

DIGITAL NOTES ON APPS DESIGN AND DEVELOPMENT

**B.TECH III YEAR – II SEM
(2017-18)**



DEPARTMENT OF INFORMATION TECHNOLOGY

**MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY
(Autonomous Institution – UGC, Govt. of India)**

(Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2015 Certified)
Maisammaguda, Dhulapally (Post Via. Hakimpet), Secunderabad – 500100, Telangana State, INDIA.



III Year B.Tech. ECE-II Sem

**OPEN ELECTIVE –IV
(R15A0568) APPS DESIGN AND DEVELOPMENT**

Objectives:

1. To have an awareness of software engineering fundamentals and practices.
2. To learn the concepts of scripting languages and multimedia used for application design.
3. To understand the methods of java programming under client/server side and data base connection.

UNIT – I: Fundamental concepts

Software Process models: The waterfall model, Incremental process models, Evolutionary process models. The Unified process. Multimedia and hypermedia, World Wide Web, overview of multimedia software tools, Graphics data types, file formats, color in image and video: color models in images, color in video.

UNIT – II: HTML Common tags

Lists, Tables, Images, Forms, Frames; XML.

UNIT - III : Introduction to Java Scripts

Objects in Java Script, Dynamic HTML with Java Script. Design of GUI.

UNIT - IV : Web Servers

Introduction to Servlets: Lifecycle of a Servlet, The Problem with Servlet. The Anatomy of a JSP Page, JSP Processing, Environment: Installing the Java: Software Development Kit, Tomcat Server. Using Cookies-Session Tracking, Security Issues.

UNIT - V : Database Access

Database Programming using JDBC, Studying Javax.sql.* package, Accessing a Database from a JSP Page, TESTING: Types of software testing ,test cases.

TEXT BOOKS:

1. Web Programming ,Building Internet Applications, CHRIS BATES II Edition, Wiley Dreamtech.
2. Programming world wide web ,SEBESTA,PEARSON.

REFERENCES:

1. Core Servlets And Java Servlets Pages Vol-1:Core Technologies BY MARTY HALL,LARRY BROWN PEARSON.
2. Software Engineering ,ROGER S PRESSMAN,TATA McGraw-HILL.
3. Software Testing Techniques, BORIS BEIZER,DREAMTECH,II EDITION.
4. Java Complete Reference ,7TH EDITION ,HERBERTSCHILDT,TMH.



MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY
DEPARTMENT OF INFORMATION TECHNOLOGY

UNIT – I: Fundamental concepts

Software Process models: The waterfall model, Incremental process models, Evolutionary process models. The Unified process. Multimedia and hypermedia, World Wide Web, overview of multimedia software tools, Graphics data types, file formats, color in image and video: color models in images, color in video.

Software Engineering:

Software Engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

A Layered Technology:

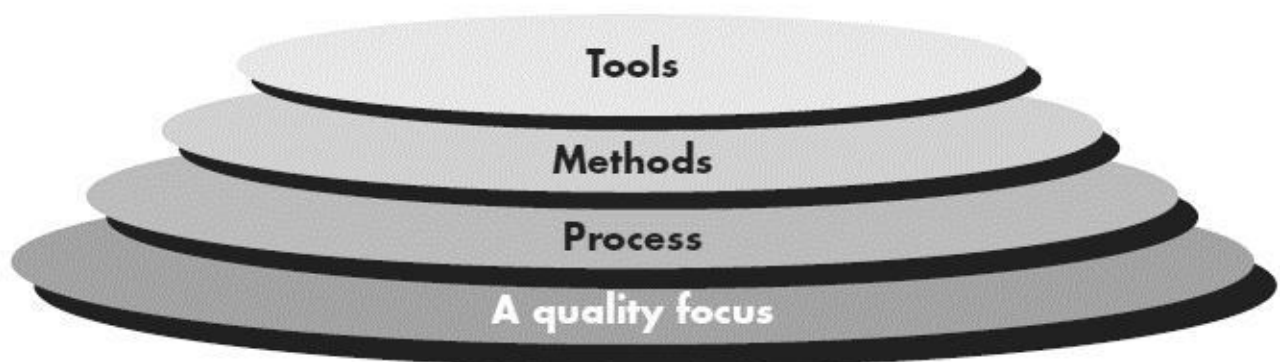


Figure: Software Engineering Layers

A quality Focus:

- Every organization is rest on its **commitment to quality**.
- Total **quality management, Six Sigma, or similar continuous improvement culture** and it is this culture ultimately leads to development of increasingly more effective approaches to software engineering.

Process:

- The software engineering process is the glue that holds the technology layers together and enables rational and timely development of computer software.
- Process defines a framework that must be established for effective delivery of software engineering technology.
- The software process forms the basis for management control of software projects and establishes the context in which technical methods are applied, work products are produced, milestones are established, quality is ensured, and change is properly managed.

Methods:

- Software engineering methods provide the technical aspects for building software.
- Methods encompass a broad array of tasks that include communication, requirements analysis, design modeling, program construction, testing, and support.

Process & Generic Process Model

- A software process is defined as a collection of work activities, actions, and tasks that are performed when some work product is to be created.
- Each of these activities, actions, and tasks reside within a framework or model that defines their relationship with the process and with one another.

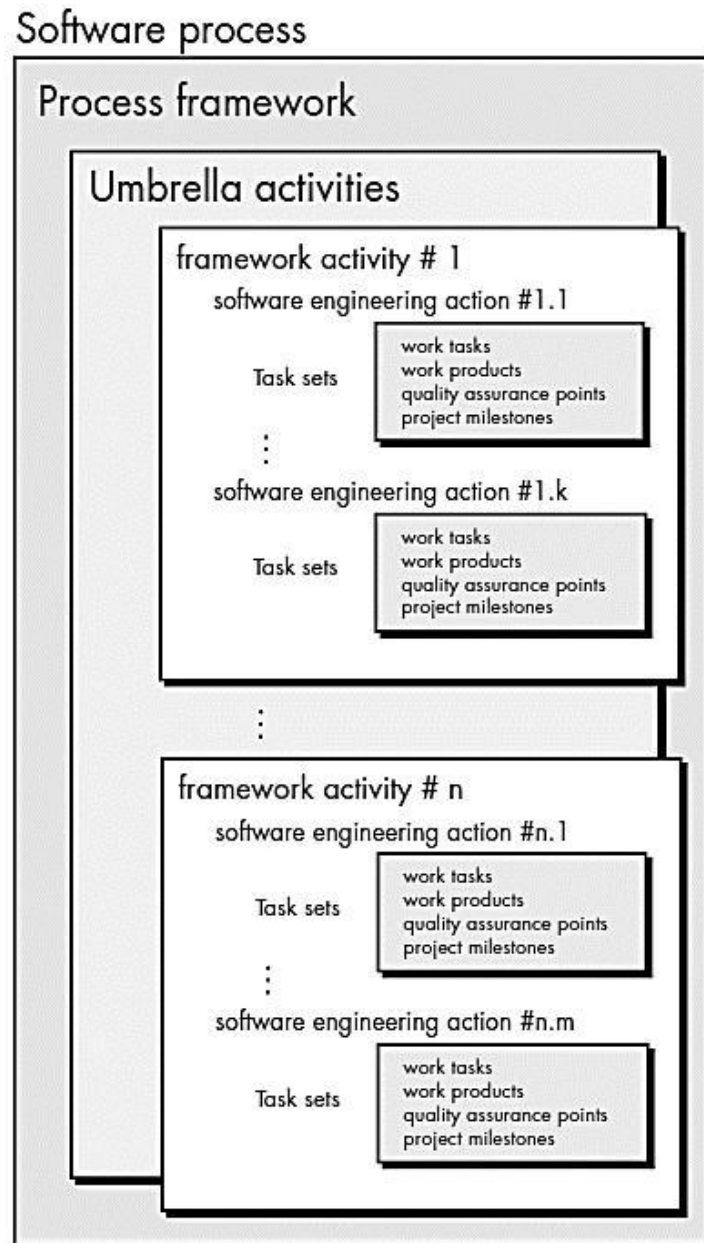
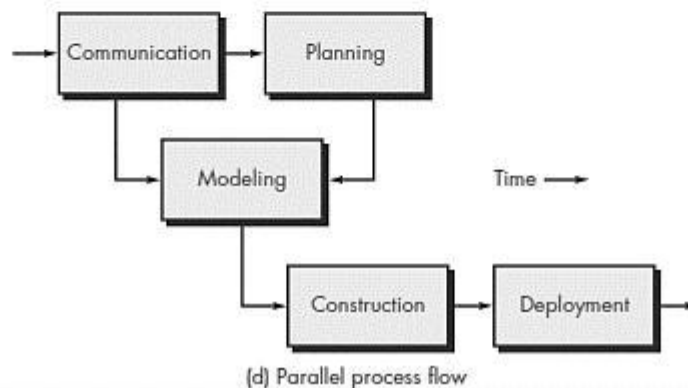
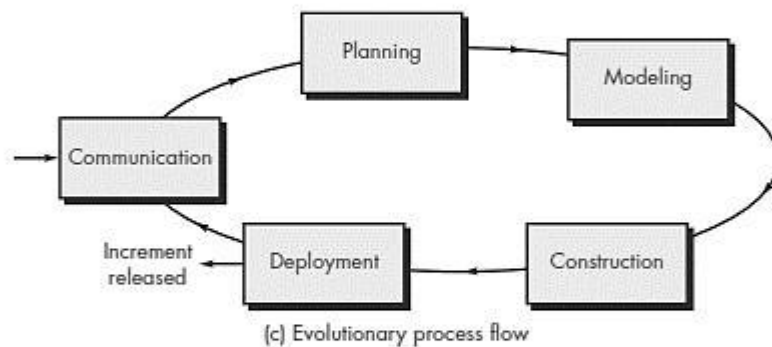
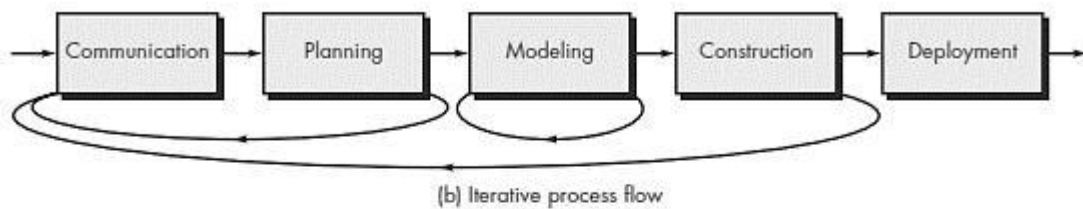
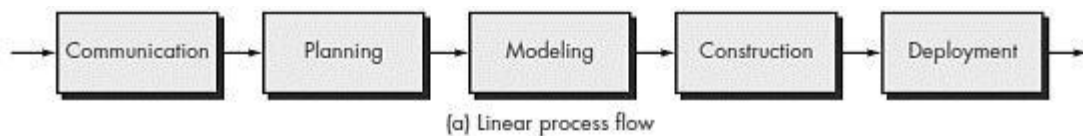


Figure: Generic Process Model

- Each framework activity is populated by a set of software engineering actions.
- Each software engineering action is defined by a task set that identifies the work tasks that are to be completed, the work products that will be produced, the quality assurance points that will be required, and the milestones that will be used to indicate progress.
- A generic process framework for software engineering defines **five framework activities—communication, planning, modeling, construction, and deployment**.
- In addition, it defines a set of umbrella activities.

Process Flow:

Process flow—describes how the framework activities and the actions and tasks that occur within each framework activity are organized with respect to sequence and time.



- A linear process flow executes each of the five framework activities in sequence, beginning with communication and culminating with deployment.
- An iterative process flow repeats one or more of the activities before proceeding to the next.
- An evolutionary process flow executes the activities in a “circular” manner. Each circuit through the five activities leads to a more complete version of the software.
- A parallel process flow executes one or more activities in parallel with other activities.

Waterfall Process Model

The waterfall model, sometimes called the classic life cycle, suggests a systematic, sequential approach to software development that begins with customer specification of requirements and progresses through planning, modeling, construction, and deployment, culminating in ongoing support of the completed software.

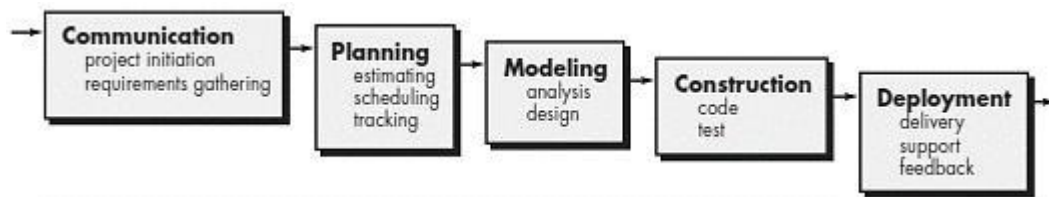


Figure: Waterfall Model

Limitations:

- The nature of the requirements will not change very much during development; during evolution.
- The model implies that you should attempt to complete a given stage before moving on to the next stage.
- Does not account for the fact that requirements constantly change.
- It also means that customers cannot use anything until the entire system is complete.
- The model implies that once the product is finished, everything else is maintenance.
- Surprises at the end are very expensive
- Some teams sit ideal for other teams to finish
- Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.

When to use waterfall model?

- Requirements are very well known, clear and fixed
- Product definition is stable
- Technology is understood
- There are no ambiguous requirements
- Ample resources with required expertise are available freely

Incremental Process Model

- The incremental model combines elements of linear and parallel process flows.
- The incremental model applies linear sequences in a staggered fashion as calendar time progresses.
- Each linear sequence produces deliverable “increments” of the software.
- For example, word-processing software developed using the incremental paradigm might deliver basic file management, editing, and document production functions in the first increment; more sophisticated editing and document production capabilities in the second increment; spelling and grammar checking in the third increment; and advanced page layout capability in the fourth increment.

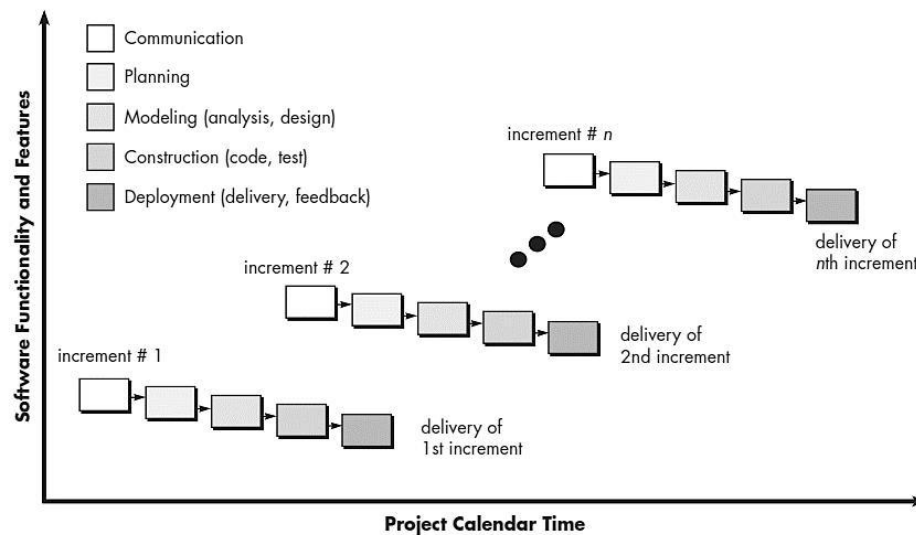


Figure: Incremental Process Model

Advantages:

- Generates working software quickly and early during the software life cycle.
- This model is more flexible – less costly to change scope and requirements.
- It is easier to test and debug during a smaller iteration.
- In this model customer can respond to each built.
- Lowers initial delivery cost.
- Easier to manage risk because risky pieces are identified and handled during iteration.

Disadvantages:

- Needs good planning and design.
- Needs a clear and complete definition of the whole system before it can be broken down and built incrementally.
- Total cost is higher than waterfall.

Evolutionary Process Models

- Evolutionary models are iterative type models.
- They allow to develop more complete versions of the software.

Following are the evolutionary process models.

1. The prototyping model
2. The spiral model
3. Concurrent development model

1. The Prototyping model

Prototype is defined as first or preliminary form using which other forms are copied .

Prototype model is a set of general objectives for software.

It does not identify the requirements like detailed input, output.

It is software working model of limited functionality.

In this model, working programs are quickly produced.

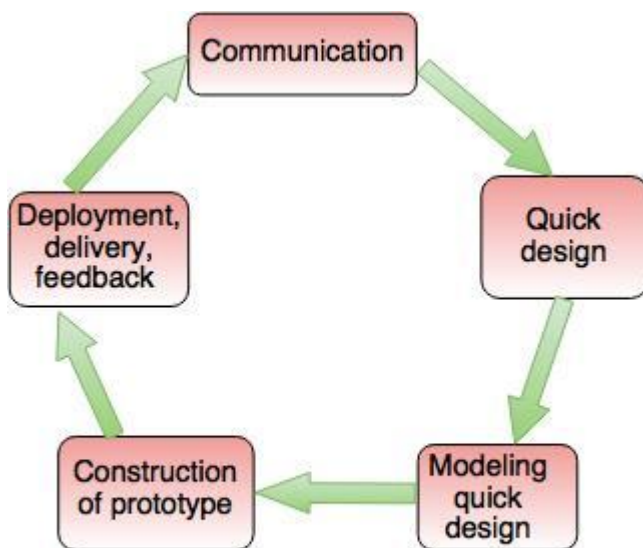


Fig. - The Prototyping Model

The different phases of Prototyping model are:

1. Communication

In this phase, developer and customer meet and discuss the overall objectives of the software.

2. Quick design

Quick design is implemented when requirements are known.

It includes only the important aspects like input and output format of the software.

It focuses on those aspects which are visible to the user rather than the detailed plan.

It helps to construct a prototype.

3. Modeling quick design

This phase gives the clear idea about the development of software because the software is now built.

It allows the developer to better understand the exact requirements.

4. Construction of prototype

The prototype is evaluated by the customer itself.

5. Deployment, delivery, feedback

If the user is not satisfied with current prototype then it refines according to the requirements of the user.

The process of refining the prototype is repeated until all the requirements of users are met.

When the users are satisfied with the developed prototype then the system is developed on the basis of final prototype.

Advantages of Prototyping Model

- Prototype model need not know the detailed input, output, processes, adaptability of operating system and full machine interaction.
- In the development process of this model users are actively involved.
- The development process is the best platform to understand the system by the user.
- Errors are detected much earlier.
- Gives quick user feedback for better solutions.

Disadvantages of Prototyping Model:

- The client involvement is more and it is not always considered by the developer.
- It is a slow process because it takes more time for development.
- Many changes can disturb the rhythm of the development team.

2. The Spiral model

Spiral model is a risk driven process model.

It is used for generating the software projects.

In spiral model, an alternate solution is provided if the risk is found in the risk analysis, then alternate solutions are suggested and implemented.

It is a combination of prototype and sequential model or waterfall model.

In one iteration all activities are done, for large project's the output is small.

The framework activities of the spiral model are as shown in the following figure.

