**Project Contents**

1. **INTRODUCTION**
2. **SYSTEM SPECIFICATION**
3. **OVER VIEW OF THE LANGUAGE USED**
4. **SYSTEM ANALYSIS**
5. **SYSTEM DESIGN**
6. **CODING**
7. **TESTING**
8. **SECURITY**
9. **SCREENS LAYOUT**
10. **CONCLUSION**
11. **BIBLIOGRAPHY**

**Introduction**

Online Food Ordering System is a part of e-commerce. E-commerce or business through net means distributing, buying, selling, marketing, and servicing of products or services over electronic systems such as the Internet and other computer networks. Thus if we own a restaurant we need to upload menu online to attract potential customers

The online food ordering system gives restaurants the ability to increase sales and expand their business by giving customers the facility to order food online. With an online restaurant menu ordering system, customers can place orders online 24 \*7. Thus it is a simple, fast and convenient food ordering system giving an edge over the competition at an affordable price.

Internet has seen a tremendous growth in terms of coverage and awareness. So giving the business an online presence has become very crucial and important.With [Online Ordering System], we can set up we restaurant menu online and the customers can easily place order with a simple mouse click. Also with a food menu online we can easily track the orders, maintain customer's database and improve the food delivery service. We can receive order through e-mails/ fax or directly view on internet.

The restaurants’ can even customize online restaurant menu and upload images easily. Having restaurant menu on internet, potential customers can easily access it and place order at their convenience.

An online food ordering system is a web-based application that stimulates the foodies (customers) to put food orders through internet by locating their favorite restaurant or nearest one. This application is based on the asp.net platform .

**Overview**

The main objective of this project is to develop an application which gives provision to the restaurant owners to flourish their business by uploading menus at no cost and will invariably lead to higher customer retention and acquisition rates.

**Features**

* Online menus (original and searchable format)
* Provision of restaurant owners to register themselves with their menu.
* Easy lookup of restaurants in your area
* Check Ratings and Review the restaurants
* Simple, fast and convenient ordering of food
* Availability of the menu online 24\*7\*365 – no need to recite the complete menu over the phone. An online menu is ready to be viewed and printed by people worldwide.
* Accurate – no more spelling out the dishes’ names.
* Menu with the actual pictures of the product thereby adding to the uniqueness of your online presence.
* Prior knowledge of time for delivery helps prepare and provide better service.
* Provides base for online promotions, electronic coupons and gift certificates without needing the costly conventional media.
* Receive direct customer feedback and suggestions.
* Keep the customers informed.

**System Specification**

**HARDWARE DESCRIPTION**

The selection of hardware is very important in the existence and proper working of any software. When selecting hardware, the size and requirements are also important.

Minimum Requirements:

Processor : Pentium II class, 450MHz

RAM : 128MB

Hard Disk Drive : 1GB

Video : 800X600, 256 colors

CD-ROM : Required

The proposed System is developed on:

Processor : INTEL CORE 2 DUO

RAM : 2GB

Hard Disk Drive : 500 GB

Key Board : Standard 101/102 or Digi Sync Family

Monitor : Display Panel (1024 X 764)

Display Adapter : Trident Super VGA

Network Adapter : SMC Ethernet Card Elite 16 Ultra

Mouse : Logitech Serial Mouse

**SOFTWARE DESCRIPTION**

Operating System : Windows 7

Front- End : PHP

Back- End : MY SQL

Project will be done in PHP 5 as front end and MY SQL 5 as back end. NetBeans (IDE) is software that connects information, people, systems and devices. It spans clients, servers and developer tools and consists of:

* NetBeans is a software development platform written in Java. The NetBeans Platform allows applications to be developed from a set of modular software components called modules.
* NetBeans is cross-platform and runs on Microsoft Windows, Mac OS X, Linux, Solaris and other platforms supporting a compatible JVM.
* NetBeans IDE is an open-source integrated development environment.
* The NetBeans PHP editor provides code templates and code generation tools, such as "getter and setter" generation, refactoring, such as "instant rename", parameter tooltips, hints, and quick fixes, and smart code completion.

**Overview of the languages used**

**NETBEANS**

NetBeans is [Sun Microsystems](https://en.wikipedia.org/wiki/Sun_Microsystems) flagship software development product for computer programmers. It centers on an integrated distribution environment which has programmers create web application that run on any platforms supported by [Sun Microsystems](https://en.wikipedia.org/wiki/Sun_Microsystems). Supported platforms include Microsoft windows, servers and workstations, Pocket PC, Smart Phones and World Wide Web browsers not the Java Virtual Machine that all other java tools target.

**APACHE**

The **Apache HTTP Server**, colloquially called **Apache** , is the world's most used web server software. Originally based on the NCSA HTTPd server, development of Apache began in early 1995 after work on the NCSA code stalled. Apache played a key role in the initial growth of the World Wide Web, quickly overtaking NCSA HTTPd as the dominant HTTP server, and has remained most popular since April 1996. In 2009, it became the first web server software to serve more than 100 .

Apache is developed and maintained by an open community of developers under the auspices of the Apache Software Foundation. Most commonly used on a Unix-like system (usually Linux), the software is available for a wide variety of operating systems besides Unix,including MicrosoftWindows, NetWare, OpenVMS, OS/2, and TPF. Released under the Apache License, Apache is free and open-source software.

Apache supports a variety of features, many implemented as compiled modules which extend the core functionality. These can range from server-side programming language support to authentication schemes. Some common language interfaces support Perl, Python, Tcl, and PHP. Popular authentication modules include mod\_access, mod\_auth, mod\_digest, and mod\_auth\_digest, the successor to mod\_digest. A sample of other features include Secure Sockets Layer and Transport Layer Security support, a proxy module , a URL rewriting module (mod\_rewrite), custom log files (mod\_log\_config), and filtering support (mod\_include and mod\_ext\_filter).

**PDO**

*PHP Data Objects* (PDO) as an abstraction layer used for accessing databases, and even speech synthesis. Some of the language's core functions, such as those dealing with strings and arrays, are also implemented as extensions. The PHP Extension Community Library (PECL) project is a repository for extensions to the PHP language.

**BENEFITS OF PDO**

PDO offers several advantages over previous versions of PDO and over other data access components. These benefits fall into the following categories:

1. Interoperability
2. Maintainability
3. Programmability
4. Salability

**System Analysis**

**DEFINING A SYSTEM**

Collections of components, which are interconnected, and work together to realize some objective, form a system. There are three major components in every system, namely input, processing and output.

## Input Output

# Processing

**SYSTEMS LIFE CYCLE**

The sequencing of various activities required for developing and maintaining systems in an ordered form is referred as Systems Life Cycle. It helps in establishing a system project plan as it gives overall list of process and sub-processes required for developing any system. Here, the systems life cycle will be discussed with reference to the development of Employee Management System.

* Broadly, following are the different activities to be considered while defining the systems development cycle for the said project:
* Problem Definition
* Systems analysis
* Study of existing system
* Drawbacks of the existing system
* Proposed system
* Systems Requirement study
* Data flow analysis
* Feasibility study
* Systems design
* Input Design (Database & Forms)
* Updation
* Query /Report Design
* Administration
* Testing
* Implementation
* Maintenance

**SYSTEM ANALYSIS**

System analysis is a logical process; the objective of this phase is not actually to solve the problem but to determine what must be done to solve the problem. The basic objective of the analysis stage is to develop the logical model of the system using tools such as the data flow diagram and elementary data description of the elementary algorithm. The logical model is Subject to review by both the management and the user who agree that the model does in fact reflect what should be done to solve the problem.

System analysis is not a precise science. It is in fact more of an art, aided by scientific approach to find definition and recording data, gathering traditional structures is only one part of the system analysis, the next step is to examine the data, assess the situation and looking at the alternatives.

**ANALYSIS AND DEVELOPMENT OF THE ACTUAL SOLUTION**

A complete understanding of the requirement for the new system is very important for the successful development of a software product. Requirement Specification is the foundation in the process of software development .All further developments like system analysis; designing and coding will depend on how accurate and well documented the Requirement Specification is.   
Requirement specification appears to be a relatively simple task, but appearance is often deceiving. There is always a chance of wrong specification because of communication gap between the user and the Developer. Requirement Specification begins with a clear statement of the problem and the task to be performed. Then the requirement is described in a technical manner in precise statements. After the initial specification reports are received, they are analyzed and redefined through customer interaction.

**PROJECT OVERVIEW**

**PRODUCT PROSPECTIVE**

It will be able to manage information about Employee in more user friendly way. This system will manage Employees information at various field offices. User ID and password has been given to all the field offices so that they can enter their employee’s information into central database. Their access to the central database is restricted to their information only. Various reports based on the data entered by employees at field offices are generated at Head Quarter. These reports are helpful in Manpower management decisions.

**USER INTERFACE**

* The system will be having user privileges based menu.
* User will have to select the options form the given menu.
* The system will be entering the information into the database to generate reports.
* The forms will be designed to enter the data.
* Buttons will be used to insert, retrieve or modify the data.
* Links will be provided to shift from one form to another.

**MEMORY CONSTRAINTS**

No memory constraints are applicable. A normal memory configuration is more than sufficient.

**PRODUCT FUNCTION**

It is advisable to have weekly data backups. The system administrator will do the data recovery. Selection of panel is user-initiated operation, while indent handling is client initiated

**CONSTRAINTS**

**GENERAL CONSTRAINTS**

1. This system will not take care of any virus problem, which might occur either on the client or the server system. Avoiding the use of pirated software and ensuring that floppies and other removable media are scanned for viruses before use could minimize the possibility of viral infection.
2. Recovery of data after a system crash will be possible only if backups are taken at regular intervals.
3. Manual interfaces cannot be fully avoided. Documented proofs like dates etc. will have to be verified by the concerned staff before entering it into the computerized system

**HARDWARE CONSTRAINTS**

Constraints of the Internet & Intranet will be applicable to the system. The performance of the system will be dependent on the network conditions like network congestion, bandwidth etc. The primary memory (RAM) and the secondary memory (Hard Disk Space) requirement of the system at the client end will be the same as that required by the web browser and the operating system. At the server end memory requirements will be that of the server software (Operating system, Database Software, etc) and the space required to store the data. The space required to store the data would increase as more and more records are added to the system.

**SECURITY CONSTRAINTS**

User will be authenticated by the use of username and passwords. This does not Provide complete security and the system could be hacked into. Use of secure Socket Layer (SSL) is recommended. Use of SSL prevents any unauthorized access as all communications are encrypted. Valid Digital Certificates are required for this at the server end and the client web browser should have support for SSL.

**ASSUMPTION AND DEPENDENCIES**

1. It is assumed that the user is familiar with the basic computer fundamentals.
2. Timely backup of data should be taken to avoid data loss in case of system crash.
3. The use of pirated software should be avoided as it may lead to data loss and system crashes due to viral infections.
4. Floppies and other removable media should be scanned for viruses before use.
5. Proper configuration of the client, database server and network is necessary for the system to function as intended.
6. It is assumed that the maintenance of the database will be assigned to the authorized person only.
7. Only authorized persons will be allowed inside the server room.

**FEASIBILITY STUDY**

**Feasibility studies** aim to objectively and rationally uncover the strengths and weaknesses of the existing business or proposed venture, opportunities and threats as presented by the [environment](http://en.wikipedia.org/wiki/Natural_environment), the [resources](http://en.wikipedia.org/wiki/Resources) required to carry through, and ultimately the prospects for [success](http://en.wikipedia.org/wiki/Success). In its simplest terms, the two criteria to judge feasibility are [cost](http://en.wikipedia.org/wiki/Cost) required and [value](http://en.wikipedia.org/wiki/Value) to be attained. As such, a well-designed feasibility study should provide a historical background of the business or project, description of the [product](http://en.wikipedia.org/wiki/Product_%28business%29) or [service](http://en.wikipedia.org/wiki/Service_%28economics%29), accounting statements, details of the [operations](http://en.wikipedia.org/wiki/Operations) and [management](http://en.wikipedia.org/wiki/Management), [marketing research](http://en.wikipedia.org/wiki/Marketing_research) and policies, financial data, legal requirements and tax obligations. Generally, feasibility studies precede technical development and [project](http://en.wikipedia.org/wiki/Project) implementation.

The main objective of the feasibility study is to treat the technical, Operational, logical and economic feasibility of developing the computerized system. All systems are feasible, given unlimited resources and infinite time. It is both necessary and prudent to evaluate the feasibility of the project at System study phase itself. The feasibility study to be conduced for this project Involves.

1. Technical Feasibility
2. Operational Feasibility
3. Economic Feasibility
4. Logical Feasibility

**TECHNICAL FEASIBILITY**

Technical feasibility includes Risk Resources availability and technologies. The management provides latest hardware and software facilities for the successful completion of the projects. With these latest hardware and software support the system will perform extremely well. The system is available through Internet.

**OPERATIONAL FEASIBILITY**

In the existing manual system it is very difficult to maintain and update huge amount of information. The development of the system was started because of the requirement put forward by the management of the concerned department. This system, will handles the request in a better way and make the process easier thus, it is sure that the system developed is operationally feasible.

**ECONOMIC FEASIBILITY**

In the economic feasibility the development cost of the system is evaluated weighing it against the ultimate benefit derived from the new system. It is found that the benefit, from the new system would be more than the cost and time involved in its development.

**LEGAL FEASIBILITY**

In the legal feasibility it is necessary to check that the software we are going to develop is legally correct which means that the ideas which we have taken for the proposed system will be legally implemented or not. So, it is also an important step in feasibility study.

### Technology and system feasibility

The assessment is based on an outline design of system requirements in terms of Input, Processes, Output, Fields, Programs, and Procedures. This can be quantified in terms of volumes of data, trends, frequency of updating, etc. in order to estimate whether the new system will perform adequately or not. Technological feasibility is carried out to determine whether the company has the capability, in terms of software, hardware, personnel and expertise, to handle the completion of the project. When writing a feasibility report the following should be taken to consideration:

* A brief description of the business to assess more possible factor/s which could affect the study
* The part of the business being examined
* The human and economic factor
* The possible solutions to the problems

At this level, the concern is whether the proposal is both technically and [legally](http://en.wikipedia.org/wiki/Legally) feasible (assuming moderate cost).

