SYSTEM ANALYSIS AND DESIGN

- Part 1: **System Analysis and Design**
- Part 2: **UML**

Part 1
System Analysis and Design

System Analysis and Design

Complete Introductory Tutorial

for Software Engineering

Table of contents

Chapter 1: Introduction to Systems

At the end of this lesson you would be able to know about system's concepts, characteristics and various types of Information systems. You would also be able to understand the system development process.

1.1. What are Systems?

To understand System Analysis and Design, one has to first understand what exactly are systems. In this session, we explore the meaning of system in accordance with analysts and designers. This session gives the reader basic concepts and terminology associated with the Systems.
It also gives the overview of various types of systems. In the broadest sense, a system is simply a set of components that interact to accomplish some purpose. They are all around us. For example, human body is a biological system. We experience physical sensations by means of a complex nervous system, a set of parts, including brain, spinal cord, nerves, and special sensitive cells under our skin, that work together to make us feel hot, cold, itchy, and so on.

Language is another example of a system. Each language has got its set of alphabets, vocabulary and grammar rules. Combination of all these make possible for one person to convey thoughts to other persons.

An organization may also be viewed as a system where all the employees interact with each other and also with the employer to make the organization a functional unit. The organization also interacts with their customers to make a complete business system.

In today's world most of the people study. To make this possible, there are education systems. Each education system contains educational institutes like preparatory schools, middle and high schools and colleges. It also contains governing bodies, people (teachers and students) and some commercial bodies, which fulfill the other needs like stationery, transportation, furniture, etc.

In our day to day life, we see many business systems. These businesses have varied objectives, which range from producing a notebook to producing aircraft. These systems have their information needs. It can be for maintaining the records for employee for their wages calculations, keeping track of their leave status, maintaining company's expenses, inquiries from customers in case the business provide some service, or for keeping track for some particular function. So maintaining data is an important and essential activity in any business. The overall data maintained constitutes what is known as Information system.

Information system is the means by which data flow from one person or department to another and can encompass everything from interoffice mail and telephone links to a computer system that generates periodic reports for various users. Information systems serve all the systems of a business, linking the different components in such a way that they effectively work towards the same purpose. These information systems are dealt in detail in section 1.1.3.

1.2. System Concepts

The term "System" is derived from the Greek word systema. It means an organized relationship among functioning units or components. We can define a System as a combination of resources or functional units working together to accomplish a given task.

The term "working together" in system definition is very important as all the components are interrelated and interdependent and can not exist independently. As the definition says, these components interact with each other to accomplish a given task, which is actually the objective of the system.

The components that comprise a system may be the various inputs required by the system, the outcomes or the outputs of the system, the resources required to make the system functional etc

1.3. System Components and Characteristics

A big system may be seen as a set of interacting smaller systems known as subsystems or functional units each of which has its defined tasks. All these work in coordination to achieve the overall objective of the system.

As discussed above, a system is a set of components working together to achieve some goal. The basic elements of the system may be listed as:
" Resources
" Procedures
" Data/Information
" Processes

Resources

Every system requires certain resources for the system to exist. Resources can be hardware, software or liveware. Hardware resources may include the computer, its peripherals, stationery etc. Software resources would include the programs running on these computers and the liveware would include the human beings required to operate the system and make it functional.

Thus these resources make an important component of any system. For instance, a Banking system cannot function without the required stationery like cheque books, pass books etc. such systems also need computers to maintain their data and trained staff to operate these computers and cater to the customer requirements.

Procedures

Every system functions under a set of rules that govern the system to accomplish the defined goal of the system. This set of rules defines the procedures for the system to Chapter 1-Introduction to Systems operate. For instance, the Banking systems have their predefined rules for providing interest at different rates for different types of accounts.

Data / Information

Every system has some predefined goal. For achieving the goal the system requires certain inputs, which are converted into the required output. The main objective of the System is to produce some useful output. Output is the outcome of processing. Output can be of any nature e.g. goods, services or information.

However, the Output must conform to the customer's expectations. Inputs are the elements that enter the system and produce Output. Input can be of various kinds, like material, information, etc.

Intermediate Data

Various processes process system's Inputs. Before it is transformed into Output, it goes through many intermediary transformations. Therefore, it is very important to identify the Intermediate Data. For example, in a college when students register for a new semester, the initial form submitted by student goes through many departments. Each department adds their validity checks on it.

Finally the form gets transformed and the student gets a slip that states whether the student has been registered for the requested subjects or not. It helps in building the System in a better way. Intermediate forms of data occur when there is a lot of processing on the input data. So, intermediate data should be handled as carefully as other data since the output depends upon it.

Processes

The systems have some processes that make use of the resources to achieve the set goal under the defined procedures. These processes are the operational element of the system.

For instance in a Banking System there are several processes that are carried out. Consider for example the processing of a cheque as a process. A cheque passes through several stages before it actually gets processed and converted. These are some of the processes of the Banking system. All these components together make a complete functional system.

Systems also exhibit certain features and characteristics, some of which are:
" Objective
" Standards
" Environment
" Feedback
" Boundaries and interfaces

Objective

Every system has a predefined goal or objective towards which it works. A system cannot exist without a defined objective. For example an organization would have an objective of earning maximum possible revenues, for which each department and each individual has to work in coordination.

Standards

It is the acceptable level of performance for any system. Systems should be designed to meet standards. Standards can be business specific or organization specific.

For example take a sorting problem. There are various sorting algorithms. But each has its own complexity. So such algorithm should be used that gives most optimum efficiency. So there should be a standard or rule to use a particular algorithm. It should be seen whether that algorithm is implemented in the system.

Environment

Every system whether it is natural or man made co-exists with an environment. It is very important for a system to adapt itself to its environment. Also, for a system to exist it should change according to the changing environment. For example, we humans live in a particular environment. As we move to other places, there are changes in the surroundings but our body gradually adapts to the new environment. If it were not the case, then it would have been very difficult for human to survive for so many thousand years.

Another example can be Y2K problem for computer systems. Those systems, which are not Y2K compliant, will not be able to work properly after year 2000. For computer systems to survive it is important these systems are made Y2K compliant or Y2K ready.

Feed Back

Feedback is an important element of systems. The output of a system needs to be observed and feedback from the output taken so as to improve the system and make it achieve the laid standards. In fig 1.1, it is shown that a system takes input. It then transforms it into output. Also some feedback can come from customer (regarding quality) or it can be some intermediate data (the output of one process and input for the other) that is required to produce final output

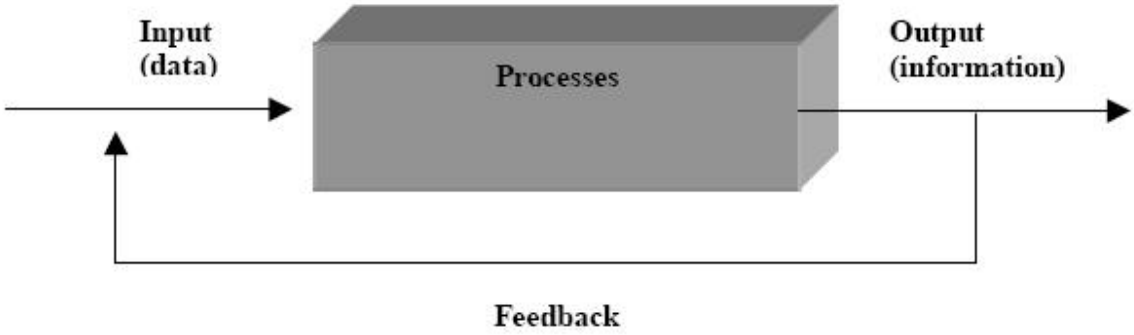


Fig 1.1
A system converts input data into output information

Boundaries and Interfaces

Every system has defined boundaries within which it operates. Beyond these limits the system has to interact with the other systems. For instance, Personnel system in an organization has its work domain with defined procedures. If the financial details of an employee are required, the system has to interact with the Accounting system to get the required details.

Interfaces are another important element through which the system interacts with the outside world. System interacts with other systems through its interfaces. Users of the systems also interact with it through interfaces. Therefore, these should be customized to the user needs. These should be as user friendly as possible.

1.4. Classifications of System

From previous section we have a firm knowledge of various system components and its characteristics. There are various types of system. To have a good understanding of these systems, these can be categorized in many ways. Some of the categories are open or closed, physical or abstract and natural or man made information systems, which are explained next.

Classification of systems can be done in many ways.

Physical or Abstract System

Physical systems are tangible entities that we can feel and touch. These may be static or dynamic in nature. For example, take a computer center. Desks and chairs are the static parts, which assist in the working of the center. Static parts don't change. The dynamic systems are constantly changing. Computer systems are dynamic system. Programs, data, and applications can change according to the user's needs.

Abstract systems are conceptual. These are not physical entities. They may be formulas, representation or model of a real system.

Open Closed System

Systems interact with their environment to achieve their targets. Things that are not part of the system are environmental elements for the system. Depending upon the interaction with the environment, systems can be divided into two categories, open and closed.

Open systems: Systems that interact with their environment. Practically most of the systems are open systems. An open system has many interfaces with its environment. It can also adapt to changing environmental conditions. It can receive inputs from, and delivers output to the outside of system. An information system is an example of this category.

Closed systems: Systems that don't interact with their environment. Closed systems exist in concept only.

Man made Information System

The main purpose of information systems is to manage data for a particular organization. Maintaining files, producing information and reports are few functions. An information system produces customized information depending upon the needs of the organization. These are usually formal, informal, and computer based.

Formal Information Systems: It deals with the flow of information from top management to lower management. Information flows in the form of memos, instructions, etc. But feedback can be given from lower authorities to top management.

Informal Information Systems: Informal systems are employee based. These are made to solve the day to day work related problems. Computer-Based Information Systems: This class of systems depends on the use of computer for managing business applications. These systems are discussed in detail in the next section.

1.5. Information System

In the previous section we studied about various classification of systems. Since in business we mainly deal with information systems we'll further explore these systems. We will be talking about different types of information systems prevalent in the industry.

Information system deals with data of the organizations. The purposes of Information system are to process input, maintain data, produce reports, handle queries, handle on line transactions, generate reports, and other output. These maintain huge databases, handle hundreds of queries etc. The transformation of data into information is primary function of information system.

These types of systems depend upon computers for performing their objectives. A computer based business system involves six interdependent elements. These are hardware (machines), software, people (programmers, managers or users), procedures, data, and information (processed data). All six elements interact to convert data into information. System analysis relies heavily upon computers to solve problems. For these types of systems, analyst should have a sound understanding of computer technologies.

In the following section, we explore three most important information systems namely, transaction processing system, management information system and decision support system, and examine how computers assist

1.6. Types of Information System

Information systems differ in their business needs. Also depending upon different levels in organization information systems differ. Three major information systems are

1. Transaction processing
2. Management information system
3. Decision support system

Figure 1.2 shows relation of information system to the levels of organization. The information needs are different at different organizational levels. Accordingly the information can be categorized as: strategic information, managerial information and operational information.

Strategic information is the information needed by top most management for decision making. For example the trends in revenues earned by the organization are required by the top management for setting the policies of the organization. This information is not required by the lower levels in the organization. The information systems that provide these kinds of information are known as Decision Support Systems.

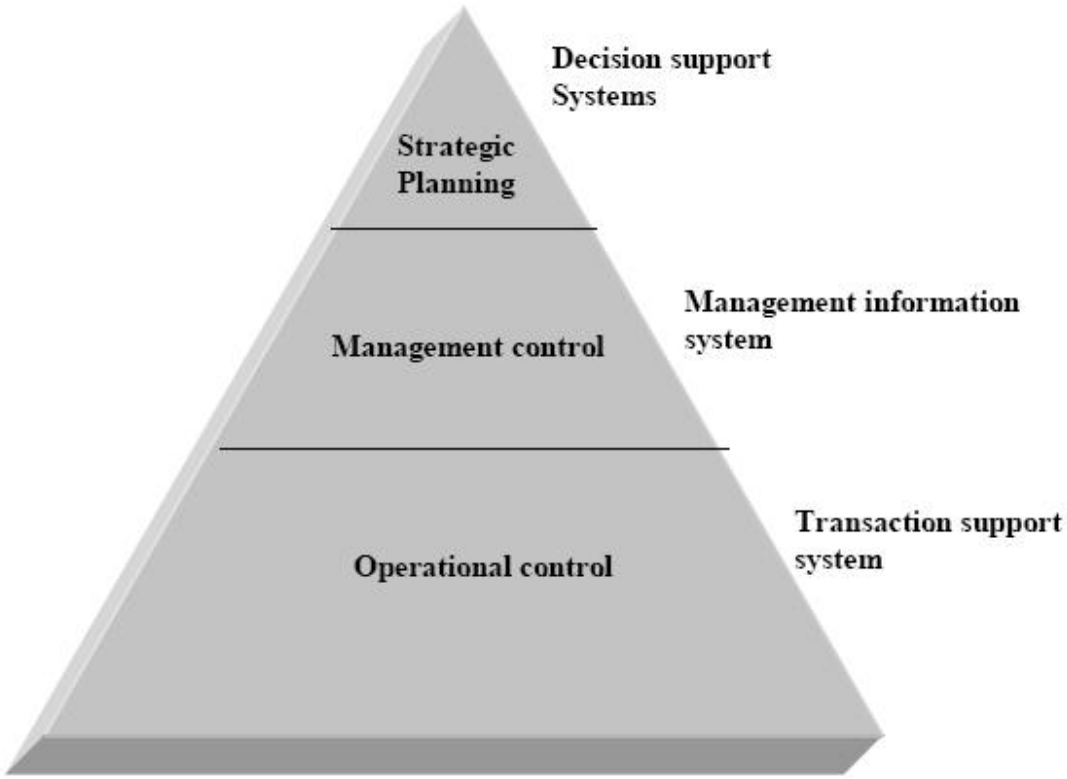


Fig 1.2
Relation of information systems to levels of organization

The second category of information required by the middle management is known as managerial information. The information required at this level is used for making short term decisions and plans for the organization. Information like sales analysis for the past quarter or yearly production details etc. fall under this category. Management information system(MIS) caters to such information needs of the organization. Due to its capabilities to fulfill the managerial information needs of the organization, Management Information Systems have become a necessity for all big organizations. And due to its vastness, most of the big organizations have separate MIS departments to look into the related issues and proper functioning of the system.

The third category of information is relating to the daily or short term information needs of the organization such as attendance records of the employees. This kind of information is required at the operational level for carrying out the day-to-day operational activities. Due to its capabilities to provide information for processing transaction of the organization, the information system is known as Transaction Processing System or Data Processing System. Some examples of information provided by such systems areprocessing of orders, posting of entries in bank, evaluating overdue purchaser orders etc.

Transaction Processing Systems

TPS processes business transaction of the organization. Transaction can be any activity of the organization. Transactions differ from organization to organization. For example, take a railway reservation system. Booking, canceling, etc are all transactions. Any query made to it is a transaction. However, there are some transactions, which are common to almost all organizations. Like employee new employee, maintaining their leave status, maintaining employees accounts, etc.

This provides high speed and accurate processing of record keeping of basic operational processes. These include calculation, storage and retrieval.

Transaction processing systems provide speed and accuracy, and can be programmed to follow routines functions of the organization.

Management Information Systems

These systems assist lower management in problem solving and making decisions. They use the results of transaction processing and some other information also. It is a set of information processing functions. It should handle queries as quickly as they arrive. An important element of MIS system is database.

A database is a non-redundant collection of interrelated data items that can be processed through application programs and available to many users.

Decision Support Systems

These systems assist higher management to make long term decisions. These type of systems handle unstructured or semi structured decisions. A decision is considered unstructured if there are no clear procedures for making the decision and if not all the factors to be considered in the decision can be readily identified in advance.

These are not of recurring nature. Some recur infrequently or occur only once. A decision support system must very flexible. The user should be able to produce customized reports by giving particular data and format specific to particular situations.

Summary of Information Systems

Catagories of Information System	Characteristices
Transaction Processing System	Substitutes computer-based processing for manual procedures. Deals with well-structured processes. Includes record keeping applications.
Management Information system	Provides input to be used in the managerial decision process. Deals with supporting well structured decision situations. Typical information requirements can be anticipated.
Decision Support Systems	Provides information to managers who must make judgements about particular situations. Supports decision-makers in situations that are not well structured.

1.7. Brief Introduction to System Analysis and Design

Till now we have studied what systems are, their components, classification of system, information system. Now we will look into the different aspects of how these systems are built.

Any change in the existing policies of an organization may require the existing information system to be restructured or complete development of a new information system. In case of an organization functioning manually and planning to computerize its functioning, the development of a new information system would be required.

The development of any information system can be put into two major phases: analysis and Design. During analysis phase the complete functioning of the system is understood and requirements are defined which lead to designing of a new system. Hence the development process of a system is also known as System Analysis and Design process. So let us now understand

1. What exactly System Analysis and Design is.?
2. Who is system analyst and what are his various responsibilities?
3. Users of the Systems?

