PRACTICAL No.1

Study the various types of development models.

A software life cycle model (SLCM) is a representation of the major components of software development work and their interrelationships in a graphical framework that can be easily understood and communicated. Just as the WBS partitions the deliverable into its component parts so the SLCM apportions the work to be done into manageable work units.

You must have a defined SLCM for your project to:

Define the work to be performed Divide up the work into manageable pieces

1. Determine project milestones at which project performance can be evaluated
2. Define the sequence of work units
3. Provide a framework for definition and storage of the deliverables produced by the project
4. Communicate your development strategy to project stakeholders

**Different life cycle models**

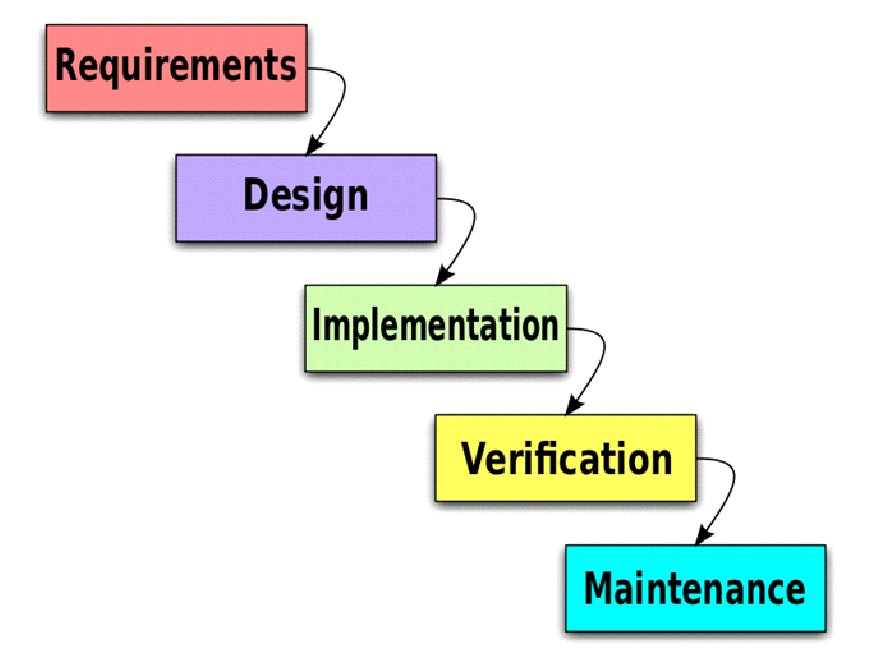
1. Classical waterfall model
2. Iterative waterfall model
3. Prototyping models
4. Evolutionary model
5. Spiral models

**Classical waterfall model**

The **waterfall model** is a sequential design process, often used in software development process, in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of Conception, Initiation, analysis, design, Construction, testing, production/implementation, and maintenance.

The waterfall development model originates in the manufacturing and construction industries; highly structured physical environments in which after-the-fact changes are prohibitively costly, if not impossible. Since no formal software development methodologies existed at the time, this hardware-oriented model was simply adapted for software development.

The first known presentation describing use of similar phases in software engineering was held by Herbert D. Belington at Symposium on advanced programming methods for digital computers on 29 June 1956.This presentation was about the development of software for SAGE. In 1983 the paper was republished with a foreword by Belington pointing out that the process was not in fact performed in a strict top-down fashion, but depended on a prototype.



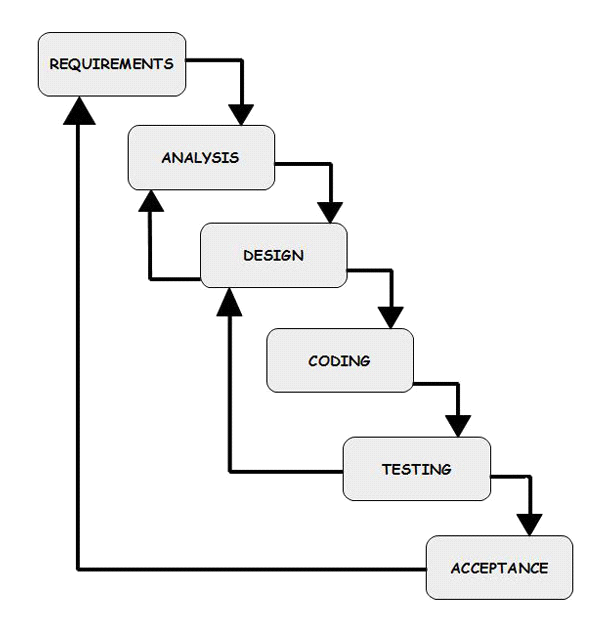
**Figure1.1 classical waterfall model**

**The stages of "The Waterfall Model" are:**

* **Requirement Analysis & Definition:** This phase is focused on possible requirements of the system for the development are captured. Requirements are gathered subsequent to the end user consultation.
* **System & Software Design:** Prior to beginning the actual coding, it is inevitable to understand what actions are to be taken and what they should like. The requirement specifications are studied in detail in this phase and the design of the system is prepared. The design specifications are the base for the implementation and unit testing model phase.
* **Implementation & Unit Testing:** Subsequent to receiving the system design documents, the work is shared into various modules and the real coding is commenced. The system is developed into small coding units. These units are later integrated in the subsequent phase. Every unit is tested for its functionality.
* **Integration & System Testing:** The modules that are divided into units are integrated into a complete system and tested for proper coordination among modules and system behaves as per the specifications. Once the testing is completed, the software product is delivered to the customer.
* **Operations & Maintenance:** It is a never ending phase. Once the system is running in production environment, problems come up. The issues that are related to the system are solved only after deployment of the system. The problems arise from time to time and need to be solved; hence this phase is referred as maintenance.