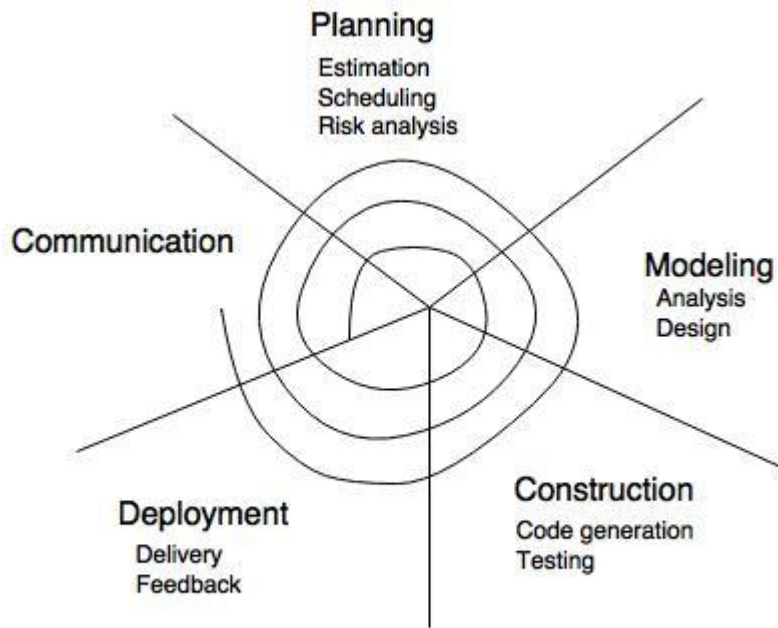


Fig. - The Spiral Model

NOTE: The description of the phases of the spiral model is same as that of the process model.

Advantages of Spiral Model

- It reduces high amount of risk.
- It is good for large and critical projects.
- It gives strong approval and documentation control.
- In spiral model, the software is produced early in the life cycle process.

Disadvantages of Spiral Model

- It can be costly to develop a software model.
- It is not used for small projects.

3. The concurrent development model

- The concurrent development model is called as concurrent model.
- The communication activity has completed in the first iteration and exits in the awaiting changes state.
- The modeling activity completed its initial communication and then go to the underdevelopment state.
- If the customer specifies the change in the requirement, then the modeling activity moves from the under development state into the awaiting change state.
- The concurrent process model activities moving from one state to another state.

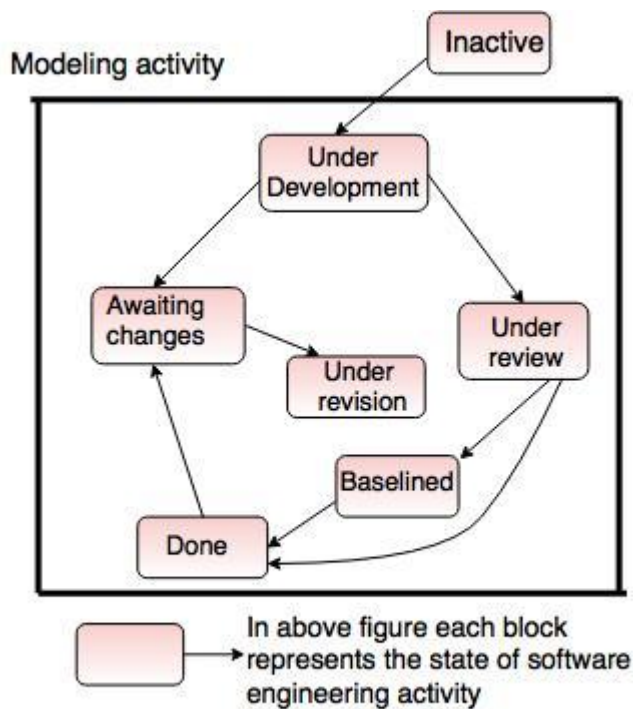


Fig. - One element of the concurrent process model

Advantages of the concurrent development model

- This model is applicable to all types of software development processes.
- It is easy for understanding and use.
- It gives immediate feedback from testing.
- It provides an accurate picture of the current state of a project.

Disadvantages of the concurrent development model

- It needs better communication between the team members.
- It requires to remember the status of the different activities.

THE UNIFIED PROCESS:

The unified process (UP) is an attempt to draw on the best features and characteristics of conventional software process models, but characterize them in a way that implements many of the best principles of agile software development.

The Unified process recognizes the importance of customer communication and streamlined methods for describing the customer's view of a system. It emphasizes the important role of software architecture and "helps the architect focus on the right goals, such as understandability, reliance to future changes, and reuse". It suggests a process flow that is iterative and incremental, providing the evolutionary feel that is essential in modern software development.

A BRIEF HISTORY:

During the 1980s and into early 1990s, object-oriented (OO) methods and programming languages gained a widespread audience throughout the software engineering community. A wide variety of object-oriented analysis (OOA) and design (OOD) methods were proposed during the same time period.

During the early 1990s James Rumbaugh, Grady Booch, and Ival Jacobson began working on a "Unified method" that would combine the best features of each of OOD & OOA. The result was UML- a unified modeling language that contains a robust notation for the modeling and development of OO systems.

By 1997, UML became an industry standard for object-oriented software development. At the same time, the Rational Corporation and other vendors developed automated tools to support UML methods.

Over the next few years, Jacobson, Rumbaugh, and Booch developed the Unified process, a framework for object-oriented software engineering using UML. Today, the Unified process and UML are widely used on OO projects of all kinds. The iterative, incremental model proposed by the UP can and should be adapted to meet specific project needs.

PHASES OF THE UNIFIED PROCESS:

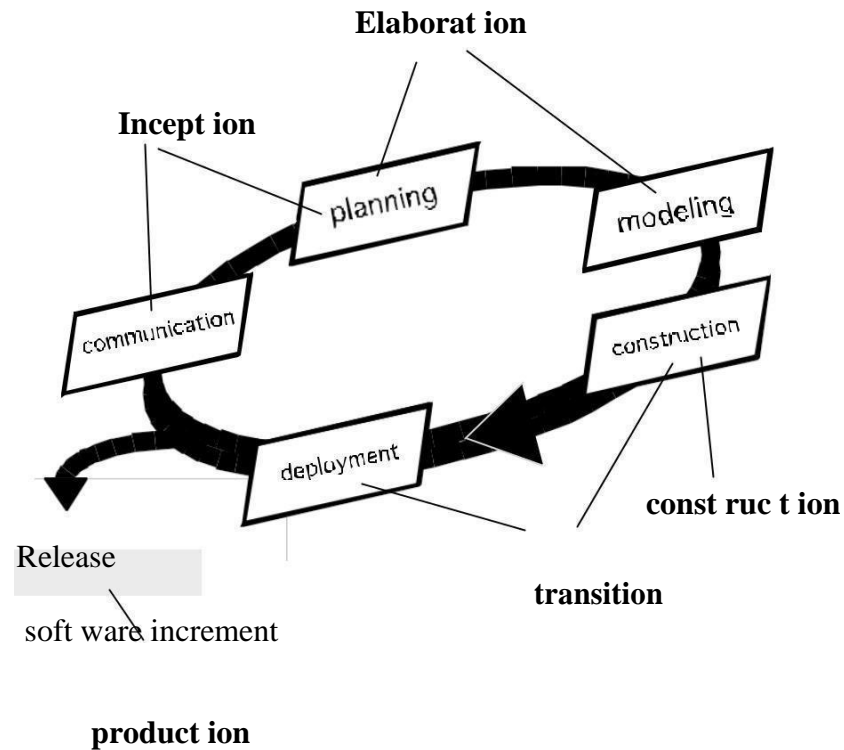
The *inception* phase of the UP encompasses both customer communication and planning activities. By collaborating with the customer and end-users, business requirements for the software are identified, a rough architecture for the system is proposed and a plan for the iterative, incremental nature of the ensuing project is developed.

The *elaboration* phase encompasses the customer communication and modeling activities of the generic process model. Elaboration refines and expands the preliminary use-cases that were developed as part of the inception phase and expands the architectural representation to include five different views of the software- the use-case model, the analysis model, the design model, the implementation model, and the deployment model.

The *construction* phase of the UP is identical to the construction activity defined for the generic software process. Using the architectural model as input, the construction phase develops or acquires the software components that will make each use-case operational for end-users. To accomplish this, analysis and design models that were started during the elaboration phase are completed to reflect the final version of the software increment.

The *transition* phase of the UP encompasses the latter stages of the generic construction activity and the first part of the generic deployment activity. Software given to end-users for beta testing, and user feedback reports both defects and necessary changes.

The *production* phase of the UP coincides with the deployment activity of the generic process. During this phase, the on-going use of the software is monitored, support for the operating environment is provided, and defect reports and requests for changes are submitted and evaluated.



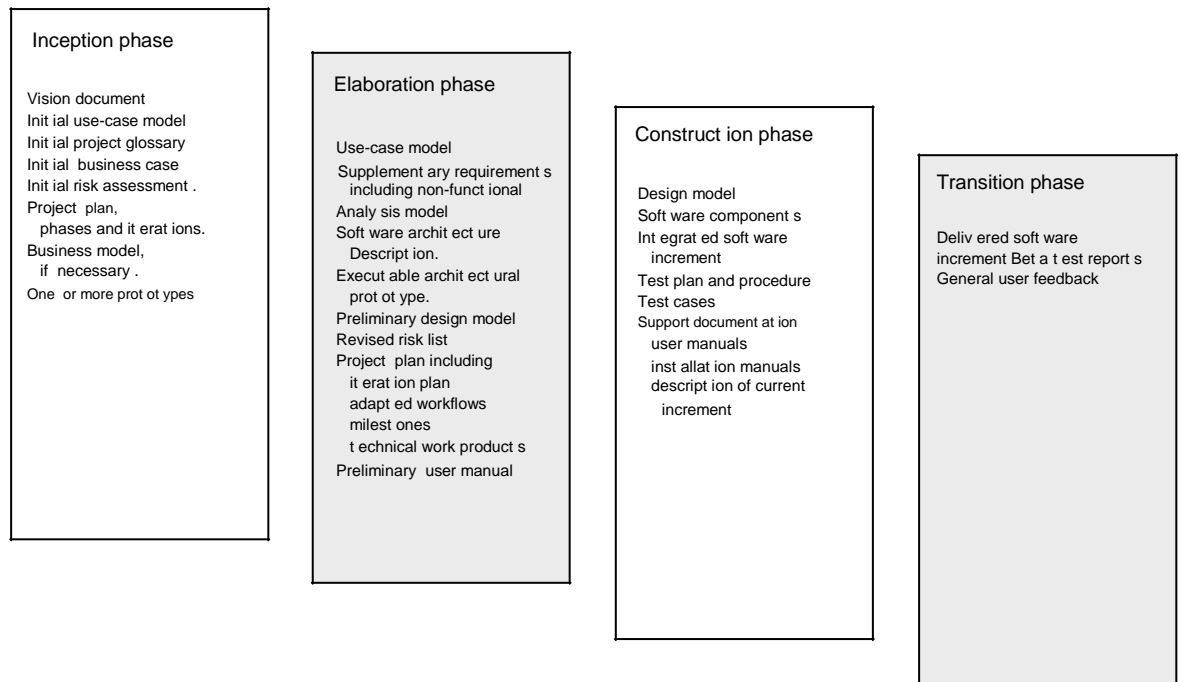
UNIFIED PROCESS WORK PRODUCTS:

During the *inception phase*, the intent is to establish an overall “vision” for the project, identify a set of business requirements, make a business case for the software, and define project and business risks that may represent a threat to success.

The ***elaboration phase*** produces a set of work products that elaborate requirements and produce an architectural description and a preliminary design. The UP analysis model is the work product that is developed as a consequence of this activity. The classes and analysis packages defined as part of the analysis model are refined further into a design model which identifies design classes, subsystems, and the interfaces between subsystems. Both the analysis and design models expand and refine an evolving representation of software architecture. In addition the elaboration phase revisits risks and the project plan to ensure that each remains valid.

The ***construction phase*** produces an implementation model that translates design classes into software components into the physical computing environment. Finally, a test model describes tests that are used to ensure that use cases are properly reflected in the software that has been constructed.

The ***transition phase*** delivers the software increment and assesses work products that are produced as end-users work with the software. Feedback from beta testing and qualitative requests for change is produced at this time.



What is Multimedia?

Multimedia can have a many definitions these include:

Multimedia means that computer information can be represented through audio, video, and animation in addition to traditional media (i.e., text, graphics drawings, images).

A good general definition is:

Multimedia is the field concerned with the computer-controlled integration of text, graphics, drawings, still and moving images (Video), animation, audio, and any other media where every type of information can be represented, stored, transmitted and processed digitally.

A **Multimedia Application** is an Application which uses a collection of multiple media sources e.g. text, graphics, images, sound/audio, animation and/or video.

Hypermedia can be considered as one of the multimedia applications.

Components of Multimedia:

Multimedia involves multiple modalities of text, audio, images, drawings, animation, and video. Examples of how these modalities are put to use:

- Video teleconferencing. □
- Distributed lectures for higher education. □
- Tele-medicine. □
- Co-operative work environments. □
- Searching in (very) large video and image databases for target visual objects. □

Multimedia and Hypermedia:

History of Multimedia:

Brief history of use of Multimedia:

- □ **Newspaper:** the first mass communication medium that uses text, graphics, and images □
- **Motion Pictures:** conceived of in 1830's in order to observe motion too rapid for reception by the human eye. Thomas Alva Edison invented motion picture camera in 1887 □
- **Wireless Radio:** 1895, Guglielmo Marconi sent first radio transmission at Pontecchio, Italy □

Television: the new medium for the 20th century, established video as a commonly available medium and has since changed the world of mass communications.

The connection between computers and ideas about multimedia covers what is actually only a short period:

- 1945: Vannevar Bush wrote a landmark article describing hypermedia system called Memex.
- 1960: Ted Nelson coined the term hypertext.
- 1968: Douglas Engelbart demonstrated the On-Line System (NLS), very early hypertext program.
- 1969: Nelson and van Dam at Brown University created an early hypertext editor called FRESS.
- 1976: MIT Architecture Machine Group proposed a Multiple Media project in Aspen Movie Map
- 1978: First hypermedia videodisk
- 1985: Negroponte and Wiesner co-founded the MIT Media Lab.
- 1989: Tim Berners-Lee proposed the World Wide Web
- 1990: Kristina Hooper Woolsey headed the Apple Multimedia Lab.
- 1991: MPEG-1 was approved as an international standard for digital video later MPEG-2, MPEG-4

The introduction of PDAs in 1991 began a new period in the use of computers in multimedia.

- 1992: JPEG was accepted as international standard for digital image compression later JPEG2000 The first MBone audio multicast on the Net was made.
- 1993: The University of Illinois National Center for Supercomputing Applications produced
- 1994: Jim Clark and Marc Andreessen created the Netscape program.
- 1996: DVD video was introduced; high quality full-length movies were distributed on a single disk.

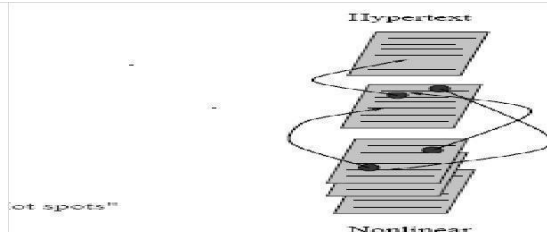
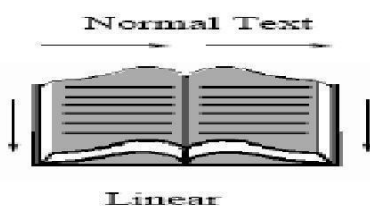
Hypermedia and Multimedia

Ted Nelson invented the term “Hyper Text” around 1965

Types of media:

Linear media: Meant to read from beginning to end. Ex: Text books

Non-linear media: Meant to be read non-linearly, by following links that point to other parts of the document or to other documents
Ex: Hyper Text system



Hypermedia: not constrained to be text-based, can include other media, e.g., graphics, images, and especially the continuous media - sound and video. Examples of Multimedia applications includes: Digital Video edition, E-magazines, WWW, Online reference books, games, home shopping, interactive TV, video conferencing, Interactive Movies. The World Wide Web (WWW) is best example of a hypermedia application.

World Wide Web:

WWW is maintained & developed by World Wide Web Consortium (W3C) and standardized by Internet Engineering Task Force (IETF). The W3C has listed the following goals for the WWW:

- Universal access of web resources .
- Effectiveness of navigating available information.
- Responsible use of posted material.

1) History of the WWW:

- 1960: Charles Goldfarb et al. developed the Generalized Markup Language (GML) for IBM.
- 1986: The ISO released a final version of the Standard Generalized Markup Language (SGML).

1990: Tim Berners-Lee invented the Hyper Text Markup Language (HTML) & Hyper Text Transfer Protocol (HTTP).

- 1993: NCSA released an alpha version of Mosaic based on the version by Marc Andreessen for
- 1994: Marc Andreessen et al. formed Mosaic Communications Corporation later named as Netscape Communications Corporation
- 1998: The W3C accepted XML version 1.0 specifications as a Recommendation. It is the main focus of W3C and supersedes HTML.

HTTP is a protocol that was originally designed for transmitting hypermedia, but can also support the transmission of any file type. HTTP is a stateless request/response protocol: no information carried over for the next request. The basic request format:

The URI (Uniform Resource Identifier): an identifier for the resource accessed, e.g. the host name, always preceded by the token “*http://*”. URL could be Universal Resource Locator, if URI is included with Query strings. Method is a way of exchanging information or performing task on URI. Two popular methods:

GET method that the information requested is in the request string itself

POST method specifies that the resource pointed to URI should consider Message body

Additional header specifies additional parameters about the client. The basic response

format: *Version Status-Code Status-*

Phrase Additional-Headers

Message-body

Status code is number that identifies response type, Status Phrase is textual description of it. Two commonly seen status codes: 200 OK - the request was processed successfully, 404 Not Found - the URI does not exist.

HTML (Hyper Text Markup Language):

HTML is a language for publishing Hypermedia on the World Wide Web - defined using SGML. HTML uses ASCII, it is portable to all different computer hardware. The current version of HTML is version 4.01 in 1999. The next generation of HTML is XHTML - a reformulation of HTML using XML. HTML uses tags to describe document elements:

<token params> - defining a starting point,

</token> - the ending point of the element.

Some elements have no ending tags. A very simple HTML page is as follows:

<HTML>

<HEAD>

<TITLE> A sample web page. </TITLE>

<META NAME = "Author" CONTENT = "Cranky Professor">

</HEAD>

<BODY>

*<P>We can put any text we like here, since this is
a paragraph element.</P>*

</BODY>

</HTML>

Naturally, HTML has more complex structures and can be mixed in with other standards. It allow integration with script languages, dynamic manipulation, modular customization with Cascading Style Sheets (CSS)

Overview of Multimedia Software Tools:

The categories of software tools briefly examined here are:

- Music Sequencing and Notation
 - Digital Audio
 - Graphics and Image Editing
 - Video Editing
 - Animation
 - Multimedia Authoring
-
- **Cakewalk:** now called Pro Audio. The term sequencer comes from older devices that stored sequences of notes ("events", in MIDI). It is also possible to insert WAV files and Windows MCI commands (for animation and video) into music tracks
 - **Cubase:** another sequencing/editing program, with capabilities similar to those of Cakewalk. It includes some digital audio editing tools.
 - **Macromedia Soundedit:** mature program for creating audio for multimedia projects and the web that integrates well with other Macromedia products such as Flash and Director.

Digital Audio:

- Digital Audio tools deal with accessing and editing the actual sampled sounds that make up audio
- **Cool Edit:** a very powerful and popular digital audio toolkit; emulates a professional audio studio – multi track productions and sound file editing including digital signal processing effects.
- **Sound Forge:** a sophisticated PC-based program for editing audio WAV files.
- **Pro Tools:** a high-end integrated audio production and editing environment - MIDI creation and manipulation; powerful audio mixing, recording, and editing software.

Graphics and Image Editing:

- **Adobe Illustrator:** a powerful publishing tool from Adobe. Uses vector graphics; graphics can be exported to Web.

