CS307 CASE TOOLS LABORATORY

OBJECTIVE:

To develop a mini-project for the following experiments listed below.

1. To Identify Project scope, Objectives and Infrastructure
2. To Develop IEEE SRS document
3. To Develop Data Dictionary and Use case Diagram
4. To Develop Activity diagram and Class diagram
5. To Develop Sequence diagrams and Collaboration Diagram
6. To Add interface to class diagram
7. Implement the design by coding.
8. To Prepare test plan
9. To perform validation testing
10. To perform Coverage analysis
11. To identify memory leaks
12. To Develop test case hierarchy
13. To perform Site check and to monitor

LIST OF EXPERIMENTS:-

1. Passport automation system.
2. Book bank.
3. Exam Registration.
4. Software personnel management system.
5. Credit card processing.
6. e-book management system.
7. Foreign trading system.
8. Conference Management System.
9. BPO Management System.
10. Online Quiz System.

INTRODUCTION TO UNIFIIED MODELING LANGUAGE (UML)

UML

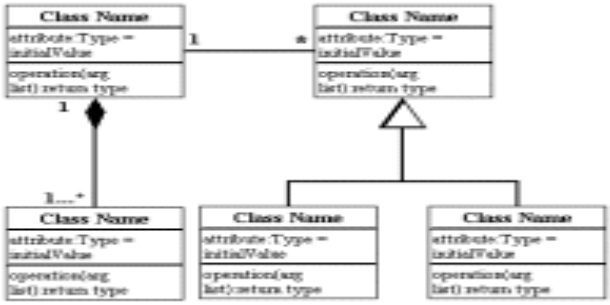
UML stands for Unified Modeling Language. This object-oriented system of notation has evolved from the work of Grady Booch, James Rum Baugh, Ivar Jacobson, and the Rational Software Corporation. These renowned computer scientists fused their respective technologies into a single, standardized model. Today, UML is accepted by the Object Management Group (OMG) as the standard for modeling object oriented programs.

Types of UML Diagrams

UML defines nine types of diagrams: class (package), object, use case, sequence, collaboration, state chart, activity, component, and deployment.

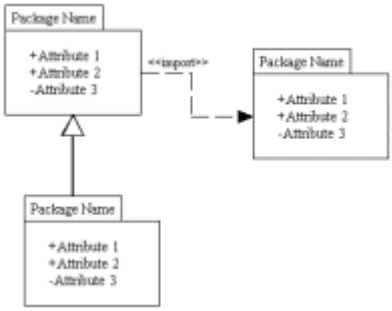
Class Diagrams

Class diagrams are the backbone of almost every object oriented method, including UML. They describe the static structure of a system.



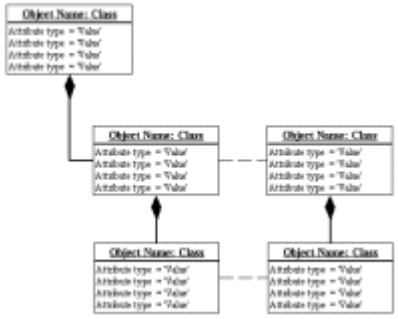
Package Diagrams

Package diagrams are a subset of class diagrams, but developers sometimes treat them as a separate technique. Package diagrams organize elements of a system into related groups to minimize dependencies between packages.



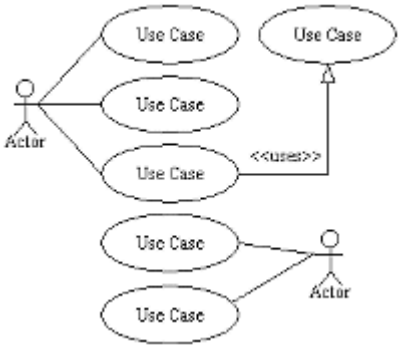
Object Diagrams

Object diagrams describe the static structure of a system at a particular time. They can be used to test class diagrams for accuracy.



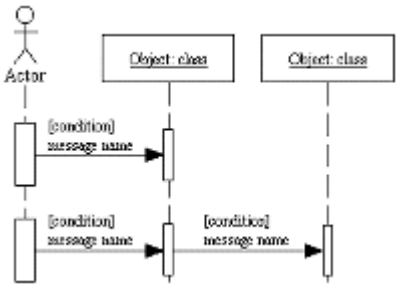
Use Case Diagrams

Use case diagrams model the functionality of system using actors and use cases.



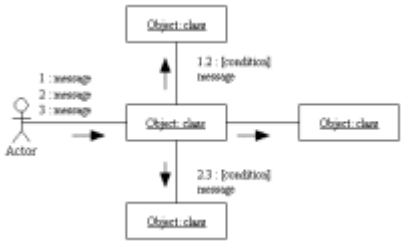
Sequence Diagrams

Sequence diagrams describe interactions among classes in terms of an exchange of messages over time.



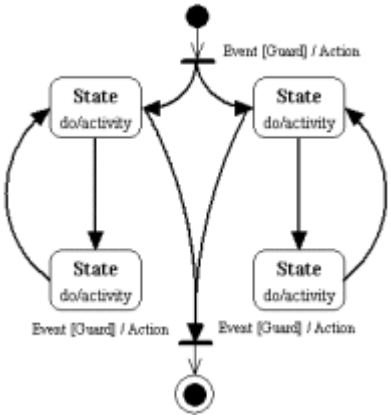
Collaboration Diagrams

Collaboration diagrams represent interactions between objects as a series of sequenced messages. Collaboration diagrams describe both the static structure and the dynamic behavior of a system.



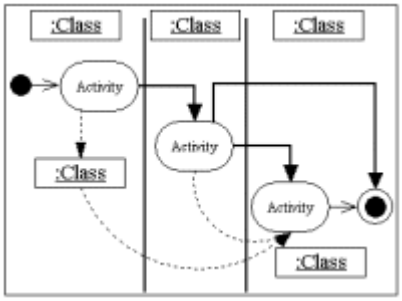
State chart Diagrams

State chart diagrams describe the dynamic behavior of a system in response to external stimuli. Statechart diagrams are especially useful in modeling reactive objects whose states are triggered by specific events.



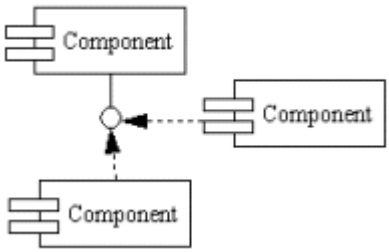
Activity Diagrams

Activity diagrams illustrate the dynamic nature of a system by modeling the flow of control from activity to activity. An activity represents an operation on some class in the system that results in a change in the state of the system. Typically, activity diagrams are used to model workflow or business processes and internal operation.



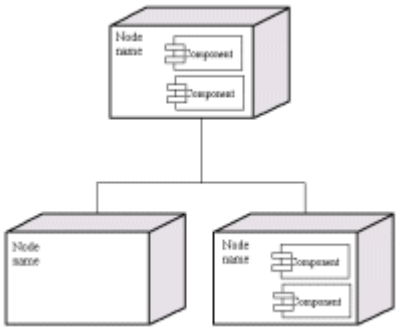
Component Diagrams

Component diagrams describe the organization of physical software components, including source code, run-time (binary) code, and executables.



