### ACKNOWLEDGEMENT

Primarily I would thank God for being able to complete this project with success. Then I would like to thank my faculty Mrs. Princy Jain & Mrs. Neeru Jain whose valuable guidance has been the once that helped me patch this project and makes it full proof success. Her suggestion and her instructions has served as the major contribution towards the completion of the project.

Then I would like to thank my parents and friends who helped me with their valuable suggestions and guidance has been helpful in various phases of the completion of the project.

Last but not least I would like to thank my classmates who have helped me a lot.

Name of Student:- DEEPIKA BHARDWAJ (190962106008)

### CERTIFICATE

This is certify that Ms. DEEPIKA BHARDWAJ has completed the project titled

“PHARMACY SALES & INVENTORY SYSTEM” satisfactorily and submitted the project report as per the guidelines of the CSS University, Meerut.

#### PROJECT GUIDE PRINCIPAL

SIGNATURE:- SIGNATURE:-

NAME:- NAME:-

DATE:- DATE:-

#### INTERNAL EXAMINER EXTERNAL EXAMINER

SIGNATURE:- SIGNATURE:-

NAME:- NAME:-

DATE:- DATE:-

* **INTRODUCTION**

## TABLE OF CONTENTS

* + Basic introduction of project
  + Aims and objectives
  + Benefits of pharmacy sales and inventory system
  + Technical details

#### SYSTEM ANALYSIS

* + Preliminary analysis & information gathering
  + Input/output
  + Feasibility study
  + System requirement specification
  + Software engineering model used
  + Cost estimation
  + Project scheduling

#### SYSTEM DESIGN

* + Project planning
  + Modules
  + Flow chart
  + Data flow diagram(DFD)
  + E-R diagram
  + Screenshots

#### TESTING

* **IMPLEMENTATION & MAINTAINANCE**

#### CONCLUSION

* **REFRENCES**

# INTRODUCTION OF PHARMACY SALES & INVENTORY SYSTEM

## BASIC INTRODUCTION OF PROJECT

The project “Pharmacy sales & Inventory System” is complete web based application. It is designed and developed especially for pharmacies. An Pharmacy Sales & Inventory System is the combination of technology (hardware and software) and processes and procedures that oversee the monitoring and maintenance of stocked products.

Pharmacy sales and Inventory System plays an important role because it reduces the stress, monitoring of products, making balance sheets and many more which was done manually.

The pharmacy sales and inventory system is a simple project that will help pharmacy business manage the medicine that they are selling and monitor their available stocks. This will also help to track the expired medicine product on hand which this system notifies them the when the expiry date of their stock will come. And by this, they will check the remaining stock of the medicine product that is expired and encode to the system to deduct the count of stock that has already for inventory monitoring and sales product availability.

A Pharmacy information system must retrieve process and update the information it obtain for safe and effective use of drugs. The pharmacy information system is normally used to support activities and the inventory.

### BENEFITS OF PHARMACY SALES & INVENTORY SYSTEM

* This pharmacy sales & inventory system is fully functional and flexible.
* It is very easy to use.
* It saves a lot of time, money and labor.
* This application can be used by any other store to automate the process of manually maintaining the records related to the subject of maintaining the stock and liquid flows.
* It tracks all the information regarding the inventory, pharmacy, sells.
* The software can print invoices, bills, receipts etc. it can also maintain the record of supplies sent in by the supplier.
* Search based on keyboard such as location, prices.
* Provides user friendly interface, satisfying the needs of the consumers.

## AIMS AND OBJECTIVES

* The main objective of the application is to automate the existing system of manually maintained records of the counter sales, purchases, reorder levels, supplier and customer monetary position and related transactions made by the seller.
* This System is to reduce costs and improve operational efficiency.
* To do this, pharmacies need complete visibility into their existing inventory levels in order to take some of the guesswork out of maintaining ideal supply levels. This involves being able to track inventory stores, orders being generated, inventory receiving – and all in real time.
* It is user friendly application for pharmacist which reduces the burdens and helps to manage all sections of pharmacy like medicine management, handling and billing etc.
* It includes safe data stores about medicine as well as fast searching, delete, and update of medicines.
* This application is reduced as much as possible to avoid errors while entering data.
* The main objective of this system is to keep records of the complete inventory and manage the details of medicines, customers, suppliers, sales.
* **Login Page**

## FEATURES

This page of the system where system users will submit their system credentials to access the data of the pharmacy sales and inventory system.

#### Home Page

This page where the system users will be redirected by default after logging into the system.

#### Medicine List Page

The page where all the medicine available at the pharmacy is listed and can be managed only by the system admin user.

#### Medicine Category Page

The page where all medicine categories are listed and can be managed.

#### Medicine Type Page

The page where all the medicine product types are listed.

#### Supplier List Page

The page where can the admin manages the list of the pharmacy’s suppliers.

#### Receiving Page

The page where will the admin encode the received medicinal product that has been bought to the supplier by the pharmacy. This the feature that updates the product stock availability in the pharmacy.

#### Customer Page

The page where all the pharmacy’s customers are listed.

#### Sales Page

The page where the sale transaction is being made. The sales list is listed on this page.

#### Inventory Page

The page where can the pharmacy management monitor the stock of their pharmacy.

#### Expired List

The page where the confirmed stock the has expired will be listed in order to deduct the available stock of a certain product in the system.

#### User list

The page where can system admin manage the list of the pharmacy sales and inventory system user list.

## TECHNICAL DETAILS

**HARDWARE & SOFTWARE REQUIREMENTS**

Project Name :- Pharmacy Sales and Inventory system RAM :- 1GB and above

Language used :- PHP

Database :- MySQL User interface design :- HTML, JS

Web browser :- Mozilla, Google Chrome, Opera Software :- XAMPP

Text editor :- Visual studio Operating System :- Window 10

# SYSTEM ANALYSIS

### PRELIMINARY ANALYSIS

Preliminary analysis is defined as the initial process at the start of a project the start of project that determines whether the concept is viable.

The main objective of preliminary analysis is to identify the customer’s needs, evaluate system concept for feasibility, perform economic and technical analysis, perform cost benefit analysis and create system definition that forms the foundation for all subsequent engineering works. There should be enough expertise available for hardware and software for doing analysis.

It looks at economic, market industry and social trends that influence the success of business endeavors associated with a proposed strategy. Preliminary analysis is repeated in situations where primary investigation trigger updates to plans. It creates a comprehensive idea is the enterprises objective and states how the outcome is meant to be expressed.

##### OBJECTIVES:

* + Clarify and understand the project request.
  + Determine the size of the project.
  + Assess costs and benefits of alternative approaches.
  + Determine the technical and operational feasibility of alternative approaches.

### INFORMATION GATHERING

The activity gathering information about the functioning of the present system. Large quantities of information need to be collected, evaluated, managed and communicated. The four most commonly used methods of gathering information are:-

#### Interviews

Interviews must be well-planned in advance. Each participant should know the objectives of the interview beforehand prepare for it.

#### Questionnaire

A questionnaire is shorter and more highly structured than interviews. It is a useful technique for obtaining the same information from a large group of users.

#### Observation

Observation require the systems analyst to go to on the work site to watch what is being done there. It is a good way of confirming or correcting information gathered by other techniques.

#### Study of existing documents

Since most organizations today are involved in a lot of paperwork, the analyst can learn a lot about the system by studying documents.

### INPUT & OUTPUT

In input, the user has to enter his or her details according to the system need to access the services that he or she needs.

In output, the user can access the information from the system after entering the input in the system.

### FEASIBILITY STUDY

The concept of feasibility is to determine whether or not a project is worth doing. The process followed in making this determination is called feasibility study. Once it has been determined that a project is feasible, the system analyst can go ahead and prepare the project specification which finalizes project requirements.

#### Types of feasibility:

* Technical feasibility
* Operational feasibility
* Economic feasibility
* Social feasibility

Here we describe only few of these in details:-

#### TECHNICAL FEASIBILITY

This is concerned with specifying equipment and software that will successfully satisfy the user requirement. Technical needs of the system include:-

* Facility to produce outputs in a given time.
* Response time under certain condition.
* Facility to communicate data to distant location.

#### OPERATIONAL FEASIBILITY

This is mainly related to human organization and political aspects. The points to be considered are:-

* + What changes will be brought with the system?
  + What organizational structures are distributed?
  + What new skills will be required? Do the existing staff members have these skills? If not, can they be trained in due course of time.

The existing staff of the college can be trained to interact with the system which is a GUI based software and is easy to use. Hence the project is operationally feasible proposed system. More commonly known as cost-benefit analysis; the procedure is to determine the benefits and savings that are expected from a proposed system and compare them with costs. If benefits outweigh costs, a decision is taken to design and implement the system.

#### ECONOMIC FEASIBILITY

Economic analysis is the most frequently used technique for evaluating the effectiveness of a is concerned with specifying equipment and software that will successfully satisfy the user requirement. Economical needs of the system include:-

* Facility to produce outputs in a given time.
* Response time under certain condition.
* Facility to communicate data to distant location.

#### SOCIAL FEASIBILITY

It is the determination of whether a proposed project will be acceptable to the people or not. This determination typically examines the probability of the project being accepted by the group directly affected the proposed system change. To solve the actual problems in an industry setting, a software or a team of engineers must incorporate development strategy that encompasses the process, methods and tools layers. This strategy is often referred to **Process Model** or a **Software Engineering Paradigm.**

### SOFTWARE REQUIREMENT SPECIFICATION (SRS)

A software requirements specification (SRS) is a document that describes the nature of a project, software application. In simple words, SRS document is a manual of a project provided it is prepared before you kick-smart a project/application.

This document also known by the name SRS report, software document. A software document is primarily prepared for a project, software or any kind of application.

There are a set of guidelines to be followed while preparing the software requirement specification document. This includes the purpose, scope, functional and nonfunctional requirements, software and hardware requirements of the project.

In addition to this, it also contain the information about environmental conditions required, safety and security requirements, software quality attributes of the project etc.

#### GOALS:-

The software requirements specification (SRS) is a communication tool between users and software designers. The specific goals of the SRS are:-

* Facilitating reviews.
* Describing the scope of the work.
* Providing a reference to software designers.
* Providing a framework for testing primary and secondary use cases.
* Including features to customer requirements.
* Providing a platform for ongoing refinement (via incomplete specs or question).

### SOFTWARE DEVELPOMENT LIFE CYCLE (SDLC)

Software Development Life Cycle (SDLC) is a framework defining tasks performed at each step in the software development process. SDLC is a structure followed by a development team within the software organization. It consists of a detailed plan describing how to develop, maintain and replace specific software. The life cycle defines a methodology for improving the quality of software and the overall development process.

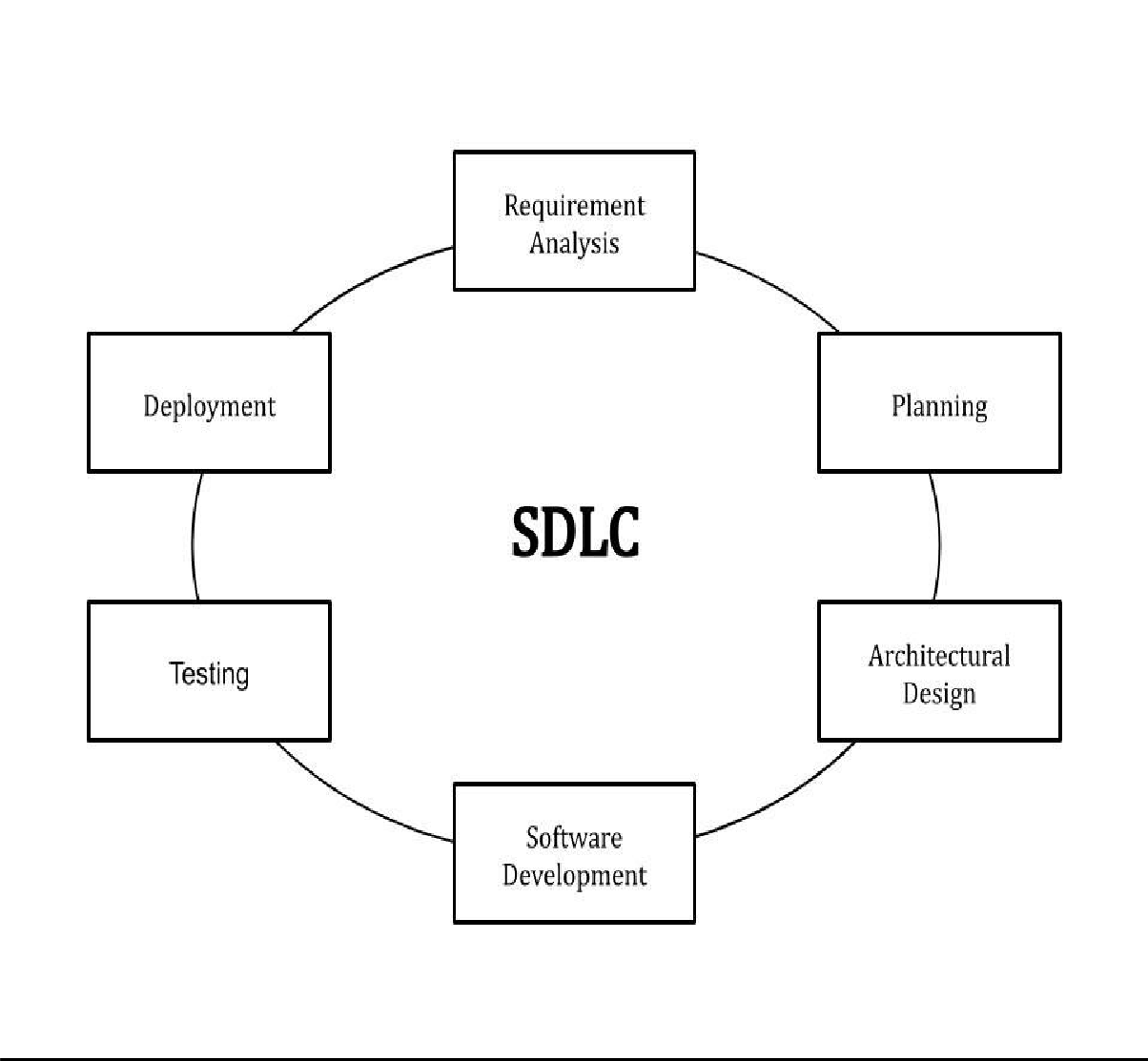
The software development life cycle is also known as the Software Development Process.

### SDLC MODELS

There are various 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.

Following are the most important and popular SDLC models:-

1. Waterfall Model
2. Incremental Process Model
   * Iterative Enhancement Model
   * The Rapid Application Development (RAD) Model
3. Evolutionary Process Model
   * Spiral Model
   * Prototyping Model



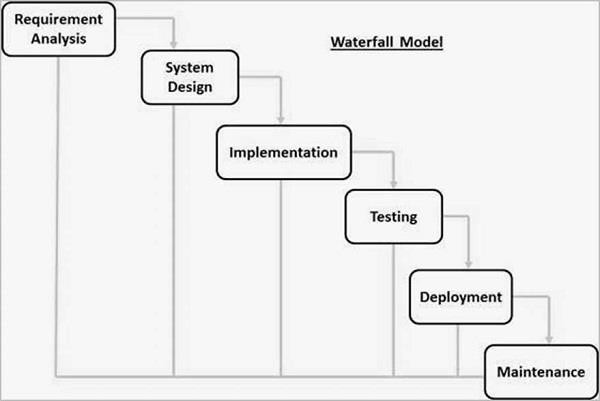
### WATERFALL MODEL

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. It was first introduced by Dr. Winston W. Royce in a paper published in 1970, the waterfall model is a software development process.

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.

The waterfall model is a sequential design process in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of Requirement Analysis, System Design, Implementation, Testing, Development and Maintenance.



#### Phases of Waterfall Model

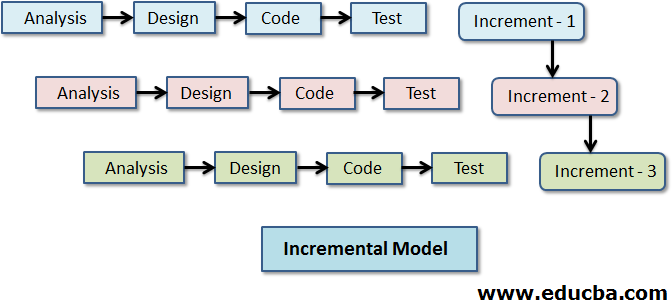
* **Requirement Gathering & Analysis:-** All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.
* **System Design:-** The requirement specifications from first phase are studied in this phase and the system design is 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 u nits, 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 un its developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.
* **Deployment of system:-** Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.
* **Maintenance:-** There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

#### INCREMENTAL PROCESS MODEL

The incremental build model is a method of software development where the model is designed, implemented and tested incrementally until the product is finished. It involves both development and maintenance. The product is defined as finished when it satisfies all of its requirements. this model combines the elements of the waterfall model with the iterative philosophy of prototyping.

The product is decomposed into a number of components, each of which are designed and build separately. Each component is delivered to the client when it is complete. This allows partial utilization of product and avoids a long development time. It also creates a large initial capital outlay with the subsequent long wait avoided. This model of development also helps ease the traumatic effect of introducing completely new system all at once.

There are some problems with the model. One is that each new build must be integrated with previous builds and any existing systems. The task of decomposing product into builds not trivial either. If there are too few builds and each build degenerates this turns into Build-And-Fix model. However if there are too many builds then there is little added utility from each build.



#### ITERATIVE ENHANCEMENT MODEL

The iterative enhancement life cycle model counters the third limitation of the Waterfall Model and tries to combine the benefits of both Prototyping and the Waterfall Model.

The basic idea is that the software should be developed in increments, where each increment adds some functional capability to the system until the full system is implemented. At each step extension and design modifications can be made. An advantage of this approach is that it can result in better testing, since testing each increment is likely to be easier than testing entire system like in the waterfall model. Furthermore, as in Prototyping, the increments provides feedback to the client which is useful for determining the final requirements of the system.

In the first step of iterative enhancement model, a simple initial implementation is done for a subset of the overall problem. This subset is the one that contains some of the key aspects of the problem which are easy to understand and implement and which forms a useful and usable system. A project control list is created which contains, in an order, all the tasks that must be performed to obtain the final implementation. This project control list gives an idea of how far the project is at any given step from the final system.

Each step consists of removing the next step from the list. Designing the implementation for the selected task, coding and testing the implementation and performing an analysis of the partial system obtained after this step and updating the list as a result of the analysis. These three phases are called the design phase, implementation phase and analysis phase. The process is iterated until the project control list is empty, at the time the final implementation of the system will be available.

#### RAPID APPLICATION DEVELOPMENT (RAD) MODEL

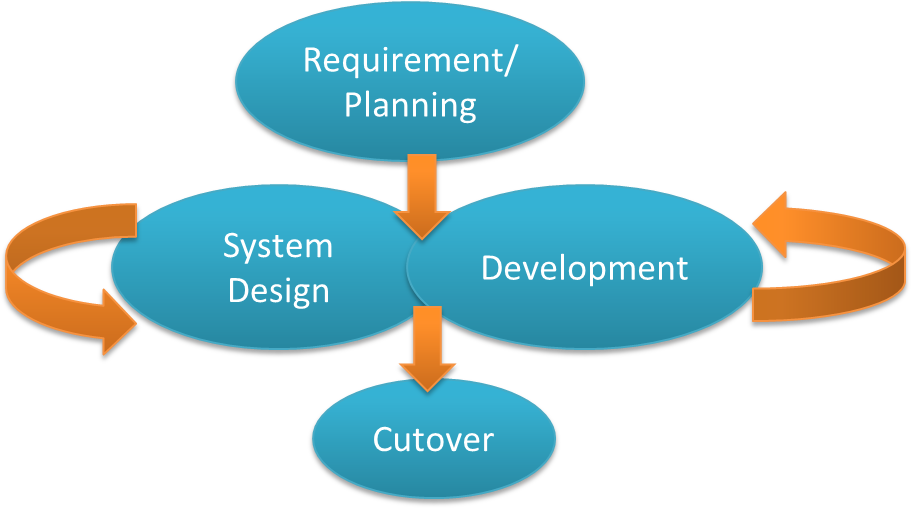
The **RAD (Rapid Application Development)** model is based on prototyping and iterative development with no specific planning involved. The process of writing the software itself involves the planning required for developing the product.