- **Adobe Photoshop:** the standard in a graphics, image processing and manipulation tool. Allows layers of images, graphics, and text that can be separately manipulated for maximum flexibility.
- Filter factory permits creation of sophisticated lighting-effects filters.
- **Macromedia Fireworks:** software for making graphics specifically for the web.
- **Macromedia Freehand:** a text and web graphics editing tool that supports many bitmap formats such as GIF, PNG, and JPEG.
- **Adobe Premiere:** an intuitive, simple video editing tool for nonlinear editing, i.e., putting video clips into any order: Video and audio are arranged in “tracks”. Provides a large number of video and audio tracks, superimpositions and virtual clips. A large library of built-in transitions, filters and motions for clips) effective multimedia productions with little effort.
- **Adobe After Effects:** a powerful video editing tool that enables users to add and change existing movies. Can add many effects: lighting, shadows, motion blurring; layers.
- **Final Cut Pro:** a video editing tool by Apple; Macintosh only.

Animation:

- **Multimedia APIs:**
 - **Java3D:** API used by Java to construct and render 3D graphics, similar to the way in which the Java Media Framework is used for handling media files. Provides a basic set of object primitives (cube, splines, etc.) for building scenes. It is an abstraction layer built on top of OpenGL or
 - **DirectX :** Windows API that supports video, images, audio and 3-D animation
 - **OpenGL:** the highly portable, most popular 3-D API.

- **Rendering Tools:**

- **3D Studio Max:** rendering tool that includes a number of very high-end professional tools for character animation, game development, and visual effects production.
- **Softimage XSI:** a powerful modeling, animation, and rendering package used for animation and special effects in films and games.

Maya: competing product to Softimage; as well, it is a complete modeling package.

Render Man: rendering package created by Pixar.

GIF Animation Packages: a simpler approach to animation, allows very quick development of effective small animations for the web.

Multimedia Authoring:

- **Macromedia Flash:** allows users to create interactive movies by using the score metaphor, i.e., a timeline arranged in parallel event sequences.
- **Macromedia Director:** uses a movie metaphor to create interactive presentations very powerful and includes a built-in scripting language, Lingo, which allows creation of complex interactive movies.
- **Authorware:** a mature, well-supported authoring product based on the conic/Flow-control metaphor.
- **Quest:** similar to Authorware in many ways, uses a type of owcharting metaphor. However, the owchart nodes can encapsulate information in a more abstract way (called frames) than simply subroutine levels.

Graphics data types

There are number of file formats used in multimedia to represent image or graphics data. In general, image or graphics data can be represented as follows:

1) 1-bit Images:

Image Consist of pixels or pels – picture elements in digital images. It contains On(1) or Off(0) bits stored in single bit. So they are also known as Binary image. It is also called as Mono chrome image because it contains no color. 640x480 image Requires 38.4 KB of storage.

2) 8-bit gray Level Images:

Consider 8-bit image, One for which each pixel has Gray value between 0 to 255 stored in single byte. Image is a Two dimensional array known Bitmap. Image resolution refers to number of pixels in digital image like 1600x1200 is high resolution where as 640x480 is low resolution with aspect ration of 4:3. Frame buffer is a hardware used to store array of pixels of image. Special hardware is used for this purpose known as Video/ Graphics card. 8-bit image is a collection of 1-bit bit planes. 640x480 image requires 300 KB of storage.

Dithering:

Printing images is a complex task, 600 Dot per Inch (dpi) laser printer can usually print a dot or not print it. However, 600x600 image will be printed in 1-inch space. Basic strategy of dithering is to trade Intensity resolution for spatial resolution. For printing 1-bit printer, dithering is used to calculate larger patterns of dots. Replace a pixel value by a larger pattern say 2x2, 4x4.

Halftone printing:

Number of printed dots approximates varying sized disks of ink, which is a analog process that uses smaller or larger filled circles of black ink to represent shading. Use NxN matrix of on-off 1- bit dots

3) 24-bit color image:

In color 24-bit images, Each pixel is represented by three bytes, usually representing components R, G, B. Each value is in range 0-255, this format supports 256x256x256 possible combined colors. 640x480 size image takes

921.6 KB of storage. Actually it is stored in 32 bits, extra byte of data for each pixel storing *alpha* value for representing special effect information

4) 8-bit color image:

Accurate color images can be obtained by quantizing color information to 8-bit, Called as 256 color image. 24 bit image is obtained from 8-bit using a concept of Lookup Table.

file formats

1) Graphics Interchange Format(GIF):

GIF was devised by UNISYS for transmitting images over phone lines. It uses Lempel-Ziv-Welch algorithm. It is limited to 8-bit color image only. It produces acceptable color with few distinctive colors. Support It supports Interlacing - successive display of pixels by 4 pass display. GIF comes in two Versions: GIF87a, GIF 89a. it supports simple animation with Graphics Control Extension Block. This provides simple control over delay time, transparency index. GIF file format includes: GIF Signature (6 Bytes), Screen Descriptor (7 Bytes), Global Color Map, Image Information, GIF Terminator. Each image can contain its own color lookup table known as Local color map.

2) Joint Photographic Experts Group(JPEG):

Most important current standard for image compression is JPEG. It was created by ISO. Eye brain system cannot see excrementally fine detail. If many changes occur within few pixels, we refer to that image segment as having High Spatial Frequency i.e. great change in (x,y) space. Color information is partially dropped or averaged & then Small blocks of image are represented in spatial frequency domain (u,v). values are divided by some large integer & truncated. In this way, small values are zeroed out. This compression scheme is lossy. It is straightforward to allow user to choose how large denominator to use & hence how much information to discard. This will allows to choose desired quality of image. Usual default quality factor is $Q=75\%$.

3) Portable Network Graphics(PNG):

It is System independent image format. Motivated by UNISYS on LZW compression method. Special features of PNG files include support 48-bit color information. Files may contain gamma correction, alpha channel information such as channel of transparency. Supports progressive display pixels in two dimensional fashion few at time over seven passes through each 8x8 block of image

4) Tagged Image File Format(TIFF):

It is developed by Aldus Corporation, support Microsoft. It supports attachments of additional information known Tags provides flexibility. Most tags are format signifiers. Different types of images: 1-bit, gray scale, 8-bit, 24-bit are supported.

5) Exchange Image File(EXIF):

It is image format for Digital cameras, published in 2002 by JEITA. Compressed EXIF files are use baseline JPEG format. Variety of tags available for higher quality printing. Picture taking conditions: light source, white balance. It also includes specification of file format for audio that accompanies digital images.

6) Graphics Animation Files:

Few formats are aimed at storing graphics animations. FLC is important animation or moving picture file format. It was originally created by Animation Pro. GL produces some what better quality moving pictures. GL animations can also usually handle larger file sizes.

7) PS & PDF:

Post Script is language for typesetting & many high end printers have PS interpreter built into them. PS is vector based, picture language. Page elements are essentially defined in terms of vectors. It includes Text as well as vector/structured graphics, bit mapped images. PS does not provide compression it self, are just stored as ASCII. Portable Document Format(PDF) includes Text and Figure language with LZW compressing method. Provide higher compression with JPEG compression.

8) Windows WMF:

Windows Meta File is native vector file format. It is collection of Graphics Device Interface(GDI) function calls. Device independent & unlimited in size.

9) Windows BMP:

Bitmap is system standard for Microsoft windows. It uses Run length encoding compression & can fairly Store 24 bit bitmap image. BMP has many different modes including uncompressed 24 bit images.

10) Macintosh PAINT & PICT:

PAINT used in MacPaint program only for 1 bit monochrome images. PICT used in MacDraw structured graphics.

11) X Windows PPM:

For X Window system, Portable Pix Map support 24 bit color bitmap & can be manipulated using many public domain graphic editors.

Color in image and video: color models in images, color in video.

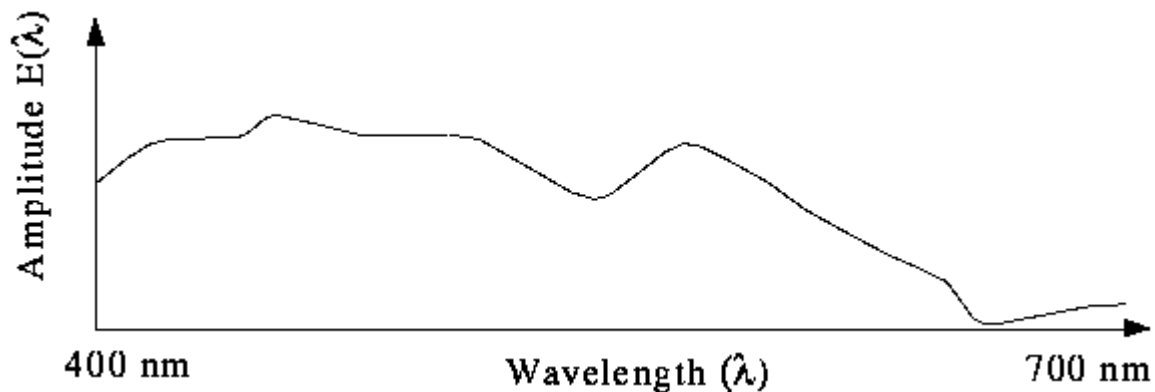
Color in Image and Video

Basics of Color

Light and Spectra

- Visible light is an electromagnetic wave in the 400 nm - 700 nm range.

Most light we see is not one wavelength, it's a combination of many wavelengths.



- The profile above is called a *spectral power distribution* or *spectrum*.

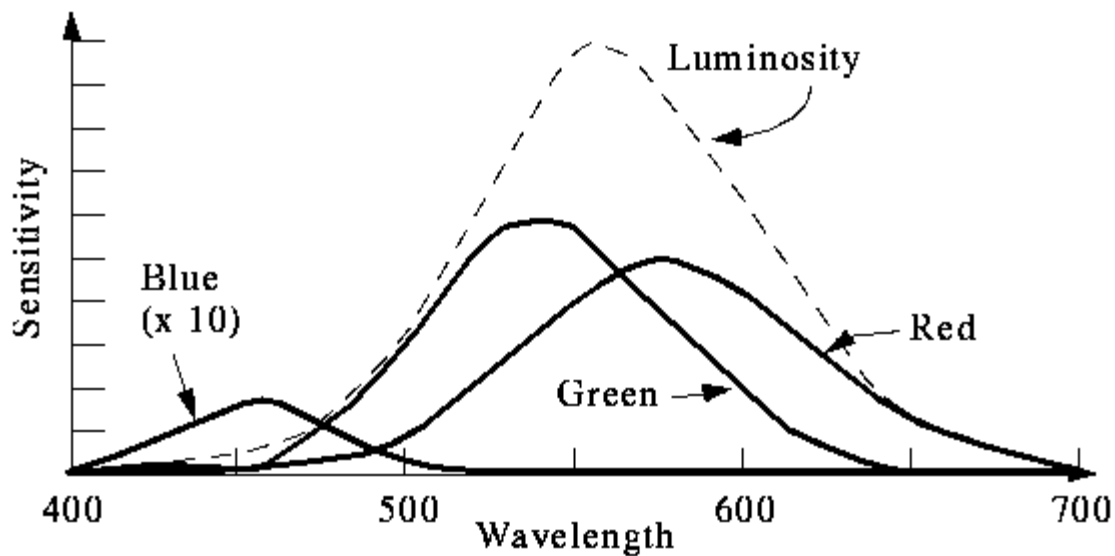
The Human Retina

- The eye is basically just a camera

Each neuron is either a *rod* or a *cone*. Rods are not sensitive to color.

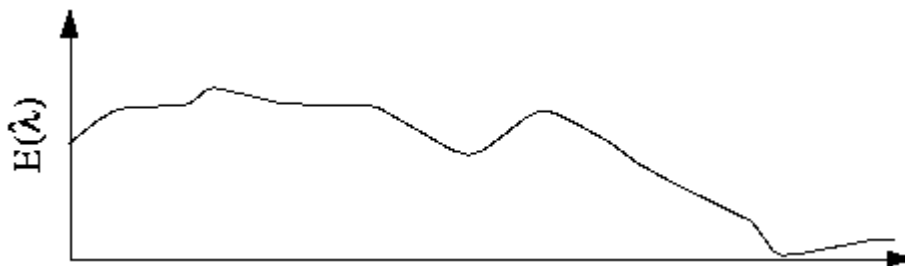
Cones and Perception

- Cones come in 3 types: red, green and blue. Each responds differently to various frequencies of light. The following figure shows the spectral sensitivity functions of the cones and the luminous-efficiency function of the human eye.



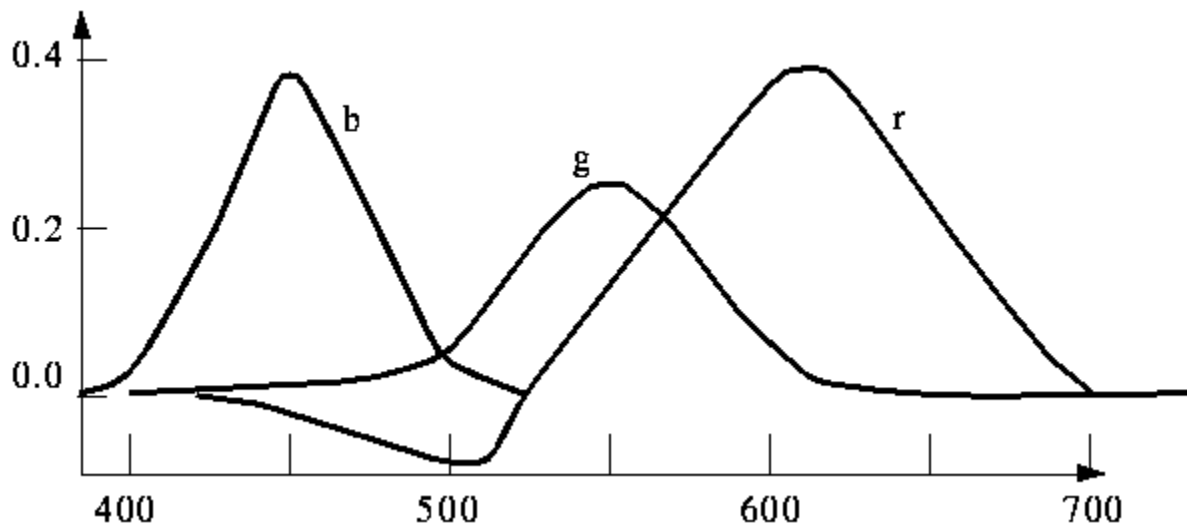
- The color signal to the brain comes from the response of the 3 cones to the spectra being observed. That is, the signal consists of 3 numbers:

$$G = \int E(\lambda) S_G(\lambda) d\lambda$$



where E is the light (spectral power distribution) and S are the spectral sensitivity functions.

- A color can be specified as the sum of three colors. So colors form a 3 dimensional vector space.
- The following figure shows the amounts of three primaries needed to match all the wavelengths of the visible spectrum.

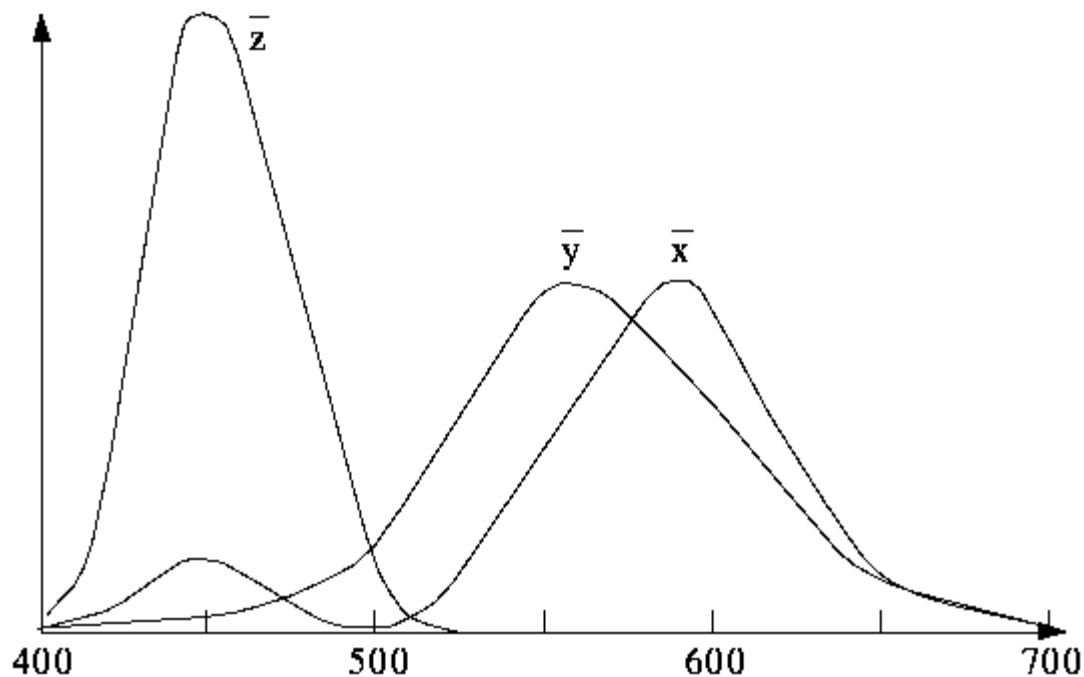


- The negative value indicates that some colors cannot be exactly produced by adding up the primaries.

CIE Chromaticity Diagram

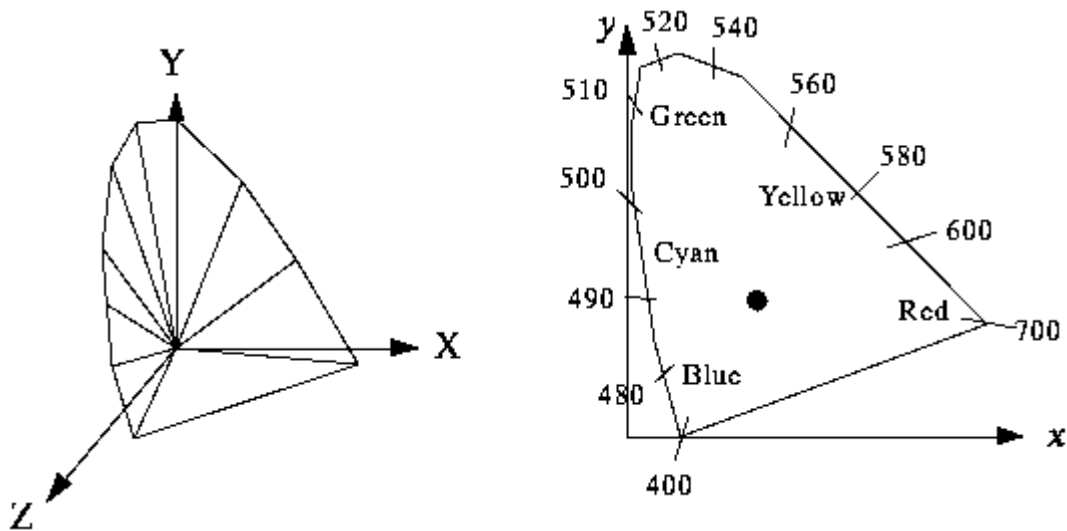
- Q: Does a set of primaries exist that span the space with only positive coefficients?
- A: Yes, but no pure colors.

In 1931, the CIE (Commission Internationale de L'Eclairage, or International Commission on Illumination) defined three standard primaries (\bar{X} , \bar{Y} , \bar{Z}). The \bar{Y} primary was intentionally chosen to be identical to the luminous-efficiency function of human eyes.



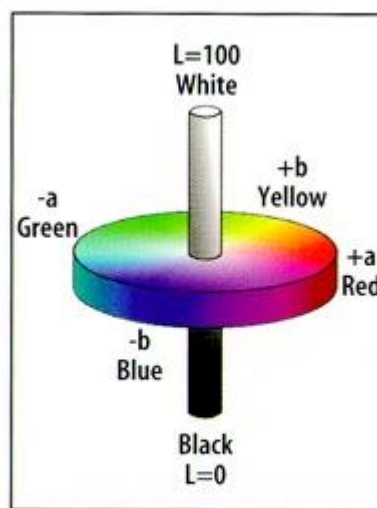
- The above figure shows the amounts of \bar{X} , \bar{Y} , \bar{Z} needed to exactly reproduce any visible color.

