1. Abstract

Introduction

The Future generation applications are scalable and offered in both hosted and license options to suit companies of all sizes with a wide range of business needs.

“Air Freight Pursue” modules enable fast interoperability with business partners (customers and service providers) throughout the air cargo supply chain.

In addition to the core airline business, the Air Freight Pursue portfolio includes extensive ground handling modules addressing the needs of cargo quality principles. Customer or Company has to fill the basic information about their cargo.

Once customer or company has finished with, the quote request will be stored into Air Freight Pursue’s database and admin will proceed the work based on request. Air Freight Pursue provides tracking facility.

Through this online tracking facility user can get real time status information on your shipment.

Description

“Air Freight Pursue” is your source for Air Freight Services with the flexibility to meet your changing needs.

With our comprehensive range of time-definite, guaranteed and cost-saving options, Air Freight Pursue can meet your air freight requirements with reliability and professionalism.

Air Freight for shipments traveling nationwide via cargo, passenger or charter aircraft. Air Freight is a service for transporting cargo by Air.

This project mainly we have two modules, they are:

1. Administration Module
2. Employee Module

**1. Administration Module**

The web-site is administrated by the web-admin and controlling the complete activities of the project. The administrator will be having all the details of flights, employees, quote request, user comments etc. Administration module consist of different menus. They are listed below:

* 1. **User Request**

This is the first menu of Administration. When customer or company request any quote then it will store in database. Web admin can see all request posted by the customer or company from this menu.

* 1. **Employee Detail**

Web admin can add new employee. When new employee entered into organization, admin will assign employee id and password for them. Web admin can update the existing employee. When employee leave the organization admin can delete that employee.

* 1. **Air Cargo Scheduling**

This is the most important part of the project. From User Request admin can see all requests from customers or companies. After that admin work takes place. Admin assign work to appropriate employee. Admin will assign flight, generate cargo id and process id. According to the customer or company admin will schedule the flight.

* 1. **Cargo Tracking Detail**

In this module admin will store data for customers or company so that they can check real time status of their cargo. In this module admin will store the data like cargo id, source place and destination place, date of transfer, weight, price and current status of cargo.

* 1. **User Comments**

It is very important for any website or organization to have feedback from the users. This website provides this facility for visitor. Visitor can place comments from home page. In this module admin can see all the comments placed by the visitors.

* 1. **Reports**

The different type of reports to track the air freight process and procedures will be followed and the employee report, cargo tracking reports will be generated.

**2. Employee Module**

Employee can log into their module by entering user id and password provided by the admin. This module has different sub modules. They are listed below:

* 1. **View Scheduling**

Employee can see scheduling of air cargo assigned by admin. Admin assigns work to employee. Employees can confirm whether admin has assign work to them.

* 1. **Flight Allocation**

After employee confirms his or her work, they need to allocate pilot for flight. Employee need to store the data into database. Based on this allocation admin can track the process.

* 1. **Change Password**

At the time of registration admin will assign password for employee. But after that employee can reset his or her password from this sub module.

**2. Introduction**

2.1 Statement of Problem

As we all know that time is most important factor in business world. The main problem in the existing system is that company or person has to visit the office of air freight service provider. They need to fill the forms manually that take more time. So existing system was very time consuming for both shippers and providers. Air freight provider need to enter all the data into different registers manually. For assigning the process to different employee and pilot, admin has to meet them personally. So for admin also, it is very difficult to maintain organization. The most important thing is cargo tracking. In existing system shippers or company has to call or visit the air freight provider office which is time consuming and money consuming too.

2.2 Objective of project

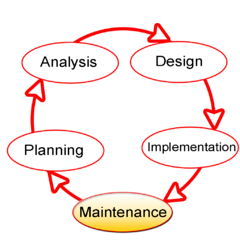
“Air Freight Pursue” is your source for Air Freight Services with the flexibility to meet your changing needs. With our comprehensive range of time-definite, guaranteed and cost-saving options, Air Freight Pursue can meet your air freight requirements with reliability and professionalism. Main objective of this project is to save time of both shippers and organization. As mentioned above time matters in business world, our aim is to make the service fully automated and time saving. Shippers or company can send their request within few second without any cost. They can track their cargo at any time from their home or office. Air Freight Pursue required no registration. With minimum effort, shipper or company will get maximum profit.

# 2.3 Methodology

# 2.3.1 Systems Development Life Cycle

The Systems Development Life Cycle (SDLC), or Software Development Life Cycle in systems engineering and software engineering, is the process of creating or altering systems, and the models and methodologies that people use to develop these systems. The concept generally refers to computer or information systems.