Deployment Diagrams

Deployment diagrams depict the physical resources in a system, including nodes, components, and connections.

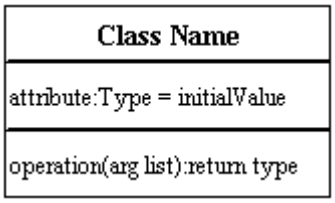


Class diagrams are the backbone of almost every object-oriented method including UML. They describe the static structure of a system.

Basic Class Diagram Symbols and Notations

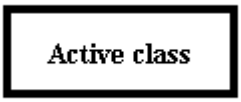
Classes represent an abstraction of entities with common characteristics. Associations represent the relationships between classes.

Illustrate classes with rectangles divided into compartments. Place the name of the class in the first partition (centered, bolded, and capitalized), list the attributes in the second partition, and write operations into the third.



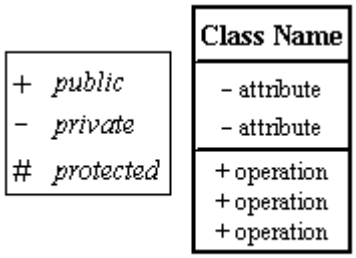
Active Class

Active classes initiate and control the flow of activity, while passive classes store data and serve other classes. Illustrate active classes with a thicker border.



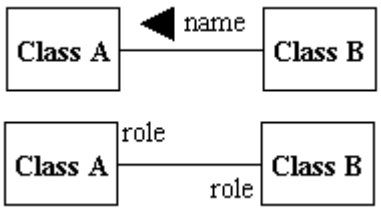
Visibility

Use visibility markers to signifiy who can access the information contained within a class. Private visibility hides information from anything outside the class partition. Public visibility allows all other classes to view the marked information. Protected visibility allows child classes to access information they inherited from a parent class.



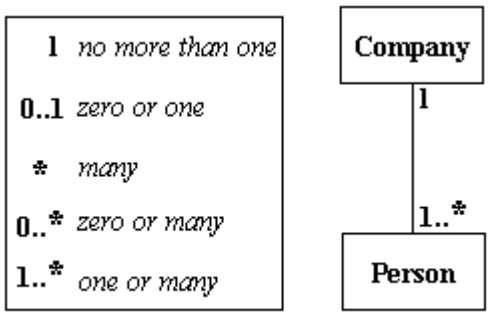
Associations

Associations represent static relationships between classes. Place association names above, on, or below the association line. Use a filled arrow to indicate the direction of the relationship. Place roles near the end of an association. Roles represent the way the two classes see each other.
Note: It's uncommon to name both the association and the class roles.



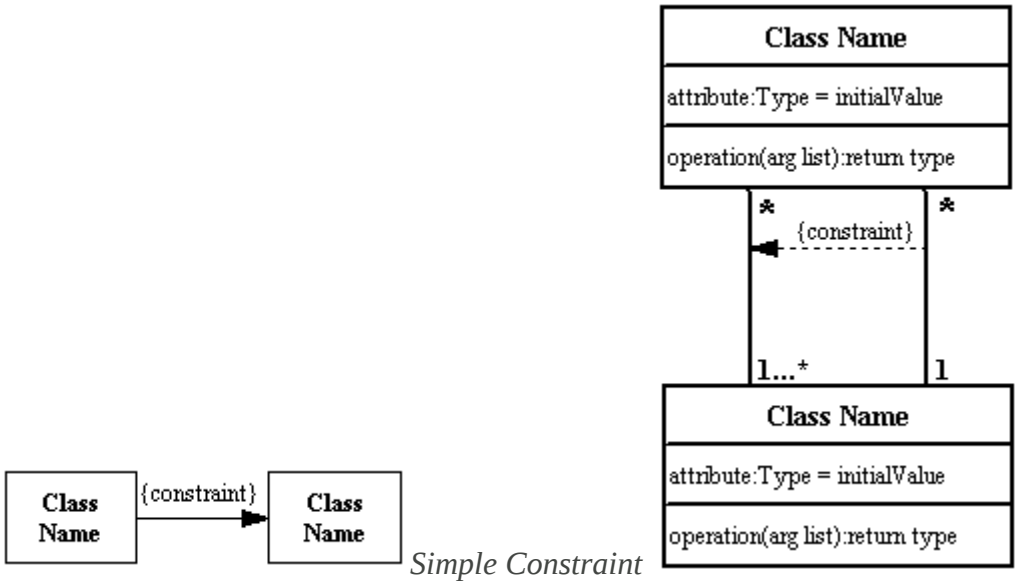
Multiplicity (Cardinality)

Place multiplicity notations near the ends of an association. These symbols indicate the number of instances of one class linked to one instance of the other class. For example, one company will have one or more employees, but each employee works for one company only.



Constraint

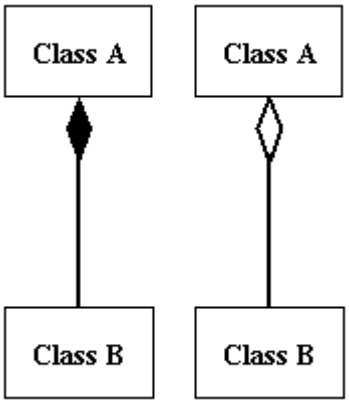
Place constraints inside curly braces {}.



Simple Constraint

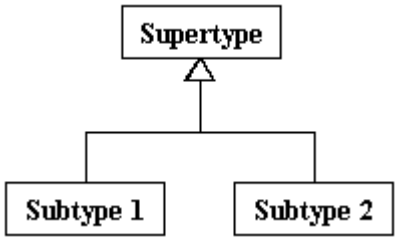
Composition and Aggregation

Composition is a special type of aggregation that denotes a strong ownership between Class A, the whole, and Class B, its part. Illustrate composition with a filled diamond. Use a hollow diamond to represent a simple aggregation relationship, in which the "whole" class plays a more important role than the "part" class, but the two classes are not dependent on each other. The diamond end in both a composition and aggregation relationship points toward the "whole" class or the aggregate.



Generalization

Generalization is another name for inheritance or an "is a" relationship. It refers to a relationship between two classes where one class is a specialized version of another. For example, Honda is a type of car. So the class Honda would have a generalization relationship with the class car.



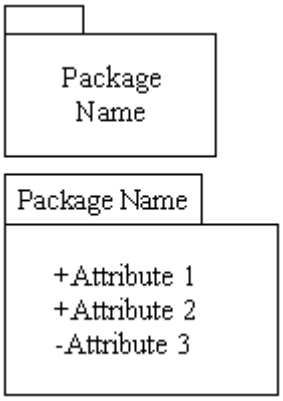
In real life coding examples, the difference between inheritance and aggregation can be confusing. If you have an aggregation relationship, the aggregate (the whole) can access only the PUBLIC functions of the part class. On the other hand, inheritance allows the inheriting class to access both the PUBLIC and PROTECTED functions of the super class.

Package diagrams organize the elements of a system into related groups to minimize dependencies among them.

Basic Package Diagram Symbols and Notations

Packages

Use a tabbed folder to illustrate packages. Write the name of the package on the tab or inside the folder. Similar to classes, you can also list the attributes of a package.