**Iterative waterfall model**

The iterative enhance model counters the third limitation of the waterfall model and tires to combine a benefit of both prototyping and the waterfall mode. The basic idea is that the software should be developed in increments, each increment adding some functional capability to the system until the full system is implemented. At each step, extensions and design modifications can be made. An advantage of this approach is that it can result in better testing because testing each increment is likely to be easier than testing the entire system as in the waterfall model. The increment models provide feedback to the client i.e., useful for determining the final requirements of the system.  
In the first step of this model, a simple initial implementation is done for a subset of the overall problem. This subset is one that contains some of the key aspects of the problem that are easy to understand and implement and which form a useful and usable system. A project control list is created that contains, in order, all the tasks that must be performed to obtain the final implementation this project control list gives a idea of how far the project is at any given step from the final system.



**Figure1.2 Iterative waterfall model**

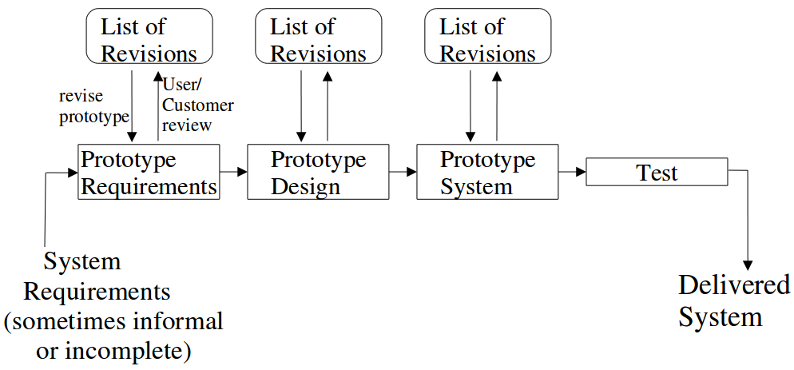
**Phase containment**

Through errors are inevitable in almost every phase of development it is desirable to detect these errors in the same phase in which they occur. This can reduce the effort required for correcting bugs it may not possible to detect all errors in the same phase in which they occur. Nevertheless, the error should be detected as early as possible.

Prototyping Model

The goal of prototyping based development is to counter the first two limitations of the waterfall model discussed earlier. The basic idea here is that instead of freezing the requirements before a design or coding can proceed, a throwaway prototype is built to understand the requirements. This prototype is developed based on the currently known requirements. Development of the prototype obviously undergoes design, coding and testing. But each of these phases is not done very formally or thoroughly. By using this prototype, the client can get an "actual feel" of the system, since the interactions with prototype can enable the client to better understand the requirements of the desired system.

Prototyping is an attractive idea for complicated and large systems for which there is no manual process or existing system to help determining the requirements. In such situations letting the client "plan" with the prototype provides invaluable and intangible inputs which helps in determining the requirements for the system. It is also an effective method to demonstrate the feasibility of a certain approach. This might be needed for novel systems where it is not clear that constraints can be met or that algorithms can be developed to implement the requirements.



**Figure 1.3 Prototyping model**

**Evolutionary model**

This life cycle model is also referred to as the successive version model and sometimes as the incremental model. In this model first a simple working system is built which subsequently undergoes many functionality improvements and additions until the desired system is realized. The evolutionary software development press is therefore sometimes referred to design a little, built a little, test a little, and deploy a little model. This is once the requirements have been specified; the design, built, test, and deployment activities are interleaved.

**Life cycle activities**

In the evolutionary life cycle model, the software requirements are first broken down into several modules that can be incrementally constructed and delivered. The development team first develops the core modules of the system. The core modules are those that donot need service from the other modules. In other hand non-core modules need services from core module. Each evolutionary version may be developed using an iterative waterfall model of development. Each successive version of the product is fully functionality software capable of performing more work than the previous version

Spiral model

The spiral model was introduced, due to the shortcomings in the waterfall and prototype models of software engineering. It is a combination of the said two models of software development. From the name of the model, it can be derived that the activities of software development are carried out like a spiral. To explain the model further, the entire software development process is broken down into small projects. The phases of the spiral model are as follows:

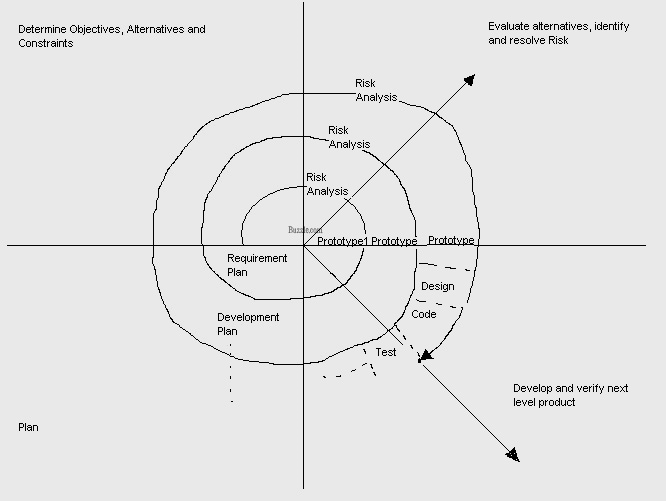
* Planning Phase
* Risk Analysis Phase
* Engineering Phase
* Coding and Implementation Phase
* Evaluation Phase

Risk handing in spiral model

A risk is essential any adverse circumstance that might hamper the successful completion of a software project. As an example, the risk involved in accessing data from a remote database can be that the data access rate might be too slow. This risk can be resolved by building a prototype of the data access subsystem and experimenting with the exact access rate.