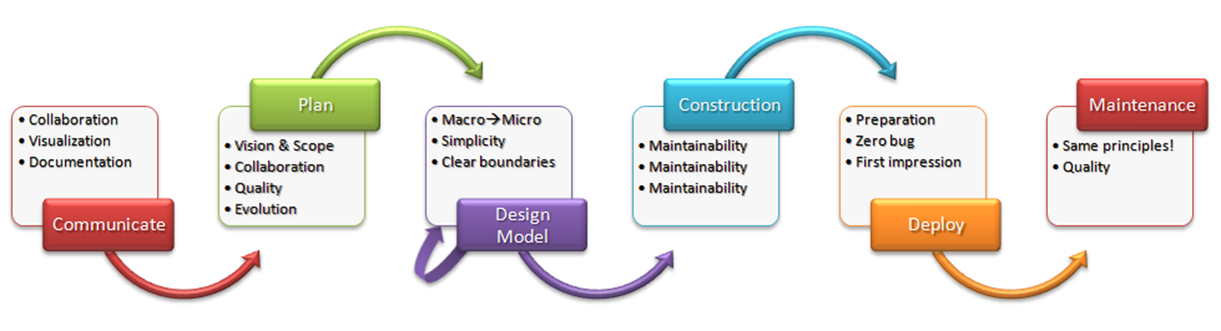
In software engineering the SDLC concept underpins many kinds of software development methodologies. These methodologies form the framework for planning and controlling the creation of an information system the software development process.



Systems Development Life Cycle (SDLC) is a logical process used by a systems analyst to develop an information system, including requirements, validation, training, and user ownership. Any SDLC should result in a high quality system that meets or exceeds customer expectations, reaches completion within time and cost estimates, works effectively and efficiently in the current and planned Information Technology infrastructure, and is inexpensive to maintain and cost-effective to enhance.

Computer systems are complex and often (especially with the recent rise of Service-Oriented Architecture) link multiple traditional systems potentially supplied by different software vendors. To manage this level of complexity, a number of systems development life cycle (SDLC) models have been created: "waterfall"; "fountain"; "spiral"; "build and fix"; "rapid prototyping"; "incremental"; and "synchronize and stabilize".

In project management a project can be defined both with a Project Life Cycle (PLC) and an SDLC, during which slightly different activities occur. According to Taylor "the project life cycle encompasses all the activities of the project, while the systems development life cycle focuses on realizing the product requirements".



### Initiation/planning

To generate a high-level view of the intended project and determine the goals of the project. The feasibility study is sometimes used to present the project to upper management in an attempt to gain funding. Projects are typically evaluated in three areas of feasibility: economical, operational or organizational, and technical. Furthermore, it is also used as a reference to keep the project on track and to evaluate the progress of the MIS team. The MIS is also a complement of those phases. This phase is also called as analysis phase.

### Requirements gathering and analysis

The goal of systems analysis is to determine where the problem is in an attempt to fix the system. This step involves breaking down the system in different pieces and drawing diagrams to analyze the situation, analyzing project goals, breaking down what needs to be created and attempting to engage users so that definite requirements can be defined. Requirements Gathering sometimes requires individuals/teams from client as well as service provider sides to get detailed and accurate requirements.

### Design

In systems design functions and operations are described in detail, including screen layouts, business rules, process diagrams and other documentation. The output of this stage will describe the new system as a collection of modules or subsystems.

The design stage takes as its initial input the requirements identified in the approved requirements document. For each requirement, a set of one or more design elements will be produced as a result of interviews, workshops, and/or prototype efforts. Design elements describe the desired software features in detail, and generally include functional hierarchy diagrams, screen layout diagrams, tables of business rules, business process diagrams, pseudo code, and a complete entity-relationship diagram with a full data dictionary. These design elements are intended to describe the software in sufficient detail that skilled programmers may develop the software with minimal additional input.

### Build or coding

Modular and subsystem programming code will be accomplished during this stage. Unit testing and module testing are done in this stage by the developers. This stage is intermingled with the next in that individual modules will need testing before integration to the main project.

### Testing

The code is tested at various levels in software testing. Unit, system and user acceptance testing’s often performed. This is a grey area as many different opinions exist as to what the stages of testing are and how much if any iteration occurs. Iteration is not generally part of the waterfall model, but usually some occur at this stage.

**Types of testing**

* Data set testing.
* Unit testing
* System testing
* Integration testing
* Black box testing
* White box testing
* Regression testing
* Automation testing
* User acceptance testing
* Performance testing

### Operations and maintenance

The deployment of the system includes changes and enhancements before the decommissioning or sunset of the system. Maintaining the system is an important aspect of SDLC. As key personnel change positions in the organization, new changes will be implemented, which will require system updates.

**WATERFALL MODEL**

The subject of Software Engineering doesn’t only deal with Software development but it about developing good software by using knowledge of available theories with the help of various defined methods and effective use of tools in hand.