**INFORMATION GATHERING**

First of all, what is information? For this inventory, the term “information” refers to data, including numerical data, facts or observations about the past or present, or projections about future possibilities. For example, you may need information from the most recent federal census in order to plan a marketing campaign. You may want to see projections concerning how many immigrant children to expect in your school district in the next five years. We tend to think of information as being readable or otherwise visible, but every fire expert knows that information also can be invisible, in the form of odors, pressures, and sounds. Identify your information gathering skills by thinking of the types of information you need, the places in which you gather it, and the ways in which you gather it.

We have taken an approach of gathering information with sensitivity and precautions.

The Information system designed for an organization must meet the requirements of the end users of the organization. To obtain what an enduser expects from the Information System the designer must gain complete knowledge of the organization’s working. It is important for the student to know the information gathering techniques so that no information is overlooked and the nature and functions of an organization are clearly understood. The main purpose of gathering information is to determine the information requirements of an organization. Information requirements are often not stated precisely by management. It is the analyst’s responsibility to prepare a precise Systems Requirements Specifications (SRS), which is easily understood (SRS) by users, as SRS document is a vital document before starting a project.

**INFORMATION GATHERING STRATEGIES**

A strategy should be evolved by the analyst to gather information. The strategy consists of identifying information sources, evolving a method of obtaining information from the identified sources and using an information flow model of organization.

**INFORMATION SOURCES**

The main sources of information are users of the system, forms and documents used in the organization, procedure manuals, rule books etc, reports used by the organization and existing computer programs(If Any).

We have collected the information about the current system from:

1. Reports
2. Personal staff
3. System Documentation
4. Trainees
5. Existing System

**INFORMATION GATHERING METHODS**

Searching for information

Information can be gathered by interviewing top-level management, middle level management and operational staff. Besides Interviews group discussions also help the analyst to gather information. It is not possible to obtain all information in a single interview, more than one interview is thus required.

**OTHER METHODS OF INFORMATION GATHERING**

Other methods of information search are:

* Systems used in other similar organization
* Observe workflow in workplace
* Repository of systems developed for similar organizations available.

**PROJECT PLANNING**

**Project planning** is part of [project management](http://en.wikipedia.org/wiki/Project_management), which relates to the use of [schedules](http://en.wikipedia.org/wiki/Schedule_%28project_management%29) such as [Gantt charts](http://en.wikipedia.org/wiki/Gantt_chart) to plan and subsequently report progress within the project environment.

Initially, the [project scope](http://en.wikipedia.org/wiki/Scope_%28project_management%29) is defined and the appropriate methods for completing the project are determined. Following this step, the [durations](http://en.wikipedia.org/wiki/Duration_%28project_management%29) for the various [tasks](http://en.wikipedia.org/wiki/Task_%28project_management%29) necessary to complete the [work](http://en.wikipedia.org/wiki/Work_%28project_management%29) are listed and grouped into a [work breakdown structure](http://en.wikipedia.org/wiki/Work_breakdown_structure). The logical [dependencies](http://en.wikipedia.org/wiki/Dependency_%28project_management%29) between tasks are defined using an [activity network diagram](http://en.wikipedia.org/wiki/Activity_network_diagram) that enables identification of the [critical path](http://en.wikipedia.org/wiki/Critical_path_method). [Float](http://en.wikipedia.org/wiki/Float_%28project_management%29) or slack time in the schedule can be calculated using [project management software](http://en.wikipedia.org/wiki/Project_management_software).[[2]](http://en.wikipedia.org/wiki/Project_planning#cite_note-1) Then the necessary [resources](http://en.wikipedia.org/wiki/Resource_%28project_management%29) can be [estimated](http://en.wikipedia.org/wiki/Estimate) and [costs](http://en.wikipedia.org/wiki/Cost) for each activity can be allocated to each resource, giving the total project cost. At this stage, the [project plan](http://en.wikipedia.org/wiki/Project_plan) may be optimized to achieve the appropriate balance between [resource usage](http://en.wikipedia.org/wiki/Resource_management) and project duration to comply with the project objectives. Once established and agreed, the plan becomes what is known as the baseline. Progress will be measured against the baseline throughout the life of the project. Analyzing progress compared to the baseline is known as [earned value management](http://en.wikipedia.org/wiki/Earned_value_management).

The inputs of the project planning phase include [Project Charter](http://en.wikipedia.org/wiki/Project_Charter) and the Concept Proposal. The outputs of the Project Planning phase include the Project Requirements, the Project Schedule, and the Project Management Plan.

Project planning is a discipline for stating how to complete a project within a certain timeframe, usually with defined stages, and with designated resources. One view of project planning divides the activity into:

* Setting objectives (these should be measurable)
* Identifying deliverables
* Planning the schedule
* Making supporting plans

Supporting plans may include those related to: human resources, communication methods, and [risk management](http://searchcio.techtarget.com/definition/enterprise-risk-management).

Computer hardware and software project planning within an [enterprise](http://searchwinit.techtarget.com/definition/enterprise) is often done using a project planning guide that describes the process that the enterprise feels has been successful in the past.

Tools popularly used for the scheduling part of a plan include the [Gantt chart](http://searchsoftwarequality.techtarget.com/definition/Gantt-chart) and the [PERT chart](http://searchsoftwarequality.techtarget.com/definition/PERT-chart).

**Beneficiaries and project management**

Planning should never start and end in an office or committee meeting. Project planning should never be done alone or in isolation from those who have to implement the plans, or who will benefit from them. In fact, the most successful and sustainable projects make an effort to involve those who are to benefit—in all stages of project planning and implementation. It is important to find out what the beneficiaries really think about the problem and about how to address it.

Project planning is done to increase the likelihood that a project will be implemented efficiently, effectively and successfully. Project planning covers the first three stages of "the project management cycle." This cycle, illustrated below, describes the various stages for conceptualising, planning, implementing and evaluating a project and recognises that even when a project is finished, it may provide the starting point for a new one.

1. **Conceptualise project scope and objectives**:

Explore the problem, identify priority needs, consider project solutions and evaluate organisational capacity.

1. **Plan the project**:

Establish the project scope; clarify goals and objectives; choose the most appropriate course of action; identify the inputs and resources required in terms of: people, materials, time and money; develop a budget and draft a project plan.

1. **Prepare project proposal:**

Present the project to important stakeholders, receive their feedback and secure the necessary material, human and financial resources.

1. **Implement the project:**

Implement the project by following a work-plan and completing pre-determined tasks and activities. Monitor progress and adjust as necessary.

1. **Evaluate the project**:

Review what has happened, consider the value of what has been achieved, and learn from that experience in order to improve future project planning.

**Determine project scope and objectives**

After selecting one solution to implement, project planners need to clearly establish the scope of the proposed project. A statement of the project scope should state broadly the general purpose and goals of the project. This broad statement should be followed by more specific objectives that will be met. The following excerpted from a 5-year project strategic plan (1998-2002) for Community Based Disaster Preparedness by the Bangladesh Red Crescent Society, provides an example of a project scope statement, a project goal statement and specific project objectives.

**PERT CHAT**

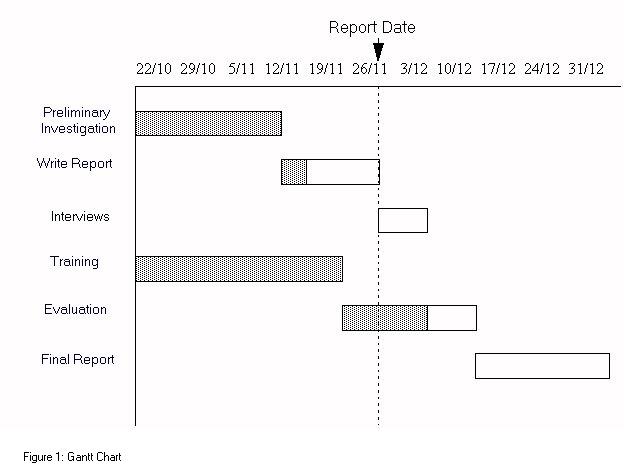
A PERT chart is a project management tool used to schedule, organize, and coordinate tasks within a project. PERT stands for Program Evaluation Review Technique, a methodology developed by the U.S. Navy in the 1950s to manage the Polaris submarine missile program. A similar methodology, the Critical Path Method (CPM) was developed for project management in the private sector at about the same time.

A PERT chart presents a graphic illustration of a project as a network diagram consisting of numbered nodes (either circles or rectangles) representing events, or milestones in the project linked by labelled vectors (directional lines) representing tasks in the project. The direction of the arrows on the lines indicates the sequence of tasks. In the diagram, for example, the tasks between nodes 1, 2, 4, 8, and 10 must be completed in sequence. These are called dependent or serial tasks. The tasks between nodes 1 and 2, and nodes 1 and 3 are not dependent on the completion of one to start the other and can be undertaken simultaneously. These tasks are called parallel or concurrent tasks. Tasks that must be completed in sequence but that don't require resources or completion time are considered to have event dependency. These are represented by dotted lines with arrows and are called dummy activities. For example, the dashed arrow linking nodes 6 and 9 indicates that the system files must be converted before the user test can take place, but that the resources and time required to prepare for the user test (writing the user manual and user training) are on another path. Numbers on the opposite sides of the vectors indicate the time allotted for the task.

The PERT chart is sometimes preferred over the Gantt chart, another popular project management charting method, because it clearly illustrates task dependencies. On the other hand, the PERT chart can be much more difficult to interpret, especially on complex projects. Frequently, project managers use both techniques.

**GANT CHAT**

A Gantt chart is a horizontal bar chart developed as a production control tool in 1917 by Henry L. Gantt, an American engineer and social scientist. Frequently used in project management, a Gantt chart provides a graphical illustration of a schedule that helps to plan, coordinate, and track specific tasks in a project.



Gantt charts may be simple versions created on graph paper or more complex automated versions created using project management applications such as Microsoft Project or Excel.

A Gantt chart is constructed with a horizontal axis representing the total time span of the project, broken down into increments (for example, days, weeks, or months) and a vertical axis representing the tasks that make up the project (for example, if the project is outfitting your computer with new software, the major tasks involved might be: conduct research, choose software, install software). Horizontal bars of varying lengths represent the sequences, timing, and time span for each task. Using the same example, you would put "conduct research" at the top of the verticle axis and draw a bar on the graph that represents the amount of time you expect to spend on the research, and then enter the other tasks below the first one and representative bars at the points in time when you expect to undertake them. The bar spans may overlap, as, for example, you may conduct research and choose software during the same time span. As the project progresses, secondary bars, arrowheads, or darkened bars may be added to indicate completed tasks, or the portions of tasks that have been completed. A vertical line is used to represent the report date.

Gantt charts give a clear illustration of project status, but one problem with them is that they don't indicate task dependencies - you cannot tell how one task falling behind schedule affects other tasks. The PERT chart, another popular project management charting method, is designed to do this. Automated Gantt charts store more information about tasks, such as the individuals assigned to specific tasks, and notes about the procedures. They also offer the benefit of being easy to change, which is helpful. Charts may be adjusted frequently to reflect the actual status of project tasks as, almost inevitably, they diverge from the original plan.

**SOFTWARE REQUIREMENT SPECIFICATION**

SRS is obtained after excessive discussions with the user. System requirements specification specifies what Information requirements will be provided. It does not specify how the system will be designed. Developing SRS is most important and difficult task of a Systems analyst.

**How SRS is developed**

Analyst examines the current system, finds out the shortcomings of the system as seen by the user. He then develops an SRS which is understandable by the user and which can be used for detailed design of the system.

**Ideal characteristics of SRS**

* Complete and Unambiguous.
* Specifies operational, tactical, and strategic information requirements
* Eliminates possible later disputes between users and Analyst
* Uses Graphical aids understood by users who are not computer literate and will also be useful in design.
* Jargon Free.

A **software requirements specification** (SRS) – a [requirements specification](http://en.wikipedia.org/wiki/Requirements_specification) for a [software system](http://en.wikipedia.org/wiki/Software_system) – is a complete description of the behavior of a system to be developed. It includes a set of [use cases](http://en.wikipedia.org/wiki/Use_case) that describe all the interactions the users will have with the software. In addition to use cases, the SRS also contains non-functional requirements. [Non-functional requirements](http://en.wikipedia.org/wiki/Non-functional_requirements) are requirements which impose constraints on the design or implementation (such as [performance engineering](http://en.wikipedia.org/wiki/Performance_engineering) requirements, [quality](http://en.wikipedia.org/wiki/Quality_%28business%29) standards, or design constraints).

Software requirements are a sub-field of [software engineering](http://en.wikipedia.org/wiki/Software_engineering) that deals with the elicitation, analysis, specification, and validation of [requirements](http://en.wikipedia.org/wiki/Requirements) for [software](http://en.wikipedia.org/wiki/Software).

The software requirement specification document enlists all necessary requirements for project development. To derive the requirements we need to have clear and thorough understanding of the products to be developed. This is prepared after detailed communications with project team and the customer. A general organization of an SRS is as follows

* Introduction
  + Purpose
  + Scope
  + Definitions
  + System overview
  + References
* Overall description
  + Product perspective
  + Product functions
  + User characteristics
  + Constraints, assumptions and dependencies
* Specific requirements
  + External interface requirements
  + Functional requirements
  + Performance requirements
  + Design constraints
  + Logical database requirement
  + Software System attributes
  + Other requirements

Software Requirement Specification (SRS) document usually contains a software vendor’s understanding of a customer’s software requirements. This document ensures that the software vendor and the customer are in agreement as to the features required in the software system being built. SRS is created after the initial requirement elicitation phase in which Software vendor interacts with the customer to understand the software needs. Usually SRS documentation is prepared by a business analyst who has some technical background.