1.8. What is System Analysis and Design?

System development can generally be thought of having two major components: systems analysis and systems design.In System Analysis more emphasis is given to understanding the details of an existing system or a proposed one and then deciding whether the proposed system is desirable or not and whether the existing system needs improvements. Thus, system analysis is the process of investigating a system, identifying problems, and using the information to recommend improvements to the system.

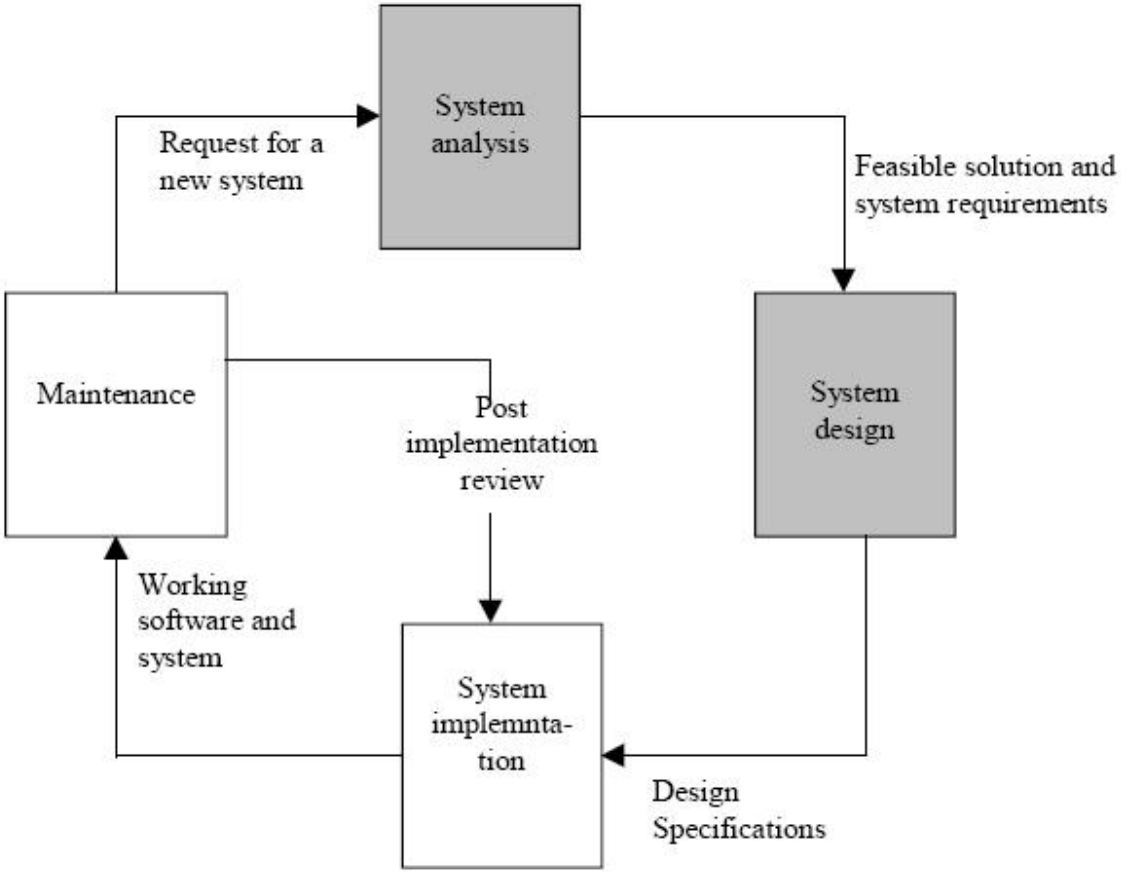


Fig: 1.3
Stages in building an improved system

Fig 1.3 shows the various stages involved in building an improved system.

System design is the process of planning a new business system or one to replace or complement an existing system.

Analysis specifies what the system should do. Design states how to accomplish the objective.

After the proposed system is analyzed and designed, the actual implementation of the system occurs. After implementation, working system is available and it requires timely maintenance. See fig 1.3.

1.9. Role of System Analyst

The system analyst is the person (or persons) who guides through the development of an information system . In performing these tasks the analyst must always match the information system objectives with the goals of the organization.

Role of System Analyst differs from organization to organization. Most common responsibilities of System Analyst are following

1) System analysis

It includes system's study in order to get facts about business activity. It is about getting information and determining requirements. Here the responsibility includes only requirement determination, not the design of the system.

2) System analysis and design:

Here apart from the analysis work, Analyst is also responsible for the designing of the new system/application.

3) Systems analysis, design, and programming:

Here Analyst is also required to perform as a programmer, where he actually writes the code to implement the design of the proposed application.

Due to the various responsibilities that a system analyst requires to handle, he has to be multifaceted person with varied skills required at various stages of the life cycle. In addition to the technical know-how of the information system development a system analyst should also have the following knowledge.

- Business knowledge: As the analyst might have to develop any kind of a business system, he should be familiar with the general functioning of all kind of businesses.
- Interpersonal skills: Such skills are required at various stages of development process for interacting with the users and extracting the requirements out of them
- Problem solving skills: A system analyst should have enough problem solving skills for defining the alternate solutions to the system and also for the problems occurring at the various stages of the development process

1.10. Who are the users of system?

The system end users of the system refer to the people who use computers to perform their jobs, like desktop operators. Further, end users can be divided into various categories.

- Very first users are the hands-on users. They actually interact with the system. They are the people who feed in the input data and get output data. Like person at the booking counter of a gas authority. This person actually sees the records and registers requests from various customers for gas cylinders.

- Other users are the indirect end users who do not interact with the systems hardware and software. However, these users benefit from the results of these systems. These types of users can be managers of organization using that system.

- There are third types of users who have management responsibilities for application systems. These oversee investment in the development or use of the system.

- Fourth types of users are senior managers. They are responsible for evaluating organization's exposure to risk from the systems failure.

Now we know what are systems and what is system analysis and design. So let us take a case in which we'll apply the concepts we have learned in the chapter. The case would be referred to as and where necessary throughout the book and in this process we will be developing the system required.

1.11. Case Study : Noida Library System

Noida Public Library is the biggest library in Noida. Currently it has about 300 members. A person who is 18 or above can become a member. There is a membership fee of Rs 400 for a year. There is a form to be filled in which person fills personal details. These forms are kept in store for maintaining members' records and knowing the membership period.

A member can issue a maximum of three books. He/she has three cards to issue books. Against each card a member can issue one book from library. Whenever a member wishes to issue a book and there are spare cards, then the book is issued. Otherwise that request is not entertained. Each book is to be returned on the specified due date. If a member fails to return a book on the specified date, a fine of Rs 2 per day after the due return date is charged. If in case a card gets lost then a duplicate card is issued. Accounts are maintained for the membership fees and money collected from the fines. There are two librarians for books return and issue transaction. Approximately 100 members come to library daily to issue and return books.

There are 5000 books available out of which 1000 books are for reference and can not be issued. Records for the books in the library are maintained. These records contain details about the publisher, author, subject, language, etc. There are suppliers that supply books to the library. Library maintains records of these suppliers.

Many reports are also produced. These reports are for details of the books available in the library, financial details, members' details, and supplier's details.

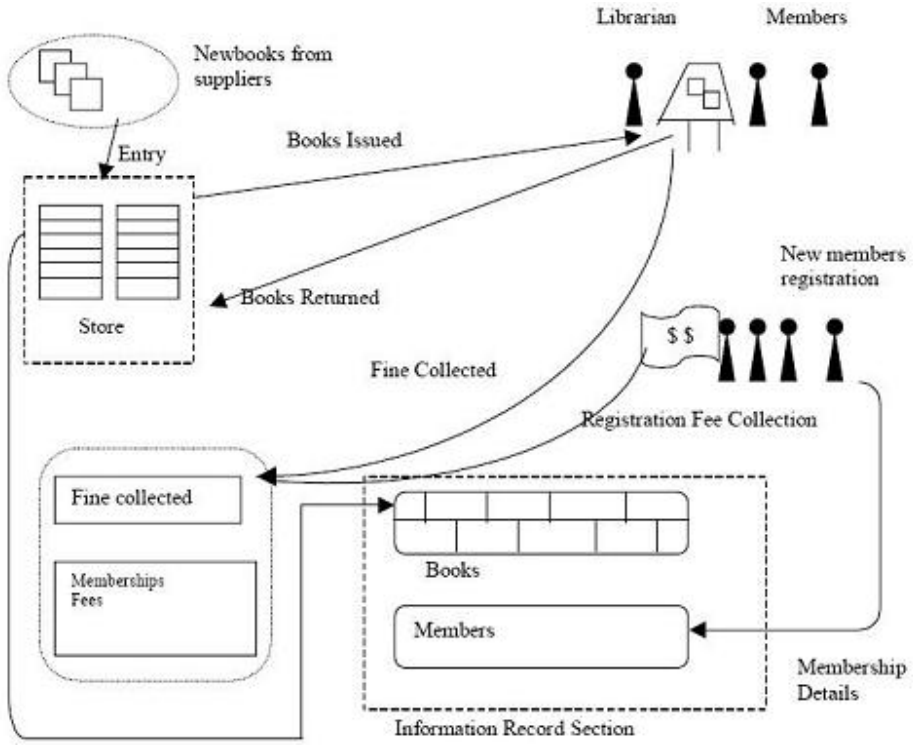
Currently all functions of the library are done manually. Even the records are maintained on papers. Now day by day members are increasing. Maintaining manual records is becoming difficult task. There are other problems also that the library staff is facing. Like in case of issue of duplicate cards to a member when member or library staff loses the card. It is very difficult to check the genuinity of the problem.

Sometimes the library staff needs to know about the status of a book as to whether it is issued or not. So to perform this kind of search is very difficult in a manual system.

Also management requires reports for books issued, books in the library, members, and accounts. Manually producing the reports is a cumbersome job when there are hundreds and thousands of records.

Management plans to expand the library, in terms of books, number of members and finally the revenue generated. It is observed that every month there are at least 50-100 requests for membership. For the last two months the library has not entertained requests for the new membership as it was difficult to manage the existing 250 members manually. With the expansion plans, the management of the library aims to increase its members at the rate of 75 per month. It also plans to increase the membership fees from 400 to 1000 for yearly and 500 for half year, in order to provide its members better services, which includes increase in number of books from 3 to 4.

Due to the problems faced by the library staff and its expansion plans, the management is planning to have a system that would first eradicate the needs of cards. A system to automate the functions of record keeping and report generation. And which could help in executing the different searches in a faster manner. The system to handle the financial details.



Applying the concepts studied in the chapter to the case study:

The first thing we studied is systems. In our case study Noida Public Library is our system. Every system is a set of some functional units that work together to achieve some objective. The main objective of library system is to provide books to its members without difficulty. Fig 1.4 depicts our library system pictorially.

Our system has many functional units. Books issue and return section, books record unit, members record unit, accounts, and report generation units are the different functional units of the library. Each functional unit has its own task. However, each of these work independently to achieve the overall objective of the library.

Later in the session, we talked about different components and characteristics of the systems. Data is an important component of any system. Here, data is pertaining to the details of members, books, accounts, and suppliers. Since people can interact with the system this system is an open system. The system is mainly concerned with the management of data it is an information system.

If this system were to be automated as conceived by the management, then role of the system analyst would be to study the system, its workings, and its existing problems. Also the analyst needs to provide a solution to the existing problem.

Now that the management has decided for an automated system the analyst would perform the above tasks. As the analyst did the study of the system, the following problems were identified

- Maintaining membership cards
- Producing reports due to large amount of data
- Maintaining accounts
- Keeping records for books in library and its members
- Performing searches

Now that the analyst has studied the system and identified the problems, it is the responsibility of the analyst to provide a solution system to the management of the library.

1.12. Review questions

1. Define the term ‘System’.

2. What are the various elements of system?

3. Identify two systems in your surroundings.

4. What is system analysis and design?

5. What are the roles of system analyst?

6. Make a list of traits that a system analyst should have.

7. Will the responsibility of a system analyst vary according to:

   (a)Organization size(for example small or large business)?
   (b) Type of organization(business, government agency, non-profit organization)?

8. Differentiate between

   a)Open and closed system
   b) Physical and abstract

9. Main aim of an information system is to process _____.

10. Transaction processing, _____ , and decision support system are three types of information system.

11. State true or false

   a) Decision support system is for middle level management.
   b) Closed systems don't interact with their environment.
   c) Transaction processing system handles day-to-day operations of the organization.
   d) Management information system deals with strategic information of organization.
   e) Problem solving and interpersonal skills are desirable for system analyst

1.13. Introduction to Systems - Summary

>> A system is a set of interdependent components, organized in a planned manner to achieve certain objectives.

>> System interacts with their environment through receiving inputs and producing outputs.

>> Systems can be decomposed into smaller units called subsystems.

>> Systems falls into three categories

>> Physical or Abstract systems

>> Open or closed system depending upon their interaction with environment.

>> Man-made such as information systems.

>> Three levels of information in organization require a special type of information.

>> Strategic information relates to long-term planning policies and upper management.

>> Managerial information helps middle management and department heads in policy implementation and control.

>> Operational information is daily information needed to operate the business.

>> Information systems are of many types. Management Information, transaction processing, and decision support systems are all information systems.

>> Transaction processing system assist in processing day to day activities of the organization

>> Management information systems are decisions oriented. These use transaction data and other information that is developed internally or outside the organization.

>> Decision support systems are built for assisting managers who are responsible for making decisions.

>> System analysis and design refers to the application of systems approach to problem solving.

Chapter 2: System Development Life Cycle models

At the end of this lesson you would be able to know the various stages involved in a system life cycle and you would be able to understand the various methodologies available for system development.

2.1. Introduction to Software life cycle models

In the last session we studied about system's concepts and different types of systems. Also, a brief introduction to system analysis and design process was presented. Let us now look into various kinds of system development methodologies.

Nearly three decades ago the operations in an organization used to be limited and so it was possible to maintain them using manual procedures. But with the growing operations of organizations, the need to automate the various activities increased, since for manual procedures it was becoming very difficult and cumbersome. Like maintaining records for a thousand plus employees company on papers is definitely a cumbersome job. So, at that time more and more companies started going for automation.

Since there were a lot of organizations, which were opting for automation, it was felt that some standard and structural procedure or methodology be introduced in the industry so that the transition from manual to automated system became easy. The concept of system life cycle came into existence then. Life cycle model emphasized on the need to follow some structured approach towards building new or improved system. There were many models suggested. A waterfall model was among the very first models that came into existence. Later on many other models like prototype, rapid application development model, etc were also introduced.

System development begins with the recognition of user needs. Then there is a preliminary investigation stage. It includes evaluation of present system, information gathering, feasibility study, and request approval. Feasibility study includes technical, economic, legal and operational feasibility. In economic feasibility cost-benefit analysis is done. After that, there are detailed design, implementation, testing and maintenance stages.

In this session, we'll be learning about various stages that make system's life cycle. In addition, different life cycles models will be discussed. These include Waterfall model, Prototype model, Object-Oriented Model, and Dynamic Systems Development Method (DSDM).

2.2. Activities involved in any Life cycle Model

Following activities is the part of any life cycle model. The sequence may be not being exactly same. However, each of these is necessarily covered in the life cycle. The following section describes each of these stages.

1. *Preliminary Investigation*
2. *Determination of System's requirements: Analysis phase*
3. *Design of System*
4. *Development of Software*
5. *System Testing*
6. *Implementation and Maintenance*

2.2.1. Preliminary Investigation

Fig 2.1 shows different stages in the system's life cycle. It initiates with a project request. First stage is the preliminary analysis. The main aim of preliminary analysis is to identify the problem. First, need for the new or the enhanced system is established. Only after the recognition of need, for the proposed system is done then further analysis is possible.

Suppose in an office all leave-applications are processed manually. Now this company is recruiting many new people every year. So the number of employee in the company has increased. So manual processing of leave application is becoming very difficult. So the management is considering the option of automating the leave processing system. If this is the case, then the system analyst would need to investigate the existing system, find the limitations present, and finally evaluate whether automating the system would help the organization.

Once the initial investigation is done and the need for new or improved system is established, all possible alternate solutions are chalked out. All these systems are known as "candidate systems". All the candidate systems are then weighed and the best alternative of all these is selected as the solution system, which is termed as the "proposed system". The proposed system is evaluated for its feasibility. Feasibility for a system means whether it is practical and beneficial to build that system.

Feasibility is evaluated from developer and customer's point of view. Developer sees whether they have the required technology or manpower to build the new system. Is building the new system really going to benefit the customer. Does the customer have the required money to build that type of a system? All these issues are covered in the feasibility study of the system. The feasibility of the system is evaluated on the three main issues: technical, economical, and operational. Another issue in this regard is the legal feasibility of the project.