There are various software development approaches defined and designed which are used/employed during development process of software, these approaches are also referred as "Software Development Process Models". Each process model follows a particular life cycle in order to ensure success in process of software development.

One such approach/process used in Software Development is "The Waterfall Model". Waterfall approach was first Process Model to be introduced and followed widely in Software Engineering to ensure success of the project. In "The Waterfall" approach, the whole process of software development is divided into separate process phases. The phases in Waterfall model are: Requirement Specifications phase, Software Design, Implementation and Testing & Maintenance. All these phases are cascaded to each other so that second phase is started as and when defined set of goals are achieved for first phase and it is signed off, so the name "Waterfall Model". All the methods and processes undertaken in Waterfall Model are more visible.

**The stages of "The Waterfall Model"**

**Requirement Analysis & Definition**: All possible requirements of the system to be developed are captured in this phase. Requirements are set of functionalities and constraints that the end-user (who will be using the system) expects from the system. The requirements are gathered from the end-user by consultation, these requirements are analyzed for their validity and the possibility of incorporating the requirements in the system to be development is also studied. Finally, a Requirement Specification document is created which serves the purpose of guideline for the next phase of the model.

**System & Software Design:** Before a starting for actual coding, it is highly important to understand what we are going to create and what it should look like? The requirement specifications from first phase are studied in this phase and system design is prepared. System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture. The system design specifications serve as input for the next phase of the model.

**Implementation & Unit Testing:** On receiving system design documents, the work is divided in modules/units and actual coding is started. The system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality; this is referred to as Unit Testing. Unit testing mainly verifies if the modules/units meet their specifications.

**Integration & System Testing:** As specified above, the system is first divided in units which are developed and tested for their functionalities. These units are integrated into a complete system during Integration phase and tested to check if all modules/units coordinate between each other and the system as a whole behaves as per the specifications. After successfully testing the software, it is delivered to the customer.

**Operations & Maintenance:** This phase of "The Waterfall Model" is virtually never ending phase (Very long). Generally, problems with the system developed (which are not found during the development life cycle) come up after its practical use starts, so the issues related to the system are solved after deployment of the system. Not all the problems come in picture directly but they arise time to time and needs to be solved; hence this process is referred as Maintenance.

**There are some disadvantages of the Waterfall Model:**

As it is very important to gather all possible requirements during the Requirement Gathering and Analysis phase in order to properly design the system, not all requirements are received at once, the requirements from customer goes on getting added to the list even after the end of "Requirement Gathering and Analysis" phase, this affects the system development process and its success in negative aspects.

The problems with one phase are never solved completely during that phase and in fact many problems regarding a particular phase arise after the phase is signed off, these results in badly structured system as not all the problems (related to a phase) are solved during the same phase.

The project is not partitioned in phases in flexible way.

As the requirements of the customer goes on getting added to the list, not all the requirements are fulfilled, this results in development of almost unusable system. These requirements are then met in newer version of the system; this increases the cost of system development.

* 1. **Limitations**

The study of the existing system has revealed many drawbacks. The main among these are:

1. Lots of paper work.
2. Time Consuming.
3. Inconsistency and inaccuracy in the maintenance of data.
4. Difficult to allocate process

**4. Technology used**

5. SRS

There are three major parties interested in a new system – the client, the users, and the developer. There is a communication gap between these people. A basic purpose of SRS is to bridge this communication gap. SRS is the medium through which the client and user needs are accurately specified; indeed SRS forms the basis of software development.

An important purpose of the process of developing an SRS is helping the client to understand their own needs. A good SRS provides many benefits. Some of the goals it accomplishes are:

* Establishing the basis for agreement between the client and supplier on what the software product will be reducing the development cost.
* The preparation of the SRS forces rigorous specification of the requirements before the design begins. Careful development of an SRS can reveal omissions, inconsistencies and misunderstanding early in the development cycle, which can considerably reduce cost.
* This software should be in position to maintain all the complete details of system i.e., Clients details, Client’s customer details, No of SMS used by clients, modifying customer details etc.

We should maintain the required details of all modules & update those details for any modifications.

* We should provide the clients with the dynamic message.
* We should be legible, readable & comprehensible.
* The software must be accessed by many clients by using different browsers.
* It must be capable of working on a variety of hardware.

**User Classes and Characteristics**

**Individual Local Developers:** Individual developers should be able to submit issues, edit issues, and perform queries on the database to discover what issues are relevant to them, which issues are open (in the case of issues to which that is relevant, such as defect reports or unsatisfied requirements), etc. These individual developers are assumed to have some knowledge of the development environment and are familiar and comfortable with basic software tools such as text editors etc. As a result, the individual developer tools will be the most "primitive" but also the most efficient for use, probably implemented as text-based command line tools. Since Yak Track is primarily intended as an easy-to-use, free tool for individual developers and small teams, this is the most critical user class to satisfy. The tools must be relatively easy to use, and extremely easy to set up.