An SRS is written in precise, clear and plain language so that it can be reviewed by a business analyst or customer representative with minimal technical expertise. However it also contains analytical models (use case diagrams, entity relationship diagrams, data dictionary etc.) which can be used for the detailed design and the development of the software system. SRS is one of the most critical pieces of software development since it acts as the bridge betweens the software developers and business analysts. An incomplete or incorrect SRS can have disastrous effects on a software project.

What is the need for an SRS document?

Software Requirements Specification is usually the first deliverable for any software project. As they say, first impression is the best impression!, and you should ensure that even the first draft of an SRS is of high quality.

The benefits of a good SRS are,

\* A contract between the customer and the software vendor – A good SRS document specifies all the features required in the final system including technical requirements and interface requirements. SRS document is used by the customer to determine whether the software vendor has provided all the features in the delivered software system. To the Software vendor it provides a solid foundation to fix the scope of the software system.

\* Enables costing and pricing of the project – A well defined SRS enables software developers to accurately estimate the amount of effort required to build the software product. Function point analysis and SMC are some the techniques adopted for estimating effort.

\* Input for detailed design – A good SRS enables experienced developers to convert the requirements directly to a technical design. For example, a well defined data dictionary can be easily converted to a database specification.

\* Management of customer expectations – Since SRS precisely defines project scope, it ensures that customer expectations don’t change during software development. If they do, SRS can be modified and costing/pricing can be done again on the changes required.

What are the contents of an effective SRS document?

There is no single precise template for writing good Software Requirement Specifications. The contents of an SRS document depends on the software product being developed and also on the expertise of the people doing the requirement elicitation. Different business/technology domains in a company usually have their own customized version of SRS template. Still a good Software Requirement Specification (SRS) usually contains project scope section, functional requirements, requirement analysis models, external interface requirements and non functional requirements. Each of these are explained below.

**Scope of the project/ Product vision:**

One of the most important items in the requirements specification is the precise scope definition of the project. Accuracy of this is important since SRS is also used for estimation and costing. This section should include a brief overview of the project and should also indicate the goals of the project including its benefits. Sometimes it is better to separate the project scope into a separate document.

If the project is for the development of a product, product vision defines the scope and the target user base of the product.

**Functional Requirements:**

Functional requirements specify the business requirements of the project in detail. Usually business requirements are specified in terms of the actions that user performs on the software system. This is known as the use case model. But not all requirements need to be specified as use cases. Functional requirements should contain a combination of use cases and plain textual description of system features. System features are specified at a higher level and use cases attempt to translate into user actions.

Again there is no fixed format for use case description, but it usually contains the following information,

\* Use case diagram – For a small systems, a single diagram can be used to depict all the use cases in the system.

\* List of actors and their details – This identifies the various types of users interacting with the software system.

\* Use case description – Purpose of the use case and how and when it is invoked by the user. This should also include an identifier for easy reference.

\* Preconditions – List of system states/conditions that must be true for the successful execution of the use case. This section is optional and could be easily incorporated into the basic steps section.

\* Basic steps – These indicates the various fine grained steps required for the execution of the use case.

\* Alternate steps – These indicate alternate events of the use case being described.

\* Business validations/rules – These indicates various types of input validations or business rules required in the use case being described.

\* Post conditions – Indicates the results of the use case. Please note that this section is optional and could be incorporated into the basic steps section.

To ensure that all the business requirements are addressed in the final software product, a traceability matrix document is used. Traceability matrix tracks each requirement through various phases of software development (detailed design, unit test plans, system testing plans, user acceptance test plans and code components). This requires that every requirement in the SRS should be identifiable by a unique number or tag.

For software projects where majority of features are available as user interfaces, it is better to complement this section with screen prototypes. These user interfaces can change during detailed design, but having a draft version of user interface in the requirements document helps a lot in communicating business requirements. However some customers insist on having finalized user interfaces in the requirements specification document.

**Requirement Analysis Models:**

Once the overall use cases in the system are identified in requirements elicitation, requirement analysis models can be developed to drill down to specifics of each requirement. For example, a use case such as “Add customer” may not specify all the customer details that needs to be captured by the system. This is usually specified in the data dictionary model and also in the screen prototype.

Requirement Analysis models act as the bridge between functional requirements and the detailed design of the software system. For example, Use cases lead to user interface design, data dictionary and entity relationship diagrams are used for designing database schema and class diagrams.

**Entity Relationship Diagrams:**

Entity relationship model diagram (ERD) is a conceptual representation of the data in a software system. During detail design this model is mapped in to the physical database model. There are different diagramming conventions available for creating ER diagrams. Following is a sample ERD in Crow’s foot notation (this is taken from the ERD of a course registration Web application requirements),

**Simple ERD diagram:**

This diagram indicates that there is one and only one instructor for a course and an instructor can have one or more courses. The relationship is captured as instructor “teaches” course.

**Data Dictionary:**

Data dictionary in a requirements document is an extension of the entity relationship diagrams. Which ER diagrams specify system entities and their relationships, a data dictionary lists all the attributes pertaining to each of those entities.

In a data dictionary, each attribute of the entity data in system is analyzed in detail including type of attribute, whether it is optional and a brief description of the attribute. Please see the sample SRS template section for more details.

In addition to the above models, sometimes it is useful to develop state transition diagrams and data flow diagrams. To describe a complex process flow or a workflow in the application, process flow diagrams or flowcharts can be used.

**External Interface Requirements:**

It is very rare that we have a standalone software system. Usually a software system interacts with a number of external applications for data input and output. For example, an e-business application usually needs to be integrated to an external payment gateway. All the external interface requirements are detailed in this section. The important thing to document here are the entities that are passed across the external interfaces.

**Non Functional Requirements:**

Non functional or technical requirements specify how the software system should operate. In contrast functional requirements specify what a software system should do. Some of the non functional requirements are derived from the functional requirements. Non functional requirements captured include performance requirements, application scalability, application security, maintainability, usability, availability, logging and auditing, data migration requirements, multi lingual support etc. Please note that only a subset of the list are applicable for a specific project.

**SOFTWARE ENGINEERING PARADIGM APPLIED**

Software Engineering is the establishment and use of sound engineering principles in order toobtain economically software that is both reliable and works efficiently on real systems.To solve actual problems in an industry / organizations setting, such as Central or State Govt.Sectors, Public / private Sectors, Colleges, Schools, etc., a Software Engineer or a team of engineers must incorporate a development strategy that encompasses the process, methods,tools layers, and the generic phases. This strategy is referred to as a Process Model or aSoftware Engineering Paradigm. Models used for system development in project “Custom er Relationship Management and Sales Force Automation” , is Prototyping. Descriptions of these models are as following-

**PROTOTYPE MODEL**

Prototyping is the process of building a model of a system. In terms of an information system, prototypes are employed to help system designers build an information system that intuitiveand easy to manipulate for end users. Prototyping is an iterative process that is part of the[analysis phase](http://www.colorado.edu/infs/jcb/class/analysis.html)of the[systems development life cycle.](http://www.colorado.edu/infs/jcb/class/sysdevel.html)  During the requirements determination portion of the systems analysis phase, system analystsgather information about the organization's current procedures and business processes relatedthe proposed information system. In addition, they study the current information system, if there is one, and conduct user interviews and collect documentation. This helps the analystsdevelop an initial set of system requirements.Prototyping can augment this process because it converts these basic, yet sometimesintangible, specifications into a tangible but limited working model of the desired informationsystem. The user feedback gained from developing a physical system that the users can touchand see facilitates an evaluative response that the analyst can employ to modify existingrequirements as well as developing new ones.

The prototyping model is a[software development process](http://en.wikipedia.org/wiki/Software_development_process)that begins with requirementscollection, followed by prototyping and user evaluation. Often the end users may not be ableto provide a complete set of application objectives, detailed input, processing, or outputrequirements in the initial stage. After the user evaluation, another prototype will be builtbased on feedback from users, and again the cycle returns to customer evaluation. The cyclestarts by listening to the user, followed by building or revising a mock-up, and letting the usertest the mock-up, then back.

**Some Advantages of Prototyping:**

* Reduces development time.
* Reduces development costs.
* Requires user involvement.
* Developers receive quantifiable user feedback.
* Facilitates system implementation since users know what to expect.
* Results in higher user satisfaction.
* Exposes developers to potential future system enhancements.

Software Engineering (SE) is a field without too much historic background since it is less than four decades old. This idea can be supported by means of the fact that its first publications and conferences were held in the late 1960s (in the 1950s there was certain research but it was covered of confusion and without any significant publication) and its academic presence did not begin to separate from Computer Science until the early 1980s; this event meant a important fact in SE research developing. This youth of the discipline of SE is resulted in an immaturity of this research field. This immaturity is verified because research which is carried out in this discipline has several deficiencies like the lack of systematic rigorous method, the lack of the evident methods of validation, etc. We can say SE research still lacks suitable scientific precision (meaning Science in a wide sense of the word and without drawing a distinction between Science, Engineering, Technology, etc.).

Besides, each scientific discipline has a certain object of study which distinguishes the process to be followed in the research. The problem of SE research basically takes root in the fact that it is not so evident which the object of study is or rather the problem is that there are several objects of study with different nature and so, there are also different research and validation processes. This problem is motivating a rising concern with research methods and the validation in SE.

The nature of SE research, basically Science or Engineering, depends on its object of study. Seen this way, and according to the object of study, the research process will be different so that the kinds of problems can be tackled by means of different research methods and even by means of different paradigms.

However, which is the object of study of the Science in contrast to the object of study of the Engineering? The fundamental difference between Science and Engineering is that while the Science pays attention to natural aspects, the

Engineering is concerned with artificial aspects [1]; that is to say, while the former deals with the study of what things are like, the latter is concerned with what they should be like in order to make it possible to construct new objects. Nevertheless, both Science and Engineering are knowledge and action since in the same way Science is also action, not just knowledge; correlatively, we may say that Engineering is also knowledge, and not just application. The difference between Science and Engineering lies in the modes of knowledge and action that they develop, not in one of them knowing and the other applying [15]. In other words, Science and Engineering will differ in the research process which is used in each one. Therefore, Engineering research differs greatly from traditional scientific research because while Sciences deal with the study of existing objects and phenomena, be it physically, metaphysically or conceptually, Engineering is based on how to do things, how to create new objects.

Applying this reflection to SE, how do we consider it, like Science or like Engineering? Opinions by several authors are not conclusive, being necessary some prerequisites so that SE can be a real Engineering. To answer this question we take as reference point other fields with a bigger historic background and, also with more maturity, such as Electronics, Chemistry or Geology; in such fields we may talk as Science as Engineering. So, in the same way Electronic Physics and Electronic Engineering, Chemistry and Chemical Engineering, Geology and Geological Engineering coexist and making a simile with these fields, we can say that SE has a double nature, of Science and Engineering, depending on the object of study; this fact determines the research process. Therefore, Software Science (using Blum terminology) comes from Computer Science and pays attention to the aspects that have to do with the study of built artefacts, like code or other kind of artefacts such as models, documents, etc; this Science deals with problems such as algorithmic complexity, software metrics, testing techniques, etc. On the other hand, we can find another kind of research problems (Engineering problems), supported by Software Science, concerned with the creation of software artefacts and we could define it as the study of transforming ideas into operations.

So, whereas Engineering problems deal with the creation of new artifacts,

Scientific problems deal with the study of existing ones. Note, that an Engineering problem becomes a Scientific problem once the object has been created; when a new artifact is created (for example a new model), by an Engineering research process, this new artifact becomes an existing one being an object of study by a scientific research process (for example, studying its correctness, its quality, etc.)

Although there are other classifications of problems in SE, these classifications are not appropriate since they are not focused on the research process. Thus, for instance, the SE Body of Knowledge project (SWEBOK [20]) sets out areas as: Software design, Software construction, SE tools and methods, etc. This schema is accurate enough but it is mainly centred on the creation of a body of knowledge with educative aims. Then, it provides invalid areas of knowledge for a classification of research problems because we can find scientific and engineering problems in each area. In this regard, for example, in SE tools and methods area, it is not the same trying to create a new method than testing a previously created method. In Software design, you can either create new models to improve the design process or study existing models and analyse their implantation and use in a company. We notice the former case is an Engineering problem while the latter case is an empirical or socio-cultural problem but all of them are included in the same area of the SWEBOK. This inclusion into the same area makes its difference in the research field impossible.

Concluding, we propose a classification of the research problems of the SE

discipline in two kinds, which differ in the nature of its object of study: those with a Scientific nature and those with an Engineering nature. As both kinds of research problems have different objects of study, they should use different research paradigms and different research methods.

**Research paradigms**

To carry out research in any previously-mentioned field, it is important to set out a paradigm for any research a researcher wants to realize. Thus, we may consider descriptive paradigms (evaluative-deductive or positivist paradigm, evaluativeinterpretive or interpretive paradigm, evaluative-critical or critical paradigm, etc,) and formulative paradigms (formulative model, formulative-process, method, algorithm, etc.). These paradigms change depending on nature of the research problem: Scientific or Engineering.

To broach Scientific research problems, evaluative paradigms are the most used, either positivist paradigms (used in empirical sciences) or interpretive or constructive paradigms (used in social and cultural problems). In this regard, for instance, we could apply positivist paradigms to testing or interpretive paradigms to organization processes which are necessary for the implantation of a tool. Although there are other paradigms, even combinations of paradigms which can give rise to mixed paradigms, they always present characteristics which allow us to include them in a behavioural- Science research paradigm.

On the other hand, to broach Engineering research problems, descriptive paradigms are used and these paradigms interact with positivist and interpretive paradigms. Thus, for instance, by means of literature reviews the researcher can try to establish the weak spots of a model and its respective technique of creation and afterwards, can try to establish a description of a new technique and the new built model.

**Research methods**

Depending on the kind of problem to solve and the context of the problem, Science or Engineering, different research methods are used. Moreover, scientific research methods can not always be applied to engineering research problems.

Scientific research problems are similar to problems broached in traditional sciences and can have either an empirical or a cultural and social nature. When the Science has an empirical nature, quantitative research methods can be applied; these methods try to solve problems like: “what model method is more efficient?”. When the Science has a social and cultural nature, qualitative research methods can be applied and these methods can seek to answer questions like: “what factors make a given software process unacceptable to the company?” or “why is one information systems development tool more acceptable than another?”. In both, it is necessary certain knowledge of the reality: the object of study is an existing object in the world. Thus, this kind of problems use the research methods proposed by traditional sciences, as they study phenomena and objects of the world regardless of how they were created.

