**Unit-1**

**Introduction to Software Engineering:**

Software is a program or set of programs containing instructions that provide desired functionality. And Engineering is the process of designing and building something that serves a particular purpose and finds a cost-effective solution to problems. 

**Software Engineering** is a systematic, disciplined, quantifiable study and approach to the design, development, operation and maintenance of a software system.

**Objectives of Software Engineering:**

1. **Maintainability:**  
   It should be feasible for the software to evolve to meet changing requirements.
2. **Efficiency:**  
   The software should not make wasteful use of computing devices such as memory, processor cycles, etc.
3. **Correctness:**  
   A software product is correct if the different requirements as specified in the SRS document have been correctly implemented.
4. **Reusability:**   
   A software product has good reusability if the different modules of the product can easily be reused to develop new products.
5. **Testability:**   
   Here software facilitates both the establishment of test criteria and the evaluation of the software with respect to those criteria.
6. **Reliability:**   
   It is an attribute of software quality. The extent to which a program can be expected to perform its desired function, over an arbitrary time period.
7. **Portability:**   
   In this case, the software can be transferred from one computer system or environment to another.
8. **Adaptability:**  
   In this case, the software allows differing system constraints and the user needs to be satisfied by making changes to the software.
9. **Interoperability**– Capability of 2 or more functional units to process data cooperatively.

# Software Process:

**Software** is the set of instructions in the form of programs to govern the computer system and to process the hardware components. To produce a software product the set of activities is used. This set is called a software process. 

**Software Development:** In this process, designing, programming, documenting, testing and bug fixing is done.

**Components of Software:**   
There are three components of the software: These are: Program, Documentation and Operating Procedures.

1. **Program:**  
   A computer program is a list of instructions that tell a computer what to do.
2. **Documentation:**  
   Source information about the product contained in design documents, detailed code comments, etc.
3. **Operating Procedures:**  
   Set of step-by-step instructions compiled by an organization to help workers carry out complex routine operations.

There are four basic key process activities:

1. **Software Specifications:**  
   In this process, detailed description of a software system to be developed with its functional and non-functional requirements.
2. **Software Development:**  
   In this process, designing, programming, documenting, testing and bug fixing is done.
3. **Software Validation:**  
   In this process, evaluation software product is done to ensure that the software meets the business requirements as well as the end user’s needs.
4. **Software Evolution –**   
   It is a process of developing software initially, then timely updating it for various reasons.

**Software Crisis:**

1. **Size and Cost:**  
   Day to day growing complexity and expectation out of software. Software is more expensive and more complex.
2. **Quality:**  
   Software products must have good quality.
3. **Delayed Delivery:**  
   Software takes longer than the estimated time to develop, which in turn leads to cost shooting up.

# Software paradigm and Software Development Life Cycle (SDLC):

Software paradigm refers to method and steps, which are taken while designing the software programming paradigm is a subset of software design paradigm which is future for other a subset of software development paradigm. Software is considered to be a collection of executable programming code, associated libraries, and documentation. Software development paradigm is also known as software engineering, all the engineering concepts pertaining to developments software applied. It consists of the following parts as Requirement Gathering, Software design, Programming, etc. The software design paradigm is a part of software development. It includes design, maintenance, programming.

[**Software development life cycle (SDLC)**](https://www.geeksforgeeks.org/software-development-life-cycle-sdlc/)**:**

SDLC is the acronym for software development life cycle. It is also called the software development process. All the tasks required for developing and maintaining software. It consists of a plan describing how to develop, maintain, replace and alter the specific software. It is a process for planning, creating, testing and information system. It is a framework of describes the activity performed at each stage of software development. It is a process used by a system analyst to develop an information system including requirements, validation, training and ownership.

**Benefits of software development life cycle:**

1. It allowed the highest level of management control.
2. Everyone understands the cost and resources required.
3. To improve the application quality and monitor the application.
4. It performs at every stage of the software development life cycle.

**Different types of software development life cycle models:**

There are various software development life cycle models. These models are referred to as the software development process models. The models defined and designed which followed during the software development process.

1. **Waterfall model:**

The waterfall model is easy to understand and simple to manage. The whole process of software development is divided into various phases. The step of requirements analysis, integration, maintenance.

1. **Iterative model:**  
   It is repetition incarnate. In short, it is breaking down the software development of large applications into smaller pieces.
2. **Spiral model:**  
   It helps the group to adopt elements of one or more process models. To develop strategies that solve uncertainty and risk.
3. **V-model:**  
   It is known as the verification and validation model. It is characterized by a corresponding testing phase for the development stage. V model joins by coding phase.
4. **Big Bang model –**  
   It focuses on all types of resources in software development and coding. Small project with smaller size development team which are working together.

**Stages of SDLC model:**  
Here, we will give you a brief overview of SDLC stages as follows.

**Stage-1: Requirement gathering:**

The feasibility report is positive towards the project and next phase start with gathering requirement from the user. Engineer communicates with the client and end-users to know their idea and which features they want to software to include.

**Stage-2: Software design:**

It is a process to transform user requirements into a suitable form. It helps programmers in software coding. There is a need for more specific and detailed requirements in software. The output of the process can directly be used in implementation in a programming language. There are three design levels as follows.

1. **Architectural design:**  
   It is the highest abstract version of the system. In a software system, many components interact with each other.
2. **High-level design:**  
   It focuses on how the system along with all its components and its can be implemented in form of modules.
3. **Detailed design:**  
   It defines the logical structure of each module and its interface to communicate with each module.

**Stage-3:** **Developing Product:**  
In this phase of SDLC, you will see how the product will be developed. It is one of the crucial parts of SDLC, it is also called the Implementation phase.

**Stage-4:** **Product Testing and Integration:**  
In this phase, we will integrate the modules and will test the overall product by using different testing techniques.

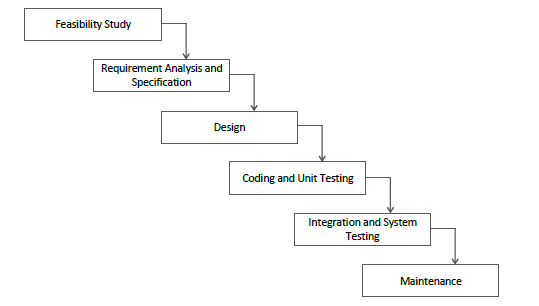
**Stage-5: Deployment and Maintenance:**  
In this phase, the actual deployment of the product, or you can say the final product will be deployed, and also, we will do maintenance of product for any future update and release of new features.

# Classical Waterfall Model:

# The classical waterfall model is the basic software development life cycle model. It is very simple but idealistic. Earlier this model was very popular but nowadays it is not used. But it is very important because all the other software development life cycle models are based on the classical waterfall model.

# The classical waterfall model divides the life cycle into a set of phases. This model considers that one phase can be started after the completion of the previous phase. That is the output of one phase will be the input to the next phase.

# Thus the development process can be considered as a sequential flow in the waterfall. Here the phases do not overlap with each other. The different sequential phases of the classical waterfall model are shown in the below figure:



Let us now learn about each of these phases in brief detail:

1. **Feasibility Study**: The main goal of this phase is to determine whether it would be financially and technically feasible to develop the software.   
   The feasibility study involves understanding the problem and then determining the various possible strategies to solve the problem. These different identified solutions are analyzed based on their benefits and drawbacks, The best solution is chosen and all the other phases are carried out as per this solution strategy.
2. **Requirements analysis and specification**:

The aim of the requirement analysis and specification phase is to understand the exact requirements of the customer and document them properly. This phase consists of two different activities.

* + **Requirement gathering and analysis:** Firstly, all the requirements regarding the software are gathered from the customer and then the gathered requirements are analyzed. The goal of the analysis part is to remove incompleteness (an incomplete requirement is one in which some parts of the actual requirements have been omitted) and inconsistencies (an inconsistent requirement is one in which some part of the requirement contradicts some other part).
  + **Requirement specification:** These analyzed requirements are documented in a software requirement specification (SRS) document. SRS document serves as a contract between the development team and customers. Any future dispute between the customers and the developers can be settled by examining the SRS document.

1. **Design**: The goal of this phase is to convert the requirements acquired in the SRS into a format that can be coded in a programming language. It includes high-level and detailed design as well as the overall software architecture. A Software Design Document is used to document all of this effort (SDD).
2. **Coding and Unit testing**: In the coding phase software design is translated into source code using any suitable programming language. Thus, each designed module is coded. The aim of the unit testing phase is to check whether each module is working properly or not.
3. **Integration and System testing**: Integration of different modules are undertaken soon after they have been coded and unit tested. Integration of various modules is carried out incrementally over a number of steps. During each integration step, previously planned modules are added to the partially integrated system and the resultant system is tested. Finally, after all the modules have been successfully integrated and tested, the full working system is obtained and system testing is carried out on this.