**Local Issue Managers:** Issue managers -- those responsible for keeping track of open issues, etc. -- must have tools capable of querying the database and relating issues to developers. The tools used for issue managers and individual developers will be very similar, as they will be doing similar tasks -- querying the database for open issues, assigning people to issues as appropriate, recategorizing issues or merging/splitting them, etc. However, issue managers may not be as comfortable with "primitive" tools as individual developers, so some thought will be given to more "script" or directive tools, *possibly* involving simple GUI elements. However, the bulk of user-interface issues will be placed on the next user class, remote users.

**Non Functional Requirements & Constraints**

It's important to note that an SRS contains functional and nonfunctional requirements only; it doesn't offer design suggestions, possible solutions to technology or business issues, or any other information other than what the development team understands the customer's system requirements to be.

A well-designed, well-written SRS accomplishes four major goals:

* It provides feedback to the customer. An SRS is the customer's assurance that the development organization understands the issues or problems to be solved and the software behavior necessary to address those problems. Therefore, the SRS should be written in natural language, in an unambiguous manner that may also include charts, tables, data flow diagrams, decision tables, and so on.
* It decomposes the problem into component parts. The simple act of writing down software requirements in a well-designed format organizes information, places borders around the problem, solidifies ideas, and helps break down the problem into its component parts in an orderly fashion.
* It serves as a product validation check. The SRS also serves as the parent document for testing and validation strategies that will be applied to the requirements for verification.

**Safety Requirements**

For the applications to work efficiently it’s recommended that no manual intervention to the database should happen.

**Security**

The user management will add on to the application security.

**Reliability**

It’s a must that regular backups should be taken. The logs for the MYSQL Server database should be enabled. From the application side the error logs will be enabled.

**Maintainability**

Maintainability is the ease with which a program / specification can be corrected if an error occurs or the customer desires a change in requirements. Specific attributes of software that relate to the ease of maintenance of the software itself is to be incorporated into the software itself.

**H/W and S/W Requirements**

HARDWARE CONFIGURATION

Processor : Intel Pentium Family

Processor Speed : 250MHz to 667 MHz

RAM : 128 MB to 512 MB

Hard Disk : 4 GB or higher

Keyboard : Standard 104 enhanced keyboard

Mouse : Local PS/2

**SOFTWARE CONFIGURATION**

Operating System : Windows 98/2000/XP/NT

Web Server : Web logic Server

Web Browser : Mozilla Fire fox

Front-End Tool : HTML

Client side Script : JavaScript

Server side Script : JSP

Back-End Tool : MSSQL Server

Dynamic Tool : CSS

Design and Implementations

Entity relationship diagram

ER Diagrams

In software engineering, an entity-relationship model (ERM) is an abstract and conceptual representation of data. Entity-relationship modeling is a database modeling method, used to produce a type of conceptual schema or semantic data model of a system, often a relational database, and its requirements in a top-down fashion. Diagrams created by this process are called entity-relationship diagrams, ER diagrams, or ERDs.

The definitive reference for entity-relationship modeling is Peter Chen's 1976 paper. However, variants of the idea existed previously, and have been devised subsequently.

## Overview

The first stage of information system design uses these models during the requirements analysis to describe information needs or the type of information that is to be stored in a database. The data modeling technique can be used to describe any ontology (i.e. an overview and classifications of used terms and their relationships) for a certain area of interest. In the case of the design of an information system that is based on a database, the conceptual data model is, at a later stage (usually called logical design), mapped to a logical data model, such as the relational model; this in turn is mapped to a physical model during physical design. Note that sometimes, both of these phases are referred to as "physical design".

There are a number of conventions for entity-relationship diagrams (ERDs). The classical notation mainly relates to conceptual modeling. There are a range of notations employed in logical and physical database design, such as IDEF1X.

An Entity relationship Diagram is a data modelling technique that creates a graphical representation of the entities and the relationships between entities.

Entity

Entity

An entity is an object or concept about which you want to store information.

Weak Entity

Entity

A weak entity is dependent on another entity to exist.

Attribute

Attribute are the properties or characteristics of an entity

Key attribute

A key attribute is the unique, distinguishing characteristic of the entity. For example an employee’s social security number might be the employee’s key attribute.

Multivalued Attribute

A multivalued attribute can have more than one value. For example, an employee entity can have multiple skill values

Relationships Relationships illustrate how two entities share information in the database structure.

