Unit -1

INTRODUCTION

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

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

This tutorial should provide you basic understanding of software product, software design and development process, software project management and design complexities etc. At the end of the tutorial you should be equipped with well understanding of software engineering concepts.

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:
(1) The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is, the application of engineering to software.
(2) 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.

Software Evolution

The process of developing a software product using software engineering principles and methods is referred to as **software evolution.** This includes the initial development of software and its maintenance and updates, till desired software product is developed, which satisfies the expected requirements.
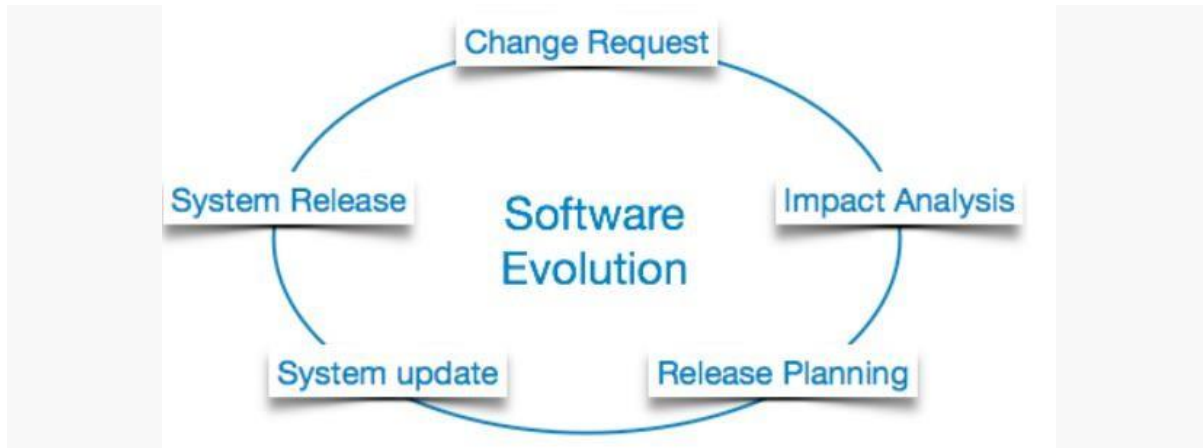
Evolution starts from the requirement gathering process. After which developers create a prototype of the intended software and show it to the users to get their feedback at the early stage of software product development. The users suggest changes, on which several consecutive updates and maintenance keep on changing too. This process changes to the original software, till the desired software is accomplished.

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

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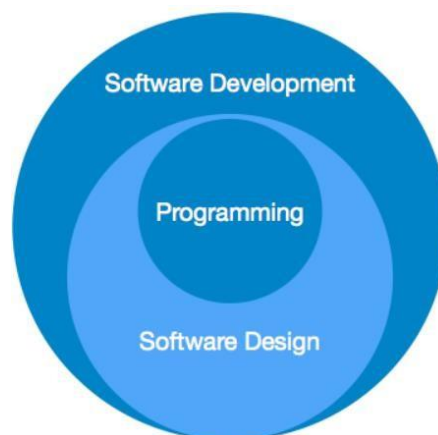
Programming paradigm is a subset of Software design paradigm which is further a subset of Software development paradigm.

Software Development Paradigm

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

- Requirement gathering
- Software design
- Programming

Software Design Paradigm

This paradigm is a part of Software Development and includes –

- Design
- Maintenance
- Programming

Programming Paradigm

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

- Coding
- Testing
- Integration

Need of Software Engineering

The need of software engineering arises because of higher rate of change in user requirements and environment on which the software is working.

- **Large software -** It is easier to build a wall than to a house or building, likewise, as the size of software become large engineering has to step to give it a scientific process.
- **Scalability-** If the software process were not based on scientific and engineering concepts, it would be easier to re-create new software than to scale an existing one.
- **Cost-** As hardware industry has shown its skills and huge manufacturing has lower down the price of computer and electronic hardware. But the cost of software remains high if proper process is not adapted.
- **Dynamic Nature-** The always growing and adapting nature of software hugely depends upon the environment in which user works. If the nature of software is always changing, new enhancements need to be done in the existing one. This is where software engineering plays a good role.
- **Quality Management-** Better process of software development provides better and quality software product.