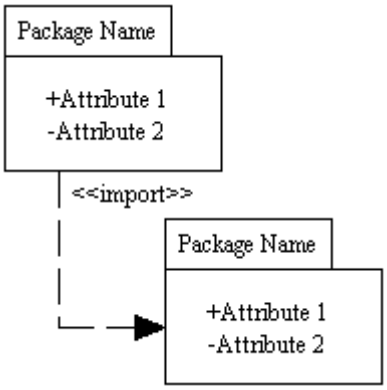


Visibility

Visibility markers signify who can access the information contained within a package. Private visibility means that the attribute or the operation is not accessible to anything outside the package. Public visibility allows an attribute or an operation to be viewed by other packages. Protected visibility makes an attribute or operation visible to packages that inherit it only.

Dependency

Dependency defines a relationship in which changes to one package will affect another package. Importing is a type of dependency that grants one package access to the contents of another package.

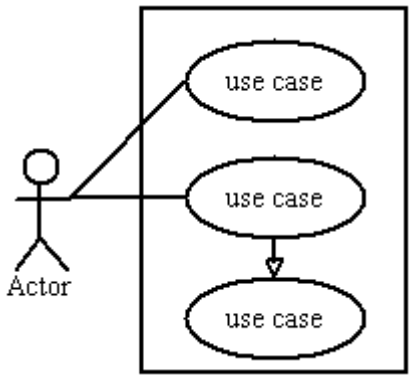


Use case diagrams model the functionality of a system using actors and use cases. Use cases are services or functions provided by the system to its users.

Basic Use Case Diagram Symbols and Notations

System

Draw your system's boundaries using a rectangle that contains use cases. Place actors outside the system's boundaries.

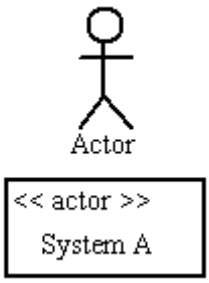


Use Case

Draw use cases using ovals. Label with ovals with verbs that represent the system's functions.

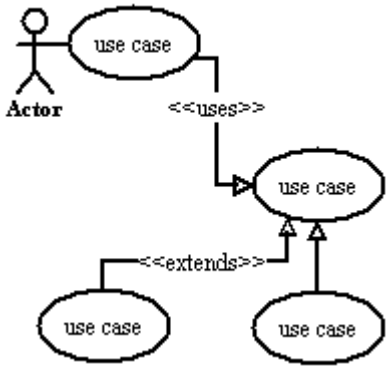
Actors

Actors are the users of a system. When one system is the actor of another system, label the actor system with the actor stereotype.



Relationships

Illustrate relationships between an actor and a use case with a simple line. For relationships among use cases, use arrows labeled either "uses" or "extends." A "uses" relationship indicates that one use case is needed by another in order to perform a task. An "extends" relationship indicates alternative options under a certain use case.

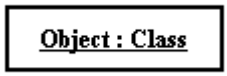


Sequence diagrams describe interactions among classes in terms of an exchange of messages over time.

Basic Sequence Diagram Symbols and Notations

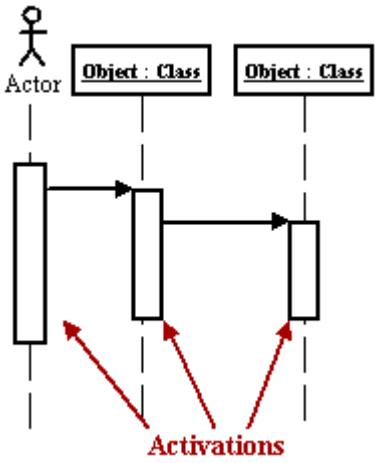
Class roles

Class roles describe the way an object will behave in context. Use the UML object symbol to illustrate class roles, but don't list object attributes.



Activation

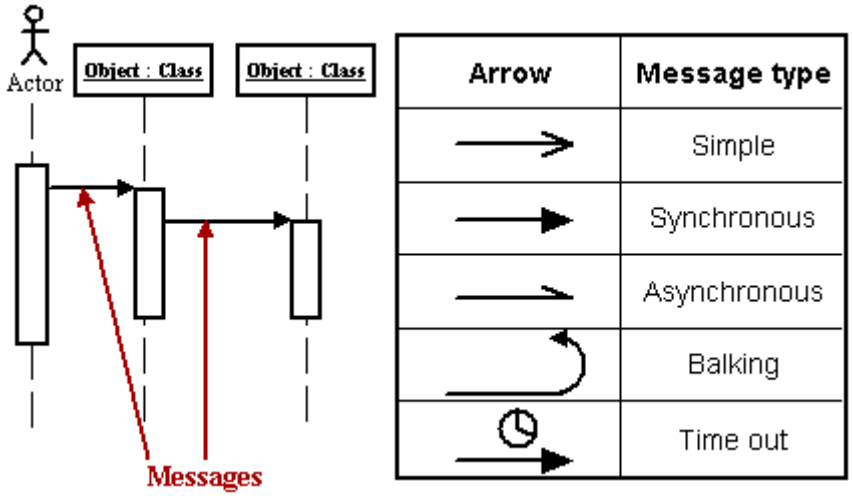
Activation boxes represent the time an object needs to complete a task.



Messages

Messages are arrows that represent communication between objects. Use half-arrowed lines to represent asynchronous messages. Asynchronous messages are sent from an object

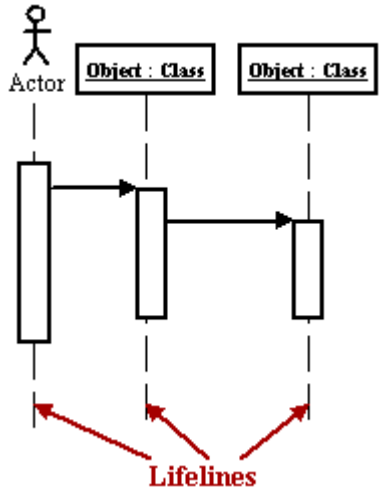
that will not wait for a response from the receiver before continuing its tasks.



Various message types for Sequence and Collaboration diagrams

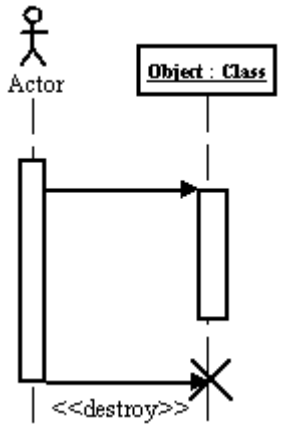
Lifielines

Lifielines are vertical dashed lines that indicate the object's presence over time.



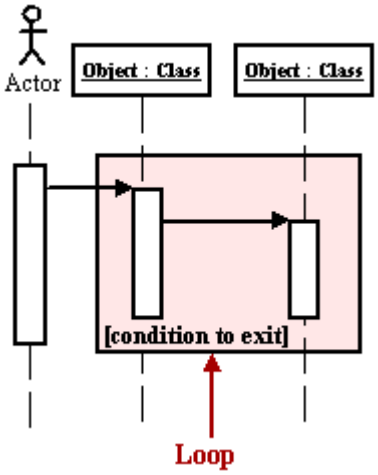
Destroying Objects

Objects can be terminated early using an arrow labeled "<< destroy >>" that points to an X.