- All visible colors are in a "horseshoe" shaped cone in the X-Y-Z space. Consider the plane $X+Y+Z=1$ and project it onto the X-Y plane, we get the *CIE chromaticity diagram* as below.



- The edges represent the "pure" colors (sine waves at the appropriate frequency)
- White (a blackbody radiating at 6447 kelvin) is at the "dot"
- When added, any two colors (points on the CIE diagram) produce a point on the line between them.
- Q: how can we find a color's complement on the CIE diagram?

L*a*b (Lab) Color Model



Lab model

- A refined CIE model, named CIE L*a*b in 1976

- Luminance: L

Chrominance: a -- ranges from green to red, b -- ranges from blue to yellow

- Used by *Photoshop*

Color Models in Images

- A color image is a 2-D array of (R,G,B) integer triplets. These triplets encode how much the corresponding phosphor should be excited in devices such as a monitor.

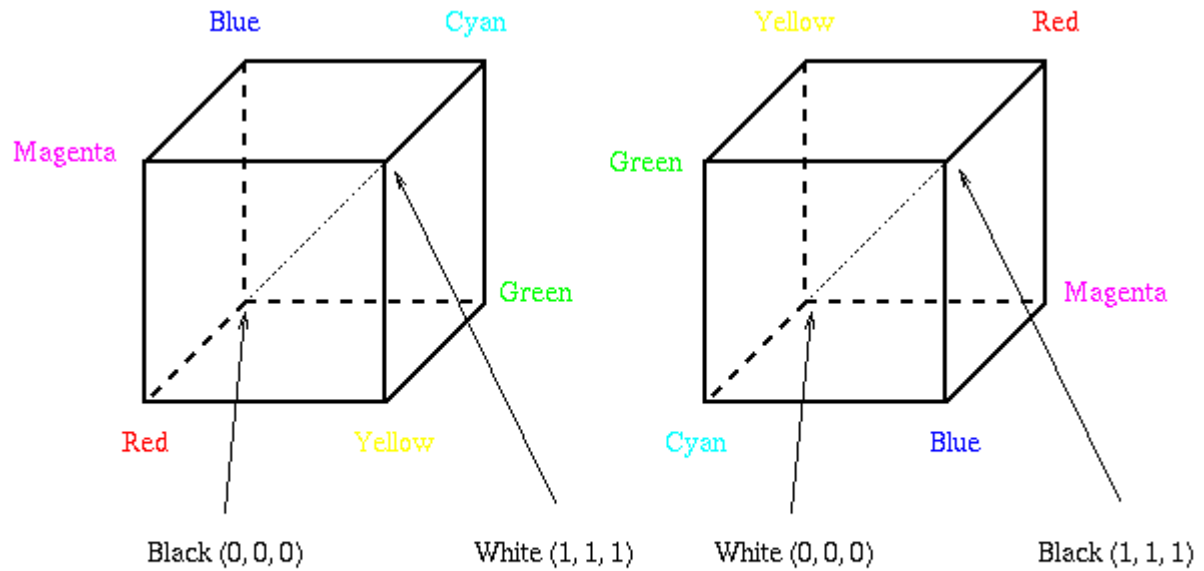
RGB Color Model for CRT Displays

- CRT displays have three phosphors (RGB) which produce a combination of wavelengths when excited with electrons.



CMY Color Model

- Cyan, Magenta, and Yellow (CMY) are complementary colors of RGB. They can be used as *Subtractive Primaries*.
- CMY model is mostly used in printing devices where the color pigments on the paper absorb certain colors (e.g., no red light reflected from cyan ink).



The RGB Cube

The CMY Cube

The RGB and CMY Cubes

Conversion between RGB and CMY:

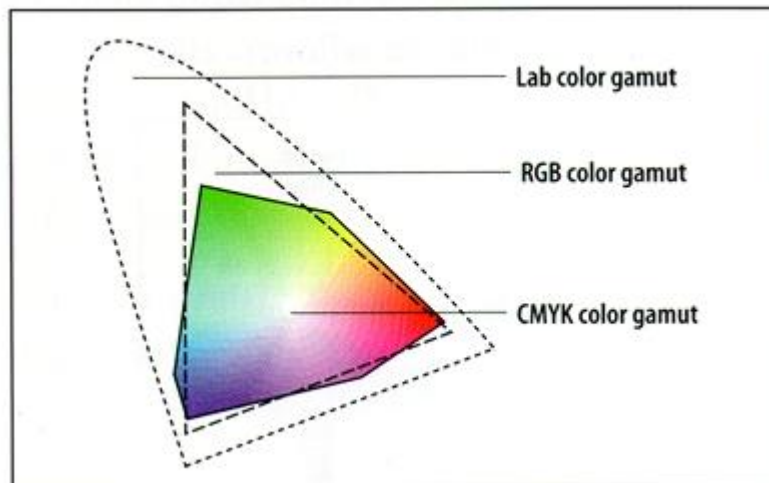
-- e.g., convert **White** from (1, 1, 1) in RGB to (0, 0, 0) in CMY.

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} C \\ M \\ Y \end{bmatrix}$$

- Sometimes, an alternative CMYK model (K stands for *Black*) is used in color printing (e.g., to produce darker black than simply mixing CMY).
 - $K := \min(C, M, Y)$, $C := C - K$, $M := M - K$, $Y := Y - K$.

Comparison of Three Color Gamuts



- The *gamut* of colors is all colors that can be reproduced using the three primaries
- The Lab gamut covers all colors in visible spectrum
- The RGB gamut is smaller, hence certain visible colors (e.g. pure yellow, pure cyan) cannot be seen on monitors
- The CMYK gamut is the smallest (but not a straight subset of the RGB gamut)

Color Models in Video

- YIQ and YUV are the two commonly used color models in video

YUV Color Model

- Initially, for PAL analog video, it is now also used in CCIR 601 standard for digital video
- Y (luminance) is the CIE Y primary.

$$Y = 0.299R + 0.587G + 0.114B$$

- *Chrominance* is defined as the difference between a color and a reference white at the same luminance. It can be represented by U and V -- the *color differences*.

$$U = B - Y$$

$$V = R - Y$$

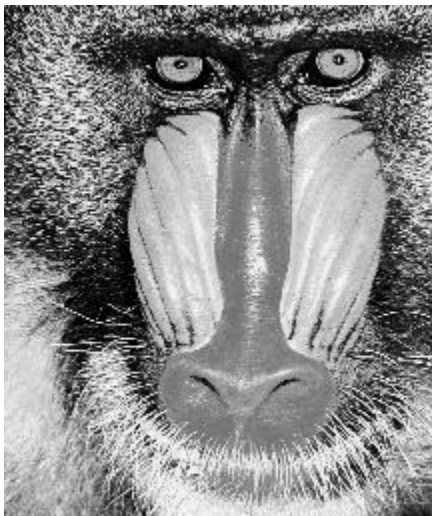
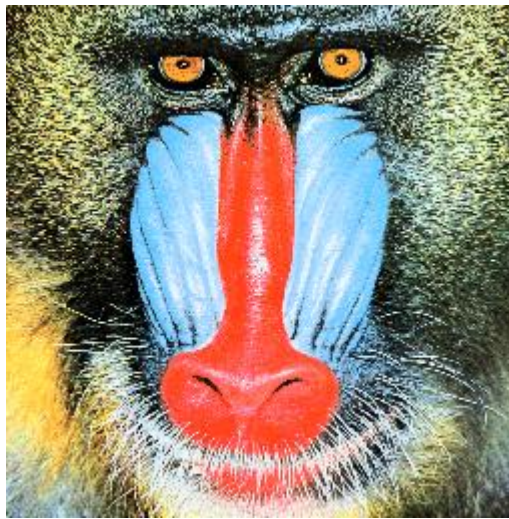
- If b/w image, then $U = V = 0$. --> No chrominance!

- ** In actual PAL implementation:

$$U = 0.492 (B - Y)$$

$$V = 0.877 (R - Y)$$

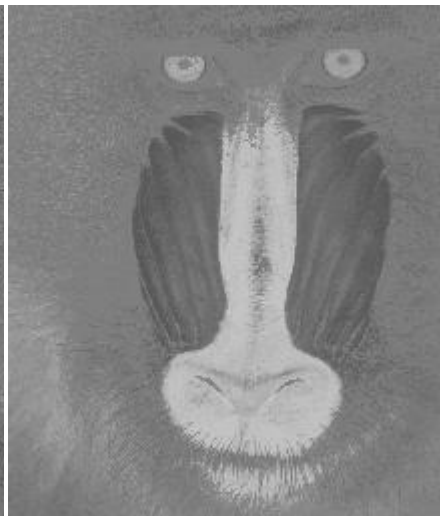
- Sample YUV Decomposition:



Y



U



V

- Eye is most sensitive to Y. In PAL, 5 (or 5.5) MHz is allocated to Y, 1.3 MHz to U and V.

YCbCr Color Model

- The YCbCr model is closely related to the YUV, it is a scaled and shifted YUV.

$$Cb = (B - Y) / 1.772 + 0.5$$

$$Cr = (R - Y) / 1.402 + 0.5$$

- The chrominance values in YCbCr are always in the range of 0 to 1.
- YCbCr is used in JPEG and MPEG.

YIQ Color Model

- YIQ is used in NTSC color TV broadcasting, it is downward compatible with B/W TV where only Y is used.
- Although U and V nicely define the color differences, they do not align with the desired human perceptual color sensitivities. In NTSC, I and Q are used instead.

I is the orange-blue axis, Q is the purple-green axis.

I and Q axes are scaled and rotated R - Y and B - Y (by 33 degrees clockwise).

$$I = 0.877(R - Y) \cos 33 - 0.492(B - Y) \sin 33$$

$$Q = 0.877(R - Y) \sin 33 + 0.492(B - Y) \cos 33$$

Namely,

$$I = 0.736(R - Y) - 0.268(B - Y) = 0.596R - 0.275G - 0.321B$$

$$Q = 0.478(R - Y) + 0.413(B - Y) = 0.212R - 0.523G + 0.311B$$

- The YIQ transform:

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

- Eye is most sensitive to Y, next to I, next to Q.

In NTSC broadcast TV, 4.2 MHz is allocated to Y, 1.5 MHz to I and 0.55 MHz to Q. For VCR, Y is cut down to 3.2 MHz and I to 0.63 MHz.

Summary

- Color images are encoded as triplets of values.
- RGB is an additive color model that is used for light-emitting devices, e.g., CRT displays

CMY is a subtractive model that is used often for printers

- Two common color models in imaging are RGB and CMY, two common color models in video are YUV and YIQ.
- YUV uses properties of the human eye to prioritize information. Y is the black and white (luminance) image, U and V are the color difference (chrominance) images. YIQ uses similar idea.
- Besides the hardware-oriented color models (i.e., RGB, CMY, YUV, YIQ), HSB (Hue, Saturation, and Brightness) and HLS (Hue, Lightness, and Saturation) are also commonly used.

UNIT – II: HTML Common tags

Lists, Tables, Images, Forms, Frames; XML.

HTML Common tags:-

HTML is the building block for web pages. HTML is a format that tells a computer how to display a web page. The documents themselves are plain text files with special "tags" or codes that a web browser uses to interpret and display information on your computer screen.

HTML stands for Hyper Text Mark-up Language

An HTML file is a text file containing small markup tags

The mark-up tags tell the Web browser how to display the page

An HTML file must have an htm or html file extension.

HTML Tags:- HTML tags are used to mark-up HTML elements .HTML tags are surrounded by the two characters < and >. The surrounding characters are called angle brackets. HTML tags normally come in pairs like **and** The first tag in a pair is the start tag, the second tag is the end tag . The text between the start and end tags is the element content . HTML tags are not case sensitive, **means the same as** .

The most important tags in HTML are tags that define headings, paragraphs and line breaks.

Tag	Description
<!DOCTYPE...>	This tag defines the document type and HTML version.
<html>	This tag encloses the complete HTML document and mainly comprises of document header which is represented by <head>...</head> and document body which is represented by <body>...</body> tags.
<head>	This tag represents the document's header which can keep other HTML tags like <title>, <link> etc.
<title>	The <title> tag is used inside the <head> tag to mention the document title.
<body>	This tag represents the document's body which keeps other HTML tags like <h1>, <div>, <p> etc.
<p>	This tag represents a paragraph.
<h1> to <h6>	Defines header 1 to header 6
 	Inserts a single line break
<hr>	Defines a horizontal rule
<!-->	Defines a comment

Headings:-

Headings are defined with the <h1> to <h6> tags. <h1> defines the largest heading while <h6> defines the smallest.

<h1>This is a heading</h1>

<h2>This is a heading</h2>

<h3>This is a heading</h3>

<h4>This is a heading</h4>

<h5>This is a heading</h5>

<h6>This is a heading</h6>

Paragraphs:-

Paragraphs are defined with the <p> tag. Think of a paragraph as a block of text. You can use the align attribute with a paragraph tag as well.

<p align="left">This is a paragraph</p>
 <p align="center">this is another paragraph</p>

Note: You must indicate paragraphs with <p> elements. A browser ignores any indentations or blank lines in the source text. Without <p> elements, the document becomes one large paragraph. HTML automatically adds an extra blank line before and after a paragraph.

The
 tag is used when you want to start a new line, but don't want to start a new paragraph. The
 tag forces a line break wherever you place it. It is similar to single spacing in a document.

This Code	output
<p>This is a para graph with line breaks</p>	This is a para graph with line breaks

Horizontal Rule The element is used for horizontal rules that act as dividers between sections like this:

The horizontal rule does not have a closing tag. It takes attributes such as align and width

Code	Output
<hr width="50%" align="center">	<hr/>

Sample html program

```
<!DOCTYPE html>
<html>
  <head>
    <title>This is document title
    </title>
  </head>
  <body>
    <h1>This is a heading</h1>
    <p>Document content goes here.....</p>
  </body>
</html>
```



- 1) Type the above program in notepad and save with some file name eg:sample.html
- 2) Open the file with browser and the webpage looks like this

Lists:-HTML offers web authors three ways for specifying lists of information.

All lists must contain one or more list elements. Lists are of three types

- 1)Un ordered list
- 2)Ordered List
- 3)Definition list

HTML Unordered Lists:An unordered list is a collection of related items that have no special order or sequence. This list is created by using HTML tag. Each item in the list is marked with a bullet.

Example

```

<!DOCTYPE html>
<html>
  <head>
    <title>HTML Unordered List</title>
  </head>
  <body>
    <ul>
      <li>Beetroot</li>
      <li>Ginger</li> <li>Potato</li>
      <li>Radish</li>
    </ul>
  </body>
</html>

```



- Beetroot
- Ginger
- Potato
- Radish

HTML Ordered Lists:- items are numbered list instead of bulleted, This list is created by using **** tag.

```

<!DOCTYPE html>
<html>
  <head>
    <title>HTML Ordered List</title>
  </head>
  <body>
    <ol>
      <li>Beetroot</li>
      <li>Ginger</li>
      <li>Potato</li>
      <li>Radish</li>
    </ol>
  </body>
</html>

```



1. Beetroot
2. Ginger
3. Potato
4. Radish

HTML Definition Lists:- HTML and XHTML supports a list style which is called definition lists where entries are listed like in a dictionary or encyclopedia. The definition list is the ideal way to present a glossary, list of terms, or other name/value list. Definition List makes use of following three tags.

- 1). <dl> - Defines the start of the list
- 2). <dt> - A term
- 3). <dd> - Term definition
- 4). </dl> - Defines the end of the list

```

<!DOCTYPE html>
<html>
  <head>
    <title>HTML Definition List</title>
  </head>
  <body>
    <dl>
      <dt><b>HTML</b></dt> <dd>This stands for Hyper Text Markup Language</dd>
      <dt><b>HTTP</b></dt> <dd>This stands for Hyper Text Transfer Protocol</dd>
    </dl>
  </body>
</html>

```

HTML

This stands for Hyper Text Markup Language

HTTP

This stands for Hyper Text Transfer Protocol

HTML tables:

The HTML tables allow web authors to arrange data like text, images, links, other tables, etc. into rows and columns of cells. The HTML tables are created using the **<table>** tag in which the **<tr>** tag is used to create table rows and **<td>** tag is used to create data cells.

Example:

```

<!DOCTYPE html>
<html>
<head>
<title>HTML Tables</title>
</head>
<body>
  <table border="1">
    <tr>
      <td>Row 1, Column 1</td> <td>Row 1, Column 2</td>
    </tr>
    <tr>
      <td>Row 2, Column 1</td> <td>Row 2, Column 2</td>
    </tr>
  </table>
</body>
</html>

```

Row 1, Column 1	Row 1, Column 2
Row 2, Column 1	Row 2, Column 2

Table Heading: Table heading can be defined using **<th>** tag. This tag will be put to replace **<td>** tag, which is used to represent actual data cell. Normally you will put your top row as table heading as shown below, otherwise you can use **<th>** element in any row.

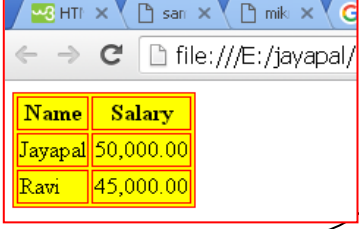
Tables Backgrounds: set table background using one of the following two ways:

- 1)bgcolor attribute - You can set background color for whole table or just for one cell.
- 2)background attribute - You can set background image for whole table or just for one cell. You can also set border color also using bordercolor attribute.

```

<!DOCTYPE html>
<html>
<head>
<title>HTML Tables</title> </head>
<body>
    <table border="1"bordercolor="red" bgcolor="yellow">
        <tr> <th>Name</th>
        <th>Salary</th> </tr>
        <tr>
            <td>Jayapal    </td> <td>50,000.00</td>
        </tr>
        <tr> <td>Ravi</td> <td>45,000.00</td>
        </tr>
    </table>
</body>
</html>

```



Name	Salary
Jayapal	50,000.00
Ravi	45,000.00

Images are very important to beautify as well as to depict many complex concepts in simple way on your web page.

Insert Image:

insert any image in the web page by using **** tag.

Attribute Values

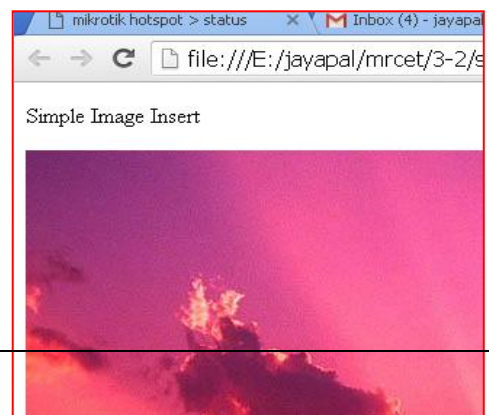
Value	Description
left	Align the image to the left
right	Align the image to the right
middle	Align the image in the middle
top	Align the image at the top

Example

```

<!DOCTYPE html>
<html>
    <head>
        <title>Using Image in Webpage</title>
    </head>
    <body> <p>Simple Image Insert</p>
        
    </body>

```



HTML FORMS:

HTML Forms are required to collect some data from the site visitor. For example, during user registration you would like to collect information such as name, email address, credit card, etc. A form will take input from the site visitor and then will post it to a back-end application such as CGI, ASP Script or PHP script etc. The back-end application will perform required processing on the passed data based on defined business logic inside the application. There are various form elements available like text fields, text area fields, drop-down menus, radio buttons, checkboxes, etc.

```
<form action="Script URL" method="GET|POST"> form elements like input, text area etc. </form>
```

Form Attributes

Apart from common attributes, following is a list of the most frequently used form attributes:

Attribute	Description
action	Backend script ready to process your passed data.
method	Method to be used to upload data. The most frequently used are GET and POST methods.
target	Specify the target window or frame where the result of the script will be displayed. It takes values like _blank, _self, _parent etc.
enctype	<p>You can use the enctype attribute to specify how the browser encodes the data before it sends it to the server. Possible values are:</p> <p>application/x-www-form-urlencoded - This is the standard method most forms use in simple scenarios.</p> <p>multipart/form-data - This is used when you want to upload binary data in the form of files like image, word file etc.</p>

HTML Form Controls :

There are different types of form controls that you can use to collect data using HTML form:

- Text Input Controls
- Checkboxes Controls
- Radio Box Controls
- Select Box Controls
- File Select boxes
- Hidden Controls
- Clickable Buttons
- Submit and Reset Button

Text Input Controls:-

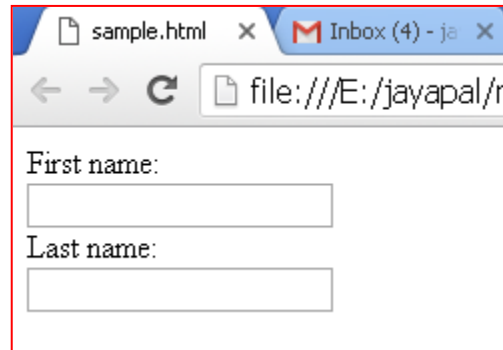
There are three types of text input used on forms:

- 1) **Single-line text input controls** - This control is used for items that require only one line of user input, such as search boxes or names. They are created using HTML `<input>` tag.