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

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

In the RAD model, the functional modules are developed in parallel as prototypes and are integrated to make the complete product for faster product delivery. Since there is no detailed preplanning, it makes it easier to incorporate the changes within the development process.

RAD projects follow iterative and incremental model and have small teams comprising of developers, domain experts, customer representatives and other IT resources working progressively on their component or prototype.



You can break down the process in a few ways, but in general, RAD follows four main phases:-

#### Phase 1: Requirement Planning

This phase is equivalent to a project scoping meeting. Although the planning phase is condensed compared to other project management methodologies, this is critical step for the ultimate success of the project. During this stage, developers, clients and team members communicate to determine the goals and expectations for the project as well as current and potential issues that would need to be addressed during the build.

A basic breakdown of this stage involves:

* + Researching the current problem
  + Defining the requirements for the project
  + Finalizing the requirements with each stakeholder’s approval.

It is important that everyone has the opportunity to evaluate the goals and expectations for the project and weigh in. by getting approval from each key stakeholder and developer, teams can avoid miscommunication and costly change orders down the road.

#### Phase 2: User Design

During this phase, users interact with systems analysts and develop models and prototypes that represent all system processes, inputs, and outputs. The RAD groups or subgroups typically use a combination of Joint Application Development (JAD) techniques and CASE tools to translate user needs into working models. User Design is a continuous interactive process that allows users to understand, modify, and eventually approve a working model of the system that meets their needs.

#### Phase 3: Construction Phase

Focuses on program and application development task similar to the SDLC. In RAD, however, user continue to participate and can still suggest changes or improvements as actual screens or reports are developed. Its tasks are programming and application development, coding, unit-integration and system testing.

#### Phase 4: Cutover Phase

Resembles the final tasks in the SDLC implementation phase, including data conversion, testing, changeover to the new system, and user training. Compared with traditional methods, the entire process is compressed. As a result, the new system is built, delivered, and placed in operation much sooner.

### EVOLUTIONARY SOFTWARE PROCESS MODEL

An evolutionary process model is a model whose stages consist of expanding increments of operational software product, with the direction of evolution being determined by operational experience. It combines elements of waterfall model and prototypes model. It develops the whole projects of number of stages, with the outcomes of one stages serving to identify the conceptual solutions for the next stage.

So the development proceeds in a series of implementations, each of which meets those requirements that are recognized and understood at the time of implementation. Each prototype is developed using a waterfall model to produce the prototypes.

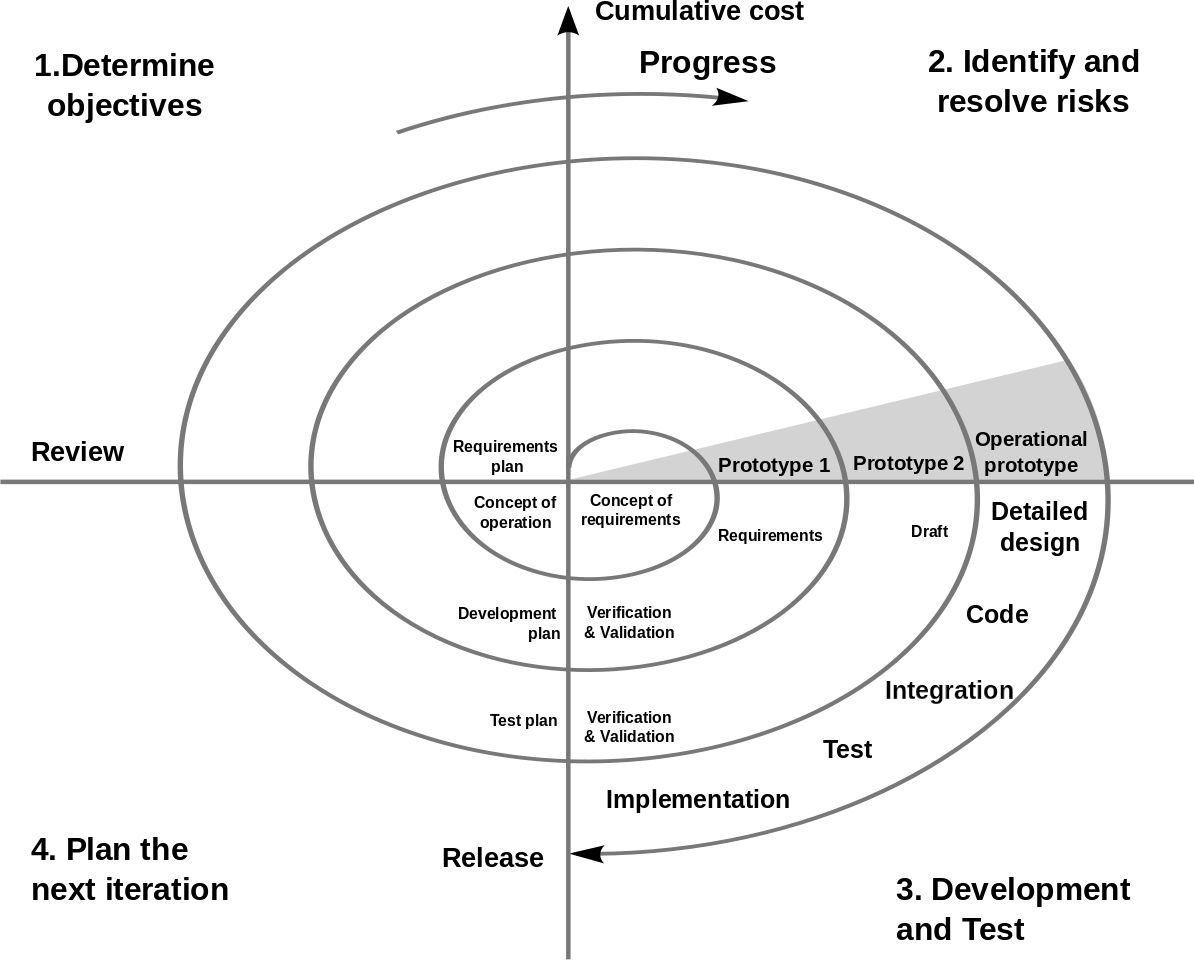
#### SPIRAL MODEL:

Spiral model is a combination of a waterfall model and iterative model. Each phase in spiral model begins with a design goal and ends with the client reviewing the progress. The Spiral model was first mentioned by Barry Boehm in his 1986 paper.

The development team in spiral –SDLC model starts with a small set of requirement and goes through each development phase for those set of requirements. the software engineering team adds functionality for the additional requirement in every- increasing spirals until the application is ready for the production phase.

The radial dimension of the model represents the cumulative cost. Each path around the spiral is inductive of increased costs. The angular dimension represents the progress made in completing each cycle. Each loop of the spiral from X-axis clockwise through 360degree represents one phase.

### SPIRAL MODEL

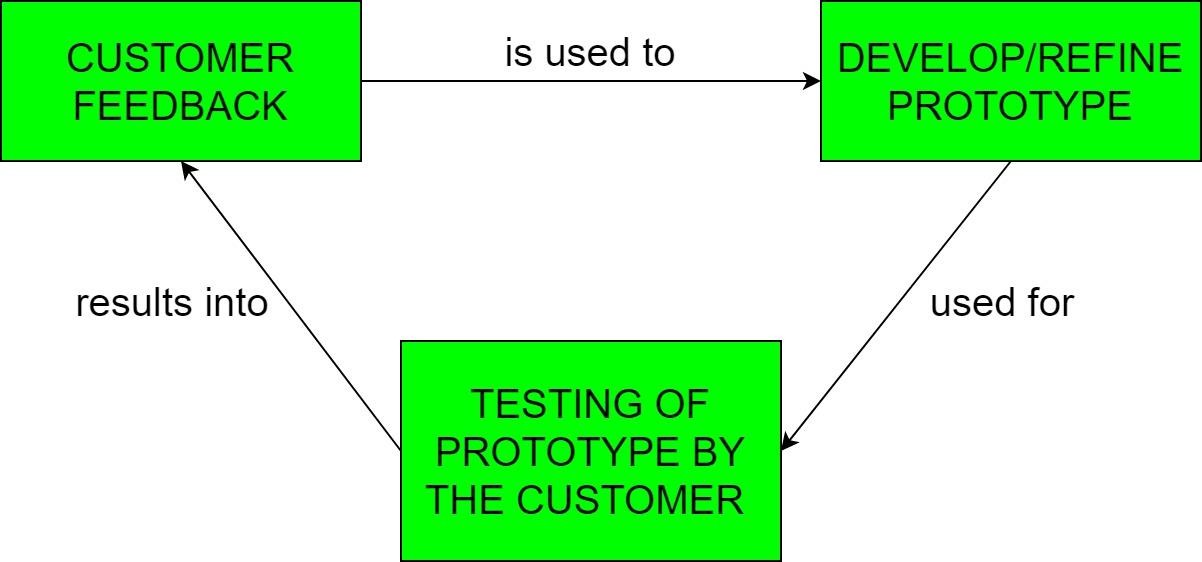


#### SPIRAL MODEL PHASES

1. **Planning:-** it includes estimating the cost, schedule and resource for the iteration. It also involves understanding the system requirements for continuous communication between the system analyst and the customer.
2. **Risk analysis:-** here is where the alternatives are analyzed and the associated risks are identified and evaluated.
3. **Development:-** it include testing, coding and developing software at the customer site. This may follow either the prototyping or classic life cycle approach.
4. **Evaluation:-** this involves a review of the proceeding developmental effort and therefor, planning for the next phase.

### 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:

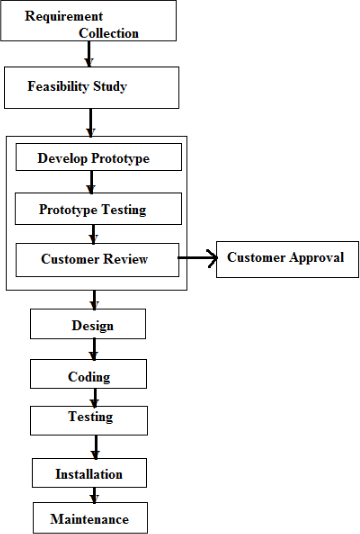


The Prototyping Model is one of the most popularly used Software Development Life Cycle Models (SDLC models). This model is used when the customers do not know the exact project requirements beforehand. In this model, a prototype of the end product is first developed, tested and refined as per customer 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 approaches for this model:-

1. **Rapid Throwaway Prototyping –** This technique offers a useful method of exploring ideas and getting customer feedback for each of them. In this method, a developed prototype need not necessarily be a part of the ultimately accepted prototype. Customer feedback helps in preventing unnecessary design faults and hence, the final prototype developed is of better quality.
2. **Evolutionary Prototyping –** In this method, the prototype developed initially is incrementally refined on the basis of customer feedback till it finally gets accepted. In comparison to Rapid Throwaway Prototyping, it offers a better approach which saves time as well as effort. This is because developing a prototype from scratch for every iteration of the process can sometimes be very frustrating for the developers.
3. **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. This method is helpful to reduce the feedback time between the user and the application development team.
4. **Extreme Prototyping –** This method is mainly used for web development. It is consists of three sequential independent phases:
   1. In this phase a basic prototype with all the existing static pages are presented in the HTML format.
   2. In the 2nd phase, Functional screens are made with a simulated data process using a prototype services layer.
   3. This is the final step where all the services are implemented and associated with the final prototype.



### COST ESTIMATION

Whether designing a building or developing software, successful projects require accurate cost estimates, cost estimations forecast the resources and associated costs needed to execute a project, which helps ensure you achieve project objectives within the approved timeline and budget.

cost estimation is a well developed discipline. By understanding the nuances of cost estimating and using standard estimation techniques, you can improve your forecasts. This complete guide to project cost estimating will walk you through the key concepts and major estimating techniques. Additionally, find how-toss templates and tips for key industries to help you get started with your estimates.

A model used for Cost Estimation

#### COCOMO (CONSTRUCTIVE COST MODEL)

The constructive Cost Model (COCOMO) is a procedural software cost estimation model developed by Barry W. Boehm. The model parameters are derived from fitting a regression formula using data from historical projects (61 projects for COCOMO 81 and 163 projects for COCOMO 11).

#### BASIC MODEL

The basic COCOMO estimate the software development effort using only lines of codes: Program size is estimated thousands of source lines of code (SLOC, KLOC)

There are three classes of software projects.

1. **Organic mode:-** A development project can be treated of the organic type, if the project deals with developing a well-understood application program, the size of the development team is reasonably small, and the team members are experienced in developing similar methods of projects.
2. **Semi-detached projects:-** A development project can be treated with semidetached type if the development consists of a mixture of experienced and inexperienced staff. Team members may have finite experience in related systems but may be unfamiliar with some aspects of the order being developed.
3. **Embedded projects:-** A development project is treated to be of an embedded type, if the software being developed is strongly coupled to complex hardware, or if the stringent regulations on the operational method exist.

#### TYPES OF COCOMO MODEL

1. **Basic COCOMO Model**

##### E=A(KLOC)^b

The above formula is used for the cost estimation of for the basic COCOMO model, and also is used in the subsequent models. The constant values a, b, c and d for the Basic Model for the different categories of system:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Software Projects | a | b | c | D |
| Organic | 2.4 | 1.05 | 2.5 | 0.38 |
| Semi Detached | 3.0 | 1.12 | 2.5 | 0.35 |
| Embedded | 3.6 | 1.20 | 2.5 | 0.32 |

The effort is measured in Person-Months and as evident from the formula is dependent on Kilo-Lines of code. The development time is measured in months. These formulas are used as such in the Basic Model calculations, as not much consideration of different factors such as reliability, expertise is taken into account, henceforth the estimate is rough.

1. **Intermediate Model –** this is extension of COCOMO model.

This estimation model makes use of set of “Cost Driver Attributes” to compute the cost of software.

#### Product attributes –

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

#### Hardware attributes –

* Run-time performance constraints
* Memory constraints
* The volatility of the virtual machine environment
* Required turnabout time

#### Personnel attributes –

* Analyst capability
* Software engineering capability
* Applications experience
* Virtual machine experience
* Programming language experience

#### Project attributes –

* Use of software tools
* Application of software engineering methods
* Required development schedule

#### Detailed Model –

Detailed COCOMO incorporates all characteristics of the intermediate version with an assessment of the cost driver’s impact on each step of the software engineering

process. The detailed model uses different effort multipliers for each cost driver attribute. In detailed COCOMO, the whole software is divided into different modules and then we apply COCOMO in different modules to estimate effort and then sum the effort.

The Six phases of detailed COCOMO are:

* + Planning and requirements
  + System design
  + Detailed design
  + Module code and test
  + Integration and test
  + Cost Constructive model

The effort is calculated as a function of program size and a set of cost drivers are given according to each phase of the software lifecycle.

### PROJECT SCHEDULING

Project scheduling is concerned with the techniques that can be employed to manage the activities that need to undertaken during the development of a project.

Scheduling is carried out in advance of the project commencing and involves:

* + Identifying the tasks that need to be carried out.
  + Estimating how long they will take.
  + Allocating resources (mainly personnel).
  + Scheduling when the tasks will occur.

Once the project is underway control needs to be exerted to ensure that the plan continues to represent the best prediction of what will occur in the future.

* + Based on what occurs during the development.
  + Often necessitates revision of the plan.

Effective project planning will help to ensure that the systems are delivered:

* + Within cost.
  + Within the time constraint.
  + To a specific standard of quality.

# SYSTEM DESIGN

#### PROJECT PLANNING

Project planning is an organized and integrated management process, which focuses on activities required for successful completion of the project. It prevents obstacles that arise in the project such as changes in projects or organization’s objectives, non- availability of resources, and so on. Project planning also helps in better utilization of resources and optimal usage of the allotted time for a project. The other objectives of project planning are listed below.

* It defines the roles and responsibilities of the project management team members. It ensures that the project management team works according to the business objectives.
* It checks feasibility of the schedule and user requirements.
* It determines project constraints.

Several individuals help in planning the project. These include senior management and project management team. Senior management is responsible for employing team members and providing resources required for the project. The project management team, which generally includes project managers and developers, is responsible for planning, determining, and tracking the activities of the project.

Project planning should be effective so that the project begins with well-defined tasks. Effective planning helps to minimize the additional costs incurred on the project while it is in progress. For effective project planning, some principles are followed. These principles are listed below:

* **Planning is necessary:-** planning should be done before a project begins. For effective planning, objectives and schedules should be clear and understandable.
* **Risk analysis:-** before starting the project, senior management and the project management team should consider the risks that may affect the project. For example, the user may desire changes in requirements while the project is in progress. In such a case, the estimation of time and cost should be done according to those requirements.
* **Tracking of project plan:-** once the project plan is prepared, it should be tracked and modified accordingly.
* **Meet quality standards and produce quality deliverables:-** the project plan should identify processes by which the project management team can ensure quality in software. Based on the process selected for ensuring quality, the time and costs for the project is estimated.
* **Description of flexibility to accommodate changes:**- the result of project planning is recorded in the form of a project plan, which should allow new changes to be accommodated when the project is in progress.

#### MODULES:-

Modules are classified as a unit that forms part of something bigger. One or more independently developed modules make up a program. For a website the modules can be categories as the following:-

* Domain name of the website
* Business email address.
* Website building software
* Website hosting
* Website design like through HTML &CSS

**System design**

System design is the process of defining the architecture, modules, interfaces, and data for a system to satisfy specified requirements. Systems design could be seen as the application of systems theory to product development. There is some overlap with the disciplines of systems analysis, systems architecture and systems engineering.

If the broader topic of product development “blends the perspective of marketing, design is the act of taking the marketing information and creating the design of the product to be manufactured. Systems design is therefore the process of defining and developing systems to satisfy specified requirements of the user.

The basic study of system design is the understanding of component parts and their subsequent interaction with one another.

**Definition:-** Systems design is the process of defining elements of a system like modules, architecture, components and their interfaces and data for a system based on the specified requirements. It is the process of defining, developing and designing systems which satisfies the specific needs and requirements of a business or organization.

**Description:-** A systemic approach is required for a coherent and well-running system. Bottom-up or Top-down approach is required to take into account all related variables of the system. A designer uses the modeling languages to express the information and knowledge in a structure of system that is defined by a consistent set of rules and definitions. The designs can be defined in graphical or textual modeling languages.

Some of the examples of graphical modeling languages area. Unified Modeling Languages(UML):- To describe software both structurally and behaviorally with graphical notation.

* 1. Flowchart :- A schematic or stepwise representation of an algorithm.
  2. Business Process Modeling Notation:- Used for process Modeling language.
  3. Systems Modeling Language(SML):- used for systems engineering.

#### Architectural design

The architectural design of a system emphasizes the design of the system architecture that describes the structure, behavior and more views of that system and analysis.