However, there is not any precise method to broach Engineering research problems and the search for a method appropriate to this field is becoming a research field in its own right. The solution of problems purely concerning Engineering requires methods of a different kind since in these cases it is directly

possible to apply neither empirical methods nor methods which have to do with social and cultural component as the object of study does not yet exist. Furthermore, in the case of Engineering, it is necessary a major component of creativity, which makes it difficult to draw up a universal method for solving problems within this field. For instance, “what research method would be valid for the specification of a new methodology for software development?”. It would be necessary to study existing methodologies, reflecting on them to determine their advantages and disadvantages and proposing a new one, which, while retaining the advantages of the methodologies studied, would, as far as possible, lack their shortcomings. Arriving at a better final proposition would largely depend on the creativity and common sense applied to the construction of the new method. This method is proposed by and it is stated the method it is applied in Engineering consist in the formulation of experiences and the identification of the best practices.

**SOFTWARE DEVELOPMENT METHODOLOGY**

A **software development methodology** or **system development methodology** in [software engineering](http://en.wikipedia.org/wiki/Software_engineering) is a framework that is used to structure, plan, and control the [process of developing](http://en.wikipedia.org/wiki/Software_development_process) an [information system](http://en.wikipedia.org/wiki/Information_system).

The software development methodology (also known as SDM) framework didn't emerge until the 1960s. According to Elliott (2004) the [systems development life cycle](http://en.wikipedia.org/wiki/Systems_development_life_cycle) (SDLC) can be considered to be the oldest formalized [methodology framework](http://en.wikipedia.org/w/index.php?title=Methodology_framework&action=edit&redlink=1) for building [information systems](http://en.wikipedia.org/wiki/Information_system). The main idea of the SDLC has been "to pursue the development of information systems in a very deliberate, structured and methodical way, requiring each stage of the [life cycle](http://en.wikipedia.org/wiki/Software_development_process) from inception of the idea to delivery of the final system, to be carried out rigidly and sequentially" within the context of the framework being applied. The main target of this methodology framework in the 1960s was "to develop large scale functional [business systems](http://en.wikipedia.org/wiki/Business_system) in an age of large scale business conglomerates. Information systems activities revolved around heavy [data processing](http://en.wikipedia.org/wiki/Data_processing) and [number crunching](http://en.wikipedia.org/wiki/Number_crunching) routines".

A software development methodology is a framework that is used to structure, plan, and control the [process of developing](http://en.wikipedia.org/wiki/Software_development_process) an [information system](http://en.wikipedia.org/wiki/Information_system) - this includes the pre-definition of specific deliverables and artifacts that are created and completed by a project team to develop or maintain an application.

Every software development methodology framework acts as a basis for applying specific approaches to develop and maintain software. Several software development approaches have been used since the origin of information technology. These are:

* Waterfall: a linear framework
* Prototyping: an iterative framework
* Incremental: a combined linear-iterative framework
* Spiral: a combined linear-iterative framework
* Rapid application development (RAD): an iterative framework

### Waterfall Development

The [Waterfall model](http://en.wikipedia.org/wiki/Waterfall_model) is a sequential development approach, in which development is seen as flowing steadily downwards (like a waterfall) through the phases of requirements analysis, design, implementation, testing (validation), integration, and maintenance. The first formal description of the method is often cited as an article published by [Winston W. Royce](http://en.wikipedia.org/wiki/Winston_W._Royce) in 1970 although Royce did not use the term "waterfall" in this article.

The basic principles are:

* Project is divided into sequential phases, with some overlap and splashback acceptable between phases.
* Emphasis is on planning, time schedules, target dates, budgets and implementation of an entire system at one time.
* Tight control is maintained over the life of the project via extensive written documentation, formal reviews, and approval/signoff by the user and information technology management occurring at the end of most phases before beginning the next phase.

The Waterfall model is a traditional engineering approach applied to software engineering. It has been widely blamed for several large-scale government projects running over budget, over time and sometimes failing to deliver on requirements due to the [Big Design Up Front](http://en.wikipedia.org/wiki/Big_Design_Up_Front) approach. Except when contractually required, the Waterfall model has been largely superseded by more flexible and versatile methodologies developed specifically for software development. See [Criticism of Waterfall model](http://en.wikipedia.org/wiki/Waterfall_model#Criticism).

### Prototyping

[Software prototyping](http://en.wikipedia.org/wiki/Software_prototyping), is the development approach of activities during [software development](http://en.wikipedia.org/wiki/Software_development), the creation of prototypes, i.e., incomplete versions of the software program being developed.

The basic principles are:[[2]](http://en.wikipedia.org/wiki/Software_development_methodology" \l "cite_note-CMS08-1)

* Not a standalone, complete development methodology, but rather an approach to handling selected parts of a larger, more traditional development methodology (i.e. incremental, spiral, or rapid application development (RAD)).
* Attempts to reduce inherent project risk by breaking a project into smaller segments and providing more ease-of-change during the development process.
* User is involved throughout the development process, which increases the likelihood of user acceptance of the final implementation.
* Small-scale mock-ups of the system are developed following an iterative modification process until the prototype evolves to meet the users’ requirements.
* While most prototypes are developed with the expectation that they will be discarded, it is possible in some cases to evolve from prototype to working system.
* A basic understanding of the fundamental business problem is necessary to avoid solving the wrong problem.

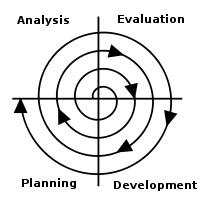
### Incremental Development

Various methods are acceptable for combining linear and iterative systems development methodologies, with the primary objective of each being to reduce inherent project risk by breaking a project into smaller segments and providing more ease-of-change during the development process.

The basic principles are:

* A series of mini-Waterfalls are performed, where all phases of the Waterfall are completed for a small part of a system, before proceeding to the next increment, or
* Overall requirements are defined before proceeding to evolutionary, mini-Waterfall development of individual increments of a system, or
* The initial software concept, requirements analysis, and design of architecture and system core are defined via Waterfall, followed by iterative Prototyping, which culminates in installing the final prototype, a working system.

### Spiral Development

[](http://en.wikipedia.org/wiki/File:Software_Development_Spiral.svg)

The [spiral model](http://en.wikipedia.org/wiki/Spiral_model) is a [software development process](http://en.wikipedia.org/wiki/Software_development_process) combining elements of both [design](http://en.wikipedia.org/wiki/Design) and [prototyping](http://en.wikipedia.org/wiki/Prototyping)-in-stages, in an effort to combine advantages of [top-down and bottom-up](http://en.wikipedia.org/wiki/Top-down_and_bottom-up_design) concepts. It is a meta-model, a model that can be used by other models.

The basic principles are:

* Focus is on risk assessment and on minimizing project risk by breaking a project into smaller segments and providing more ease-of-change during the development process, as well as providing the opportunity to evaluate risks and weigh consideration of project continuation throughout the life cycle.
* "Each cycle involves a progression through the same sequence of steps, for each part of the product and for each of its levels of elaboration, from an overall concept-of-operation document down to the coding of each individual program."
* Each trip around the spiral traverses four basic quadrants: (1) determine objectives, alternatives, and constraints of the iteration; (2) evaluate alternatives; Identify and resolve risks; (3) develop and verify deliverables from the iteration; and (4) plan the next iteration.
* Begin each cycle with an identification of stakeholders and their win conditions, and end each cycle with review and commitment.

### Rapid Application Development

[Rapid application development](http://en.wikipedia.org/wiki/Rapid_application_development) (RAD) is a software development methodology, which involves iterative development and the construction of [prototypes](http://en.wikipedia.org/wiki/Prototype). Rapid application development is a term originally used to describe a [software development process](http://en.wikipedia.org/wiki/Software_development_process) introduced by [James Martin](http://en.wikipedia.org/wiki/James_Martin_%28author%29) in 1991.

The basic principles are:

* Key objective is for fast development and delivery of a high quality system at a relatively low investment cost.
* Attempts to reduce inherent project risk by breaking a project into smaller segments and providing more ease-of-change during the development process.
* Aims to produce high quality systems quickly, primarily via iterative Prototyping (at any stage of development), active user involvement, and computerized development tools. These tools may include [Graphical User Interface](http://en.wikipedia.org/wiki/Graphical_User_Interface) (GUI) builders, [Computer Aided Software Engineering](http://en.wikipedia.org/wiki/Computer_Aided_Software_Engineering) (CASE) tools, [Database Management Systems](http://en.wikipedia.org/wiki/Database_Management_System) (DBMS), [fourth-generation programming languages](http://en.wikipedia.org/wiki/Fourth-generation_programming_language), code generators, and object-oriented techniques.
* Key emphasis is on fulfilling the business need, while technological or engineering excellence is of lesser importance.
* Project control involves prioritizing development and defining delivery deadlines or “timeboxes”. If the project starts to slip, emphasis is on reducing requirements to fit the timebox, not in increasing the deadline.
* Generally includes [joint application design](http://en.wikipedia.org/wiki/Joint_application_design) (JAD), where users are intensely involved in [system design](http://en.wikipedia.org/wiki/System_design), via consensus building in either structured workshops, or electronically facilitated interaction.
* Active user involvement is imperative.
* Iteratively produces production software, as opposed to a throwaway prototype.
* Produces documentation necessary to facilitate future development and maintenance.
* Standard systems analysis and design methods can be fitted into this framework.

**System Design**

**DEFINITION**

The most creative and challenging face of the system development is System Design. It provides the understanding and procedural details necessary for implementing the system recommended in the feasibility study. Design goes through the logical and physical stages of development.

In designing a new system, the system analyst must have a clear understanding of the objectives, which the design is aiming to fulfill. The first step is to determine how the output is to be produced and in what format. Second, input data and master files have to be designed to meet the requirements of the proposed output. The operational phases are handled through program construction and testing.

Design of a system can be defined as a process of applying various techniques and principles for the purpose of defining a device, a process or a system in sufficient detail to permit its physical realization. Thus system design is a solution to “how to” approach to the creation of a new system. Thus important phase provides the understanding and the procedural details necessary for implementing the system recommended in the feasibility study. The design step provides a data design, architectural design, and a procedural design.

Design is the most important part of the development phase for any product or systembecause design is the place where quality is fostered. Design is the only thing, which accurately translates a customer’s requirement in to a finished software product or system. The design step produces a data design, an architectural design, and Interface Design and a ProceduralDesign.System specialists often refer to this stage as logical description, in contrast to the process of developing the program software, which is referred to as physical design. The system designdescribes the data to be input, calculated or stored. Individual data items and calculationprocedures are written in detail. The procedures tell how to process the data and produce theoutput.

**REQUIREMENT ANALYSIS**

  Requirement Analysis is the first technical step in the software engineering process. It is at thispoint that a general statement of software scope is refined into a concrete specification thatbecomes a foundation for all the software engineering activities that follow.Analysis must focus on the informational, functional, and behavioral domains of a problem. Tobetter understand what is required, models are created, the problem is partitioned, andrepresentation that depict the essence of requirements and later, implementation detail, aredeveloped.

**REQUIREMENT DETERMINATION**

An information system is intended to meet the diverse needs of an organization.The customer relationship management system and sale automation system is required tocontain the following features:

* It should contain all the information about the customer such as name, address, salary,occupation, date of birth, phone number.
* It should maintain product details such as product name, features, price, modelnumber, colors.
* It should maintain salesman data such as salesman name, address, salary, occupation,date of birth, phone number, date of joining, manager name.
* It should also maintain the quota of each salesman.
* It should maintain the transaction of products such as sale of the products, addition of the product in the organization etc.
* It should be able to notify that the product stock is about to finish and reorder isrequired.
* It should check the performance of the salesman by comparing it with some standardsset.
* It should feature the security check so that only the authorized person is allowed tomanipulate database as per the level assigned.
* It should be user-friendly software.

**FUNCTIONAL REQUIREMENT**

Data should be given in a correct form in order to avoid getting erroneous results. The value of transportation cost and the assignment cost should be known prior to calculate the optimaltransportation and assignment cost.

* In order to prepare the budgets the proper and the correct data is to be considered.
* The end users should input correct figures so that the Ratio Analysis may result as perexpectations.
* Data should be checked for validity
* Consistency between the information provided in different forms under a schemeshould be checked
* Messages should be given for improper input data, and invalid data item should beignored.

**PERFORMANCE REQUIREMENTS**

The following characteristics were taken care of in developing the system.

1. USER FRIENDLINESS:

The system is easy to learn and understand. A naive user can alsouse the system effectively, without any difficulty.

2. RESPONSE TIME:

Response time of all the operations is kept low. This has been madepossible by careful planning.

3. ERROR HANDLING:

Response to user errors and undesired situations have been takencare of to ensure that system operates without halting in case of such situation and propererror messages are given to user. Different validations are on different field so that onlycorrect data will be entering in the database.

4. SAFETY:

The system is able to avoid catastrophic behavior.

5. ROBUSTNESS:

The system recovers from the undesired events without humanintervention.

6. SECURITY:

The system provides protection of information through the mechanism of password incorporated in it. Therefore only authorized people can access it database.There are three category of user in the system administrator, salesman and customerservice executive. All have different authorities of using the system.

7. ACCURACY:

The system is highly accurate thus its utility is very high.

8. COST ELEMENT:

  Servicing a given demand in the system doesn’t require a lot of money.

**5.2. OUTPUT DESIGN**

In the output design, the emphasis is on producing a hard copy of the information requested or displaying the output on the CRT screen in a predetermined format. Two of the most output media today are printers and the screen. Most users now access their reports from a hard copy or screen display. Computer’s output is the most important and direct source of information to the user, efficient, logical, output design should improve the systems relations with the user and help in decision-making.