System testing consists of three different kinds of testing activities as described below:

* + **Alpha testing:** Alpha testing is the system testing performed by the development team.
  + **Beta testing:** Beta testing is the system testing performed by a friendly set of customers.
  + **Acceptance testing:** After the software has been delivered, the customer performed acceptance testing to determine whether to accept the delivered software or reject it.

1. **Maintenance:** Maintenance is the most important phase of a software life cycle. The effort spent on maintenance is 60% of the total effort spent to develop a full software. There are basically three types of maintenance:
   * **Corrective Maintenance:** This type of maintenance is carried out to correct errors that were not discovered during the product development phase.
   * **Perfective Maintenance:** This type of maintenance is carried out to enhance the functionalities of the system based on the customer’s request.
   * **Adaptive Maintenance:** Adaptive maintenance is usually required for porting the software to work in a new environment such as working on a new computer platform or with a new operating system.

**Advantages of Classical Waterfall Model:**

The classical waterfall model is an idealistic model for software development. It is very simple, so it can be considered the basis for other software development life cycle models. Below are some of the major advantages of this SDLC model:

* This model is very simple and is easy to understand.
* Phases in this model are processed one at a time.
* Each stage in the model is clearly defined.
* This model has very clear and well-understood milestones.
* Process, actions and results are very well documented.
* Reinforces good habits: define-before- design,   
  design-before-code.
* This model works well for smaller projects and projects where requirements are well   
  understood.

**Drawbacks of Classical Waterfall Model:**

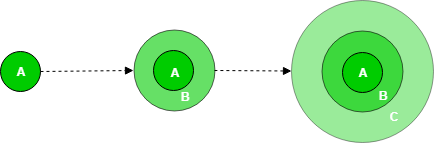
The classical waterfall model suffers from various shortcomings, basically, we can’t use it in real projects, but we use other software development lifecycle models which are based on the classical waterfall model. Below are some major drawbacks of this model:

* **No feedback path:** In the classical waterfall model evolution of software from one phase to another phase is like a waterfall. It assumes that no error is ever committed by developers during any phase. Therefore, it does not incorporate any mechanism for error correction.
* **Difficult to accommodate change requests:** This model assumes that all the customer requirements can be completely and correctly defined at the beginning of the project, but actually customers’ requirements keep on changing with time. It is difficult to accommodate any change requests after the requirements specification phase is complete.
* **No overlapping of phases:** This model recommends that a new phase can start only after the completion of the previous phase. But in real projects, this can’t be maintained. To increase efficiency and reduce cost, phases may overlap.

# Incremental Process Model:

Incremental process model is also known as the Successive version model.

First, a simple working system implementing only a few basic features is built and then that is delivered to the customer. Then thereafter many successive iterations/ versions are implemented and delivered to the customer until the desired system is released.



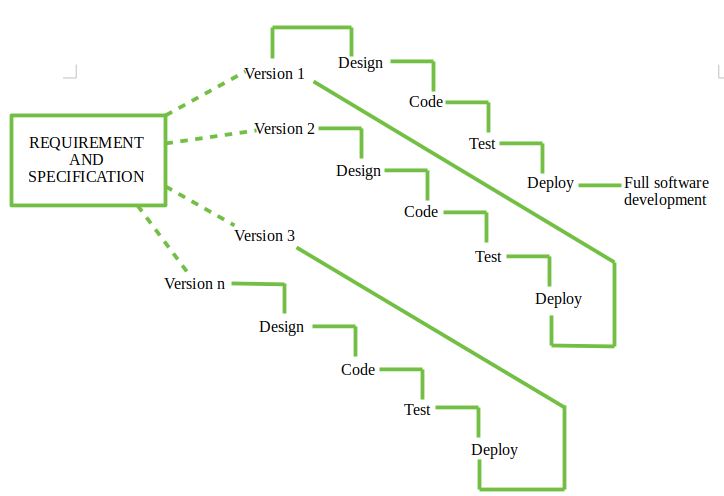
A, B, C are modules of Software Products that are incrementally developed and delivered.

**Life cycle activities:**

Requirements of Software are first broken down into several modules that can be incrementally constructed and delivered. At any time, the plan is made just for the next increment and not for any kind of long-term plan. Therefore, it is easier to modify the version as per the need of the customer. The development Team first undertakes to develop core features (these do not need services from other features) of the system.

Once the core features are fully developed, then these are refined to increase levels of capabilities by adding new functions in Successive versions. Each incremental version is usually developed using an iterative waterfall model of development.

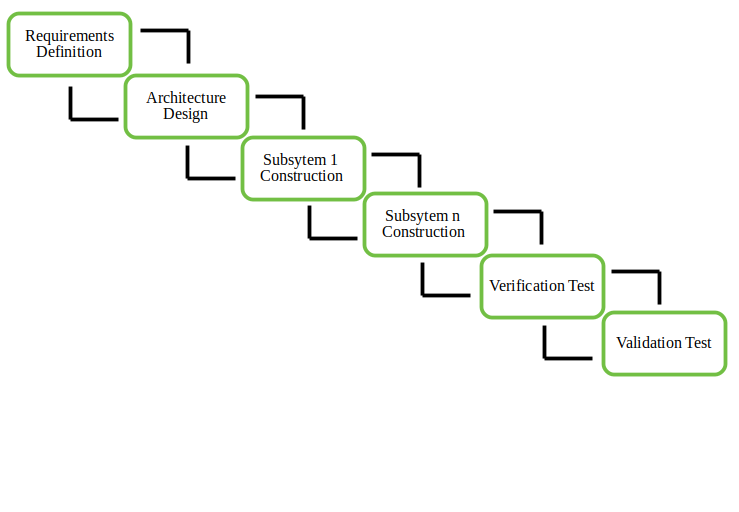
As each successive version of the software is constructed and delivered, now the feedback of the Customer is to be taken and these were then incorporated into the next version. Each version of the software has more additional features than the previous ones.



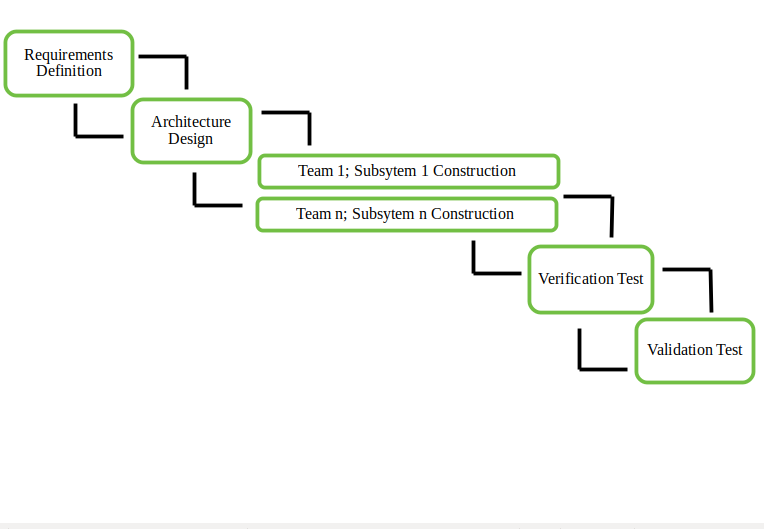
After Requirements gathering and specification, requirements are then split into several different versions starting with version-1, in each successive increment, the next version is constructed and then deployed at the customer site. After the last version (version n), it is now deployed at the client site.

**Types of Incremental Model:**

1. **Staged Delivery Model –**Construction of only one part of the project at a time.



1. **Parallel Development Model –**Different subsystems are developed at the same time. It can decrease the calendar time needed for the development, i.e., TTM (Time to Market) if enough resources are available.



**When to use this –** 

1. Funding Schedule, Risk, Program Complexity, or need for early realization of benefits.
2. When Requirements are known up-front.
3. When Projects have lengthy developments schedules.
4. Projects with new Technology.
   * Error Reduction (core modules are used by the customer from the beginning of the phase and then these are tested thoroughly)
   * Uses divide and conquer for a breakdown of tasks.
   * Lowers initial delivery cost.
   * Incremental Resource Deployment.

* Requires good planning and design.
* The total cost is not lower.
* Well defined module interfaces are required.