#### Physical design

The physical design relates to the actual input and output processes of the system. This is explained in terms of how data is input into a system, how it is verified/authenticated, how it is processed, and how it is displayed. In physical design, the following requirements about the system are decided.

1. Input requirement,
2. Output requirement,
3. Storage requirement,
4. Processing requirements,
5. System control and backup or recovery.

Put another way, the physical portion of the system design can generally be broken down into three sub-tasks:-

1. User interface design
2. Data design
3. Process design

User interface design is concerned with how users add information to the system and with how the system presents information back to them. Data design is concerned with how data moves through the system, and with how and where it is validated, secured and/or transformed as it flows into, through and out of the system. At the end of the system design phase, documentation describing the three sub-tasks is produced and made available for use in the next phase.

Physical design, in the context, does not refer to the tangible physical design of an

information system. To use an analogy, a personal computer’s physical design involves input via a keyboard, processing within the CPU, and output via a monitor, printer, etc. It involves a detailed design of a user and a product database structure processor. The H/S personal specification is developed for the proposed system.

#### Alternative design methodologies

**Rapid application development (RAD)**

Rapid application development(RAD) is a methodology in which a system designer produces prototypes for an end-user. The end-user reviews the prototype, and offers feedback on its suitability. This process is repeated until the end-user is satisfied with the final system.

#### Joint application design(JAD)

Joint application design (JAD) is a methodology which evolved from RAD, in which a system designer consults with a group consisting of the following parties:

* + Executive sponsor
  + System designer
  + Managers of the system

JAD involves a number of stages, in which the group collectively develops an agreed pattern for the design and implementation of the system.

#### Basic Steps :-

1. **Clarify and agree on the scope of the system**

* **User cases** (description of sequences of events that, taken together, lead to a system doing something useful)
* Who is going to use it?
* How are they going to use it?

##### Constraints

* Mainly identify traffic data handling constraints at scale.
* Scale of the system such as requests per second, per requests types, data written per second, data read per second)
* Special system requirements such as multi-threading, read or write oriented.

#### High level architecture design (Abstract design)

* Sketch the important components and connections between them, but don’t go into some details.
* Application service layer (serves the requests)
* List different services required.
* Data storage layer
* Eg. Usually a scalable system includes webserver (load balancer), service (service partition), database (master/slave database cluster) and caching systems.

#### Component design

* Component + specific APIs required for each of them.
* Object oriented design for functionalities.
* Map features to modules: one scenario for one module.
* Consider the relationships among modules:
  + Certain functions must have unique instance (singletons)
  + Core object can be made up of many other objects (composition).
  + One object is another object (inheritance).
* Database schema design.

#### Understanding Bottlenecks

* Perhaps your system needs a load balancer and many machines behind it to handle the user requests. \*Or maybe the data is so huge that you need to distribute your database on multiple machines. What are some of the downsides that occur from doing that?
* Is the database too slow and does it need some in-memory caching?

#### Scaling

* **Vertical scaling**

o You scale by adding more power(CPU,RAM) to your existing machine.

#### Horizontal scaling

o You scale by adding more machines into your pool of resources.

#### Caching

* Load balancing helps you scale horizontally across an ever-increasing number of servers, but caching will enable you to make vastly better use of the resources you already have, as well as making otherwise unattainable product requirements feasible.
* Application caching requires explicit integration in the application code itself. Usually it will check if a value is in the cache; if not, retrieve the value from the database.
* Database caching tends to be “free”. When you flip your database on, you’re going to get some level of default configuration which will be optimized for a generic use case, and by tweaking them to your system’s access patterns you can generally squeeze a great deal of performance improvement.
* In memory caches are most potent in terms of raw performance. This is because they store their entire set of data in memory and accesses to RAM are orders of magnitude faster than those to disk.

#### Load balancing

* Public servers of a scalable web service are hidden behind a load balancer. This load balancer evenly distributes load onto your group/cluster of application servers.
* Types: Smart client (hard to get it perfect). Hardware load balancers ($$$ but reliable), Software load balancers (Hybrid- works for most systems).

#### Database replication

o Database replication is the frequent electronic copying data from a database in one computer or server to a database in another so that all users share the same level of information. The result is a distributed database in which users can access data relevant to their tasks without interfering with the work of others. The implementation of database replication for the purpose of eliminating data ambiguity or inconsistency among users is known as normalization.

#### Database partitioning

o Partitioning of relational data usually refers to decomposing your tables either row- wise (horizontally) or column-wise (vertically).

#### Map reduce

* For sufficiently small systems you can often get away with abhor queries on a SQL database, but that approach ,may not scale up trivially once the quantity of data stored or write-load requires sharing your databases, and will usually require dedicated slaves for the purpose of performing these queries.
* Adding a map-reduce layer makes it possible to perform data and/or processing intensive operations in a reasonable amount of time. You might use it for calculating suggested users in a social graph, or for generating analytics reports.

#### Platform layer (services)

* Separating the platform and web application allow you to scale the pieces independently. If you add a new API, you can add platform servers without adding unnecessary capacity for your web application tier.
* Adding a platform layer can be a way to reuse your infrastructure for multiple products or interfaces without writing too much redundant boilerplate code for dealing with caches, databases, etc.

#### Key topics for designing a system

1. **Concurrency**

* Do you understand threads, deadlock, and starvation? Do you know how to parallelize algorithms? Do you understand consistency and coherence?

#### Networking

* Do you roughly understand IPC and TCP/IP? Do you know the difference between throughout and latency, and when each is the relevant factor?

#### Abstraction

* You should understand the systems you’re building upon. Do you know roughly how an OS, file system, and database work? Do you know about the various levels of caching in a modern OS?

#### Real-world performance

* You should be familiar with the speed of everything your computer can do, including the relative performance of RAM, disk, SSD and your network.

#### Estimation

* Estimation, especially in the form of a back-of the envelope calculation, is important because it helps you narrow down the list of possible solutions to only the ones that are feasible. Then you have only a few prototypes or micro-benchmarks to write.

#### Availability & Reliability

* Are you thinking about how things can fail, especially in a distributed environment? Do know how to design a system to cope with network failures? Do you understand durability?

#### Web app system design considerations:

* Security (CORS)
* Using CDN h
* A content delivery network (CDN) is a system of distributed servers that deliver webpages and other web content to a user based on the geographic locations of the user, the origin of the webpage and a content delivery server.
* This services is effective in speeding the delivery of content of websites with high traffic and websites that have global reach. The closer the CDN server is to the user geographically, the faster the content will be delivered to the user.
* CDNs also provide protection from large surges in traffic.
* Full text search

o Which archive fast search responses because, instead of searching the text directly, it searches an index instead.

* Offline support/ progressive enhancement

o Service workers

* Web workers
* Server side rendering
* Asynchronous loading of assets
* Minimizing network requests
* Developer productivity/ tooling
* Accessibility
* Internationalization
* Responsive design
* Browser compatibility

### FLOWCHART

A flowchart is a type of diagram that represents an algorithm, workflow or process. The flowchart shows the steps as boxes of various kinds, and their order by connecting the boxes with arrows. This diagrammatic representation illustrates a solution model to a given problem. Flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields.

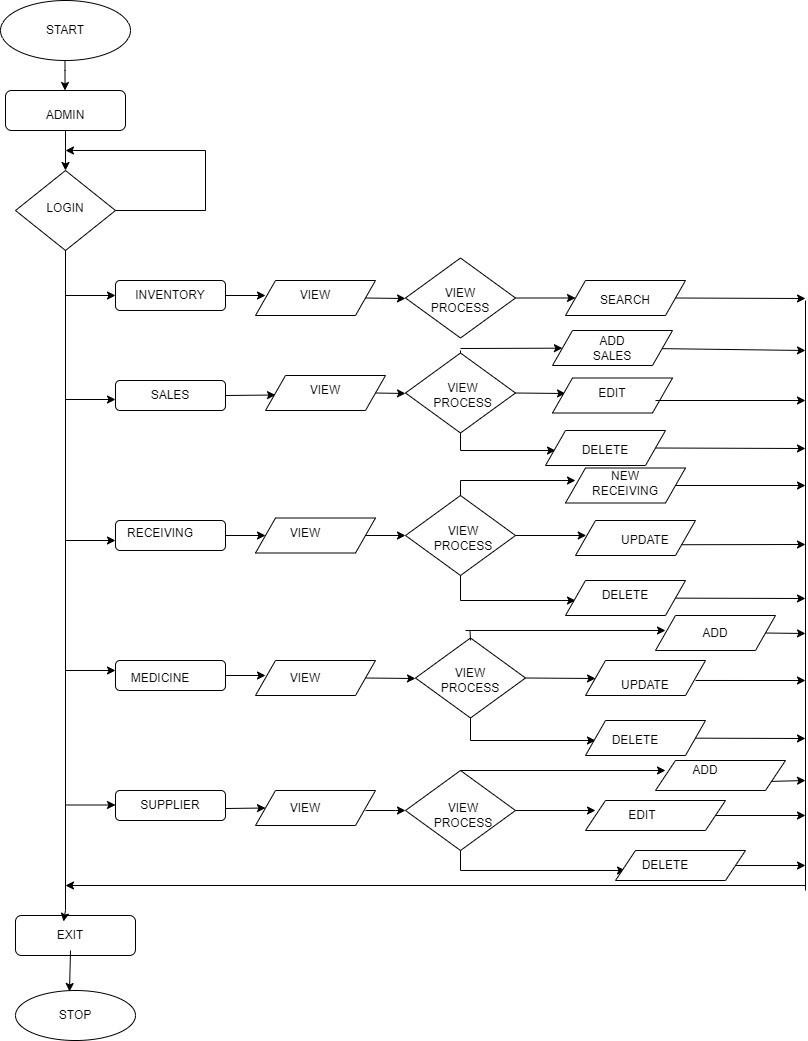
Flowcharts display the steps in code as shapes connected together with arrows. The main goal is to create a rough draft of a solution to a coding problem.

#### SYMBOLS OF FLOWCHART

Common flowcharting symbols and examples follow. When first reaching this section, focus on the simple symbols.

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Symbol Name** | **Purpose** |
| Start Stop | Start/Stop | Used at the beginning and end of the algorithm to show start and end of the program. |
| Process | Process | Indicates processes like mathematical operations. |
| Input/ Output | Input/ Output | Used for denoting program inputs and outputs. |
| Decision | Decision | Stands for decision statements in a program, where answer is usually Yes or No. |
| Arrow | Arrow | Shows relationships between different shapes. |
| On-page Connector | On-page Connector | Connects two or more parts of a flowchart, which are on the same page. |

|  |  |  |
| --- | --- | --- |
| Off-page Connector | Off-page Connector | Connects two parts of a flowchart which are spread over different pages. |



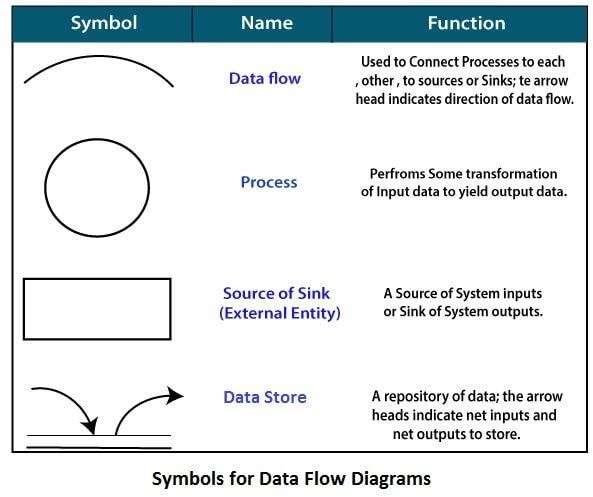
### DATA FLOW DIAGRAM

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It can be manual, automated, or a combination of both.

It shows how data enters and leaves the system, what changes the information, and where data is stored.

The objective of a DFD is to show the scope and boundaries of a system as a whole. It may be used as a communication tool between a system analyst and any person who plays a part in the order that acts as a starting point for redesigning a system. The DFD is also called as a data flow graph or bubble chart.

Standard symbols for DFDs are derived from the electric circuit diagram analysis and are shown in fig:

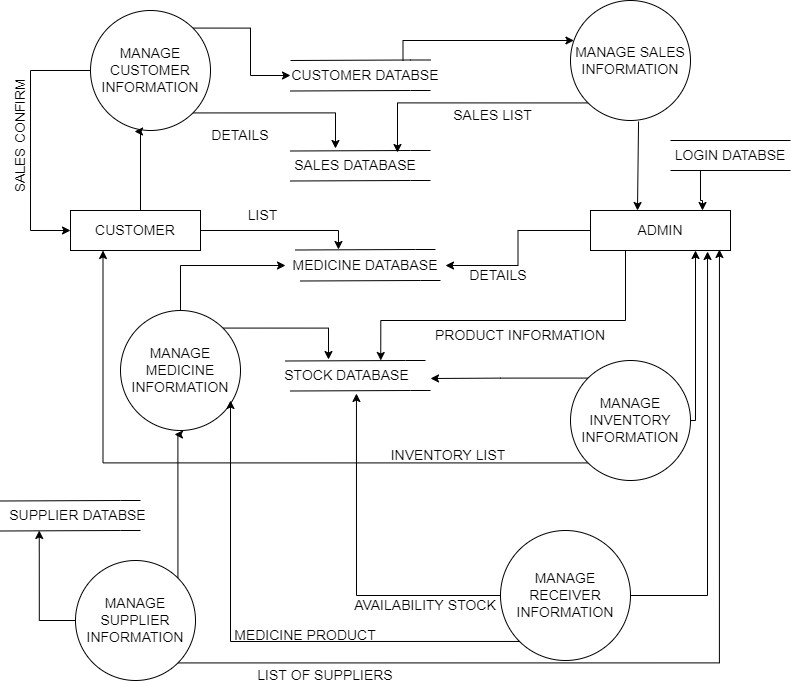


**Circle:** A circle (bubble) shows a process that transforms data inputs into data outputs.

**Data Flow:** A curved line shows the flow of data into or out of a process or data store.

**Data Store:** A set of parallel lines shows a place for the collection of data items. A data store indicates that the data is stored which can be used at a later stage or by the other processes in a different order. The data store can have an element or group of elements.

**Source or Sink:** Source or Sink is an external entity and acts as a source of system inputs or sink of system outputs.



### DFD OF PHARMACY SALES & INVENTORY SYSTEM

**E-R DIAGRAM**

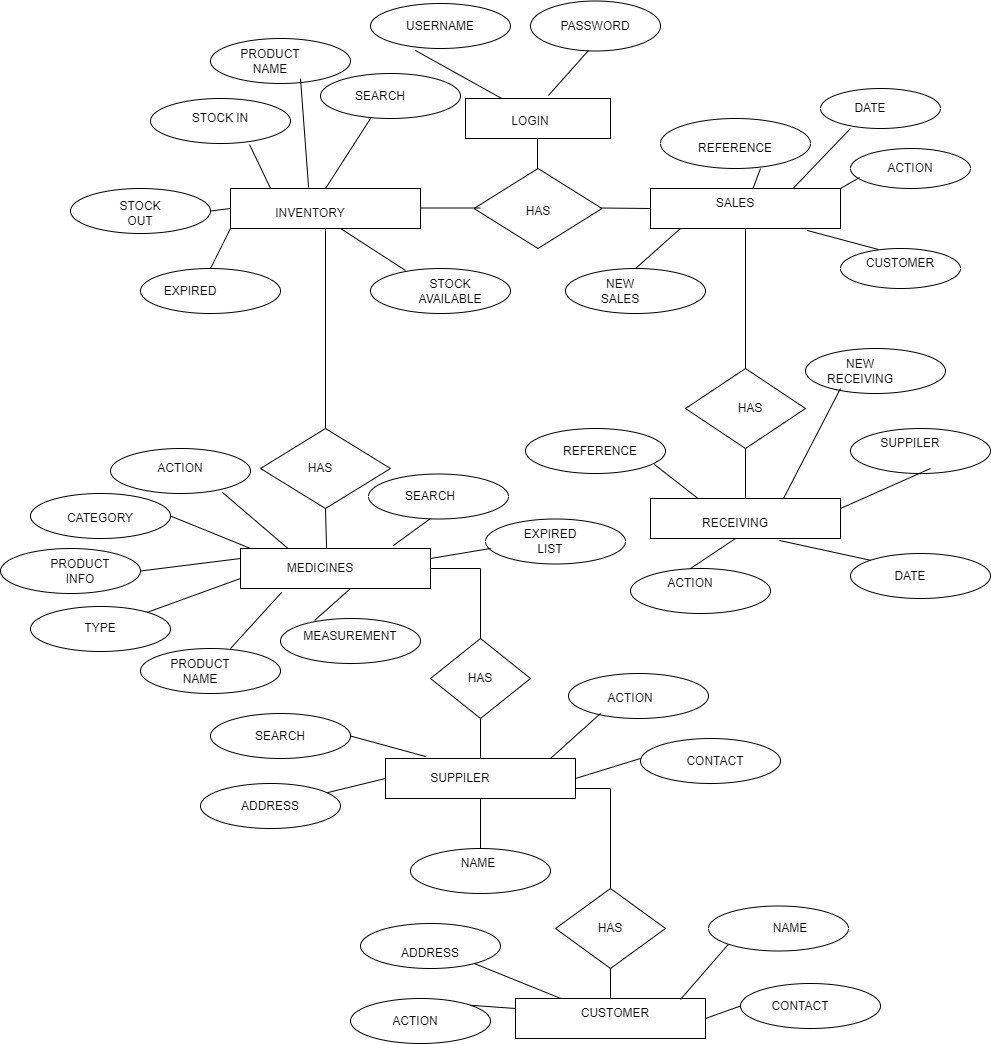
ER-modelling is a data modelling method used in software engineering to produce a conceptual data model of an information system. Diagrams created using this ER- modelling method are called Entity-Relationship Diagrams or ER diagrams or ERDs.

##### Purpose of ERD

* + The database analyst gains a better understanding of the data to be contained in the database through the step of constructing the ERD.
  + The ERD serves as a documentation tool.
  + Finally, the ERD is used to connect the logical structure of the database to users. In particular, the ERD effectively communicates the logic of the database to users.