As the outputs are the most important source of information to the user, better design should improve the system’s relation and also should help in decision-making. The output device’s capability, print capability, print capability, response time requirements etc should also be considered form design elaborates the way output is presented and layout available for capturing information. It’s very helpful to produce the clear, accurate and speedy information for end users.

**5.3. INPUT DESIGN**

In the input design, user-oriented inputs are converted into a computer based system format. It also includes determining the record media, method of input, speed of capture and entry on to the screen. Online data entry accepts commands and data through a keyboard. The major approach to input design is the menu and the prompt design. In each alternative, the user’s options are predefined. The data flow diagram indicates logical data flow, data stores, source and destination. Input data are collected and organized into a group of similar data. Once identified input media are selected for processing.

In this software, importance is given to develop Graphical User Interface (GUI), which is an important factor in developing efficient and user-friendly software. For inputting user data, attractive forms are designed. User can also select desired options from the menu, which provides all possible facilities.

Also the important input format is designed in such a way that accidental errors are avoided. The user has to input only just the minimum data required, which also helps in avoiding the errors that the users may make. Accurate designing of the input format is very important in developing efficient software. The goal or input design is to make entry as easy, logical and free from errors.

**5.4. LOGICAL DESIGN**

Logical data design is about the logically implied data. Each and every data in the form can be designed in such a manner to understand the meaning. Logical data designing should give a clear understanding and idea about the related data used to construct a form.

# Data Integrity

**Data Integrity** in its broadest meaning refers to the trustworthiness of [system resources](http://en.wikipedia.org/wiki/System_resource) over their entire [life cycle](http://en.wikipedia.org/wiki/Life_cycle). In more analytic terms, it is "the representational faithfulness of [information](http://en.wikipedia.org/wiki/Information) to the true state of the object that the information represents, where representational faithfulness is composed of four essential qualities or core attributes: completeness, currency/timeliness, accuracy/correctness and validity/authorization."[[1]](http://en.wikipedia.org/wiki/Data_integrity#cite_note-0) The concept of business rules is already widely used nowadays and is subdivided into six categories which include data rules. Data is further subdivided Data Integrity Rules, [data sourcing](http://en.wikipedia.org/w/index.php?title=Data_sourcing&action=edit&redlink=1) rules, [data extraction](http://en.wikipedia.org/wiki/Data_extraction) rules, [data transformation](http://en.wikipedia.org/wiki/Data_transformation) rules and [data deployment](http://en.wikipedia.org/w/index.php?title=Data_deployment&action=edit&redlink=1) rules.

Data Integrity is very important in [database](http://en.wikipedia.org/wiki/Database) operations in particular and [Data Warehousing](http://en.wikipedia.org/wiki/Data_Warehousing) and [Business Intelligence](http://en.wikipedia.org/wiki/Business_Intelligence) in general. Because Data Integrity ensured that data is of high quality, correct, consistent and accessible, in is important to follow rules governing Data Integrity.

A [Data Value](http://en.wikipedia.org/wiki/Data_value) Rule or Conditional Data Value Rule specifies data domains. The difference between the two is that the former specifies the domain of allowable values for a data attribute which applies to all situation while the latter does not apply to all situations but only when there exceptions or certain conditions that applies.

[Data Structure](http://en.wikipedia.org/wiki/Data_structure) Rule defines that cardinality of data for a data relation in cases where there are no conditions of exceptions which apply. This rule makes data structure very easy to understand. A conditional data structure rule is slightly different in that is governs when conditions or exceptions apply on data cardinality for a data relation.

A [Data Derivation](http://en.wikipedia.org/w/index.php?title=Data_derivation&action=edit&redlink=1) Rule specifies the how a data value is derived based on algorithm, contributors and conditions. It also specifies the conditions on how the data value could be re-derived.

A [Data Retention](http://en.wikipedia.org/wiki/Data_retention) Rule specifies the length of time of data values which can be retained in a particular database. It is specifies what can be done with data values when its use for a database expires A data occurrence retention rule specifies the length of time the data occurrence is retained and what can be done with data when it is no longer useful. A data attribute retention rule is similar to a data retention rule but the data attribute retention rule only applies to specific data values rather than the entire data occurrence.

These Data Integrity Rules, like any other rules, are totally without meaning when they are not implemented and enforced.

In order to achieve Data Integrity, these rules should be consistently and routinely applied to all data which are entering the Data Warehouse or any Data Resource for that matter. There should be no waivers or exceptions for the enforcement of these rules because any slight relaxation of enforcement could mean a tremendous error result.

As much as possible, these Data Integrity Rules must be implemented in as close to the initial capture of data so that early detection and correction of potential breach of integrity can be taken action. This can greatly prevent errors and inconsistencies from entering the database.

With strict implementation and enforcement of these Data Integrity Rules, data error rates could be much lower so less time is spent on trying to troubleshoot and trace faulty computing results. This translates to savings from manpower expense.

Since there is low error rate, there can only be high quality data that can be had to provide better support in the statistical analysis, trend and pattern spotting, and decision making tasks of a company. In today's digital age, information one major key to success and having the right information means having better edge over the competitors.

Most narrowly, data with integrity has a complete or whole structure. All characteristics of the data including [business rules](http://en.wikipedia.org/wiki/Business_rule), rules for how pieces of data relate, dates, definitions and lineage must be correct for data to be complete.

Per the discipline of data architecture, when functions are performed on the data the functions must ensure integrity. Examples of functions are transforming the data, storing the history, storing the definitions (Metadata) and storing the lineage of the data as it moves from one place to another. The most important aspect of data integrity per the data architecture discipline is to expose the data, the functions and the data's characteristics.

Data that has integrity is identically maintained during any operation (such as transfer, storage or retrieval). Put simply in business terms, data integrity is the assurance that data is consistent, certified and can be reconciled.

In terms of a [database](http://en.wikipedia.org/wiki/Database) data integrity refers to the process of ensuring that a database remains an accurate reflection of the universe of discourse it is modelling or representing. In other words there is a close correspondence between the facts stored in the database and the real world it models.

## Types of Integrity Constraints

Data integrity is normally enforced in a [database system](http://en.wikipedia.org/wiki/Database_system) by a series of integrity constraints or rules. Three types of integrity constraints are an inherent part of the relational data model: entity integrity, referential integrity and domain integrity.

[Entity integrity](http://en.wikipedia.org/wiki/Entity_integrity) concerns the concept of a [primary key](http://en.wikipedia.org/wiki/Primary_key). Entity integrity is an integrity rule which states that every table must have a primary key and that the column or columns chosen to be the primary key should be unique and not null.

[Referential integrity](http://en.wikipedia.org/wiki/Referential_integrity) concerns the concept of a [foreign key](http://en.wikipedia.org/wiki/Foreign_key). The referential integrity rule states that any foreign key value can only be in one of two states. The usual state of affairs is that the foreign key value refers to a primary key value of some table in the database. Occasionally, and this will depend on the rules of the business, a foreign key value can be null. In this case we are explicitly saying that either there is no relationship between the objects represented in the database or that this relationship is unknown.

[Domain integrity](http://en.wikipedia.org/w/index.php?title=Domain_integrity&action=edit&redlink=1) specifies that all columns in relational database must be declared upon a defined domain. The primary unit of data in the relational data model is the data item. Such data items are said to be non-decomposable or atomic. A domain is a set of values of the same type. Domains are therefore pools of values from which actual values appearing in the columns of a table are drawn.

If a database supports these features it is the responsibility of the database to insure data integrity as well as the [consistency model](http://en.wikipedia.org/wiki/Consistency_model) for the data storage and retrieval. If a database does not support these features it is the responsibility of the application to insure data integrity while the database supports the [consistency model](http://en.wikipedia.org/wiki/Consistency_model) for the data storage and retrieval.

Having a single, well controlled, and well defined data integrity system increases stability (one centralized system performs all data integrity operations), performance (all data integrity operations are performed in the same tier as the [consistency model](http://en.wikipedia.org/wiki/Consistency_model)), re-usability (all applications benefit from a single centralized data integrity system), and maintainability (one centralized system for all data integrity administration).

Today, since all modern [databases](http://en.wikipedia.org/wiki/Database) support these features (see [Comparison of relational database management systems](http://en.wikipedia.org/wiki/Comparison_of_relational_database_management_systems)), it has become the defacto responsibility of the database to insure data integrity. Out-dated and legacy systems that use file systems (text, spreadsheets, ISAM, flat files, etc.) for their [consistency model](http://en.wikipedia.org/wiki/Consistency_model) lack any kind of data integrity model. This requires companies to invest a large amount of time, money, and personnel in the creation of data integrity systems on a per application basis that effectively just duplicate the existing data integrity systems found in modern databases. Many companies, and indeed many database systems themselves, offer products and services to migrate out-dated and legacy systems to modern databases to provide these data integrity features. This offers companies a substantial savings in time, money, and resources because they do not have to develop per application data integrity systems that must be re-factored each time business requirements change.

**DATA FLOW DIAGRAM**

A Data Flow Diagram (DFD) is a diagram that describes the flow of data and the processes that change data throughout a system. It’s a structured analysis and design tool that can be used for flowcharting in place of or in association with information. Oriented and process oriented system flowcharts. When analysts prepare the Data Flow Diagram, they specify the user needs at a level of detail that virtually determines the information flow into and out of the system and the required data resources. This network is constructed by using a set of symbols that do not imply physical implementations. The Data Flow Diagram reviews the current physical system, prepares input and output specification, specifies the implementation plan etc.

Four basic symbols are used to construct data flow diagrams. They are symbols that represent data source, data flows, and data transformations and data storage. The points at which data are transformed are represented by enclosed figures, usually circles, which are called nodes.

**DATA FLOW DIAGRAM SYMBOLS:-**

* **Source or Destination of data**
* **Data Flow**
* **Process**
* **Storage**

**Steps to Construct Data Flow Diagrams**

Four steps are commonly used to construct a DFD

* Process should be named and numbered for easy reference. Each name should be representative of the process.
* The destination of flow is from top to bottom and from left to right.
* When a process is exploded in to lower level details they are numbered.
* The names of data stores, sources and destinations are written in capital letters.

**Rules for constructing a Data Flow Diagram**

* Arrows should not cross each other.
* Squares, circles and files must bear names.
* Decomposed data flow squares and circles can have same names.
* Draw all data flow around the outside of the diagram.

**Data Flow Diagram**

LOGIN

Website Administrator

User

Add/Edit Master Entries

Master Details

Order Details

Food Order

Food Delivery

Administrator

Username

Password

Item

Order

Add/Edit Subitem

Subitem

Item Details

Add/Edit

Item

Subitem Details

Company Banner

Add/Edit

Banner

Banner

Details

Deliver Order

Get Order

Order

Make Order

Get Order

**ER Diagram-**

M

1

M

1

Tbl\_orderdetails

M

1

Tbl\_subitemdetails

Tbl\_itemdetails

Tbl\_customerdetails

Tnl\_adminlogin

M

1

Tbl\_companybanner

Tbl\_companydetails

**TABLE SPECIFICATION**

Table Name: **tbl\_adminlogindetails**

|  |  |
| --- | --- |
| Field Name | Data Type |
| userid (PK) | **int** |
| username | **Varchar(50)** |
| password | **Varchar(50)** |
| userimage | **Varchar(50)** |

Table Name: **tbl\_companybanner**

|  |  |
| --- | --- |
| Field Name | Data Type |
| company\_bannerid (PK) | **int** |
| company\_banner | **Varchar(50)** |
| company\_id (FK) | **int** |

Table Name**: tbl\_companydetails**

|  |  |
| --- | --- |
| Field Name | Data Type |
| company\_id (PK) | **int** |
| company\_name | **Varchar(50)** |
| company\_address | **Varchar(50)** |
| company\_email | **Varchar(50)** |
| company\_mobile | **Varchar(50)** |
| company\_landline | **Varchar(50)** |
| company\_website | **Varchar(50)** |
| company\_type | **Varchar(50)** |
| company\_service | **Varchar(50)** |
| company\_logo | **Varchar(50)** |

Table Name**: tbl\_customerdetails**

|  |  |
| --- | --- |
| Field Name | Data Type |
| customerid (PK) | **int** |
| name | **Varchar(50)** |
| emailid | **Varchar(50)** |
| address | **Varchar(50)** |
| mobileno | **Varchar(50)** |
| area | **Varchar(50)** |
| landmark | **Varchar(50)** |
| customerstatus | **Varchar(50)** |
| orderdate | **date** |
| ordertime | **Varchar(50)** |
| delhiverydate | **date** |
| delhiverytime | **Varchar(50)** |
| city | **Varchar(50)** |

Table Name**: tbl\_itemdetails**

|  |  |
| --- | --- |
| Field Name | Data Type |
| itemid (PK) | **int** |
| itemname | **Varchar(50)** |
| itemtype | **Varchar(50)** |
| itemstatus | **Varchar(50)** |

Table Name**: tbl\_orderdetails**

|  |  |
| --- | --- |
| Field Name | Data Type |
| orderid (PK) | **int** |
| Customerid (FK) | **int** |
| Subitemid (FK) | **int** |
| quantity | **Varchar(50)** |
| price | **float** |
| totalprice | **float** |

Table Name**: tbl\_subitemdetails**

|  |  |
| --- | --- |
| Field Name | Data Type |
| subitemid (PK) | **int** |
| subitemname | **Varchar(50)** |
| itemprice | **float** |
| subitemimage | **Varchar(50)** |
| subitemstatus | **Varchar(50)** |
| itemid | **int** |

# User Interface Design

**User interface design** or **user interface engineering** is the design of [computers](http://en.wikipedia.org/wiki/Computer), [appliances](http://en.wikipedia.org/wiki/Home_appliance), machines, [mobile communication devices](http://en.wikipedia.org/wiki/Communication), [software](http://en.wikipedia.org/wiki/Software) applications, and [websites](http://en.wikipedia.org/wiki/Website) with the focus on the [user's experience](http://en.wikipedia.org/wiki/User_experience) and interaction. The goal of user interface design is to make the user's interaction as simple and efficient as possible, in terms of accomplishing user goals—what is often called [user-centered design](http://en.wikipedia.org/wiki/User-centered_design). Good user interface design facilitates finishing the task at hand without drawing unnecessary attention to itself. [Graphic design](http://en.wikipedia.org/wiki/Graphic_design) may be utilized to support its [usability](http://en.wikipedia.org/wiki/Usability). The design process must balance technical functionality and visual elements (e.g., [mental model](http://en.wikipedia.org/wiki/Mental_model)) to create a system that is not only operational but also usable and adaptable to changing user needs.