Characteristics of good software

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

- Operational
- Transitional
- Maintenance

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

Operational

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

- Budget
- Usability
- Efficiency
- Correctness
- Functionality
- Dependability
- Security
- Safety

Transitional

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

- Portability
- Interoperability
- Reusability
- Adaptability

Maintenance

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

- Modularity
- Maintainability
- Flexibility
- Scalability

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

Software Development Life Cycle, SDLC for short, is a well-defined, structured sequence of stages in software engineering to develop the intended software product. SDLC is a process that consists of a series of planned activities to develop or alter the Software Products.

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 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.

- 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.

SDLC Models

There are various software development life cycle models defined and designed which are followed during the software development process. These models are also referred as Software Development Process Models. Each process model follows a Series of steps unique to its type to ensure success in the process of software development.

Following are the most important and popular SDLC models followed in the industry −

- Waterfall Model
- Iterative Model
- Spiral Model
- V-Model
- Big Bang Model

Other related methodologies are Agile Model, RAD Model, Rapid Application Development and Prototyping Models.

The Waterfall Model was the 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.

The 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. This means that any phase in the development process begins only if the previous phase is complete. In this waterfall model, the 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 this Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.

The following illustration is a representation of the different phases of the Waterfall Model.
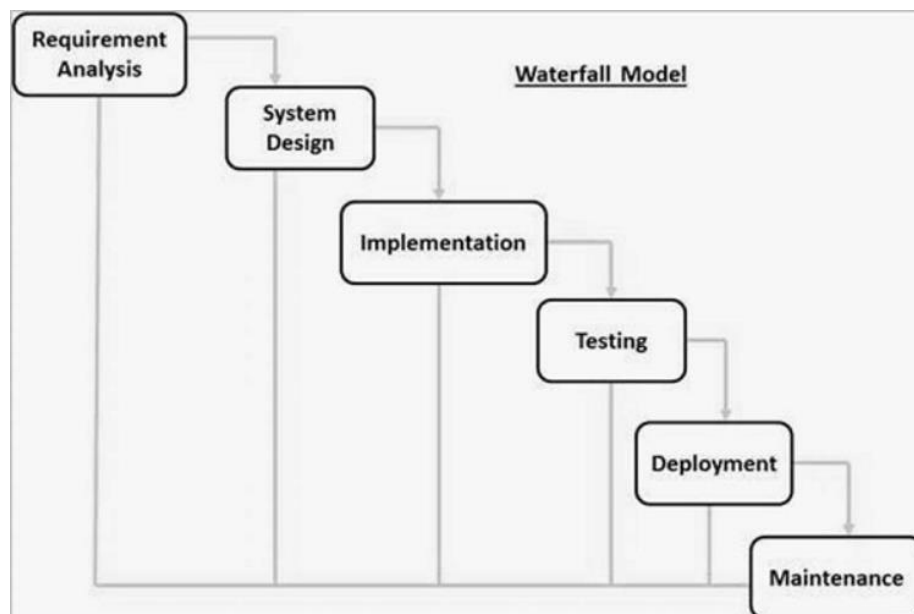
The sequential phases in Waterfall model are –

- **Requirement Gathering and analysis** – All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.

- **System Design** – The requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.

- **Implementation** – With inputs from the system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.

- **Integration and Testing** − All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entiresystem is tested for any faults and failures.

- **Deployment of system** − Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.

- **Maintenance** − There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

All these phases are cascaded to each other in which progress is seen as flowing steadily downwards (like a waterfall) through the phases. The next phase is started only after the defined set of goals are achieved for previous phase and it is signed off, so the name "Waterfall Model". In this model, phases do not overlap.

Waterfall Model - Application

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

- Requirements are very well documented, clear and fixed.