Loops

A repetition or loop within a sequence diagram is depicted as a rectangle. Place the condition for exiting the loop at the bottom left corner in square brackets [ ].

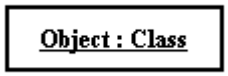


A collaboration diagram describes interactions among objects in terms of sequenced messages. Collaboration diagrams represent a combination of information taken from class, sequence, and use case diagrams describing both the static structure and dynamic behavior of a system.

Basic Collaboration Diagram Symbols and Notations

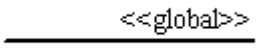
Class roles

Class roles describe how objects behave. Use the UML object symbol to illustrate class roles, but don't list object attributes.



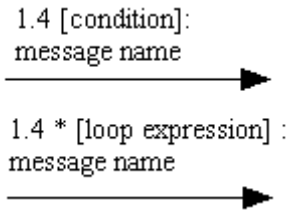
Association roles

Association roles describe how an association will behave given a particular situation. You can draw association roles using simple lines labeled with stereotypes.



Messages

Unlike sequence diagrams, collaboration diagrams do not have an explicit way to denote time and instead number messages in order of execution. Sequence numbering can become nested using the Dewey decimal system. For example, nested messages under the first message are labeled 1.1, 1.2, 1.3, and so on. The a condition for a message is usually placed in square brackets immediately following the sequence number. Use a * after the sequence number to indicate a loop.

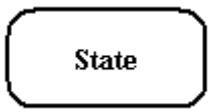


A state chart diagram shows the behavior of classes in response to external stimuli. This diagram models the dynamic flow of control from state to state within a system.

Basic Statechart Diagram Symbols and Notations

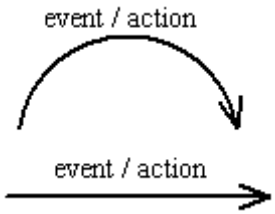
States

States represent situations during the life of an object.



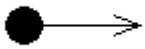
Transition

A solid arrow represents the path between different states of an object. Label the transition with the event that triggered it and the action that results from it.



Initial State

A filled circle followed by an arrow represents the object's initial state.

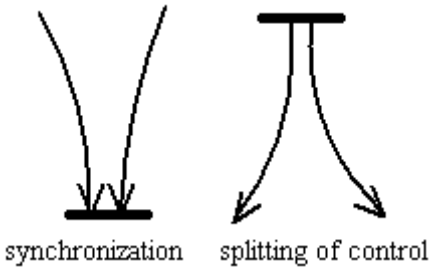


Final State

An arrow pointing to a filled circle nested inside another circle represents the object's final state.

Synchronization and Splitting of Control

A short heavy bar with two transitions entering it represents a synchronization of control. A short heavy bar with two transitions leaving it represents a splitting of control that creates multiple states.



What is a UML Activity Diagram?

An activity diagram illustrates the dynamic nature of a system by modeling the flow of control from activity to activity. An activity represents an operation on some class in the system that results in a change in the state of the system. Typically, activity diagrams are used to model workflow or business processes and internal operation. Because an activity diagram is a special kind of state chart diagram, it uses some of the same modeling conventions.

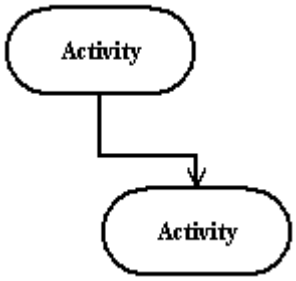
Basic Activity Diagram Symbols and Notations

Action states

Action states represent the no interruptible actions of objects.

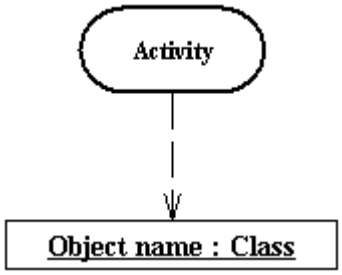
Action Flow

Action flow arrows illustrate the relationships among action states.



Object Flow

Object flow refers to the creation and modification of objects by activities. An object flow arrow from an action to an object means that the action creates or influences the object. An object flow arrow from an object to an action indicates that the action state uses the object.



Initial State

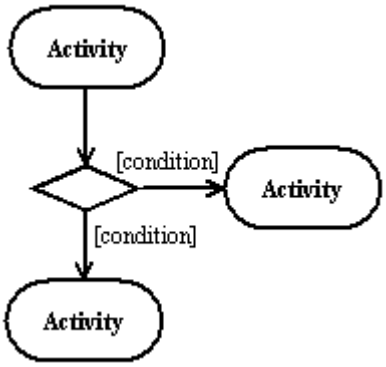
A filled circle followed by an arrow represents the initial action state.

Final State

An arrow pointing to a filled circle nested inside another circle represents the final action state.

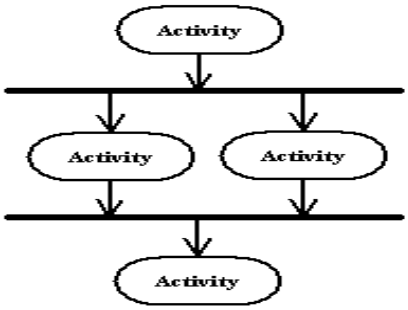
Branching

A diamond represents a decision with alternate paths. The outgoing alternates should be labeled with a condition or guard expression. You can also label one of the paths "else."



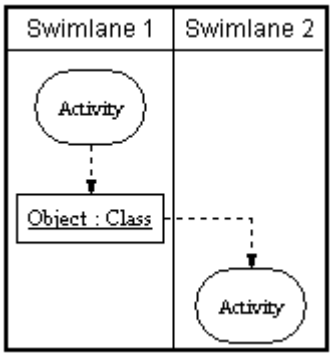
Synchronization

A synchronization bar helps illustrate parallel transitions. Synchronization is also called forking and joining.



Swim lanes

Swim lanes group related activities into one column

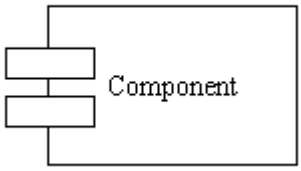


A component diagram describes the organization of the physical components in a system.

Basic Component Diagram Symbols and Notations

Component

A component is a physical building block of the system. It is represented as a rectangle with tabs.

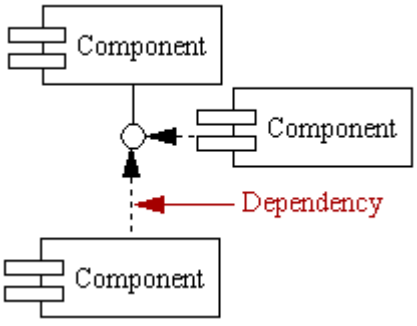


Interface

An interface describes a group of operations used or created by components.

Dependencies

Draw dependencies among components using dashed arrows.

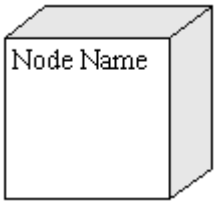


Deployment diagrams depict the physical resources in a system including nodes, components, and connections.

Basic Deployment Diagram Symbols and Notations

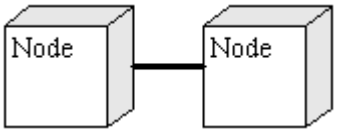
Component

A node is a physical resource that executes code components.