**Phases of spiral model**

Each phase in this model is split into four sectors. In first quadrant, some features of the product are identified based on the severity of the risk and how crucial it is to overall product development implementation of the identified features form the objective of the phase. The objectives are investigated, elaborated and analyzed. Based on this, the risk involved in meeting the phase objectives are identified. Also alternative solution possible for the phase under considerations is proposed. During the second quadrant the alternative solutions are evaluated to select the best possible solution. To ­­be able to do this the solutions are evaluated by developing an appropriate prototype. Activities during the third quadrant consist od developing and verifying the the next level of the product. At the end of the third iteration, the identified features have been implemented and the next version of the product is available. Activities during the fourth quadrant concern reviewing the result of the stages traversed so far with the customer and planning the next iteration around the spiral. Therefore while using this model, the project manager need to play a very crucial role in turning the spiral model to the specific project.



**Figure 1.4 Spiral model**

PRACTICAL No.2

To study and develop: Software Requirements Specification

A software requirements specification (SRS) is a comprehensive description of the intended purpose and environment for [software](http://searchsoa.techtarget.com/definition/software) under development. The SRS fully describes what the software will do and how it will be expected to perform. An SRS minimizes the time and effort required by developers to achieve desired goals and also minimizes the development cost. A good SRS defines how an [application](http://searchsoftwarequality.techtarget.com/definition/application) will interact with system [hardware](http://searchcio-midmarket.techtarget.com/definition/hardware), other programs and human users in a wide variety of real-world situations. Parameters such as operating speed, [response time](http://searchnetworking.techtarget.com/definition/response-time), [availability](http://searchnetworking.techtarget.com/definition/availability), [portability](http://searchstorage.techtarget.com/definition/portability), maintainability,  [footprint](http://whatis.techtarget.com/definition/footprint), security and speed of recovery from adverse events are evaluated.

# Introduction

# Purpose

The purpose of this document is to present a detailed description of the Web News Aggregator. It will explain the purpose and features of the system, the interfaces of the system, what the system will do, the constraints under which it must operate and how the system will react to external stimuli. It provides the news of all occurrences around the globe viz. Technology news, Entertainment news, Sports news, India news, Mobile news, Programming news.

# Scope

This application will be a Web news aggregator for a local public. This application will be designed to maximize the news providing capabilities to the public for better grasp up all the world’s news in the best possible and simple way. It will work on every system having browser support. It will be cross platform web news aggregator & it will be open source which means its source code will be available to download and anybody can change it according to their needs. Thus it will be the complete open source cross platform news aggregate solution for the users. The Overall Description section, of this document gives an overview of the functionality of the product. It describes the informal requirements and is used to establish a context for the technical requirements specification.

The Requirements Specification section, of this document is written primarily for the developers and describes in technical terms the details of the functionality of the product. Both sections of the document describe the same software product in its entirety, but are intended for different audiences and thus use different language

# Definitions, Acronyms and Abbreviations

HTTP: Hypertext Transfer Protocol is a transaction oriented client/server protocol between a web browser & a Web Server.

HTTPS:Secure Hypertext Transfer Protocol is a HTTP over SSL (secure socket layer).

TCP/IP:Transmission Control Protocol/Internet Protocol, the suite of communication to connect hosts on the Internet. TCP/IP uses several protocols, the two main ones being TCP and IP.

HTML: Hypertext Markup Language (HTML) is the main markup language for creating web pages and other information that can be displayed in a web browser.HTML is written in the form of HTML elements consisting of tags enclosed in angle brackets (like <html>), within the web page content. HTML tags most commonly come in pairs like <h1> and </h1>, although some tags, known as empty elements, are unpaired, for example <img>.

# References

Books: 1. Programming PHP, 3rd Edition by Kevin Tatroe, Peter Macintyre

2. Learning PHP, MySQL, JavaScript, and CSS, 2nd Edition

Websites: <http://w3schools.com/>

<http://php.net/manual/en/tutorial.php>

<http://www.homeandlearn.co.uk/php/php.html>

# Processing Environment H/W, S/W

This shows the hardware and software required for the system to run. E.g., where will we install the software, what are the minimum hardware requirements of the system? The hardware specified should be less costly, easily maintained and can incorporate changes and flexible while using it.

**Hardware Specification**

|  |  |
| --- | --- |
| Hardware Type | Specifications(environment) |
| 1. Personal Computer | * 1. memory requirement of 40GB hard disk & 2 GB RAM   2. Pentium 3 or higher   3. keyboard   4. mouse   5. monitor   6. DVD writer |
| 2. Printer | Inkjet , laser can be used |
| 3. UPS | UPS with minimum battery life of 12 hours |
| 4. Scanner | Preferably flatbed scanner |

**Table no. 2.1 Hardware Specification**

**Software Specification**

|  |  |
| --- | --- |
| Hardware Type | Software(environment) |
| Operating System  Application Software | Windows XP onwards.  Client Side: Browser  Server Side: Apache web server |

**Table no. 2.2 Software Specification**

1

# Technologies to be used

# 1. Programming languages

PHP: PHP is a server-side scripting language designed for web development but also used as a general-purpose programming language. PHP code is interpreted by a web server with a PHP processor module which generates the resulting web page: PHP commands can be embedded directly into an HTML source document rather than calling an external file to process data. It has also evolved to include a command-line interface capability and can be used in standalone graphical applications.

HTML, XML: Hyper Text Markup Language and Extensible markup Language are the predominant markup languages for web pages. It provides a means to describe the structure of text-based information in a document and to supplement that text with interactive forms, embedded images, and other objects.

JavaScript: A client side scripting language used to create dynamic web content and user interface.

# Tools & Development Environment

Xampp Server: XAMPP is a protocol which allows messages to be sent in XML, the messages are constrained to type, it can be anything, IM messages, or an application talking to another application over the internet. Jabber has the most popular implementation and it is the same protocol used by Gtalk. There might be PHP library that allows you to use XAMPP for communication.