Interface design is involved in a wide range of projects from computer systems, to cars, to commercial planes; all of these projects involve much of the same basic human interactions yet also require some unique skills and knowledge. As a result, designers tend to specialize in certain types of projects and have skills centered around their expertise, whether that be [software design](http://en.wikipedia.org/wiki/Software_design), user research, [web design](http://en.wikipedia.org/wiki/Web_design), or [industrial design](http://en.wikipedia.org/wiki/Industrial_design).

## Processes

There are several phases and processes in the user interface design, some of which are more demanded upon than others, depending on the project. (Note: for the remainder of this section, the word system is used to denote any project whether it is a web site, application, or device.)

* Functionality requirements gathering – assembling a list of the functionality required by the system to accomplish the goals of the project and the potential needs of the users.
* User analysis – analysis of the potential users of the system either through discussion with people who work with the users and/or the potential users themselves. Typical questions involve:
  + What would the user want the system to do?
  + How would the system fit in with the user's normal [workflow](http://en.wikipedia.org/wiki/Workflow) or daily activities?
  + How technically savvy is the user and what similar systems does the user already use?
  + What interface look & feel styles appeal to the user?
* [Information architecture](http://en.wikipedia.org/wiki/Information_architecture) – development of the process and/or information flow of the system (i.e. for phone tree systems, this would be an option tree flowchart and for web sites this would be a site flow that shows the hierarchy of the pages).
* Prototyping – development of [wireframes](http://en.wikipedia.org/wiki/Website_wireframe), either in the form of [paper prototypes](http://en.wikipedia.org/wiki/Paper_prototypes) or simple interactive screens. These prototypes are stripped of all look & feel elements and most content in order to concentrate on the interface.
* [Usability testing](http://en.wikipedia.org/wiki/Usability_testing) – testing of the prototypes on an actual user—often using a technique called [think aloud protocol](http://en.wikipedia.org/wiki/Think_aloud_protocol) where you ask the user to talk about their thoughts during the experience.
* Graphic Interface design – actual [look & feel](http://en.wikipedia.org/wiki/Look_%26_feel) design of the final [graphical user interface](http://en.wikipedia.org/wiki/Graphical_user_interface) (GUI). It may be based on the findings developed during the usability testing if usability is unpredictable, or based on communication objectives and styles that would appeal to the user. In rare cases, the graphics may drive the prototyping, depending on the importance of visual form versus function. If the interface requires multiple [skins](http://en.wikipedia.org/wiki/Skin_%28computing%29), there may be multiple interface designs for one [control panel](http://en.wikipedia.org/wiki/Control_panel_%28engineering%29), functional feature or [widget](http://en.wikipedia.org/wiki/Widget_%28computing%29). This phase is often a collaborative effort between a [graphic designer](http://en.wikipedia.org/wiki/Graphic_design) and a user interface designer, or handled by one who is proficient in both disciplines.

User interface design requires a good understanding of user needs.

## Requirements

The dynamic characteristics of a system are described in terms of the dialogue requirements contained in seven principles of part 10 of the ergonomics standard, the [ISO 9241](http://en.wikipedia.org/wiki/ISO_9241). This standard establishes a framework of ergonomic "principles" for the dialogue techniques with high-level definitions and illustrative applications and examples of the principles. The principles of the dialogue represent the dynamic aspects of the interface and can be mostly regarded as the "feel" of the interface. The seven dialogue principles are:

* Suitability for the task: the dialogue is suitable for a task when it supports the user in the effective and efficient completion of the task.
* Self-descriptiveness: the dialogue is self-descriptive when each dialogue step is immediately comprehensible through feedback from the system or is explained to the user on request.
* Controllability: the dialogue is controllable when the user is able to initiate and control the direction and pace of the interaction until the point at which the goal has been met.
* Conformity with user expectations: the dialogue conforms with user expectations when it is consistent and corresponds to the user characteristics, such as task knowledge, education, experience, and to commonly accepted conventions.
* Error tolerance: the dialogue is error tolerant if despite evident errors in input, the intended result may be achieved with either no or minimal action by the user.
* Suitability for individualization: the dialogue is capable of individualization when the interface software can be modified to suit the task needs, individual preferences, and skills of the user.
* Suitability for learning: the dialogue is suitable for learning when it supports and guides the user in learning to use the system.

The concept of usability is defined in Part 11 of the [ISO 9241](http://en.wikipedia.org/wiki/ISO_9241) standard by effectiveness, efficiency, and satisfaction of the user. Part 11 gives the following definition of usability:

* Usability is measured by the extent to which the intended goals of use of the overall system are achieved (effectiveness).
* The resources that have to be expended to achieve the intended goals (efficiency).
* The extent to which the user finds the overall system acceptable (satisfaction).

Effectiveness, efficiency, and satisfaction can be seen as quality factors of usability. To evaluate these factors, they need to be decomposed into sub-factors, and finally, into usability measures.

The information presentation is described in Part 12 of the [ISO 9241](http://en.wikipedia.org/wiki/ISO_9241) standard for the organization of information (arrangement, alignment, grouping, labels, location), for the display of graphical objects, and for the coding of information (abbreviation, color, size, shape, visual cues) by seven attributes. The "attributes of presented information" represent the static aspects of the interface and can be generally regarded as the "look" of the interface. The attributes are detailed in the recommendations given in the standard. Each of the recommendations supports one or more of the seven attributes. The seven presentation attributes are:

* Clarity: the information content is conveyed quickly and accurately.
* Discriminability: the displayed information can be distinguished accurately.
* Conciseness: users are not overloaded with extraneous information.
* Consistency: a unique design, conformity with user’s expectation.
* Detectability: the user’s attention is directed towards information required.
* Legibility: information is easy to read.
* Comprehensibility: the meaning is clearly understandable, unambiguous, interpretable, and recognizable.

The user guidance in Part 13 of the [ISO 9241](http://en.wikipedia.org/wiki/ISO_9241) standard describes that the user guidance information should be readily distinguishable from other displayed information and should be specific for the current context of use. User guidance can be given by the following five means:

* Prompts indicating explicitly (specific prompts) or implicitly (generic prompts) that the system is available for input.
* Feedback informing about the user’s input timely, perceptible, and non-intrusive.
* Status information indicating the continuing state of the application, the system’s hardware and software components, and the user’s activities.
* Error management including error prevention, error correction, user support for error management, and error messages.
* On-line help for system-initiated and user initiated requests with specific information for the current context of use.

Coding

Coding is the phase of Software Life Cycle that produces the Actual code that will be deliveredto the customer as the running System. The other phases of the life cycle may also developcode, such as prototypes, tests, and test drivers, but these are for the use by the developer.Individual modules developed in this system are also tested before being delivered to the nextphase.The design must be translated into a machine- readable form. This is what coding. Coding isbasically translating the design into a machine-readable form. The code generation stepperforms this task. If design is performed in a detailed manner, code generation can beaccomplished mechanistically.While converting design into coding, following points are to be considered:

* Is a Code a proper translation of design?
* Have the Coding standards been followed?
* Are the data types and data declarations proper?
* Are the physical constants correct?

**CODE EFFICIENCY**

The next thing, which we have to consider, is the “Code Efficiency”. The code that we write should be well efficient and error free. Because as much our code is efficient and error free,out software will be that much effective. The efficiency of the code or program depends on :

**Readability:**

Coding should be much and more readable as in 4GL such as Coding in php,mysql etc.

**Portability:**

Coding or program should be written in such a form such that it could be run on differentmachines with little or no change. Thus, out coding should be as much as portable as it can.

**Easy Debugging:**

Coding should be such that errors could easily be removed or debugged. That is our codingshould be as much as error free as possible.

**Easy Software Development:**

Software could easily be developed. Commands of programming language are similar tonatural languages (English).

**Short & Precise Coding:**

Coding should not be lengthy. It should be short & precise. This factor is most important for theefficient coding.

**Optimization of Code:**

Code Optimization is the last & final step in Code generation phase. Code optimization comesafter the code generation. We have to optimize the code after generating it.Code optimization also plays a vital role as the efficiency of code plays in the successfuldevelopment of the system. Code Optimization also effects the efficiency of code. Our codewill be more efficient if it is more optimized.

To produce faster and more compact, the code generator should include some form of “CodeOptimization”. This may exploit techniques such as the use of special purpose machine

instructions or addressing modes, register optimization, etc. this code optimization mayincorporate both machine dependent and machine independent techniques.

**VALIDATION**

Validation refers to the process of using software in a live environment to find errors.Validation refers to the set of activities that ensure that the software that has been built istraceable to customer requirements. In validation testing, requirements are established as partof software requirement analysis are validated against the software that has been constructed.The feedback from the validation phase generally brings some changes in the software to dealwith errors and failures that are uncovered. Then a set of user sites is selected for putting the system into use on a live basis. These beta test sites use the system in day-to-day activities;they process live transactions and produce normal system output.

Validation Checks maycontinue for several months depending on the requirements of the end-users for whom theSoftware is developed. During the course of validating the system, failure may occur and thesoftware will be changed. Continue use may bring more failures and the need for still morechanges.

In the last stage of software development, software is completely assembled as apackage, some interfacing errors have been uncovered and corrected, and a final series of software tests – validation testing or validation checks may begin. Validation succeeds whenthe software functions in a manner that can be reasonably expected by the customer.

Validation Criteria is the most important and ironically, the most often neglectedsection of Software requirements Specification.

Software validations are achieved through a series of black-box tests that demonstrateconformity with requirements. A test plan outlines the classes of tests to be conducted and atest procedure define test cases that will be used to demonstrate conformity withrequirements. Both the plan and procedure are designed to ensure that all performancerequirements are attained, documentation is correct, and human-engineered and otherrequirements are met (e.g., transportability, compatibility, error recovery, maintainability).

In computing, the process of checking input data to ensure that it is complete, accurate, andreasonable. Although it would be impossible to guarantee that only valid data are entered into a computer, a suitable combination of validation checks should ensure that most errors are detected.

**Validation Checks in the Project**

There is a package in the project, validate\_data which contains various functions for validationon different field. Some function used for validation check in the system are-

**boolean isNotAlphabets(String)-**

This function checks that given string containalphabets only or not. For example in case of Salesman name field, it should containalphabets only not numeric or special character.

**boolean isNumber(String)-**

This function return true if passed string contain numberonly otherwise false. In case of phone no, mobile number etc validation is done withthis method.

**boolean isEmail(String)-**

This function checks validity of email entered in email field.

**boolean isDate(String)-**

This function checks validity of date entered. If date greaterthan current date or format of date is not correct or date supplied does not exist than itreturns false otherwise true.

# ERROR HANDLING

**Error handling** is the process of responding to the occurrence of exceptional conditions during computation, conditions requiring special processing, often changing the normal flow of [program](http://en.wikipedia.org/wiki/Computer_program) [execution](http://en.wikipedia.org/wiki/Execution_%28computing%29). It is provided by specialized [programming language](http://en.wikipedia.org/wiki/Programming_language) constructs or [computer hardware](http://en.wikipedia.org/wiki/Computer_hardware) mechanisms.

In general, an exception is handled (resolved) by saving the current state of execution in a predefined place and switching the execution to a specific [subroutine](http://en.wikipedia.org/wiki/Subroutine) known as an exception handler. If exceptions are continuable, the handler may later resume the execution at the original location using the saved information. For example, a [floating point](http://en.wikipedia.org/wiki/Floating_point) [divide by zero](http://en.wikipedia.org/wiki/Division_by_zero) exception will typically, by default, allow the program to be resumed, while an [out of memory](http://en.wikipedia.org/wiki/Out_of_memory) condition might not be resolvable transparently.

Alternative approaches to exception handling in software are [error checking](http://en.wikipedia.org/wiki/Error_checking), which maintains normal program flow with later explicit checks for contingencies, using returned values or some auxiliary global variable such as C's [errno](http://en.wikipedia.org/wiki/Errno); or input validation.

### Verification of Error Handling

The point of exception handling routines is to ensure that the code can handle error conditions. In order to establish that exception handling routines are sufficiently robust, it is necessary to present the code with a wide spectrum of invalid or unexpected inputs, such as can be created via software [fault injection](http://en.wikipedia.org/wiki/Fault_injection) and [mutation testing](http://en.wikipedia.org/wiki/Mutation_testing) (which is also sometimes referred to as [fuzz testing](http://en.wikipedia.org/wiki/Fuzz_testing)). One of the most difficult types of software for which to write exception handling routines is protocol software, since a robust protocol implementation must be prepared to receive input that does not comply with the relevant specification(s).

In order to ensure that meaningful regression analysis can be conducted throughout a [software development lifecycle process](http://en.wikipedia.org/wiki/Software_development_process), any exception handling verification should be highly automated, and the test cases must be generated in a scientific, repeatable fashion. Several commercially available systems exist that perform such testing.

In runtime engine environments such as [Java](http://en.wikipedia.org/wiki/Java_%28programming_language%29) or [.NET](http://en.wikipedia.org/wiki/.NET_Framework), there exist tools that attach to the runtime engine and every time that an exception of interest occurs, they record debugging information that existed in memory at the time the exception was thrown ([call stack](http://en.wikipedia.org/wiki/Call_stack) and [heap](http://en.wikipedia.org/wiki/Heap_%28data_structure%29) values). These tools are called [automated exception handling](http://en.wikipedia.org/wiki/Automated_exception_handling) or error interception tools and provide 'root-cause' information for exceptions.