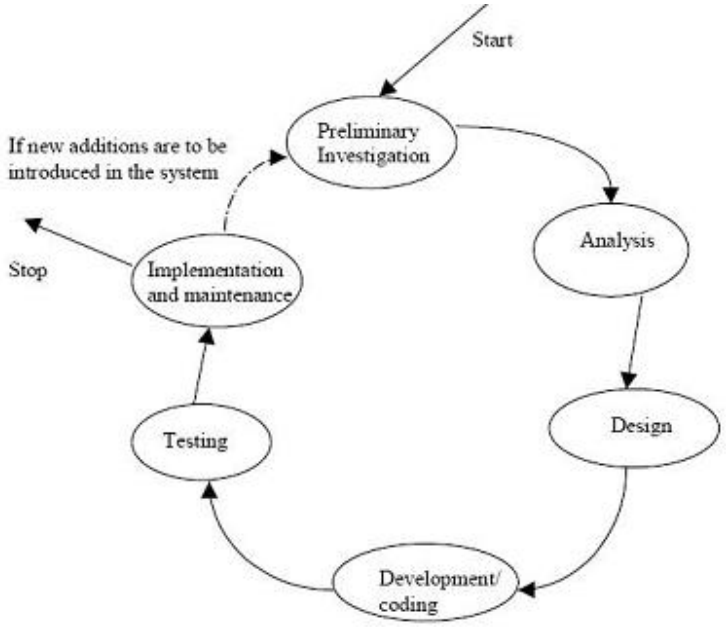


Fig. 2.1
Various stages in System Development

1. **Technical feasibility**: Can the development of the proposed system be done with current equipment, existing software technology, and available personnel? Does it require new technology?

2. **Economic feasibility**: Are there sufficient benefits in creating the system to make the costs acceptable? An important outcome of the economic feasibility study is the cost benefit analysis.

3. **Legal feasibility**: It checks if there are any legal hassle in developing the system.

4. **Operational feasibility**: Will the system be used if it is developed and implemented? Will there be resistance from users that will undermine the possible application benefits?

The result of the feasibility study is a formal document, a report detailing the nature and scope of the proposed solution. It consists of the following:

- Statement of the problem
- Details of findings
- Findings and recommendations in concise(súc tích) form

Once the feasibility study is done then the project is approved or disapproved according to the results of the study. If the project seems feasible and desirable then the project is finally approved otherwise no further work is done on it.

2.2.2. Determination of System's requirements: Analysis phase

After preliminary investigation, analysis phase begins. Analysis is a detailed study of the various operations performed by the system, relationships among the various sub-systems or functional units and finally the relationships outside the system. Major questions under consideration during analysis are:

1. What is being done?

2. How is it being done?

3. Does a problem exist?

4. If a problem exists, how severe is it?

5. How frequently does it occur?

6. What is the underlining cause for the problem?

Study is conducted to find user's information requirements. Proper functioning of the current system is also studied. Many tools are used during analysis. Data flow diagrams, on-site observations, and questionnaires are some examples.

Once the analysis is completed, the analyst has a firm knowledge of what is to be done.

2.2.3. Design of System

After the system has been analyzed by the analyst, the design stage of system life cycle begins. In design phase, the structure or design for the proposed system is finalized. Structure of files, databases, input, output, processes, and screens(interfaces) are decided. After design is finalized, it is clearly documented in what is called Design Document.

This design document contains various graphical representations of reports, user interaction screens, etc. This design document is referred by developers during development of system.

Correct designing of the system is very crucial. If we have a wrong design we won't be able to a get correct desired system. From various studies it is observed that nearly 50% errors are made during design phase in any software development while 33% errors are pertaining to program logic and only 17% are syntactical errors. Also cost of fixing errors is maximum for design phase. Cost of fixing errors for design phase nearly amounts to 80% while it is only 20% for logic and programming.

Cost of change increases when errors are discovered in later stages for system development. Mostly design errors are discovered in development, testing, testing, or maintenance phase. So the cost of change due to changes in design (which might happen due to some error or improper design. is very high. So design process should be handled very carefully.

2.2.4. Development of Software

In this phase, the actual development of the system takes place. That is, design representations are translated into actual programs. Software developers may install (or modify and then install) purchased software or they may write new, custom-designed programs.

Programmers are also responsible for documenting the program, providing an explanation of how procedures are coded. Documentation is essential to test the program and carry on maintenance once the application has been installed. It is also helpful to user in knowing the system well.

2.2.5. System Testing

After a system has been developed, it is very important to check if it fulfills the customer requirements. For this purpose, testing of the system is done. For testing the systems, various test cases are prepared. A test case is a certain made up situation on which system is exposed so as to find the behavior of system in that type of real situation. These test cases require data. The data can be also made up artificial data or the real data provided by the user.

There are various types of tests which are used to test the system. These include unit, integration, and acceptance testing.

The smallest unit of software design is module. In unit testing these modules are tested. Since the modules are very small even individual programmer can test them. Once the individual modules are tested, these are integrated to build the complete system. But testing individual module doesn't guarantee if the system will work properly when these units are integrated.

Acceptance testing ensures that the system meets all the requirements. If it fulfils the needs then the system is accepted by the customer and put into use.

2.2.6. Implementation and Maintenance

Implementation of system means putting up system on user's site. Like any system, there is an aging process. Therefore, the system requires periodic maintenance.

Maintenance can be for software or hardware. User priorities, changes in organizational requirements, or environmental factors call for system enhancements. This is very crucial for the system's life.

2.3. Different Life Cycles Models

We have studied the various stages that are involved in the development of systems. There are system development models, which follow these stages. There is sequential traditional model also called waterfall model. Along with this there are many other approaches to system development. There are

<> Iterative Prototype Model

<> Dynamic Systems Development Model (DSDM)

<> Spiral Model, Incremental Model

<> Object-Oriented Model.

In the following section, we'll be discussing waterfall model, prototype model, DSDM and object-oriented model

2.4. Traditional / Waterfall Software Development Life Cycle Model

Waterfall model is a systematic and sequential approach towards software development. This model follows the stages, which we have studied in section 2.1.2. Each stage begins or originates only after the previous stage has finished. There are different levels, corresponding to which each stage starts. Probably due to different leveling of stages, this model is called Waterfall model.

In this model, the phases are organized in a linear order. The first stage deals with the system study. The rest of the stages are analysis, coding, testing and maintenance. Next stage can begin only when the previous stage is over. We'll be taking our case study 'Library Management System' initiated in lesson 1, to illustrate this development model

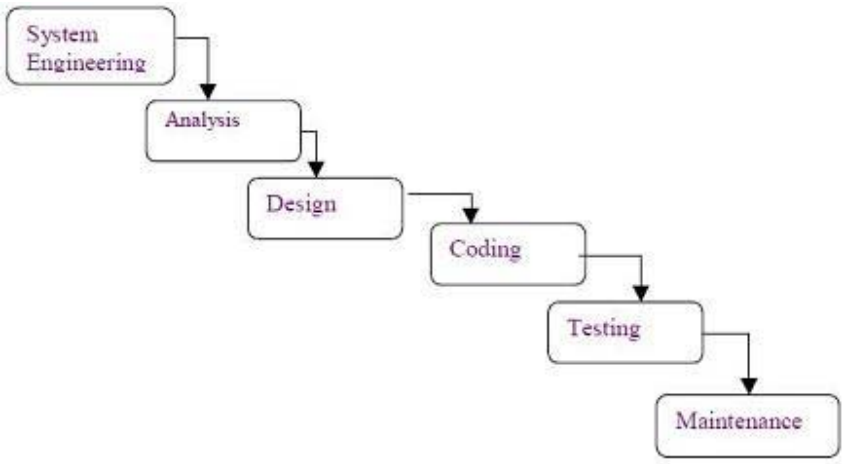


Fig2.2:
Waterfall Model

2.5. CASE STUDY: Library Management System

Referring to the case of 'Library management system' introduced in the first chapter, let us develop this system on Waterfall approach.

In Waterfall model, first stage is preliminary analysis, which deals with the study of the current system, finding problems and establishing whether the new system will benefit the organization. In this library presently all transactions are done manually. Each member is allowed a certain number of books to borrow. He/she has cards to borrow books. Against each card a member can borrow one book from library.

Whenever a member wishes to issue a book and there are spare cards, then the book is issued. Otherwise that request is not entertained. The numbers of members are increasing day by day. It is becoming a problem to manage the cards of members. So an automated system is required that can maintain the details about the books borrowed by different system. As we have studied the system, we now know about the existing problems. And we can positively establish the need for the automated record keeping system. After this preliminary investigation, second phase of Waterfall model, i.e., analysis phase begins.

In analysis, a detail study of system is performed. Each operation is looked into more details. From the preliminary analysis we know that the members are increasing and managing their cards is now difficult task for the library staff.

Let us find out why increasing members is a problem. There might be a case when a card gets misplaced either by library staff or by member itself. If this is indeed the case then a duplicate card is made. But a member can lie about it and can make a duplicate card even if the card is not lost. In that case that particular member is having more than maximum allowed cards. There is no means to identify such members.

Presently there is a need to have a system that can record the details about the books issued to members. Card system needs to be discarded. To solve this situation our library needs a computer application that has a database that contains details corresponding to each book issued. It should also have facilities to check if the number of books issued to a particular member doesn't exceed the maximum books allowed to him/her. Now we know what type of system is required, we can move to the next stage of waterfall model that is design stage. From analysis phase it is clear that we need a database to implement our new system.

In design stage it is decided as to which type of database will be used in system. Secondly we'll identify what data should this database store and in what format. After that various operations like issuing and returning books are finalized. Various checks like number of books issued not to exceed the maximum number are finalized. Various interfaces that are to be used for input are decided.

Once all these details are finalized, these are properly documented .These documents are used in building the system during the development stage, which follows the design stage. Using the design details the new system is built. Only things that are identified during the design stage are incorporated into the system.

There is no deviation from the design specifications. Suppose in the design phase the database to be used in the system is decided to be Oracle RDBMS then during development of the system only Oracle database is used. Similarly each design specification is taken care of and whole new system is build.

Now we have developed our new system. But before it is implemented at our user's site it needs to be tested for accuracy. So the system is tested to determine if it is performing correctly according to the requirements identified during the analysis phase. Each function of the system is tested. For example, we have issue book and return book functions. In return function it is checked if it is incrementing the count of variable that signifies how many books that member can issue more. In the issue function, it should be tested that this variable is decreased by one unit. Similarly whole of the system is tested.

After testing the system, it is implemented at the user's site. Now we have tested our new system for the library, it is implemented at the user's site i.e., library in our case.

After implementation, the systems require time to time maintenance. Maintenance can be for the software and hardware. Suppose two years after the implementation of the system, there occurs an alarming increase in number of members of the library. A situation can arise when all the memory for keeping the details have been exhausted then there will be a need to increase the memory of the system. Similarly the speed of processing requests might slow down. Then there will be a need for a faster processor. All these issues are maintenance issues.

So maintenance, though it is final stage of the system development, is equally important. In this section we explored how we can solve our case study problem 'Library management system' using traditional Waterfall model. Using Waterfall model is not mandatory. It is possible to follow other approaches to solve the same problem. It depends upon the developer, which approach to follow. In the next section, we illustrate case study using another approach i.e. Prototyping. This is an iterative approach towards system development.

2.6. Alternative Development Models

In this section, we will discuss in brief, some of the other models prevalent in the industry for systems development. Prototype, Dynamic System Development and Object Oriented methods are discussed.

a. Prototyping

b. Object Oriented Methodology

c. Dynamic System Development Method

2.6.1. Prototyping Software Life Cycle Model

We have seen the sequential approach towards system development using Waterfall model, now we'll look at an iterative approach using Prototyping Model.

Before we go into details of this approach and we apply this to our case, we need to understand what exactly is a prototype. A prototype is a working system that is developed to test ideas about the new system. And prototyping is a process of building a model of the system to be developed.

This approach is used when it is difficult to know all the requirements in the beginning of the project. Such situations arise in the following cases:

***** The user has not provided all the requirements.
***** No other system like the proposed system was built earlier.

This model can be also used in a situation when the customer wants a quick delivery. May be not the entire system but a part of it. The complete system is based on an iterative model where the customer keeps specifying the requirements according to the model made available and the changes are incorporated to build a better model.

CASE STUDY : Library management system

Now we'll look at our case study 'Library management system' using Prototype model. Fig 2.3 shows various stages involved in the Prototype model.

In our case the customer i.e., our library management are not technical people. All they know is that they need a system for their library that can automate the functions of the library. Only functions they have identified at this stage are issue and return of books. They want the new system as early as possible.

In this scenario, Prototyping approach can be used for system development.

First stage is requirement gathering. For our case the only requirements that the library staff has identified are having an automated system for books return and issue functions. So the analyst starts working on the presently identified requirements of system.

After requirements gathering, design stage begins. This is of very short period. Design is often very quick. Design considerations are same as discussed earlier in section 2.2.1. Database, procedures, checks, screen layouts, etc are decided. These are properly documented.

After design is finalized a working prototype or model is built. It is built to show the functionality of the system to the user before the development of the actual product. Once you have a prototype, it is shown to the customer. The customer verifies the prototype. In case there are some suggestions from customers then again that functionality is added to the prototype and again it is evaluated by customer. This cycle gets repeated till the time the customer is fully satisfied with the prototype. Then the actual product is built. The same prototype can be used for the final product or a new product can be built based on the prototype.

In our case, in the first customer evaluation stage suppose our customer identifies that they require one more function that could handle fine amounts collected in lieu of late return of books. Then our designer will again go to the design stage and incorporate this function in the prototype. This cycle will be repeated till the customer is fully satisfied with the prototype.

The prototyping model of System Development is thus very different from the traditional Waterfall Approach. In this model, the phases of development are not sequential, but they are iterative , where any changes or additions suggested by the end user are incorporated by modifying the prototype model and again giving it to the user.

Advantages of Prototyping may be listed as:

***** Users are actively involved in the development

***** It provides a better system to users, as users have natural tendency to change their mind in specifying requirements and this method of developing systems supports this user tendency.

***** Since in this methodology a working model of the system is provided, the users get a better understanding of the system being developed.

***** Errors can be detected much earlier as the system is mode side by side.

***** Quicker user feedback is available leading to better solutions.

Disadvantages

***** Leads to implementing and then repairing way of building systems.

***** Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans.

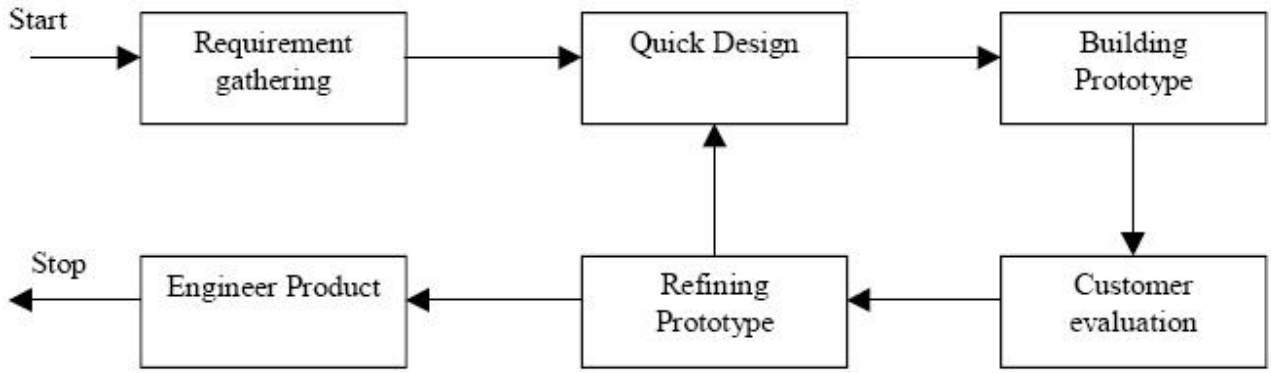


Fig. 2.3
Prototype Model

2.6.2. Object Oriented Methodology

We live in a world of objects. These objects exist in nature, in man-made entities, in business, and in the products that we use. They can be categorized, described, organized, combined, manipulated and created. Therefore, an object-oriented view has come into picture for creation of computer software. An object-oriented approach to the development of software was proposed in late 1960s.

Object-Oriented development requires that object-oriented techniques be used during the analysis, and implementation of the system. This methodology asks the analyst to determine what the objects of the system are, how they behave over time or in response to events, and what responsibilities and relationships an object has to other objects. Object-oriented analysis has the analyst look at all the objects in a system, their commonalties, difference, and how the system needs to manipulate the objects.

Object Oriented Process

The Object Oriented Methodology of Building Systems takes the objects as the basis. For this, first the system to be developed is observed and analyzed and the requirements are defined as in any other method of system development. Once this is done, the objects in the required system are identified. For example in case of a Banking System, a customer is an object, a chequebook is an object, and even an account is an object.