Abode Dreamweaver: Dreamweaver combines HTML and CSS plain text editing with a WYSIWYG tool which lets you check progress as you go or edit WYSIWYG on the fly. It's the industry standard tool for this sort of thing, although there are many, many programs which do more-or-less the same thing.

3. Bootstrap & Reveal.js

**Bootstrap**: Twitter Bootstrap is a free collection of tools for creating websites and web applications. It contains HTML and CSS-based design templates for typography, forms, buttons, charts, navigation and other interface components, as well as optional JavaScript extensions.

**Reveal.js**: A framework for easily creating beautiful presentations using HTML. Check out the live demo. reveal.js comes with a broad range of features including nested slides, markdown contents, PDF export, speaker notes and a JavaScript API. It's best viewed in a browser with support for CSS 3D transforms but fallbacks are available to make sure your presentation can still be viewed elsewhere.

# Overview

# Overall Description:

2.1.1 Product Perspective

## News Empire is aimed towards the users who want to grasp all the news around the globe from the comfort of their home and in a simple possible way. This project will change the way users read news. It should be user-friendly, ‘quick to learn’ and reliable application for the above purpose. It is intended to be a stand-alone product and should not depend on the availability of other software. It should run on both UNIX and Windows based platform.

## 2.1.2 Product Functions

User: Guests

Functions: A Customer can browse through different categories of news according to his needs. There are different section available for guests to choose from viz. Technology, Entertainment, Sports, India news, World news, Mobile, Programming, Design, Development & Quotes section. They almost cover all the news in the real time.

# User Characteristics

* + - The user should be familiar with the Internet.
    - The user should be familiar with browsing different websites.

# Constraints

# Limited to HTTP/HTTPS.

# No multilingual support

# Use-Case Diagram

# 

**Figure No.2.1 Use Case Diagram**

# Assumptions and Dependencies

# It is assumed that users are only interested in the predefined news sections.

# Roles and tasks are predefined.

# Functional requirements:

# Secure connection and news reading facilites for Customers.

# Browsing through the internet to see the items that are there in each category of the news app.

# Adequate reading mechanisms for easy and quick access to particular news.

# Maintaining news of regular websites of different needs.

# Non-Functional requirements:

# 24 X 7 availability

# Better component design to get better performance at peak time

1. **Conclusion**

Thus this news app provides the single solution for all the news around the globe viz. Technology, Entertainment, Sports, India news, World news, Mobile, Programming, Design, Development & Quotes in the real time, & thus solves our purpose of making this unique web app. Sometimes users are required not to read the full article but to read just the titles. Thus this app only deals with titles and if user wishes to read the further article, he/she is taken to the source website of the articles. It is a fluid news aggregator, thus will work equally well on any system or device.

PRACTICAL No.3

To study and develop Data Flow diagram (DFD)

The DFD (also known as a bubble chart) is a hierarchical graphical model of a system that shows the different processing activities or functions that the system performs and the data interchange among these functions. Each function is considered as a processing station (or process) that consumes some input data and produces some output data. The system is represented in terms of the input data to the system, various processing carried out on these data, and the output data generated by the system. A DFD model uses a very limited number of primitive symbols to represent the functions performed by a system and the data flow among these functions.

Entity



Process



Data Flow



Data Store



Output



**Importance of DFDs in a good software design**

The main reason why the DFD technique is so popular is probably because of the fact that DFD is a very simple formalism – it is simple to understand and use. Starting with a set of high-level functions that a system performs, a DFD model hierarchically represents various sub-functions. In fact, any hierarchical model is simple to understand. Human mind is such that it can easily understand any hierarchical model of a system – because in a hierarchical model, starting with a very simple and abstract model of a system, different details of the system are slowly introduced through different hierarchies. The data flow diagramming technique also follows a very simple set of intuitive concepts and rules. DFD is an elegant modeling technique that turns out to be useful not only to represent the results of structured analysis of a software problem, but also for several other applications such as showing the flow of documents or items in an organization.

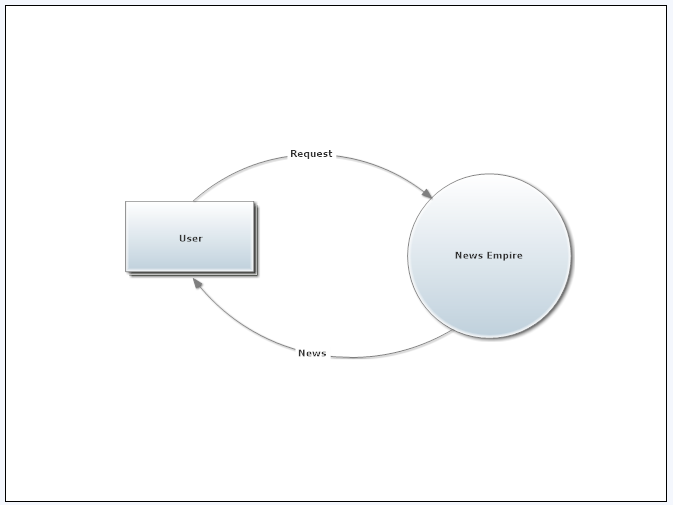
**LEVEL 0:** It is also called context-level data flow diagram which shows the interaction between the system and external agents which act as data sources and data sinks. On the context diagram the system's interactions with the outside world are modeled purely in terms of data flows across the *system boundary*. The context diagram shows the entire system as a single process, and gives no clues as to its internal organization.

**LEVEL 1:** This context-level DFD is next "exploded", to produce a Level 1 DFD that shows some of the detail of the system being modeled. The Level 1 DFD shows how the system is divided into sub-systems (processes), each of which deals with one or more of the data flows to or from an external agent, and which together provide all of the functionality of the system as a whole. It also identifies internal data stores that must be present in order for the system to do its job, and shows the flow of data between the various parts of the system.