**PARAMETERS CALLING/PASSING**

In [computer programming](http://en.wikipedia.org/wiki/Computer_programming), a **parameter** is a special kind of [variable](http://en.wikipedia.org/wiki/Variable_%28programming%29), used in a [subroutine](http://en.wikipedia.org/wiki/Subroutine) to refer to one of the pieces of data provided as input to the subroutine.[[1]](http://en.wikipedia.org/wiki/Parameter_%28computer_programming%29#cite_note-0) These pieces of data are called **arguments**. An ordered list of parameters is usually included in the definition of a subroutine, so that, each time the subroutine is called, its arguments for that call can be assigned to the corresponding parameters.

Just as in standard mathematical usage, the argument is thus the actual value passed to a function, procedure, or routine (such as 37 in log(37)), whereas the parameter is a reference to that value inside the implementation of the function (log in this case). See the [Parameters and arguments](http://en.wikipedia.org/wiki/Parameter_%28computer_programming%29#Parameters_and_arguments) section for more information.

In the most common case, [call-by-value](http://en.wikipedia.org/wiki/Call-by-value), a parameter acts within the subroutine as a [local](http://en.wikipedia.org/wiki/Local_variable) (isolated) copy of the argument, but in other cases, e.g. [call-by-reference](http://en.wikipedia.org/wiki/Call-by-reference), the argument supplied by the caller can be affected by actions within the called subroutine (as discussed in [evaluation strategy](http://en.wikipedia.org/wiki/Evaluation_strategy)).

The semantics for how parameters can be declared and how the arguments get passed to the parameters of subroutines are defined by the language, but the details of how this is represented in any particular computer system depend on the [calling conventions](http://en.wikipedia.org/wiki/Calling_convention) of that system.

## Parameters and arguments

These two terms are sometimes loosely used interchangeably; in particular, "argument" is sometimes used in place of "parameter". Nevertheless, there is a difference. Properly, parameters appear in procedure definitions; arguments appear in procedure calls.

A parameter is an intrinsic property of the procedure, included in its definition. For example, in many languages, a minimal procedure to add two supplied integers together and calculate the sum total would need two parameters, one for each expected integer. In general, a procedure may be defined with any number of parameters, or no parameters at all. If a procedure has parameters, the part of its definition that specifies the parameters is called its parameter list.

By contrast, the arguments are the values actually supplied to the procedure when it is called. Unlike the parameters, which form an unchanging part of the procedure's definition, the arguments can, and often do, vary from call to call. Each time a procedure is called, the part of the procedure call that specifies the arguments is called the argument list.

**VALIDATION CHECKS**

In [computer science](http://en.wikipedia.org/wiki/Computer_science), **data validation** is the process of ensuring that a program operates on clean, correct and useful data. It uses routines, often called "[validation rules](http://en.wikipedia.org/wiki/Validation_rule)" or "check routines", that check for correctness, meaningfulness, and security of data that are input to the system. The rules may be implemented through the automated facilities of a [data dictionary](http://en.wikipedia.org/wiki/Data_dictionary), or by the inclusion of explicit [application program](http://en.wikipedia.org/wiki/Application_program) validation logic.

For business applications, data validation can be defined through [declarative](http://en.wikipedia.org/wiki/Declarative_programming) [data integrity](http://en.wikipedia.org/wiki/Data_integrity) rules, or [procedure-based](http://en.wikipedia.org/wiki/Imperative_programming) [business rules](http://en.wikipedia.org/wiki/Business_rules).[[1]](http://en.wikipedia.org/wiki/Data_validation#cite_note-0) Data that does not conform to these rules will negatively affect business process execution. Therefore, data validation should start with business process definition and set of business rules within this process. Rules can be collected through the requirements capture exercise.[[2]](http://en.wikipedia.org/wiki/Data_validation#cite_note-1)

The simplest data validation verifies that the characters provided come from a valid set. For example, telephone numbers should include the [digits](http://en.wikipedia.org/wiki/Numerical_digit) and possibly the characters +, -, (, and ) (plus, minus, and brackets). A more sophisticated data validation routine would check to see the user had entered a valid country code, i.e., that the number of digits entered matched the convention for the country or area specified.

Incorrect data validation can lead to [data corruption](http://en.wikipedia.org/wiki/Data_corruption) or a [security vulnerability](http://en.wikipedia.org/wiki/Software_security_vulnerability). Data validation checks that data are valid, sensible, reasonable, and secure before they are processed.

Testing

Testing is a process to show the correctness of the program. Testing is needed to show completeness, to improve the quality of the software and to provide the maintenance aid. Some testing standards are therefore necessary reduce the testing costs and operation time. Testing software extends throughout the coding phase and it represents the ultimate review of configurations, design and coding. Based on the way the software reacts to these testing, we can decide whether the configuration that has been built is study or not. All components of an application are tested, as the failure to do so many results in a series of bugs after the software is put to use.

**Software testing** is an investigation conducted to provide stakeholders with information about the quality of the product or service under test.[[1]](http://en.wikipedia.org/wiki/Software_testing#cite_note-0) Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include, but are not limited to, the process of executing a program or application with the intent of finding [software bugs](http://en.wikipedia.org/wiki/Software_bug) (errors or other defects).

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

1. meets the requirements that guided its design and development;
2. works as expected; and
3. can be implemented with the same characteristics.

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

Different software development models will focus the test effort at different points in the development process. Newer development models, such as [Agile](http://en.wikipedia.org/wiki/Agile_software_development), often employ [test driven development](http://en.wikipedia.org/wiki/Test_driven_development) and place an increased portion of the testing in the hands of the developer, before it reaches a formal team of testers. In a more traditional model, most of the test execution occurs after the requirements have been defined and the coding process has been completed.

## Overview

Testing can never completely identify all the defects within software.[[2]](http://en.wikipedia.org/wiki/Software_testing#cite_note-1) Instead, it furnishes a criticism or comparison that compares the state and behavior of the product against [oracles](http://en.wikipedia.org/wiki/Oracle_%28software_testing%29)—principles or mechanisms by which someone might recognize a problem. These oracles may include (but are not limited to) specifications, [contracts](http://en.wikipedia.org/wiki/Design_by_Contract),[[3]](http://en.wikipedia.org/wiki/Software_testing#cite_note-2) comparable products, past versions of the same product, inferences about intended or expected purpose, user or customer expectations, relevant standards, applicable laws, or other criteria.

Every software product has a target audience. For example, the audience for video game software is completely different from banking software. Therefore, when an organization develops or otherwise invests in a software product, it can assess whether the software product will be acceptable to its end users, its target audience, its purchasers, and other stakeholders. **Software testing** is the process of attempting to make this assessment.

A study conducted by [NIST](http://en.wikipedia.org/wiki/NIST) in 2002 reports that software bugs cost the U.S. economy $59.5 billion annually. More than a third of this cost could be avoided if better software testing was performed.

### Scope

A primary purpose of testing is to detect software failures so that defects may be discovered and corrected. Testing cannot establish that a product functions properly under all conditions but can only establish that it does not function properly under specific conditions. The scope of software testing often includes examination of code as well as execution of that code in various environments and conditions as well as examining the aspects of code: does it do what it is supposed to do and do what it needs to do. In the current culture of software development, a testing organization may be separate from the development team. There are various roles for testing team members. Information derived from software testing may be used to correct the process by which software is developed.

### Functional vs non-functional testing

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

Non-functional testing refers to aspects of the software that may not be related to a specific function or user action, such as [scalability](http://en.wikipedia.org/wiki/Scalability) or other [performance](http://en.wikipedia.org/wiki/Performance), behavior under certain [constraints](http://en.wikipedia.org/wiki/Constraints), or [security](http://en.wikipedia.org/wiki/Computer_security). Non-functional requirements tend to be those that reflect the quality of the product, particularly in the context of the suitability perspective of its users.

### Defects and failures

Not all software defects are caused by coding errors. One common source of expensive defects is caused by requirement gaps, e.g., unrecognized requirements that result in errors of omission by the program designer.[[15]](http://en.wikipedia.org/wiki/Software_testing#cite_note-14) A common source of requirements gaps is [non-functional requirements](http://en.wikipedia.org/wiki/Non-functional_requirements) such as [testability](http://en.wikipedia.org/wiki/Software_testability), [scalability](http://en.wikipedia.org/wiki/Scalability), [maintainability](http://en.wikipedia.org/wiki/Maintainability), [usability](http://en.wikipedia.org/wiki/Usability), [performance](http://en.wikipedia.org/wiki/Computer_performance), and [security](http://en.wikipedia.org/wiki/Computer_security).

Software faults occur through the following processes. A programmer makes an [error](http://en.wikipedia.org/wiki/Human_error) (mistake), which results in a [defect](http://en.wikipedia.org/wiki/Fault_%28technology%29) (fault, bug) in the software [source code](http://en.wikipedia.org/wiki/Source_code). If this defect is executed, in certain situations the system will produce wrong results, causing a [failure](http://en.wikipedia.org/wiki/Failure).[[16]](http://en.wikipedia.org/wiki/Software_testing#cite_note-ctfl-15) Not all defects will necessarily result in failures. For example, defects in [dead code](http://en.wikipedia.org/wiki/Dead_code) will never result in failures. A defect can turn into a failure when the environment is changed. Examples of these changes in environment include the software being run on a new [computer hardware](http://en.wikipedia.org/wiki/Computer_hardware) platform, alterations in [source data](http://en.wikipedia.org/wiki/Source_data) or interacting with different software.[[16]](http://en.wikipedia.org/wiki/Software_testing#cite_note-ctfl-15) A single defect may result in a wide range of failure symptoms.

### Finding faults early

It is commonly believed that the earlier a defect is found the cheaper it is to fix it. The following table shows the cost of fixing the defect depending on the stage it was found. For example, if a problem in the requirements is found only post-release, then it would cost 10–100 times more to fix than if it had already been found by the requirements review. Modern continuous deployment practices, and cloud-based services may cost less for re-deployment and maintenance than in the past.

### Compatibility Testing

A common cause of software failure (real or perceived) is a lack of its [compatibility](http://en.wikipedia.org/wiki/Computer_compatibility) with other [application software](http://en.wikipedia.org/wiki/Application_software), [operating systems](http://en.wikipedia.org/wiki/Operating_system) (or operating system [versions](http://en.wikipedia.org/wiki/Software_versioning), old or new), or target environments that differ greatly from the original (such as a [terminal](http://en.wikipedia.org/wiki/Computer_terminal) or [GUI](http://en.wikipedia.org/wiki/GUI) application intended to be run on the [desktop](http://en.wikipedia.org/wiki/Desktop_metaphor) now being required to become a [web application](http://en.wikipedia.org/wiki/Web_application), which must render in a [web browser](http://en.wikipedia.org/wiki/Web_browser)). For example, in the case of a lack of [backward compatibility](http://en.wikipedia.org/wiki/Backward_compatibility), this can occur because the programmers develop and test software only on the latest version of the target environment, which not all users may be running. This results in the unintended consequence that the latest work may not function on earlier versions of the target environment, or on older hardware that earlier versions of the target environment was capable of using. Sometimes such issues can be fixed by proactively [abstracting](http://en.wikipedia.org/wiki/Abstraction_%28computer_science%29) operating system functionality into a separate program [module](http://en.wikipedia.org/wiki/Modular_programming) or [library](http://en.wikipedia.org/wiki/Library_%28computing%29).

### Input Combinations and Preconditions

A very fundamental problem with software testing is that testing under *all* combinations of inputs and preconditions (initial state) is not feasible, even with a simple product.[[13]](http://en.wikipedia.org/wiki/Software_testing#cite_note-Kaner1-12)[[18]](http://en.wikipedia.org/wiki/Software_testing#cite_note-17) This means that the number of [defects](http://en.wikipedia.org/wiki/Software_bug) in a software product can be very large and defects that occur infrequently are difficult to find in testing. More significantly, [non-functional](http://en.wikipedia.org/wiki/Non-functional_requirements) dimensions of quality (how it is supposed to *be* versus what it is supposed to *do*)—[usability](http://en.wikipedia.org/wiki/Usability), [scalability](http://en.wikipedia.org/wiki/Scalability), [performance](http://en.wikipedia.org/wiki/Computer_performance), [compatibility](http://en.wikipedia.org/wiki/Backward_compatibility), [reliability](http://en.wikipedia.org/wiki/Reliability_%28engineering%29)—can be highly subjective; something that constitutes sufficient value to one person may be intolerable to another.

### Static vs. Dynamic Testing

There are many approaches to software testing. [Reviews](http://en.wikipedia.org/wiki/Code_review), [walkthroughs](http://en.wikipedia.org/wiki/Software_walkthrough), or [inspections](http://en.wikipedia.org/wiki/Software_inspection) are considered as [static testing](http://en.wikipedia.org/wiki/Static_testing), whereas actually executing programmed code with a given set of [test cases](http://en.wikipedia.org/wiki/Test_case) is referred to as [dynamic testing](http://en.wikipedia.org/wiki/Dynamic_testing). Static testing can be (and unfortunately in practice often is) omitted. Dynamic testing takes place when the program itself is used for the first time (which is generally considered the beginning of the testing stage). Dynamic testing may begin before the program is 100% complete in order to test particular sections of code (modules or discrete [functions](http://en.wikipedia.org/wiki/Function_%28computer_science%29)). Typical techniques for this are either using [stubs](http://en.wikipedia.org/wiki/Method_stub)/drivers or execution from a [debugger](http://en.wikipedia.org/wiki/Debugger) environment. For example, [spreadsheet](http://en.wikipedia.org/wiki/Spreadsheet) programs are, by their very nature, tested to a large extent interactively ("[on the fly](http://en.wikipedia.org/wiki/On_the_fly)"), with results displayed immediately after each calculation or text manipulation.

### Software Verification and Validation

Software testing is used in association with [verification and validation](http://en.wikipedia.org/wiki/Verification_and_Validation_%28software%29):[[19]](http://en.wikipedia.org/wiki/Software_testing" \l "cite_note-tran-18)

* Verification: Have we built the software right? (i.e., does it match the specification).
* Validation: Have we built the right software? (i.e., is this what the customer wants).

The terms verification and validation are commonly used interchangeably in the industry; it is also common to see these two terms incorrectly defined. According to the IEEE Standard Glossary of Software Engineering Terminology:

Verification is the process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.

