SYNOPSIS

INDEX

TITLE OF THE PROJECT

EMOTIONS

(Art & Craft Website)

Chapter 1

INTRODUCTION

1.1 Description of the System

About the project

The main aspect of the website for anart and craft business is covered in this project. The project reflects the overall growth of the website on the basis of feedbacks to the organization. The administrator has full access privilege in the project. After logging into the software he decides whether to update and make changes or not. Accordingly registrations are confirmed to the users. The website owners have permissions to add new information, login window, registration forms in this system. The total data is with the administrator. Any wrong entry can be updated but it can be done only by the administrator. Any person can see the data available related to their queries. Once logged out the Admin cannot enter the software without proper login. Unauthorized access to any user is not allowed. The user can only send request to the administrator for dealership but it will be accepted or denied by the administrator.

1.2 SYSTEM REQUIREMENT SPECIFICATION REPORT

1.2(a)Objective

The objective of preparing the software requirement specification is to represent the requirements of the software in such a manner that ultimately leads to successful software implementation. It is the result of the analysis process of the software development. It should contain all the data the software is going to process, the function it will provide, and the behavior it will exhibit.
This Software Requirements Specifications (SRS) is defined in IEEE Std. 830-1993, IEEE Recommended Practice for Software RequirementsSpecifications.

1.2(b)Overview

The software requirement specification (SRS) is very important part of the software building process, which describes the actual user level requirement from technical point of view i.e., what the user exactly wants or for what purpose we are making everything. The objective of preparing the software requirement specification is to represent the requirements of the software in such a manner that ultimately leads to successful software implementation. It is the result of the analysis process of the software development. It should contain all the data the software is going to process, the function it will provide, and the behavior it will exhibit. This Software Requirements Specifications (SRS) is defined in IEEE Std. 830-1993, IEEE Recommended Practice for Software Requirements Specifications.

1.2(c) Purpose

The purpose of the document is to collect and analyze all assorted ideas that have come up to define the system, its requirements with respect to consumers. Also, we shall predict and sort out how we hope this product will be used in order to gain a better understanding of the project, outline concepts that may be developed later, and document ideas that are being considered, but may be discarded as the product develops.

In short, the purpose of this SRS document is to provide a detailed overview of our software product, its parameters and goals. This document describes the project's target audience and its user interface, hardware and software requirements. It defines how our client, team and audience see the product and its functionality. Nonetheless, it helps any designer and developer to assist in software delivery lifecycle (SDLC) processes.

1.2(d) Scope

Primarily, the scope pertains to the EMOTIONS product features for sellingart and craft items and decorative products. It focuses on the company, the stakeholders and applications, which allow for online sales, distribution and marketing of products.

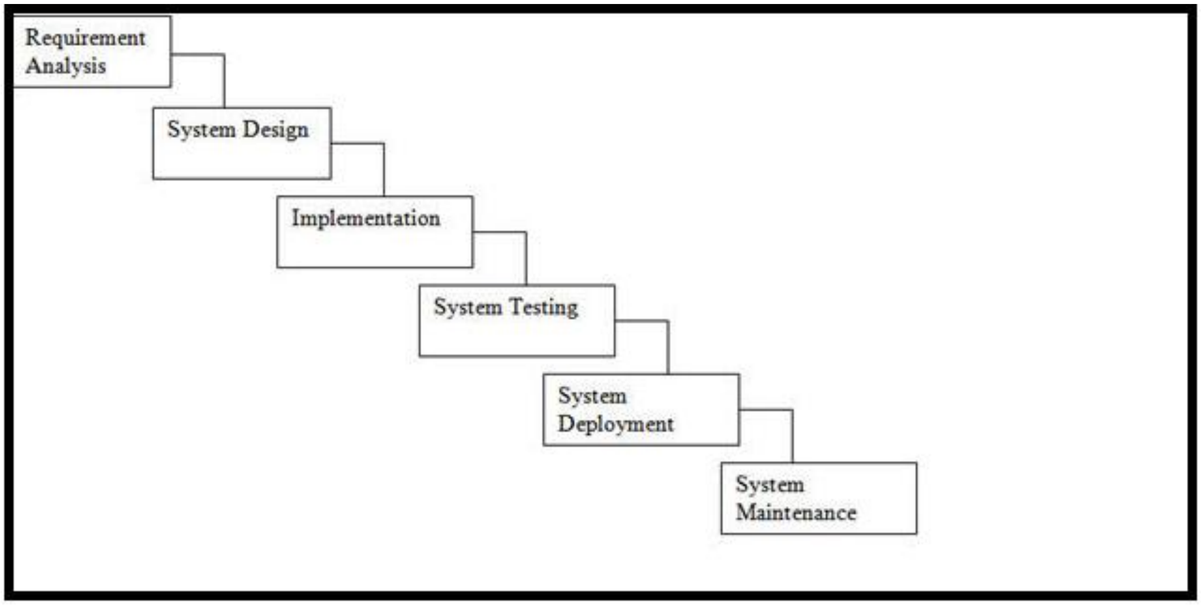
This SRS is also aimed at specifying requirements of software to be developed but it can also be applied to assist in the selection of in-house and commercial software products. The standard can be used to create software requirements specifications directly or can be used as a model for defining an organization or project specific standard. It does not identify any specific method, nomenclature or tool for preparing an SRS.

Chapter 2

SURVEY OF TECHNOLOGIES

The technology that has been used for carrying out this project is one of the mostly used techniques of System Development Life Cycle (SDLC) process i.e., the Waterfall Model.

2.1 WATERFALL MODEL



- Waterfall Model is the simplest model with each phase having a well-defined starting and ending point, with identifiable deliveries to the next phase.
- The model consists of six distinct stages, namely:

  - **Requirement Analysis Phase**
  - **Specification Phase**
  - **System and Software design Phase**
  - **Implementation and Testing Phase**
  - **Integration and System Testing Phase**
  - **Maintenance Phase**

2.1(a) <u>Reasons For Choosing Waterfall Model:</u>

- ➢ In this model, a detailed checking is done at each and every step by software quality assurance (SQA) group and also by the clients. That is why it involves less risk of rejection by clients.
- ➢ It is very easy to understand and implement.
- ➢ It is documentation driven, that is, documentation is produced at every stage.
- ➢ Testing is inherent to every phase of the waterfall model. So there is no need for a separate testing phase.
- ➢ Good progress tracking due to clear development stages.
- ➢ Getting the requirements and design out of the way first also improves quality; it's much easier to catch and correct possible flaws at the design stage.

Finally, the first two phases end in the production of a formal specification, the waterfall model can aid efficient knowledge transfer when team members are dispersed in different locations.

✓ FEASIBILITY ANALYSIS

Feasibility study is the determination of whether or not a project is worth doing. The processfollowed in making this determination is called feasibility study. This type of study determines it can and should be taken more than once, it has been determined that the project is feasible; the analyst can go ahead and prepare the project specification, which finalize project necessary requirements.

The importance outcome of the proposed system is the determination whether the system requested is feasible or not. This feasibility has been checked by mock visitor by visiting the website in running mode.

- **TECHNICAL FEASIBILITY**
- **OPERATIONAL FEASIBILITY**
- **ECONOMIC FEASIBILITY**

Technical Feasibility:

This is concerned with specifying equipment and software that will successfully satisfy the user requirement; the technical needs of the system may vary considerably, but might include:

<u>The facility to produce outputs in a given time</u>:

- Response time under certain conditions.
- Ability to process a certain volume of transaction at a Particular speed.
- Facility to communicate data to distant location.