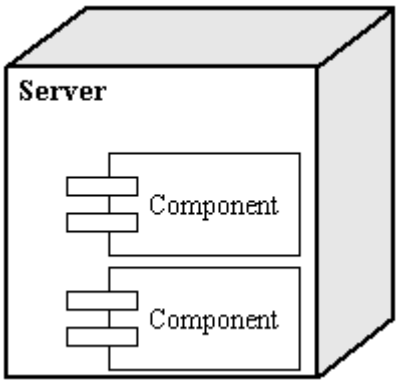
Association

Association refers to a physical connection between nodes, such as Ethernet.



Components and Nodes

Place components inside the node that deploys them.



PROBLEM STATEMENTS

1. PASSPORT AUTOMATION SYSTEM

Passport Automation System was developed to support and control the issue of new ePassport. The e-Passport System utilizes the latest technology to enhance the operation efficiency and accuracy while, at the same time, to minimize manpower required through automation and streamlined procedures. To facilitate ePassport applicants, e-Passport System supports application submission through multiple channels. Apart from the traditional submission channels, via submission by post, by drop-in boxes or in-person, applicants aged 18 or above may also choose to submit applications through the Internet or by the innovative ePassport self-service kiosks.

The new e-Passport System is integrated with state-of-the-art printing and security technologies. To help counter passport forgeries, illegal immigration and other transnational crimes facilitated by passport frauds, advanced security features and sophisticated printing techniques have been deployed in e-Passport System. Laser engraving technology is also applied to engrave the personal data and photo image onto the ePassport. The system employs biometric technologies to verify the applicant's identity

An eCabinet system was developed to strengthen inventory management and distribution of ePassport. The system supports automatic check-in, check-out, retrieval, stock-taking, storage and inventory control of ePassports. This efficient and automatic system minimizes manpower required and human errors while maintaining a highly secure storage environment for ePassports.

2. BOOK BANK

A student of any course should be allowed to borrow books. Book bank books are issued only for their respective courses. A limitation is imposed on the number of books issued to a student.

A maximum of 4 books from book bank per student is allowed. The books from book bank are issued for the entire semester. The software takes the current system date as the date of issue and calculates the corresponding date of return. A bar code detector is used to save the student's information as well as book information. The due date for return of the book is stamped on the book. Any person can return the issued books. The student information is displayed using the bar code detector. The system displays the student details on whose

name the books were issued as well as the date of issue and return of the book. The system operator verifies the duration of the issue and if the book is returned after the specified due date, a fine of Re.1 is charged for each day. The information is saved and the corresponding updating are done in the database.

The system should be able to provide information like:

i) Availability of a particular book.

ii) Availability of books of any particular author.

iii) Number of copies available of the desired book.

The system should be reserve a book for a particular student for 24 hours ifi that book is not currently available. The system should be able to generate the reports regarding the details of the book available in the library at any given time. The corresponding printouts for each entry (issue/return) made in the system should be generated. Security provisions like login authenticity should be provided. Each user should have a user id and password. Record of the users of system should be kept in the log file. Provision should be made for full backup of the system.

3. EXAM REGISTRATION

The web-based examination system assists an examinator to create an exam by adding and modifying questions, supplying possible results of a question and selecting the intended group of students. Moreover the examinator will have an overview of the results of an exam by student, group or other factors. A student can solve exams for which they are enrolled and view results and corrections of previous exams. The product benefits the examinator by simplifying and speeding up the work of creating and correcting an exam and provides a easy way to follow the results of a student or group of students. An administrator will setup and maintain the enrollments and courses. The product will be entirely accessed by a user-friendly web-interface which means that a minimum amount of computer knowledge is required to interact with it.

Authentication: Every user (examiner, administrator etc) is authenticated before allowing to login.

Automation: Examination is automated completely. The questions are asked randomly by the system. The checking of questions is done by the system using pre-defined answers fed to the system. Hence results are available immediately after the exam. All exam data is available on the intranet/internet depending upon access rights. Modules need to be cleared sequentially hence flags are set. Passing criteria can be easily set or modified. Time-line can be set for clearing exam and modules.

Study material: Study material is available online and is accessible depending on modules created.

Study material: Various results are prepared and also in-built functionality is available for analysis of results.

Monitoring of Candidate: Expert system can monitor candidates performance. The rules are added/modified by experts when and as required.

4. SOFTWARE PERSONNEL MANAGEMENT SYSTEM

Software personnel management system provides services to collect organize and manage the information about each employee in a company or organization.

- Collecting and manage the personal details about each employee.(eg: phone number,address,etc..,)
- Current status of the employee's present project.(whether the project is in designing stage or implementation stage like that.)
- Employee's schedule of the current week. (e.g.: meeting, appointments or any other commitments)
- History of projects done by the employee
- Details about the projects assigned to the employee.

5. CREDIT CARD PROCESSING

Stage 1.) Credit card processing starts the moment the credit card is keyed into the processing terminal.

Stage 2.) The movement of funds begins when the customer gives the merchant their credit/debit information. Initially the funds are authorized but are not placed into the merchant's account until the end of the day. Funds are moved between the customer's bank and then to the merchant's bank that does the processing.

Stage 3.) The authorization in Stage 2 occurs after the customer's information, i.e. name, account number, is sent to the merchant's processing bank by the debit/credit card companies who are: Visa, Master Card, Discover and American Express. These 'gateways' handle the process for using these cards.

Stage 4.) The merchant decides if they want to handle American Express and Discover; all of American Expresses and Discovery's applications, are handled by them individually which entails banking relationships with merchants etc., The companies issues their own cards, processing set-up and all systems related to the transaction(s) is developed and managed by them.

Stage 5.) The merchant decides if they want to handle Visa and Master Card, it is then automatically set up when the merchant decides that they want to accept these cards.

Stage 6.) Transactions initiated at the merchant's terminal have all the customer's information and is handled by a 800 number or by the internet to the bank that handles the processing for the merchant.

Stage 7.) At the moment of transaction the merchant knows within a minute of it ifi it has been accepted or declined. Before midnight, on the day of the transaction, all transactions are placed together or "batched". The merchant, by this batching has to pay a "batch fee."

Stage 8.) The funds arrive in the merchant's account no later than 48 hours of the initial transaction. More often it occurs within 24 hours.

6. E-BOOK MANAGEMENT SYSTEM

e-book management is primarily an e-book cataloging program. It is designed around the concept of the logical book, where a single database entry corresponds with the same book in a variety of formats. It supports the following formats for cataloging: AZW, AZW1, CBR, CBZ, CHM, EPUB, FB2, HTML, IMP, LIT, LRF, LRX, MOBI, ODT, OEBZIP, OPF, PDB, PDF, PML, PMLZ, PRC, RAR, RB, RTF, SNB, TPZ, TXT and ZIP.

The system supports sorting the books in its database by:

- Title

- Author
- Date
- Publisher
- Rating
- Size (Max size of all formats)
- Series

It also supports extra metadata fields that can be searched on:

- Comments: A general purpose field that can be used to describe the book.
- Tags: A flexible system to categorize books.

It also supports metadata retrieval via the Internet for a book based on its ISBN number or its title/author, instead of manually entering the metadata.