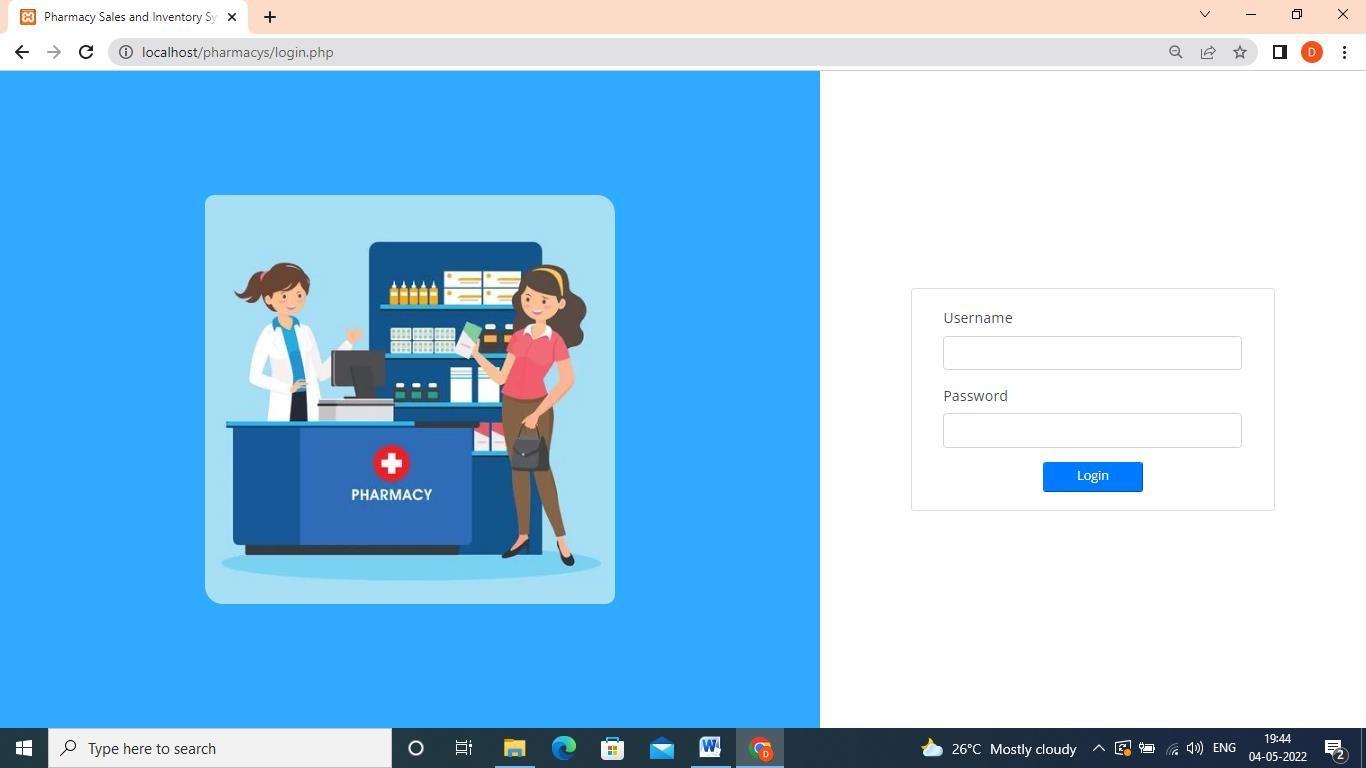
##### USE OF ER DIAGRAMS

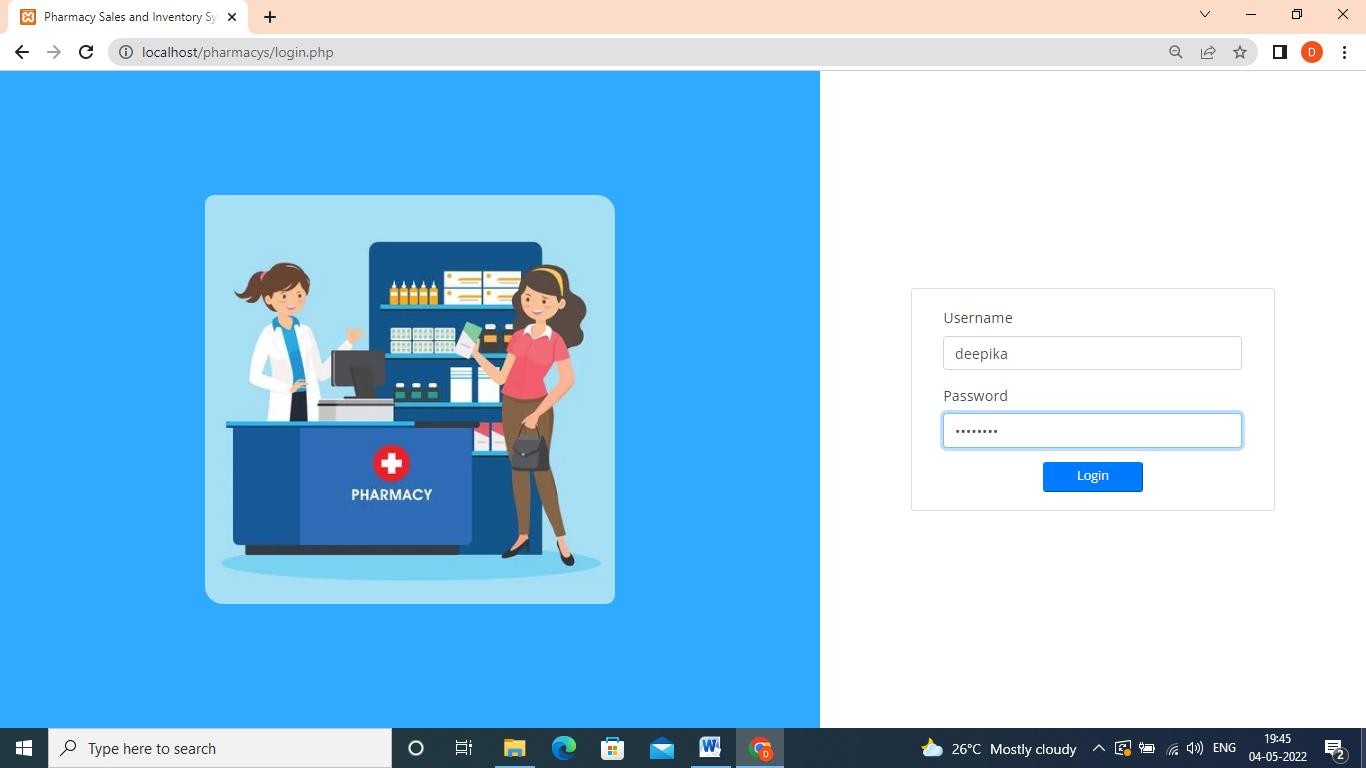
* + Helps you to define terms related to entity relationship modelling.
  + Provide a preview of how all your tables should connect, what fields are going to be on each table.
  + Helps to describe entities, attributes, relationships.
  + ER diagrams can be used by database designers as a blueprint for implementing data in specific software application.

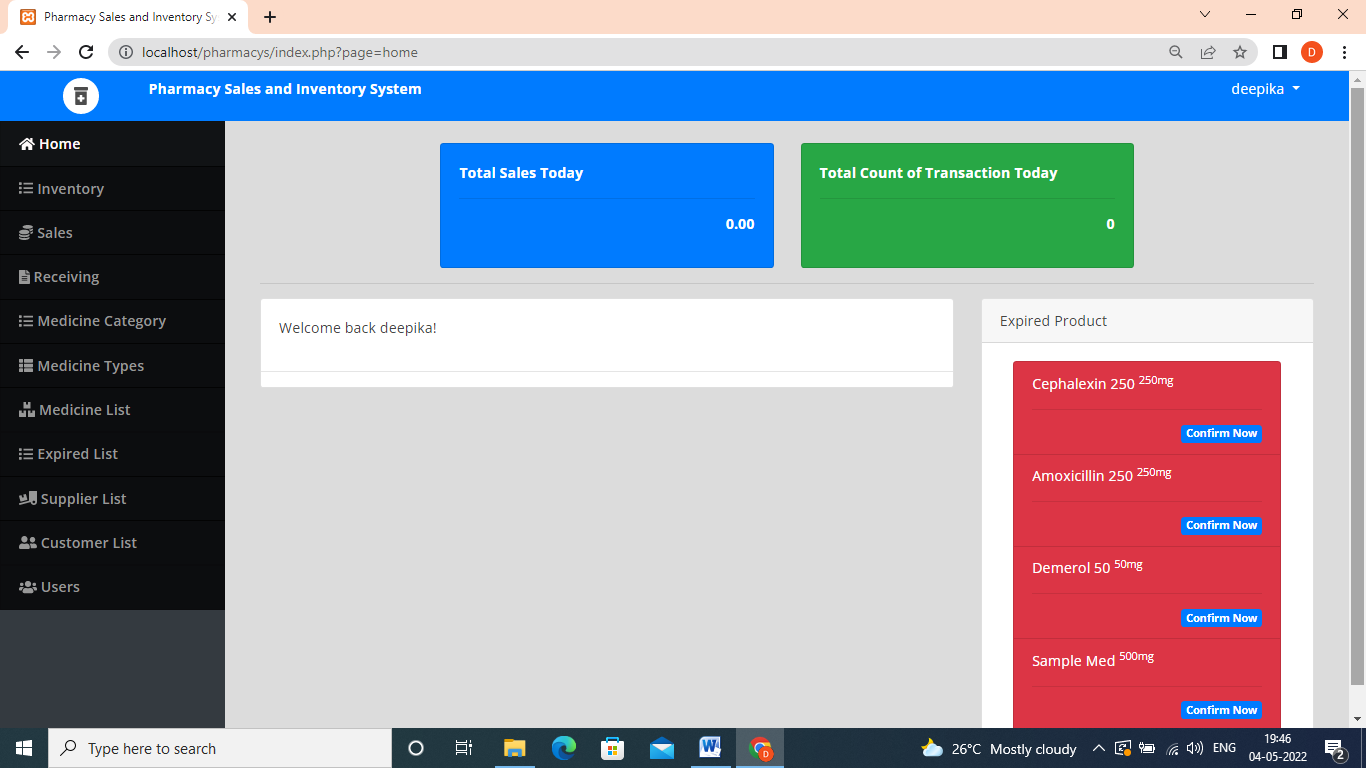


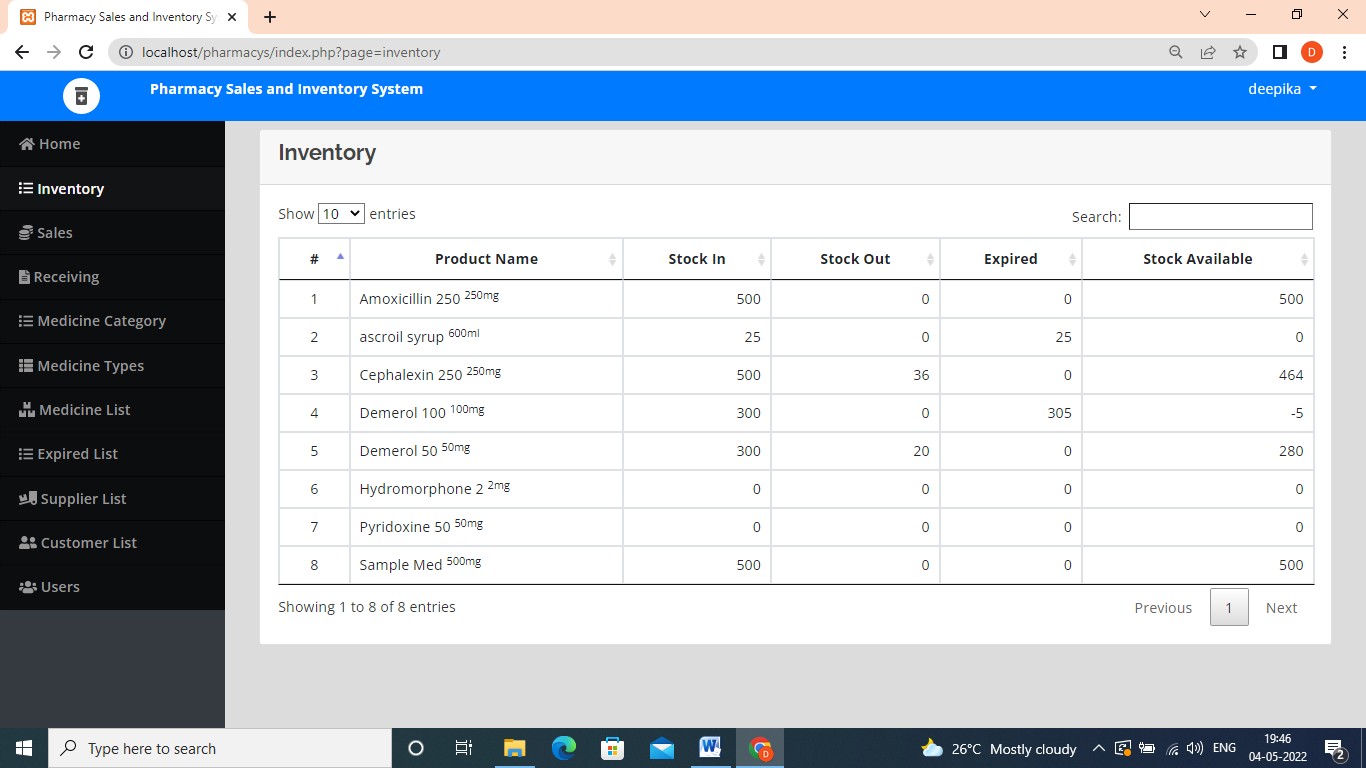
**ER- DIAGRAM OF PHARMACY SALES & INVENTORY SYSTEM**