Operational Feasibility:

It 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?

Generally project will not be rejected simply because of operational infallibility but such considerations are likely to critically affect the nature and scope of the eventual recommendations.

Economic Feasibility:

Economic analysis is the most frequently used technique for evaluating the effectiveness of a proposed system. More frequently known as cost / benefit analysis; the procedure is to determine the benefits and saving 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. Otherwise, further justification or alternative in the proposed system will have to be made if it is to have a change of being approved. This is an ongoing effort that improves in accuracy at each phase of the system life cycle**.**

Chapter 3

REQUIREMENTS AND ANALYSIS

3.1 Problem Definition

To develop a web based application to improve the service to the customers and company which in turn increases the sales and profit in “ EMOTIONS “.

The system will be capable of maintaining details of various customers, products and storing all the day to day transactions, such as generation of shipment address bills, handling customers, product receipts and updating of stores.

3.2 Requirement Specification

The requirement describes “what” of a system and not the “how” of a system.

Types of Requirements:

There are two types of requirements:

1. Functional requirements
2. Non-functional requirements

Functional Requirements

They are also called product features. Sometimes, functional requirements may also specify what the software should not do.

Functional Requirements should include:

- Descriptions of data to be entered into the system
- Descriptions of operations performed by each screen
- Descriptions of work-flows performed by the system
- Descriptions of system reports or other outputs
- Who can enter the data into the system?
- How the system meets applicable regulatory requirements.

The functional specification is designed to be read by a general audience. Readers should understand the system, but no particular technical knowledge should be required to understand the document.

Features may be additional functionality, or differ from the basic functionality along some quality attribute.

One strategy for quickly penetrating a market, is to produce the core, or stripped down, basic product, and adding features to variants of the product to be released shortly thereafter.

Non - Functional Requirements

In systems engineering and requirements engineering, a **non-functional requirement** is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors. This should be contrasted with functional requirements that define specific behavior or functions.

Security:-The Information should be Secure; there should not be any kind of malfunctioning. All the results, details of Products taken and Classes are stored securely in the system. System Information will not be changed by any person rather than the management. Application will allow only valid users to access the system. Access to any application resource will depend upon user’s designation. There are two types of users namely Administrator and User. Security is based upon the individual user ID and Password.

Reliability:- System should be reliable. It should keep secure all the information regarding to particular User, Products. It should work effectively in tremendous rush. The system must give the perfect calculation and perfect results in kind of damn situation. The Particular result must be listed in to the particular user only; there should not be any kind of data integrity or other problem between Administrator and Customer.

Flexibility:-System is working easily on the Intranet with the username and password of the user. The Institute has given the rights to the staff and the users to use the system with their username. The system can also work on other kind of technology with the little modification. System should be quite flexible to install and maintain.

Efficiency:- System should be efficient enough to meet all kinds of requirements as required by the Administrator and Customer. The system should not hang or lose its efficiency in any kind of worse conditions. It should provide the correct output in all manners.

User Friendliness:- System should be user friendly, so that any user can access the system.

Availability:-The order system being an online system should be available anytime.Through the system should be available 24*7 some features may be restricted.Administrator may allow the specific test to be available only at certain time like scheduled classes.

Usability:- The links are provided for each form. The user is facilitated to view and make entries in the forms. Validations are provided in each field to avoid inconsistent or invalid entry in the databases. Some forms consists Hyper Links, which provides further details. Reports screen contains text boxes and drop down lists, so that reports can be produced.

Maintainability:- The installation and operation manual of class management system will be provided to the user.

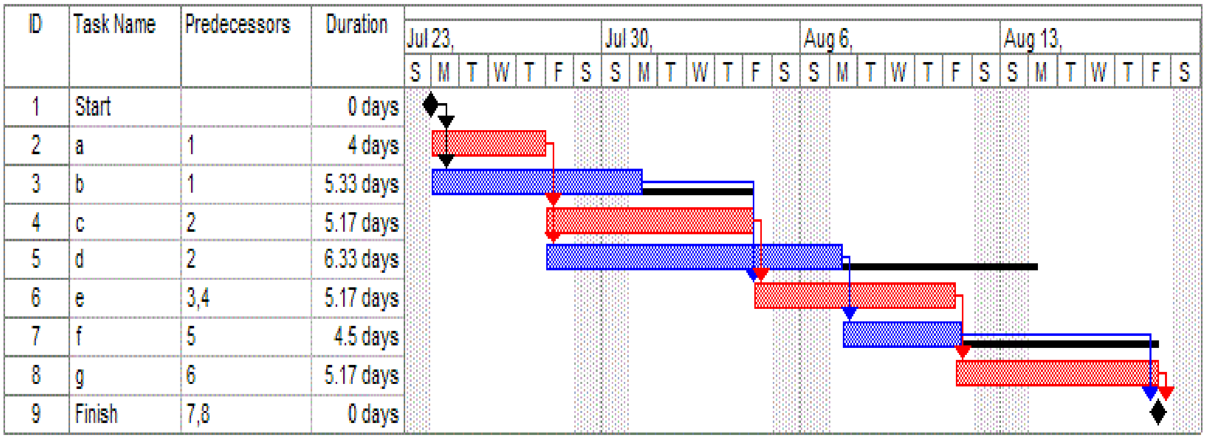
Portability:- The application is developed in ASP.NET. It would be portable to other operating system provided .NET Framework is available for the OS. As the database is made in DB2,porting the database to another database server would require some development effort.

3.3 Planning and Scheduling

A **Gantt chart** is a type of bar chart, devised by Henry Gantt in the 1910s, that illustrates a project schedule. Gantt charts illustrate the start and finish dates of the terminal elements and summary elements of a project. Terminal elements and summary elements comprise the work breakdown structure of the project. Modern Gantt charts also show the dependency (i.e., precedence network) relationships between activities. Gantt charts can be used to show current schedule status using percent-complete shadings and a vertical line.

Although now regarded as a common charting technique, Gantt charts were considered revolutionary when first introduced. This chart is also used in information technology to represent data that has been collected.

Following is the **Gantt chart** describing the project schedule for **"Online Grocers "**



ID	Task Name	Predecessors	Duration
1	Start		0 days
2	a	1	4 days
3	b	1	5.33 days
4	c	2	5.17 days
5	d	2	6.33 days
6	e	3,4	5.17 days
7	f	5	4.5 days
8	g	6	5.17 days
9	Finish	7,8	0 days

The **program** (or **project**) **evaluation and review technique**, commonly abbreviated **PERT**, is a statistical tool, used in project management, which was designed to analyze and represent the tasks involved in completing a given project. First developed by the United States Navy in the 1950s, it is commonly used in conjunction with the critical path method (**CPM**).