`<input type="text">` defines a one-line input field for **text input**:

Example:

```
<form>
  First name:<br>
  <input type="text" name="firstname"><br>
  Last name:<br>
  <input type="text" name="lastname">
</form>
```

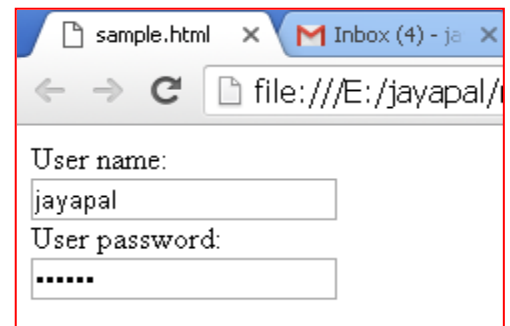


- 2) **Password input controls** - This is also a single-line text input but it masks the character as soon as a user enters it. They are also created using HTML `<input>` tag.

Input Type Password

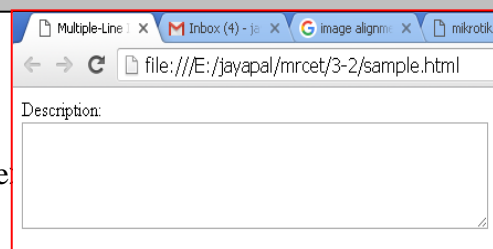
`<input type="password">` defines a **password field**:

```
<form>
  User name:<br>
  <input type="text" name="username"><br>
  User password:<br>
  <input type="password" name="psw">
</form>
```



- 3) **Multi-line text input controls** - This is used when the user is required to give details that may be longer than a single sentence. Multi-line input controls are created using HTML `<textarea>` tag.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Multiple-Line Input Control</title>
  </head>
  <body>
    <form> Description: <br />
      <textarea rows="5" cols="50" name="description"> Enter description here... </textarea>
    </form>
  </body>
</html>
```

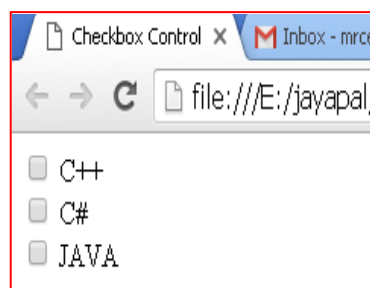


Checkboxes Controls:-

Checkboxes are used when more than one option is required to be selected. They are also created using HTML `<input>` tag but type attribute is set to checkbox.

Here is an example HTML code for a form with two checkboxes:

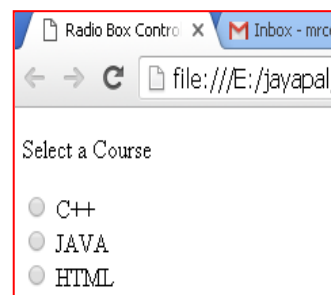
```
<!DOCTYPE html>
<html> <head> <title>Checkbox Control</title> </head>
<body>
  <form>
    <input type="checkbox" name="C++" value="on"> C++
    <br>
    <input type="checkbox" name="C#" value="on"> C#
    <br>
    <input type="checkbox" name="JAVA" value="on"> JAVA
  </form>
</body> </html>
```



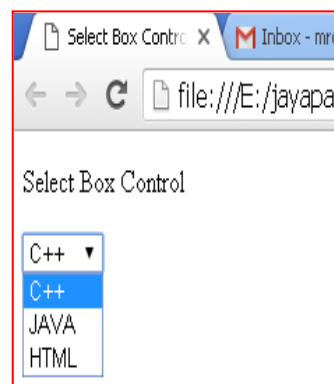
Radio Button Control:-

Radio buttons are used when out of many options, just one option is required to be selected. They are also created using HTML `<input>` tag but type attribute is set to radio.

```
<!DOCTYPE html>
<html> <head> <title>Radio Box Control</title> </head>
<body> <p>Select a Course</p>
  <form>
    <input type="radio" name="subject" value="C++"> C++
    <br>
    <input type="radio" name="subject" value="JAVA"> JAVA
    <br>
    <input type="radio" name="subject" value="HTML"> HTML
  </form> </body> </html>
```



```
<!DOCTYPE html>
<html>
<head>
  <title>Select Box Control</title>
</head>
<body>
  <form>
    <select name="dropdown">
      <option value="C++" selected>C++</option>
      <option value="JAVA">JAVA</option>
      <option value="HTML">HTML</option>
    </select>
  </form>
</body>
</html>
```

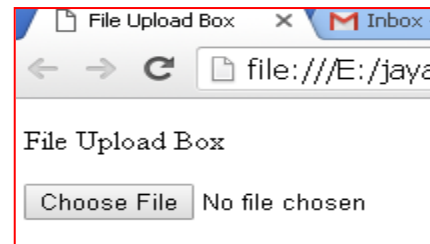


File Select boxes:- If you want to allow a user to upload a file to your web site, you will need to use a file upload box, also known as a file select box. This is also created using the `<input>` element but type attribute is set to **file**.

```

<!DOCTYPE html>
<html>
  <head>
    <title>File Upload Box</title>
  </head>
  <body>
    <p>File Upload Box</p>
    <form>
      <input type="file" name="fileupload" accept="image/*" />
    </form>
  </body>
</html>

```

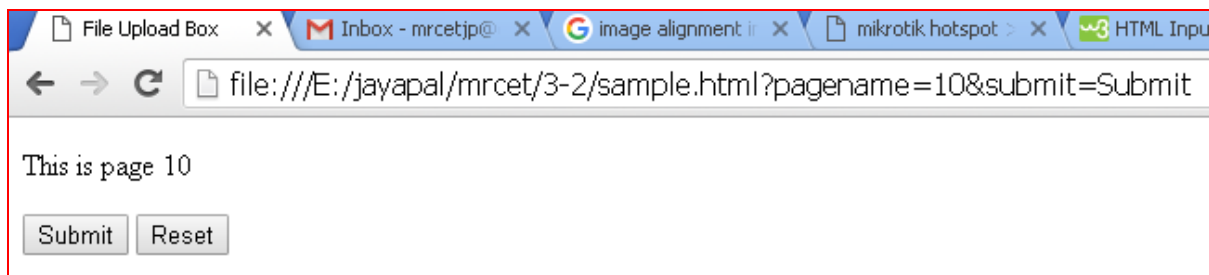


Hidden Controls:- Hidden form controls are used to hide data inside the page which later on can be pushed to the server. This control hides inside the code and does not appear on the actual page. For example, following hidden form is being used to keep current page number. When a user will click next page then the value of hidden control will be sent to the web server and there it will decide which page will be displayed next based on the passed current page.

```

<html> <head> <title>File Upload Box</title> </head>
  <body>
    <form>
      <p>This is page 10</p>
      <input type="hidden" name="pagename" value="10" />
      <input type="submit" name="submit" value="Submit" />
      <input type="reset" name="reset" value="Reset" />
    </form>
  </body> </html>

```



Button Controls:-

There are various ways in HTML to create clickable buttons. You can also create a clickable button using `<input>` tag by setting its type attribute to **button**. The type attribute can take the following values:

Type	Description
submit	This creates a button that automatically submits a form.
reset	This creates a button that automatically resets form controls to their initial values.
button	This creates a button that is used to trigger a client-side script when the user clicks that button.
image	This creates a clickable button but we can use an image as background of the button.

```

<!DOCTYPE html>
<html>
<head>
<title>File Upload Box</title>
</head>
<body>
<form>
<input type="submit" name="submit" value="Submit" />
<input type="reset" name="reset" value="Reset" />
<input type="button" name="ok" value="OK" />
<input type="image" name="imagebutton" src="test1.png" />
</form>
</body> </html>

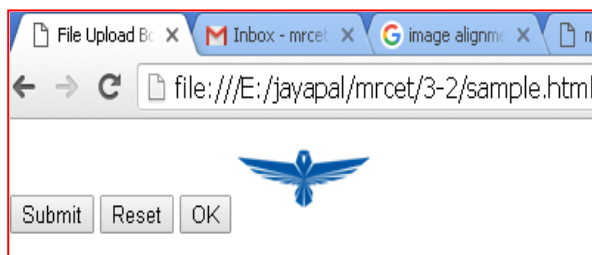
```

HTML frames: These are used to divide your browser window into multiple sections where each section can load a separate HTML document. A collection of frames in the browser window is known as a frameset. The window is divided into frames in a similar way the tables are organized: into rows and columns. To use frames on a page we use <frameset> tag instead of <body> tag. The <frameset> tag defines, how to divide the window into frames. The **rows** attribute of <frameset> tag defines horizontal frames and **cols** attribute defines vertical frames. Each frame is indicated by <frame> tag and it defines which HTML document shall open into the frame

```

<frameset cols="25%,50%,25%">
<frame src="frame_a.htm">
<frame src="frame_b.htm">
<frame src="frame_c.htm">
</frameset>

```



```

<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    <iframe src="sample1.html" height="400" width="400" frameborder="1">
    <h1>This is a Heading</h1>
    <p>This is a paragraph.</p>
    </iframe>
  </body>
</html>

```



XML - XML stands for **Extensible Mark-up Language**, developed by W3C in 1996. It is a text-based mark-up language derived from Standard Generalized Mark-up Language (SGML). XML 1.0 was officially adopted as a W3C recommendation in 1998. XML was designed to carry data, not to display data. XML is designed to be self-descriptive. XML is a subset of SGML that can define your own tags. A Meta Language and tags describe the content. XML Supports CSS, XSL, DOM. XML does not qualify to be a programming language as it does not performs any computation or algorithms. It is usually stored in a simple text file and is processed by special software that is capable of interpreting XML.

The Difference between XML and HTML

1. HTML is about displaying information, where asXML is about carrying information. In other words, XML was created to structure, store, and transport information. HTML was designed to display the data.
2. Using XML, we can create own tags where as in HTML it is not possible instead it offers several built in tags.
3. XML is platform independent neutral and language independent.
4. XML tags and attribute names are case-sensitive where as in HTML it is not.
5. XML attribute values must be single or double quoted where as in HTML it is not compulsory.
6. XML elements must be properly nested.
7. All XML elements must have a closing tag.

Well Formed XML Documents

A "Well Formed" XML document must have the following correct XML syntax:

- XML documents must have a root element
- XML elements must have a closing tag(start tag must have matching end tag).
- XML tags are case sensitive
- XML elements must be properly nested Ex:<one><two>Hello</two></one>
- XML attribute values must be quoted

XML with correct syntax is "Well Formed" XML. XML validated against a DTD is "Valid" XML.

What is Markup?

XML is a markup language that defines set of rules for encoding documents in a format that is both human-readable and machine-readable.

Example for XML Document

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?> <!--xml declaration-->
<note>
<to>MRCET</to>
<from>MRGI</from>
<heading>KALPANA</heading>

```

```
<body>Hello, world! </body>
</note>
```

- Xml document begins with XML declaration statement: `<? xml version="1.0" encoding="ISO-8859-1"?>` .
- The next line describes the **root element** of the document: **<note>**.
- This element is "the parent" of all other elements.
- The next 4 lines describe **4child elements** of the root: to, from, heading, and body. And finally the last line defines the end of the root element : **< /note>**.
- The XML declaration has no closing tag i.e. `</?xml>`
- The **default standalone value** is set to **no**. Setting it to **yes** tells the processor there are no external declarations (DTD) required for parsing the document. The file name extension used for xml program is .xml.

Valid XML document

If an XML document is well-formed and has an associated Document Type Declaration (DTD), then it is said to be a valid XML document. We will study more about DTD in the chapter XML - DTDs.

XML DTD

Document Type Definition purpose is to define the structure of an XML document. It defines the structure with a list of defined elements in the xml document. Using DTD we can specify the various elements types, attributes and their relationship with one another. Basically DTD is used to specify the set of rules for structuring data in any XML file.

Why use a DTD?

XML provides an application independent way of sharing data. With a DTD, independent groups of people can agree to use a common DTD for interchanging data. Your application can use a standard DTD to verify that data that you receive from the outside world is valid. You can also use a DTD to verify your own data.

DTD - XML building blocks

Various building blocks of XML are-

1. Elements: The basic entity is **element**. The elements are used for defining the tags. The elements typically consist of opening and closing tag. Mostly only one element is used to define a single tag.

Syntax1: `<!ELEMENT element-name (element-content)>`

Syntax 2: `<!ELEMENT element-name (#CDATA)>`

#CDATA means the element contains character data that is not supposed to be parsed by a parser. or

Syntax 3: `<!ELEMENT element-name (#PCDATA)>`

#PCDATA means that the element contains data that IS going to be parsed by a parser. or

Syntax 4: `<!ELEMENT element-name (ANY)>`

The keyword ANY declares an element with any content.

Example:

```
<!ELEMENT note (#PCDATA)>
```

Elements with children (sequences)

Elements with one or more children are defined with the name of the children elements inside the parentheses:

```
<!ELEMENT parent-name (child-element-name)> EX:<!ELEMENT student (id)>
                                     <!ELEMENT id (#PCDATA)> or
```

```
<!ELEMENT element-name (child-element-name, child-element-name,.....)>
```

Example: `<!ELEMENT note (to,from,heading,body)>`

When children are declared in a sequence separated by commas, the children must appear in the same sequence in the document. In a full declaration, the children must also be declared, and the children can also have children. The full declaration of the note document will be:

```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to    (#CDATA)>
<!ELEMENT from  (#CDATA)>
<!ELEMENT heading (#CDATA)>
<!ELEMENT body  (#CDATA)>
```

2. Tags

Tags are used to markup elements. A starting tag like `<element_name>` mark up the beginning of an element, and an ending tag like `</element_name>` mark up the end of an element.

Examples:

A body element: `<body>body text in between</body>`.

A message element: `<message>some message in between</message>`

3. Attribute: The attributes are generally used to specify the values of the element. These are specified within the double quotes. Ex: `<flag type="true">`

4. Entities

Entities as variables used to define common text. Entity references are references to entities. Most of you will know the HTML entity reference: " " that is used to insert an extra space in an HTML document. Entities are expanded when a document is parsed by an XML parser.

The following entities are predefined in XML:

< (<), > (>), & (&), " (") and ' (').

5. CDATA: It stands for character data. CDATA is text that will **NOT be parsed by a parser**. Tags inside the text will NOT be treated as markup and entities will not be expanded.

6. PCDATA: It stands for Parsed Character Data(i.e., text). Any parsed character data should not contain the markup characters. The markup characters are `<` or `>` or `&`. If we want to use these characters then make use of `<`, `>` or `&`. Think of character data as the text found between the start tag and the end tag of an XML element. PCDATA is text that will be **parsed by a parser**. Tags inside the text will be treated as markup and entities will be expanded.

```
<!DOCTYPE note
[
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to  (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)> ]>
```

Where PCDATA refers parsed character data. In the above xml document the elements to, from, heading, body carries some text, so that, these elements are declared to carry text in DTD file.

This definition file is stored with **.dtd** extension.

DTD identifier is an identifier for the document type definition, which may be the path to a file on the system or URL to a file on the internet. If the DTD is pointing to external path, it is called External Subset. The square brackets [] enclose an optional list of entity declarations called Internal Subset.

Types of DTD:

1. Internal DTD
2. External DTD

1. Internal DTD

A DTD is referred to as an internal DTD if elements are declared within the XML files. To refer it as internal DTD, standalone attribute in XML declaration must be set to yes. This means, the declaration works independent of external source.

Syntax:

The syntax of internal DTD is as shown:

```
<!DOCTYPE root-element [element-declarations]>
```

Where root-element is the name of root element and element-declarations is where you declare the elements.

Example:

Following is a simple example of internal DTD:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE address [
  <!ELEMENT address (name,company,phone)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT company (#PCDATA)>
  <!ELEMENT phone (#PCDATA)>
]>
<address>
  <name>Kalpana</name>
  <company>MRCET</company>
  <phone>(040) 123-4567</phone>
</address>
```

Let us go through the above code:

Start Declaration- Begin the XML declaration with following statement `<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>`

DTD- Immediately after the XML header, the document type declaration follows, commonly referred to as the DOCTYPE:

```
<!DOCTYPE address [
```

The DOCTYPE declaration has an exclamation mark (!) at the start of the element name. The DOCTYPE informs the parser that a DTD is associated with this XML document.

DTD Body- The DOCTYPE declaration is followed by body of the DTD, where you declare elements, attributes, entities, and notations:

```
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone_no (#PCDATA)>
```

Several elements are declared here that make up the vocabulary of the `<name>` document. `<!ELEMENT name (#PCDATA)>` defines the element name to be of type `"#PCDATA"`. Here `#PCDATA` means parseable text data. End Declaration - Finally, the declaration section of the DTD is closed using a closing bracket and a closing angle bracket (`]>`). This effectively ends the definition, and thereafter, the XML document follows immediately

XML Schemas

- ✓ XML Schema is commonly known as XML Schema Definition (XSD). It is used to describe and validate the structure and the content of XML data. XML schema defines the elements, attributes and data types. Schema element supports Namespaces. It is similar to a database schema that describes the data in a database. XSD extension is “.xsd”.
- ✓ This can be used as an alternative to XML DTD. The XML schema became the W#C recommendation in 2001.
- ✓ XML schema defines elements, attributes, element having child elements, order of child elements. It also defines fixed and default values of elements and attributes.
- ✓ XML schema also allows the developer to use **data types**.

Syntax : You need to declare a schema in your XML document as follows:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

Example : contact.xsd

The following example shows how to use schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="contact">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xs:string" />
      <xs:element name="company" type="xs:string" />
      <xs:element name="phone" type="xs:int" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

The basic idea behind XML Schemas is that they describe the legitimate format that an XML document can take.

XML Document: myschema.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<contact xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:noNamespaceSchemaLocation="contact.xsd">
<name>KALPANA</name>
<company>04024056789</company>
<phone>9876543210</phone>
</contact>
```

Limitations of DTD:

- There is no built-in data type in DTDs.
- No new data type can be created in DTDs.
- The use of cardinality (no. of occurrences) in DTDs is limited.
- Namespaces are not supported.
- DTDs provide very limited support for modularity and reuse.
- We cannot put any restrictions on text content.
- Defaults for elements cannot be specified.

DTDs are written in a non-XML format and are difficult to validate

Strengths of Schema:

- XML schemas provide much greater specificity than DTDs.
- They supports large number of built-in-data types.
- They are namespace-aware.
- They are extensible to future additions.
- They support the uniqueness.
- It is easier to define data facets (restrictions on data).