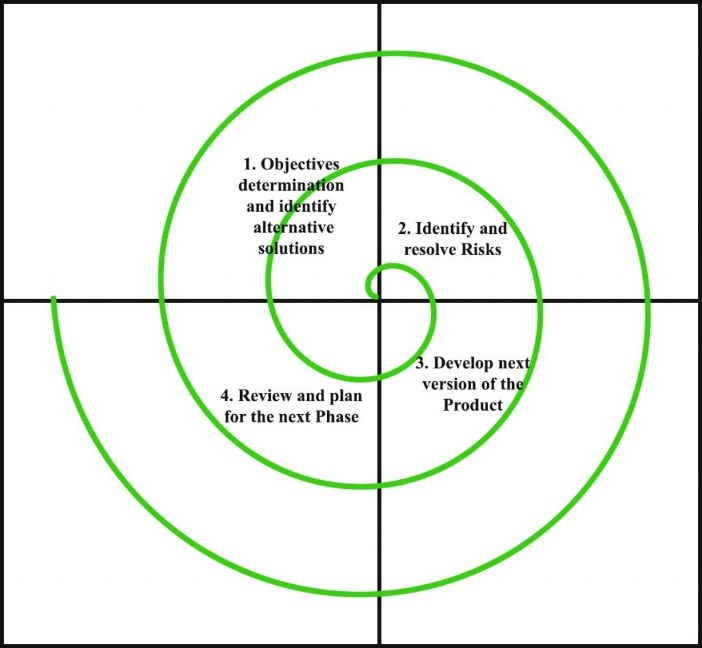
# Spiral Model:

**Spiral model** is one of the most important Software Development Life Cycle models, which provides support for **Risk Handling**. In its diagrammatic representation, it looks like a spiral with many loops. The exact number of loops of the spiral is unknown and can vary from project to project. Each loop of the spiral is called a **Phase of the software development process.**

The exact number of phases needed to develop the product can be varied by the project manager depending upon the project risks. As the project manager dynamically determines the number of phases, so the project manager has an important role to develop a product using the spiral model.

The Radius of the spiral at any point represents the expenses(cost) of the project so far, and the angular dimension represents the progress made so far in the current phase.

**The below diagram shows the different phases of the Spiral Model:**



Each phase of the Spiral Model is divided into four quadrants as shown in the above figure. The functions of these four quadrants are discussed below-

1. **Objectives determination and identify alternative solutions:** Requirements are gathered from the customers and the objectives are identified, elaborated, and analyzed at the start of every phase. Then alternative solutions possible for the phase are proposed in this quadrant.
2. **Identify and resolve Risks:** During the second quadrant, all the possible solutions are evaluated to select the best possible solution. Then the risks associated with that solution are identified and the risks are resolved using the best possible strategy. At the end of this quadrant, the Prototype is built for the best possible solution.
3. **Develop next version of the Product:** During the third quadrant, the identified features are developed and verified through testing. At the end of the third quadrant, the next version of the software is available.
4. **Review and plan for the next Phase:** In the fourth quadrant, the Customers evaluate the so far developed version of the software. In the end, planning for the next phase is started.

**Risk Handling in Spiral Model:**

A risk is any adverse situation that might affect the successful completion of a software project. The most important feature of the spiral model is handling these unknown risks after the project has started. Such risk resolutions are easier done by developing a prototype. The spiral model supports coping up with risks by providing the scope to build a prototype at every phase of the software development.

The[**Prototyping Model**](https://www.geeksforgeeks.org/software-engineering-prototyping-model/) also supports risk handling, but the risks must be identified completely before the start of the development work of the project. But in real life project risk may occur after the development work starts, in that case, we cannot use the Prototyping Model. In each phase of the Spiral Model, the features of the product dated and analyzed, and the risks at that point in time are identified and are resolved through prototyping. Thus, this model is much more flexible compared to other SDLC models.

**Why Spiral Model is called Meta Model?**

The Spiral model is called a Meta-Model because it subsumes all the other SDLC models. For example, a single loop spiral actually represents the [Iterative Waterfall Model](https://www.geeksforgeeks.org/software-engineering-iterative-waterfall-model/). The spiral model incorporates the stepwise approach of the [Classical Waterfall Model](https://www.geeksforgeeks.org/software-engineering-classical-waterfall-model/). The spiral model uses the approach of the [Prototyping Model](https://www.geeksforgeeks.org/software-engineering-prototyping-model/)by building a prototype at the start of each phase as a risk-handling technique. Also, the spiral model can be considered as supporting the [Evolutionary model](https://www.geeksforgeeks.org/software-engineering-evolutionary-model/) – the iterations along the spiral can be considered as evolutionary levels through which the complete system is built.

**Advantages of Spiral Model**:

Below are some advantages of the Spiral Model.

1. **Risk Handling:** The projects with many unknown risks that occur as the development proceeds, in that case, Spiral Model is the best development model to follow due to the risk analysis and risk handling at every phase.
2. **Good for large projects:** It is recommended to use the Spiral Model in large and complex projects.
3. **Flexibility in Requirements:** Change requests in the Requirements at later phase can be incorporated accurately by using this model.
4. **Customer Satisfaction:** Customer can see the development of the product at the early phase of the software development and thus, they habituated with the system by using it before completion of the total product.

**Disadvantages of Spiral Model**:

Below are some main disadvantages of the spiral model.

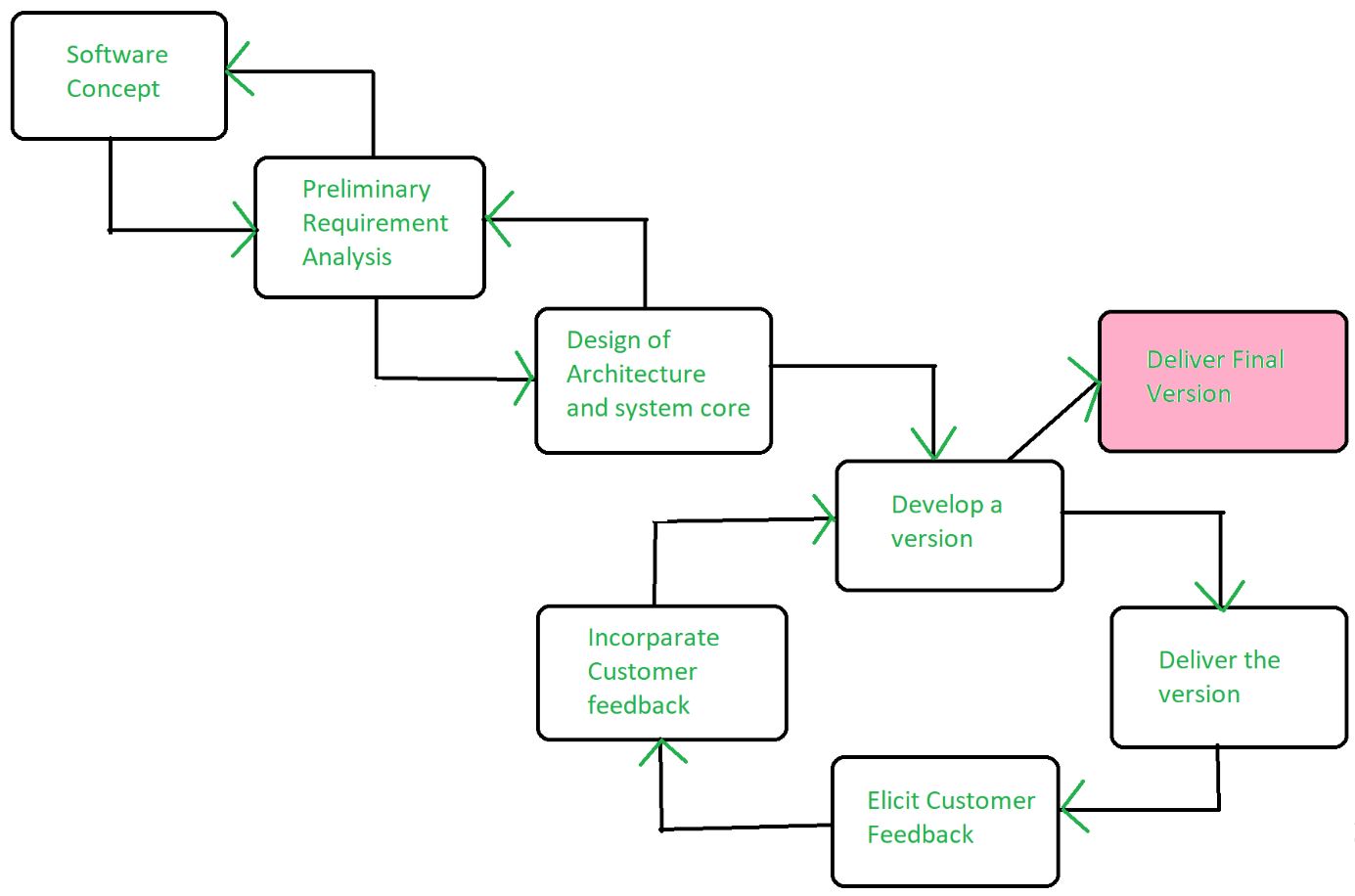
1. **Complex:** The Spiral Model is much more complex than other SDLC models.
2. **Expensive:** Spiral Model is not suitable for small projects as it is expensive.
3. **Too much dependability on Risk Analysis:** The successful completion of the project is very much dependent on Risk Analysis. Without very highly experienced experts, it is going to be a failure to develop a project using this model.
4. **Difficulty in time management:** As the number of phases is unknown at the start of the project, so time estimation is very difficult.