Relationship

**Detailed Entity relationship diagram**

**Employee**

1: n

**Has**

N: N

**Contains**

1: n

N: Ns

**Admin Login**

**Organization info**

Activity Diagrams

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

## Construction

Activity diagrams are constructed from a limited repertoire of shapes, connected with arrows. The most important shape types:

* rounded rectangles represent activities;
* diamonds represent decisions;
* a black circle represents the start (initial state) of the workflow;
* An encircled black circle represents the end (final state).

Arrows run from the start towards the end and represent the order in which activities happen.

Hence they can be regarded as a form of flowchart. Typical flowchart techniques lack constructs for expressing concurrency. However, the join and split symbols in activity diagrams only resolve this for simple cases; the meaning of the model is not clear when they are arbitrarily combined with decisions or loops

.

1. For emp login

User name

Password

Yes

Verify username and password

Valid

Invalid

Allocate process

Store info

b.For admin

User name

Password

Yes

Verify username and password

Invalid

Valid

website info edit

Ada,update,delete employee

Generate scheduling

Store info in database

**Data Flow Diagram**

A data flow diagram is graphical tool used to describe and analyze movement of the data through a system. These are the central tool and the basis form which the other components are developed. The transformation of the data from the input to output, through processed, may be described logically and independently of physical components associated with the system. These are known as the logical dataflow diagram. The physical data flow diagram show actual implements and movement of the data between people, departments and workstation, a full description of system actually consists of set of data flow diagrams. Using two familiar notations Yourdon, Gane and Sarson notation develops the data flow diagrams. Each component in DFD is labeled with a descriptive name. Process is further identified with number that will be used for identification purpose. The development of DTD’s is done in several levels. Each process in lower level diagram can be broken down into a more detailed DFD in the next level. Top-level diagram is often called context diagram. It consist a single process bit, which plays vital role studying the current system. The process at the first level DFD.

The idea behind the explosion of a process into more process is that understanding at one level of details is exploded into greater detailed at the next level. This is done until further explosion is necessary and an adequate amount of detail is described for analyst to understand the process.

Larry Constantine first developed the DFD as a way of expressing system requirements in a graphical form, this lead to the modular design.

A DFD is also known as a “Bubble Chart” has the purpose of clarifying system requirements and identifying major transformation the will become program in system design. So it is the starting point of the design to lowest level of details. A DFD consists of series of bubbles joined by data flows in the system.

**DFD Symbols**

In the DFD, there are four Symbols

1. A square define a source or destination system data
2. An arrow identified data flow. It is the pipeline through which the information flow
3. A circle or a bubble represents a process that transforms
4. Incoming data flow into outgoing data flows
5. An open rectangle is a data store, data at rest or a temporary of data

Process that transformation data flow

Source or Destination of data

**Constructing a DFD**

Several rules of thumb are used in drawing DFD’s

* Process should be named and numbered for an easy reference. Each name should be representative of the process.
* The direction of flow is top to bottom and from left to right. Data traditionally flow from source to the destination although they may flow back to the source. One way to indicate this is to draw long flow line back to the source. An alternative way is to repeat the source symbol as a destination. Since it is used more than once in the DFD it is marked with a short diagonal.
* When a process is exploded into lower level details, they are numbered.
* The names of data stores and destinations are written in capital letters. Process and dataflow names have the first letter of each work capitalized.

A DFD typically shows the minimum contents of data store. Each data store should contain all the data elements that flow in and out.

Questionnaires should contain all the data elements that flow in and out. Missing interface redundancies and like is than accounted for often through interviews.

**Features of DFD’s**

1. The DFD shows flow of data, not of control loops and decision are controlled consideration do not appear on a DFD.
2. The DFD does not indicate the time factor involved in any process whether the data flows take place daily, weekly, monthly or yearly.
3. The sequence of events is not brought out on the DFD.

**Types Of Data Flow Diagrams**

1. Current Physical
2. Current Logical
3. New Logical
4. New Physical

**Current Physical:**

In current physical DFD processes label include the name of people or their positions or the names of computer system that might provide some of the overall system processing label includes an identification of the technology used to process the data. Similarly data flows and data stores are often labels with the names of the actual physical media on which data are stored such as file folders, computer files, business from or computer tapes.

**Current Logical**

The physical aspects at the system are removed as mush as possible so that the current system is reduced to its essence to the data and the processors that transforms them regardless of actual physical form.

**New Logical**

This is exactly like a current logical model if the user were completely happy with he user were completely happy with functionality of the current system but had problems with how it was implemented typically through the new logical model will differ from current logical model while having additional functions, absolute function removal and inefficient flows recognized.

**New Physical**

The new physical represents only the physical implementation of the new system.

**Rules Governing The DFD’s Process**

* No process can have only outputs
* No process can have only inputs. If an object has only inputs than it must be a sink.
* A process has a verb phrase label.