Following is the **PERT chart** describing the project schedule for **“Online Grocers “**

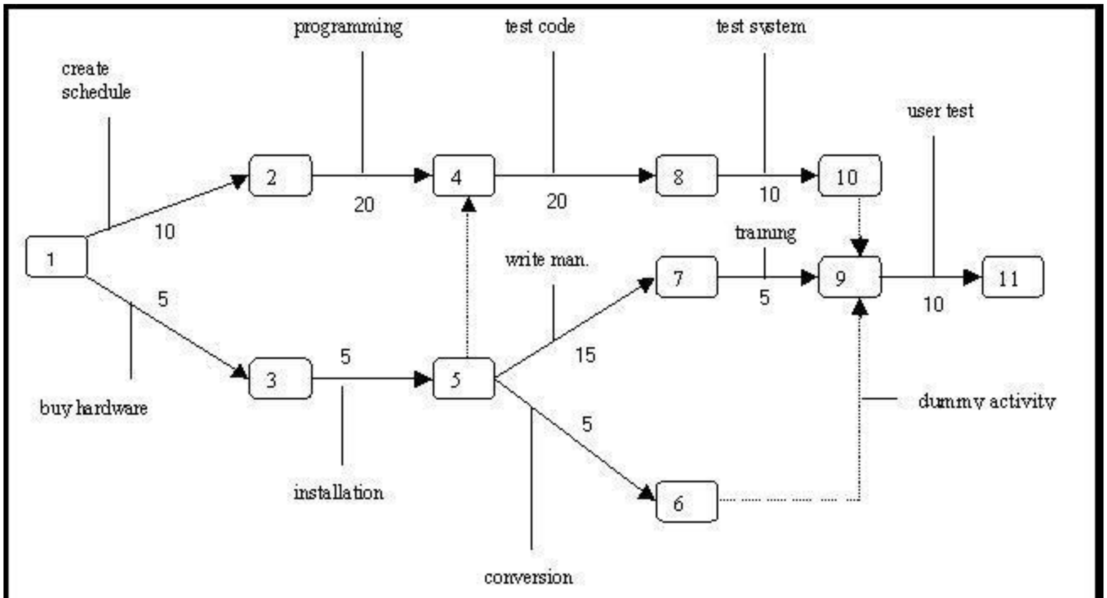


Fig. 1:
PERT Chart

* Numbered rectangles are nodes and represent events or milestones.
* Directional arrows represent dependent tasks that must be completed sequentially.
* Diverging arrow directions (e.g. 1-2 & 1-3) indicate possibly concurrent tasks
* Dotted lines indicate dependent tasks that do not require resources.

Chapter 4

PROJECT CATEGORY

4.1 Object Oriented Programming

Object Oriented Programming (OOP) is a programming paradigm that is based on the concept of objects. An object is a data structure that contains data (fields) and functions (methods).Objects are instances of classes. In OOP a class can be compared with a blueprint or a template for objects. Class is a description of what data and methods should have an object of this class.

Object Oriented Programming is based on the following concepts:

1. **Classes of objects**.
2. **Instances of classes** (objects).
3. **Encapsulation** - a class encapsulates all the fields and functions that are performed on the fields of a class. The results of encapsulation are:
    - Restriction to access some of the object's data from outside of class.
    - Bundling data to functions inside a class.

    The encapsulation is described in details in "C++ Encapsulation" topic.
4. **Polymorphism** - a way to use the same interface for the different data types. In simple words it can be described as using the same name for member functions that have different arguments. Polymorphism is not only related to member functions. It's discussed in more details in "C++ Polymorphism"
5. **Inheritance** - a class can inherit some properties from another class. This means that a child class can use some of the functionality of parent class. You can find more information about inheritance in C++ Inheritance.
6. **Abstraction** - consists in hiding the details of some processes and data and representing only necessary information and result outside the class. The detailed description of the abstraction concept can be found in "C++ Abstraction".
7. **Overloading** - represents a kind of polymorphism. There is a possibility to overload already existing functions and operators to make them work with new data types. The overloading is described in "C++ Overloading"

8. **Error handling -** some of the errors can appear in run time. Because of this, there is a need to handle errors to make programs safe. The mechanism of C++ error handling is described in "C++ Exception Handling".

4.2 Relational Database Management System

A database is a logically coherent collection of data with some inherent meaning, representing some aspect of real world and which is designed, built and populated with data for a specific purpose.

It is a collection of programs that enables user to create and maintain a database. In other words it is general-purpose software that provides the users with the processes of defining, constructing and manipulating the database for various applications.

Advantages of DBMS :

=> Redundancy is controlled.
=> Unauthorised access is restricted.
=> Providing multiple user interfaces.
=> Enforcing integrity constraints.
=> Providing backup and recovery.

Describe the three levels of data abstraction:

=> Physical level: The lowest level of abstraction describes how data are stored.
=> Logical level: The next higher level of abstraction, describes what data are stored in database and what relationship among those data.
=> View level: The highest level of abstraction describes only part of entire database.

CHAPTER5

TOOLS AND PLATFORM

5.1Front-end (Asp.net using C#):-

The software will be Web based, intuitive, easy to use and distribute. The front end will be developed in **ASP.Net using C#. ASP.NET** is a web application framework developed and marketed by Microsoft to allow programmers to build dynamic web sites, web applications and web services. ASP.NET is built on the Common Language Runtime (CLR), allowing programmers to write ASP.NET code using any supported .NET language. The ASP.NET SOAP extension framework allows ASP.NET components to process SOAP messages.

5.2Back-end (SQL Server-2008):-

Reports are prepared using **Crystal Report 10.0** the backend will be stored in **MS SQL-Server 2005.** SQL Server 2005 released in October 2005, is the successor to SQL Server 2000. It included native support for managing XML data, in addition to relational data. For this purpose, it defined an xml data type that could be used either as a data type in database columns or as literals in queries. XML columns can be associated with XSD schemas; XML data being stored is verified against the schema.

5.3 Software and Hardware Requirements

Hardware Requirements:

Secondary Storage	180 GB HDD
Memory	2 GB RAM.

Software Requirements:

Operating System	Windows xp/vista/windows7
Framework	ASP.NET framework 4
Frontend	.net with c#
Backend	SQL Server 2008
Designing Tool	Microsoft Visual Studio

3.5 Constraints On The System And The Project

Emotions is developed on ASP.Net framework and Microsoft Visual Studio 2013 version with SQL version 2008 and will run only on these platforms.

Chapter 6

SYSTEM ANALYSIS

The process of studying a procedure or business in order to identify its goals and purposes and create systems and procedures that will achieve them in an efficient way. the act, process, or profession of studying an activity (as a procedure, a business, or a physiological function) typically by mathematical means in order to define its goals or purposes and to discover operations and procedures for accomplishing them most efficiently.

In INFORMATION PROCESSING, a phase of SYSTEMS ENGINEERING. The principal objective of the systems-analysis phase is the specification of what the system needs to do to meet the requirements of end users. In the systems-design phase such specifications are converted to a hierarchy of charts that define the data required and the processes to be carried out on the data so that they can be expressed as instructions of a computer program. Many information systems are implemented with generic software, rather than with such custom-built programs.

Chapter 7

CONCEPTUAL MODEL

7.1**Physical Design**

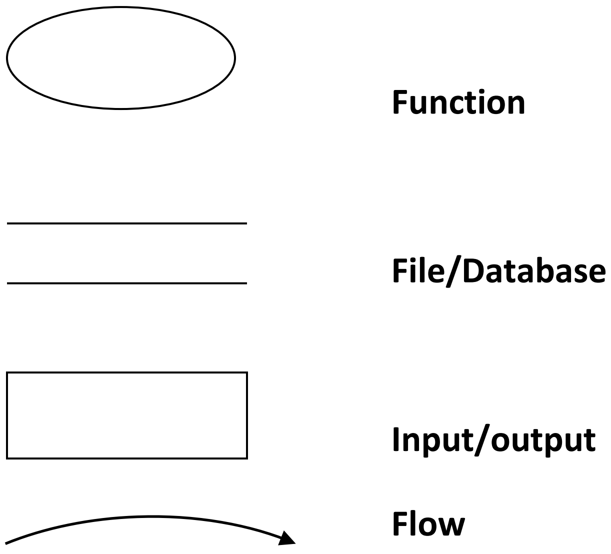
7.1(a)**Data Flow Diagram**

A Data Flow Diagram (DFD) is a graphical representation of the "flow" of data through an Information System. A data flow diagram can also be used for the visualization of Data Processing. It is common practice for a designer to draw a context-level DFD first which shows the interaction between the system and outside entities. This context-level DFD is then "exploded" to show more detail of the system being modeled.