# Evolutionary Model:

# Evolutionary model is a combination of [Iterative](https://www.geeksforgeeks.org/software-engineering-iterative-waterfall-model/)and [Incremental model](https://www.geeksforgeeks.org/software-engineering-incremental-process-model/) of software development life cycle. Delivering your system in a big bang release, delivering it in incremental process over time is the action done in this model. Some initial requirements and architecture envisioning need to be done.

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

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

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


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

**Application of Evolutionary Model:**

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

**Advantages:**

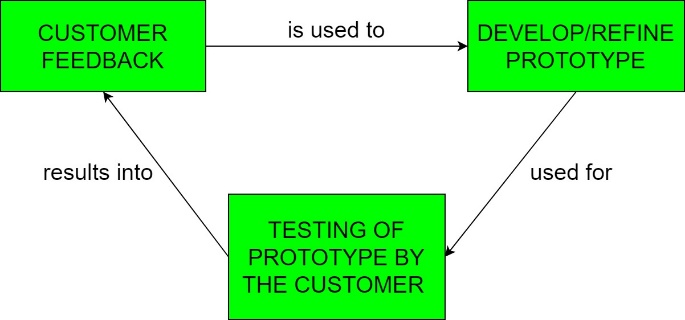
* In evolutionary model, a user gets a chance to experiment partially developed system.
* It reduces the error because the core modules get tested thoroughly.

**Disadvantages:**

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

# Prototyping Model:

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



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

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

There are four types of models available:

**A) Rapid Throwaway Prototyping:**

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

**B) Evolutionary Prototyping:**

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

**C) Incremental Prototyping:**

In this type of incremental Prototyping, the final expected product is broken into different small pieces of prototypes and being developed individually. In the end, when all individual pieces are properly developed, then the different prototypes are collectively merged into a single final product in their predefined order. It’s a very efficient approach that reduces the complexity of the development process, where the goal is divided into sub-parts and each sub-part is developed individually.

The time interval between the project’s beginning and final delivery is substantially reduced because all parts of the system are prototyped and tested simultaneously. Of course, there might be the possibility that the pieces just do not fit together due to some lack of ness in the development phase – this can only be fixed by careful and complete plotting of the entire system before prototyping starts.

**D) Extreme Prototyping:**

This method is mainly used for web development. It is consisting of three sequential independent phases:

**D.1)**In this phase a basic prototype with all the existing static pages are presented in the HTML format.

**D.2)** In the 2nd phase, Functional screens are made with a simulated data process using a prototype services layer.

**D.3)** This is the final step where all the services are implemented and associated with the final prototype.

**Advantages:**

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

**Disadvantages:**

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

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

# Object-oriented Life Cycle Model:

The Object-Oriented approach of Building Systems takes the objects as the basis. For this, first the system to be developed is observed and analyzed and the requirements are defined as in any other method of system development. Once this is often done, the objects in the required system are identified.

For example, in the case of a Banking System, a customer is an object, a chequebook is an object and even an account is an object. Object-oriented model employs an object-oriented strategy. The primary objectives are:

**1.** Object-oriented analysis,

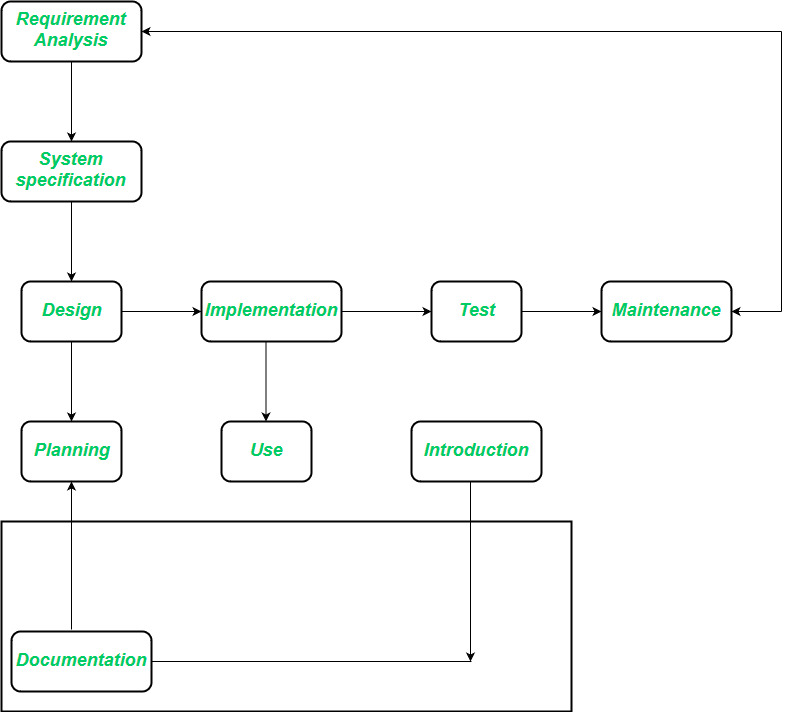
**2.** Object-oriented design,

**3.** Object-oriented programming.

Object-oriented analysis develops an object-oriented model of the application domain. Object-oriented design develops an object-oriented model of the software system. Object oriented programming realizes the software design with an object-oriented programming language that supports direct implementation of objects, classes and inheritance. There are a variety of object-oriented methodologies such as:

* **Object Identification:** System objects and their characteristics and events.
* **Object Organization:** Shows how objects are related via “part-of” relationships.
* **Object Interfaces:** Shows how objects interact with other objects.

These activities tend to be overlapping and in general and parallel.



The requirements analysis stage strives to achieve an understanding of the client’s application domain. The task that a software solution must address emerge in the course of requirement analysis. The requirement analysis phase remains completely independent of an implementation technique that might be applied later. In the system specification section, the wants definition describes what the software product must do, but not how this goal is to be achieved. One point of divergence from conventional phase model arises because implementation with object-oriented programming is marked by the assembly of already existing components. The class library serves as a tool that extends beyond the scope of an individual project because class provided by one project can increase productivity in subsequent projects.

**Advantages of Object-Oriented Life Cycle Model:**

* Design is no longer carried out independently of the later implementation because during the design phase we must consider which components are available for the solution of the problem.
* Design and implementation become more closely associated.
* Duration of the implementation phase is reduced.
* A new job title emerges, the class librarian, who is responsible for ensuring the efficient usability of the class library.

**System Engineer:**

A System Engineer is a person who deals with the overall management of engineering projects during their life cycle (focusing more on physical aspects). They follow, an interdisciplinary approach governing the total technical and managerial effort required to transform requirements into solutions. They are generally focused with all aspects of computer-based system development not only this but also hardware, software and process engineering etc. are included.

**Systems engineering** is an [interdisciplinary](https://en.wikipedia.org/wiki/Interdisciplinary) field of [engineering](https://en.wikipedia.org/wiki/Engineering) and [engineering management](https://en.wikipedia.org/wiki/Engineering_management) that focuses on how to design, integrate, and manage [complex systems](https://en.wikipedia.org/wiki/Complex_system) over their [life cycles](https://en.wikipedia.org/wiki/Enterprise_life_cycle). At its core, systems engineering utilizes [systems thinking](https://en.wikipedia.org/wiki/Systems_thinking) principles to organize this body of knowledge. The individual outcome of such efforts, an **engineered system**, can be defined as a combination of components that work in [synergy](https://en.wikipedia.org/wiki/Synergy) to collectively perform a useful [function](https://en.wikipedia.org/wiki/Function_(engineering)).