It also supports a built-in web server that allows users to access their e-book collection using an Internet browser and an Internet connection. The web server also allows the user to e-mail books and download news automatically

7. FOREIGN TRADING SYSTEM

The main users of the systems are the traders, who perform individual trades from their desks, mostly over the phone. Traders are supported by personal workstations, connected to larger corporate servers (or administrators in our parlance). Traders rely on many sources of market information, some informal and some formal; the workstation is used to convey some of this market information, which is provided by the system of interest in two forms: pricing information and market news, from both wire services and financial information providers. Pricing information includes current prices of specifiic commodities (e.g. oil) for dififerent time periods. Market news includes press releases, demand and supply forecasts, wire stories, etc. The system must collect, filter, and disseminate these news to traders.

A trader executes a trade after completing a negotiation with another trader (called a counterparty in commodities parlance). To complete the trade, a formal contract must be generated specifiying the terms of the deal (e.g. volume, price, quantity, delivery date, mode of transportation, or combination thereof). In addition, the trader's position in the

commodity being traded must be updated appropriately. The position is captured in a schedule often known in the industry as a slate, which gives a summary of all agreed to receipts and deliveries for a given <commodity, location, month> combination, as defined by all contracts pertaining to that combination. Receipts and deliveries of a trade are called its legs.

When a counterparty trader wishes to trade a commodity she contacts a company trader and the request to trade the commodity is announced.The company trader analyzes the offer to trade. For a detailed discussion of this phase see the “analyze potential trade” use case.

The terms and conditions of the deal are then discussed and negotiated with the counterparty trader. This constitutes acceptance of the trade.Upon acceptance, a new trade is created

The details of the trade are registered with the Trade Administrator

The Trade Administrator makes a permanent, persistent record of the trade details and notifiies the Position Administrator of the trade.Ifi the trade falls within an existing slate, that slate is simply updated with the details of the new trade.

Ifi no previous trades for the same commodity, movement location and movement period have been made, the Position Administrator creates a new slate. In either case a permanent, persistent record of the slate state is made.

The Trade Administrator provides the trade information to the Market Data Service for internal news distribution

The Trade Administrator requests the Contract Administrator to produce a formal printed contract

8. CONFERENCE MANAGEMENT SYSTEM

In recent years, with the improvement of network technologies and hardware supports, we can find that network have become an important part in our daily lifie. In traditional, scholars can get new knowledge and exchange their ideas with others by joining a conference. But, however this will cause time and cost consuming. Web-based conference management system that supports most administrative tasks of conference, workshop, and seminar organizers. The system manages all vital processes a conference may have, including user registration, paper submission and review, collecting payments from participants and invoicing. System users (authors, reviewers, track chairs, conference

chairs, and conference managers) can easily access the authorized parts of the system from his or her computer with a web browser and internet connection. Install, configure, and customize the system according to your needs. All configurations can be managed through an administrative control panel. Multiple tracks can be created, each of which can be a session or panel.

Depending on your choice, the type of submission can be an abstract, full proposal, abstract and proposal together or abstract followed by presentation. The system can process any kind of file formats submitted. The system provides a built-in communication system to send automated or manual emails to all users involved. The email templates can be easily customized.

Author: Submits papers and/or supporting materials to the system via his account.

Reviewer: Reviews assigned papers and submits review reports via his account.

Track Chair: Manages the submission and review process for the tracks assigned.

Conference Chair: Oversees the submission and review process for all the tracks.

Registration Manager: Manages online payments sent by users via the system.

Conference Manager: Manages the entire system by using the admin panel.

9. BPO MANAGEMENT SYSTEM

BPO Management System provides business and technology solutions to help its clients worldwide improve their business performance. It includes Human Resources (HRO), Information Technology (ITO), Enterprise Content Management (ECM) and Health care all of which are focused on supporting the back–office functions of the underserved middle-market enterprises. The company's Human Resources Outsourcing division offers BPO services to support customers and human resources administrative functions. This divisions services include a robust human resource information management system, HR Advocate, and professional services related to the installation, and ongoing administration and maintenance of the system. Its Enterprise Content Management division provides services related to data and records management of enterprises and governmental agencies. It offers imaging and data capture solutions, workflow and electronic forms solutions, document collaboration technology, eReview collaborative software products for Web-based conferencing and document sharing, and WebWorks software for project-based document and team management solutions, as well as provides process management/workflow solution for financial service providers. The company's Information Technology Outsourcing (ITO) division offers data centre outsourcing

services for middle-market enterprises, including enterprise-scale mainframe/server hosting, wide-area network management, and business recovery solutions. This division supports hosts, and manages information technology infrastructure components, networks, and applications; offers remote managed services to support other business application servers, networks, and related desktop environment; and provides hosting and managed application support services. Its Healthcare Administration, and Financial and Accounting Outsourcing division provides integrated solutions and services for health benefit administrators and health insurance claim processors.

One of the key challenges is to provide data entry/data validation services which are efficient and effective way of getting the source documents from different customers and accurately route the same to different operators for processing. Documents need to manage between the outsourcing company and the off-shore company. Multiple clients need to be managed by the company. Security of the documents has to be ensured so that there is no unauthorized access of the documents to other organizations. Quick turnaround times have to be managed. Appropriate process flow of the documents has to be present in the system to check the status of the documents at any point of time. BPO experts will evaluate your current outsourced operations for efficiency and effectiveness to prepare an outsourcing plan that is custom designed for your business.

10. ONLIE QUIZ SYSTEM

Online Quiz System has to be developed for conducting Preliminary stage of quiz as part of Department Technical symposium **PROTOCOL**.

The System developed should contain the following features

1. The number of participants should be of 30.

2. The duration for quiz is 30 minutes so a timer has to be kept which should show time duration.

3. The number of questions should be 40 chosen randomly from the database in 3 different areas namely i) Technical ii) logical reasoning and iii) General Knowledge.

4. The questions should be of objective type with multiple options. For each correct answer the participant will receive 1Point and for wrong answer .25Point will be deducted.

5. At the end of the quiz the users' score along with information whether he has been selected or not has to be displayed

6. Once a Student answered a question, he can't change the answer later.

2.IEEE SOFTWARE REQUIREMENTS SPECIFIICATION TEMPLATE

2.1 Introduction The introduction of the SRS should provide an overview of the entire SRS. It should contain the following
Subsections:
a) Purpose;
b) Scope;
c) Definitions, acronyms, and abbreviations;
d) References;
e) Overview.

2.1.1 Purpose

This subsection should
a) Delineate the purpose of the SRS;
b) Specify the intended audience for the SRS.

5.1.2 Scope

This subsection should
a) Identify the software product(s) to be produced by name (e.g., Host DBMS, Report Generator, etc.);
b) Explain what the software product(s) will, and, ifi necessary, will not do;
c) Describe the application of the software being specified, including relevant benefits, objectives, and goals;
d) Be consistent with similar statements in higher-level specifications (e.g., the system requirements specification), ifi they exist.

5.1.3 Definitions, acronyms, and abbreviations

This subsection should provide the definitions of all terms, acronyms, and abbreviations required to properly interpret the SRS. This information may be provided by reference to one or more appendixes in the SRS or by reference to other documents.