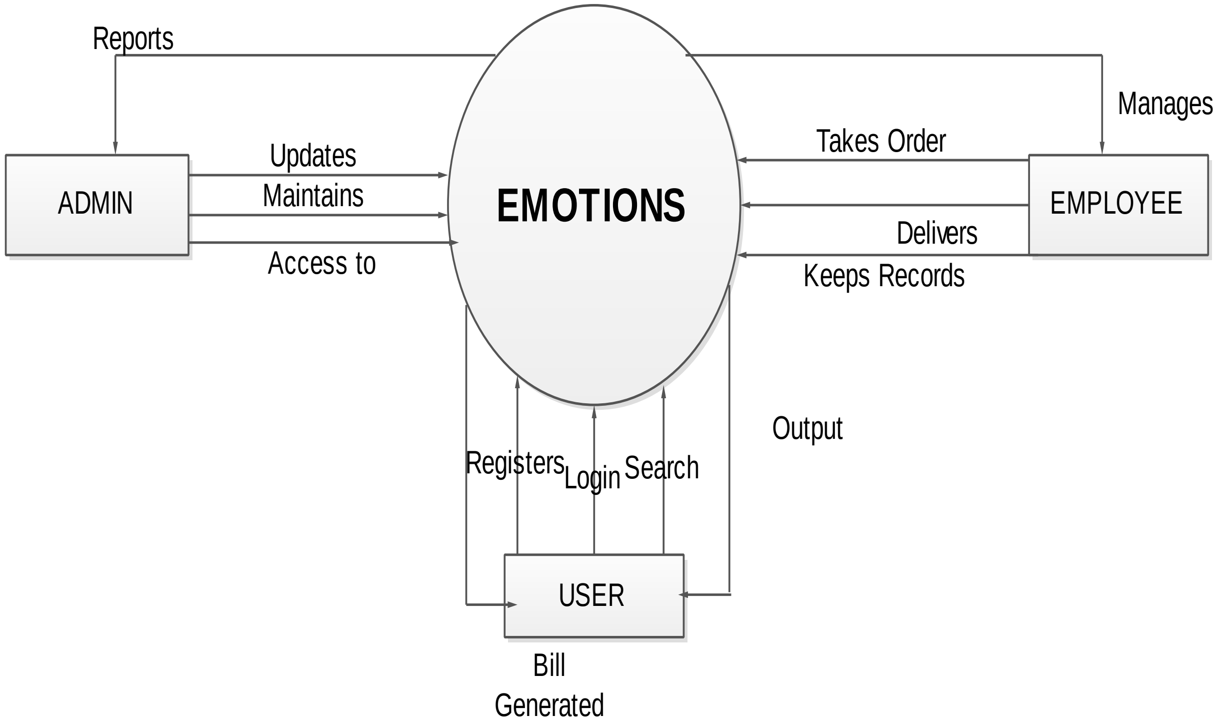
A DFD represents flow of data through a system. Data flow diagrams are commonly used during problem analysis. It views a system as a function that transforms the input into desired output. A DFD shows movement of data through the different transformations or processes in the system.

Dataflow diagrams can be used to provide the end user with a physical idea of where the data they input ultimately has an effect upon the structure of the whole system from order to dispatch to restock how any system is developed can be determined through a dataflow diagram. The appropriate register saved in database and maintained by appropriate authorities.