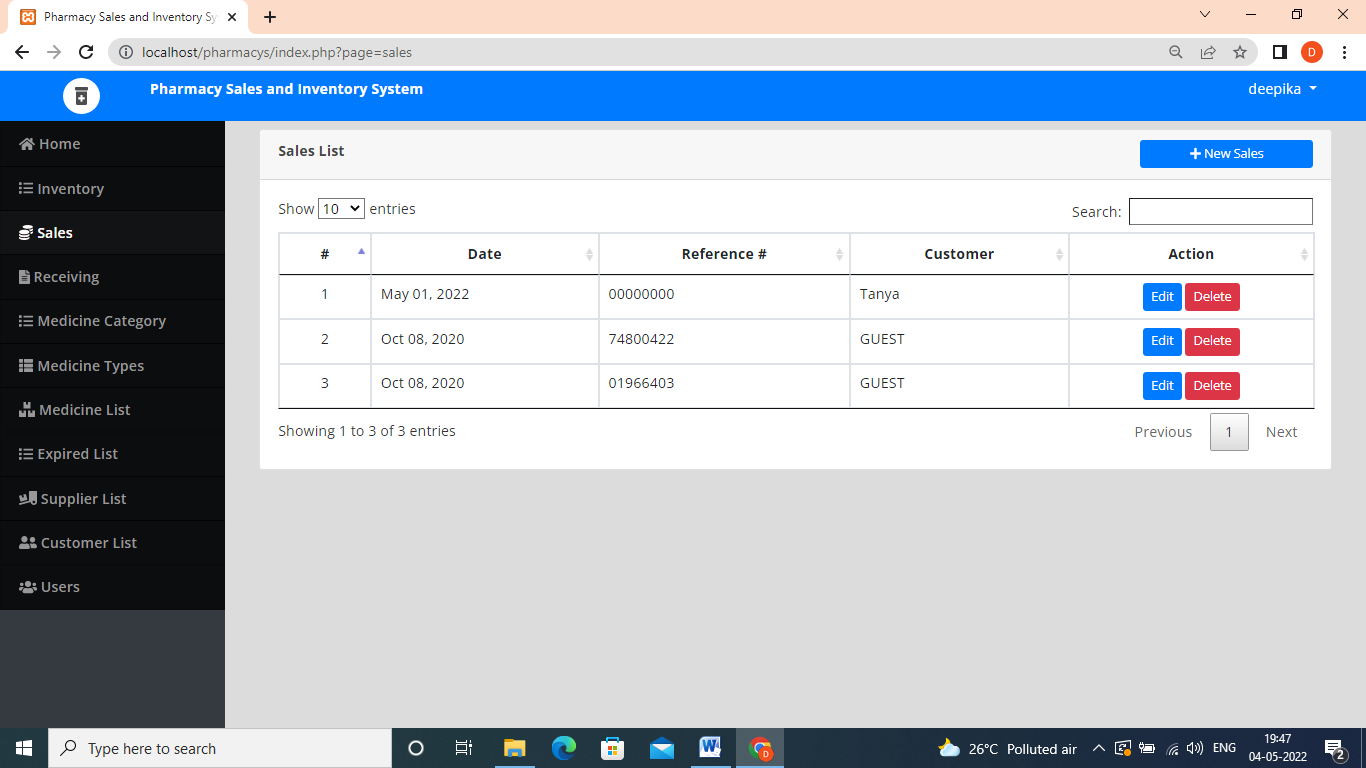
# SCREENSHOTS

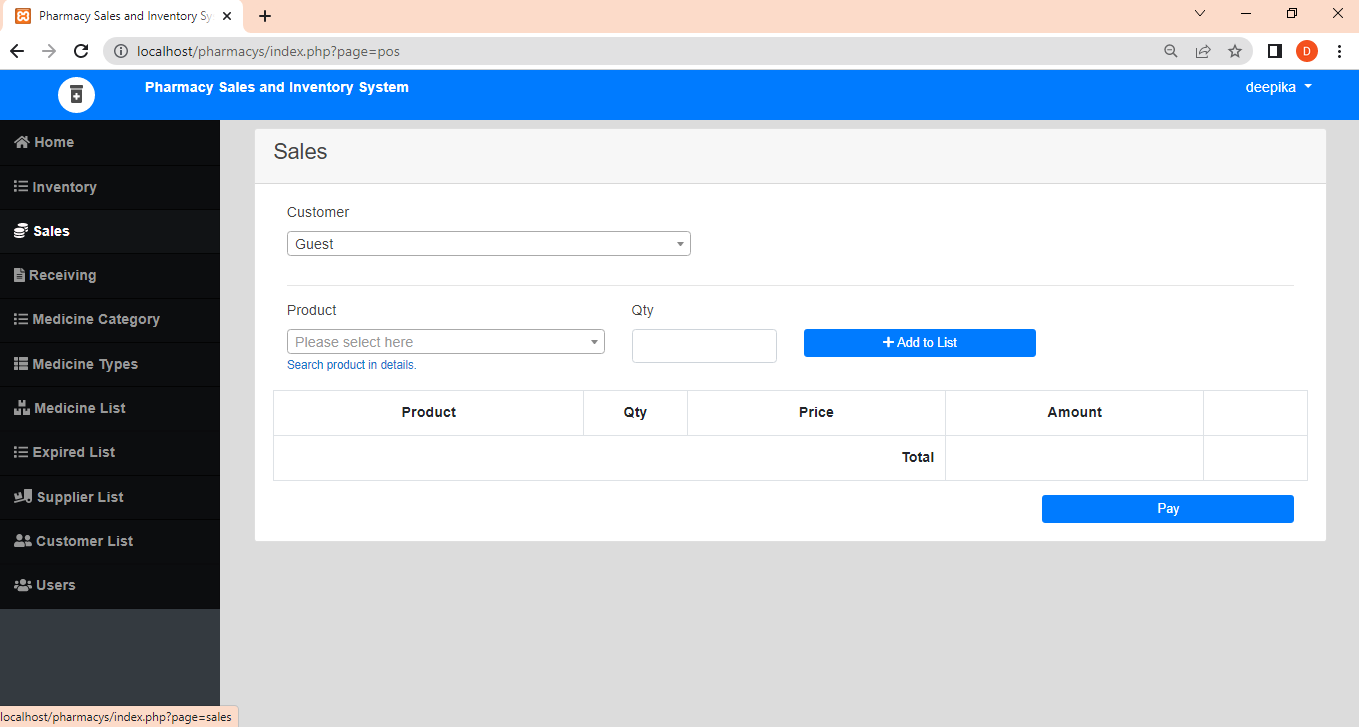


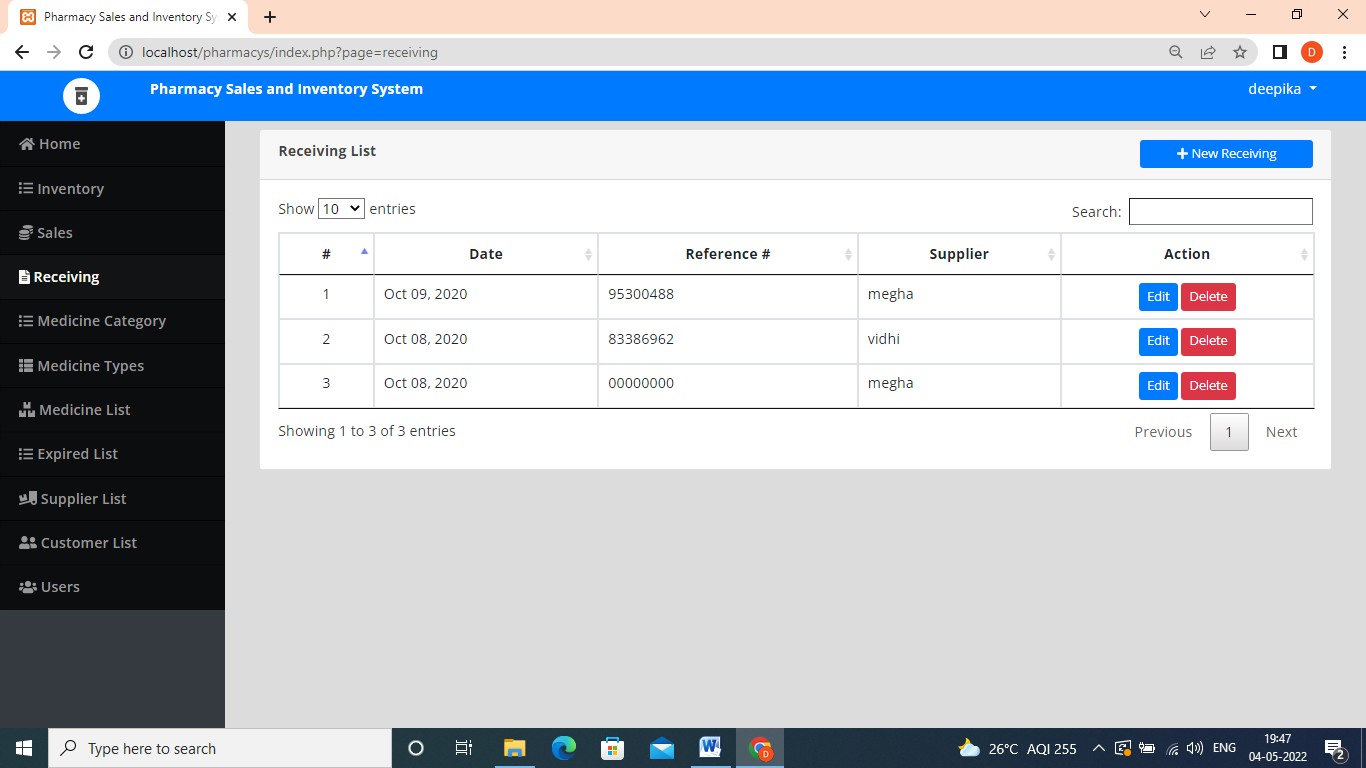


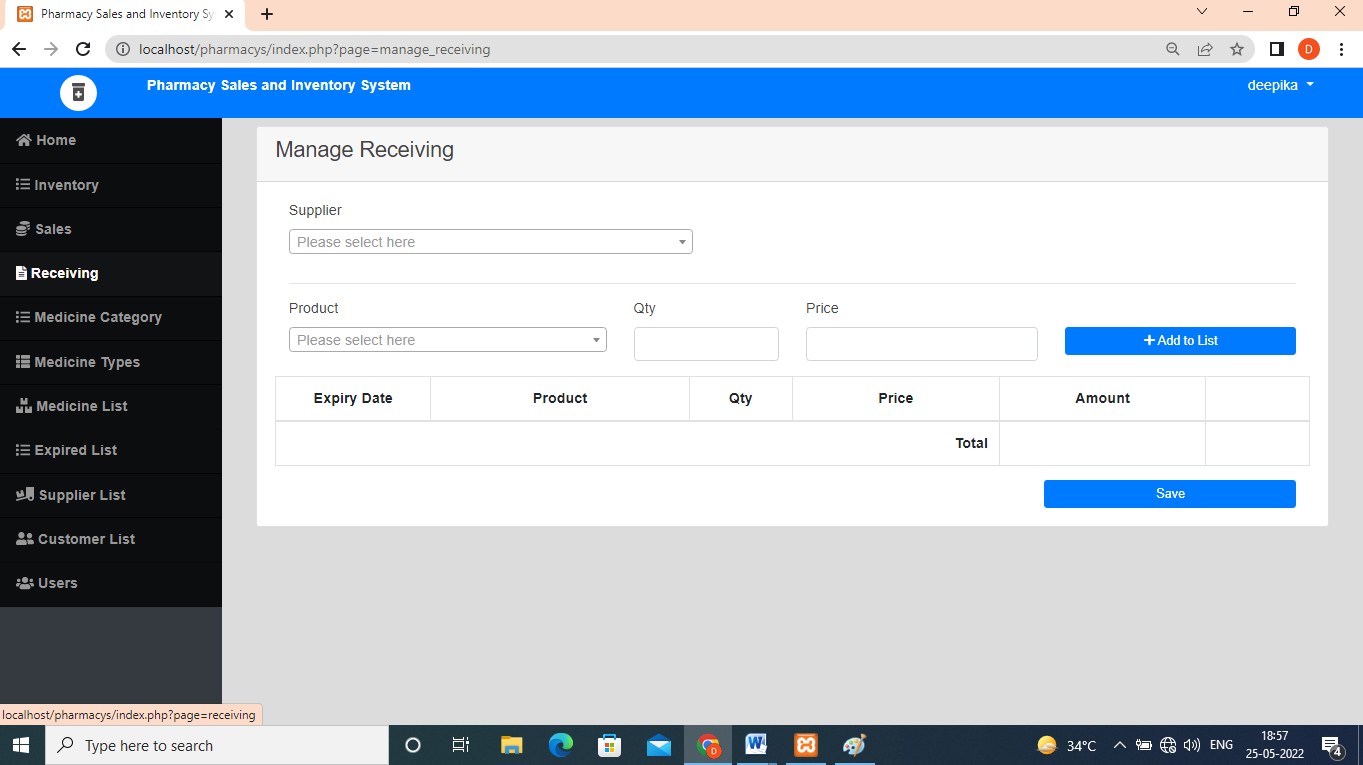


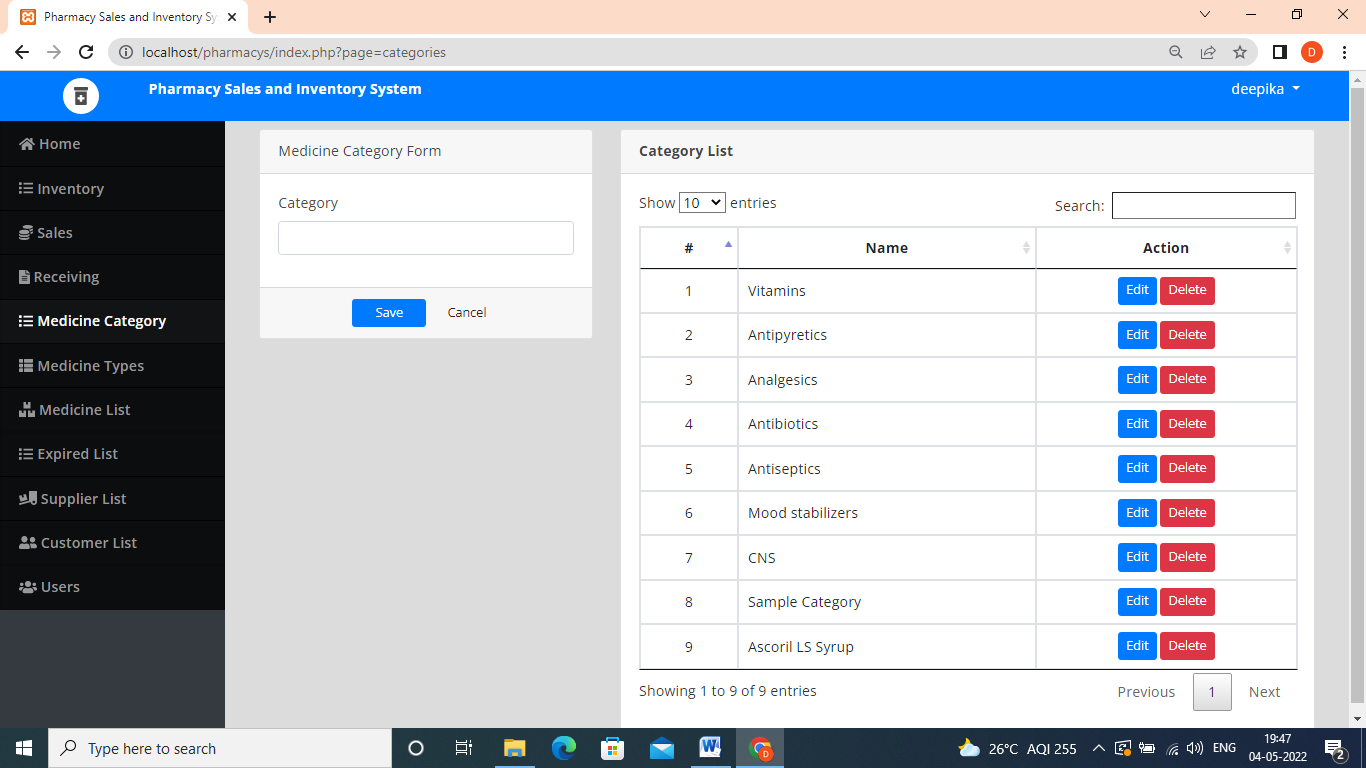


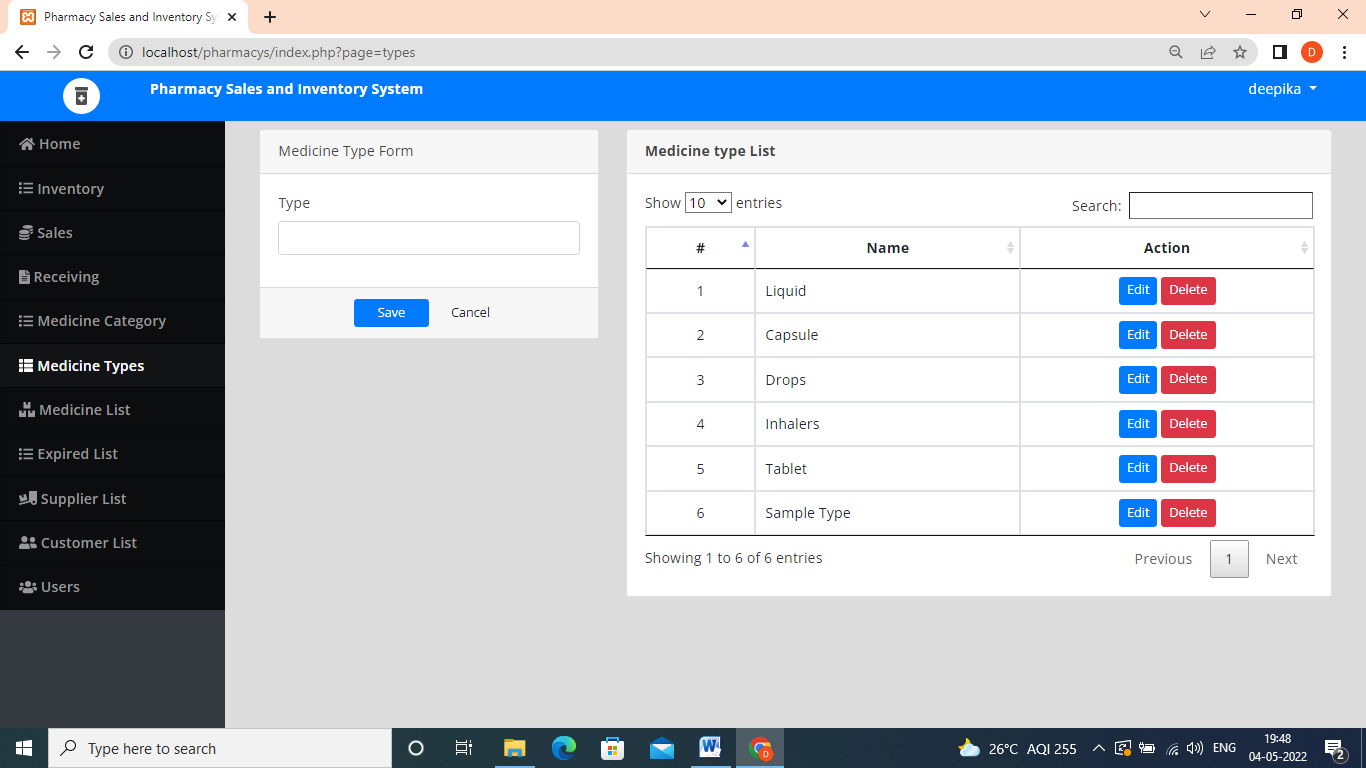


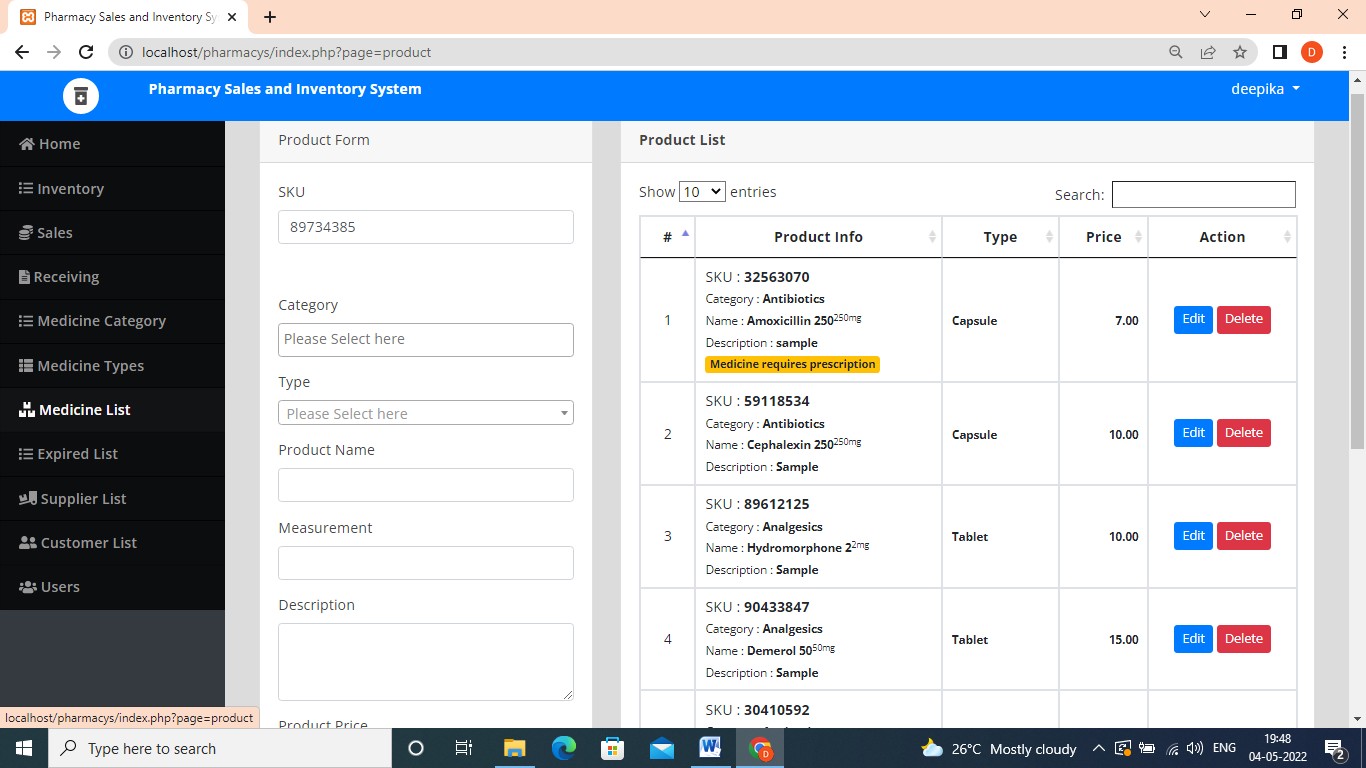


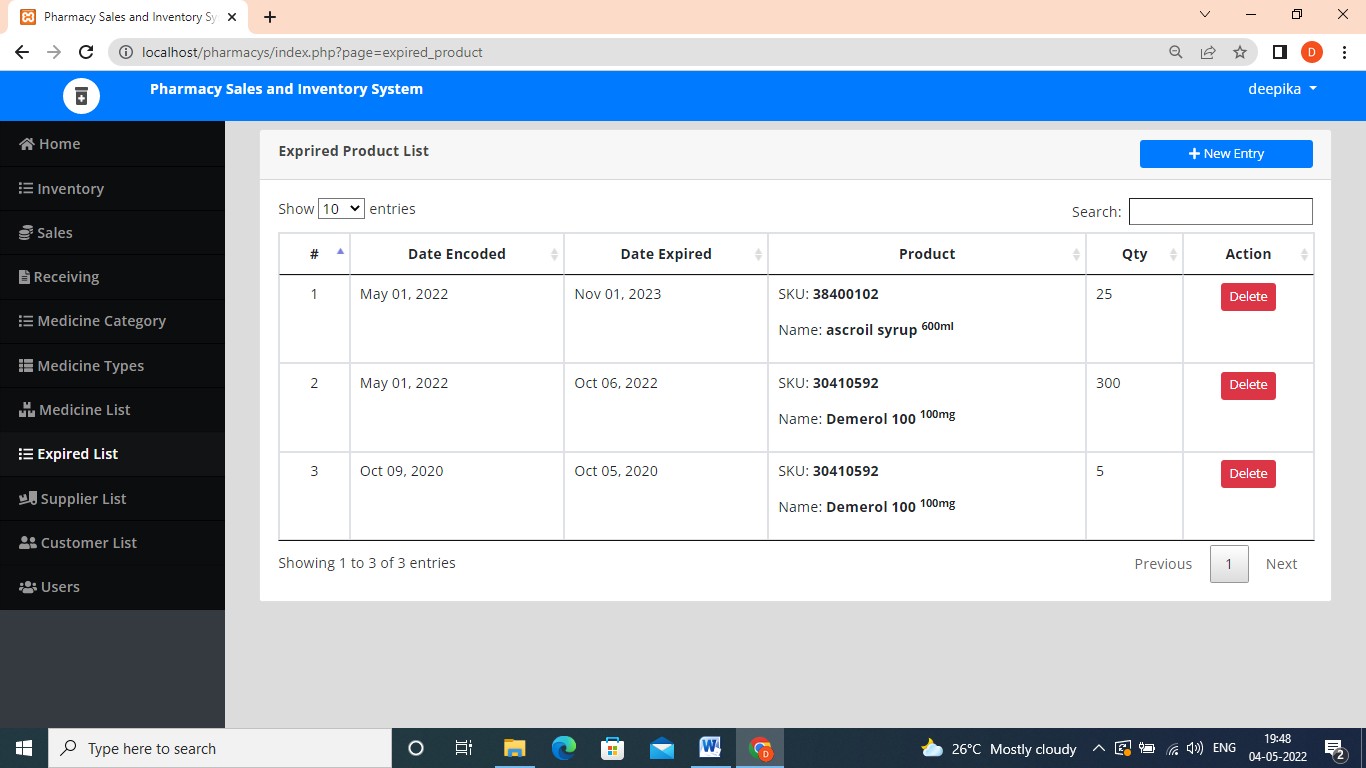


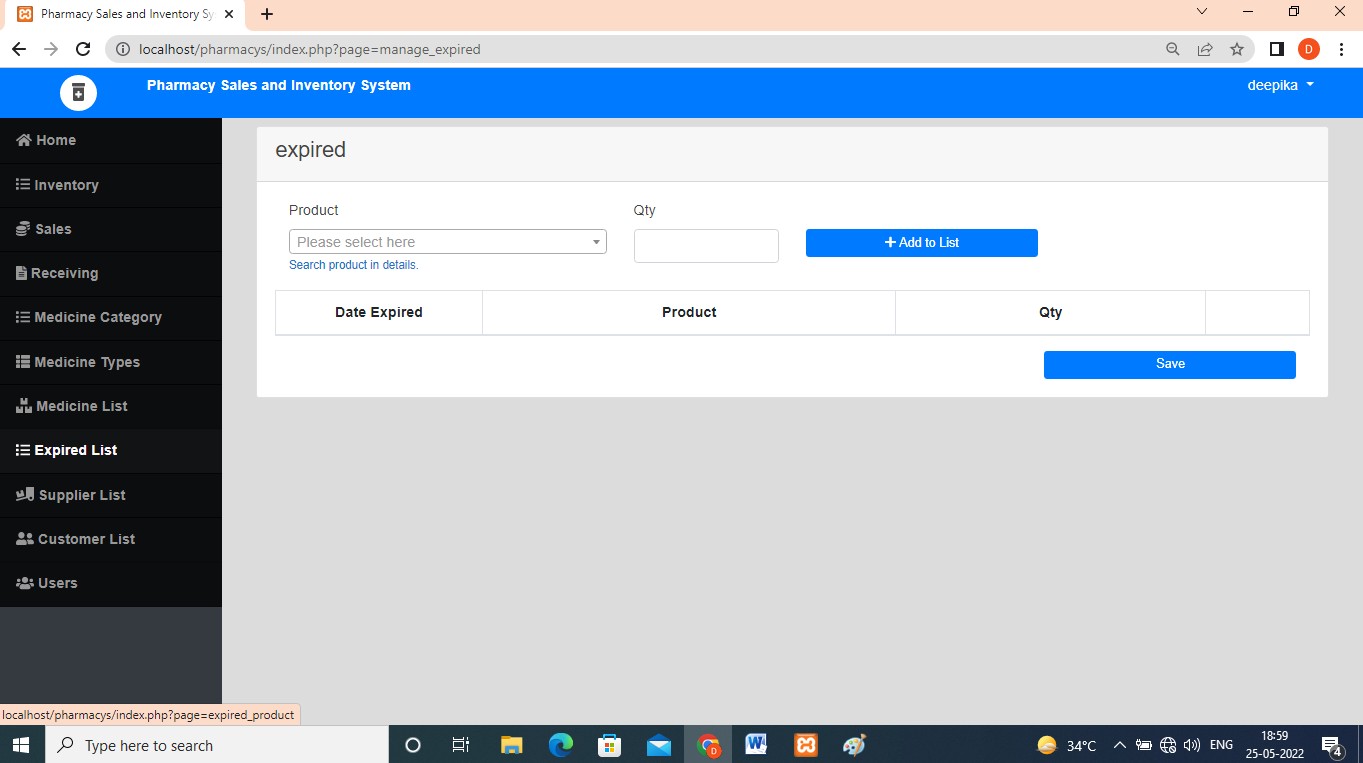


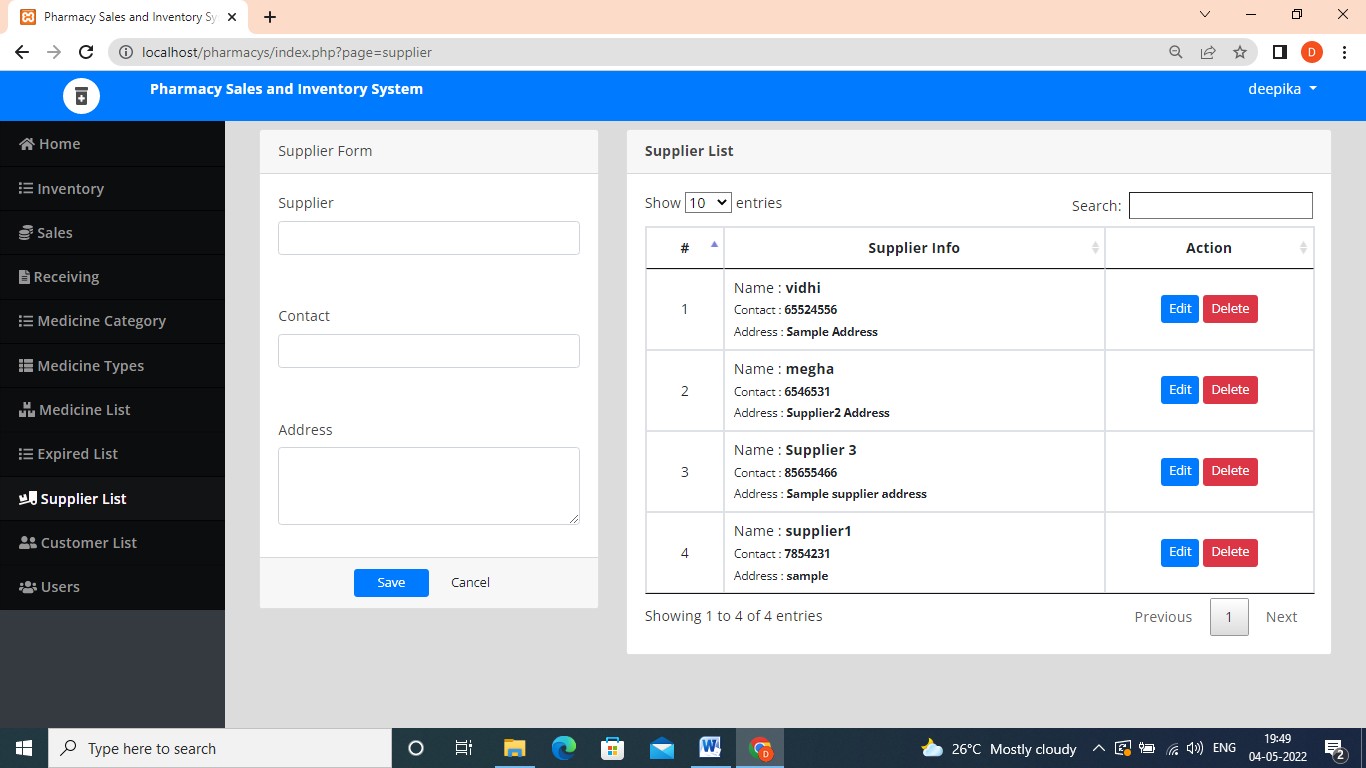


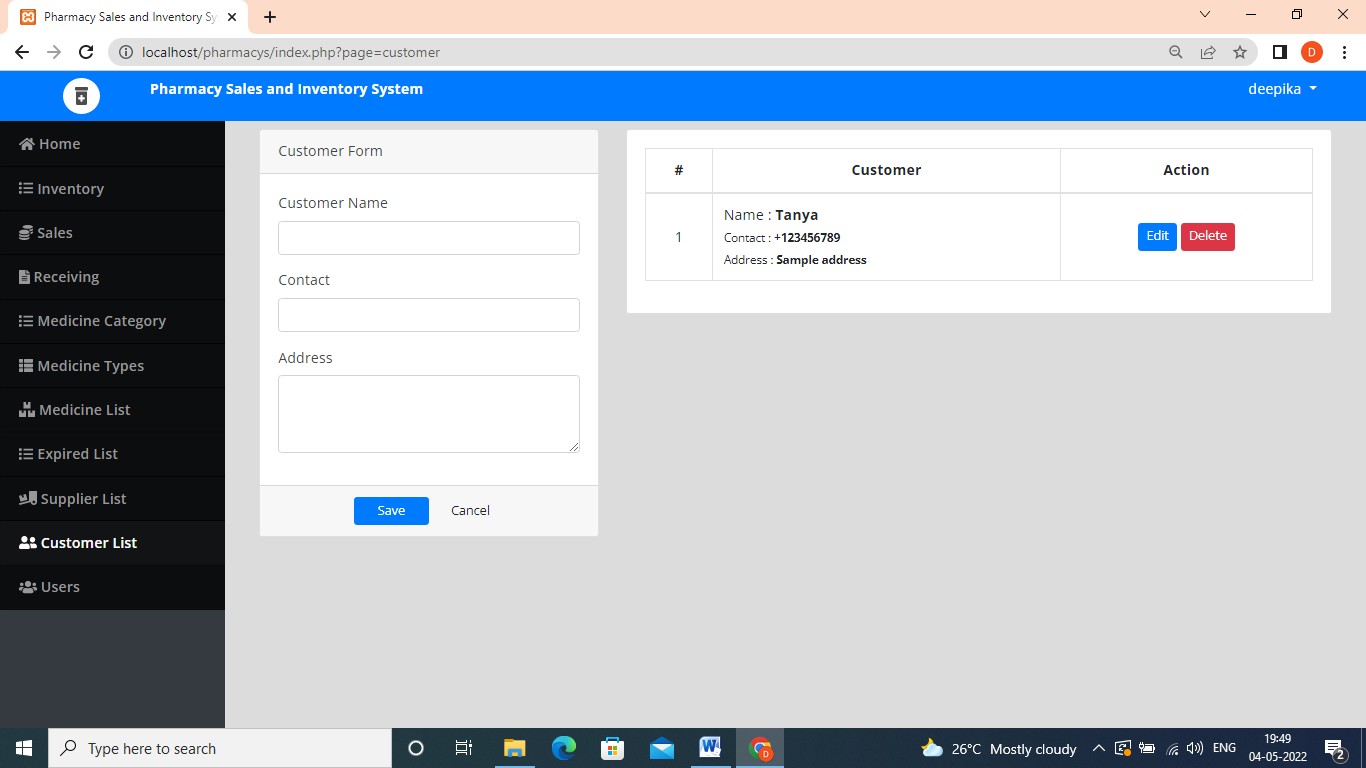


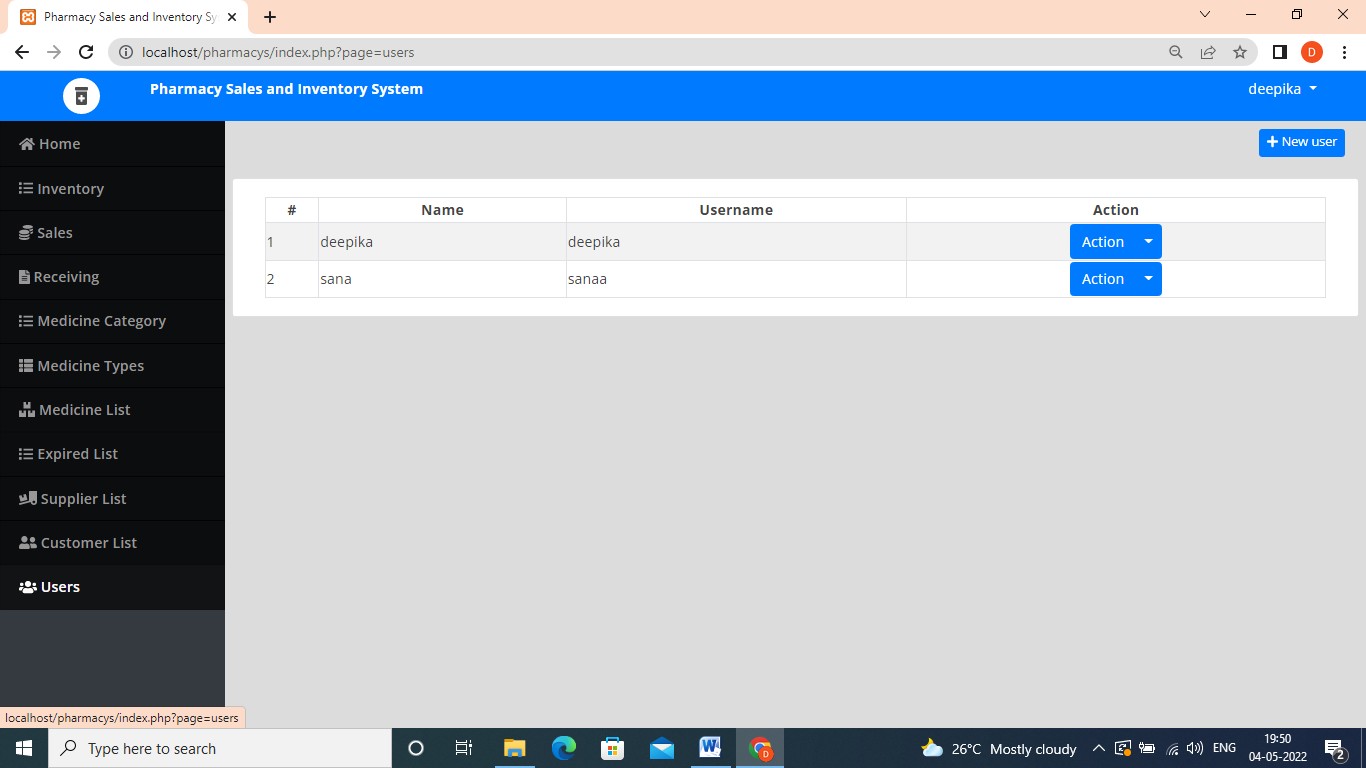


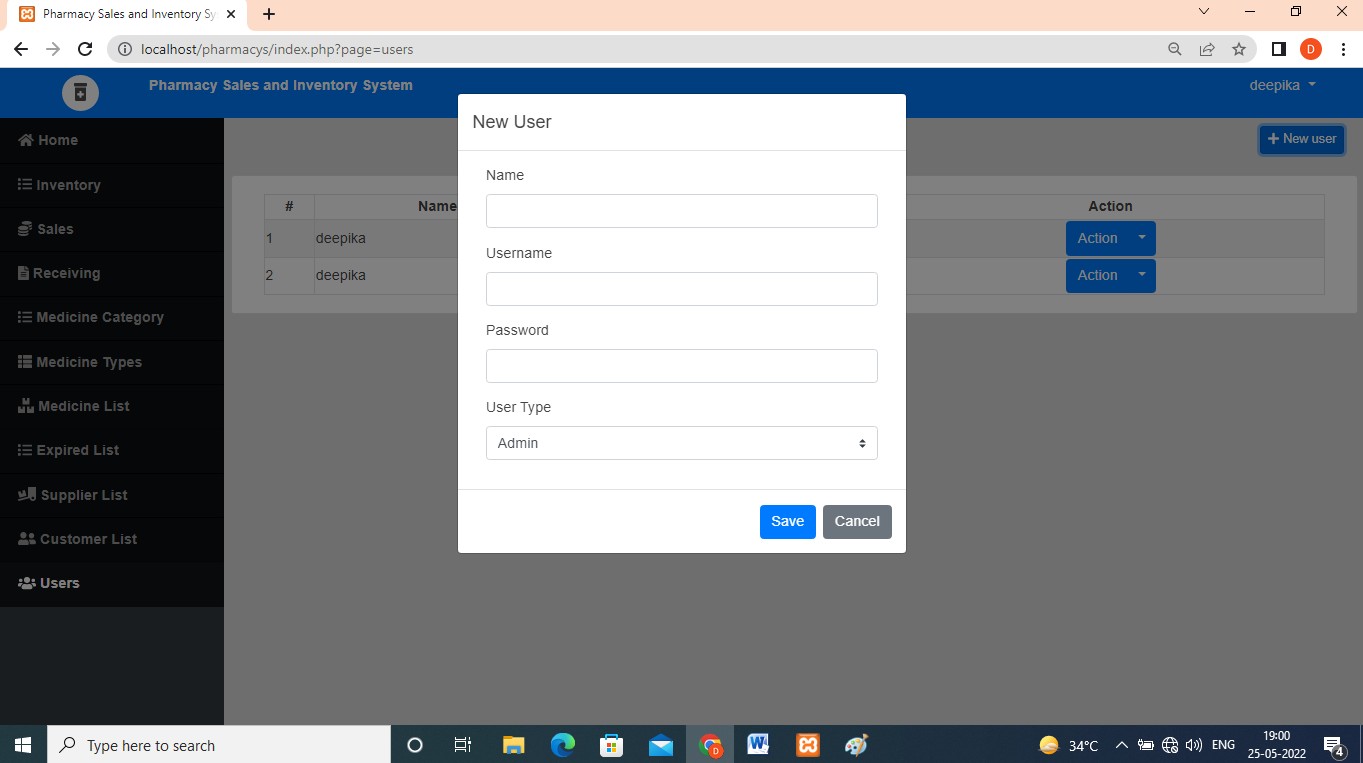


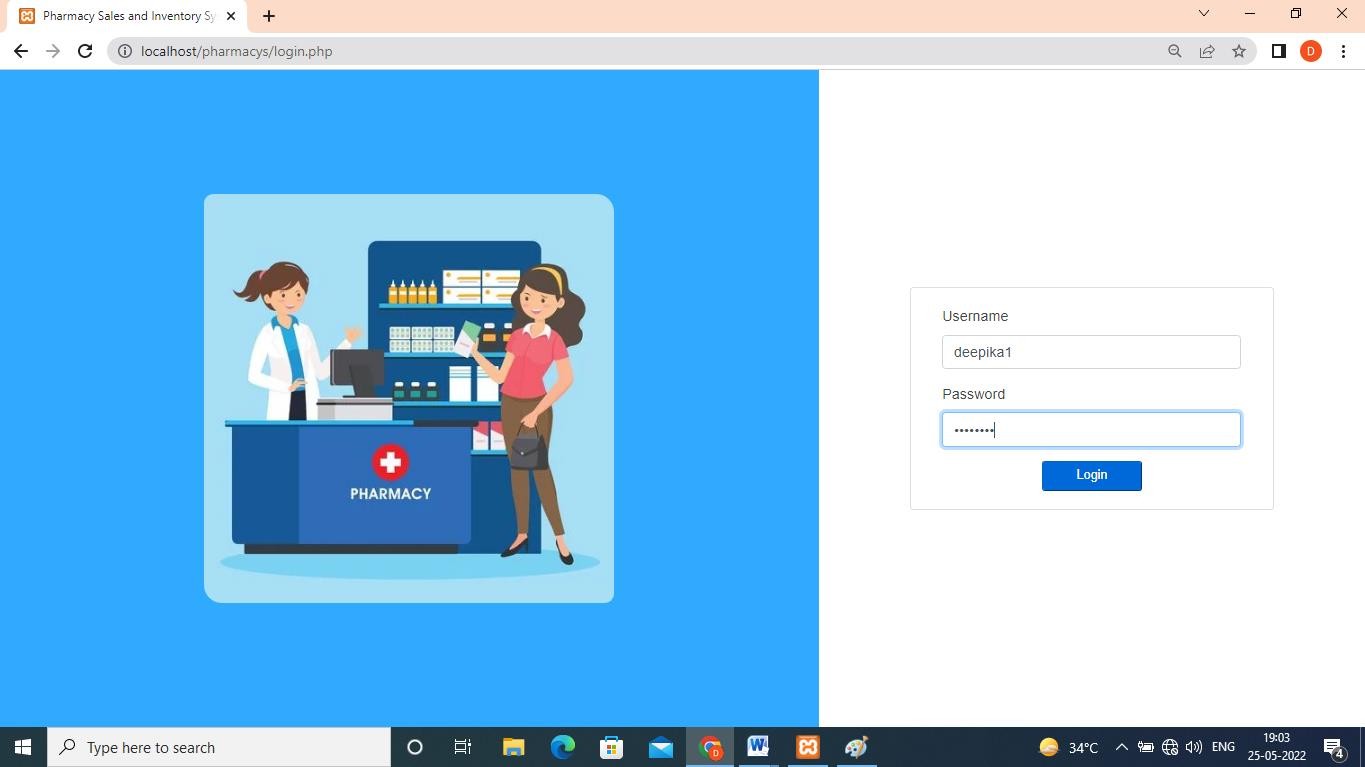


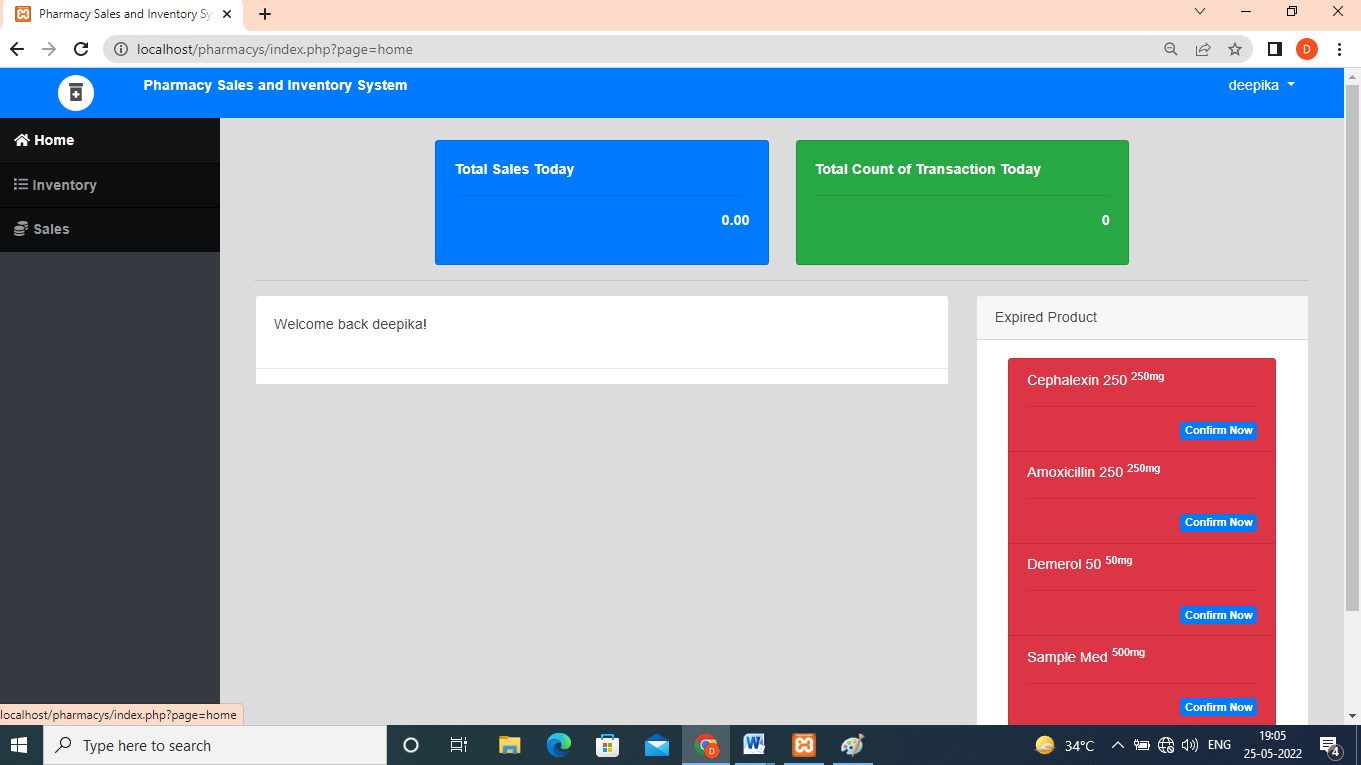


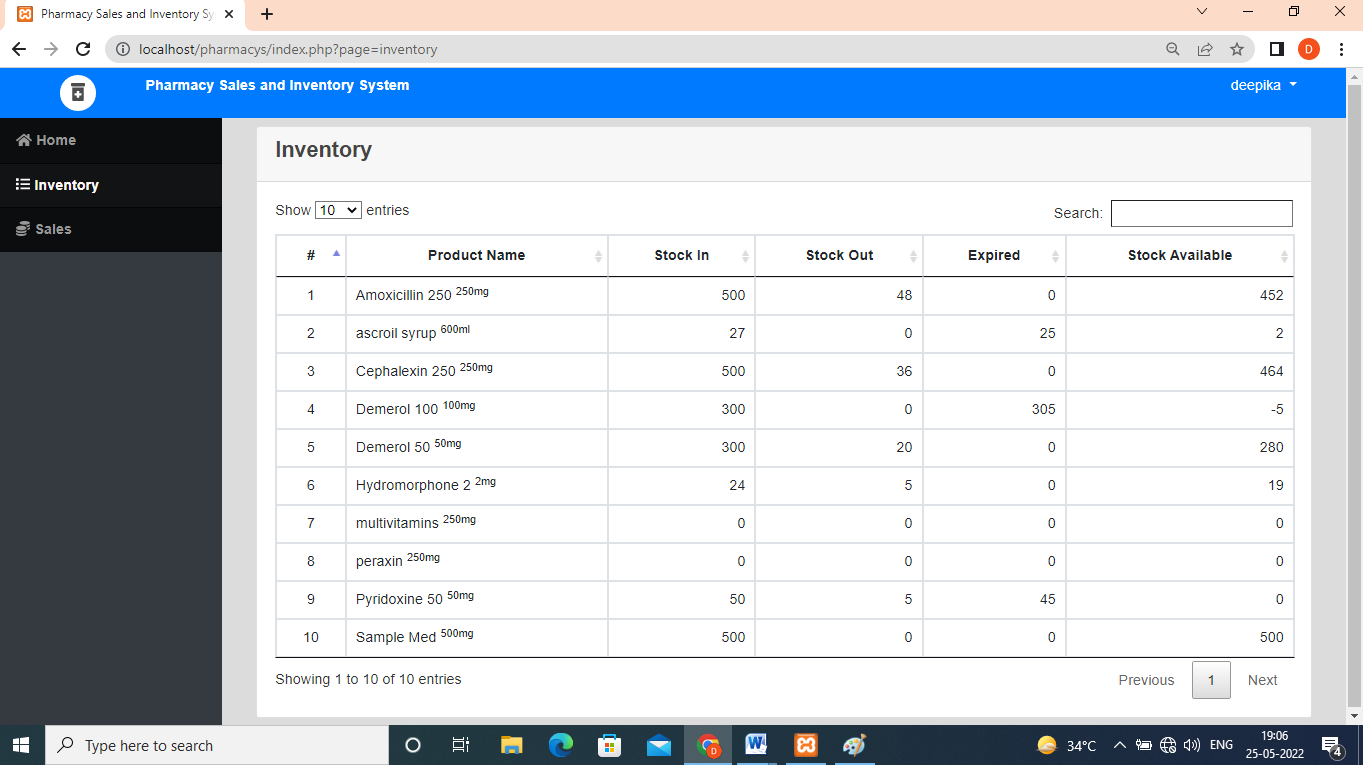


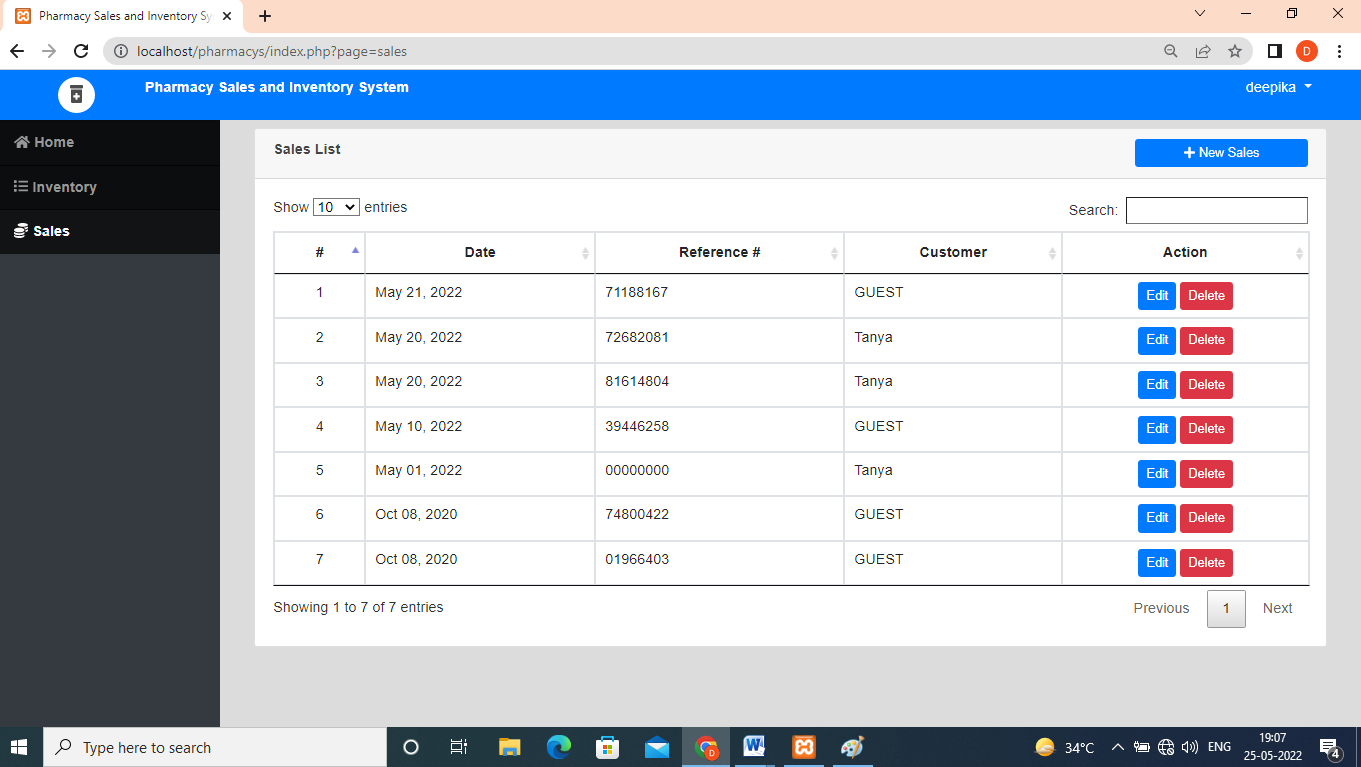












**TESTING**

**Software testing** is an investigation conducted to provide stakeholders with information about the quality of the software product or service under test. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risk of software implementation. Test techniques include the process of executing a program or application with intent of finding software bugs (errors or other defects), and verifying that the software product is fit for use.

software testing involves the execution of software component or system component to evaluate one or more properties of interest. In general, these properties indicate the extent to which the component or system under test :

* + - Meets the requirement that guided its design and development,
    - responds correctly to all kinds of inputs,
    - performs its functions within an acceptable time,
    - it is sufficiently usable,
    - can be installed and run in its intended environments, and
    - achieves the general result its stakeholders desire.

As the number of possible tests for even simple software components is practically infinite, all software testing uses some strategy to select tests that are feasible for the available time and resources. As a result, software testing typically (but not exclusively) attempts to execute a program or application with the intent of finding software bugs (error or other defects). The job of testing is an iterative process as when one bug is fixed, it can illuminate other, deeper bugs, or can even create new ones.

Software testing can provide objective, independent information about the quality of software and risk of its failure to users or sponsors.

Software testing can be conducted as soon as executable software (even if partially complete) exists. The overall approach to software development often determines when and how testing is conducted. For example, in a phased process, most testing occurs after system requirements have been defined and then implemented in testable programs.

#### Testing approach

##### Static, dynamic and passive testing

There are many approaches available in software testing. Reviews, walkthroughs or inspections are referred to as static testing whereas executing programmed code with a

given set of test cases is referred to as dynamic testing.

Static testing is often implicit, like Proof-reading, plus when programming tools/text editors check source code structure or compilers (pre-compilers) check Syntax and data flow as static program analysis. Dynamic testing takes place when the program itself is run. Dynamic testing may begin before the program is 100% complete in order to test particular sections of code and are applied to discrete functions or modules. Typical techniques for these are either using stubs/drivers or execution from a debugger environment.

Static testing involves verification, whereas dynamic testing involves validation.

Passive testing means verifying the system behavior without any interaction with the software product. Contrary to active testing, testers do not provide any test data but look at system logs and traces. They mind for patterns and specific behavior in order to make some kind of decisions. This is related to offline runtime verification and log analysis.

##### Exploratory approach

Exploratory testing is an approach to software testing that is concisely described as simultaneous learning, test design and test execution. Cem Kaner, who coined the term in 1984, defines exploratory testing as "a style of software testing that emphasizes the personal freedom and responsibility of the individual tester to continually optimize the quality of his/her work by treating test-related learning, test design, test execution, and test result interpretation as mutually supportive activities that run in parallel throughout the project."

##### The "box" approach

Software testing methods are traditionally divided into white and black-box testing. These two approaches are used to describe the point of view that a test engineer takes when designing test cases. A hybrid approach called grey-box testing may also be applied to software testing methodology. With the concept of grey-box testing which develops tests from specific design elements gaining prominence, this "arbitrary distinction" between black and white-box testing has faded somewhat.