In simple terms, Object Modeling is based on identifying the objects in a system and their interrelationships. Once this is done, the coding of the system is done. Object Modeling is somewhat similar to the traditional approach of system designing, in that it also follows a sequential process of system designing but with a different approach. The basic steps of system designing using Object Modeling may be listed as:

1. System Analysis
2. System Design
3. Object Design
4. Implementation

2.6.2.1. System Analysis

As in any other system development model, system analysis is the first phase of development in case of Object Modeling too. In this phase, the developer interacts with the user of the system to find out the user requirements and analyses the system to understand the functioning.

Based on this system study, the analyst prepares a model of the desired system. This model is purely based on what the system is required to do. At this stage the implementation details are not taken care of. Only the model of the system is prepared based on the idea that the system is made up of a set of interacting objects. The important elements of the system are emphasized.

2.6.2.2. System Design

System Design is the next development stage where the overall architecture of the desired system is decided. The system is organized as a set of subsystems interacting with each other. While designing the system as a set of interacting subsystems, the analyst takes care of specifications as observed in system analysis as well as what is required out of the new system by the end user.

As the basic philosophy of Object-Oriented method of system design is to perceive the system as a set of interacting objects, a bigger system may also be seen as a set of interacting smaller subsystems that in turn are composed of a set of interacting objects. While designing the system, the stress lies on the objects comprising the system and not on the processes being carried out in the system as in the case of traditional Waterfall Model where the processes form the important part of the system.

2.6.2.3. Object Design

In this phase, the details of the system analysis and system design are implemented. The Objects identified in the system design phase are designed. Here the implementation of these objects is decided as the data structures get defined and also the interrelationships between the objects are defined.

Let us here deviate slightly from the design process and understand first a few important terms used in the Object-Oriented Modeling.

As already discussed, Object Oriented Philosophy is very much similar to real world and hence is gaining popularity as the systems here are seen as a set of interacting objects as in the real world. To implement this concept, the process-based structural programming is not used; instead objects are created using data structures. Just as every programming language provides various data types and

various variables of that type can be created, similarly, in case of objects certain data types are predefined.

For example, we can define a data type called pen and then create and use several objects of this data type. This
concept is known as creating a class.

Class: A class is a collection of similar objects. It is a template where certain basic characteristics of a set of objects are defined. The class defines the basic attributes and the operations of the objects of that type. Defining a class does not define any object, but it only creates a template. For objects to be actually created instances of the class are created as per the requirement of the case.

Abstraction: Classes are built on the basis of abstraction, where a set of similar objects are observed and their common characteristics are listed. Of all these, the characteristics of concern to the system under observation are picked up and the class definition is made. The attributes of no concern to the system are left out. This is known as abstraction.

The abstraction of an object varies according to its application. For instance, while defining a pen class for a stationery shop, the attributes of concern might be the pen color, ink color, pen type etc., whereas a pen class for a manufacturing firm would be containing the other dimensions of the pen like its diameter, its shape and size etc.

Inheritance: Inheritance is another important concept in this regard. This concept is used to apply the idea of reusability of the objects. A new type of class can be defined using a similar existing class with a few new features. For instance, a class vehicle can be defined with the basic functionality of any vehicle and a new class called car can be derived out of it with a few modifications. This would save the developers time and effort as the classes already existing are reused without much change.

Coming back to our development process, in the Object Designing phase of the Development process, the designer decides onto the classes in the system based on these concepts. The designer also decides on whether the classes need to be created from scratch or any existing classes can be used as it is or new classes can be inherited from them.

2.6.2.4. Implementation

During this phase, the class objects and the interrelationships of these classes are translated and actually coded using the programming language decided upon. The databases are made and the complete system is given a functional shape.

The complete OO methodology revolves around the objects identified in the system. When observed closely, every object exhibits some characteristics and behavior. The objects recognize and respond to certain events. For example, considering a Window on the screen as an object, the size of the window gets changed when resize button of the window is clicked.

Here the clicking of the button is an event to which the window responds by changing its state from the old size to the new size. While developing systems based on this approach, the analyst makes use of certain models to analyze and depict these objects. The methodology supports and uses three basic Models:

" **Object Model -** This model describes the objects in a system and their interrelationships. This model observes all the objects as static and does not pay any attention to their dynamic nature.

" **Dynamic Model -** This model depicts the dynamic aspects of the system. It portrays the changes occurring in the states of various objects with the events that might occur in the system.

" **Functional Model -** This model basically describes the data transformations of the system. This describes the flow of data and the changes that occur to the data throughout the system.

While the Object Model is most important of all as it describes the basic element of the system, the objects, all the three models together describe the complete functional system.

As compared to the conventional system development techniques, OO modeling provides many benefits. Among other benefits, there are all the benefits of using the Object Orientation. Some of these are:

" Reusability - The classes once defined can easily be used by other applications. This is achieved by defining classes and putting them into a library of classes where all the classes are maintained for future use. Whenever a new class is needed the programmer looks into the library of classes and if it is available, it can be picked up directly from there.

" Inheritance - The concept of inheritance helps the programmer use the existing code in another way, where making small additions to the existing classes can quickly create new classes.

" Programmer has to spend less time and effort and can concentrate on other aspects of the system due to the reusability feature of the methodology.

" Data Hiding - Encapsulation is a technique that allows the programmer to hide the internal functioning of the objects from the users of the objects. Encapsulation separates the internal functioning of the object from the external functioning thus providing the user flexibility to change the external behaviour of the object making the programmer code safe against the changes made by the user.

" The systems designed using this approach are closer to the real world as the real world functioning of the system is directly mapped into the system designed using this approach.

2.7. Dynamic System Development Method

Dynamic System Development Method is another approach to system development, which, as the name suggests, develops the system dynamically. This methodology is independent of tools, in that it can be used with both structured analysis and design approach or object-oriented approach.

The DSDM development process is dynamic as it is a Rapid Application Development method that uses incremental prototyping. This method is particularly useful for the systems to be developed in short time span and where the requirements cannot be frozen at the start of the application building. Whatever requirements are known at a time, design for them is prepared and design is developed and incorporated into system. In DSDM, analysis, design and development phase can overlap. Like at one time some people will be working on some new requirements while some will be developing something for the system. In DSDM, requirements evolve with time.

DSDM has a five-phase life cycle given as (see fig. 2.4):

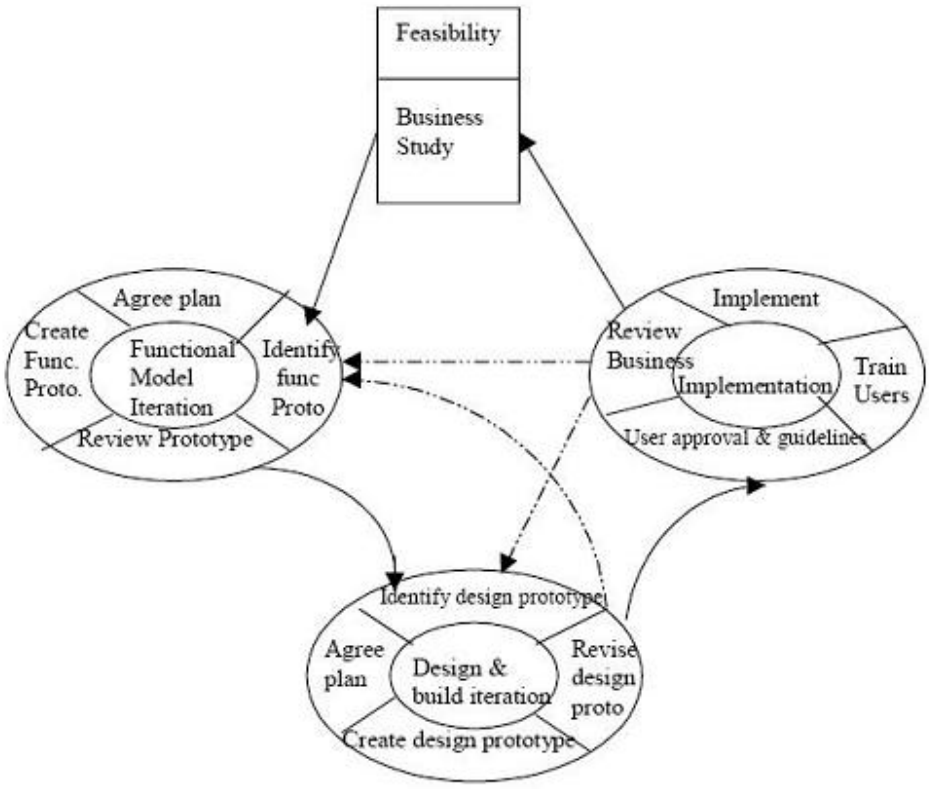


Fig. 2.4
Dynamic Systems Development Model
Source: DSDM consortium

Feasibility study

In this phase the problem is defined and the technical feasibility of the desired application is verified. Apart from these routine tasks, it is also checked whether the application is suitable for Rapid Application Development (RAD) approach or not. Only if the RAD is found as a justified approach for the desired system, the development continues.

Business study

In this phase the overall business study of the desired system is done. The business requirements are specified at a high level and the information requirements out of the system are identified. Once this is done, the basic architectural framework of the desired system is prepared.

The systems designed using RAD should be highly maintainable, as they are based on the incremental development process. The maintainability level of the system is also identified here so as to set the standards for quality control activities throughout the development process.

Functional Model Iteration

This is one of the two iterative phases of the life cycle. The main focus in this phase is on building the prototype iteratively and getting it reviewed from the users to bring out the requirements of the desired system. The prototype is improved through demonstration to the user, taking the feedback and incorporating the changes. This cycle is repeated generally twice or thrice until a part of functional model is agreed upon. The end product of this phase is a functional model consisting of analysis model and some software components containing the major functionality.

Design and Build Iteration

This phase stresses upon ensuring that the prototypes are satisfactorily and properly engineered to suit their operational environment. The software components designed during the Functional modeling are further refined till they achieve a satisfactory standard. The product of this phase is a tested system ready for implementation.

There is no clear line between these two phases and there may be cases where while some component has flown from the functional modeling to the design and build modeling while the other component has not yet been started. The two phases, as a result, may simultaneously continue.

Implementation

Implementation is the last and final development stage in this methodology. In this phase the users are trained and the system is actually put into the operational environment. At the end of this phase, there are four possibilities, as depicted by figure :

" Everything was delivered as per the user demand, so no further development required.

" A new functional area was discovered, so return to business study phase and repeat the whole process

" A less essential part of the project was missed out due to time constraint and so development returns to the functional model iteration.

" Some non-functional requirement was not satisfied, so development returns to the design and build iterations phase.

DSDM assumes that all previous steps may be revisited as part of its iterative approach. Therefore, the current step need be completed only enough to move to the next step, since it can be finished in a later iteration. This premise is that the business requirements will probably change anyway as understanding increases, so any further work would have been wasted.

According to this approach, the time is taken as a constraint i.e. the time is fixed, resources are fixed while the requirements are allowed to change. This does not follow the fundamental assumption of making a perfect system the first time, but provides a usable and useful 80% of the desired system in 20% of the total development time. This approach has proved to be very useful under time constraints and varying requirements.

2.8. Comparing Different Life Cycle Models

Waterfall model:

Major limitations:

" Real projects rarely follow sequential flow. Iterations are required.

" Requirements may not be clearly defined in the beginning itself.

" Major errors may require going back to Design phase or even Feasibility study phase.

" Customer would have no idea of the working program until the end. Very less interaction with the customer.

" Major blunders may be noticed only after development. " Product delivery takes much time.

Advantages:

" It is the oldest method. It is easily understood.

Prototyping model:

Limitations:

" Because of quick design, there can be compromise on quality.

Advantages:

" Fits user work patterns.

" Reduces risk of uncertainty.

" Allows early stoppage of unsuccessful projects.

" User interaction and involvement is more

" Less chances of error during system design.

DSDM Model:

Limitations:

" It is a relatively new model. It is not very common. So it is difficult to understand.

Advantages:

" Active user participation throughout the life of the project and iterative nature of development improves quality of the product.

" DSDM ensures rapid deliveries.

" Both of the above factors result in reduced project costs.

Object Oriented Method:

Advantages:

" OO Model closely represents the problem domain. Because of this, it is easier to produce and understand designs.

" The objects in the system are immune to requirement changes. Therefore, allows changes more easily.

" OO designs encourage more re-use. New applications can use the existing modules, thereby reduces the development cost and cycle time.

" OO approach is more natural. It provides nice structures for thinking and abstracting and leads to modular design.

2.9. Case Study: Library Management System(Preliminary analysis)

Presently all the functions of the "Noida Public Library" are done manually. For issue of books, each member has four library cards. Against each library card, a book is issued to the member. When the member returns that book the card is given back to the member.

However, this approach is very cumbersome. Sometimes the card is misplaced by library staff or by the member. Then a new card has to be issued. Number of members and the books are also increasing. It is becoming difficult to maintain all the records manually.

Therefore, the managing committee of the library has decided to go for automation of library functions. They have given this project to private company. The analyst who is dealing with this project needs to know the basic functioning of the library. The analyst needs to identify the entire problems associated with the present system.

2.10. Review Questions

Fill in the blanks

1. Different stages in the system life cycle model are preliminary analysis, _____, design, _____, testing, _____ & maintenance.

2. When the customer wants quick delivery _____ is best suited.

3. Waterfall model follows a systematic and _____ approach.

4. DSDM is a standard for _____ methodology.

5. What are the various stages in waterfall, prototyping & DSDM?

6. What are the advantages of prototype methodology over traditional waterfall model?

7. Provide five examples of s/w development projects that would be amenable(tuân theo) to prototyping. Name two or three that would be more difficult to prototype.

8. Research the literature on CASE tools and write a brief paper describing how modeling and simulation tools work.

9. What is difference between an object used in OOA and an object used during data modeling.

10. What is the difference between analysis and design? Can one begin to design without analysis? Why

2.11. Summary of System Development Life Cycle Models

Key points covered in this lesson are
>>The system's development life cycle consists of many phases. These include Analysis

1. Design
2. Coding
3. Testing
4. Maintenance

>>Various life cycle models are available in the industry.

1. Traditional (Waterfall)
2. Prototype
3. Object Oriented
4. Dynamic systems development model

>>Each model has its own limitations and advantages. These should be taken into consideration while deciding for the methodology.

Chapter 3: Preliminary Analysis

At the end of this chapter you will know the various activities performed in preliminary analysis of system development. And you will able to understand the various techniques used during the software estimation.

In the last two chapters we studies about systems, their components, different types of systems and different system development models. From this session onwards we'll be looking into details of various activities involved in the development of a system.

First stage of any system development is preliminary analysis. Understanding the customer request for the new or improved system is the first activity performed in preliminary analysis. Secondly thinking of possible solutions. After that, each possible solution is looked more carefully and it is ascertained whether that system is practically possible or not. In the end of preliminary analysis all the findings and recommendations for which solution to use, are presented to management, which finally decide whether to accept the proposal or reject it.

3.1. Preliminary Analysis

The main objectives of preliminary analysis is to identify the customer's needs, evaluate system concept for feasibility, perform economic and technical analysis, perform cost benefit analysis and create system definition that forms the foundation for all subsequent engineering works.

There should be enough expertise(sự tinh thông) available for hardware and software for doing analysis.

While performing analysis, the following questions arise.

" How much time should be spent on it?
As such, there are no rules or formulas available to decide on this. However, size, complexity, application field, end-use, contractual obligation are few parameters on which it should be decided.

" Other major question that arises is who should do it.
Well an experienced well-trained analyst should do it. For large project, there can be an analysis team.

After the preliminary analysis, the analyst should report the findings to management, with recommendations outlining the acceptance or rejection of the proposal.

3.1.1. Request Clarification

Initially when the customer makes a request for a system, the customer itself is not sure about the exact requirements of the system. In order to know what exactly the customer expects out of system, the analyst personally meets the end user or customer. For this analyst should be very tactful and have a good communications skills so that he/she is able to extract maximum information.

In this way analyst is able to know about the exact requirements for the desired system. All information gathered during the identification period is recorded in System Concept Document.

3.1.2. Feasibility Study

After request clarification, analyst proposes some solutions. After that for each solution it is checked whether it is practical to implement that solution.

This is done through feasibility study. In this various aspects like whether it is technically or economically feasible or not. So depending upon the aspect on which feasibility is being done it can be categorized into four classes:

- Technical
- Economic
- Legal
- Operational

The outcome of the feasibility study should be very clear. It should answer the following issues.

- Is there an alternate way to do the job in a better way?
- What is recommended?

3.1.3. Technical Feasibility

In technical feasibility the following issues are taken into consideration.

Whether the required technology is available or not

- Whether the required resources are available -
  - Manpower- programmers, testers & debuggers
  - Software and hardware

Once the technical feasibility is established, it is important to consider the monetary(tiền tệ) factors also. Since it might happen that developing a particular system may be technically possible but it may require huge investments and benefits may be less. For evaluating this, economic feasibility of the proposed system is carried out.

3.1.4. Economic Feasibility

For any system if the expected benefits equal or exceed the expected costs, the system can be judged to be economically feasible. In economic feasibility, cost benefit analysis is done in which expected costs and benefits are evaluated. Economic analysis is used for evaluating the effectiveness of the proposed system.