**LEVEL 2:** DFD level-1 is exploded further into LEVEL-2 DFD.

DFD of Affiliate online marketing system (Macintosh Data Recovery)

**Context Level**



**Figure 3.1 DFD Level 0**

**PRACTICAL NO.4**

**To study and develop: Data dictionary**

A data dictionary is a collection of descriptions of the data objects or items in a data model for the benefit of programmers and others who need to refer to them. A first step in analyzing a system of objects with which users interact is to identify each object and its relationship to other objects. This process is called data modeling and results in a picture of object relationships. After each data object or item is given a descriptive name, its relationship is described (or it becomes part of some structure that implicitly describes relationship), the type of data (such as text or image or binary value) is described, possible predefined values are listed, and a brief textual description is provided. This collection can be organized for reference into a book called a data dictionary. There are two other types of Data Dictionaries-

* Active Data Dictionary- A data dictionary that is automatically updated by the DBMS every time the database is accessed.
* Passive Data Dictionary- Similar to Active DD however it is not automatically updated and usually requires a batch process to be run.
* There is a third style of data dictionaries known as a middleware data dictionary. Middleware is computer software that connects software components or applications. The software consists of a set of services that allows multiple processes running on one or more machines to interact. Traditional data dictionaries provide structure and basic function to the database. Middleware data dictionaries are located within the DBMS itself and operate on a higher level. Middleware data dictionaries can provide alternate entity relationship structures that can be tailored to fit different users that interact with the same database. Middleware data dictionaries can also assist in query optimization as well as distributed database. Middleware also helps database designers by reducing the amount of time it takes to create forms, queries, reports, menus and many other database components. They do this by automatically generating SQL code for common items such as forms and views. Some middleware data dictionaries can also help with data security as well as database optimization. It is a growing field with many new companies entering the market.

A data dictionary may contain information such as:

* Database design information
* Stored SQL procedures
* User permissions
* User statistics
* Database process information
* Database growth statistics
* Database performance statistics

**PRACTICAL No. 5**

**To study and develop: E-R diagram**

An ER model is an abstract way to describe a database. Describing a database usually starts with a relational database, which stores data in tables. Some of the data in these tables point to data in other tables - for instance, your entry in the database could point to several entries for each of the phone numbers that are yours. The ER model would say that you are an entity, and each phone number is an entity, and the relationship between you and the phone numbers is 'has a phone number'. Diagrams created to design these entities and relationships are called entity–relationship diagrams or ER diagrams.

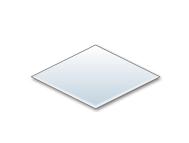
There are three basic elements in ER models:

* Entities are the "things" about which we seek information.
* Attributes are the data we collect about the entities.
* Relationships provide the structure needed to draw information from multiple entities.

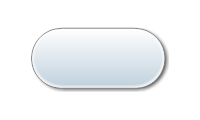
Entity



Relationship



Attribute



 There are three types of relationships between entities:

* **One-to-one**: one instance of an entity (A) is associated with one other instance of another entity (B). For example, in a database of employees, each employee name (A) is associated with only one social security number (B).
* **One-to-many**: one instance of an entity (A) is associated with zero, one or many instances of another entity (B), but for one instance of entity B there is only one instance of entity A. For example, for a company with all employees working in one building, the building name (A) is associated with many different employees (B), but those employees all share the same singular association with entity A.
* **Many-to-many**: one instance of an entity (A) is associated with one, zero or many instances of another entity (B), and one instance of entity B is associated with one, zero or many instances of entity A. For example, for a company in which all of its employees work on multiple projects, each instance of an employee (A) is associated with many instances of a project (B), and at the same time, each instance of a project (B) has multiple employees (A) associated with it.

How do we start an ERD?

1. Define Entities: these are usually nouns used in descriptions of the system, in the discussion of business rules, or in documentation; identified in the narrative.

2. Define Relationships: these are usually verbs used in descriptions of the system or in discussion of the business rules (entity \_ entity); identified in the narrative (see highlighted items above).

3. Add attributes to the relations; these are determined by the queries, and may also suggest new entities, e.g. grade; or they may suggest the need for keys or identifiers.

4. Add cardinality to the relations

Many-to-Many must be resolved to two one-to-many with an additional entity

Usually automatically happens

Sometimes involves introduction of a link entity (which will be all foreign key) Examples: Patient-Drug

6. Represent that information with symbols.

**Limitations**

* ER models assume information content that can readily be represented in a relational database. They describe only a relational structure for this information.
* They are inadequate for systems in which the information cannot readily be represented in relational form, such as with semi-structured data.
* For many systems, the possible changes to the information contained are nontrivial and important enough to warrant explicit specification.
* Someauthors have extended ER modeling with constructs to represent change, an approach supported by the original author; an example is Anchor Modeling. An alternative is to model change separately, using a process modeling technique. Additional techniques can be used for other aspects of systems. For instance, ER models roughly correspond to just 1 of the 14 different modeling techniques offered by UML.
* ER modeling is aimed at specifying information from scratch. This suits the design of new, standalone information systems, but is of less help in integrating pre-existing information sources that already define their own data representations in detail.
* Even where it is suitable in principle, ER modeling is rarely used as a separate activity. One reason for this is today's abundance of tools to support diagramming and other design support directly on relational database management systems. These tools can readily extract database diagrams that are very close to ER diagrams from existing databases, and they provide alternative views on the information contained in such diagrams.
* The enhanced entity–relationship model (EER modeling) introduces several concepts which are not present in ER modeling, which are closely related to object-oriented design, like is-a relationship.