##### White-box testing

White-box testing (also known as clear box testing, glass box testing, transparent box testing, and structural testing) tests internal structures or workings of a program, as opposed to the functionality exposed to the end-user. In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs. This is analogous to testing nodes in a circuit, e.g. in-circuit testing (ICT).

While white-box testing can be applied at the unit, integration and system levels of the software testing process, it is usually done at the unit level. It can test paths within a unit, paths between units during integration, and between subsystems during a system– level test. Though this method of test design can uncover many errors or problems, it might not detect unimplemented parts of the specification or missing requirements.

Techniques used in white-box testing include:

* + - API testing (application programming interface) – testing of the application using public and private APIs (application programming interfaces).
    - Code coverage – creating tests to satisfy some criteria of code coverage (e.g., the test designer can create tests to cause all statements in the program to be executed at least once).
    - Fault injection methods – intentionally introducing faults to gauge the efficacy of testing strategies.
    - Mutation testing methods.
    - static testing methods.

Code coverage tools can evaluate the completeness of a test suite that was created with any method, including black-box testing. This allows the software team to examine parts of a system that are rarely tested and ensures that the most important function points have been tested. Code coverage as a software metric can be reported as a percentage for:

* + - Function coverage, which reports on functions executed.
    - Statement coverage, which reports on the number of lines executed to complete the test.
    - Decision coverage, which reports on the whether both the true and false branch of a given test has been executed.

100% statement coverage ensures that all code paths, or branches (in terms of control flow) are executed at least once. This is helpful in ensuring correct functionality, but not sufficient since the same code may process different inputs correctly or incorrectly.

Pseudo-tested functions and methods are those that are covered but not specified (it is possible to remove their body without breaking any test case).

#### Black-box testing



Black-box diagram

Black-box testing (also known as functional testing) treats the software as a "black box," examining functionality without any knowledge of internal implementation, without seeing the source code. The testers are only aware of what the software is supposed to do, not how it does it. Black-box testing methods include: equivalence partitioning,

boundary value analysis, all-pairs testing, state transition tables, decision table testing, fuzz testing, model-based testing, use case testing, exploratory testing, and specification-based testing.

Specification-based testing aims to test the functionality of software according to the applicable requirements. This level of testing usually requires thorough test cases to be provided to the tester, who then can simply verify that for a given input, the output value (or behavior), either "is" or "is not" the same as the expected value specified in the test case. Test cases are built around specifications and requirements, i.e., what the application is supposed to do. It uses external descriptions of the software, including specifications, requirements, and designs to derive test cases. These tests can be functional or non-functional, though usually functional.

Specification-based testing may be necessary to assure correct functionality, but it is insufficient to guard against complex or high-risk situations.

One advantage of the black box technique is that no programming knowledge is required. Whatever biases the programmers may have had, the tester likely has a different set and may emphasize different areas of functionality. On the other hand, black-box testing has been said to be "like a walk in a dark labyrinth without a flashlight. "Because they do not examine the source code, there are situations when a tester writes many test cases to check something that could have been tested by only one test case or leaves some parts of the program untested.

