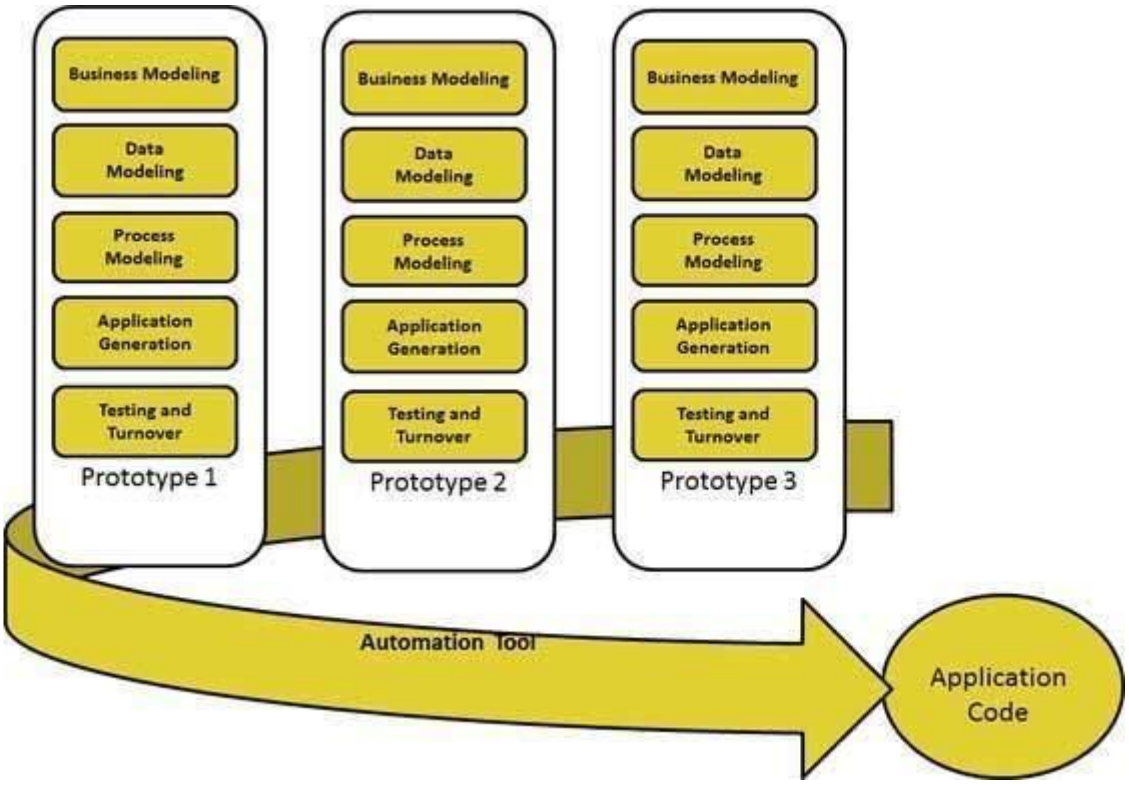
The most important aspect for this model to be successful is to make sure that the prototypes developed are reusable.

RAD Model Design

RAD model distributes the analysis, design, build, and test phases into a series of short, iterative development cycles. Following are the phases of RAD Model:

- **Business Modeling:** The business model for the product under development is designed in terms of flow of information and the distribution of information between various business channels. A complete business analysis is performed to find the vital information for business, how it can be obtained, how and when is the information processed and what are the factors driving successful flow of information.
- **Data Modeling:** The information gathered in the Business Modeling phase is reviewed and analyzed to form sets of data objects vital for the business. The attributes of all data sets is identified and defined. The relation between these data objects are established and defined in detail in relevance to the business model.
- **Process Modeling:** The data object sets defined in the Data Modeling phase are converted to establish the business information flow needed to achieve specific business objectives as per the business model. The process model for any changes or enhancements to the data object sets is defined in this phase. Process descriptions for adding , deleting, retrieving or modifying a data object are given.
- **Application Generation:** The actual system is built and coding is done by using automation tools to convert process and data models into actual prototypes.
- **Testing and Turnover:**The overall testing time is reduced in RAD model as the prototypes are independently tested during every iteration. However the data flow and the interfaces between all the components need to be thoroughly tested with complete test coverage. Since most of the programming components have already been tested, it reduces the risk of any major issues.

Following image illustrates the RAD Model:



RAD Model Vs Traditional SDLC

The traditional SDLC follows a rigid process models with high emphasis on requirement analysis and gathering before the coding starts. It puts a pressure on the customer to sign off the requirements before the project starts and the customer doesn.t get the feel of the product as there is no working build available for a long time.

The customer may need some changes after he actually gets to see the software, however the change process is quite rigid and it may not be feasible to incorporate major changes in the product in traditional SDLC.

RAD model focuses on iterative and incremental delivery of working models to the customer. This results in rapid delivery to the customer and customer involvement during the complete development cycle of product reducing the risk of non conformance with the actual user requirements.

RAD Model Application

RAD model can be applied successfully to the projects in which clear modularization is possible. If the project cannot be broken into modules, RAD may fail. Following are the typical scenarios where RAD can be used:

- RAD should be used only when a system can be modularized to be delivered in incremental manner.
- It should be used if there.s high availability of designers for modeling.
- It should be used only if the budget permits use of automated code generating tools.
- RAD SDLC model should be chosen only if domain experts are available with relevant business knowledge.
- Should be used where the requirements change during the course of the project and working prototypes are to be presented to customer in small iterations of 2-3 months.

RAD Model Pros and Cons

RAD model enables rapid delivery as it reduces the overall development time due to reusability of the components and parallel development.

RAD works well only if high skilled engineers are available and the customer is also committed to achieve the targeted prototype in the given time frame. If there is commitment lacking on either side the model may fail.

Following table lists out the pros and cons of RAD Model:

Pros	Cons
• Changing requirements can be accommodated. • Progress can be measured. • Iteration time can be short with use of powerful RAD tools. • Productivity with fewer people in short time. • Reduced development time. • Increases reusability of components	• Dependency on technically strong team members for identifying business requirements. • Only system that can be modularized can be built using RAD. • Requires highly skilled developers/designers. • High dependency on modeling skills. • Inapplicable to cheaper projects as cost of modeling and automated code generation is very high.

• Quick initial reviews occur • Encourages customer feedback • Integration from very beginning solves a lot of integration issues.	• Management complexity is more. • Suitable for systems that are component based and scalable. • Requires user involvement throughout the life cycle. • Suitable for project requiring shorter development times.

Prototype

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. • Reduces time and cost as the defects can be detected much earlier. • Quicker user feedback is available leading to better solutions. • Missing functionality can be identified easily • Confusing or difficult functions can be identified	• Users may get confused in the prototypes and actual systems. • Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans. • Developers may try to reuse the existing prototypes to build the actual system, even when its not technically feasible • The effort invested in building prototypes may be too much if not monitored properly