**PRACTICAL No. 6**

**To study and develop: Structure Chart.**

A **Structure Chart** (SC) in [software engineering](http://en.wikipedia.org/wiki/Software_engineering) and [organizational theory](http://en.wikipedia.org/wiki/Organizational_theory), is a [chart](http://en.wikipedia.org/wiki/Chart) which shows the breakdown of a system to its lowest manageable levels. They are used in [structured programming](http://en.wikipedia.org/wiki/Structured_programming) to arrange program modules into a tree. Each module is represented by a box, which contains the module's name. The tree structure visualizes the relationships between modules.

A structure chart is a [top-down modular design](http://en.wikipedia.org/wiki/Top-down_design) tool, constructed of squares representing the different modules in the [system](http://en.wikipedia.org/wiki/System), and lines that connect them. The lines represent the connection and or ownership between activities and sub activities as they are used in organization.

In [structured analysis](http://en.wikipedia.org/wiki/Structured_analysis) structure charts, according to Wilber (2009), "are used to specify the high-level design, or architecture, of a [computer program](http://en.wikipedia.org/wiki/Computer_program). As a design tool, they said the programmer in dividing and conquering a large software problem, that is, recursively breaking a problem down into parts that are small enough to be understood by a human brain. The process is called [top-down design](http://en.wikipedia.org/wiki/Top-down_design), or [functional decomposition](http://en.wikipedia.org/wiki/Functional_decomposition). Programmers use a structure chart to build a program in a manner similar to how an architect uses a blueprint to build a house. In the design stage, the chart is drawn and used as a way for the client and the various software designers to communicate. During the actual building of the program (implementation), the chart is continually referred to as "the master-plan".

A structure chart depicts

* the size and complexity of the system, and
* number of readily identifiable functions and modules within each function and
* Whether each identifiable function is a manageable entity or should be broken down into smaller components.

A structure chart is also used to [diagram](http://en.wikipedia.org/wiki/Diagram) associated elements that comprise a run stream or thread. It is often developed as a hierarchical, but other representations are allowable. The representation must describe the breakdown of the [configuration system](http://en.wikipedia.org/wiki/Configuration_system) into [subsystems](http://en.wikipedia.org/wiki/Subsystem)  and the lowest manageable level. An accurate and complete structure chart is the key to the determination of the configuration items, and a visual representation of the configuration system and the internal interfaces among its CIs. During the configuration control process, the structure chart is used to identify CIs and their associated artifacts that a proposed change may impact.

**Building Blocks:** The basic building blocks which are used to design structure charts are the following:

• **Rectangular boxes:** Represents a module.

• **Module invocation arrows:** Control is passed from one module to another module in the direction of the connecting arrow.

• **Data flow arrows:** Arrows are annotated with data name; named data passes from one module to another module in the direction of the arrow.

• **Library modules:** Represented by a rectangle with double edges.

• **Selection:** Represented by a diamond symbol.

• **Repetition:** Represented by a loop around the control flow arrow.

**Structure chart of NewsEmpire application**

**NewsEmpire**

**Android news**

**Bollywood News**

**Figure No. 6.1 Structured Chart**

Sports News

Android News

**Administration**

**ATM**

NEWS EMPIRE

Bollywood News

INDIA NEWS

**Figure No. 6.2 Structured Chart**

**PRACTICAL No. 7**

**To study and develop: Object oriented design and user interface design.**

**Object Oriented Analysis**

The purpose of object oriented analysis is to develop solution models that satisfy the customer requirement. The object oriented analysis generates model of the problem domain. It represents classes, objects and interaction between objects. We use UML (Unified Modeling Language) to represent the analysis details. Use Case Diagram, Class diagram, are used to model the Object Oriented Analysis using UML. Object-oriented development is the process of turning an idea or a problem specification into an object-oriented program. That program consists of a group of objects that communicate with one another.The objects are created as the program executes, and

They are instances of classes that exist in the program's source code. Object-oriented analysis (OOA) transforms a glimmer of a product into an object-oriented model of that problem. A definition of the functional requirements of the system to be developed and a description of the classes in the problem domain are included in a typical object-oriented analysis model. Some analysis models also include a definition of the high-level system components (such as subsystems) and their interactions, as well as the general interactions among instances of the problem domain classes.

**Object Oriented Design**

The Object Oriented Design converts the Object Oriented Analysis model into a design model. This serves an outline for software construction. Object Oriented Design supports following object oriented concepts such as Abstraction, Information Hiding, Functional Independence, and Modularity. Design is the initial step in moving towards from the problem domain to the solution domain. A detailed design includes specification of all the classes with its attributes, detailed interface. The purpose of design is to specify a working solution that can be easily translated into a programming language code. Object-oriented design (OOD) entails transforming the analysis model into a feasible design. Much of design involves refining the analysis model through the introduction of classes and algorithms that define exactly how certain required features are realized. Another common design activity is the development of overall system architecture. This entails organizing the system as a set of major building blocks, such as subsystems or components. For a concurrent system, the architecture includes the basic task or process structure. For a distributed system, it includes the organization of hardware in terms of processors and their interconnections. Once the design reaches a sufficient level of specificity, it is translated into an implementation through object-oriented programming. This task can be either relatively straightforward or rather challenging, depending upon the amount of detail in the design (and perhaps on other factors, too).

The Object Oriented Design is classified into

* Architectural Design
* Detailed Design

**Architectural Design:** Architectural design divides the system into different sub systems known as packages. Then the dependency, relationship and communication between the packages are also identified. Package diagram is use to represent architectural design using UML.

**Detailed Design:** It describes the detailed description of the classes that is all the attributes (Variables and functions). The detailed class diagram represents the detailed design using UML.