#### Component interface testing

Component interface testing is a variation of black-box testing, with the focus on the data values beyond just the related actions of a subsystem component. The practice of component interface testing can be used to check the handling of data passed between various units, or subsystem components, beyond full integration testing between those units. The data being passed can be considered as "message packets" and the range or data types can be checked, for data generated from one unit, and tested for validity before being passed into another unit. One option for interface testing is to keep a separate log file of data items being passed, often with a timestamp logged to allow analysis of thousands of cases of data passed between units for days or weeks. Tests can include checking the handling of some extreme data values while other interface variables are passed as normal values. Unusual data values in an interface can help explain unexpected performance in the next unit.

#### Visual testing

The aim of visual testing is to provide developers with the ability to examine what was happening at the point of software failure by presenting the data in such a way that the developer can easily find the information she or he requires, and the information is expressed clearly.

At the core of visual testing is the idea that showing someone a problem (or a test

failure), rather than just describing it, greatly increases clarity and understanding. Visual testing, therefore, requires the recording of the entire test process – capturing everything that occurs on the test system in video format. Output videos are supplemented by real-time tester input via picture-in-a-picture webcam and audio commentary from microphones.

Visual testing provides a number of advantages. The quality of communication is increased drastically because testers can show the problem (and the events leading up to it) to the developer as opposed to just describing it and the need to replicate test failures will cease to exist in many cases. The developer will have all the evidence she or he requires of a test failure and can instead focus on the cause of the fault and how it should be fixed.

Ad hoc testing and exploratory testing are important methodologies for checking software integrity, because they require less preparation time to implement, while the important bugs can be found quickly. In ad hoc testing, where testing takes place in an improvised, impromptu way, the ability of the tester(s) to base testing off documented methods and then improvise variations of those tests can result in more rigorous examination of defect fixes. However, unless strict documentation of the procedures are maintained, one of the limits of ad hoc testing is lack of repeatability.

#### Grey-box testing

Grey-box testing (American spelling: gray-box testing) involves having knowledge of internal data structures and algorithms for purposes of designing tests while executing those tests at the user, or black-box level. The tester will often have access to both "the source code and the executable binary." Grey-box testing may also include reverse engineering (using dynamic code analysis) to determine, for instance, boundary values or error messages. Manipulating input data and formatting output do not qualify as grey-box, as the input and output are clearly outside of the "black box" that we are

calling the system under test. This distinction is particularly important when conducting integration testing between two modules of code written by two different developers, where only the interfaces are exposed for the test.

By knowing the underlying concepts of how the software works, the tester makes better-informed testing choices while testing the software from outside. Typically, a grey-box tester will be permitted to set up an isolated testing environment with activities such as seeding a database. The tester can observe the state of the product being tested after performing certain actions such as executing SQL statements against the database and then executing queries to ensure that the expected changes have been reflected. Grey-box testing implements intelligent test scenarios, based on limited information. This will particularly apply to data type handling, exception handling, and so on.

**Testing levels**

Broadly speaking, there are at least three levels of testing: unit testing, integration testing, and system testing. However, a fourth level, acceptance testing, may be included by developers. This may be in the form of operational acceptance testing or be simple end-user (beta) testing, testing to ensure the software meets functional expectations.

Tests are frequently grouped into one of these levels by where they are added in the software development process, or by the level of specificity of the test.

#### Unit testing

Unit testing refers to tests that verify the functionality of a specific section of code, usually at the function level. In an object-oriented environment, this is usually at the class level, and the minimal unit tests include the constructors and destructors.