In economic feasibility, the most important is cost-benefit analysis. As the name suggests, it is an analysis of the costs to be incurred(phải gánh chịu) in the system and benefits derivable out of the system

3.1.5. Cost Benefit Analysis

Developing an IT application is an investment. Since after developing that application it provides the organization with profits. Profits can be monetary or in the form of an improved working environment. However, it carries risks, because in some cases an estimate can be wrong. And the project might not actually turn out to be beneficial.
Cost benefit analysis helps to give management a picture of the costs, benefits and risks. It usually involves comparing alternate investments.
Cost benefit determines the benefits and savings that are expected from the system and compares them with the expected costs.
The cost of an information system involves the development cost and maintenance cost. The development costs are one time investment whereas maintenance costs are recurring . The development cost is basically the costs incurred during the various stages of the system development Each phase of the life cycle has a cost. Some examples are :
Personnel
Equipment
Supplies

Overheads
Consultants' fees.

Cost and Benefit Categories (CBA)

In performing CBA it is important to identify cost and benefit factors. Cost and benefits can be categorized into the following categories.
There are several cost factors/elements. These are hardware, personnel, facility, operating, and supply costs.
In a broad sense the costs can be divided into two types
1. Development costs- Development costs that are incurred during the development of the system are one time investment.
Wages
Equipment
2. Operating costs, e.g. , Wages
Supplies
Overheads
Another classification of the costs can be:
Hardware/software costs:
It includes the cost of purchasing or leasing of computers and it's peripherals. Software costs involves required s/w costs.

Personnel costs:

It is the money, spent on the people involved in the development of the system. These expenditures include salaries, other benefits such as health insurance, conveyance allowance, etc.

Facility costs:

Expenses incurred during the preparation of the physical site where the system will be operational. These can be wiring, flooring, acoustics, lighting, and air conditioning.

Operating costs:

Operating costs are the expenses required for the day to day running of the system. This includes the maintenance of the system. That can be in the form of maintaining the hardware or application programs or money paid to professionals responsible for running or maintaining the system.

Supply costs:

These are variable costs that vary proportionately with the amount of use of paper, ribbons, disks, and the like. These should be estimated and included in the overall cost of the system.

Benefits

We can define benefit as
Profit or Benefit = Income - Costs

Benefits can be accrued(được sinh ra) by :

- Increasing income, or
- Decreasing costs, or
- both

The system will provide some benefits also. Benefits can be tangible or intangible, direct or indirect. In cost benefit analysis, the first task is to identify each benefit and assign a monetary value to it.

The two main benefits are improved performance and minimized processing costs.

Further costs and benefits can be categorized as

Tangible or Intangible Costs and Benefits

Tangible cost and benefits can be measured. Hardware costs, salaries for professionals, software cost are all tangible costs. They are identified and measured.. The purchase of hardware or software, personnel training, and employee salaries are example of tangible costs. Costs whose value cannot be measured are referred as intangible costs. The cost of breakdown of an online system during banking hours will cause the bank lose deposits.

Benefits are also tangible or intangible. For example, more customer satisfaction, improved company status, etc are all intangible benefits. Whereas improved response time, producing error free output such as producing reports are all tangible benefits. Both tangible and intangible costs and benefits should be considered in the evaluation process.

Direct or Indirect Costs and Benefits

From the cost accounting point of view, the costs are treated as either direct or indirect. Direct costs are having rupee value associated with it. Direct benefits are also attributable to a given project. For example, if the proposed system that can handle more transactions say 25% more than the present system then it is direct benefit.

Indirect costs result from the operations that are not directly associated with the system. Insurance, maintenance, heat, light, air conditioning are all indirect costs.

Fixed or Variable Costs and Benefits

Some costs and benefits are fixed. Fixed costs don't change. Depreciation of hardware, insurance, etc are all fixed costs. Variable costs are incurred on regular basis. Recurring period may be weekly or monthly depending upon the system. They are proportional to the work volume and continue as long as system is in operation.

Fixed benefits don't change. Variable benefits are realized on a regular basis.

Performing Cost Benefit Analysis (CBA)

Example:

Cost for the proposed system (figures in Rs lakhs)

Cost Category \ Year	1	2	3	4	5
Hardware	30				
Software	30				
Personnel	10	12	14	16	18
Maintenance	0	2	3	4	5
Cost at year end	70	24	17	20	23
Cumulative costs	70	94	111	131	154

Benefit for the propose system

Benefits \ Year	1	2	3	4	5
From finished reports	15	15	15	15	15
Increase in sales	25	35	45	55	65
Benefits at year end	40	50	60	70	80
Cumulative benefits	40	90	150	220	300

Profit = Benefits - Costs
= 300 -154
= Rs 146 lakhs

Since we are gaining, this system is feasible.

Steps of CBA can briefly be described as:

Estimate the development costs, operating costs and benefits
Determine the life of the system

When will the benefits start to accrue?
When will the system become obsolete?

Determine the interest rate
This should reflect a realistic low risk investment rate.

Select Evaluation Method

When all the financial data have been identified and broken down into cost categories, the analyst selects a method for evaluation.

There are various analysis methods available. Some of them are following.

1. Present value analysis
2. Payback analysis
3. Net present value
4. Net benefit analysis
5. Cash-flow analysis
6. Break-even analysis

Present value analysis:

It is used for long-term projects where it is difficult to compare present costs with future benefits. In this method cost and benefit are calculated in term of today's value of investment.

To compute the present value, we take the following formula Where,

$$PV = \frac{F}{(1 + i)^n}$$

i is the rate of interest &
n is the time

Example:

Present value of $3000 invested at 15% interest at the end of 5th year is calculates as

$P = 3000/(1 + .15)^5$
$= 1491.53$

Table below shows present value analysis for 5 years

Year	*Estimation Future Value*	*Present Value*	*Cumulative present Value of Benefits*
1	*3000*	*2608.69*	*2608.69*
2	*3000*	*2268.43*	*4877.12*
3	*3000*	*1972.54*	*6949.66*
4	*3000*	*1715.25*	*8564.91*
5	*3000*	*1491.53*	*10056.44*

Net Present Value : NPA

The net present value is equal to benefits minus costs. It is expressed as a percentage of the investment.

Net Present Value= Benefits - Costs
% = Net Present Value/Investments

Example: Suppose total investment is $50000 and benefits are $80000
Then Net Present Value = $(80000 - 50000)
= $30000
% = 30000/80000
=.375

Break-even Analysis:

Once we have determined what is estimated cost and benefit of the system it is also essential to know in what time will the benefits are realized. For that break-even analysis is done.

Break -even is the point where the cost of the proposed system and that of the current one are equal. Break-even method compares the costs of the current and candidate systems. In developing any candidate system, initially the costs exceed those of the current system. This is an investment period. When both costs are equal, it is break-even. Beyond that point, the candidate system provides greater benefit than the old one. This is return period.

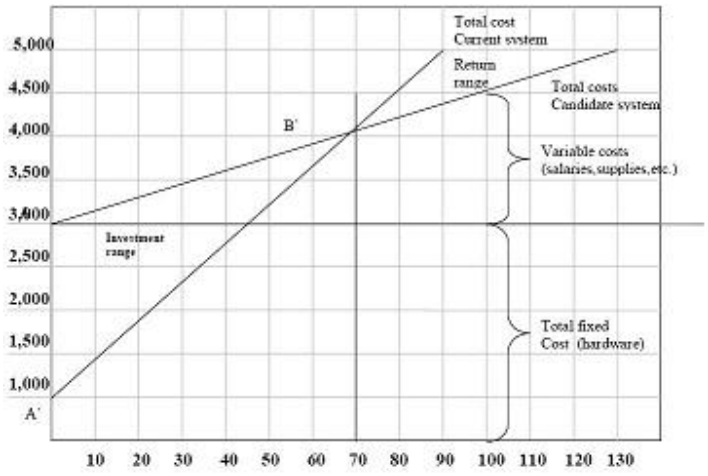


Fig. 3.1
Break-even Chart

Fig. 3.1 is a break-even chart comparing the costs of current and candidate systems. The attributes are processing cost and processing volume. Straight lines are used to show the model's relationships in terms of the variable, fixed, and total costs of two processing methods and their economic benefits. B' point is break-even. Area after B' is return period. A'AB' area is investment area. From the chart, it can be concluded that when the transaction are lower than 70,000 then the present system is economical while more than 70,000 transaction would prefer the candidate system.

Cash-flow Analysis:

Some projects, such as those carried out by computer and word processors services, produce revenues from an investment in computer systems. Cash-flow analysis keeps track of accumulated costs and revenues on a regular basis

3.1.6. Operational Feasibility

Operational feasibility is mainly concerned with issues like whether the system will be used if it is developed and implemented. Whether there will be resistance from users that will effect the possible application benefits? The essential questions that help in testing the operational feasibility of a system are following.

" Does management support the project?

" Are the users not happy with current business practices? Will it reduce the time (operation) considerably? If yes, then they will welcome the change and the new system.

" Have the users been involved in the planning and development of the project? Early involvement reduces the probability of resistance towards the new system.

" Will the proposed system really benefit the organization? Does the overall response increase? Will accessibility of information be lost? Will the system effect the customers in considerable way?

3.1.7. Legal Feasibility

It includes study concerning contracts, liability, violations, and legal other traps frequently unknown to the technical staff. Whatever the system, it has to be legally approved before we implement the system.

3.1.8. Request Approval

Those projects that are evaluated to be feasible and desirable are finally approved. After that, they are put into schedule. After a project is approved, it's cost, time schedule and personnel requirements are estimated.

Not all project proposals turn out to be feasible. Proposals that don't pass feasibility test are often discarded. Alternatively, some rework is done and again they are submitted as new proposals. In some cases, only a part is workable. Then that part can be combined with other proposals while the rest of it is discarded.

After the preliminary analysis is done and if the system turns out to be feasible, a more intensive analysis of the system is done. Various fact finding techniques are used for this purpose. In the next chapter, we'll look into these fact finding techniques.

When we do the economic analysis, it is meant for the whole system. But for the people who develop the system, the main issue is what amount of their effort goes into the building the system. So how much should be charged from the customer for building the system. For this, estimation is done by the developer. In estimation, software size and cost is estimated. Now we will study, the various techniques used in the software estimation.

3.2. Estimation

Software measurements, just like any other measurement in the physical world, can be categorized into Direct measures and Indirect measures. Direct measures of the software process include measurement of cost and effort applied. Direct measures of a product include Lines Of Code (LOC) produced, execution speed, memory size, and defects reported over some set of time. Indirect measures of the software process include measurement of resulting features of the software. Indirect measures of the product include its functionality, quality, complexity, efficiency, reliability, maintainability etc.

In the starting era of computers, software cost was a small proportion of the cost for complete computer based system. An error in estimation of software didn't have a major impact. But, Today software is the most expensive element in many computer-based systems. Large cost estimation errors can make the difference between profit and loss. Cost overruns can be disastrous for the developer.

Estimation is not an exact science. Too many variables - human, technical, environmental, and political - can affect the ultimate cost of software and effort applied to develop it.

In order to make a reliable cost and effort estimation there are lot of techniques that provide estimates with acceptable mount of risk.

These methods can be categorized into Decomposition techniques and Empirical Estimation models. According to Decomposition techniques the software project is broken into major functions and software engineering activities (like testing, coding etc.) and then the cost and effort estimation of each component can be achieved in a stepwise manner. Finally the consolidated estimate is the clubbed up entities resulting from all the individual components. Two methods under this head are Lines of Code (LOC) and Function Points Estimation (FP). Further Empirical Estimation models are used to offer a valuable estimation approach, which is based on historical data (experience).

3.2.1. Lines of code (LOC)

LOC method measures software and the process by which it is being developed. Before an estimate for software is made, it is important and necessary to understand software scope and estimate its size.

LOC is a direct approach method and requires a higher level of detail by means of decomposition and partitioning. In contrast, Function Points (FP) is indirect an approach method where instead of focusing on the function it focuses on the domain characteristics (discussed later in the chapter).

From statement of software scope software is decomposed into problem functions that can each be estimated individually. LOC or FP (estimate variables) is then calculated for each function.

An expected value is then computed using the following formula.

$$EV = (S_{opt} + 4S_m + S_{pess}) / 6$$

where,
EV stand for the estimation variable.
S_{opt} stand for the optimistic estimate.
S_m stands for the most likely estimate.
S_{pess} stand for the pessimistic estimate.

It is assumed that there is a very small probability that the actual size results will fall outside the optimistic or pessimistic value.

Once the expected value for estimation variable has been determined, historical LOC or FP data are applied and person months, costs etc are calculated using the following formula.

Productivity = KLOC / Person-month
Quality = Defects / KLOC
Cost = $ / LOC
Documentation = pages of documentation / KLOC

Where,
KLOC stand for no. of lines of code (in thousands).
Person-month stand for is the time(in months) taken by developers to finish the product.
Defects stand for Total Number of errors discovered

Example:

Problem Statement: Take our Library management system case. Software developed for library will accept data from operator for issuing and returning books. Issuing and returning will require some validity checks. For issue it is required to check if the member has already issued the maximum books allowed. In case for return, if the member is returning the book after the due date then fine has to be calculated. All the interactions will be through user interface. Other operations include maintaining database and generating reports at regular intervals.

Major software functions identified.
1. User interface
2. Database management
3. Report generation

For user interface
Sopt : 1800
Sm : 2000
Spess : 4000

EV for user interface
EV = (1800 + 4*2000 + 4000) / 6
EV = 2300

For database management
S_{opt} : 4600
S_m : 6900
S_{pess} : 8600

EV for database management
EV = (4600 + 4*6900 + 8600) / 6
EV = 6800

For report generation
S_{opt} : 1200
S_m : 1600
S_{pess} : 3200

EV for report generation
EV = (1200 + 4*1600 + 3200) / 6
EV = 1800

Function	***Estimated LOC***
User Interface	2300
Database management	6800
Report Generation	1800
Estimated LOC	10900

Suppose productivity = 3000 LOC/ person-month

Development Cost = $8000 per month

person-month = productivity / LOC
= 3000/ 10900
= 3.64

Cost of software = Development Cost * person-month
= 8000 * 3.64
= $ 29120

Cost / LOC = 29120/10900
= 2.67 $/LOC

3.2.2. FP Estimation

Function oriented metrics focus on program "functionality" or "utility". Albrecht first proposed function point method, which is a function oriented productivity measurement approach.

Five information domain characteristics are determined and counts for each are provided and recorded in a table.

- Number of user inputs
- Number of user outputs
- Number of user inquires
- Number of files
- Number of external interfaces

Once the data have been collected, a complexity value is associated with each count. Each entry can be simple, average or complex. Depending upon these complexity values is calculated.

To compute function points, we use

FP = count-total X [0.65 + 0.01 * SUM(Fi)]

Where, count-total is the sum of all entries obtained from fig. 3.2

Fi(I= 1 to 14) are complexity adjustment values based on response to questions(1-14) given below. The constant values in the equation and the weighing factors that are applied to information domain are determined empirically.

Fi
1. Does the system require reliable backup and recovery?
2. Are data communications required?
3. Are there distributed processing functions?
4. Is performance critical?
5. Will the system run in an existing, heavily utilized operational environment?
6. Does the system require on-line entry?
7. Does the on-line data entry require the input transaction to be built over multiple screens or operations?
8. Are the inputs, outputs, files, or inquiries complex
9. Is the internal processing complex?
10. Is the code designed to be reusable?
11. Are master files updated on-line?
12. Are conversion and installations included in the design?
13. Is the system designed for multiple installations in different organizations?
14. Is the application designed to facilitate change and ease of use by the user?

Rate each factor on a scale of 0 to 5

0 - No influence
1 - Incidental
2 - Moderate
3 - Average
4 - Significant
5 - Essential

Count-total is sum of all FP entries.

Once function points have been calculated, productivity, quality, cost and documentation can be evaluated.