SCHEMA STRUCTURE

The Schema Element

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

Element definitions

As we saw in the chapter XML - Elements, elements are the building blocks of XML document. An element can be defined within an XSD as follows:

```
<xs:element name="x" type="y"/>
```

Data types:

These can be used to specify the type of data stored in an Element.

- String (xs:string)
- Date (xs:date or xs:time)
- Numeric (xs:integer or xs:decimal)
- Boolean (xs:boolean)

EX: Sample.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs=http://www.w3.org/XMLSchema>
  <xs:element name="sname" type="xs:string"/>
  /* <xs:element name="dob" type="xs:date"/>
     <xs:element name="dobtime" type="xs:time"/>
     <xs:element name="marks" type="xs:integer"/>
     <xs:element name="avg" type="xs:decimal"/>
     <xs:element name="flag" type="xs:boolean"/> */
</xs:schema>
```

Sample.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<sname xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="sample.xsd">
  Kalpana /*yyyy-mm-dd 23:14:34 600 92.5 true/false */
</sname>
```

Definition Types

You can define XML schema elements in following ways:

Simple Type - Simple type element is used only in the context of the text. Some of predefined simple types are: xs:integer, xs:boolean, xs:string, xs:date. For example:

```
<xs:element name="phone_number" type="xs:int" />
<phone>9876543210</phone>
```

UNIT - III : Introduction to Java Scripts

Objects in Java Script, Dynamic HTML with Java Script. Design of GUI.

Introduction to JavaScript

A number of technologies are present that develops the static web page, but we require a language that is dynamic in nature to develop web pages a client side. Dynamic HTML is a combination of content formatted using HTML, cascading stylesheets, a scripting language and DOM.

JavaScript originates from a language called LiveScript. The idea was to find a language which can be used at client side, but not complicated as Java. JavaScript is a simple language which is only suitable for simple tasks.

Benefits of JavaScript

Following are some of the benefits that JavaScript language possess to make the web site

- dynamic. It is widely supported in browser
- It gives easy access to document object and can manipulate most of
- them. JavaScript can give interesting animations with many multimedia
- datatypes. Special plug-in are not required to use JavaScript JavaScript
- is secure language
- JavaScript code resembles the code of C language, The syntax of both the language is very close to each other.

A Sample JavaScript program

```
<html>
<head><title>java script program</title>
<script language="javascript">
function popup()
{
var major=parseInt(navigator.appVersion);
var minor=parseInt(navigator.appVersion);
var agent=navigator.userAgent.toLowerCase();
document.write(agent+" "+major);
window.alert(agent+" "+major);
}
function farewell()
{
window.alert("Farewell and thanks for visiting");
}
</script>
</head>
<body onLoad="popup()" onUnload="farewell()">
</body>
</html>
```

- JavaScript program contains variables, objects and functions.
- Each line is terminated by a semicolon. Blocks of code must be surrounded by curly brackets.

- Functions have parameters which are passed inside parenthesis
- Variables are declared using the keyword var.

JavaScript program that shows the use of variables, datatypes

```
<html>
<head>
<title> My Sample JavaScript program</title>
<script language="javascript">
functiondisp()
{
varrno,sname,br,pr;
rno=prompt("Enter your registration number");
sname=prompt("Enter your Name");
br=prompt("Enter your branch Name");
pr=prompt("Enter the percentage");
document.writeln("<h2> Your Registration No. is :</h2>" + rno.toUpperCase());
document.writeln("<h2> Your Name is :</h2>" + sname.toUpperCase());
document.writeln("<h2> Your Branch Name is :</h2>" + br.toUpperCase());
document.writeln("<h2> Your Overall Percentage is :</h2>" + pr);
document.close();
}
</script>
</head>
<body onLoad="disp()">
</body>
</html>
```

JavaScript program showing the using of constructs

```
<html>
<head><title> Factorial</title></head>
<body>
<script language="javascript">
function fact(n)
{
vari,f=1;
for(i=1;i<=n;i++)
{
f=f*i;
}
return(f);
}
varx,n,f;
x=prompt("Enter the number");
f=fact(x);
document.writeln("Factorial of "+x+" is "+f);
document.close();
</script>
```

</body>

</html>

Variables

Variables are like storage units. You can create variables to hold values. It is ideal to name a variable something that is logical, so that you'll remember what you are using it for. For example, if you were writing a program to

numbers, it could be confusing if you called your variables numberOne, numberTwo, numberThree because you may forget which one is the divisor, which one is the dividend, and which one is the quotient. A more logical approach would be to name them just that: divisor, dividend, quotient.

It is important to know the proper syntax to which variables must conform:

- They must start with a letter or underscore (" _")
- Subsequent characters can also be digits (0-9) or letters (A-Z and/or a-z). Remember, JavaScript is case-sensitive. (That means that MyVariable and myVariable are two different names to JavaScript, because they have different capitalization.)

Some examples of legal names are Number_hits, temp99, and _name.

When you declare a variable by assignment outside of a function, it is called a global variable, because it is available everywhere in the current document. When you declare a variable within a function, it is called a local variable, because it is available only within the function. Using var is optional, but you need to use it if you have a variable that has been declared global and you want to re-declare it as a local variable inside a function.

Variables can store all kinds of data (see below, Values of Variables, section 3.2). To assign a value to a variable, you use the following notation:

```
dividend = 8;
```

```
divisor = 4;
```

```
myString = "I may want to use this message multiple  
times"; message = myString;
```

Let's say the main part of the function will be dividing the dividend by the divisor and storing that number in a variable called quotient. I can write this line of code in my program: quotient = divisor*dividend, and I have both stored the value of the quotient to the variable quotient and I have declared the variable at the same time. If I had wanted to, I could have declared it along with my other assigned variables above, with a value of null. After executing the program, the value of quotient will be 2.

That way, when you call it in your program, you do not have to retype the same sentence over and over again, and if you want to change the content of that message, you only have to change it once -- in the variable declaration.

Values of Variables

JavaScript recognizes the following types of values:

- Numbers, such as 42 or 3.14159
- Logical (Boolean) values, either true or false
- Strings, such as "Howdy!"
- null, a special keyword which refers to nothing

This relatively small set of types of values, or data types, enables you to perform useful functions with your applications. There is no explicit distinction between integer and real-valued numbers.

Data Type Conversion

JavaScript is a loosely typed language. That means you do not have to specify the data type of a variable when you declare it, and data types are converted automatically as needed during script execution. So, for example, you could define a variable as follows:

```
var answer = 42
```

 And later, you could assign the same variable a string value, for example,

```
answer = "Thanks for all the fish..."
```

In expressions involving numeric and string values, JavaScript converts the numeric values to strings. For example,

consider the following statements:

```
x = "The answer is " + 42
```

```
y = 42 + " is the answer."
```

(The + sign tells JavaScript to concatenate, or stick together, the two strings. For example, if you write:

```
message = "Hello" + "World"
```

Literals

You use literals to represent values in JavaScript. These are fixed values, not variables, that you literally provide in your script. Examples of literals include: 1234, "This is a literal," and true.

Integers

Integers can be expressed in decimal (base 10), hexadecimal (base 16), and octal (base 8). A decimal integer literal consists of a sequence of digits without a leading 0 (zero). A leading 0 (zero) on an integer literal indicates it is in octal; a leading 0x (or 0X) indicates hexadecimal. Hexadecimal integers can include digits (0-9) and the letters a-f and A-F. Octal integers can include only the digits 0-7.

Some examples of integer literals are: 42, 0xFFFF, and -345.

Floating-point literals

A floating-point literal can have the following parts: a decimal integer, a decimal point ((".")), a fraction (another decimal number), an exponent, and a type suffix. The exponent part is an "e" or "E" followed by an integer, which can be signed (preceded by "+" or "-"). A floating-point literal must have at least one digit, plus either a decimal point or "e" (or "E").

Some examples of floating-point literals are 3.1415, -3.1E12, .1e12, and 2E-12

Boolean literals

The Boolean type has two literal values: true and false.

String literals

A string literal is zero or more characters enclosed in double (") or single (') quotation marks. A string must be delimited by quotation marks of the same type; that is, either both single quotation marks or double quotation marks. The following are examples of string literals:

- "blah"
- 'blah'
- "1234"
- "one line \n another line"

escaping characters

For characters not listed in the preceding table, a preceding backslash is ignored, with the exception of a quotation mark and the backslash character itself.

You can insert quotation marks inside strings by preceding them with a backslash. This is known as escaping the quotation marks. For example,

```
var quote = "He read \"The Cremation of Sam McGee\" by R.W. Service."  
document.write(quote)
```

The result of this would be

He read "The Cremation of Sam McGee" by R.W. Service.

To include a literal backslash inside a string, you must escape the backslash character. For example, to assign the

file path c:\temp to a string, use the following:

```
var home = "c:\\temp"
```

Arrays

An Array is an object which stores multiple values and has various properties. When you declare an array, you must declare the name of it, and then how many values it will need to store. It is important to realize that each value is stored in one of the elements of the array, and these elements start at 0. This means that the first value in the array is really in the 0 element, and the second number is really in the first element. So for example, if I want to store 10 values in my array, the storage elements would range from 0-9. The notation for declaring an array looks like this:

```
myArray = new Array(10); foo = new Array(5);
```

Initially, all values are set to null. The notation for assigning values to each unit within the array looks like this:

```
myArray[0] = 56;
```

```
myArray[1] = 23;
```

```
myArray[9] = 44;
```

Operators

JavaScript has many different operators, which come in several flavors, including binary. This tutorial will cover some of the most essential assignment, comparison, arithmetic and logical operators.

Selected assignment operators

An assignment operator assigns a value to its left operand based on the value of its right operand. The basic assignment operator is equal (=), which assigns the value of its right operand to its left operand. The other operators are shorthand for standard operations. Find an abridged list of shorthand operators below:

Shorthand operator	Meaning
x += y	x = x + y
x -= y	x = x - y
x *= y	x = x * y
x /= y	x = x / y

.

Comparison operators

A comparison operator compares its operands and returns a logical value based on whether the comparison is true or not. The operands can be numerical or string values. When used on string values, the comparisons are based on the standard lexicographical ordering.

They are described in the following table.

Operator	Description	Example
Equal (==)	Evaluates to true if the operands are equal.	<code>x == y</code> evaluates to true if x equals y.
Not equal (!=)	Evaluates to true if the operands are not equal.	<code>x != y</code> evaluates to true if x is not equal to y.
Greater than (>)	Evaluates to true if left operand is greater than right operand.	<code>x > y</code> evaluates to true if x is greater than y.
Greater than or equal (>=)	Evaluates to true if left operand is greater than or equal to right operand.	<code>x >= y</code> evaluates to true if x is greater than or equal to y.
Less than (<)	Evaluates to true if left operand is less than right operand.	<code>x < y</code> evaluates to true if x is less than y.
Less than or equal (<=)	Evaluates to true if left operand is less than or equal to right operand.	<code>x <= y</code> evaluates to true if x is less than or equal to y.

EXAMPLES: `5 == 5` would return TRUE. `5 != 5` would return FALSE. (The statement 'Five is not equal to five.' is patently false.) `5 <= 5` would return TRUE. (Five is less than or equal to five. More precisely, it's exactly equal to five, but JavaScript could care less about boring details like that.)

Selected Arithmetic Operators

Arithmetic operators take numerical values (either literals or variables) as their operands and return a single numerical value. The standard arithmetic operators are addition (+), subtraction (-), multiplication (*), division (/) and remainder (%). These operators work as they do in other programming languages, as well as in standard algebra.

Since programmers frequently need to add or subtract 1 from a variable, JavaScript has shortcuts for doing this.

`myVar++` adds one to the value of `myVar`, while `myVar--` subtracts one from `myVar`.

EXAMPLES: Let `x = 3`. `x++` bumps `x` up to 4, while `x--` makes `x` equal to 2.

Logical Operators

Logical operators take Boolean (logical) values as operands and return a Boolean value. That is, they evaluate whether each subexpression within a Boolean expression is true or false, and then execute the operation on the respective truth values. Consider the following table:

Operator	Usage	Description
and (&&)	expr1 && expr2	True if both logical expressions expr1 and expr2 are true. False otherwise.
or ()	expr1 expr2	True if either logical expression expr1 or expr2 is true. False if both expr1 and expr2 are false.
not (!)	!expr	False if expr is true; true if expr is false.

EXAMPLES: Since we have now learned to use the essential operators, we can use them in conjunction with one another. See if you can work out why the following examples resolve the way they do: If $x = 4$ and $y = 7$, $((x + y + 2) == 13) \&\& (((x + y) / 2) == 2)$ returns FALSE.

If $x = 4$ and $y = 7$, $((y - x + 9) == 12) || ((x * y) == 2)$ returns TRUE.

If $x = 4$ and $y = 7$, $!((x/2 + y) == 9) || ((x * (y/2)) == 2)$ returns FALSE.

Using JavaScript Objects

When you load a document in your web browser, it creates a number of JavaScript objects with properties and capabilities based on the HTML in the document and other pertinent information. These objects exist in a hierarchy that reflects the structure of the HTML page itself.

The pre-defined objects that are most commonly used are the window and document objects. The window has methods that allow you to create new windows with the open() and close() methods. It also allows you to create message boxes using alert(), confirm(), and prompt(). Each displays the text that you put between the parentheses.

For example, the following code:

```
alert("This is an alert box")
```

The objects in this pre-defined hierarchy can be accessed and modified. To refer to specific properties, you must specify the property name and all its ancestors, spelling out the complete hierarchy until the document object. A period, '.', is used in between each object and the name of its property. Generally, a property / object gets its name from the NAME attribute of the HTML tag. For example, the following refers to the *value* property of a text field named *text1* in a form named *myform* in the current document. `document.myform.text1.value`

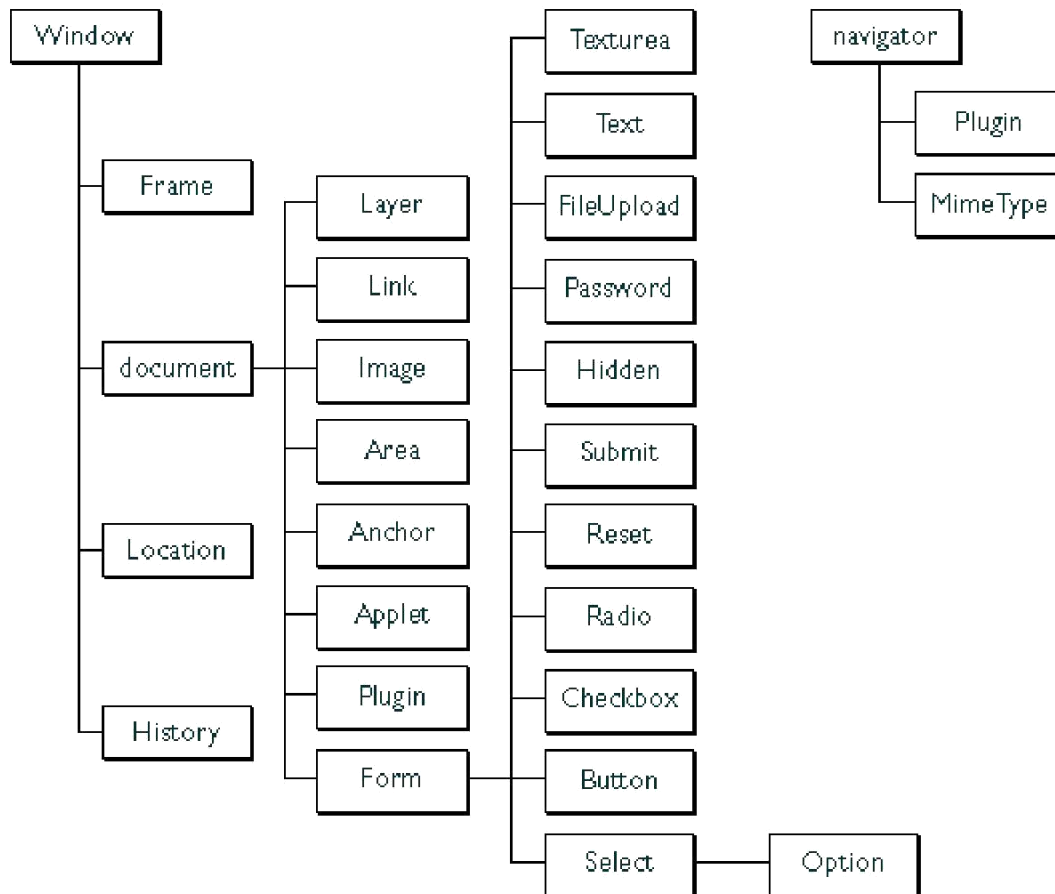
Form elements can also be accessed through the aforementioned forms array of the document object. In the above

example, if the form named *myform* was the first form on the page, and *text1* was the third field in the form, the following also refers to that field's value property.

`document.forms[0].elements[2].value` Functions (capabilities) of an object can similarly be accessed using the period notation. For example, the following instruction resets the 2nd form in the document.

```
document.forms[2].reset();
```

Click on one of the objects below to view the Netscape documentation on the specific properties and methods that that object has:



Functions

Functions are one of the fundamental building blocks in JavaScript. A function is a JavaScript procedure -- a set of statements that performs a specific task. A function definition has these basic parts:

- The function keyword
- A function name
- A comma-separated list of arguments to the function in parentheses
- The statements in the function in curly braces: { }

Defining a Function

When defining a function, it is *very* important that you pay close attention to the syntax. Unlike HTML, JavaScript is case sensitive, and it is very important to remember to enclose a function within curly braces { }, separate parameters with commas, and use a semi-colon at the end of your line of code.

It's important to understand the difference between *defining* and *calling* a function.

Defining the function names the function and specifies what to do when the function is called. You define a function within the <SCRIPT>...</SCRIPT> tags within the <HEAD>...</HEAD> tags. In defining a function, you must also declare the variables which you will be calling in that function. Here's an example of *defining* a function:

```

Function popupalert() {

    alert("This is an alert box.");

}
  
```

Notice the parentheses after the function name. It is imperative that you include these parentheses, even if they are empty. If you want to pass a *parameter* into the function, you would include that parameter inside of the parentheses. A parameter is a bit of extra information that can be different each time the function is run. It is stored in a variable and can be accessed just like any other variable. Here's an example of a function that takes a parameter:

```
Function anotherAlert(word) {  
  
    alert(word + ' is the word that you clicked on');  
  
}
```

When you call this function, you need to pass a parameter (such as the word that the user clicked on) into the function. Then the function can use this information. You can pass in a different word as a parameter each time you call the function, and the alert box will change appropriately. You'll see how to pass a parameter a little later on. You can pass in multiple parameters, by separating them with a comma. You would want to pass in a few parameters if you have more than one variable that you either want to change or use in your function. Here are two examples of passing in multiple parameters when you are defining the function:

```
functionsecondAlert(word,password) {  
  
    confirm(word + ' is the word that you clicked on. The  
  
        secret password is ' + password);  
  
}  
functionthirdAlert(word,password) {  
  
    confirm(word + ' is the word you clicked on. Please  
  
        take note of the password, ' + password);  
  
}
```

You'll notice that the same parameters are passed into both of these functions. However, you can pass in whatever values you want to use (see this same example below in calling the function).

Calling a Function

Calling the function actually performs the specified actions. When you call a function, this is usually within the

BODY of the HTML page, and you usually pass a parameter into the function. A parameter is a variable from outside

of the defined function on which the function will act.

Here's an example of calling the same function:

```
popupalert();
```

For the other example, this is how you may call it:

```
<A HREF="#top" onClick="anotherAlert('top')">top</A>
```

Here is the same example with multiple parameters that was shown above:

```
<A HREF="#top" onClick="secondAlert('awesome','pandas')">awesome</A>
```

```
<A HREF="#top" onClick="thirdAlert('computers','insert')">computers</A>
```

You'll notice in the code that different values for the variables `word` and `password` are passed in. These values here are what the function will need to perform the actions in the function. Make sure that the values you pass in are in the correct order because the function will take them in and assign these values to the parameters in the parentheses of the function declaration. Once you pass values into your function, you can use them however you want within your function.