- Product definition is stable.

- Technology is understood and is not dynamic.

- There are no ambiguous requirements.

- Ample resources with required expertise are available to support the product.

- The project is short.

Waterfall Model - Advantages

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

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

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

- Simple and easy to understand and use

- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.

- Phases are processed and completed one at a time.

- Works well for smaller projects where requirements are very well understood.

- Clearly defined stages.

- Well understood milestones.

- Easy to arrange tasks.

- Process and results are well documented.

Waterfall Model - Disadvantages

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

The major disadvantages of the Waterfall Model are as follows −

- No working software is produced until late during the life cycle.

- High amounts of risk and uncertainty.

- Not a good model for complex and object-oriented projects.

- Poor model for long and ongoing projects.

- Not suitable for the projects where requirements are at a moderate to high risk of changing. So, risk and uncertainty is high with this process model.

- It is difficult to measure progress within stages.

- Cannot accommodate changing requirements.

- Adjusting scope during the life cycle can end a project.

- Integration is done as a "big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early.

The spiral model combines the idea of iterative development with the systematic, controlled aspects of the waterfall model. This Spiral model is a combination of iterative development process model and sequential linear development model i.e. the waterfall model with a very high emphasis on risk analysis. It allows 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 phase also includes understanding the system requirements by continuous communication between the customer and the system analyst. At the end of the spiral, the product is deployed in the identified market.

Design

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

Construct or Build

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

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

Evaluation and Risk Analysis

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

The following illustration is a representation of the Spiral Model, listing the activities in each phase.
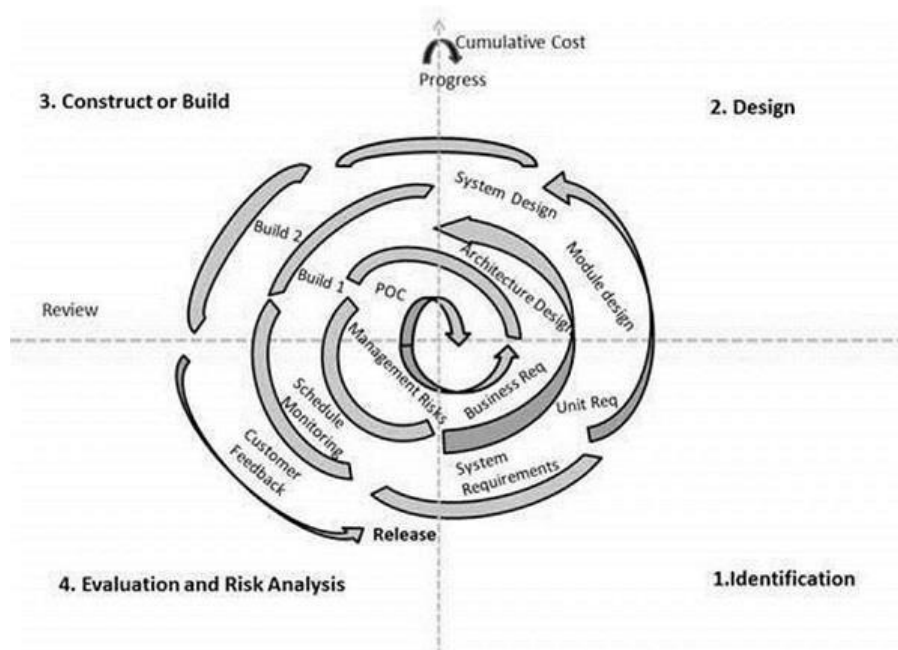
Based on the customer evaluation, the software development process enters the next iteration and subsequently follows the linear approach to implement the feedback suggested by the customer. The process of iterations along the spiral continues throughout the life of the software.

Spiral Model Application

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

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

- When there is a budget constraint and risk evaluation is important.

- For medium to high-risk projects.

- Long-term project commitment because of potential changes to economic priorities as the requirements change with time.

- Customer is not sure of their requirements which is usually the case.