**Systems Engineering Methods:**

* Stakeholder Analysis
* Interface Specification
* Design Tradeoffs
* Configuration Management
* Systematic Verification and Validation
* Requirements Engineering

**Difference between System Engineer and Software Engineer:**

|  |  |  |
| --- | --- | --- |
| **Sl.No.** | **SYSTEM ENGINEER** | **SOFTWARE ENGINEER** |
| 01. | A System Engineer is a person who deals with the overall management of engineering projects during their life cycle (focusing more on physical aspects). | A Software Engineer is a person who deals with the designing and developing good quality of software applications/software products. |
| 02. | System Engineers follows an interdisciplinary approach governing the total technical and managerial effort required to transform requirements into solutions. | Software Engineers follows a systematic and disciplined approach for software design, development, deployment and maintenance of software applications. |
| 03. | In general, they are concerned with all aspects of computer-based system development including hardware, software and process engineering. | In general, they are concerned with all aspects of software development, infrastructure, control, applications and databases in the system. |
| 04. | One thing software engineering can learn from system engineering i.e Consideration of trade-offs and use of framework methods. | One thing system engineering can learn from software engineering i.e Disciplined approach to cost estimation. |
| 05. | System engineers mostly focus on users and domains. | Software engineers mostly focus on developing good software. |
| 06. | Systems Engineering Methods are Stakeholder Analysis, Interface Specification, Design Tradeoffs, Configuration Management, Systematic Verification and Validation, Requirements Engineering etc. | Software Engineering Methods are Modeling, Incremental Verification and Validation, Process Improvement, Model-Driven Development, Agile Methods, Continuous Integration etc. |
| 07. | It ensures correct external interfaces, interfaces among subsystems and software. | It makes interfaces among software module, data and communication path work. |
| 08. | System Engineers requires a broader education background like Engineering, Mathematics and Computer science etc. | While Software Engineers requires Computer Science or Computer Engineering background. |

**Computer-based systems:**

The computer-based system **consists of all components necessary to capture, process, transfer, store, display and manage information**. Components include software, processors, networks, buses, firmware, application-specific integrated circuits, storage devices and humans (who also process information).

Typical examples of computer-based systems are **medical systems, process control systems, communications systems, weapon systems and large information systems**.

A computer-based system makes use of a variety of system elements.

**Software**: programs, data structures and related work products.

**Hardware**: electronic devices that provide computing capabilities.

**People**: Users and operators of hardware and software.

**Database**: A large, organized collection of information that is accessed via S/w and persists over time.

**Documentation**: Manuals, on-line help files.

**Procedures**: the steps that define the specific use of each system element.

# Verification and Validation:

Verification and Validation is the process of investigating that a software system satisfies specifications and standards and it fulfills the required purpose. **Barry Boehm** described verification and validation as the following:

**Verification:** Are we building the product right?  
**Validation:** Are we building the right product?

**Verification:**

Verification is the process of checking that a software achieves its goal without any bugs. It is the process to ensure whether the product that is developed is right or not. It verifies whether the developed product fulfills the requirements that we have.  
Verification is **Static Testing**.

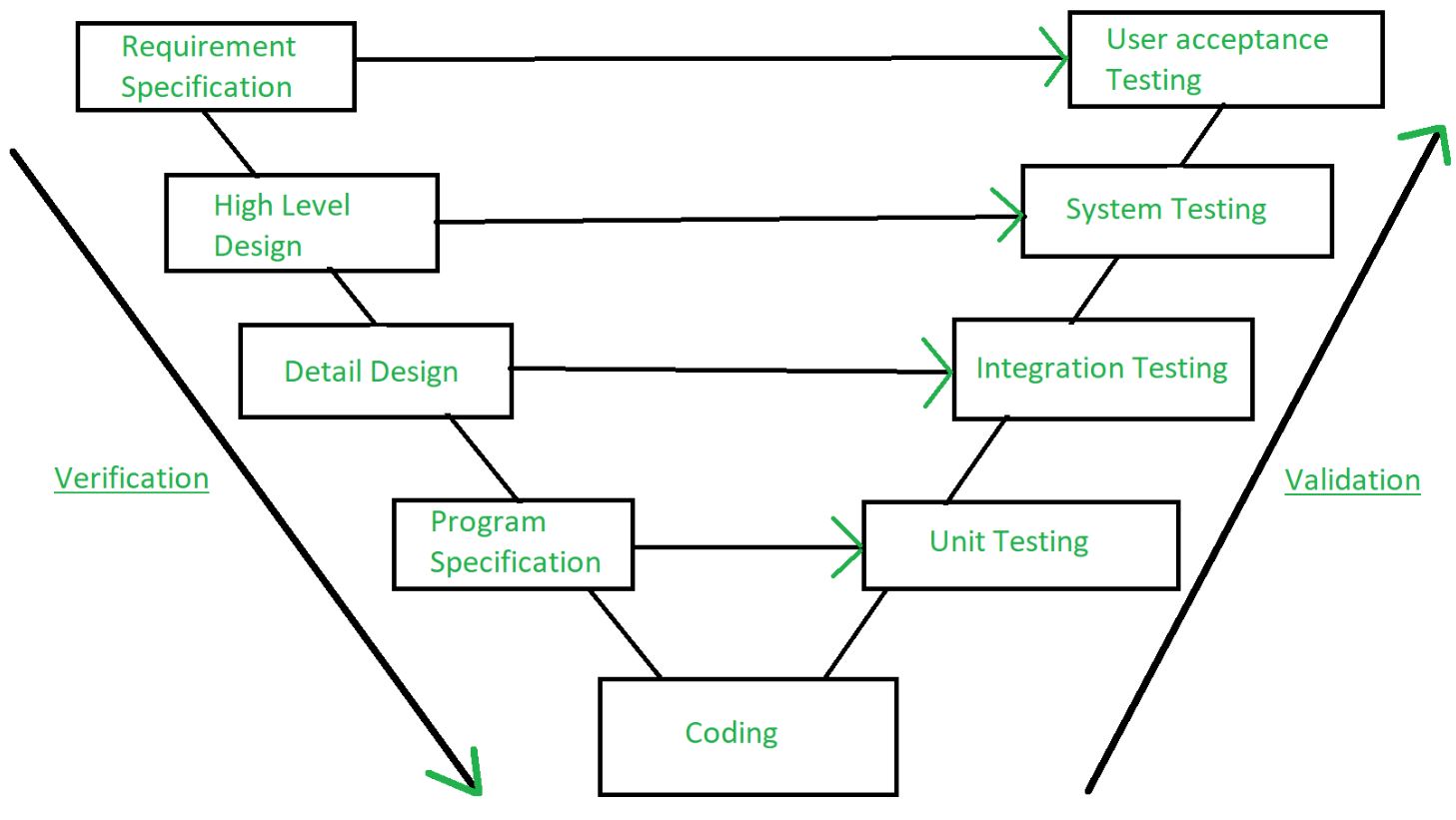
Activities involved in verification:

1. Inspections
2. Reviews
3. Walkthroughs
4. Desk-checking

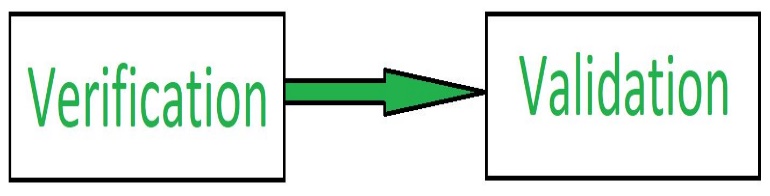
**Validation:**  
Validation is the process of checking whether the software product is up to the mark or in other words product has high level requirements. It is the process of checking the validation of product i.e. it checks what we are developing is the right product. it is validation of actual and expected product.  
Validation is the **Dynamic Testing**.

Activities involved in validation:

1. Black box testing
2. White box testing
3. Unit testing
4. Integration testing



**Note:** Verification is followed by Validation.



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

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

## **What is SDLC?**

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

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



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

### **Stage 1: Planning and Requirement Analysis**

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

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

### **Stage 2: Defining Requirements**

Once the requirement analysis is done the next step is to clearly define and document the product requirements and get them approved from the customer or the market analysts. This is done through an **SRS (Software Requirement Specification)** document which consists of all the product requirements to be designed and developed during the project life cycle.

### **Stage 3: Designing the Product Architecture**

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

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

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

### **Stage 4: Building or Developing the Product**

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

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

### **Stage 5: Testing the Product**

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

### **Stage 6: Deployment in the Market and Maintenance**

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

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

**Development Process:**

**What is software development?**

Software development is the process programmers use to build computer programs. The process, also known as the Software Development Life Cycle (SDLC), includes several phases that provide a method for building products that meet technical specifications and user requirements.

The SDLC provides an international standard that software companies can use to build and improve their computer programs. It offers a defined structure for development teams to follow in the design, creation and maintenance of high-quality software. The aim of the IT software development process is to build effective products within a defined budget and timeline.

## **Key steps in the software development process:**

There are six major steps in the software development life cycle, including:

### **1. Needs Identification:**

Needs identification is a market research and brainstorming stage of the process. Before a firm builds software, it needs to perform extensive market research to determine the product's viability. Developers must identify the functions and services the software should provide so that its target consumers get the most out of it and find it necessary and useful. There are several ways to get this information, including feedback from potential and existing customers and surveys.

The IT teams and other divisions in the company must also discuss the strengths, weaknesses and opportunities of the product. Software development processes start only if the product satisfies every parameter necessarily for its success.