**Data Store**

* Data cannot move directly from one data store to another data store, a process must move data.
* Data con not move directly from an outside source to a data store, a process, which receives, must move data from the source and place the data into data store.
* A data store has a noun phrase label.

**Source or sink**

The origin or destination of data

* Data cannot move direly form a source to sink it must be moved by process
* A source and or sink has a noun phrase land

**Data Flow**

* A data flow has only one destination of flow between symbols. It may flow in both directions between a process and a data store to show a read before an update. The later is usually indicated however by two separate arrows since these happen at different type.
* A join in DFD means that exactly the same data comes from any of two or more different processes data store or sink to a common location.
* A data flow cannot go directly back to the same process it leads. There must be at least one other process that handles the data flow produce some other data flow returns the original data into the beginning process
* A data flow to a data store means updates ( delete or change)
* A data flow from a data store means retrieve or use.
* A data flow has a noun phrase label more than one data flow noun phrase can appear on a single arrow as long as all of the flows on the same arrow move together as one package.

Sequence diagram

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart.

Sequence diagrams are sometimes called Event-trace diagrams, event scenarios, and timing diagrams.

**Overview**

A sequence diagram shows, as parallel vertical lines (lifelines), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

For instance, the UML 1.x diagram on the right describes the sequences of messages of a (simple) restaurant system. This diagram represents a Patron ordering food and wine, drinking wine then eating the food, and finally paying for the food. The dotted lines extending downwards indicate the timeline.

Time flows from top to bottom. The arrows represent messages (stimuli) from an actor or object to other objects. For example, the Patron sends message 'pay' to the Cashier. Half arrows indicate asynchronous method calls.

The UML 2.0 Sequence Diagram supports similar notation to the UML 1.x Sequence Diagram with added support for modeling variations to the standard flow of events.

**Diagram building blocks**

If the lifeline is that of an object, it demonstrates a role. Note that leaving the instance name blank can represent anonymous and unnamed instances.In order to display interaction, messages are used. These are horizontal arrows with the message name written above them. Solid arrows with full heads are synchronous calls, solid arrows with stick heads are asynchronous calls and dashed arrows with stick heads are return messages. This definition is true as of UML 2, considerably different from UML 1.x.

Activation boxes, or method-call boxes, are opaque rectangles drawn on top of lifelines to represent that processes are being performed in response to the message (Execution Specifications in UML).Objects calling methods on themselves use messages and add new activation boxes on top of any others to indicate further level of processing.

When an object is destroyed (removed from memory), an X is drawn on top of the lifeline, and the dashed line ceases to be drawn below it (this is not the case in the first example though). It should be the result of a message, either from the object itself, or another.A message sent from outside the diagram can be represented by a message originating from a filled-in circle (found message in UML) or from a border of sequence diagram (gate in UML).

UML 2 has introduced significant improvements to the capabilities of sequence diagrams. Most of these improvements are based on the idea of interaction fragments which represent smaller pieces of an enclosing interaction.Multiple interaction fragments are combined to create a variety of combined fragments, which are then used to model interactions that include parallelism, conditional branches, optional interactions etc.

**Usage and limitations**

Some systems have simple dynamic behavior that can be expressed in terms of specific sequences of messages between a small, fixed number of objects or processes. In such cases sequence diagrams can completely specify the system's behavior. Often, behavior is more complex, e.g. when the set of communicating objects is large or highly variable, when there are many branch points (e.g. exceptions), when there are complex iterations, or synchronization issues such as resource contention.

In such cases, sequence diagrams cannot completely describe the system's behavior, but they can specify typical use cases for the system, small details in its behavior, and simplified overviews of its behavior.

**Sequence Diagrams**

**1. Admin Module**

Update/delete info.

admin

Login page

Validate

Add info

View info

Request for submit

Login page username & if valid if valid

Password

if invalid

info

If invalid

**2. Employee module**

Add info

Validate

Login page

Employee

View info

Request for submit

Login page username if valid

Valid password

if invalid info

If invalid

**Use case diagram**

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases.

The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

**Overview**

Use Case diagrams are formally included in two modeling languages defined by the OMG. Both the UML and SysML standards define a graphical notation for modeling use cases with diagrams. One complaint about the standards has been that they do not define a format for describing these use cases. Generally, both graphical notation and descriptions are important as they document the use case, showing the purpose for which an actor uses a system.

The use case diagram shows the position or context of the use case among other use cases. As an organizing mechanism, a set of consistent, coherent use cases promotes a useful picture of system behavior, a common understanding between the customer/owner/user and the development team.