- Requirements are complex and need evaluation to get clarity.

- New product line which should be released in phases to get enough customer feedback.

- Significant changes are expected in the product during the development cycle.

Spiral Model - Pros and Cons

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

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

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

The advantages of the Spiral SDLC Model are as follows –

- Changing requirements can be accommodated.

- Allows extensive use of prototypes.

- Requirements can be captured more accurately.

- Users see the system early.

- Development can be divided into smaller parts and the risky parts can be developed earlier which helps in better risk management.

The disadvantages of the Spiral SDLC Model are as follows –

- Management is more complex.

- End of the project may not be known early.

- Not suitable for small or low risk projects and could be expensive for small projects.

- Process is complex

- Spiral may go on indefinitely.

- Large number of intermediate stages requires excessive documentation.

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

The V-Model is an extension of the waterfall model and is based on the 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 the next phase starts only after completion of the previous phase.

V-Model - Design

Under the 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. The Coding Phase joins the two sides of the V-Model.

The following illustration depicts the different phases in a V-Model of the SDLC.
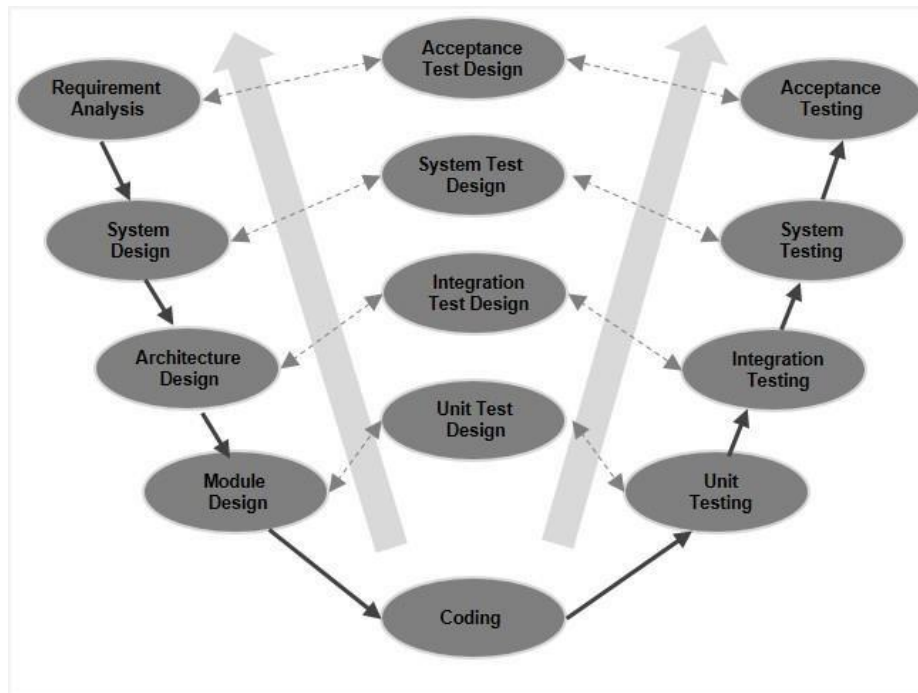
V-Model - Verification Phases

There are several Verification phases in the V-Model, each of these are explained in detail below.

Business Requirement Analysis

This is the first phase in the development cycle where the product requirements are understood from the customer's perspective. This phase involves detailed communication with the customer to understand his expectations and exact requirement. This is a very important activity and needs 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 is time to design the complete system. The system design will have the understanding and detailing the complete hardware and communication setup for the product under development. The system test plan is developed based on the system design. Doing this at an earlier stage leaves more time for the 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. The 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. The unit tests are an essential part of any development process and helps eliminate the maximum faults and errors at a very early stage. These 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

The different Validation Phases in a V-Model are explained in detail below.

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 this 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 the same as the 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 a disciplined domain.

The following pointers are some of the most suitable scenarios to use the V-Model application.