### **2. Requirement Analysis:**

Requirement analysis is the second phase in the software development life cycle. Here, stakeholders agree on the technical and user requirements and specifications of the proposed product to achieve its goals. This phase provides a detailed outline of every component, the scope, the tasks of developers and testing parameters to deliver a quality product.

The requirement analysis stage involves developers, users, testers, project managers and quality assurance. This is also the stage where programmers choose the software development approach such as the waterfall or V model. The team records the outcome of this stage in a Software Requirement Specification document which teams can always consult during the project implementation.

### **3. Design:**

Design is the third stage of the software development process. Here, architects and developers draw up advanced technical specifications they need to create the software to requirements. Stakeholders will discuss factors such as risk levels, team composition, applicable technologies, time, budget, project limitations, method and architectural design.

The Design Specification Document (DSD) specifies the architectural design, components, communication, front-end representation and user flows of the product. This step provides a template for developers and testers and reduces the chances of flaws and delays in the finished product.

### **4. Development and Implementation:**

The next stage is the development and implementation of the design parameters. Developers code based on the product specifications and requirements agreed upon in the previous stages. Following company procedures and guidelines, front-end developers build interfaces and back-ends while database administrators create relevant data in the database. The programmers also test and review each other's code.

Once the coding is complete, developers deploy the product to an environment in the implementation stage. This allows them to test a pilot version of the program to make performance match the requirements.

### **5. Testing:**

The testing phase checks the software for bugs and verifies its performance before delivery to users. In this stage, expert testers verify the product's functions to make sure it performs according to the requirements analysis document.

Testers use exploratory testing if they have experience with that software or a test script to validate the performance of individual components of the software. They notify developers of defects in the code. If developers confirm the flaws are valid, they improve the program, and the testers repeat the process until the software is free of bugs and behaves according to requirements.

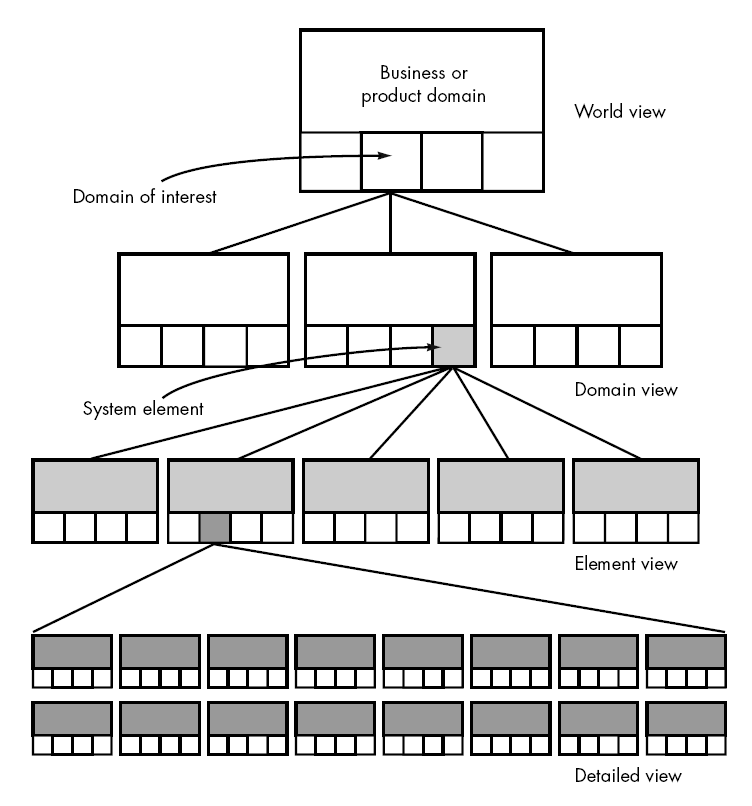
### **6. Deployment and Maintenance:**

Once the software is defect-free, the developers can deliver it to customers. After the release of a software's production version, the IT software development company creates a maintenance team to manage issues clients encounter while using the product. Maintenance can be a hot-fix if it is a minor issue but severe software failures require an update.

# THE SYSTEM ENGINEERING HIERARCHY:

# System Engineering encompasses a collection of top-down and methods to navigate the hierarchy illustrated in figure. The system engineering process usually begins with a “world view”. That is, the entire business or product domain is examined to ensure that the proper business or technology context can be established.

# The world view is refined to focus more fully on a specific domain of interest. Within a specific domain, the need for targeted system elements is analysed. Finally, the analysis, design, and construction of a targeted system elements are initiated. At the top of the hierarchy, a very broad context is established and, at the bottom, detailed activities, performed by the relevant engineering disciplines are conducted.

  
  
The world view (WV) is composed of a set of domains (D1), which can each be a system or system of systems in its own right  
  
WV = {D1 D2 D3………,Dn }  
  
Each domain is composed of specific elements (Ej) each of which serves some role in accomplishing the objective and goals of the domain and component:  
  
Di = {E1 E2 E3 …… Em}  
  
Finally, each element is implemented by specifying the technical components (Ck) that achieve the necessary function for an element:  
  
Ej = {C1, C2 C3 …….. C k}  
  
In the software context, a component could be a computer program, a reusable program component, a module, a class or object or even a programming language statement.

## **System Modeling:**

## System modeling is an important element of the system engineering process. Whether the focus is on the world view or the detailed view, the engineer creates models that: • Define the process that serves the needs of the view under consideration. • Represent the behavior of the processes and the assumption on which the behavior is based. To construct a system model, the engineer should consider a number of restraining factors: 1. Assumption that reduce the number of possible permutations and variations, thus enabling a model to reflect the problem in a reasonable manner. 2. Simplifications that the model has to create in a timely manner. 3. Limitations that help to bound the system. 4. Constraints that will guide the manner in which the model is created and the approach taken when the model is implemented. 5. Preference that indicate the preferred architecture for all data, functions, and technology.

## **System Simulations:**

## Many computer- based systems interact with the real world in a reactive fashion. That is, real- world events are monitored by the hardware and software that from the computer- based system, and based on these events; the system imposes control on the machine, process, and even people who cause the events to occur. Real- time and embedded systems often fall into the reactive systems category.

# Software Requirements:

The software requirements are description of features and functionalities of the target system. Requirements convey the expectations of users from the software product. The requirements can be obvious or hidden, known or unknown, expected or unexpected from client’s point of view.

## **Requirement Engineering:**

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

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

## **Requirement Engineering Process**

It is a four steps process, which includes –

* Feasibility Study
* Requirement Gathering
* Software Requirement Specification
* Software Requirement Validation

Let us see the process briefly -

### **Feasibility Study:**

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

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

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

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

### **Requirement Gathering:**

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

### **Software Requirement Specification:**

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

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

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

SRS should come up with following features:

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

### **Software Requirement Validation:**

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

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

# Functional vs Non-Functional Requirements:

Requirements analysis is very critical process that enables the success of a system or software project to be assessed. Requirements are generally split into two types: Functional and Non-functional requirements.

**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.

**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

Following are the differences between Functional and Non-Functional Requirements

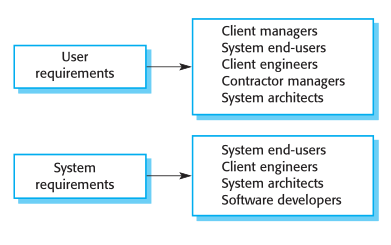
|  |  |
| --- | --- |
| **Functional Requirements** | **Non-Functional Requirements** |
| A functional requirement defines a system or its component. | A non-functional requirement defines the quality attribute of a software system. |
| It specifies “What should the software system do?” | It places constraints on “How should the software system fulfill the functional requirements?” |
| Functional requirement is specified by User. | Non-functional requirement is specified by technical peoples e.g. Architect, Technical leaders and software developers. |
| It is mandatory. | It is not mandatory. |
| It is captured in use case. | It is captured as a quality attribute. |
| Defined at a component level. | Applied to a system as a whole. |
| Helps you verify the functionality of the software. | Helps you to verify the performance of the software. |
| Functional Testing like System, Integration, End to End, API testing, etc are done. | Non-Functional Testing like Performance, Stress, Usability, Security testing, etc are done. |
| Usually easy to define. | Usually more difficult to define. |

# User and System Requirements:

The requirements for a system are the descriptions of the services that a system should provide and the constraints on its operation. These requirements reflect the needs of customers for a system that serves a certain purpose such as controlling a device, placing an order or finding information. The process of finding out, analyzing, documenting and checking these services and constraints is called requirements engineering (RE).