These types of tests are usually written by developers as they work on code (white-box style), to ensure that the specific function is working as expected. One function might have multiple tests, to catch corner cases or other branches in the code. Unit testing alone cannot verify the functionality of a piece of software, but rather is used to ensure that the building blocks of the software work independently from each other.

Unit testing is a software development process that involves a synchronized application of a broad spectrum of defect prevention and detection strategies in order to reduce software development risks, time, and costs. It is performed by the software developer or engineer during the construction phase of the software development life cycle. Unit testing aims to eliminate construction errors before code is promoted to additional testing; this strategy is intended to increase the quality of the resulting software as well as the efficiency of the overall development process.

Depending on the organization's expectations for software development, unit testing might include static code analysis, data-flow analysis, metrics analysis, peer code reviews, code coverage analysis and other software testing practices.

#### Integration testing

Integration testing is any type of software testing that seeks to verify the interfaces between components against a software design. Software components may be integrated in an iterative way or all together ("big bang"). Normally the former is considered a better practice since it allows interface issues to be located more quickly and fixed.

Integration testing works to expose defects in the interfaces and interaction between integrated components (modules). Progressively larger groups of tested software components corresponding to elements of the architectural design are integrated and tested until the software works as a system.

Integration tests usually involve a lot of code, and produce traces that are larger than those produced by unit tests. This has an impact on the ease of localizing the fault when an integration test fails. To overcome this issue, it has been proposed to automatically cut the large tests in smaller pieces to improve fault localization.

#### System testing

System testing tests a completely integrated system to verify that the system meets its requirements. For example, a system test might involve testing a login interface, then creating and editing an entry, plus sending or printing results, followed by summary processing or deletion (or archiving) of entries, then logoff.

#### Operational acceptance testing

Operational acceptance is used to conduct operational readiness (pre-release) of a product, service or system as part of a quality management system. OAT is a common type of non-functional software testing, used mainly in software development and software maintenance projects. This type of testing focuses on the operational readiness of the system to be supported, or to become part of the production environment. Hence, it is also known as operational readiness testing (ORT) or Operations readiness and assurance (OR&A) testing. Functional testing within OAT is limited to those tests that are required to verify the non-functional aspects of the system.

In addition, the software testing should ensure that the portability of the system, as well as working as expected, does not also damage or partially corrupt its operating environment or cause other processes within that environment to become inoperative.

#### Installation testing

Most software systems have installation procedures that are needed before they can be used for their main purpose. Testing these procedures to achieve an installed software system that may be used is known as installation testing.

#### Compatibility testing

A common cause of software failure (real or perceived) is a lack of its compatibility with other application software, operating systems (or operating system versions, old or new), or target environments that differ greatly from the original (such as a terminal or GUI application intended to be run on the desktop now being required to become a Web application, which must render in a Web browser). For example, in the case of a lack of backward compatibility, this can occur because the programmers develop and test software only on the latest version of the target environment, which not all users may be running. This results in the unintended consequence that the latest work may not function on earlier versions of the target environment, or on older hardware that earlier versions of the target environment were capable of using. Sometimes such issues can be fixed by proactively abstracting operating system functionality into a separate program

module or library.

#### Smoke and sanity testing

Sanity testing determines whether it is reasonable to proceed with further testing. Smoke testing consists of minimal attempts to operate the software, designed to determine whether there are any basic problems that will prevent it from working at all. Such tests can be used as build verification test.

#### Regression testing

Regression testing focuses on finding defects after a major code change has occurred. Specifically, it seeks to uncover software regressions, as degraded or lost features, including old bugs that have come back. Such regressions occur whenever software functionality that was previously working correctly, stops working as intended.

Typically, regressions occur as an unintended consequence of program changes, when the newly developed part of the software collides with the previously existing code.

Regression testing is typically the largest test effort in commercial software development, due to checking numerous details in prior software features, and even new software can be developed while using some old test cases to test parts of the new design to ensure prior functionality is still supported.

Common methods of regression testing include re-running previous sets of test cases and checking whether previously fixed faults have re-emerged. The depth of testing depends on the phase in the release process and the risk of the added features. They can either be complete, for changes added late in the release or deemed to be risky, or be very shallow, consisting of positive tests on each feature, if the changes are early in the release or deemed to be of low risk. In regression testing, it is important to have strong assertions on the existing behavior. For this, it is possible to generate and add new assertions in existing test cases, this is known as automatic test amplification.

#### Acceptance testing

Acceptance testing can mean one of two things:

1. A smoke test is used as a build acceptance test prior to further testing, e.g., before integration or regression.
2. Acceptance testing performed by the customer, often in their lab environment on their own hardware, is known as user acceptance testing (UAT). Acceptance testing may be performed as part of the hand-off process between any two phases of development.

#### Alpha testing

Alpha testing is simulated or actual operational testing by potential users/customers or an independent test team at the developers' site. Alpha testing is often employed for off-

the-shelf software as a form of internal acceptance testing before the software goes to beta testing.

#### Beta testing

Beta testing comes after alpha testing and can be considered a form of external user acceptance testing. Versions of the software, known as beta versions, are released to a limited audience outside of the programming team known as beta testers. The software is released to groups of people so that further testing can ensure the product has few faults or bugs. Beta versions can be made available to the open public to increase the feedback field to a maximal number of future users and to deliver value earlier, for an extended or even indefinite period of time (perpetual beta).

#### Functional Vs non-functional testing

Functional testing refers to activities that verify a specific action or function of the code. These are usually found in the code requirements documentation, although some development methodologies work from use cases or user stories. Functional tests tend to answer the question of "can the user do this" or "does this particular feature work."

Non-functional testing refers to aspects of the software that may not be related to a specific function or user action, such as scalability or other performance, behavior under certain constraints, or security. Testing will determine the breaking point, the point at which extremes of scalability or performance leads to unstable execution. Non- functional requirements tend to be those that reflect the quality of the product, particularly in the context of the suitability perspective of its users.

#### Continuous testing

Continuous testing is the process of executing automated tests as part of the software delivery pipeline to obtain immediate feedback on the business risks associated with a software release candidate. Continuous testing includes the validation of both functional requirements and non-functional requirements; the scope of testing extends from validating bottom-up requirements or user stories to assessing the system requirements associated with overarching business goals.

#### Destructive testing

Destructive testing attempts to cause the software or a sub-system to fail. It verifies that the software functions properly even when it receives invalid or unexpected inputs, thereby establishing the robustness of input validation and error-management routines. Software fault injection, in the form of fuzzing, is an example of failure testing. Various commercial non-functional testing tools are linked from the software fault injection page; there are also numerous open-source and free software tools available that perform destructive testing.

#### Software performance testing

Performance testing is generally executed to determine how a system or sub-system performs in terms of responsiveness and stability under a particular workload. It can also serve to investigate, measure, validate or verify other quality attributes of the system, such as scalability, reliability and resource usage.

Load testing is primarily concerned with testing that the system can continue to operate under a specific load, whether that be large quantities of data or a large number of users. This is generally referred to as software scalability. The related load testing activity of when performed as a non-functional activity is often referred to as endurance testing. Volume testing is a way to test software functions even when certain components (for example a file or database) increase radically in size. Stress testing is a way to test reliability under unexpected or rare workloads. Stability testing (often referred to as load or endurance testing) checks to see if the software can continuously function well in or above an acceptable period.

There is little agreement on what the specific goals of performance testing are. The terms load testing, performance testing, scalability testing, and volume testing, are often used interchangeably.

Real-time software systems have strict timing constraints. To test if timing constraints are met, real-time testing is used.

#### Usability testing

Usability testing is to check if the user interface is easy to use and understand. It is concerned mainly with the use of the application. This is not a kind of testing that can be automated; actual human users are needed, being monitored by skilled UI designers.

#### Accessibility testing

Accessibility testing may include compliance with standards such as:

* Americans with Disabilities Act of 1990
* Section 508 Amendment to the Rehabilitation Act of 1973
* Web Accessibility Initiative (WAI) of the World Wide Web Consortium (W3C).

#### Security testing

Security testing is essential for software that processes confidential data to prevent system intrusion by hackers.

The International Organization for Standardization (ISO) defines this as a "type of testing conducted to evaluate the degree to which a test item, and associated data and information, are protected so that unauthorised persons or systems cannot use, read or modify them, and authorized persons or systems are not denied access to them."

#### Internationalization and localization

Testing for internationalization and localization validates that the software can be used with different languages and geographic regions. The process of pseudo localization is used to test the ability of an application to be translated to another language, and make it easier to identify when the localization process may introduce new bugs into the product.

Globalization testing verifies that the software is adapted for a new culture (such as different currencies or time zones).

Actual translation to human languages must be tested, too. Possible localization and globalization failures include:

* Software is often localized by translating a list of strings out of context, and the translator may choose the wrong translation for an ambiguous source string.
* Technical terminology may become inconsistent, if the project is translated by several people without proper coordination or if the translator is imprudent.
* Literal word-for-word translations may sound inappropriate, artificial or too technical in the target language.
* Untranslated messages in the original language may be left hard coded in the source code.
* Some messages may be created automatically at run time and the resulting string may be ungrammatical, functionally incorrect, misleading or confusing.
* Software may use a keyboard shortcut that has no function on the source language's keyboard layout, but is used for typing characters in the layout of the target language.
* Software may lack support for the character encoding of the target language.
* Fonts and font sizes that are appropriate in the source language may be inappropriate in the target language; for example, CJK characters may become unreadable, if the font is too small.
* A string in the target language may be longer than the software can handle. This may make the string partly invisible to the user or cause the software to crash or malfunction.
* Software may lack proper support for reading or writing bi-directional text.
* Software may display images with text that was not localized.
* Localized operating systems may have differently named system configuration files and environment variables and different formats for date and currency.

#### Development testing

Development Testing is a software development process that involves the synchronized application of a broad spectrum of defect prevention and detection strategies in order to reduce software development risks, time, and costs. It is performed by the software developer or engineer during the construction phase of the software development lifecycle. Development Testing aims to eliminate construction errors before code is promoted to other testing; this strategy is intended to increase the quality of the resulting software as well as the efficiency of the overall development process.

Depending on the organization's expectations for software development, Development Testing might include static code analysis, data flow analysis, metrics analysis, peer code reviews, unit testing, code coverage analysis, traceability, and other software testing practices.

#### A/B testing

A/B testing is a method of running a controlled experiment to determine if a proposed change is more effective than the current approach. Customers are routed to either a current version (control) of a feature, or to a modified version (treatment) and data is collected to determine which version is better at achieving the desired outcome.

#### Concurrent testing

Concurrent or concurrency testing assesses the behavior and performance of software and systems that use concurrent computing, generally under normal usage conditions. Typical problems this type of testing will expose are deadlocks, race conditions and problems with shared memory/resource handling.

#### Conformance testing or type testing

In software testing, conformance testing verifies that a product performs according to its specified standards. Compilers, for instance, are extensively tested to determine whether they meet the recognized standard for that language.

# IMPLEMENTATION & MAINTAINANCE

#### MAINTENANCE

Software Maintenance is the process of modifying a software product after it has been delivered to the customer. The main purpose of software maintenance is to modify and update software applications after delivery to correct faults and to improve performance.

#### Need for Maintenance –

Software Maintenance must be performed in order to:

* Correct faults.
* Improve the design.
* Implement enhancements.
* Interface with other systems.
* Accommodate programs so that different hardware, software, system features, and telecommunications facilities can be used.
* Migrate legacy software.
* Retire software.

##### TYPES OF MAINTENANCE

1. **Corrective maintenance:**

Corrective maintenance of a software product may be essential either to rectify some bugs observed while the system is in use, or to enhance the performance of the system.

##### Adaptive maintenance:

This includes modifications and updations when the customers need the product to run on new platforms, on new operating systems, or when they need the product to interface with new hardware and software.

##### Perfective maintenance:

A software product needs maintenance to support the new features that the users want or to change different types of functionalities of the system according to the customer demands.

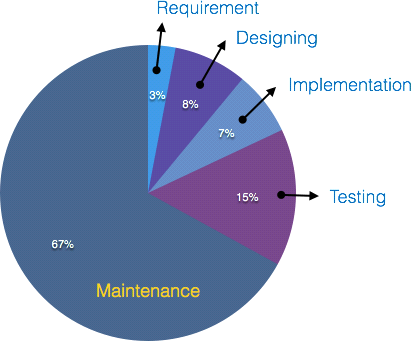
##### Preventive maintenance:

This type of maintenance includes modifications and updations to prevent future problems of the software. It goals to attend problems, which are not significant at this moment but may cause serious issues in future.

##### COST OF MAINTENANCE

Reports suggest that the cost of maintenance is high. A study on estimating software maintenance found that the cost of maintenance is as high as 67% of the cost of entire

software process cycle.



On an average, the cost of software maintenance is more than 50% of all SDLC phases. There are various factors, which trigger maintenance cost go high, such as:

Real-world factors affecting maintenance cost

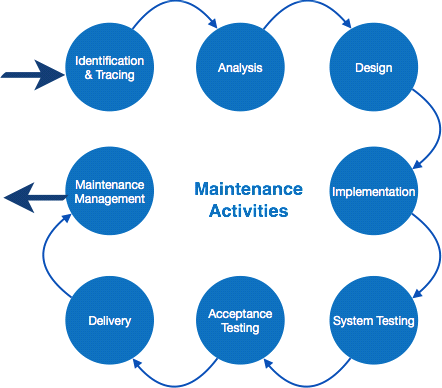
* The standard age of any software is considered up to 10 to 15 years.
* Older software’s, which were meant to work on slow machines with less memory and storage capacity cannot keep themselves challenging against newly coming

enhanced software’s on modern hardware.

* As technology advances, it becomes costly to maintain old software.
* Most maintenance engineers are newbie and use trial and error method to rectify problem.
* Often, changes made can easily hurt the original structure of the software, making it hard for any subsequent changes.
* Changes are often left undocumented which may cause more conflicts in future. Software-end factors affecting maintenance cost
* Structure of Software Program
* Programming Language
* Dependence on external environment
* Staff reliability and availability

##### Maintenance Activities

IEEE provides a framework for sequential maintenance process activities. It can be used in iterative manner and can be extended so that customized items and processes can be included.



These activities go hand-in-hand with each of the following phase:

* **Identification & Tracing** - It involves activities pertaining to identification of requirement of modification or maintenance. It is generated by user or system may itself report via logs or error messages. Here, the maintenance type is classified also.
* **Analysis** - The modification is analyzed for its impact on the system including safety and security implications. If probable impact is severe, alternative solution is looked for. A set of required modifications is then materialized into requirement specifications. The cost of modification/maintenance is analyzed and estimation is concluded.
* **Design** - New modules, which need to be replaced or modified, are designed against requirement specifications set in the previous stage. Test cases are created for validation and verification.
* **Implementation** - The new modules are coded with the help of structured design created in the design step. Every programmer is expected to do unit testing in parallel.
* **System Testing** - Integration testing is done among newly created modules. Integration testing is also carried out between new modules and the system. Finally the system is tested as a whole, following regressive testing procedures.
* **Acceptance Testing** - After testing the system internally, it is tested for acceptance with the help of users. If at this state, user complaints some issues they are addressed or noted to address in next iteration.
* **Delivery** - After acceptance test, the system is deployed all over the organization either by small update package or fresh installation of the system. The final testing takes place at client end after the software is delivered.

Training facility is provided if required, in addition to the hard copy of user manual.

* **Maintenance management** - Configuration management is an essential part of system maintenance. It is aided with version control tools to control versions, semi-version or patch management.

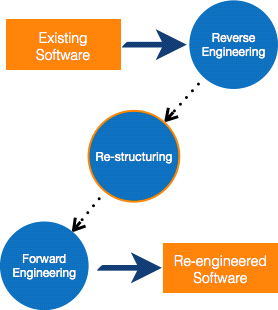
#### Software Re-engineering

When we need to update the software to keep it to the current market, without impacting its functionality, it is called software re-engineering. It is a thorough process where the design of software is changed and programs are re-written.

Legacy software cannot keep tuning with the latest technology available in the market. As the hardware become obsolete, updating of software becomes a headache. Even if software grows old with time, its functionality does not.

For example, initially Unix was developed in assembly language. When language C came into existence, Unix was re-engineered in C, because working in assembly language was difficult.

Other than this, sometimes programmers notice that few parts of software need more maintenance than others and they also need re-engineering.



##### RE-ENGINEERING PROCESS

* **Decide** what to re-engineer. Is it whole software or a part of it?
* **Perform** Reverse Engineering, in order to obtain specifications of existing software.
* **Restructure Program** if required. For example, changing function-oriented

programs into object-oriented programs.

* **Re-structure data** as required.
* **Apply Forward engineering** concepts in order to get re-engineered software. There are few important terms used in Software re-engineering

##### REVERSE ENGINEERING

It is a process to achieve system specification by thoroughly analyzing, understanding the existing system. This process can be seen as reverse SDLC model, i.e. we try to get higher abstraction level by analyzing lower abstraction levels.

An existing system is previously implemented design, about which we know nothing. Designers then do reverse engineering by looking at the code and try to get the design. With design in hand, they try to conclude the specifications. Thus, going in reverse from code to system specification.



##### PROGRAM RESTRUCTURING

It is a process to re-structure and re-construct the existing software. It is all about re- arranging the source code, either in same programming language or from one programming language to a different one. Restructuring can have either source code- restructuring and data-restructuring or both.

Re-structuring does not impact the functionality of the software but enhance reliability and maintainability. Program components, which cause errors very frequently can be changed, or updated with re-structuring.

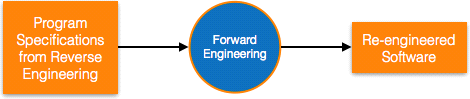
The dependability of software on obsolete hardware platform can be removed via re- structuring.

##### FORWARD ENGINEERING

Forward engineering is a process of obtaining desired software from the specifications in hand which were brought down by means of reverse engineering. It assumes that there was some software engineering already done in the past.

Forward engineering is same as software engineering process with only one difference

– it is carried out always after reverse engineering.

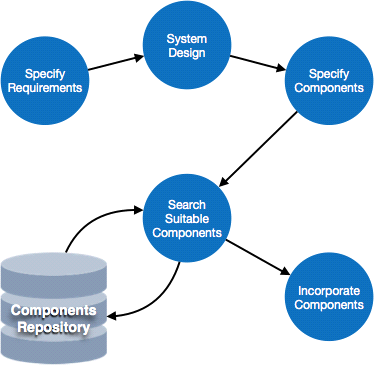


#### Component reusability

A component is a part of software program code, which executes an independent task in the system. It can be a small module or sub-system itself.

##### REUSE PROCESS

Two kinds of method can be adopted: either by keeping requirements same and adjusting components or by keeping components same and modifying requirements.



* **Requirement Specification** - The functional and non-functional requirements are specified, which a software product must comply to, with the help of existing system, user input or both.
* **Design** - This is also a standard SDLC process step, where requirements are defined in terms of software parlance. Basic architecture of system as a whole and its sub- systems are created.
* **Specify Components** - By studying the software design, the designers segregate the entire system into smaller components or sub-systems. One complete software design turns into a collection of a huge set of components working together.
* **Search Suitable Components** - The software component repository is referred by designers to search for the matching component, on the basis of functionality and intended software requirements..
* **Incorporate Components** - All matched components are packed together to shape them as complete software.

**REFERENCES**

* + System analysis and design by V.Rajaraman.
  + Software Engineering by K.K Aggrawal.
  + [https://www.google.com](https://www.google.com/)
  + [https://www.tutorialspoint.com](https://www.tutorialspoint.com/)
  + [https://www.geeksforgeeks.org](https://www.geeksforgeeks.org/)
  + [https://en.wikipedia.org](https://en.wikipedia.org/)