Productivity = FP / person-month
Quality = defects / FP
Cost = $ / FP
Documentation = pages of documentation / FP

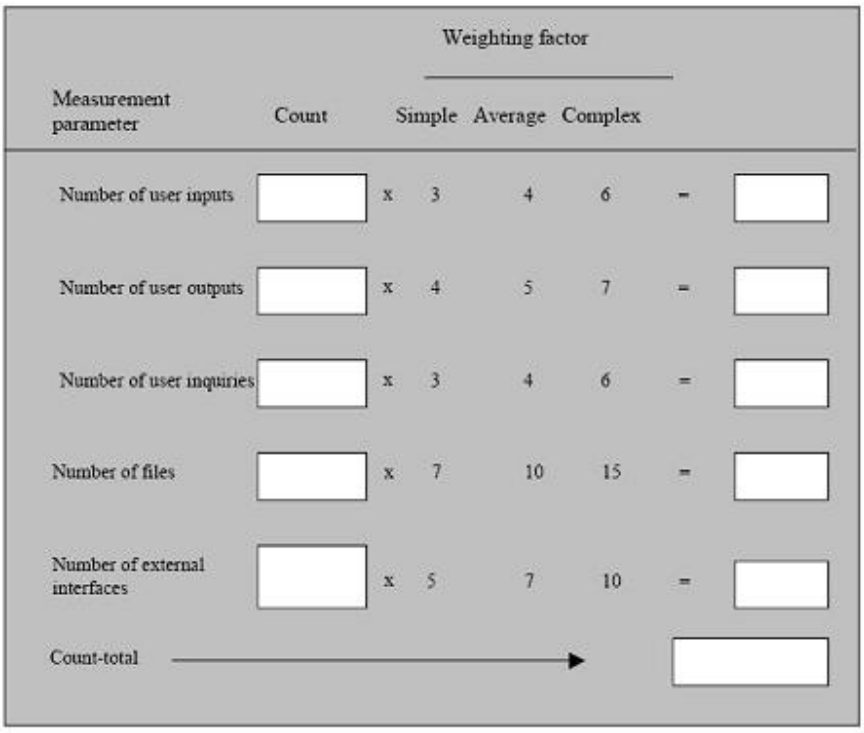


Fig. 3.2
Computing function-point metrics

Example: Same as LOC problem

Information Domain Values	**Opt**	**likely**	**Pess**	**est Account**	**wt**	**FP Account**
Number of Inputs	4	10	16	10	4	40
Number of Outputs	4	7	16	8	5	40
Number of Inquiries	5	12	19	12	4	48
Number of Files	3	6	9	6	10	60
Number of external interfaces	2	2	3	2	7	14
Count Total						**202**

Complexity weighing factors are determined and the following results are obtained.

Factor	**Value**
Backup Recovery	4
Data Communication	1
Distributed processing	0
Perfomance Critical	3
Existing Operating Environment	2
On-line data entry	5
Input transaction over multiple screens	5
Master file updated online	3
Information domain values complex	3
Internal processing complex	2
Code design for reuse	0
Conversion/Installation in design	1
Multiple installations	3
Application designed for change	5

Sum (Fi)	37

Estimated number of FP :

FPestimated = count-total * [0.65 + .01 * sum(Fi)]
FPestimated = 206

From historical data, productivity is 55.5 FP/person-month and development cost is $8000 per month.

Productivity = FP/ person-month
person-month = FP/Productivity
= 202/55.5

= 3.64 person-month

Total Cost = Development cost * person-month
= 8000 * 3.64
=$29100

Cost per FP = Cost/FP
= 29100/202
= $144.2per FP

3.2.3. Empirical Estimation

In this model, empirically derived formulas are used to predict data that are a required part of the software project-planning step. The empirical data are derived from a limited sample of projects. Resource models consist of one or more empirically derived equations. These equations are used to predict effort (in person-month), project duration, or other pertinent project data. There are four classes of resource models:

" Static single-variable models
" Static multivariable models
" Dynamic multivariable models
" Theoretical models

Static single-variable model has the following form

Resource = c1 X (estimated characteristics c2)

Where,
Resource could be effort, project duration, staff size, or lines of software documentation.

c1 and c2 are constants derived from data of past projects.

Estimated characteristics is line of code, effort (if estimated), or other software characteristics.

The basic version of the Constructive Cost Model, or COCOMO, presented in the next section is an example of a static-variable model.

Static multivariable models also use historical data to derive empirical relationships. A typical model of this category takes the form

Resource = c11e1 + c12e2+ ……

Where ei is the ith software characteristics and ci1,ci2 are empirically derived constants for the ith characteristics.

In dynamic multivariable models, resource requirements are projected as a function of time. If the model is derived empirically, resources are defined in a series of time steps that allocate some percentage of effort (or other resource) to each step in the software engineering process. Further, each step may be divided into tasks. A theoretical approach to dynamic multivariable modeling hypothesizes a continuous "resource expenditure curve" and from it, derives equations that model behavior of the resource. The Putnam Estimation Model is an example of theoretical dynamic multivariable model.

3.2.4. COCOMO, COnstructive COst MOdel

COCOMO, COnstructive COst MOdel is static single-variable model. Barry Boehm introduced COCOMO models. There is a hierarchy of these models.

Model 1:

Basic COCOMO model is static single-valued model that computes software development effort (and cost) as a function of program size expressed in estimated lines of code.

Model 2:

Intermediate COCOMO model computes software development effort as a function of program size and a set of "cost drivers" that include subjective assessments of product, hardware, personnel, and project attributes.

Model 3:

Advanced COCOMO model incorporates all characteristics of the intermediate version with an assessment of the cost driver's impact on each step, like analysis, design, etc.

COCOMO can be applied to the following software project's categories.

Organic mode:

These projects are very simple and have small team size. The team has a good application experience work to a set of less than rigid requirements. A thermal analysis program developed for a heat transfer group

Semi-detached mode:

These are intermediate in size and complexity. Here the team has mixed experience to meet a mix of rigid and less than rigid requirements. A transaction processing system with fixed requirements for terminal hardware and database software is an example of this.

Embedded mode:

Software projects that must be developed within a set of tight hardware, software, and operational constraints. For example, flight control software for aircraft.

The basic COCOMO model takes the form

$$E = a_b\,(KLOC)\exp(b_b)$$

$$D = C_b(E)\exp(d_b)$$

where,
E is the effort applied in person-month,
D is the development time in chronological month,

KLOC is the estimated number of delivered lines(expressed in thousands) of code for project, The a_b and c_b and the exponents b_b and d_b are given in the table below.

Software Project	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	.38
Semi-detached	3.0	1.12	2.5	.35
Embedded	3.6	1.20	2.5	.32

Fig. 3.3
Basic COCOMO

The basic model is extended to consider a set of "cost drivers attributes". These attributes can be grouped together into four categories.

1) Product attributes

a) Required software reliability.
b) Complexity of the project.
c) Size of application database.

2) Hardware attributes

a) Run-time performance constraints.
b) Volatility of the virtual machine environment.
c) Required turnaround time.
d) Memory constraints.

3) Personnel attributes

a) Analyst capability.
b) Software engineer capability.
c) Virtual machine experience.
d) Application experience.
e) Programming language experience.

4) Project attributes

a) Application of software engineering methods.
b) Use of software tools.
c) Required development schedule.

Each of the 15 attributes is rated on a 6-point scale that ranges from "very low" to "very high" in importance or value. Based on the rating, an effort multiplier is determined from the tables given by Boehm. The product of all multipliers results in an effort adjustment factor (EAF). Typical values of EAF range from 0.9 to 1.4.

Example : Problem Statement same as LOC problem refer section 3.2.1

KLOC = 10.9
$E = a_b (KLOC)^{exp(b_b)}$
= 2.4(10.9)exp(1.05)
= 29.5 person-month

$D = C_{b(E)}^{exp(d_b)}$
= 2.5(29.5)exp(.38)
= 9.04 months

The intermediate COCOMO model takes the following form.

E = ai(LOC)exp(bi) X EAF

Where,
E is the effort applied in person-months,
LOC is the estimated number of delivered lines of code for the project.
The coefficient ai and the exponent bi are given in the table below.

Software project	a_i	b_i
Organic	3.2	1.05
Semidetached	3.0	1.12
Embedded	2.8	1.20

Fig. 3.4
Intermediate COCOMO

3.2.5. Preliminary Analysis - Case study: Library Management System

Again referring back to our "Library management system" discussed in earlier chapters, we now apply to it the concepts that we have studied in this chapter.

Preliminary Analysis:

Request Clarification.

First the management of the library approached this ABC Software Ltd. for their request for the new automated system. What they stated in their request was that they needed a system for their library that could automate its various functions. And provide faster response.

From this request statement, it is very difficult for the analyst to know what exactly the customer wants. So in order to get information about the system, the analyst visits the library site and meets the

staff of the library. Library staff is going to be the end user of the system. Analyst asks various questions from the staff so that the exact requirements for the system become clear. From this activity, the analyst is able to identify the following requirements for the new system:

" Function for issue of books
" Function for return of books that can also calculate the fine if the book is returned after the due date
" Function for performing different queries
" Report generation functions
" Function for maintaining account s
" Maintaining the details for members, books, and suppliers in some structured way.

Now that the requirements are known, the analyst proposes solution system.

Solution: The desired system can be implemented with Oracle RDBMS in the back end with Visual Basic as the front end. It will have modules for handling issue and return functions, generating reports, performing checks, and maintaining accounts. It will also store the data relating to books, members, and suppliers in a structures way. In our case,
the data will be maintained in a relational way.

Feasibility Study

Now the next stage in preliminary analysis is to determine whether the proposed solution is practical enough to be implemented. For this feasibility study is done.

First technical feasibility is done.

Major issues in technical feasibility are to see if the required resources- trained manpower, software and hardware are available or not.

ABC Software Ltd. is big IT Company. It has developed similar projects using Oracle and VB. It has a special team that is formed to work in this combination of projects, that is, Oracle and Visual Basic. So manpower is readily available. The software is available with the company since it has already worked with the same software earlier also. So our
solution is technically feasible.

Technical feasibility doesn't guarantee if the system will be beneficial to the system if developed. For this economic feasibility is done.

First task that is done in economic analysis is to identify the cost and benefit factors in the system proposed. In our case, the analyst has identified the following costs and benefits.

Costs

Cost	Cost per unit	Quantity	Total Cost
Software			
Oracle	50,000	1	50,000
Visual basic	30,000	1	30,000
Windows Server 2003	15,000	1	15,000
Windows XP professional	5,000	4	5,000
Hardware			
Central Computer	100,000	1	100,000
Client Machine	50,000	4	50,000
Development	50,000	1	50,000
Analyst	50,000	1	50,000
Developer	20,000	2	40,000

Training	20,000	1	20,000
Data Entry	5,000	1	5,000
Warranty (1 month)			
Professional	20,000	1	20,000
Total Cost			5,55,000

Benefits

According to new policy: A member is required to pay Rs 500 for a half yearly membership and Rs 1000 for a year membership.

Expected increase in number of members : 75 per month
40 new members for 1 year and 35 new members for half year

Free collected from new members in one year = 12 (40 * 1000 + 35 * 500)
= Rs 6,90,000

For four years = 4 * 6,90,000
= 27,60,000

Now using Net present value method for cost benefit analysis we have,

Net present value(or gain) = Benefits - Costs
= 27,60,000 - 5,55,000
= 22,10,000

Gain % = Net present value / investment
= 22,10,000 / 5,55,000
= 4.018

Overall Gain = 401.8 % in four year

For each year

First year

Investment = 5,50,000
Benefit = 6,90,000

Net present value for first year = 6,90,000 - 5,50,000
= 1,40,000

Gain % = 1,40,000 / 5,50,000
= .254
= 25.4 % in first year

Second Year

Investment = 5,50,000
Benefit = 13,80,000

Net present value for second year	= 13,80,000 - 5,50,000 = 8,30,000
Gain %	= 830000/550000 = 1.50
	= 150 % at the end of second year

Third Year

Investment	= 5,50,000
Benefit	= 20,70,000
Net present value for third year	= 20,70,000 - 5,50,000 = 15,20,000
Gain %	= 1520000/550000 = 2.76
	= 276 % at the end of third year

Fourth Year

Invetment	= 550,000
Benefit	= 2760000
Net Present Value for fourth year	= 2760000 - 550000 = 2210000
Gain %	= 2210000 / 550000 = 4.018
	= 401.8 % at end of fourth year

From CBA we have found that it is economically feasible since it is showing great gains(about 400%).

After economic feasibility, operational feasibility is done. In this, major issue is to see if the system is developed what is the likelihood that it'll be implemented and put to operation? Will there be any resistance from its user?

It is very clear that the new automated system will work more efficiently and faster. So the users will certainly accept it. Also they are being actively involved in the development of the new system. Due to this fact they will know the system well and will be happy to use a new improved system. So our system is operationally feasible. After the feasibility study has been done and it is found to be feasible, the management has approved this project. So further work can de done that is the design of system that will be discussed in the later chapters.

3.3. Preliminary Analysis - Self Assessment Questions

Fill in the blanks

1) Preliminary analysis includes _____, _____ and _____.

2) Feasibility study includes _____, _____, _____, and _____.

3) Economic feasibility includes _____-.

4) Match the following:

a. Present value analysis	1. A percentage
b. Return-on-investment	2. A year
c. Payback analysis	3. Time value of money

5) Development a list of s/w characteristics(e.g concurrent operation, graphical output, etc) that affect the complexity of a program. Prioritize the list.

3.4. Preliminary Analysis - Summary

Key points covered in this chapter are:

>> Preliminary analysis consists of

" Request clarification
" Feasibility study
" Request approval

>> In feasibility study following areas are covered.

" Technical feasibility
" Economic feasibility
" Legal feasibility
" Operational feasibility.

>> Data analysis is a prerequisite to cost/benefit analysis.

>> Costs and benefits can be classified into tangible or intangible, fixed or variable and direct or indirect.

>> In developing cost estimates, hardware, personnel, facility, operating, and supply costs all should be considered.

>> There are a number of cost-benefit analysis

" Net benefit analysis
" Present value analysis
" Net present value
" Payback analysis
" Break-even analysis
" Cash-flow analysis

>> Once the evaluation of the project is complete, actual results are compared against standards or alternative investments.

>> Estimation is very important activity in the development of any system as large cost estimation errors can make the difference between profit and loss. And, cost overruns can be disastrous for the developer.

>> LOC is the direct measure the direct measure of software and the process by which it is developed.

>> There are other methods also.

" FP estimation
" Empirical Estimation

>> In empirical method, there is CCCOMO estimation model.

>> There are three types of COCOMO models

>> Basic COCOMO model is a static single-valued model that computes software development effort (and cost) as a function of program size expressed in estimated lines of code

>> Intermediate COCOMO model computes software development effort as a function of program size and a set of cost drivers that include subjective assessment of product, hardware, personnel, and project attributes.

>> Advanced COCOMO model incorporates all characteristics of the intermediate version with an assessment of the cost driver's impact on each step, like analysis, design, etc

3.5. Preliminary Analysis - Exercises

1. Development a checklist for attribute to be considered when the feasibility of a system is to be evaluated.

2. What is preliminary analysis and what is the analyst's main function during this phase of the systems process?

3. Why is it difficult to determine user requirements? Illustrate.

4. What is the purpose of a preliminary investigation? What outcome is expected? Who carries out this investigation?

5. What is an infeasible project? How are infeasible projects handles?

6. List the inputs to preliminary analysis.

7. List the outputs from preliminary analysis.

8. What are the components of the feasibility study?

9. The following list of items relates to costs and benefits of a computerized marketing system $2000 savings in wages of one employee.

---Better accuracy of results but at a s/w cost of $60,000
---More timely information.
---Development cost of $60,000

---Initial operational cost of $10,000
---Fringe benefits equal to 15% of wages.
---Provide information not previously available
---Cost of security equals $10,000
---Provides greater customer satisfaction
---Provides better managerial control
---Results in larger sales
---Annual operational costs of $6,000
---Telecommunication cost of $300/month
---Faster delivery of outputs.

Identify which of these items are:
a) Tangible benefits
b) Intangible benefits
c) Tangible fixed costs
d) Tangible variable costs

Chapter 4: Fact Finding and Decision Making Techniques

At the end of this chapter you would be able to know the various fact finding techniques and you would also be able to understand techniques used for decision making

4.1. Fact Finding Techniques

Fact-finding is an important activity in system investigation. In this stage, the functioning of the system is to be understood by the system analyst to design the proposed system. Various methods are used for this and these are known as fact-finding techniques. The analyst needs to fully understand the current system.

The analyst needs data about the requirements and demands of the project undertaken and the techniques employed to gather this data are known as fact-finding techniques.

Various kinds of techniques are used and the most popular among them are interviews, questionnaires, record reviews, case tools and also the personal observations made by the analyst himself. Each of these techniques is further dealt with.

4.1.1. Interviews

Interview is a very important data gathering technique as in this the analyst directly contacts system and the potential user of the proposed system.

One very essential aspect of conducting the interview is that the interviewer should first establish a rapport with the interviewee . It should also be taken into account that the interviewee may or may not be a technician and the analyst should prefer to use day to day language instead of jargon and technical terms.

For the interview to be a success the analyst should be appropriately prepared, as he needs to be beforehand aware of what exactly needs to be asked and to what depth. Also he should try to gather maximum relevant information and data. As the number and type of respondents vary, the analyst should be sensitive to their needs and nature.

The advantage of interviews is that the analyst has a free hand and he can extract almost all the information from the concerned people but then as it is a very time consuming method, he should also employ other means such as questionnaires, record reviews, etc.

This may also help the analyst to verify and validate the information gained. Interviewing should be approached, as logically as possible and from a general point of view the following guides can be very beneficial for a successful interview.

1. Set the stage for the interview.
2. Establish rapport; put the interviewee at ease.
3. Phrase questions clearly and succinctly.
4. Be a good listener; avoid arguments.
5. Evaluate the outcome of the interview.

The interviews are of two types namely structured and unstructured.