Some of the problems that arise during the requirements engineering process are a result of failing to make a clear separation between these different levels of description. we distinguish between them by using the term user requirements to mean the high-level abstract requirements and system requirements to mean the detailed description of what the system should do. User requirements and system requirements may be defined as follows:

1. **User requirements** are statements, in a natural language plus diagrams, of what services the system is expected to provide to system users and the constraints under which it must operate. The user requirements may vary from broad statements of the system features required to detailed, precise descriptions of the system functionality.
2. **System requirements** are more detailed descriptions of the software system’s functions, services, and operational constraints. The system requirements document (sometimes called a functional specification) should define exactly what is to be implemented. It may be part of the contract between the system buyer and the software developers.
3. You need to write requirements at different levels of detail because different types of readers use them in different ways. Figure below shows the types of readers of the user and system requirements.



1. The readers of the user requirements are not usually concerned with how the system will be implemented and may be managers who are not interested in the detailed facilities of the system. The readers of the system requirements need to know more precisely what the system will do because they are concerned with how it will support the business processes or because they are involved in the system implementation.

**Requirements Engineering Process:**

Requirement Engineering is the process of defining, documenting and maintaining the requirements. It is a process of gathering and defining service provided by the system. Requirements Engineering Process consists of the following main activities:

* Requirements elicitation
* Requirements specification
* Requirements verification and validation
* Requirements management

**Requirements Elicitation:**

It is related to the various ways used to gain knowledge about the project domain and requirements. The various sources of domain knowledge include customers, business manuals, the existing software of same type, standards and other stakeholders of the project.

The techniques used for requirements elicitation include interviews, brainstorming, task analysis, Delphi technique, prototyping, etc.

**Requirements specification:**

This activity is used to produce formal software requirement models. All the requirements including the functional as well as the non-functional requirements and the constraints are specified by these models in totality. During specification, more knowledge about the problem may be required which can again trigger the elicitation process.

The models used at this stage include ER diagrams, data flow diagrams(DFDs), function decomposition diagrams(FDDs), data dictionaries, etc.

**Requirements verification and validation:**

**Verification:** It refers to the set of tasks that ensures that the software correctly implements a specific function.  
**Validation:**

It refers to a different set of tasks that ensures that the software that has been built is traceable to customer requirements.

If requirements are not validated, errors in the requirement definitions would propagate to the successive stages resulting in a lot of modification and rework.  
The main steps for this process include:

* The requirements should be consistent with all the other requirements i.e no two requirements should conflict with each other.
* The requirements should be complete in every sense.
* The requirements should be practically achievable.

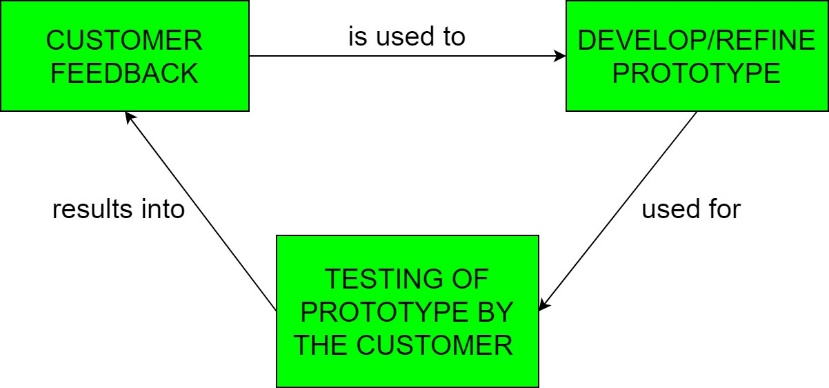
Reviews, buddy checks, making test cases, etc. are some of the methods used for this.

**Requirements management:**

Requirement management is the process of analyzing, documenting, tracking, prioritizing and agreeing on the requirement and controlling the communication to relevant stakeholders. This stage takes care of the changing nature of requirements. It should be ensured that the SRS is as modifiable as possible so as to incorporate changes in requirements specified by the end users at later stages too. Being able to modify the software as per requirements in a systematic and controlled manner is an extremely important part of the requirements engineering process.

# Software Prototyping:

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



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

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

There are four types of models available:

**A) Rapid Throwaway Prototyping:**

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

**B) Evolutionary Prototyping:**

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

**C) Incremental Prototyping:**

In this type of incremental Prototyping, the final expected product is broken into different small pieces of prototypes and being developed individually. In the end, when all individual pieces are properly developed, then the different prototypes are collectively merged into a single final product in their predefined order. It’s a very efficient approach that reduces the complexity of the development process, where the goal is divided into sub-parts and each sub-part is developed individually.

The time interval between the project’s beginning and final delivery is substantially reduced because all parts of the system are prototyped and tested simultaneously. Of course, there might be the possibility that the pieces just do not fit together due to some lack of ness in the development phase – this can only be fixed by careful and complete plotting of the entire system before prototyping starts.

**D) Extreme Prototyping:**

This method is mainly used for web development. It is consists of three sequential independent phases:

**D.1)**In this phase a basic prototype with all the existing static pages are presented in the HTML format.

**D.2)**  In the 2nd phase, Functional screens are made with a simulated data process using a prototype services layer.

**D.3)** This is the final step where all the services are implemented and associated with the final prototype.

**Advantages:**

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

**Disadvantages:**

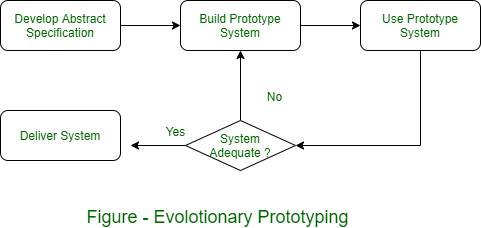
* Costly w.r.t time as well as money.
* There may be too much variation in requirements each time the prototype is evaluated by the customer.
* Poor Documentation due to continuously changing customer requirements.
* It is very difficult for developers to accommodate all the changes demanded by the customer.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

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

# Prototyping Approaches in Software Process:

The prototyping approaches in software process are as follows:

**1. Evolutionary Prototyping:** This prototype approach is based on the idea of developing an initial implementation, exposing user commentary, and going through several stages until a sufficient system has been developed as shown in figure.



The advantages to adopting this approach to software development are-

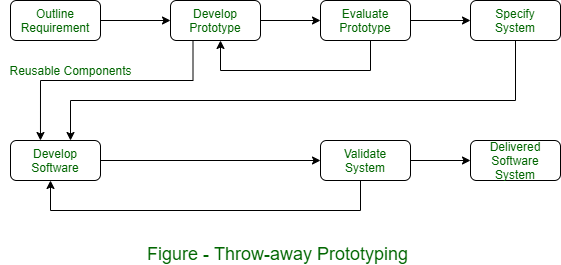
**[a]. Accelerated Delivery of the System –** Software change is required in the change of business speed. Fast delivery and usability is more important than description of long-term software maintainability functionality.

**[b]. User Engagement with the System –** The users involvement with the development process is not only to meet their needs, but also means that the system has made a commitment to it and probably wants it to work. There are some problems with evolutionary prototyping which are particularly important when large, long -lifetime systems are to be developed.

* + **Management Problems :** Software management is the key structure for large organizations to deal with software process models that generate regular delivery to assess progress.
  + **Maintenance Problems :** This type of problems means that it is difficult to understand anyone other than the original developers.
  + **Contractual Problems :** The contractual model between a customer and a software developer is based around a system specification. without system specification, it becomes difficult to design contracts for system development. Developers are unlikely to accept a fixed price contract and cannot control the changes requested by the end-users.

**2. Throw-away Prototyping:**

This type of approach extends the process of requirements analysis by reducing overall life-cycle costs. The main function of the prototype is to clarify the requirements and provide additional information for managers to assess process risks. This prototype is not used for further system development based on its evaluation. The software process model, based on the initial throw-away prototyping stage is shown in figure.



There are some problems with this approach as follows-

* Important features are left out of the prototype to simplify the rapid implementation. It is not possible to prototype some of the important parts of the system such as safety-critical functions.
* An implementation does not have any legal contract between customer and contractor.
* The non-functional requirements which concern reliability, robustness and safety cannot be adequately tested in prototype implementation.

# Rapid Prototyping Techniques:

# What is Rapid prototyping?

**Rapid prototyping (RP)**quickly creates a physical part directly from its CAD model data using various manufacturing techniques. Rapid prototyping can be used at any stage of the product development cycle for any components or sub-components. Prototyping can be repeated numerous times along the new [product design process](https://engineeringproductdesign.com/knowledge-base/product-design-process/) using the test data to achieve the desired part.

**Rapid prototyping** is a relatively new term and, in its simplest form, creating a prototype quickly to visually and functionally evaluate a part or some part features. Sometimes individual parts are rapidly created separately and assembled to test the prototype product.

## **Why is Rapid prototyping important?**

Companies need to develop and introduce new products faster to remain competitive in this fast-moving modern-day consumer market. Since faster product development and technology innovation are vital to a company’s success, rapid prototyping becomes the most crucial element of new product development. Rapid prototyping achieves the following objectives.