5.1.4 References

This subsection should
a) Provide a complete list of all documents referenced elsewhere in the SRS;
b) Identify each document by title, report number (if applicable), date, and publishing organization;
c) Specify the sources from which the references can be obtained.
This information may be provided by reference to an appendix or to another document.

5.1.5 Overview

This subsection should
a) Describe what the rest of the SRS contains;
b) Explain how the SRS is organized.

5.2 Overall description

This section of the SRS should describe the general factors that affect the product and its requirements. This section does not state specific requirements. Instead, it provides a background for those requirements, which are defined in detail in Section 3 of the SRS, and makes them easier to understand. This section usually consists of six subsections, as follows:
a) Product perspective;
b) Product functions;
c) User characteristics;
d) Constraints;
e) Assumptions and dependencies;
f) Apportioning of requirements.

5.2.1 Product perspective

This subsection of the SRS should put the product into perspective with other related products. If the product is independent and totally self-contained, it should be so stated here. If the SRS defines a product that is a component of a larger system, as frequently occurs, then this subsection should relate the requirements of that larger system to functionality of the software and should identify interfaces between that system and the software. A block diagram showing the major components of the larger system, interconnections, and external interfaces can be helpful.

This subsection should also describe how the software operates inside various constraints. For example,

these constraints could include

a) System interfaces;
b) User interfaces;
c) Hardware interfaces;
d) Software interfaces;
e) Communications interfaces;
f) Memory;
g) Operations;
h) Site adaptation requirements.

5.2.1.1 System interfaces

This should list each system interface and identify the functionality of the software to accomplish the system requirement and the interface description to match the system.

5.2.1.2 User interfaces

This should specify the following:

a) *The logical characteristics of each interface between the software product and its users.* This includes those configuration characteristics (e.g., required screen formats, page or window layouts, content of any reports or menus, or availability of programmable function keys) necessary to accomplish the software requirements.

b)*All the aspects of optimizing the interface with the person who must use the system.* This may simply comprise a list of do's and dont's on how the system will appear to the user. One example may be a requirement for the option of long or short error messages. Like all others, these requirements should be verifiable,

5.2.1.3 Hardware interfaces

This should specify the logical characteristics of each interface between the software product and the hardware components of the system. This includes configuration characteristics (number of ports, instruction sets, etc.). It also covers such matters as what devices are to be supported, how they are to be supported, and protocols. For example, terminal support may specify full-screen support as opposed to line-by-line support.

5.2.1.4 Software interfaces

This should specify the use of other required software products (e.g., a data management system, an operating system, or a mathematical package), and interfaces with other application systems (e.g., the linkage between an accounts receivable system and a general ledger system). For each required software product, the following should be provided:

Name;
Mnemonic;
Specification number;
Version number;
Source.

For each interface, the following should be provided:

Discussion of the purpose of the interfacing software as related to this software product.

Definition of the interface in terms of message content and format. It is not necessary to detail any well-documented interface, but a reference to the document defining the interface is required.

5.2.1.5 Communications interfaces

This should specify the various interfaces to communications such as local network protocols, etc.

5.2.1.6 Memory constraints

This should specify any applicable characteristics and limits on primary and secondary memory.

5.2.1.7 Operations

This should specify the normal and special operations required by the user such as

a) The various modes of operations in the user organization (e.g., user-initiated operations)

b) Periods of interactive operations and periods of unattended operations;

c) Data processing support functions;

d) Backup and recovery operations.

5.2.1.8 Site adaptation requirements

This should

a) Define the requirements for any data or initialization sequences that are specific to a given site, mission, or operational mode (e.g., grid values, safety limits, etc.);

b) Specify the site or mission-related features that should be modified to adapt the software to a particular installation.

5.2.2 Product functions

This subsection of the SRS should provide a summary of the major functions that the software will perform. For example, an SRS for an accounting program may use this part to address customer account maintenance, customer statement, and invoice preparation without mentioning the vast amount of detail that each of those functions requires.

Sometimes the function summary that is necessary for this part can be taken directly from the section of the higher-level specification (if one exists) that allocates particular functions to the software product. Note that for the sake of clarity

a) The functions should be organized in a way that makes the list of functions understandable to the customer or to anyone else reading the document for the First time.

b) Textual or graphical methods can be used to show the different functions and their relationships. Such a diagram is not intended to show a design of a product, but simply shows the logical relationships among variables.

5.2.3 User characteristics

This subsection of the SRS should describe those general characteristics of the intended users of the product including educational level, experience, and technical expertise. It should not be used to state specific requirements, but rather should provide the reasons why certain specific requirements are later specified

5.2.4 Constraints

This subsection of the SRS should provide a general description of any other items that will limit the developer options. These include

a) Regulatory policies;

b) Hardware limitations (e.g., signal timing requirements);

c) Interfaces to other applications;

d) Parallel operation;

e) Audit functions;

f) Control functions;

g) Higher-order language requirements;

h) Signal handshake protocols (e.g., XON-XOFF, ACK-NACK);
i) Reliability requirements;
j) Criticality of the application;
k) Safety and security considerations.

5.2.5 Assumptions and dependencies

This subsection of the SRS should list each of the factors that affect the requirements stated in the SRS. These factors are not design constraints on the software but are, rather, any changes to them that can affect the requirements in the SRS. For example, an assumption may be that a specific operating system will be available on the hardware designated for the software product. Ifi, in fact, the operating system is not available,
the SRS would then have to change accordingly.

5.2.6 Apportioning of requirements

This subsection of the SRS should identify requirements that may be delayed until future versions of the system.

5.3 Specific requirements

This section of the SRS should contain all of the software requirements to a level of detail sufficient to enable designers to design a system to satisfy those requirements, and testers to test that the system satisfies those requirements. Throughout this section, every stated requirement should be externally perceivable by users, operators, or other external systems. These requirements should include at a minimum a description of every input (stimulus) into the system, every output (response) from the system, and all functions performed by the system in response to an input or in support of an output. As this is often the largest and most important part of the SRS, the following principles apply:

a) Specific requirements should be stated in conformance with all the characteristics described in 4.3.
b) Specific requirements should be cross-referenced to earlier documents that relate.
c) All requirements should be uniquely identifiable.
d) Careful attention should be given to organizing the requirements to maximize readability.

Before examining specific ways of organizing the requirements it is helpful to understand the various items that comprise requirements as described in 5.3.1 through 5.3.7.

5.3.1 External interfaces

This should be a detailed description of all inputs into and outputs from the software system. It should complement the interface descriptions in 5.2 and should not repeat information there.It should include both content and format as follows:

a) Name of item;
b) Description of purpose;
c) Source of input or destination of output;
d) Valid range, accuracy, and/or tolerance;
e) Units of measure;
f) Timing;
g) Relationships to other inputs/outputs;
h) Screen formats/organization;
i) Window formats/organization;
j) Data formats;
k) Command formats;
l) End messages.

5.3.2 Functions

Functional requirements should define the fundamental actions that must take place in the software in accepting and processing the inputs and in processing and generating the outputs. These are generally listed as shall statements starting with The system shall
These include
a) Validity checks on the inputs
b) Exact sequence of operations
c) Responses to abnormal situations, including
1) Overflow
2) Communication facilities
3) Error handling and recovery
d) Effect of parameters
e) Relationship of outputs to inputs, including
1) Input/output sequences
2) Formulas for input to output conversion
It may be appropriate to partition the functional requirements into sub functions or sub processes. This does not imply that the software design will also be partitioned that way.