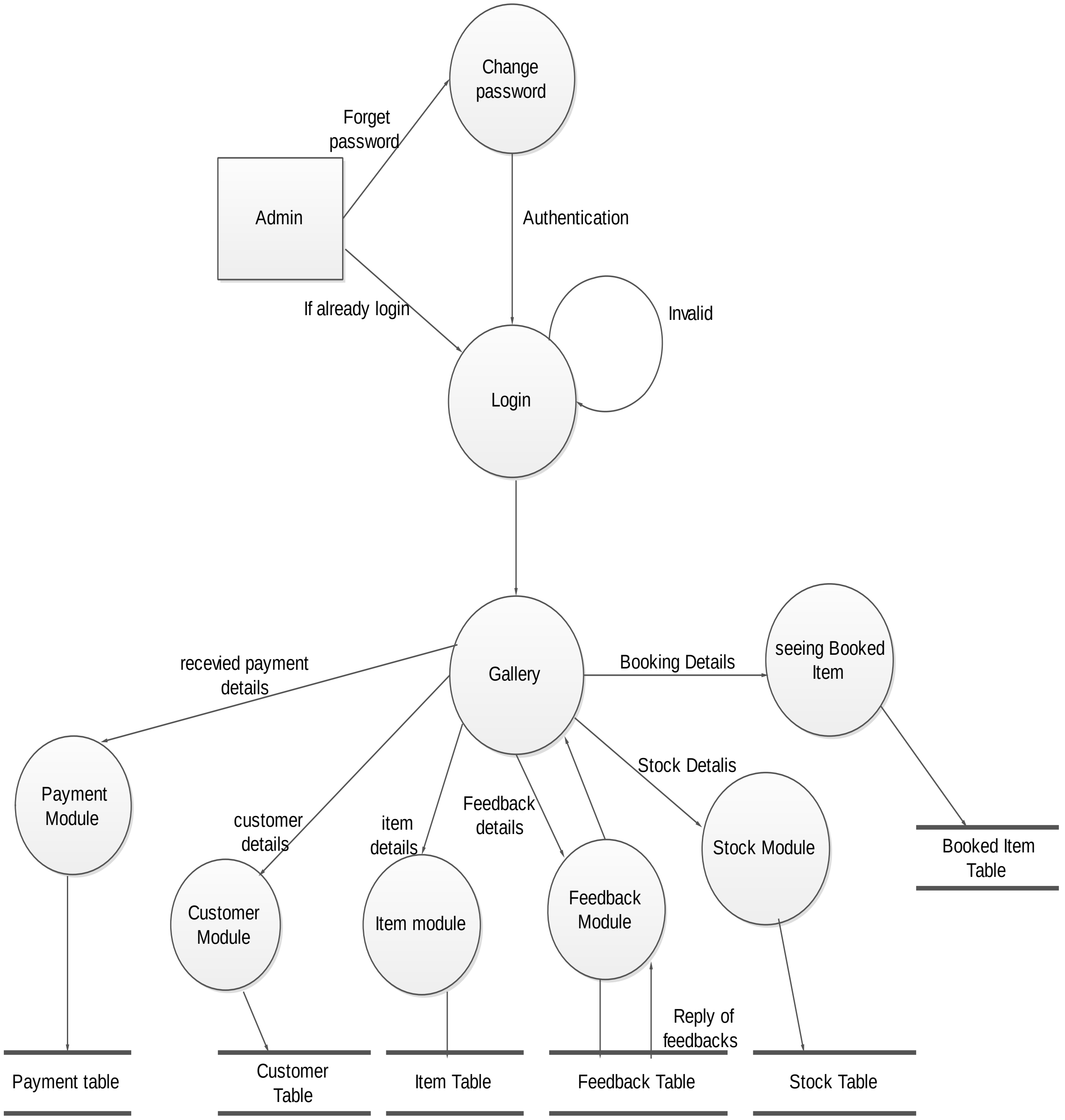
Data Flow Diagram Notation



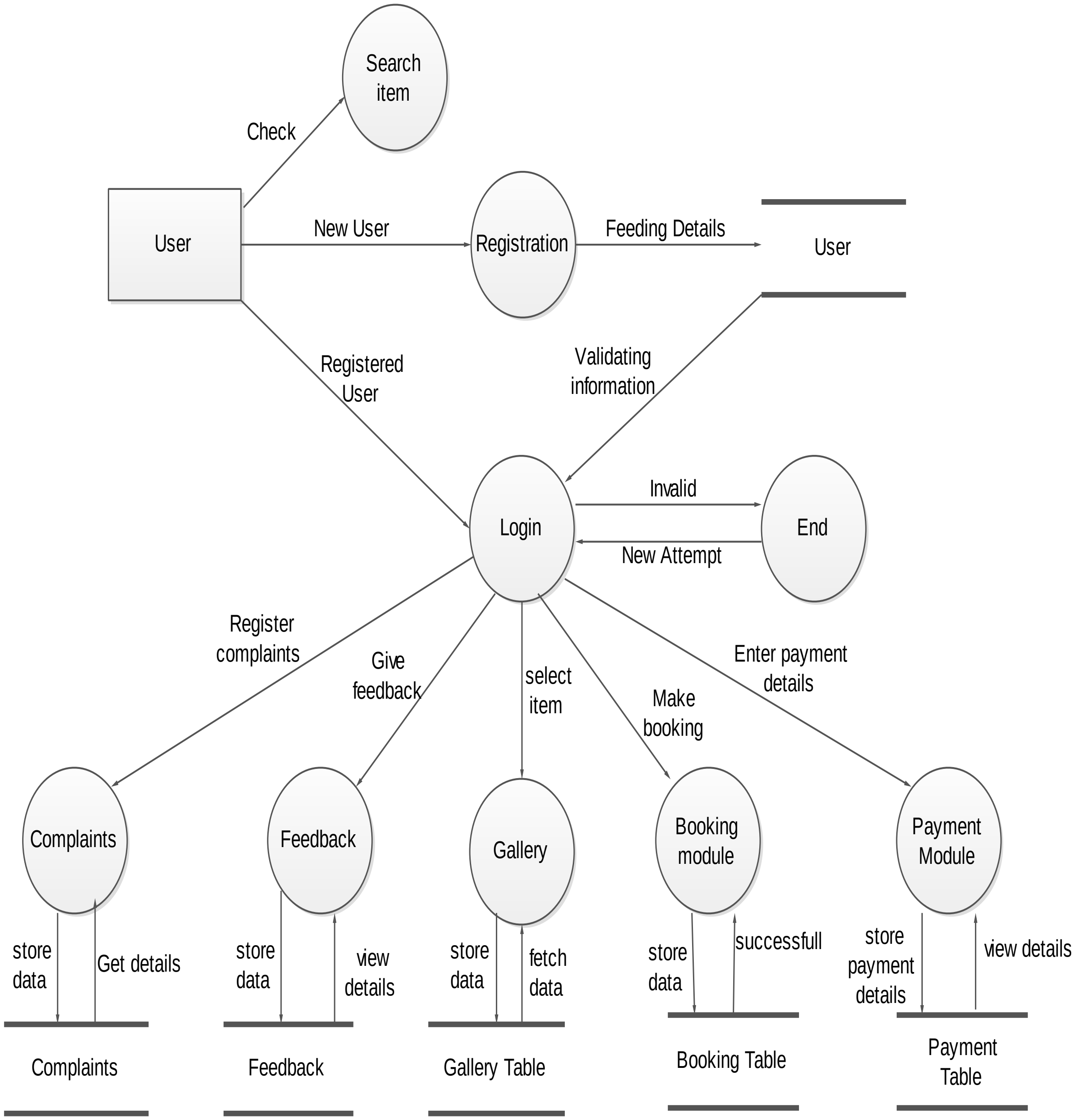
Context Level (0 Level) DFD



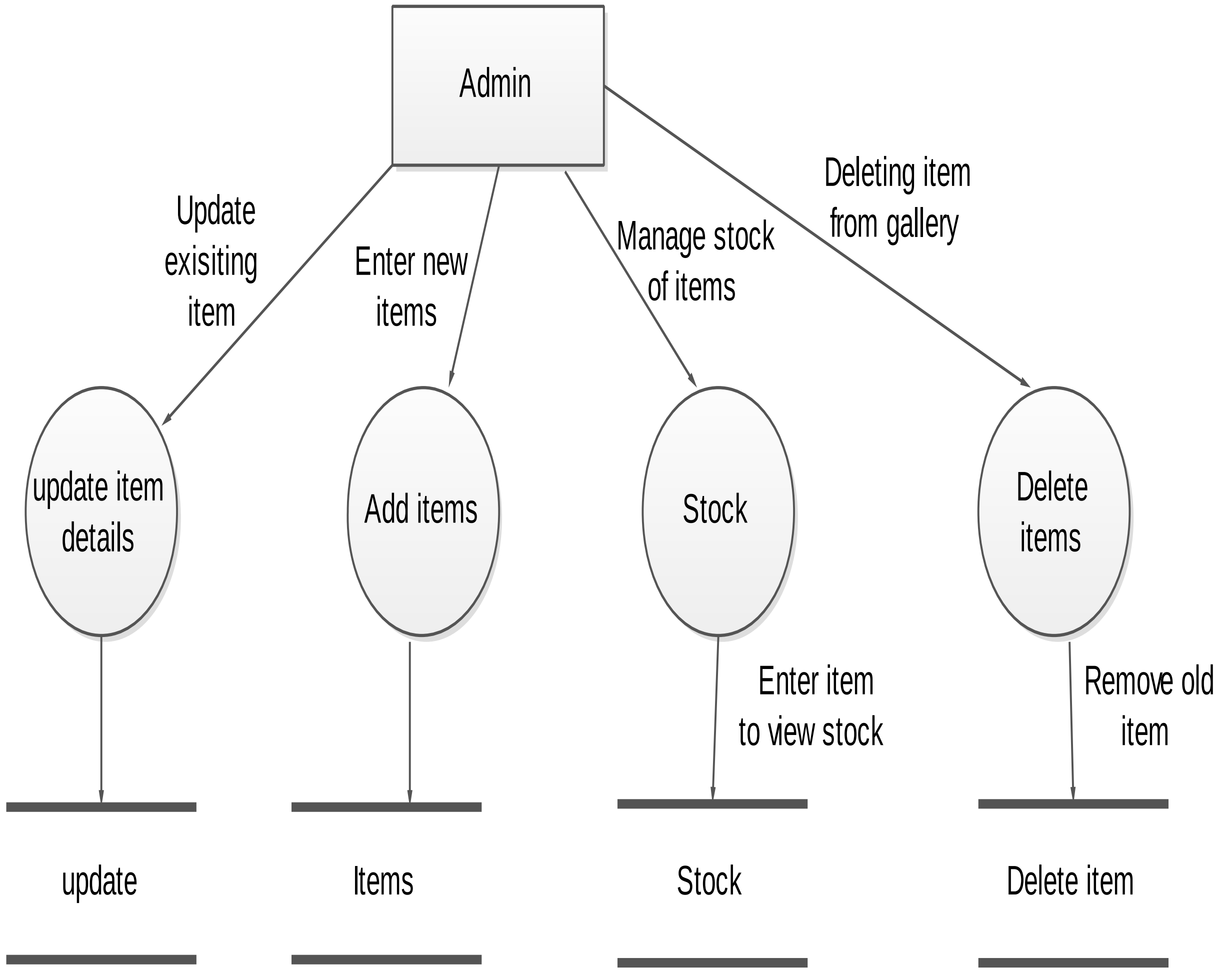
1st level Admin view DFD :-



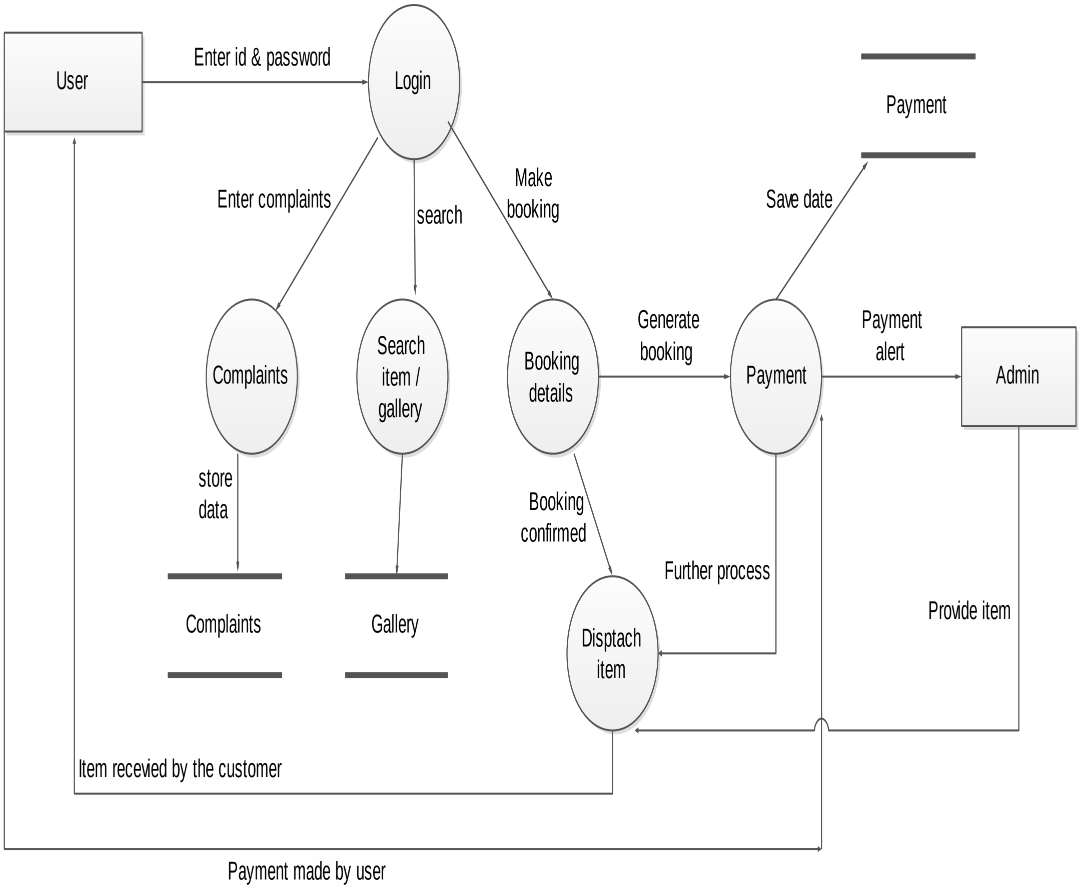
1st level user view DFD :-



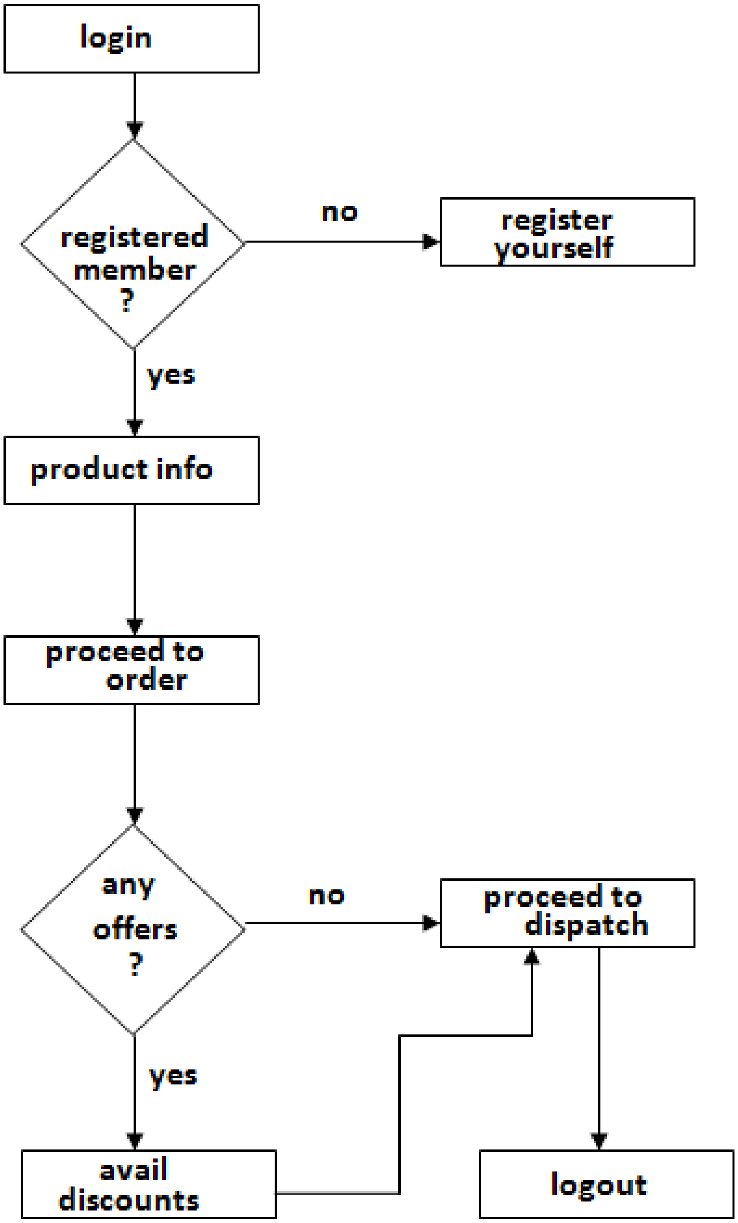
2nd level admin DFD :-



2nd level user DFD :-



7.1(b)Flow Chart



7.2Database Design

7.2(a)Entity Relationship Diagram

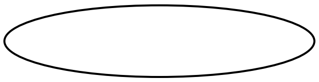
An entity-relationship (ER) diagram is a specialized graphic that illustrates the interrelationships between entities in a database. ER diagrams often use symbols to represent three different types of information. Boxes are commonly used to represent entities. Diamonds are normally used to represent relationships and ovals are used to represent attributes

An **entity-relationship model** (ERM) in software engineering is an abstract and conceptual representation of data. Entity-relationship modeling is a relational schema database modeling method, used to produce a type of conceptual schema or semantic data model of a system, often a relational database, and its requirements in a top-down fashion.