This diagram describes the functionality of a simple *Restaurant System*. Use cases are represented by ovals and the actors are represented by stick figures. The Patron actor can Order Food (and optionally Order Wine), Eat Food (and optionally Drink Wine), Pay for Food. Multiple actors participate in some of the use cases. In such cases, the label on the association between the actor and the use case throws some light on the actor's contribution to the use case. The box defines the boundaries of the Restaurant System, i.e., the use cases shown are part of the system being modeled, the actors are not.

**Diagram building blocks**

Interaction among actors is not shown on the use case diagram. If this interaction is essential to a coherent description of the desired behavior, perhaps the system or use case boundaries should be re-examined. Alternatively, interaction among actors can be part of the assumptions used in the use case.

### Actor Generalization

One popular relationship between Actors is Generalization/Specialization. This is useful in defining overlapping roles between actors. The notation is a solid line ending in a hollow triangle drawn from the specialized to the more general actor.

### Use Case Relationships

Three relationships among use cases are used often in practice.

#### Include

In one form of interaction, a given use case may *include* another. "Include is a Directed Relationship between two use cases, implying that the behavior of the included use case is inserted into the behavior of the including use case"

The first use case often depends on the outcome of the included use case. This is useful for extracting truly common behaviors from multiple use cases into a single description. The notation is a dashed arrow from the including to the included use case, with the label "«include»". This usage resembles a macro expansion where the included use case behavior is placed inline in the base use case behavior. There are no parameters or return values. To specify the location in a flow of events in which the base use case includes the behavior of another, you simply write include followed by the name of use case you want to include, as in the following flow for track order..

#### Extend

In another form of interaction, a given use case (the extension) may *extend* another. This relationship indicates that the behavior of the extension use case may be inserted in the extended use case under some conditions. The notation is a dashed arrow from the extension to the extended use case, with the label "«extend»". Notes or constraints may be associated with this relationship to illustrate the conditions under which this behavior will be executed.

Modelers use the «extend» relationship to indicate use cases that are "optional" to the base use case. Depending on the modeler's approach "optional" may mean "potentially not executed with the base use case" or it may mean "not required to achieve the base use case goal".

#### Generalization

In the third form of relationship among use cases, a *generalization/specialization* relationship exists. A given use case may have common behaviors, constraints and assumptions to the general use case, describe them once, and deal with it in the same way, except for the details in the specialized cases. The notation is a solid line ending in a hollow triangle drawn from the specialized to the more general use case.

**Use Case’s Description:**

Admin

Employee

8. Testing

Testing is a process used to help identify the correctness, completeness and quality of developed computer software. With that in mind, testing can never completely establish the correctness of computer software.

The quality of the application can and normally does vary widely from system to system but some of the common quality attributes include reliability, stability, portability, maintainability and usability. Refer to the ISO standard ISO 9126 for a more complete list of attributes and criteria.

Testing helps is verifying and Validating if the Software is working as it is intended to be working. Things involve using Static and Dynamic methodologies to Test the application.

* 1. Testing is a process of executing a program with the intent of finding an error.
  2. A good test case is one that has a high probability of finding an as yet undiscovered error.
  3. A successful test is one that uncovers an as yet undiscovered error.

Testing should systematically uncover different classes of errors in a minimum amount of time and with a minimum amount of effort. A secondary benefit of testing is that it demonstrates that the software appears to be working as stated in the specifications.

The data collected through testing can also provide an indication of the software's reliability and quality. But, testing cannot show the absence of defect, it can only show that software defects are present.

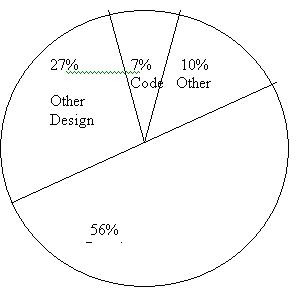
# Testing Start Process

Testing early in the life cycle reduces the errors. Test deliverables are associated with every phase of development. The goal of Software Tester is to find bugs, find them as early as possible, and make them sure they are fixed.

The number one cause of Software bugs is the Specification. There are several reasons specifications are the largest bug producer.

In many instances a Spec simply isn’t written. Other reasons may be that the spec isn’t thorough enough, it’s constantly changing, or it’s not communicated well to the entire team. Planning software is vitally important. If it’s not done correctly bugs will be created.

The next largest source of bugs is the Design, That’s where the programmers lay the plan for their Software. Compare it to an architect creating the blue print for the building, Bugs occur here for the same reason they occur in the specification. It’s rushed, changed, or not well communicated.



Coding errors may be more familiar to you if you are a programmer. Typically these can be traced to the Software complexity, poor documentation, schedule pressure or just plain dump mistakes. It’s important to note that many bugs that appear on the surface to be programming errors can really be traced to specification.

The other category is the catch-all for what is left. Some bugs can blamed for false positives, conditions that were thought to be bugs but really weren’t.