**Unified Modeling Language (UML)**

UML, as the name implies, is a modeling language. It may be used to visualize, specify, construct, and document the artifacts of a software system. It provides a set of notations (e.g. rectangles, lines, ellipses, etc.) to create a visual model of the system. Like any other language, UML has its own syntax (symbols and sentence formation rules) and semantics (meanings of symbols and sentences).Also, we should clearly understand that UML is not a system design or development methodology, but can be used to document object-oriented and analysis results obtained using some methodology.

**UML diagrams**

UML can be used to construct nine different types of diagrams to capture five different views of a system. Just as a building can be modeled from several views (or perspectives) such as ventilation perspective, electrical perspective, lighting perspective, heating perspective, etc.; the different UML diagrams provide different perspectives of the software system to be developed and facilitate a comprehensive understanding of the system. Such models can be refined to get the actual implementation of the system.

The UML diagrams can capture the following five views of a system**:**

• User’s view

• Structural view

• Behavioral view

• Implementation view

• Environmental view

**User’s view:** This view defines the functionalities (facilities) made available by the system to its users. The users’ view captures the external users’ view of the system in terms of the functionalities offered by the system. The users’ view is a black-box view of the system where the internal structure, the dynamic behavior of different system components, the implementation etc. are not visible. The users’ view is very different from all other views in the sense that it is a functional model compared to the object model of all other views. The users’ view can be considered as the central view and all other views are expected to conform to this view. This thinking is in fact the crux of any user centric development style.

**Structural view:** The structural view defines the kinds of objects (classes) important to the understanding of the working of a system and to its implementation. It also captures the relationships among the classes (objects). The structural model is also called the static model, since the structure of a system does not change with time.

**Behavioral view:** The behavioral view captures how objects interact with each other to realize the system behavior. The system behavior captures the time-dependent (dynamic) behavior of the system.

**Implementation view:** This view captures the important components of the system and their dependencies.

**Environmental view:** This view models how the different components are implemented on different pieces of hardware.

**User Interface Design**

A system in which the interaction occurs at a level which is understandable to the user will be accepted faster than a system where it is not. A system which is available at irregular intervals or gives incomprehensible error messages is likely to meet resistance. A 1992 survey found that 48% of the code of applications was devoted to the user interface and about 50% of the development time was devoted to implementing that part of the application. Often the user interface is one of the most critical factors as regards to the success or failure of a computerized system. Yet, most software engineers know fairly little about this aspect of our trade. Users judge the quality of a software system by the degree in which it helps them to accomplish their tasks and by the sheer joy they have in using it. This judgment is to a large extent determined by the quality of the user interface. Good user interfaces contribute to a system's quality in the following ways:

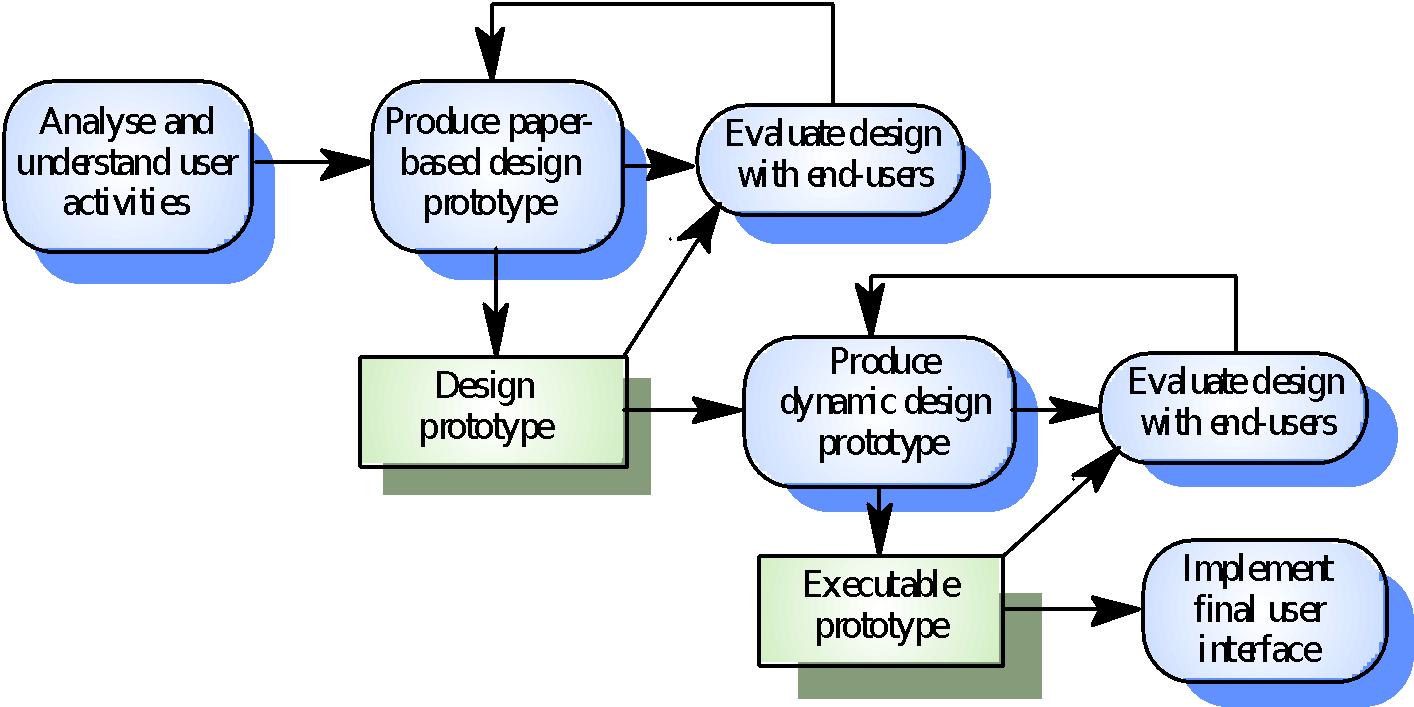
* **Increased efficiency:** If the system fits the way its users work and if it has a good ergonomic design, users can perform their tasks efficiently. They do not lose time struggling with the functionality and its appearance on the screen.
* **Improved productivity:** A good interface does not distract the user, but rather allows him to concentrate on the task to be done.
* **Reduced Errors:** Many so-called 'human errors' can be attributed to poor user interface quality. Avoiding inconsistencies, ambiguities, and so on, reduces user errors.
* **Reduced Training:** A poor user interface hampers learning. A well-designed user interface encourages its users to create proper models and reinforces learning, thus reducing training time.
* **Improved Acceptance:** Users prefer systems whose interface is well-designed. Such systems make information easy to find and provide the information in a form which is easy to use.