- 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 - Pros and Cons

The advantage of the V-Model method is that it is 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 advantages of the V-Model method are as follows –

- 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.

The disadvantages of the V-Model method are as follows –

- 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.

The Software Prototyping refers to building software application prototypes which displays 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 a stepwise approach explained 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. While, 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 the actual implementation. The changes accepted are again incorporated in the new Prototype developed and the cycle repeats until the customer expectations are met.

Prototypes can have horizontal or vertical dimensions. A 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. By using evolutionary prototyping, the 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 follows.

The advantages of the Prototyping Model are as follows −

- Increased user involvement in the product even before its implementation.
- 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.

The Disadvantages of the Prototyping Model are as follows −

- Risk of insufficient requirement analysis owing to too much dependency on the prototype.
- 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 it is not technically feasible.

- The effort invested in building prototypes may be too much if it is not monitored properly.

Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not.

Testing is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements.

This tutorial will give you a basic understanding on software testing, its types, methods, levels, and other related terminologies.

Why to Learn Software Testing?

In the IT industry, large companies have a team with responsibilities to evaluate the developed software in context of the given requirements. Moreover, developers also conduct testing which is called **Unit Testing**. In most cases, the following professionals are involved in testing a system within their respective capacities –

- Software Tester
- Software Developer
- Project Lead/Manager
- End User

Different companies have different designations for people who test the software on the basis of their experience and knowledge such as Software Tester, Software Quality Assurance Engineer, QA Analyst, etc.

Applications of Software Testing

- **Cost Effective Development** - Early testing saves both time and cost in many aspects, however reducing the cost without testing may result in improper design of a software application rendering the product useless.

- **Product Improvement** - During the SDLC phases, testing is never a time-consuming process. However diagnosing and fixing the errors identified during proper testing is a time-consuming but productive activity.

- **Test Automation** - Test Automation reduces the testing time, but it is not possible to start test automation at any time during software development. Test automaton should be started when the software has been manually tested and is stable to some extent. Moreover, test automation can never be used if requirements keep changing.

- **Quality Check** - Software testing helps in determining following set of properties of any software such as

    - Functionality
    - Reliability
    - Usability
    - Efficiency
    - Maintainability
    - Portability

What is Testing?

Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not. In simple words, testing is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements.

According to ANSI/IEEE 1059 standard, Testing can be defined as - A process of analyzing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item.

Who does Testing?

It depends on the process and the associated stakeholders of the project(s). In the IT industry, large companies have a team with responsibilities to evaluate the developed software in context of the given requirements. Moreover, developers also conduct testing which is called **Unit Testing**. In most cases, the following professionals are involved in testing a system within their respective capacities –

- Software Tester
- Software Developer
- Project Lead/Manager
- End User

Different companies have different designations for people who test the software on the basis of their experience and knowledge such as Software Tester, Software Quality Assurance Engineer, QA Analyst, etc.

It is not possible to test the software at any time during its cycle. The next two sections state when testing should be started and when to end it during the SDLC.

When to Start Testing?

An early start to testing reduces the cost and time to rework and produce error-free software that is delivered to the client. However in Software Development Life Cycle (SDLC), testing can be started from the Requirements Gathering phase and continued till the deployment of the software.

It also depends on the development model that is being used. For example, in the Waterfall model, formal testing is conducted in the testing phase; but in the incremental model, testing is performed at the end of every increment/iteration and the whole application is tested at the end.

Testing is done in different forms at every phase of SDLC –

- During the requirement gathering phase, the analysis and verification of requirements are also considered as testing.
- Reviewing the design in the design phase with the intent to improve the design is also considered as testing.
- Testing performed by a developer on completion of the code is also categorized as testing.

When to Stop Testing?

It is difficult to determine when to stop testing, as testing is a never-ending process and no one can claim that software is 100% tested. The following aspects are to be considered for stopping the testing process –

- Testing Deadlines
- Completion of test case execution
- Completion of functional and code coverage to a certain point
- Bug rate falls below a certain level and no high-priority bugs are identified
- Management decision