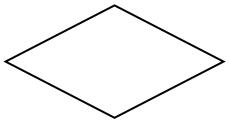
Symbols used in this E-R Diagram:

Entity: Entity is a “thing” in the real world with an independent existence. An entity may be an object with a physical existence such as person, car or employee. Entity symbol is as follows

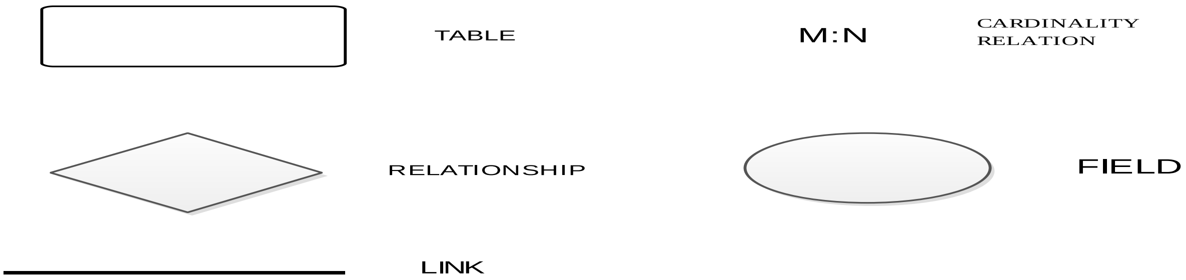
Attribute: Attribute is a particular property that describes the entity. Attribute symbol is

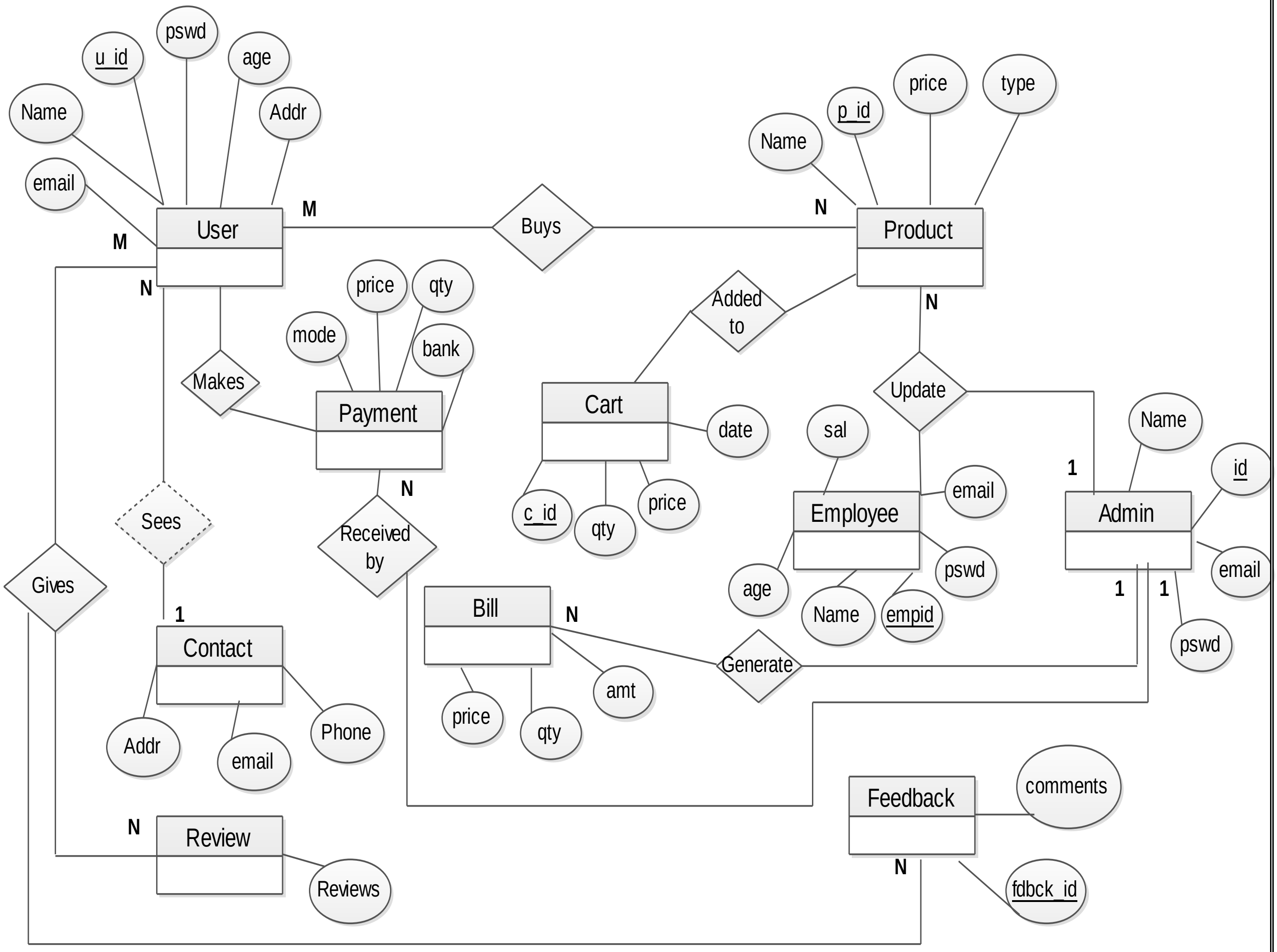


Relationship: Relationship will be several implicit relationships among various entity types whenever an attribute of one entity refers to another entity type some relationship exits. Relationship symbol is:



Key attributes: An entity type usually has an attribute whose values are distinct for each individual entity in the collection. Such an attribute is called key attribute. Key attribute symbol is as follows:




pswd
u_id
age
Name
Addr
email
User
M
M
N
Buys
N
Product
Name
p_id
price
type
price
qty
mode
bank
Makes
Payment
Added to
N
Cart
date
Update
sal
Name
Sees
Received by
N
c_id
qty
price
Employee
email
1
Admin
id
email
pswd
1
1
pswd
age
Name
empid
Gives
1
Contact
Bill
N
amt
Generate
price
qty
Addr
email
Phone
Feedback
comments
N
Review
Reviews
N
fdbck_id

User Table:

Attribute Name	Data Type	Field Size	Description
user_id	numeric	18	It is the unique key assigned to each customer.
passwd	nvaechar	50	It is the password for customer account.
name	nvarchar	50	Name of the customer.
address	nvarchar	50	Address of the customer.
contact	numeric	18	Contact details of the customer.
email	nvarchar	50	E-mail address of the customer.
age	numeric	18	Age of the Customer
sex	char		Sex of the Customer

Admin Login Table:

Attribute Name	Data Type	Field Size	Description
admin_id	numeric	18	Admin login id.
passwd	nvarchar	50	Admin login password.
name	nvarchar	50	Name of the Admin.
email	nvarchar	50	E-mail address of the admin.

Product Table:

Attribute Name	Data Type	Field Size	Description
product_id	numeric	18	It is the unique key assigned to each product.
name	nvarchar	50	Name of the product.
price	numeric	18	Price of the product.
type	nvarchar	50	Type of the Product.
G_id	Int	Foreign key	Unique identity of gallery
Image	Varchar(500)	Not null	Images of item provide by admin
Description	Varchar(450)	Not null	Details of item
Price	Big int	Not null	Amount of item

Cart Table:

Attribute Name	Data Type	Field Size	Description
cart_id	numeric	18	It is the unique key assigned to each customer’s cart.
product_id	numeric	18	It is the unique key assigned to each product.
name	nvarchar	50	Name of the customer.
quantity	numeric	18	Quantity of the Product.
price	numeric	18	Price details of the product.
amount	numeric	18	Total amount.
date	date		Date of the product added to the cart.