Validation is the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements.

According to the IS0 9000 standard:

Verification is confirmation by examination and through provision of objective evidence that specified requirements have been fulfilled.

Validation is confirmation by examination and through provision of objective evidence that the requirements for a specific intended use or application have been fulfilled.

### Software Testing Team

Software testing can be done by software testers. Until the 1980s the term "software tester" was used generally, but later it was also seen as a separate profession. Regarding the periods and the different goals in software testing,[[20]](http://en.wikipedia.org/wiki/Software_testing" \l "cite_note-19) different roles have been established: manager, test lead, test designer, tester, automation developer, and test administrator.

### Software Quality Assurance (SQA)

Though controversial, software testing is a part of the [software quality assurance](http://en.wikipedia.org/wiki/Software_quality_assurance) (SQA) process.[[13]](http://en.wikipedia.org/wiki/Software_testing#cite_note-Kaner1-12) In SQA, software process specialists and auditors are concerned for the software development process rather than just the artifacts such as documentation, code and systems. They examine and change the software engineering process itself to reduce the number of faults that end up in the delivered software: the so-called *defect rate.*

What constitutes an "acceptable defect rate" depends on the nature of the software; A flight simulator video game would have much higher defect tolerance than software for an actual airplane.

Although there are close links with SQA, testing departments often exist independently, and there may be no SQA function in some companies.

Software testing is a task intended to detect defects in software by contrasting a computer program's expected results with its actual results for a given set of inputs. By contrast, QA (quality assurance) is the implementation of policies and procedures intended to prevent defects from occurring in the first place.

**Objectives of Testing:**

This section introduces the concept of testing and how important is, for the successful implementation of the project. Different phases of testing are described along with the level of testing incorporated in this particular project.

Testing is vital to the success of any system. Testing is done at different stages within the phase. System testing makes a logical assumption that if all phases of the system are correct, the goals will be achieved successfully. Inadequate testing at all leads to errors that may come up after a long time when correction would be extremely difficult. Another objective of testing is its utility as a user-oriented vehicle before implementation. The testing of the system was done on both artificial and live data.

Testing involves operation of a system or application under controlled conditions and evaluating the results (e.g., “if the user is in interface A of the application while using hardware B and does C, then D should not happen”). The controlled conditions should include both normal and abnormal conditions.

Typically, the project team includes a mix of testers and developers who work closely together, with the overall QA processes being monitored by the project managers.

**Types of Testing**

**Black Box Testing**

Black box testing also called behavioral testing, focuses on the functional requirements of software. This testing approach enables the software engineer to derive the input conditions that will fully exercise all requirements for a program.

Also known as functional testing, this is a software testing technique whereby the tester does not know the internal working of the item being tested. Black-box test design treats the system as a “black-box”, so it does not

explicitly use knowledge of the internal structure. Black-box test design is usually described as focusing on testing functional requirements. Synonyms for black-box includes: behavioral, functional, opaque-box and closed-box.

Black box testing attempts to find the errors like

* Incorrect or missing functions
* Interface errors
* Errors in data structures or external database access
* Behavior or performance errors
* Initialization and termination errors

In Black box testing software is exercised over a full range of inputs and outputs are observed for correctness.

**White Box Testing**

White box test design allows one to peek inside the “box”, and it focuses specifically on using internal knowledge of the software to guide the selection of test data. Synonyms for white-box include: structural, glass-box and clear-box.

Testing involves

* Unit testing
* Integration testing
* Acceptance testing

The first level of test is unit testing. The purpose of unit testing is to ensure that each program is fully tested.

The second step is integration testing. In this individual program units or programs are integrated and tested as a complete system to ensure that the software requirements are met.

Acceptance Testing involves planning and the execution of various types of tests in order to demonstrate that the implemented software system satisfies the requirements. Finally our project meets the requirements after going through all the levels of testing.

**Condition Testing**

An improvement over White-box testing, the process of condition testing ensures that a controlling expression has been adequately exercised whist the software is under test by constructing a *constraint set* for every expression and then ensuring that every member on the constraint set is included in the values whish are presented to the expression

**Data Life-Cycle Testing**

It is based upon the consideration that in the software code, a variable is at some stage created, and subsequently may have its value changed or used in a controlling expression several times before being destroyed. If only locally declared Boolean used in control conditions are considered then an examination of the sources code will indicate the place in the source code where the variable is created, places where it is given a value is used as a part of a control expression and the place where it is destroyed.

This approach to testing requires all possible feasible lifecycles of the variable to be covered whilst the module is under test.

**Unit Testing**

The purpose of this phase is to test the individual units of the developing software component. This phase is recursive and is to be repeated, as many as there are, levels of testing. In the DGLW project, each individual form has been tested using techniques of testing namely: Client side testing using JavaScript.

Each individual form has been validated so that user enters only valid data at every time.

**Functional Testing:**

This is done for each module / sub module of the system. Functional testing serve as a means of validating whether the functionality of the system Confers the original user requirement i.e. does the module do what it was supposed to do? Separate schedules were made for functional testing. It involves preparation of the test data, writing of test cases, testing for conformance to test cases and preparation of bugs listing for non-conformities.

**System Testing:**

System testing is done when the entire system has been fully integrated. The purpose of the system testing is to test how the different modules interact with each other and whether the entire system provides the functionality that was expected.

System testing consists of the following steps:

1. Program Testing
2. String Testing
3. System Testing
4. System Documentation
5. User Acceptance Testing

**Various Levels Of Testing**

Before implementation the system is tested at two levels:

Level 1

Level 2

**Level 1 Testing (Alpha Testing)**

At this level a test data is prepared for testing. Project leaders test the system on this test data keeping the following points into consideration:

* Proper error handling
* Exit Pints in code
* Exception handling
* Input / Output format
* Glass box testing
* Black box testing

If the system is through with testing phase at LEVEL 1 then it is passed on to LEVEL 2.

**Level 2 Testing (Beta Testing)**

Here the testing is done on the live database. If errors are detected then it is sent back to LEVEL 1 for modification otherwise it is passed on to LEVEL 3.

This is the level at which the system actually becomes live and implemented for the use of END USERS.

**Recovery & Security**

A forced system failure is induced to test a backup recovery procedure for file integrity. Inaccurate data are entered to see how the system responds in terms of error detection and protection. Related to file integrity is a test to demonstrate that data and programs are secure from unauthorized access.

**Usability Documentation & Procedure:**

The usability test verifies the user-friendly nature of the system. This relates to normal operating and error-handling procedures.

**Quality Assurance**

Proper documentation is must for mainframe of any software. Apart from In-line documentation while coding. Help coding, help files corresponding to each program were prepared so as to tackle the person-dependency of the existing system.

**System Implementation**

During the implementation stage the system is physically created. Necessary programs are coded, debugged and documented. A new hardware is selected, ordered and installed.

**System Specification**

Every computer system consists of three major elements.

1. The Hardware

2. Application Software such as visual studio.

3. Operating system

For successful operation of the package following must be kept in mind:

Too many packages should not be used, as very few systems may have all those packages installed due to memory problem. Thus, the compatibility of the system developed will get reduced.

**Hardware Requirements**

Intel Pentium processor at 500 MHz or faster, minimum of 364 MB available disk space for installation (including IBM SDK), minimum of 256 MB memory,512 MB recommended, CD-ROM drive.

**Installation**

The Application installation scripts have to be generated from the current server where the application source code is saved and installed in the main server from where the application is to be run. This was done using a special code, which generates all SQL-Statements to insert preliminary data (like menu entries, code in code directories etc) at server and the operational modules of the application made available to the end users successfully.

**Implementation**

The system is still under construction few reports are yet to me made after that this system will be implanted at client side. Users will be given a training to use the package and special workshops are conducted by the NIC for the purpose. And according to their feedback the changes are implanted in the software.

**White box Testing**

White box testing is also called Glassbox testing is a test case design control; structure of the procedural design to derive test cases using Whitebox testing method, the software engineer can derive the test cases that guarantee that all independent paths within the module have been exercised at least once.

Exercise all logic decisions on their true or false sides. Execute all loops at their boundaries and within their operational bounds. Exercise internal data structure to ensure their validity.

**Security**

The system security problem can be divided into four related issues: security, integrity, privacy and confidentiality. They determine the file structure, data structure and access procedures.

System security refers to the technical innovations and procedures applied to the hardware and operating systems to protect against deliberate or accidental damage from a defined threat. In contrast, data security is the protection of data from loss, disclosure, modifications and destruction.

System integrity refers to the proper functioning of programs, appropriate physical security and safety against external threats such as eavesdropping and wiretapping. In comparison, data integrity makes sure that do not differ from original from others and how the organization can be protected against unwelcome, unfair or excessive dissemination of information about it.

The term confidentiality is a special status given to sensitive information in a data base to minimize the possible invasion of privacy. It is an attribute of information that characterizes its need for protection. System security is the technical means of providing such protection. In contrast privacy is largely a procedural matter of how information is used.

**Security** is the degree of protection against danger, damage, loss, and [crime](http://en.wikipedia.org/wiki/Crime). Security as a form of protection is structures and processes that provide or improve security as a condition. The Institute for Security and Open Methodologies ([ISECOM](http://en.wikipedia.org/wiki/ISECOM)) in the [OSSTMM](http://en.wikipedia.org/w/index.php?title=OSSTMM&action=edit&redlink=1) 3 defines security as "a form of protection where a separation is created between the assets and the threat". This includes but is not limited to the elimination of either the asset or the threat. Security as a national condition was defined in a United Nations study (1986), so that countries can develop and progress safely.

Security has to be compared to related concepts: [safety](http://en.wikipedia.org/wiki/Safety), [continuity](http://en.wiktionary.org/wiki/continuity), [reliability](http://en.wiktionary.org/wiki/reliability). The key difference between security and reliability is that security must take into account the actions of people attempting to cause destruction.

Different scenarios also give rise to the context in which security is maintained:

* With respect to classified matter, the condition that prevents unauthorized persons from having [access](http://en.wikipedia.org/wiki/Access_control) to official [information](http://en.wikipedia.org/wiki/Information) that is safeguarded in the interests of [national security](http://en.wikipedia.org/wiki/National_security).
* Measures taken by a military unit, an activity or installation to protect itself against all acts designed to, or which may, impair its effectiveness.

## Security by design

The technologies of computer security are based on [logic](http://en.wikipedia.org/wiki/Logic). As security is not necessarily the primary goal of most computer applications, designing a program with security in mind often imposes restrictions on that program's behavior.

There are 4 approaches to [security](http://en.wikipedia.org/wiki/Security) in [computing](http://en.wikipedia.org/wiki/Computing), sometimes a combination of approaches is valid:

1. Trust all the software to abide by a security policy but the software is not trustworthy (this is [computer insecurity](http://en.wikipedia.org/wiki/Computer_insecurity)).
2. Trust all the software to abide by a security policy and the software is validated as trustworthy (by tedious branch and path analysis for example).
3. Trust no software but enforce a security policy with [mechanisms](http://en.wikipedia.org/wiki/Protection_mechanism) that are not trustworthy (again this is [computer insecurity](http://en.wikipedia.org/wiki/Computer_insecurity)).
4. Trust no software but enforce a security policy with trustworthy hardware mechanisms.

Many systems have unintentionally resulted in the first possibility. Since approach two is expensive and non-deterministic, its use is very limited. Approaches one and three lead to failure. Because approach number four is often based on hardware mechanisms and avoids abstractions and a multiplicity of degrees of freedom, it is more practical. Combinations of approaches two and four are often used in a layered architecture with thin layers of two and thick layers of four.

There are various strategies and techniques used to design security systems. However, there are few, if any, effective strategies to enhance security after design. One technique enforces the [principle of least privilege](http://en.wikipedia.org/wiki/Principle_of_least_privilege) to great extent, where an entity has only the privileges that are needed for its function. That way even if an [attacker](http://en.wikipedia.org/wiki/Attacker) gains access to one part of the system, fine-grained security ensures that it is just as difficult for them to access the rest.

Furthermore, by breaking the system up into smaller components, the complexity of individual components is reduced, opening up the possibility of using techniques such as [automated theorem proving](http://en.wikipedia.org/wiki/Automated_theorem_proving) to prove the correctness of crucial software subsystems. This enables a [closed form solution](http://en.wikipedia.org/wiki/Closed_form_solution) to security that works well when only a single well-characterized property can be isolated as critical, and that property is also assessable to math. Not surprisingly, it is impractical for generalized correctness, which probably cannot even be defined, much less proven. Where formal correctness proofs are not possible, rigorous use of [code review](http://en.wikipedia.org/wiki/Code_review) and [unit testing](http://en.wikipedia.org/wiki/Unit_testing) represent a best-effort approach to make modules secure.

The design should use "[defense in depth](http://en.wikipedia.org/wiki/Defense_in_depth_%28computing%29)", where more than one subsystem needs to be violated to compromise the integrity of the system and the information it holds. Defense in depth works when the breaching of one security measure does not provide a platform to facilitate subverting another. Also, the cascading principle acknowledges that several low hurdles does not make a high hurdle. So cascading several weak mechanisms does not provide the safety of a single stronger mechanism.

Subsystems should default to secure settings, and wherever possible should be designed to "fail secure" rather than "fail insecure" (see [fail-safe](http://en.wikipedia.org/wiki/Fail-safe) for the equivalent in safety engineering). Ideally, a secure system should require a deliberate, conscious, knowledgeable and free decision on the part of legitimate authorities in order to make it insecure.

In addition, security should not be an all or nothing issue. The designers and operators of systems should assume that security breaches are inevitable. Full [audit trails](http://en.wikipedia.org/wiki/Audit_trail) should be kept of system activity, so that when a security breach occurs, the mechanism and extent of the breach can be determined. Storing audit trails remotely, where they can only be appended to, can keep intruders from covering their tracks. Finally, [full disclosure](http://en.wikipedia.org/wiki/Full_disclosure) helps to ensure that when bugs are found the "[window of vulnerability](http://en.wikipedia.org/wiki/Vulnerability_%28computing%29)" is kept as short as possible.