4.1.1.1. Structured Interview

Structured interviews are those where the interviewee is asked a standard set of questions in a particular order. All interviewees are asked the same set of questions.

The questions are further divided in two kinds of formats for conducting this type of interview. The first is the open-response format in which the respondent is free to answer in his own words. An example of open-response is "Why are you dissatisfied with the current leave processing method?" The other option is of closed-response format, which limits the respondents to opt their answer from a set of already prescribed choices. An example of such a question might be "Are you satisfied with the current leave processing methods?" or "Do you think that the manual leave processing procedure be changed with some automated procedure?"

4.1.1.2. Unstructured Interview

The unstructured interviews are undertaken in a question-and-answer format. This is of a much more flexible nature than the structured interview and can be very rightly used to gather general information about the system.

Here the respondents are free to answer in their own words. In this way their views are not restricted. So the interviewer get a bigger area to further explore the issues pertaining to a problem.

4.1.2. Questionnaires

Questionnaires are another way of information gathering where the potential users of the system are given questionnaires to be filled up and returned to the analyst.

Questionnaires are useful when the analyst need to gather information from a large number of people. It is not possible to interview each individual. Also if the time is very short, in that case also questionnaires are useful. If the anonymity of the respondent is guaranteed by the analyst then the respondent answers the questionnaires very honestly and critically.

The questionnaire may not yield the results from those respondents who are busy or who may not give it a reasonable priority.

The analyst should sensibly design and frame questionnaires with clarity of it's objective so as to do justice to the cost incurred on their development and distribution. Just like the interviews and on the same lines questionnaires are of two types i.e. *Open-Response Based* and the *Closed-Response Based*

4.1.2.1. Open-Response Based Questionnaires

The objective of open-response questionnaire is to gather information and data about the essential and critical design features of the system. The open-ended question requires no response direction or specific response.

This form is also used to learn about the feelings, opinions, and experiences of the respondents. This information helps in the making the system effective because the analyst can offer subsequent modifications as per the knowledge gained.

4.1.2.2. Closed-Response Based Questionnaires

The objective of closed-response questionnaire is to collect the factual information of the system. It gives an insight in how the people dealing with the system behave and how comfortable are they with it. In this case the respondents have to choose from a set of given responses. Thus the respondent can express their liking for the most favorable one from the possible alternatives.

The closed questions can be of various types and the most common ones are listed below.

1. Fill-in-the-blanks.

2. Dichotomous i.e. Yes or No type.

3. Ranking scale questions ask the respondents to rank a list of items in the order of importance or preference.

4. Multiple-choice questions which offer respondents few fixed alternatives to choose from.

5. Rating scale questions are an extension of the multiple-choice questions. Here the respondent is asked to rate certain alternatives on some given scale.

4.1.2.3. Open Response-Based Vs Closed Response-Based

The basic comparison between the two can be made on the grounds of the format used. Open form offers more flexibility and freedom to the respondents whereas the closed form is more specific in nature.

Open-ended questionnaires are useful when it is required to explore certain situation.They also require a lot of time of the analyst for evaluation.

Closed questionnaires are used when factual information is required. Closed questions are quick to analyze but typically most costly to prepare but they are more suitable to obtain factual and common information.

It is the job of the analyst to decide which format should be employed and what exactly should be it's objective. The care should be taken to ensure that all the parts of the form are easy to understand for the respondents so that they can answer with clarity

4.1.3. Record Reviews

Records and reports are the collection of information and data accumulated over the time by the users about the system and it's operations. This can also put light on the requirements of the system and the modifications it has undergone. Records and reports may have a limitation if they are not up-to-date or if some essential links are missing. All the changes, which the system suffers, may not be recorded.

The analyst may scrutinize the records either at the beginning of his study which may give him a fair introduction about the system and will make him familiar with it or in the end which will provide the analyst with a comparison between what exactly is/was desired from the system and it's current working.

One drawback of using this method for gathering information is that practically the functioning of the systems is generally different from the procedure shown in records. So analyst should be careful in gathering in gathering information using this method.

4.1.4. On-site Observation

On-site observations are one of the most effective tools with the analyst where the analyst personally goes to the site and discovers the functioning of the system. As an observer, the analyst can gain first hand knowledge of the activities, operations, processes of the system on-site, hence here the role of an analyst is of an information seeker.

This information is very meaningful as it is unbiased and has been directly taken by the analyst. This exposure also sheds some light on the actual happenings of the system as compared to what has already been documented, thus the analyst gets closer to the system. This technique is also time-consuming and the analyst should not jump to conclusions or draw inferences from small samples of observation rather the analyst should be more patient in gathering the information. This method is however less effective for learning about people's perceptions, feelings and motivations.

4.2. Decision Making and Documentation

Decision-making is an integral part of any business no matter how small, simple or big and complex it may be. Thus decisions have to be made and set procedures are to be followed as the subsequent actions. Thus while analyzing and designing a business system, analyst also needs to identify and document any decision policies or business rules of the system being designed. There are various tools and techniques available to the analyst for this, like decision trees, decision tables or structured English.

To analyze procedures and decisions the first step to be taken is to identify conditions and actions of all possible activities. Conditions are the possibilities or the possible states of any given entity, which can be a person, place, thing, or any event. Conditions are always in a flux i.e. they keep on varying time to time and object to object and based only on these conditions the decisions are made therefore conditions are also put as decision variables.

Documentation comes as an aid in this condition based decision process. As the whole web of all the possible combination of conditions and decisions is usually very large and cumbersome it becomes extremely important to document these so that there are no mistakes committed during the decision process.

Here comes the role of documenting tools, which are available to the analyst. Tools, which are usually used, are decision trees, decision tables, Structured English and the various CASE tools. The basic role of these tools is to depict the various conditions, their possible combinations and the subsequent decisions.

This has to be done without harming the logical structure involved. Once all of the parameters are objectively represented the decision process becomes much simpler, straightforward and almost error free.

4.2.1. Decision Trees

Decision tree is a tree like structure that represents the various conditions and the subsequent possible actions. It also shows the priority in which the conditions are to be tested or addressed. Each of its branches stands for any one of the logical alternatives and because of the branch structure, it is known as a tree.

The decision sequence starts from the root of the tree that is usually on the left of the diagram. The path to be followed to traverse the branches is decided by the priority of the conditions and the respectable actions. A series of decisions are taken, as the branches are traversed from left to right. The nodes are the decision junctions. After each decision point there are next set of decisions to be considered. Therefore at every node of the tree represented conditions are considered to determine which condition prevails before moving further on the path.

This decision tree representation form is very beneficial to the analyst. The first advantage is that by using this form the analyst is able to depict all the given parameters in a logical format which enables the simplification of the whole decision process as now there is a very remote chance of committing an error in the decision process as all the options are clearly specified in one of the most simplest manner.

Secondly it also aids the analyst about those decisions, which can only be taken when couple or more conditions should hold true together for there may be a case where other conditions are relevant only if one basic condition holds true.

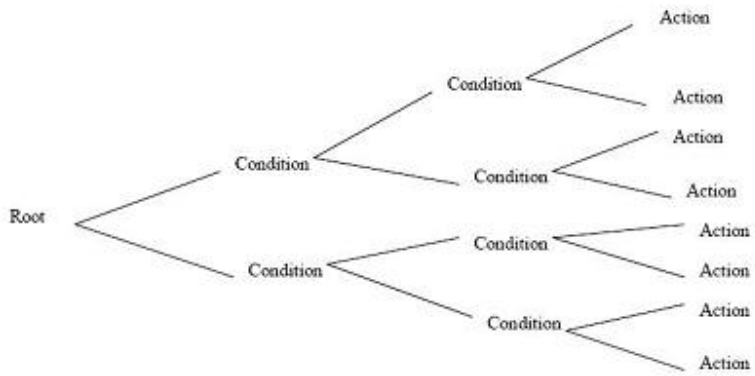


Fig. 4.1
Decision Tree

In our day-to-day life, many a times we come across complex cases where the most appropriate action under several conditions is not apparent easily and for such a case a decision tree is a great aid. Hence this representation is very effective in describing the business problems involving more then one dimension and parameters.

They also point out the required data, which surrounds the decision process. All the data used in the decision making should be first described and defined by the analyst so that the system can be designed to produce correct output data.
Consider for example the discount policy of a saree manufacturer for his customers.

According to the policy the saree manufacturer give discount to his customers based on the type of customer and size of their order. For the individual, only if the order size is 12 or more, the manufacturer gives a discount of 50% and for less than 12 sarees the discount is 30%. Whereas in case of shopkeeper or retailers, the discount policy is different. If the order is less than 12 then there is 15% discount. For 13 to 48 sarees order, the discount is 30%, for 49 to 84 sarees 40% and for more than 85 sarees the discount is 50%. The decision policy for discount percentage can be put in the form of a
decision tree displayed in fig 4.2

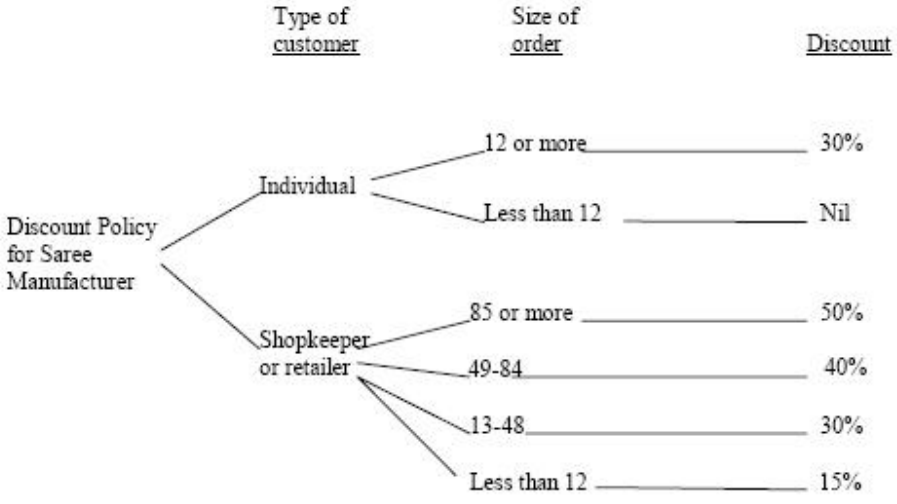


Fig. 4.2
A Sample Decision Tree

The decision trees are not always the most appropriate and the best tool for the decision making process. Representing a very complex system with this tool may lead to a huge number of branches with a similar number of possible paths and options.

For a complex problem, analyzing various situations is very difficult and can confuse the analyst

4.2.2. Decision Tables

A decision table is a table with various conditions and their corresponding actions. Decision tree is a two dimensional matrix. It is divided into four parts, condition stub, action stub, condition entry, and action entry. See fig 4.3. Condition stub shows the various possible conditions.

Condition entry is used for specifying which condition is being analyzed. Action stub shows the various actions taken against different conditions.

And action entry is used to find out which action is taken corresponding to a particular set of conditions.

The steps to be taken for a certain possible condition are listed by action statements. Action entries display what specific actions to be undertaken when selected conditions or combinations of conditions are true. At times notes are added below the table to indicate when to use the table or to distinguish it from other decisions tables.

The right side columns of the table link conditions and actions and form the decision rules hence they state the conditions that must be fulfilled for a particular set of actions to be taken. In the decision trees, a fixed ordered sequence is followed in which conditions are examined. But this is not the case here as the decision rule incorporates all the required conditions, which must be true.

Developing Decision Tables

Before describing the steps involved in building the decision table it is important to take a note of few important points. Every decision should be given a name and the logic of the decision table is independent of the sequence in which condition rules are written but the action takes place in the order in which events occur. Wherever possible, duplication of
terms and meaning should be avoided and only the standardized language must be used.

The steps of building the concerned tables are given below.

1. Firstly figure out the most essential factors to be considered in making a decision.

This will identify the conditions involved in the decision. Only those conditions should be selected which have the potential to either occur or not but partial occurrences are not permissible.

2. Determine the most possible steps that can take place under varying conditions and not just under current condition. This step will identify the actions.

3. Calculate all the possible combinations of conditions.

For every N number of conditions there are 2*2*2…. (N times) combinations to be considered.

4. Fill the decision rules in the table.

Entries in a decision table are filled as Y/N and action entries are generally marked as "X". For the conditions that are immaterial a hyphen "-" is generally put. Decision table is further simplified by eliminating and consolidating certain rules. Impossible rules are eliminated. There are certain

conditions whose values do not affect the decision and always result in the same action. These rules can be consolidated into a single rule.

Example: Consider the recruitment policy of ABC Software Ltd.

It the applicant is a BE then recruit otherwise not. If the person is from Computer Science, put him/her in the software development department and if the person is from non-computer science background put him/her in HR department. If the Person is from Computer Science and having experience equal to or greater than three years, take him/her as Team leader and if the experience is less than that then take the person as Team member. If the person recruited is from non Computer Science background, having experience less than three years, make him/her Management Trainee otherwise Manager

Condition Stub	***Condition Entry***					
	1	***2***	***3***	***4***	***5***	***6***
Customer is individual ?	***Y***	***Y***				
Customer shopkeeper or retailer ?			***Y***	***Y***	***Y***	***Y***
Order-size 85 copies or more ?			***Y***			
Order-size 49-84 sarees ?				***Y***		
Order-size 13-48 copies ?					***Y***	
Order-size 12 or more ?		***Y***				
Order-size less than 12?	***Y***					***Y***
Allow 50% discount		***X***	***X***			
Allow 40% discount				***X***		
Allow 30% discount	***X***				***X***	
Allow 15% discount						***X***

Fig 4.3

Decision table-Discount Policy

The first decision table for the problem stated above can be drawn as shown in fig 4.4

Condition Stub	Condition Entry							
	1	2	3	4	5	6	7	8
Is person BE ?	Y	N	Y	N	Y	N	Y	N
Is person Comp Sci. Grad CS ?	Y	Y	N	N	Y	Y	N	N
Work Experience >= 3	Y	Y	Y	Y	N	N	N	N
Recruit	X		X		X		X	
Don't Recruit		X		X		X		X
S/W Deptt	X				X			
HR Deptt			X				X	
TeamLeader	X							
TeamMem					X			
MgtTrainee							X	
Manager			X					
Action Stub	Action Entry							

Fig. 4.4
Decision Table

This table can further be refined by combining condition entries 2, 4, 6, and 8. The simplified table is displayed in figure 4.5.

BE ?	Y	Y	Y	Y	N
CS?	Y	N	Y	N	-
WEX >= 3	Y	Y	N	N	-
Recruit	Y	Y	Y	Y	
Don't recruit					X
S/W Deptt	Y		Y		
HR Deptt		Y		Y	
TeamLeader	Y				
TeamMem			Y		
MgtTrainee				Y	
Manager		Y			

Fig. 4.5
Simplified Decision Table

4.2.3. Structured English

Structured English is one more tool available to the analyst. It comes as an aid against the problems of ambiguous language in stating condition and actions in decisions and procedures. Here no trees or tables are employed, rather with narrative statements a procedure is described. Thus it does not show but states the decision rules. The analyst is first required to identify the conditions that occur in the process, subsequent decisions, which are to be made and the alternative actions to be taken.

Here the steps are clearly listed in the order in which they should be taken. There are no special symbols or formats involved unlike in the case of decision trees and tables, also the entire procedure can be stated quickly as only English like statements are used.

Structured English borrows heavily from structured programming as it uses logical construction and imperative statements designed to carry out instructions for actions. Using "IF", "THEN", "ELSE" and "So" statement decisions are made. In this structured description terms from the data dictionary are widely used which makes the description compact and straight.

Developing Structured Statements

Three basic types of statements are employed to describe the process.

1. Sequence Structures - A sequence structure is a single step or action included in a process. It is independent of the existence of any condition and when encountered it is always taken. Usually numerous such instructions are used together to describe a process.

2. Decision Structures - Here action sequences described are often included within decision structures that identify conditions. Therefore these structures occur when two or more actions can be taken as per the value of a specific condition. Once the condition is determined the actions are unconditional.

```
COMPUTE_DISCOUNT
See Number of Sarees
IF order is from individual
        and-if order is for 12 or more sarees
                THEN : Discount is 50%
        ELSE order is for fewer than 12 sarees
                SO: Discount is 30%
ELSE order is from shopkeeper or retailer
        SO-IF order is for 85 sarees or more
                Discount is 50%
            ELSE IF order is for 49 to 84 sarees
                        Discount is 40%
            ELSE IF order is for 48 to 13 sarees
                        Discount is 30%
            ELSE  order is for less than 12 sarees
             SO:    Discount is 15%
```

Fig. 4.6
An example of Structured English

3. Iteration Structures- these are those structures, which are repeated, in routing operations such as DO WHILE statements.

The decision structure of example discussed in previous sections(see decision table in fig 4.3) may be given in structured English as in fig 4.6

4.2.4. Data Dictionary

As the name suggests the data dictionary is a catalog or repository of data terms such as data elements, data structures etc. Data dictionary is a collection of data to be captured and stored in the system, inputs to the systems and outputs generated by the systems. So let's first know more about what are these data elements and structures.