Try it for yourself:

When you click on the words below, a confirmation box will pop up and then the link will bring you to the top of the page.

awesome

computers

If/Else Statements

if statements execute a set of commands if a specified condition is true. If the condition is false, another set of statements can be executed through the use of the `else` keyword.

The main idea behind if statements is embodied by the sentence: "If the weather's good tomorrow, we'll go out and have a picnic and Lisa will do cartwheels -- else, we'll stay in and Catherine will watch TV." As you can see, the idea is quite intuitive and, surprisingly enough, so is the syntax:

```
if(condition) {  
    statements1  
}  
-or-  
if(condition) {  
    statements1  
}  
else{  
    statements2  
}
```

(An **if** statement does not require an **else** statement following it, but an **else** statement must be preceded by an **if** statement.)

condition can be any JavaScript expression that evaluates to true or false. Parentheses are required around the condition. If *condition* evaluates to true, the statements in *statements1* are executed.

statements1 and *statements2* can be any JavaScript statements, including further nested **if** statements.

Multiple

statements must be enclosed in braces.

Here's an example:

```
if(weather == 'good') {  
    go_out(we);  
    have_a_picnic(we);  
    do_cartwheels(Lisa);  
}
```

```
    watch_TV(Catherine);  
}
```

Loops

Loops are an incredibly useful programming tool. Loops handle repetitive tasks extremely well, especially in the context of consecutive elements. Arrays immediately spring to mind here, since array elements are numbered consecutively. It would be quite intuitive (and equally practical), for instance, to write a loop that added 1 to each element within an array. Don't worry if this doesn't make a lot of sense now, it will, after you finish reading the tutorial.

The two most common types of loops are for and while loops:

for Loops

A for loop constitutes a statement which consists of three expressions, enclosed in parentheses and separated by semicolons, followed by a block of statements executed in the loop.

This definition may, at first, sound confusing. Indeed, it is hard to understand for loops without seeing them in action.

A for loop resembles the following:

```
for (initial-expression; condition; increment-expression) {  
    statements  
}
```

The *initial-expression* is a statement or variable declaration. (See the section on variables for more information.) It is typically used to initialize a counter variable. This expression may optionally declare new variables with the var keyword.

The *condition* is evaluated on each pass through the loop. If this condition evaluates to true, the statements in *statements* are performed. When the condition evaluates to false, the execution of the for loop stops. This conditional test is optional. If omitted, the condition always evaluates to true.

The *increment-expression* is generally used to update or increment the counter variable.

The *statements* constitute a block of statements that are executed as long as *condition* evaluates to true. This can be a single statement or multiple statements. Although not required, it is good practice to indent these statements from the beginning of the for statement to make your code more readable.

Check out the following for statement. It starts by declaring the variable *i* and initializing it to zero. It checks

whether *i* is less than nine, performs the two successive statements, and increments *i* by one after each pass through the loop:

```
var n = 0;  
for (var i = 0; i < 3; i++) {  
    n += i;  
    alert("The value of n is now " + n);  
}
```

while Loops

The while loop, although most people would not recognize it as such, is for's twin. The two can fill in for one another - using either one is only a matter of convenience or preference according to context. while creates a loop that evaluates an expression, and if it is true, executes a block of statements. The loop then repeats, as long as the specified condition is true.

The syntax of while differs slightly from that of for:

```
while(condition) {  
    statements  
}
```

condition is evaluated before each pass through the loop. If this condition evaluates to true, the statements in the succeeding block are performed. When *condition* evaluates to false, execution continues with the statement following *statements*.

statements is a block of statements that are executed as long as the *condition* evaluates to true. Although not required, it is good practice to indent these statements from the beginning of the statement. The following while loop iterates as long as *n* is less than three.

```
var n = 0;
var x = 0;
while(n < 3) {
    n++;
    x += n;
    alert("The value of n is " + n + ". The value of x is " + x);
}
```

Try it for yourself: Click this link

Commenting

Comments allow you to write notes to yourself within your program. These are important because they allow someone to browse your code and understand what the various functions do or what your variables represent. Comments also allow you to understand your code if it's been a while since you last looked at it.

In JavaScript, you can write both one-line comments and multiple line comments. The notation for each is different though. For a one line comment, you precede your comment with `//`. This indicates that everything written on that line, after the `//`, is a comment and the program should disregard it.

For a multiple-line comment, you start with `/*` and end with `*/`. It is nice to put an `*` at the beginning of each line just so someone perusing your code realizes that he/she is looking at a comment (if it is really long this helps). This is not necessary though.

The following are examples of comments in JavaScript.

`// This is a single line comment.`

`/* This is a multiple line comment with only one line. */`

`/* This is a multiple line comment.`

`* The star (*) at the beginning of this line is optional.`

`* So is the star at the beginning of this line. */`

JavaScript program using objects

```
<html>
<head>
<script language="javascript">
function demo1()
{
  Popup("Hello");
  Obj= new sample (2, 4);
  alert(obj.x + obj.y);
}
function sample(x,y)
{
  this.x=x;
  this.y=y;
}
</script>
</head>
```

```
<body onLoad="demo1()">
</body>
</html>
```

Regular Expression

A script language may take name data from a user and have to search through the string one character at a time. The usual approach in scripting language is to create a pattern called a regular expression which describes a set of characters that may be present in a string.

```
var pattern = "target";
var string = "can you find the target";
string.match(pattern);
```

But the above code can also be written using regular expression as a parameter, as shown below.

```
var pattern = new RegExp("target");
var string = "can you find the target";
pattern.exec(string);
```

Regular expression is a javascript object. Dynamic patterns are created using the keyword new.

```
regex = new RegExp("feroz | btech");
```

JavaScript code to implement RegExp

```
<html>
<head>
<body>
<script language="javascript">
var re = new RegExp("*A | a+mer");
var msg=" Have you met Btech recently";
var res= re.exec(msg);
if(res)
{
alert( " I found " + res*0+);
}
else
{
alert(" I didn't find it");
}
</script>
</body>
</html>
```

Functions:

Regular Expressions are manipulated using the functions which belong to either the RegExp or String class.

Class String functions

match(pattern)

This function searches a matching pattern. Returns array holding the results.

replace(pattern1, pattern2)

Searches for pattern1. If the search is successful pattern1 is replaced with pattern2.

search(pattern)

Searches for a pattern in the string. If the match is successful, the index of the start of the match is returned. If the search fails, the function returns -1.

Class RegExp functions

exec(string)

Executes a search for a matching pattern in its parameter string. Returns an array holding the results of the operation.

test(string)

Searches for a match in its parameter string. Returns true if a match is found, otherwise returns false.

Built in objects:

The document object

A document is a web page that is being either displayed or created. The document has a number of properties that can be accessed by JavaScript programs and used to manipulate the content of the page.

Write or writeln

Html pages can be created using JavaScript. This is done by using the write or writeln methods of the document object.

```
Document.write("<body>");
```

```
Document.write("<h1> Hello </h1>");
```

The form object

Two aspects of the form can be manipulated through JavaScript. First, most commonly and probably most usefully, the data that is entered onto your form can be checked at submission. Second you can actually build forms through JavaScript.

Example : Validate.js

```
function validate()
{
var t1=document.forms[0].elements;
var t2=parent.frames['f4'].document;
var bg1=t1.bg.value;
var c1=t1.c.value;
t2.open();
t2.write("<body bgcolor="+bg1+">");
t2.write("Candidate name is : "+c1);
t2.write("</body>");
t2.close();
}
```

Mypage.html

```
<html>
```

```
<head>
```

```
<script language = "javascriptsrc= "D:\Documents and Settings \ p6 \ validate.js">
```

```
</script>
```

```
</head>
```

```
<body>
```

```
<form>
```

The browser object

Some of the properties of the browser object is as follows

Navigator.appCodeName : The internal name for the browser.

Navigator.appVersion:This is the public name of the browser.

Navigator.appVersion:The version number, platform on which the browser is running.

Navigator.userAgent :The strings appCodeName and appVersion concatenated together.

The Date object

JavaScript provides functions to perform many different date manipulation. Some of the functions are mentioned below.

Date() : Construct an empty date object.

Date(year, month, day [,hour, minute, second]) :Create a new Date object based upon numerical values for the year, month and day. Optional time values may also be supplied. getDate():Return the day of the month

getDay():Return an integer representing the day of the week.

getFullYear():Return the year as a four digit number.

getHours():Return the hour field of the Date object.

getMinutes():Return the minutes field of the Date object.

getSeconds():Return the second field of the Date object.

setDate(day):Set the day value of the object. Accepts values in the range 1 to 31.

setFullYear(year [,month, day]):Set the year value of the object. Optionally also sets month and day values.

toString():Returns the Date as a string.

Events:

JavaScript is a event-driven system. Nothing happens unless it is initiated by an event outside the script. The table below shows event, event handler and the description about the event handler.

The following are events used with the elements

Event Attribute	Description	Tags/elements used
onblur	When the field lost focuses and Enters into another field usually By tag or mouse click	Area,button,input,select, textarea
onchange	Whenever the text or options are Modified	Input,select,textarea
onclick	On clicking the button or reset or submit etc	Most elements
ondblclick	Clicking twice	Most elements
onfocus	Got focus in the field or entering into the field	Area,button,input,select, textarea
Onkeydown	Key pressed down	Most elements

onload	When the document is loaded	Body,frameset
Onmousedown	Moving mousedown	Most elements
Onmousemove	Moving mouse	Most element
Onmouseout	Mouse moved away	Mostelement
Onmouseover	Placingmouse on it	Most
Onreset	Clicking on reset button	Form
Onselect	Selecting text	Input.textarea
Onsubmit	Clicking on submit button	Form
Onunload	When unloaded from memory	Body ,frameset

Dynamic HTML with JavaScript

Data Validation

Data validation is the common process that takes place in the web sites. One common request is for a way of validating the username and password. Following program shows the validation of data which uses two frames, in one frame user is going to enter the data and in the other frame equivalent result is going to be displayed.

Example JavaScript code for data validation

Mypage.html

```

<html>
<head>
<title>frame page </title>
</head>
<frameset rows="20%,*">
<frame name="f1" src="">
<frameset cols="20%,*">
<frame name="f2" src="">
<frameset cols="50%,*">
<frame name="f3" src="D:\Documents and Settings\Btech\Desktop\btech\p6\reg.html">
<frame name="f4" src="D:\Documents and
Settings\Btech\Desktop\btech\p6\profile.html">
</frameset>
</frameset>
</frameset>
</html>

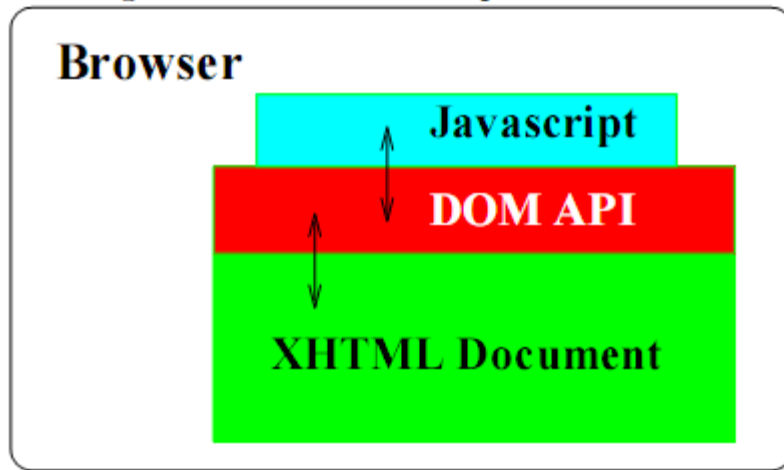
```

Document Object Model and Dynamic HTML

The term Dynamic HTML, often abbreviated as DHTML, refers to the technique of making Web pages dynamic by client-side scripting to manipulate the document content and presentation. Web pages can be made more lively, dynamic, or interactive by DHTML techniques. With DHTML you can prescribe actions triggered by browser events to make the page more lively and responsive. Such actions may alter the content and appearance of any parts of the page. The changes are fast and efficient because they are made by the browser without having to network with any servers.

Contrary to what the name may suggest, DHTML is not a markup language or a software tool. It is a technique to make dynamic Web pages via client-side programming. In the past, DHTML relies on browser/vendor specific features to work. Making such pages work for all browsers requires much effort, testing, and unnecessarily long programs. Standardization efforts at W3C and elsewhere are making it possible to write standard based DHTML that work for all compliant browsers. Standard-based DHTML involves three aspects:

Figure 10.1: DOM Compliant Browser

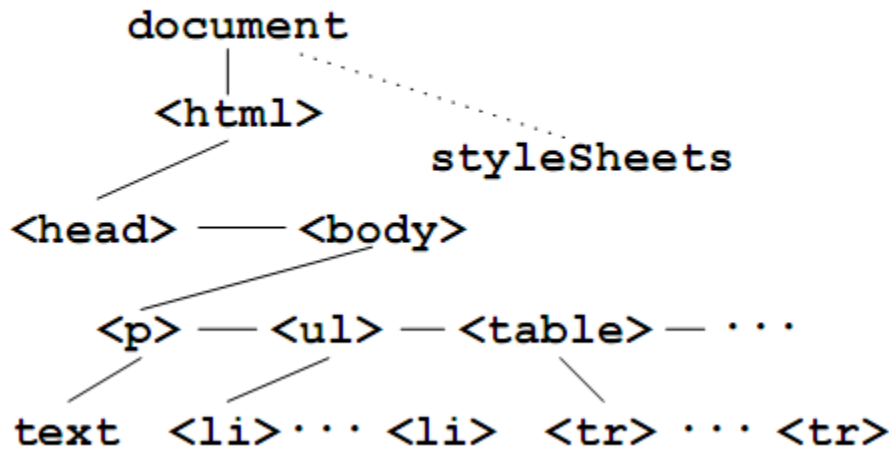


1. Javascript—for cross-browser scripting
2. Cascading Style Sheets (CSS)—for style and presentation control
3. Document Object Model (DOM)—for a uniform programming interface to access and manipulate the Web page as a document

10.1 What Is DOM?

With cooperation from major browser vendors, the W3C is establishing the Document Object Model (DOM) as a standard application programming interface (API) for scripts to access and manipulate HTML and XML documents. Compliant clients, including browsers and other user agents, provide the DOM specified API to access and modify the document being processed (Figure 10.1). The DOM API gives a logical view of the document where objects

Figure 10.2: DOM Tree Structure



represent different parts: windows, documents, elements, attributes, texts, events, style sheets, style rules, etc. These DOM objects are organized into a tree structure (the DOM tree) to reflect the natural organization of a document. HTML elements are represented by tree nodes and organized into a hierarchy. Each Web page has a document node at the root of the tree. The head and body nodes become child nodes of the document node (Figure 10.2).

DOM Tree Nodes

The DOM tree for a Web page consists of different types of nodes (of type Node) including:

HTML Document —Root of the DOM tree providing access to page-wide quantities, style sheets, markup elements, and, in most cases, the element as a child node.

HTML Element —Internal and certain leaf nodes on the DOM tree representing an HTML markup element. The HTML Element interface provides access to element attributes and child nodes that may represent text and other HTML elements. Because we focus on the use of DOM in DHTML, we will use the terms element and HTML element interchangeably. The `document.getElementById(id)` call gives you any element with the given id.

Attr —An attribute in an HTML Element object providing the ability to access and set an attribute. The name field (a string) of an Attr object is read-only while the value field can be set to a desired string. The attributes field of an HTML Element object gives you a NamedNodeMap of Attr objects. Use the length property and the item(index) method of the named node map to visit each attribute. All DOM indices are zerobased.

Text —A leaf node containing the text inside a markup element. If there is no markup inside an element's content, the text is contained in a single Text object that is the only child of the element. The wholeText (or data) field returns the entire text as a string. Set the data string or call the replaceWholeText(str) method to make str the new text.

Getting Started with DOM

Let's create a simple calculator (Ex: DomCalc) to demonstrate DOM and DHTML. The user enters an arithmetic expression and clicks a button to perform the required computations. The answer is displayed in the regular running text of the page (Figure 10.6). The HTML source shows the code for the input control (line A), the GO button (line B) and the for displaying the computed result (line C).

The calculator is initialized immediately after page loading. The init and the comp (line B) event handlers are in the Javascript file domcalc.js:

```
var answer;
function init()
{ answer = document.getElementById("ans")
  .firstChild; // (D)
  comp("uin");
```

Figure 10.8: A DOM Calculator

}

```
function comp(id) {
var el = document.getElementById(id); // (E)
var res = eval(el.value); // (F)
answer.data = res; // (G)
}
```

Graphical User Interface - Layout and Design

Learning Objectives

You will be able...

- ❑ ...to explain what a Graphical User Interface (GUI) is and what it is good for.
- ❑ ...to design a Graphical User Interface for an interactive map.

Introduction

Don't you think that it is easier to click on a button to initialize a computer action than typing specific commands in a command line? Surely, if you are a computer expert you would say that the command line is more comfortable but if you are a beginner you would agree that clicking on buttons is the easier way to handle the computer's actions. That is why Graphical User Interfaces were invented. They ease the handle of computer programs.



GUI (screenshot © Ubuntu)

When designing a Graphical User Interface, it is important that the needs, wants, and limitations of the end users (who finally use the program) are given extensive attention. There exist a few rules for the design of Graphical User Interfaces which will be listed in this lesson.

Since the topic of this module is multimedia cartography, we will concentrate on Graphical User Interfaces of interactive maps.

Graphical User Interface - Layout and Design

1.1. User Interface versus Graphical User Interface

Learning Objectives

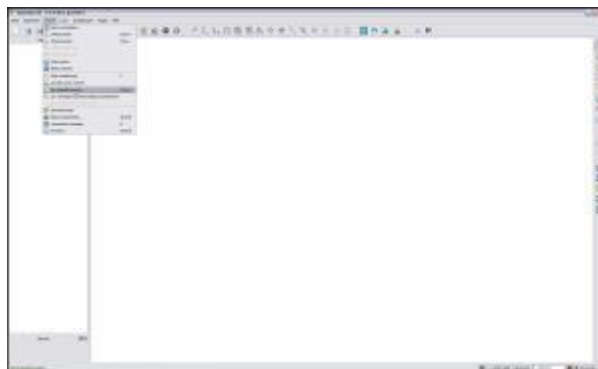
You will be able...

1. ...to explain what a User Interface is.
2. ...to describe what a Graphical User Interface (GUI) is and list at least three features of a today's GUI.

Introduction

If you click on an arbitrary button (pdf, glossary or help button, etc.) or interactive item (interactive index, link, etc.) while reading this lesson, you interact with the computer system.

When you sit in front of the screen of a running computer, you always face a User Interface (UI). As the name implies, a User Interface is an interface between the user and the computer. The first user interfaces were command-line interfaces where you only could interact with the computer by typing commands on the keyboard (some Unix users still use this kind of interface). Today, we (Mac, Linux and Windows-Users) expect to interact with the computer using a mouse, launching programs by clicking on icons, and manipulate various windows on the screen using graphical controls. Such user interfaces are called Graphical User Interfaces (GUI), since they use graphics and pictures to represent the input and output of a program.



GUI (screenshot © Quantum GIS)

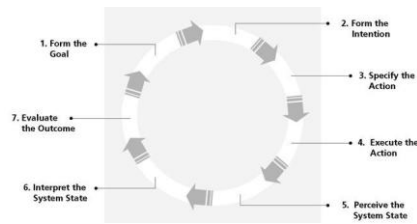
1.1.1. Interacting with a System

Before talking about User Interfaces and Graphical User Interfaces, we want to begin with the basics. How machines and humans interact.