There may be duplicate bugs, multiple ones that resulted from the square root cause. Some bugs can be traced to Testing errors.

Costs: The costs re logarithmic- that is, they increase tenfold as time increases. A bug found and fixed during the early stages when the specification is being written might cost next to nothing, or 10 cents in our example. The same bug, if not found until the software is coded and tested, might cost $1 to $10. If a customer finds it, the cost would easily top $100.

# Testing Stop Process

This can be difficult to determine. Many modern software applications are so complex, and run in such as interdependent environment, that complete testing can never be done. "When to stop testing" is one of the most difficult questions to a test engineer. Common factors in deciding when to stop are:

* Deadlines (release deadlines, testing deadlines.)
* Test cases completed with certain percentages passed
* Test budget depleted
* Coverage of code/functionality/requirements reaches a specified point
* The rate at which Bugs can be found is too small
* Beta or Alpha Testing period ends
* The risk in the project is under acceptable limit.

Practically, we feel that the decision of stopping testing is based on the level of the risk acceptable to the management. As testing is a never ending process we can never assume that 100 % testing has been done, we can only minimize the risk of shipping the product to client with X testing done. The risk can be measured by Risk analysis but for small duration / low budget / low resources project, risk can be deduced by simply: -

* Measuring Test Coverage.
* Number of test cycles.
* Number of high priority bugs.

**8.1Test Plan**

Test plan consist of following points:

1. Title of the Project: “Air Freight Pursue”
2. Objective of the document: - In this test plan we are covering the activities and functionality of different modules and their sub modules. In this document we covering what kind of test cases should described.
3. Scope of the document: in this document, in each phase, what are going to do and how are going to do. In this section we are mentioning which requirements are testing. Take one module i.e. user login module.

For user module the client requirements are:

* + User name must be not less than 4 character and not greater than 10.
  + Username must start with character, not with digits.
  + Username should not contain any special character.
  + Password field should fill with at lest 4character.

The above requirements are only for login module and other functional requirements can declared in this section. The requirements which are declared in this section it may depend on other requirements on different module.

1. Objective of Testing: the main objective of testing in this application is to chances of preventing the defects on the client environment.
2. Brief Description of the application:-
3. Critical Functionalities: - in this section we are discussing Key roles & Causes for success of application.
4. Test Data Requirements and Collection:-

In this section we are collecting the requirements for the application. From the different resources.

Such as

* Collecting from the client.
* Refereeing the existing applications which are similar to current one’s

8) Training Requirements:-

Training requirements are focused in 2 areas

1. Technology – in this we are discussing about the new technology in the market and scope of that technologies in the future.
2. Domain- in this we are discussing about the Existing personal Training knowledge.

9) Resource Requirements: - In this we discussing about the required resources for the application such

* + Employees
  + Software Licenses
  + Bridge Number

10) Scheduling: - In this section we will be discussing about Staring Date and End Date of the application are given.

11) Input Criteria: - there are different criteria conditions i.e. Unit Testing, Release Note & Installation.

12) Exit Criteria: - In this only evaluation documents only accepted.

13) Risk Analysis: - In this we are converse about Risk Analysis such as

* Risk on Resources.
* Risk on TimeLine.
* Risks are identified by Preparing Solution Plan.

**8.2 Types of Tests Carried:-**

In our application we are performing the Gray Box testing (white and black).

Test case for Login Form:

|  |  |  |
| --- | --- | --- |
| Steps | Descriptions | Executed Result |
| 1. | Launch the Application | Application Should be Launched. |
| 2 | Enter Valid Username & Password In respective fields | Data should be entered in fields. |
| 3 | Click on Login” Button | Field should be validating. If not show error Message |

10. CONCLUSION

The project “Air Freight Pursue” is successfully completed in time with all deliverables meeting the requirements. The system objectives were fulfillment with all the requirements specified by the analysis done were completed satisfactorily.

The software will be launched for the internal use of Cegonsoft Pvt. Ltd. and it is efficiently handled the complete process and is satisfied with performance of the system. The project has been of great help to me in gaining valuable information on web application. It has given me a great satisfaction in having designed an application that has importance on real world.

The system is fully functional, easy to use with no user training or orientation necessary. The screens are GUI based data entry screens and very interactive. Proper status messages are displayed and hence easy for user to interact and understand.

Using the features of PHP the project proved to be successful in performing complex diagnostics, route data between screens and generate required responses. They were many tangible and intangible benefits in implementing the System. Thus the project was successfully developed and completed using PHP and MYSQL Server as the back end.

# Software Scope

The software is a web based application; it’s a collection of Party Halls where user can search the Party Halls from the web freely. In this project the user can directly interact with the form they can get required information if it is in the database

The Objective of this software is to make people to browse this site and give them a user-friendly site.