Payment Table:

Attribute Name	Data Type	Field Size	Description
payment_id	numeric	18	It is the unique key assigned to each payment transaction.
user_id	numeric	18	Unique key assigned to each customer.
quantity	numeric	18	Quantity of the product.
price	numeric	18	Price of the product.
amount	numeric	18	Amount to be paid.
date	date		Date of payment received.
payment mode	nvarchar	50	Mode of making payment.
bank name	nvarchar	50	Name of the Bank.
card_no	numeric	18	Number on the card.
exp_date	date		Expiry date on the card.
cvv_no	numeric	18	CVV number on the card.
address	nvarchar	50	address of the customer.
user name	nvarchar	50	Name of the customer.
time	time		Time of making Payment.

Feedback Table:

Attribute Name	Data Type	Field Size	Description
user_id	numeric	18	It is the unique key assigned to each customer.
fdbck_id	numeric	18	Unique key assigned to each feedback entry.
detail	nvarchar	50	Details of the feedback.
contact	numeric	18	Contact details of the customer.
email	nvarchar	50	E-mail address of the customer.

Bill Table:

Attribute Name	Data Type	Field Size	Description
user_id	numeric	18	It is the unique key assigned to each customer.
product_id	numeric	18	It is the unique key assigned to each product.
name	nvarchar	50	Name of the customer.
quantity	numeric	18	Quantity of the Product.
price	numeric	18	Price details of the product.
amount	numeric	18	Total amount.
date	date		Date of the product added to the cart.
email	nvarchar	50	Email id of the customer.

Contact Table:

Attribute Name	Data Type	Field Size	Description
address	nvarchar	50	Address for the company..
contact_no	numeric	18	Contact details of the company.
email	nvarchar	50	Email id of the company.

Login Table:-

Field name	Data type(size)	Constraints	Description
User_psw	Varchar(30)	Not null	User password will be shown.
User_name	Varchar(20)	Not null	User name will be shown.
User_id	Varchar(25)	Primary key	User id will be shown

LIST OF MODULES:

Overview: The Software should contain the following modules on the basis of their functionality that are able to perform the desired functions:

- Registration Module
- Login Module
- Product Module
- Update Module
- Cart Module
- Payment Module
- Bill Module
- Feedback Module
- Contact Module

- **Registration Module**

The main purpose of this module is to retrieve the customer's information for the evaluation purpose for creating a customer login account. It is upto admin whether to reject or approve the registrations and admin is responsible for the evaluation of the information.

- **Login Module**

This module is designed for the managing purpose by the admin. The admin is provided a username and password with which he/she can retrieve the information of feedback, registration, other information and also can update them.
Also, for customers a login window is created where they can also keep track of their own information, products, etc.

- **Product Module**

The module is designed in such a way that gives customer a wide view of widest range of products and categories, from where they can select products, their basic information and buy them.

- **Update Module**

The module is specially designed for the admin. The admin has full authority to update the information of every other modules as and when required.

- **Cart Module**

This module enables the customer to add products to their cart while shopping online by logging into their accounts.

- **Payment Module**

The module is designed for the criteria of Cash on Delivery (COD) and Online Payment. On the basis of which the module provides the information regarding the date of payment received, total amount, bank details and payment successful or unsuccessful status on the basis of which further steps of functions can be taken out.

- **Bill Module**

The main purpose of this module is to create the bill according to the purchase details and total amount payable by the customer and send a copy to the customer's email id and save one in the customer's account.

- **Feedback Module**

The main purpose of this module is to retrieve the customer’s reviews and suggestions and evaluating them for future purpose.

- **Contact Module**

The module provides both contact details, address details and email id of the company so as the customers gets in touch with the company.

OUTPUT FROM THE SYSTEM

- List Of Category
- List Of User
- List Of Product
- List Of Feedback
- List Of Contact
- List Of Payments.
- List Of Sales
- List of Cart

TESTING

Software testing is an investigation conducted to provide stakeholders with information about the quality of the 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 risks of software implementation. Test techniques include the process of executing a program or application with the 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 a 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 requirements that guided its design and development,
- responds correctly to all kinds of inputs,
- performs its functions within an acceptable time,
- 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 (errors 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. In contrast, under an Agile approach, requirements, programming, and testing are often done concurrently.

Roles

Software testing can be done by software testers. Until the 1980s, the term "software tester" was used generally, but later it was also seen as a separate profession. Regarding the periods and the different goals in software testing, different roles have been established: *manager*, *test lead*, *test analyst*, *test designer*, *tester*, *automation developer*, and *test administrator*.

Testing methods

Static vs. dynamic testing

There are many approaches available in software testing. Reviews, walkthroughs, or inspections are referred to as static testing, whereas actually executing programmed code with a given set of test cases is referred to as dynamic testing. Static testing is often implicit, as proofreading, 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 this are either using stubs/drivers or execution from a debugger environment.

Static testing involves verification, whereas dynamic testing involves validation. Together they help improve software quality. Among the techniques for static analysis, mutation testing can be used to ensure the test cases will detect errors which are introduced by mutating the source code.

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.

White-box testing

Main article: White-box testing

White-box testing (also known as clear box testing, glass box testing, transparent box testing and structural testing, by seeing the source code) 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 – 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 whether both the True and the 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.

Black-box testing

Black-box 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.

This method of test can be applied to all levels of software testing: unit, integration, system and acceptance. It typically comprises most if not all testing at higher levels, but can also dominate unit testing as well.

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.

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 is not required to have full access to the software's source code. Manipulating input data and formatting output do not qualify as grey-box, because 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 test.

However, tests that require modifying a back-end data repository such as a database or a log file does qualify as grey-box, as the user would not normally be able to change the data repository in normal production operations.[*citation needed*] Grey-box testing may also include reverse engineering to determine, for instance, boundary values or error messages.

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

There are generally four recognized levels of tests: unit testing, integration testing, component interface testing, and system testing. Tests are frequently grouped by where they are added in the software development process, or by the level of specificity of the test. The main levels during the development process as defined by the SWEBOK guide are unit-, integration-, and system testing that are distinguished by the test target without implying a specific process model. Other test levels are classified by the testing objective.

There are two different levels of tests from the perspective of customers: low-level testing (LLT) and high-level testing (HLT). LLT is a group of tests for different level components of software application or product. HLT is a group of tests for the whole software application or product.

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 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. Rather than replace traditional QA focuses, it augments it. Unit testing aims to eliminate construction errors before code is promoted to QA; this strategy is intended to increase the quality of the resulting software as well as the efficiency of the overall development and QA 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 verification 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.

Component interface testing

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. 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.

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 logon 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 which 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.

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)..

FUTURE SCOPE

i. The Indian Art Gallery at present not provides booking of products through telephony, but we will provide it in future.

ii. Automatic Mail sending facility is provided to the customer that purchases the product.

iii. Live customer Help will be provided in future.

iv. Websites will be mounted using secure http connection.

v. Artist and Admin are responsible for internal affairs like processing orders, assure home delivery, getting customer’s delivery-time feedback, updating order’s status and answering client’s queries online.

vi. Payment module as well as Add to cart is working offline but, in live project payments as well as Add to cart will be performing task by system only.

BIBLIOGRAPHY

❖Secondly use for this project use www.w3school.com

❖ Professional ASP.Net 4.0 by Black Mobile.

❖ The Complete Reference MYSQL

❖ The Complete Reference HTML.

Websites referred:-

http://www.google.co.in

http://www.c-sharpcorner.com

https://www.msdn.com