**Graphical user interfaces:** Most users of business systems interact with these systems through graphical interfaces although, in some cases, legacy text-based interfaces are still used. A graphical user interface (GUI) is a human-computer interface (i.e., a way for humans to interact with computers) that uses windows, icons and menus and which can be manipulated by a mouse (and often to a limited extent by a keyboard as well).GUIs stand in sharp contrast to command line interfaces (CLIs), which use only text and are accessed solely by a keyboard. The most familiar example of a CLI to many people is MS-DOS. Another example is Linux when it is used in console mode (i.e., the entire screen shows text only).

**GUI advantages**

* They are easy to learn and use.
* Users without experience can learn to use the system quickly.
* The user may switch quickly from one task to another and can interact with several different applications.
* Information remains visible in its own window when attention is switched.
* Fast, full-screen interaction is possible with immediate access to anywhere on the screen.

**User interface design process**



**Figure 7.1 User interface design process**

**UI design principles**

* UI design must take account of the needs, experience and capabilities of the system users
* Designers should be aware of people’s physical and mental limitations (e.g. limited short-term memory) and should recognize that people make mistakes
* UI design principles underlie interface designs although not all principles are applicable to all designs

**Interaction styles**

* Direct manipulation
* Menu selection
* Form fill-in
* Command language
* Natural language

**PRACTICAL No. 8**

**Study various types of 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, but are not limited to, the process of executing a program or application with the intent of finding software bugs (errors or other defects).

Software testing can be stated as the process of validating and verifying that a computer program/application/product:

* meets the requirements that guided its design and development,
* works as expected,
* can be implemented with the same characteristics,
* Satisfies the needs of stakeholders.

Software testing, depending on the testing method employed, can be implemented at any time in the development process. Traditionally most of the test effort occurs after the requirements have been defined and the coding process has been completed, but in the agile approaches most of the test effort is on-going. As such, the methodology of the test is governed by the chosen software development methodology.

* **Verification**

Verification is the process to make sure the product satisfies the conditions imposed at the start of the development phase. In other words, to make sure the product behaves the way we want it to.

* **Validation**

Validation is the process to make sure the product satisfies the specified requirements at the end of the development phase. In other words, to make sure the product is built as per customer requirements.

#### **Basics of software testing**

There are two basics of software testing:

* Blackbox testing
* Whitebox testing

#### **Black-box testing**

Black-box testing treats the software as a "black box", examining functionality without any knowledge of internal implementation. The tester is 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.

**White-box testing**

White-box testing (also known as clear box testing, glass box testing, transparent box testing, and structural testing) is a method of testing software that tests internal structures or workings of an application, as opposed to its functionality (i.e. black-box testing). 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.

White-box test design techniques include:

* Control flow testing
* Data flow testing
* Branch testing
* Path testing
* Statement coverage
* Decision coverage

**Types of testing**

There are many types of testing like

* Unit Testing
* Integration Testing
* Functional Testing
* System Testing
* Stress Testing
* Performance Testing
* Usability Testing
* Acceptance Testing
* Regression Testing
* Alpha Testing
* Beta Testing

**Unit Testing**

Unit testing is the testing of an individual unit or group of related units. It falls under the class of white box testing. It is often done by the programmer to test that the unit he/she has implemented is producing expected output against given input.

**Integration Testing**

Integration testing is testing in which a group of components are combined to produce output. Also, the interaction between software and hardware is tested in integration testing if software and hardware components have any relation. It may fall under both white box testing and black box testing.

**Functional Testing**

Functional testing is the testing to ensure that the specified functionality required in the system requirements works. It falls under the class of black box testing.

**System Testing**

System testing is the testing to ensure that by putting the software in different environments (e.g., Operating Systems) it still works. System testing is done with full system implementation and environment. It falls under the class of black box testing.

**Stress Testing**

Stress testing is the testing to evaluate how system behaves under unfavorable conditions. Testing is conducted at beyond limits of the specifications. It falls under the class of black box testing.

**Performance Testing**

Performance testing is the testing to assess the speed and effectiveness of the system and to make sure it is generating results within a specified time as in performance requirements. It falls under the class of black box testing.

**Usability Testing**

Usability testing is performed to the perspective of the client, to evaluate how the GUI is user-friendly? How easily can the client learn? After learning how to use, how proficiently can the client perform? How pleasing is it to use its design? This falls under the class of black box testing.

**Acceptance Testing**

Acceptance testing is often done by the customer to ensure that the delivered product meets the requirements and works as the customer expected. It falls under the class of black box testing.

**Regression Testing**

Regression testing is the testing after modification of a system, component, or a group of related units to ensure that the modification is working correctly and is not damaging or imposing other modules to produce unexpected results. It falls under the class of black box testing.

**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. The software is released to groups of people so that further testing can ensure the product has few faults or bugs. Sometimes, beta versions are made available to the open public to increase the feedback field to a maximal number of future users.