First we want to describe the stages of actions users go through when faced with the task of using a system. According to (1988) there are seven stages of a typical user interaction with a generic interaction system:

2. Forming the goal: I want to do something in a program.
3. Forming the intention: I have to start the program.
4. Specifying the action: I have to click on a button to open the program.
5. Executing the action: I click on the button.
6. Perceive the system state: Note that the computer operates.
7. Interpret the system state: The computer opens (hopefully) the right program.
8. Semantically evaluating the interaction outcome: Note that the right program is open.

Graphical User Interface - Layout and Design



Product Designing according to (Marinilli 2002)

First, the user forms a conceptual intention from her/his goal. Second, s/he tries to adapt this intention to the commands provided by the system and from these commands carries out the action. Then, the user attempts to understand the outcomes of her/his actions. This is particularly important for computer systems, where the inner workings are hidden and users have to figure out the internal state only from few hints.

1.1.2. Types of User Interfaces

To work with a system, the users need to be able to control the system and assess the state of the system.

Definition of User Interface

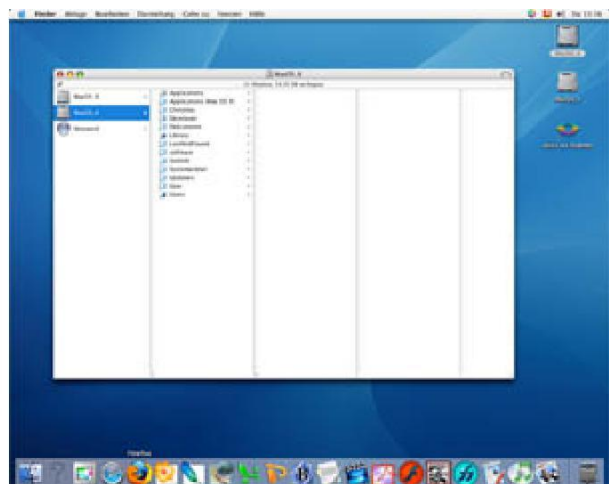
In computer science and human-computer interaction, the user interface (of a computer program) refers to the graphical, textual and auditory information the program presents to the user. The user employs several control sequences (such as keystrokes with the computer keyboard, movements of the computer mouse, or selections with the touchscreen) to control the program. (Wikipedia)

There exist several types of user interfaces. We here give you just two examples:

1. **Command-Line Interface (CLI):** The user provides the input by typing a command string with the computer keyboard and the system provides output by printing text on the computer monitor (Wikipedia).
2. **Graphical User Interface (GUI):** The use of pictures rather than just words to represent the input and output of a program (Linux junkies). Input is accepted via devices such as keyboard and mouse.

```
0:\Temp\Test>python2svg.exe seenganz seenganz.svg 0.1
python2svg.exe: Version 0.5, 2005-09-13
Usage: python2svg.exe --input sourceShapeFile --output sourceOutput.svg --roundval
1 0.1 [--scale 250000] [--inputunits u1] [--outputunits u2] [--referenceframe] [--add]
you have to specify an input file (shp)?
0:\Temp\Test>python2svg.exe --input seenganz --output seenganz.svg --roundval 0.1
working on layer seenganz ...
converting shapefile to a temporary sqlfile ... done.
tablename: seenganz
The following attributes are available. Please select the attributes you want to
include in the SVG export:
Attribute=gid. Type=serial Do you want to include it (y/n)?_
```

Command-Line Interface



Graphical User Interface (screenshot © Apple)

Graphical User Interfaces will be discussed in detail in the following chapters.

Graphical User Interface - Layout and Design

1.1.3. Graphical User Interface

Definition of Graphical User Interface (GUI)

Graphical User Interfaces use pictures and graphics instead of just words to represent the input and output of a program. The program displays certain icons, buttons, dialogue boxes etc. on the screen and the user controls the program mainly by moving a pointer on the screen (typically controlled by a mouse) and selecting certain objects by pressing buttons, etc. (Linux junkies)

Short History of GUI

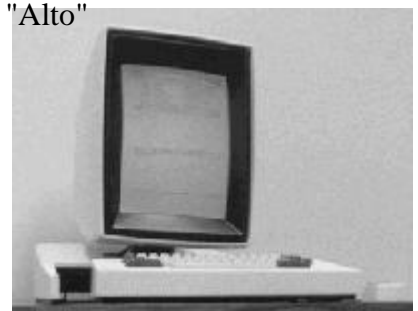
"Today, almost everybody in the developed world interacts with personal computers in some form or another. We use them at home and at work, for entertainment, information, and as tools to leverage our knowledge and intelligence. It is pretty much assumed whenever anyone sits down to use a personal computer that it will operate with a graphical user interface. We expect to interact with it primarily using a mouse, launch programs by clicking on icons, and manipulate various windows on the screen using graphical controls. But this was not always the case." (Reimer 2005)

until 1970 Command Line Interfaces: Text-based user interfaces requiring commands to be typed on the keyboard.



from 1973 GUI-Prototypes: Development of the first operational "Alto" Computer at the Xerox Palo Alto Research Center (PARC). The Alto is the first system to pull together all of the elements of the modern Graphical User Interface. (toastytech.com) Features:

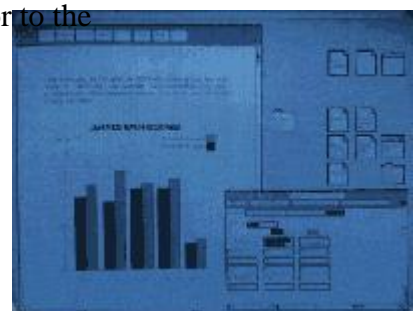
- 3-button mouse
- Bit-mapped display
- Use of graphical windows



(toastytech.com)

1981 Xerox introduces the "Star", the commercial successor to the Alto. (toastytech.com)
Notable Features:

- Double-clickable icons
- Overlapping windows
- Dialog Boxes
- 1024*768 monochrome display



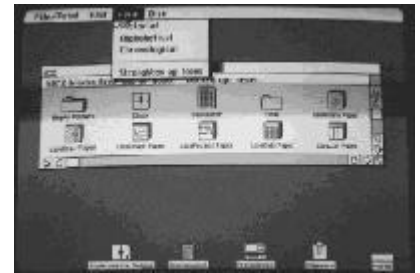
(toastytech.com)

Graphical User Interface - Layout and Design

1983

Apple introduces the new computer "Lisa".
(toastytech.com)
Notable Features:

- Pull down menus
- Menu Bars



(toastytech.com)

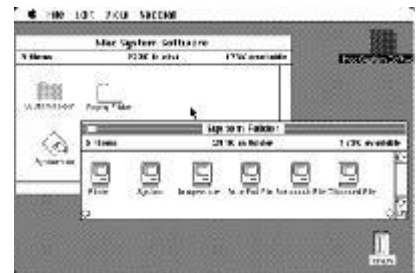
Visi Corp releases Visi On, the first integrated graphical software environment for IBM PCs.
(toastytech.com)



(toastytech.com)

1984

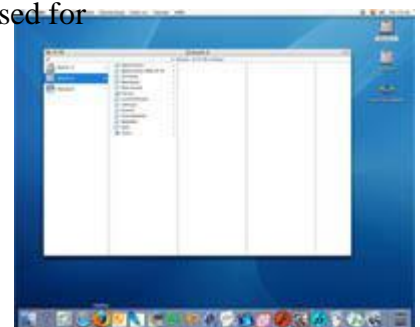
Apple introduces the Macintosh. (toastytech.com)



(toastytech.com)

Today

Graphical User Interfaces are very common and are used for almost all programs.



(screenshot © Apple)

Graphical User Interface - Layout and Design

1. Semantically evaluating the interaction outcome

User Interface

In computer science and human-computer interaction, the user interface (of a computer program) refers to the graphical, textual and auditory information the program presents to the user, and the control sequences (such as keystrokes with the computer keyboard, movements of the computer mouse, and selections with the touch screen) the user employs to control the program. (Wikipedia)

Graphical User Interface

"Graphical User Interface is the use of pictures rather than just words to represent the input and output of a program. A program with a GUI runs under some windowing system (e.g. The X Window System, Microsoft Windows, Mac OS). The program displays certain icons, buttons, dialogue boxes etc. in its windows on the screen and the user controls it mainly by moving a pointer on the screen (typically controlled by a mouse) and selecting certain objects by pressing buttons on the mouse while the pointer is pointing at them." (Linuxjunkies)

Graphical User Interface - Layout and Design

1.2. Graphical User Interface Design

Learning Objectives

You will be able...

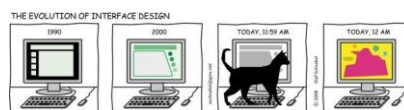
- // ...to name the main principle of User-Centered-Design.
- // ...to list at least four of the "Golden GUI Design Rules".
- // ...to list the four GUI control groups.
- // ...to list the four ways to catch the user's attention.

Introduction

Designing a user interface may seem a simple and side aspect of the development of an entire application. In fact it is, perhaps, the most important part of the development of an application.

Marinilli says: *"Designing professional user interfaces is not only a matter of a good graphic artist and some good ideas. Unfortunately, people creating user interfaces just go for a product, without even being aware of the basics or the theoretical principles behind it."* (Marinilli 2002)

When designing a user interface you always have to be aware that you create a product that is used by other people. Often, designers are too busy to create an award-winning, cool product without being aware that the product may be a complete mystery and unusable for the end users.



A user interface must be user-friendly (Schnabel 2008)

This unit will deal with some of the theory behind quality user interfaces, especially graphical user interfaces. We will show you what GUI controls exist and how you can get the user's attention on the screen.

1.2.1. Visual Communication

The three basic principles of visual communication according to (1992) are:

- **Organisation:** Give the user a simple, clear, and consistent conceptual structure.
- **Economy:** Maximise the effectiveness of a minimal set of tools.
- **Communication:** Adjust your presentation to the intake capacity of your users.



Designing a non-economic user Interface (Schnabel 2008)

1.2.2. User Interface Design

Design in General

"Design is inherently creative and unpredictable. " (Shneiderman et al. 2005)

(1985) characterize design as following: Design is a process; it is not a state and it cannot be adequately represented statically.

graphical User Interface - Layout and Design

GUI Design

An end user will never be able to read the designer's mind. What might seem an easy application for the designer team might be awkward and difficult to employ by the end user. (Marinilli 2002)



Product Designing (Schnabel 2008)

Desktop Metaphor

The function of a Graphical User Interface is to facilitate the handling of an application by means of graphical elements.

The design of today's graphical user interfaces often uses the so called "desktop metaphor".



Virtual Desktop (Desktop Metaphor)
(Wikipedia)

Real Desktop (Wikipedia)

The desktop metaphor itself has been extended and stretched with various implementations. Today, we find trash cans on the desktop as well as disks and filing cabinets.

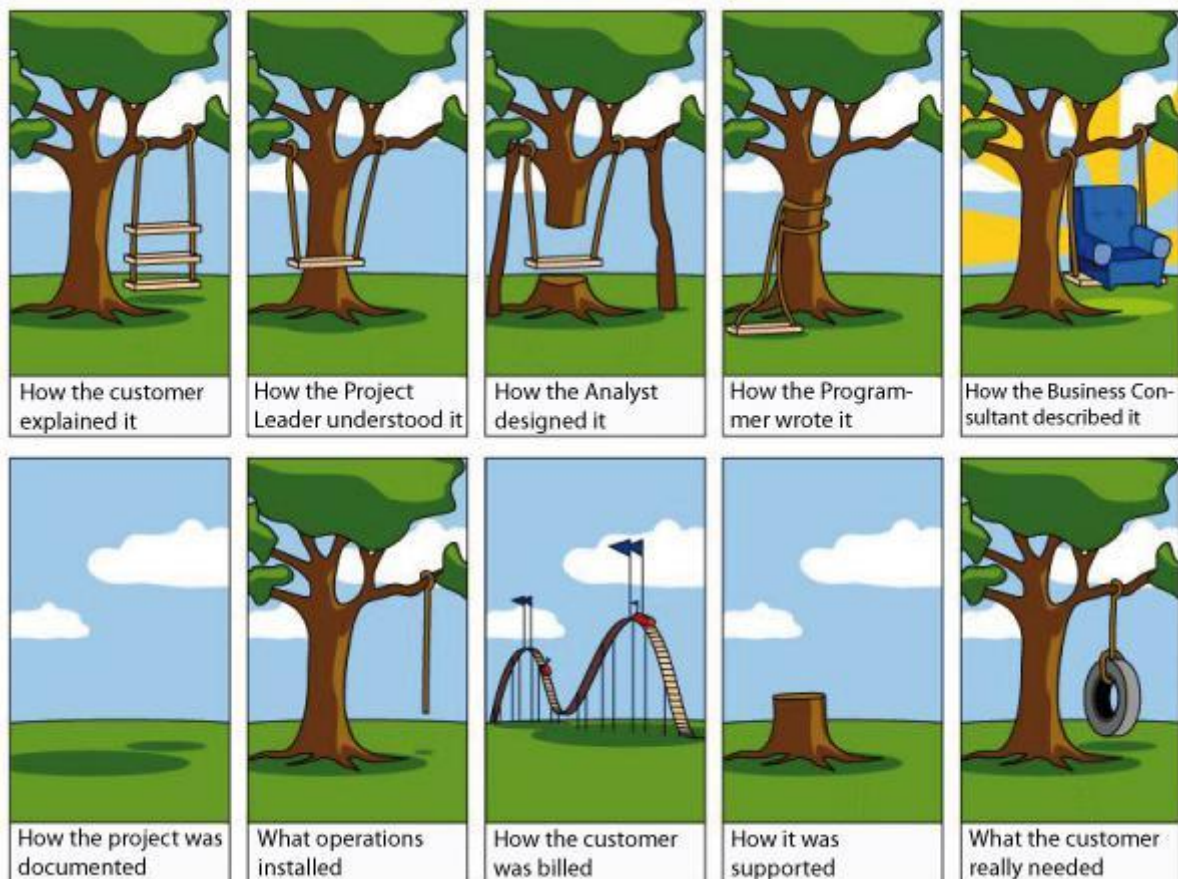
1.2.3. User-Centered Design

What is User-Centered Design?

The process from the idea of a project until its realization and usage, often ends up as following:

¹A metaphor is a figure of speech that implies comparison between two unlike entities, as distinguished from simile, an explicit comparison signaled by the words "like" or "as". The metaphor makes a qualitative leap from a reasonable, perhaps prosaic comparison, to an identification or fusion of two objects, to make one new entity partaking of the characteristics of both.

Graphical User Interface - Layout and Design



As User Interface Design should not end up (enabled.com)

User Centered-Design helps to avoid that processes end up as it is shown in the image above.

"User Centered-Design (UCD) is a design philosophy and a process in which the needs, wants, and limitations of the end-user of an interface or document are given extensive attention at each stage of the design process. an interface to understand intuitively what a first-time user of their design experiences, and what each user's learning curve may look like." (Wikipedia)

User Centered Design concerns itself with:

- **Usefulness**

Usefulness relates to relevance: do the functions, information, etc. match what the user actually needs? (Katz-Haas 1998)

- **Usability**

Usability relates to ease-of-use - a simple concept, but not always easy or intuitive to implement. (Katz-Haas 1998)

- **Visual Design**

Refers to layout recommendations and to the use of graphical elements.

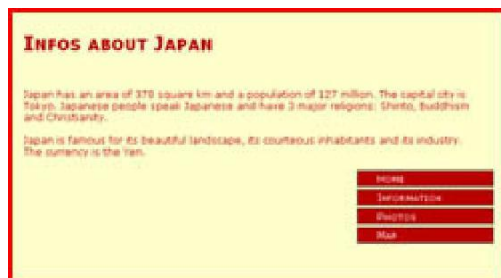
1.2.4. General Design Principles

According to (2005) there are eight principles, called "Golden Rules", that are applicable in most interactive systems.

Graphical User Interface - Layout and Design

- **Strive for consistency.** Inconsistencies force people to spend extra time trying to figure out how to find the answers to questions they have. This rule is the most frequently violated one, but following it can be tricky because there are many forms of consistency. Consistent sequences of actions should be required in similar situations; identical terminology should be used in prompts, menus, and help screens; and consistent color, layout capitalization, fonts, and so on should be employed throughout. Exceptions, such as required confirmation of the delete command or no echoing of passwords, should be comprehensible and limited in number.

The following examples show an inconsistent and a consistent thematic navigation.



Inconsistent Navigation



Consistent Navigation

- **Cater to universal usability.** Recognise the needs of diverse users and design for plasticity, facilitating transformation of content. Novice-expert differences, age ranges, disabilities, and technology diversity each enrich the spectrum of requirements that guides design. Adding features for novices, such as explanations, and features for experts, such as shortcuts and faster pacing, can enrich the interface design and improve perceived system quality.
- **Offer informative feedback.** For every user action, there should be system feedback. For frequent and minor actions, the response can be modest, whereas for infrequent and major actions, the response should be more substantial. Visual presentation of the objects of interest provides a convenient environment for showing changes explicitly.

The following examples show two applications which (do not) provide feedback for a minor action, namely clicking on a text.

Which two feedbacks are implemented in the top right example?

- . Text Highlighting
- . Hand symbol

- . **Design dialogs to yield closure.** Sequences of actions should be organized into groups with a beginning, middle, and end. Informative feedback at the completion of a group of actions gives operators the satisfaction of accomplishment, a sense of relief, the signal to drop contingency plans from their minds, and a signal to prepare for the next group of actions. For example, e-commerce web sites move users from selecting products to the checkout, ending with a clear confirmation page that completes the transaction.
- . **Prevent errors.** As much as possible, design the system such that users cannot make serious errors; for example, gray out menu items that are not appropriate and do not allow alphabetic characters in numeric entry fields. If a user makes an error, the interface should detect the error and offer simple, constructive, and specific instructions for recovery. For example, users should not have to retype an entire name-address form if they enter an invalid zip code, but rather should be guided to repair only the faulty part. Erroneous actions should leave the system state unchanged, or the interface should give instructions about restoring the state.
- . **Permit easy reversal of actions.** As much as possible, actions should be reversible. This feature relieves anxiety, since the user knows that errors can be undone, thus encouraging exploration of unfamiliar options. The units of reversibility may be a single action, a data-entry task, or a complete group of actions, such as entry of a name and address block.
- . **Support internal locus of control.** Experienced operators strongly desire the sense that they are in charge of the interface and that the interface responds to their actions. Surprising interface actions, tedious sequences of data entries, inability to obtain or difficulty in obtaining necessary information, and inability to produce the action desired all build anxiety and dissatisfaction.

Additionally, there is one basic principle which counts for all User Interface designers:

Keep your message as simple as possible. Use only the amount of text and graphics as is absolutely necessary to get your point across! (Skaalid 1999)

1.2.5. GUI Controls

A graphical user interface basically consists of windows and containers (Boxes/Panels/Panes). Within these elements, there are various GUI Control tools. There exist four groups of GUI Controls:

- . Input Elements (input field, slider, spin button, etc.)
- . Output Elements (output field, status bar, etc.)

- Action Elements (toggle button, etc.)



Design of GUI Controls

Many GUI Controls are buttons. A button is a widget which is clickable and which provides the user a simple way to trigger an event, such as selecting the zoom function of an interactive map (Wikipedia). Buttons are always decorated with button graphics (icons), which are placed inside the buttons. These icons identify the action, setting mode, or other function represented by the button.



Buttons in the Open Office application (OpenOffice.org 2006)

There are a few design guidelines for the designing of the button icons and the buttons themselves (Sun Developer Network (SDN)):

- Design icons to identify clearly the action represented by the button.
- Keep the drawing style of the icons symbolic; too much detail can make it more difficult for users to understand what the icon represents and thus what a button does.