* Faster new product development – Rapid prototyping plays a vital role in the process of creating successful products because it speeds up the new product development process.
* Early-stage design/concept validation of the form, fit, and function of the design.
* Final stage product verification against the technical requirement and business objectives.
* It allows functionality testing to test the objectives of the concept and to finalize the product specification.
* The prototype gives the end-user, client, customer, user participants hands-on user experience to get feedback.

### **Rapid Prototyping Applications:**

* Visual prototypes
* Concept models
* Functional prototypes
* Pre-production prototypes
* Production tools prototypes
* Production moulds for prototypes

## **Types of Rapid prototypes**

 Rapid prototypes can be classified in terms of accuracy or “Fidelity”. The degree of prototype accuracy can vary from low-fidelity to high-fidelity in functionality, appearance, user interface and size.

**Low-fidelity prototype** – Very simple and produced very quickly to test the broader concept. e.g. Paper sketches to cardboard mock-ups.

**High-fidelity prototype** – These prototypes appear and function as similar and closer to the final product.

### **Advantages of Rapid Prototyping:**

* Reduced design & development time
* Reduced overall product development cost
* Elimination or reduction of risk
* Allows functionality testing at a fraction of the cost
* Improved and increased user involvement during design stages of NPD
* Ability to evaluate human factors and ergonomics

### **Disadvantages of Rapid Prototyping:**

* Lack of accuracy – If the function of the product relies heavily on the accuracy of the parts, then rapid prototype parts and assembly might not be able to f the same accuracy.
* Added initial costs – Rapid prototyping costs money due to technologies used and faster turnaround required.
* Some rapid prototyping processes are still expensive and not economical.
* Reduced material properties like surface finish and strength.
* Rapid prototyping required skilled labour.
* Limited material range.
* Overlooking some key features because they cannot be prototyped affects the prototype testing.
* End-user confusion, customers mistaking it for the finished project/developer misunderstanding of user objectives.

**Rapid Prototyping Techniques:**

Choosing a suitable **rapid prototyping technology**is critical to the prototype’s success. Each rapid prototyping technique has its compromise in terms of cost, speed, material compatibility of the feature, fidelity level and development stage.

# Rapid prototyping techniques

Rapid prototyping doesn’t need to be limited to one process; one can use more than one manufacturing technique to assemble a prototype.

Following are the types of rapid prototyping technology available for engineering product designers:

1. Additive manufacturing – Stereolithography (SLA), Selective laser sintering (SLS), [Direct metal laser sintering(DMLS)](https://engineeringproductdesign.com/knowledge-base/direct-metal-laser-sintering/), [Fused Deposition Modelling (FDM)](https://engineeringproductdesign.com/knowledge-base/fused-deposition-modeling/), MJF, [Binder jetting](https://engineeringproductdesign.com/knowledge-base/binder-jetting/) and Poly jetting
2. Other techniques – CNC Machining Prototyping, [Vacuum casting](https://engineeringproductdesign.com/knowledge-base/vacuum-casting/), [Investment casting](https://engineeringproductdesign.com/knowledge-base/investment-casting/)

### **Stereolithography (SLA):**

Stereolithography (SLA) uses a laser to cure UV-curable resin to build parts from a pool of liquid resins. They are best for lower end functional prototypes, patterns, mould and production tools. SLA provides product designers and engineers with the ability to rapid prototype parts with excellent surface finish and good dimensional accuracy.

### **Selective Laser Sintering (SLS):**

SLS uses a laser to produce functional parts from polymer powder by sintering. Due to SLS parts’ internal porosity, they are brittle but have excellent tensile strength. Generally, SLS has a larger build volume and can generate parts with highly complex geometry and produce long-lasting prototypes.

### **Direct Metal Laser Sintering (DMLS):**

DMLS is an additive manufacturing technology that involves melting and fusing layers of metallic powder using a computer-controlled, high-power laser beam.

### **Fused Deposition Modelling (FDM):**

FDM technology creates 3D parts by melting and extruding thermoplastic resins onto a Build platform layer by layer and letting it re-solidify.

### **Multi Jet Fusion (MJF):**

Multi-jet fusion is a powder bed fusion 3D printing technology in which fusing agent bonds powdered material and then heats to fuse them to produce 3D parts. MJF can print highly accurate and durable rigid parts using Nylon PA11, PA12 Nylon and PP. They can also print flexible TPU parts.

### **Binder Jetting:**

Binder jetting is one of 7 additive manufacturing technologies that can produce metal and colour plastic parts. Compared to DMLS, they are very cheap to make metal prototypes. In binder jetting, the binding agent is selectively deposited into a powder bed where it solidifies to create a 3d part.

### **Poly Jetting:**

Polyjet is a material jetting technology that produces smooth and accurate parts. The technology uses photopolymer to create parts by jetting onto the build platform and curing them using a UV light.

The surface texture, colour and different properties are crucial for concept models and full assembly prototypes to test user handling.

### **CNC Machining:**

For parts where its functionality relies on high accuracy and tolerance, CNC machining is still the best prototyping method available. CNC machining is a subtractive manufacturing process in which blocks and rods of metal and engineering plastics are milled and turned to create highly accurate complex parts. There is a wide variety of materials available to cater for any application.

### **Vacuum Casting – Urethane Casting:**

Vacuum casting or Urethane casting uses silicone moulds to make plastic and rubber components under vacuum. They are one of the best processes to rapid prototype injection moulding plastic parts. Therefore, they are ideally suited for small to medium pre-production parts that can be used for functional testing.

### **Investment Casting:**

Investment casting creates parts using wax patterns coated with refractory material to make an expendable mould. Then the wax pattern is melted away before pouring molten metal. Once the metal solidifies within the mould, the foundry breaks the refractory mould to remove the cast parts.

# User Interface Prototyping:

User interface (UI) prototyping is an iterative analysis technique in which users are actively involved in the mocking-up of the UI for a system. UI prototypes have several purposes:

* As an analysis artifact that enables you to explore the problem space with your stakeholders.
* As a requirements artifact to [initially envision](http://agilemodeling.com/essays/initialRequirementsModeling.htm) the system.
* As a design artifact that enables you to explore the solution space of your system.
* A vehicle for you to communicate the possible UI design(s) of your system.
* A potential foundation from which to continue developing the system

# 

The first step is to analyze the user interface needs of your users. User interface modeling moves from requirements definition into analysis at the point you decide to evolve all or part of your [essential user interface prototype](http://agilemodeling.com/artifacts/essentialUI.htm) into a traditional UI prototype. This implies you convert your hand-drawings, flip-chart paper and sticky notes into something a little more substantial. You begin this process by making platform decisions which in effect is an architectural decision.

For example, do you intend to deploy your system so it runs in an Internet browser, as an application with a windows-based graphical user interface (GUI), as a cross-platform Java application, or as a mainframe-based set of "green screens?" Different platforms lead to different prototyping tools, for a browser-based application, you need to use an HTML-development tool, whereas a Java-based application would require a Java development tool and a different approach to the user interface design.

**Software Documentation:**

**Software documentation** is a written piece of text that is often accompanied with a software program. This makes the life of all the members associated with the project easier. It may contain anything from API documentation, build notes or just help content. It is a very critical process in software development. It’s primarily an integral part of any computer code development method.

**Types Of Software Documentation:**

1. **Requirement Documentation:**  
   It is the description of how the software shall perform and which environment setup would be appropriate to have the best out of it. These are generated while the software is under development and is supplied to the tester groups too.
2. **Architectural Documentation:**  
   Architecture documentation is a special type of documentation that concerns the design. It contains very little code and is more focused on the components of the system, their roles and working. It also shows the data flows throughout the system.
3. **Technical Documentation:**  
   These contain the technical aspects of the software like API, algorithms etc. It is prepared mostly for the software devs.
4. **End-user Documentation:**  
   As the name suggests these are made for the end user. It contains support resources for the end user.

**Importance of software documentation:**

For a programmer reliable documentation is always a must the presence keeps track of all aspects of an application and helps in keeping the software updated.

**Advantage of Software Documentation:**

* The presence of documentation helps in keeping the track of all aspects of an application and also improves on the quality of the software product.
* The main focuses are based on the development, maintenance and knowledge transfer to other developers.
* Helps development teams during development.
* Helps end-users in using the product.
* Improves overall quality of software product
* It cuts down duplicative work.
* Makes easier to understand code.
* Helps in establishing internal co-ordination in work.

**Disadvantage of Software Documentation:**

* The documenting code is time consuming.
* The software development process often takes place under time pressure, due to which many times the documentation updates don’t match the updated code.
* The documentation has no influence on the performance of an application.
* Documenting is not so fun, its sometime boring to a certain extent.