5.3.3 Performance requirements

This subsection should specify both the static and the dynamic numerical requirements placed on the software or on human interaction with the software as a whole. Static numerical requirements may include the following:
a) The number of terminals to be supported;
b) The number of simultaneous users to be supported;
c) Amount and type of information to be handled.
Static numerical requirements are sometimes identified under a separate section entitled Capacity.Dynamic numerical requirements may include, for example, the numbers of transactions and tasks and the amount of data to be processed within certain time periods for both normal and peak workload conditions.All of these requirements should be stated in measurable terms. For example,
95% of the transactions shall be processed in less than 1 s.
rather than, *An operator shall not have to wait for the transaction to complete.*
NOTE Numerical limits applied to one specific function are normally specified as part of the processing subparagraph description of that function.

5.3.4 Logical database requirements

This should specify the logical requirements for any information that is to be placed into a database. This may include the following:
a) Types of information used by various functions;
b) Frequency of use;
c) Accessing capabilities;
d) Data entities and their relationships;
e) Integrity constraints;
f) Data retention requirements.

5.3.5 Design constraints

This should specify design constraints that can be imposed by other standards, hardware limitations, etc.

5.3.5.1 Standards compliance

This subsection should specify the requirements derived from existing standards or regulations. They may
include the following:
a) Report format;

b) Data naming;
c) Accounting procedures;
d) Audit tracing.
For example, this could specify the requirement for software to trace processing activity. Such traces are needed for some applications to meet minimum regulatory or Financial standards. An audit trace requirement may, for example, state that all changes to a payroll database must be recorded in a trace Fle with before and after values.

5.3.6 Software system attributes

There are a number of attributes of software that can serve as requirements. It is important that required attributes be specified so that their achievement can be objectively verified. Subclasses 5.3.6.1 through 5.3.6.5 provide a partial list of examples.

5.3.6.1 Reliability

This should specify the factors required to establish the required reliability of the software system at time of delivery.

5.3.6.2 Availability

This should specify the factors required to guarantee a defined availability level for the entire system such as checkpoint, recovery, and restart.

5.3.6.3 Security

This should specify the factors that protect the software from accidental or malicious access, use, modification, destruction, or disclosure. Specific requirements in this area could include the need to
a) Utilize certain cryptography techniques;
b) Keep specific log or history data sets;
c) Assign certain functions to different modules;
d) Restrict communications between some areas of the program;
e) Check data integrity for critical variables.

5.3.6.4 Maintainability

This should specify attributes of software that relate to the ease of maintenance of the software itself. There may be some requirement for certain modularity, interfaces, complexity, etc. Requirements should not be placed here just because they are thought to be good design practices.

5.3.6.5 Portability

This should specify attributes of software that relate to the ease of porting the software to other host machines and/or operating systems. This may include the following:
a) Percentage of components with host-dependent code;
b) Percentage of code that is host dependent;
c) Use of a proven portable language;
d) Use of a particular compiler or language subset;
e) Use of a particular operating system.

5.3.7 Organizing the specific requirements

For anything but trivial systems the detailed requirements tend to be extensive. For this reason, it is recommended that careful consideration be given to organizing these in a manner optimal for understanding. There is no one optimal organization for all systems. Different classes of systems lend themselves to different organizations of requirements in Section 3 of the SRS. Some of these organizations are described in 5.3.7.1 Through 5.3.7.7.

5.3.7.1 System mode

Some systems behave quite differently depending on the mode of operation. For example, a control system may have different sets of functions depending on its mode: training, normal, or emergency. When organizing this section by mode, the outline in A.1 or A.2

should be used. The choice depends on whether interfaces and performance are dependent on mode.

5.3.7.2 User class

Some systems provide different sets of functions to different classes of users. For example, an elevator control system presents different capabilities to passengers, maintenance workers, and Fire Fighters. When organizing this section by user class, the outline in A.3 should be used.

5.3.7.3 Objects

Objects are real-world entities that have a counterpart within the system. For example, in a patient monitoring system, objects include patients, sensors, nurses, rooms, physicians, medicines, etc. Associated with each object is a set of attributes (of that object) and functions (performed by that object). These functions are also called services, methods, or processes. When organizing this section by object, the outline in A.4 should be used. Note that sets of objects may share attributes and services. These are grouped together as classes.

5.3.7.4 Feature

A feature is an externally desired service by the system that may require a sequence of inputs to effect the desired result. For example, in a telephone system, features include local call, call forwarding, and conference call. Each feature is generally described in a sequence of stimulus-response pairs. When organizing this section by feature, the outline in A.5 should be used.

5.3.7.5 Stimulus

Some systems can be best organized by describing their functions in terms of stimuli. For example, the functions of an automatic aircraft landing system may be organized into sections for loss of power, wind shear, sudden change in roll, vertical velocity excessive, etc. When organizing this section by stimulus, the outline in A.6 should be used.

5.3.7.6 Response

Some systems can be best organized by describing all the functions in support of the generation of a response. For example, the functions of a personnel system may be organized into sections corresponding to all functions associated with generating paychecks, all functions associated with generating a current list of employees, etc. The outline in A.6 (with all occurrences of stimulus replaced with response) should be used.

5.3.7.7 Functional hierarchy

When none of the above organizational schemes prove helpful, the overall functionality can be organized into a hierarchy of functions organized by common inputs, common outputs, or common internal data access. Data flow diagrams and data dictionaries can be used to show the relationships between and among the functions and data. When organizing this section by functional hierarchy, the outline in A.7 should be used.

5.3.8 Additional comments

Whenever a new SRS is contemplated, more than one of the organizational techniques given in 5.3.7.7 may be appropriate. In such cases, organize the specific requirements for multiple hierarchies tailored to the specific needs of the system under specification. For example, see A.8 for an organization combining user class and feature. Any additional requirements may be put in a separate section at the end of the SRS. There are many notations, methods, and automated support tools available to aid in the documentation of requirements. For the most part, their usefulness is a function of organization. For example, when organizing by mode, Finite state machines or state charts may prove helpful; when organizing by object, object-oriented analysis may prove helpful; when organizing by feature, stimulus-response sequences may prove helpful; and when

organizing by functional hierarchy, data flow diagrams and data dictionaries may prove helpful. In any of the outlines given in A.1 through A.8, those sections called Functional Requirement may be described in native language (e.g., English), in pseudo code, in a system definition language, or in four subsections titled: Introduction, Inputs, Processing, and Outputs.

5.4 Supporting information

The supporting information makes the SRS easier to use. It includes the following:

a) Table of contents;

b) Index;

c) Appendixes.

5.4.1 Table of contents and index

The table of contents and index are quite important and should follow general compositional practices.

5.4.2 Appendixes

The appendixes are not always considered part of the actual SRS and are not always necessary. They may include

a) Sample input/output formats, descriptions of cost analysis studies, or results of user surveys;

b) Supporting or background information that can help the readers of the SRS;

c) A description of the problems to be solved by the software;

d) Special packaging instructions for the code and the media to meet security, export, initial loading, or other requirements. When appendixes are included, the SRS should explicitly state whether or not the appendixes are to be considered part of the requirements.