Data element

The smallest unit of data, which can not be further decomposed, is known as a data element. For example any number digit or an alphabet will qualify to be data elements. Data element is the data at the most fundamental level. These elements are used to as building blocks for all other data in the system. At times data elements are also referred as data item, elementary item or just as field. There is a very little chance that only by them data element can convey some meaning.

Data structure

Data elements when clubbed together as a group make up a data structure. These data elements are related to one another and together they stand for some meaning. Data structures are used to define or describe the system's components.

Data dictionary entries consist of a set of details about the data used or produced in the system such as data flows, data stores and processes. For each item the dictionary records its name, description, alias and its length. The data dictionary takes its shape during the data flow analysis and its contents are used even till the system design. It very reasonable to know why the data dictionary is so essential. There are numerous important reasons.

In a system there is data volume flow in the form of reports, documents etc. In these transactions either the existing data is used or new data items are created. This poses a potential problem for the analyst and thus developing and using a well-documented dictionary can be a great help.

Now consider a case where everyone concerned with the system derives different meanings for the same data items. This problem can continue until the meaning of all data items and others are well documented so that everyone can refer to the same source and derive the same common meaning.

Documentation in data dictionary is further carried on to record about the circumstances of the various process of the system. A data dictionary is always an added advantage for the system analysis. From the data dictionary one can determine about the need of the new features or about the changes required. Thus they help in evaluating the system and in locating the errors about the system description, which takes place when the contents of the dictionary are themselves not up to the mark.

4.2.5. CASE (Computer Aided Software Engineering) Tools

A tool is any device, object or a kind of an operation used to achieve a specific task. The complete and correct description of the system is as important as the system itself. The analyst uses case tool to represent and assemble all the information and the data gathered about the system.

Most of the organizations have to follow some kind of procedures and they are required to make all sorts of decisions from time to time. For the job of the analyst both these procedures and decision-making processes of the business system under investigation are equally important.

Expressing business processes and rules in plain text is very cumbersome and difficult process. It requires a lot of effort. Moreover, it does not guarantee if the reader will understand it well. So representing these things graphically is a good choice. So CASE Tools are useful in representing business rules and procedures of organization in graphical way. It requires less effort and it is easy to understand.

Some CASE tools are designed for creating new applications and not for maintaining or enhancing existing ones hence if an organization is in a maintenance mode, it must have a CASE tool that supports the maintenance aspect of the software developed. Many a times the large projects are too big to be handled by a single analyst thus here the CASE too must be compatible enough to allow for partitioning of the project.

Efficient and better CASE tools collect a wide range of facts, diagrams, and rules, report layouts and screen designs. The CASE tool must format the collected data into a meaningful document ready to use.

Tools for DFD or a data flow diagram is a perfect example of a good CASE tool and it will be dealt with in the sessions

4.3. Fact Finding Techniques - Case Study : Library Management System

In chapter 3, we discussed how the analyst performed the preliminary analysis. But we didn't look into the actual methods that the analyst employed to gather the information about the system. In our case the analyst used on-site observations, interviewed the staff members and used questionnaires for both staff and members of the library. Now, we will see how our analyst employed these methods.

Fact Finding Techniques

On-site Observation

Our analyst wanted to see the functioning of library. So analyst visited the library for two days and observed librarian issuing and returning books. The analyst also inspected the place where the cards are stored and from that it was seen that it was a real mess. To see if a particular book is already issued, it is a difficult and effort intensive process. The analyst also saw the records for books, members, and accounts. From site visit our analyst had a good understanding of the functioning of the system. After this, the analyst performed some personal interviews of library staff and few members. In the next section we'll look at these interviews.

Interviews

Interviews are useful to gather information from individuals. Given below is the interview between the analyst and one of the librarians, during the information gathering stage of the development of our library system.

Analyst's interview with Librarian

Analyst: Hi, I have come to talk to you regarding the functioning of your library.

Librarian: Hello, do come in. I was expecting you.

Analyst: I'll come straight to the point. Don't hesitate, you can be as much open you want. There are no restrictions.

Librarian: I'll give you my whole contribution

Analyst: Tell me are you excited about the idea of having an automated system for your library?

Librarian: Yes, I do. Very much. After all it's gonna reduce our loads of work.

Analyst: Will you elaborate on it?

Librarian: Major problem is managing the cards of members. There are so many of them. Many times cards get lost. Then we have to issue a duplicate card for it. But there is a flaw in it. It is difficult to find out if it is genuinely the case. Member can lie about it so that he/she gets an extra card. And we can't do anything about it.

Analyst: What do you think be ideal solution to this?

Librarian: There should be no cards at all. All the information should be put into computer. It'll be easy for us to check how many books we have already to a particular member.

Analyst: How often you get new members?

Librarian: Very often. At about 50 to 100 members in a month. But for two months we have freezed the membership because it is already very difficult to manage the existing 250 members. But if this whole system gets computerised then we'll open the membership. From this system, the management hopes to earn huge revenues.

Analyst: Could you explain how?

Librarian: Look every month we get about 50-100 memberships requests. After the new system is built, we will open membership to our library. There is a membership fees to be paid. Management is planning to change the membership criteria. It is planning to increase fee from 400 to 500 for half yearly and 1000 for the whole year. So in this way, we plan to get huge revenues after we have an automated system.

Analyst: Do you have different member categories?

Librarian: No, we don't have any categorisation for members. All are treated at par.

Analyst: How many books are there?

Librarian: About 5000 books

Analyst: Do you people keep records for them?

Librarian: Yes.

Analyst: Do you want facility of booking a particular title in advance?

Librarian: No we don't want any such facility. It is an overhead. So we don't have any such facility presently.

Analyst: How do you categorise your books?

Librarian : By subject.

Analyst: Would you prefer online registration for users rather than the printed form?

Librarian: Yes , we really would. Sometimes we lose these forms then we don't have any information about that particular member. It will be better to have it on computer.

Analyst: Do you have any other expectation or suggestion for the new system?

Librarian: It should be able to produce reports faster.

Analyst: Reports? I completely forgot about them. What reports you people produce presently?

Librarian: Well first is for books in the library, another for members listing, one for our current supplier of books, and reports for finance.

Analyst: Do you have some format for them?

Librarian: Yes we do have and we want that the same format be used by the new system.

Analyst : Yes we'll take care of that. Any other suggestions?

Librarian: No. You have already covered all the fields.

Analyst: Thanks for your co-operation. It was nice talking to you.

Librarian: My pleasure. Bye.

Our analyst took interviews of few members of the library in order to know about their viewpoint about the new system. One of such interview is given below.

Analyst interview with one member

Venue: Reading Room

Analyst: Hello. If you are free, I need to ask you few questions.

Member: Sure. I pleasure.

Analyst: Do you know the library people are planning to have an automated system?

Member: Yes , I do and I'm feeling good about it.

Analyst: Are you ready to pay more if there is a computerised system?

Member: In the overall functioning is going to improve then I think no one will object to paying more. It should help us finding the books easily. But by what amount, it should matter.

Analyst: Well as far as I know they are planning to hike the membership fee from 400 to 500 for half year and 1000 for full year.

Member: That would be too much. Then in that case, they should increase the number of books to be issued. Also the number of days a book can be kept by member should also be increased.

Analyst: What you do think, how much books to be allowed for issue and for how many days.

Member: Well these people should increase number of books from 3 to at least 4. And the number of days for which a book is kept should be increased by 4 days. Presently it is for 10 days. It should be 14 days. Only then the fee
hike will be justified.

Analyst: Yes, they have such plans.

Member: Then it should not bother members.

Analyst: Are you keen on online registration of members instead of normal paper one?

Member: Yes. It'll be a good practice.

Analyst: Should there be a facility to reserve a book in advance?

Member: Presently they have many copies of a single title. Usually a book is always available. I never have felt the need to reserve a book in advance.

Analyst: On what basis a book should be categorised?

Member: Well, it should be on the basis of subject.

Analyst: What do you think on what basis a search for a particular book can be done?

Member: It can be searched using subject or title.

Analyst: How often you visit this library?

Member: Daily

Analyst: Do you think magazines and cassettes should be made available in the library?

Member: I think it's a good idea.

Analyst: Do you like this library?

Member: Yes, very much. That's why I come here daily.

Analyst: Have you ever recommended this library to your friends, relatives, or to your acquaintances?

Member: Yes I very often do.

Analyst: Till now, to how many you have recommended?

Member: About 30 people.

Analyst: And how many of them have actually become its members?

Member: 25 people.

Analyst: That's really nice. People actually take your recommendation very seriously. Thank You. It was nice talking to you.

Member: Thank You.

After interviewing different people, analyst got to know about their opinions.

Questionnaires

Since the time was less it was not practical to interview every library staff. So to get the opinion of all staff, the analyst distributed questionnaires to all of them.

The questionnaire given to library staff is given below.

Instructions: Answer as specified by the format. Put NA for non-applicable situation.

1. What are your expectations out of the new system (computer based)? Rate the following on a scale of 1-4 giving a low value for low priority.

a) better cataloguing
b) better managing of users
c) better account and books management
d) computer awareness
e) any other _____

2. How many users are you expecting?
_____

3. How many books are there in library?
_____

4. How you want the books to be categorized for searching (like by title, author name or by subject) ?
_____
_____
_____

5. Is there any difference in the roles (privileges) of two members?
Yes\No Please specify if Yes
_____
_____

6. Do you want facility of booking a title in advance
Yes\No

7. Do you have data of book entered into some kind of database?
Yes\No

8. How do you want users to be categorized?
_____ or
_____

9. Would you like online registration for users rather than printed form?
Yes/No

10. Do you already have some existing categorization of books on the basis as specified in question 4 above
Yes/ No

11. Any other specific suggestion / expectation out of the proposed system.
_____
_____

Questionnaire for library staff

In order to get the views of the existing members, the analyst also distributed questionnaires to the member also. The questionnaires used by analyst for the library members is shown in fig 4.8 .

Instruction: Answer as specified by the format. Put NA for non-applicable situation.

1. Are you willing to pay extra for a library if it is fully computerized and eases finding of book, etc.
Yes\No
_____ (if Yes, how much extra are you willing to pay)

2. What you feel should be necessary for a book to be searched?
(by topic, by title, by author ,….)
_____
_____
_____
_____

3. Are you keen on online registration instead of the normal paper one.
Yes/No

4. How many titles do you feel should be issued to a single member?
_____

5. What should be the maximum duration for the issue of certain book to a member?
_____ Days.

6. Should there be a facility to reserve a book in advance?
Yes/No

7. How often do you visit the library? Choose One.
a) daily
b) once in two days
c) weekly
d) bi-weekly
e) monthly

8. Should there be a facility to reserve a book on phone?
Yes/No

9. Should magazines and cassettes be included in the library
Yes/No

10. Do you recommend this library to your friends, relatives, or acquaintances?
Yes/No (if yes, to how many you recommended and out of them how many actually became the members)

Recommended :_____ Became Members : _____

Now we'll look at the techniques that the analyst employed to document the various business rules of the library.

Analyst identified the following business rules.

1) Procedure for becoming a member of library.

Anyone whose age is 18 or more than that can become a member of library. There are two types of memberships depending upon the duration of membership. First is for 6 months and other is for 1 year. 6 months membership fee is Rs 500 and 1 year membership fee is Rs 1000.

The decision tree illustrating the business rule is given below.

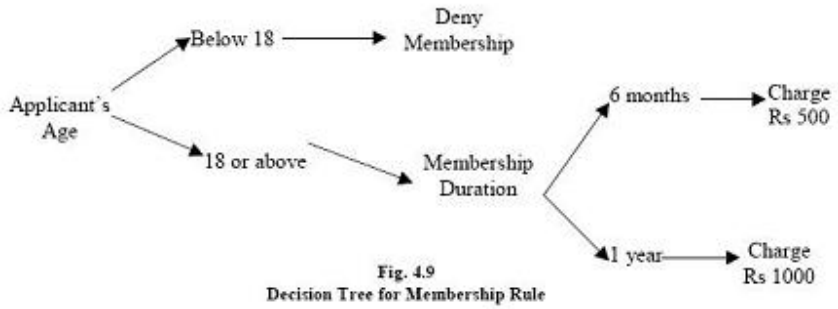


Fig. 4.9
Decision Tree for Membership Rule

Is Age < 18	***Y***		
Age > = 18		***Y***	***Y***
Is Memebership for 6 months ?		***Y***	
Is Memebership for 12 months ?			***Y***
Grant Membership		***X***	***X***
Deny Membership	***X***		
Charge Membership Rs. 500		***X***	
Charge Membership Rs. 1000			***X***

Fig 4.10

Decision table for membership rule

Rule for Issuing Books

If the number of books already issued is equal to 4 then no more books is issued to that member. If it is less than 4 then that book is issued.

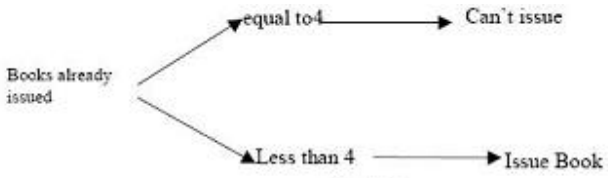


Fig. 4.11
Decision Tree for Issue of Books

Are Book Already Issued= 4	Y	
Are books Already issued <4		Y
Don't Issue	X	
Issue		X

Fig. 4.12
Decision table for Book Issue rule

Returning Books

Whenever a member returns a book, it is checked if the book is being returned after the due return date. If this is the case, then a fine of Rs 2 per day after the return date is charged. If the book is returned on the due date or before that, then no fine is charged.

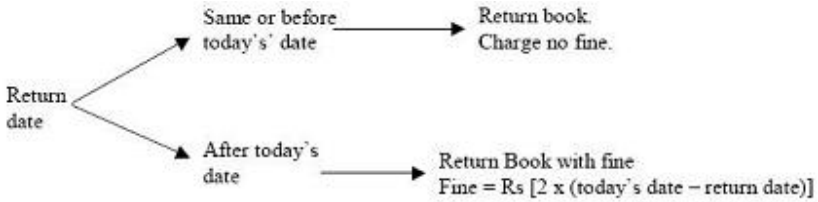


Fig. 4.13
Decision Tree for Return of Books

Is Return Date <= Today's date	Y	
Is Return Date > Today's date		Y
Return book without charging any fine	X	
Return book with fine Fine = Rs 2 x [Today's date – Return Date]		X

Fig. 4.14
Decision table for return of books

Now the analyst has a good understanding of the requirements for the new system, we can move to the designing. Design of the system will be discussed in the later chapters.

4.4. Fact Finding Techniques - Self Assessment

1. What are the two types of interviews used in fact finding techniques?
2. How many types of questionnaires are used? Compare them.

State True or False

1. In open response questionnaires, the respondent is free to answer in his/her own words.
2. Decision tree is two dimensional matrix showing different conditions and their respective actions taken.

4.5. Fact Finding Techniques - Summary

Key points covered in this chapter are:

" Fact finding techniques are very useful during the analysis phase. There are many fact finding techniques available like interviews, questionnaires, on site observation etc.

" Each of the technique has its own format, suitability, merits and demerits. The analyst has to make a judgement on which format to be used to collect what information and who should be the respondents.

" Decision-making is an integral part of any business setup. To facilitate the decision making process, which is based on the possibilities of various conditions and their combination there, arises a need to document them that will make the whole process easier.

" First step in documenting is the identification of the systems conditions and actions. Then the available tools are used. The three tools available are decision trees, decision tables and Structured English.

" In the decision trees the decision variables are represented in a tree's branch structure. The conditions are depicted sequentially and the root of the tree is the starting point.

" Decision tables links the conditions and actions by using decision rules, which state all the conditions that must be true for a particular set of actions, are taken.

" Structured English is employed to state decision rules and it has three kinds of statements namely sequence, decision and iteration structures. This format is very concise and compact and also very easy to understand.

4.6. Fact Finding Techniques - Exercises

1. What type information is often best obtained through interview?

2. Which method does an analyst use to gather data?

3. What is fact-finding?

4. Make a list of dos and don'ts that you should follow when interviewing a user.

5. List different types of questionnaire formats.

6. What are the differences between open and closed-form questionnaires? For what types of information is each form most useful?

7. When is record review an effective data collection method? What is the purpose of this method?

8. What role does observation play in system investigation?

9. What are conditions and actions? What are their roles in decision analysis?

10. In what way do decision trees assist in decision analysis? Explain how an analyst should develop a decision tree.

11. What advantages do decision trees present for analysis?

12. How does the purpose of decision tables differ from that of decision trees? What components make up a decision tables?

13. How do analysts develop a decision table?

14. How a decision table is developed?

15. How does structured English differ from the decision tree and decision table? What advantages does it offer over the other two methods?

16. Summarize the advantages and limitations of interviews and questionnaires.

17. Explain the differences between

(a) structured and unstructured interviewing and

(b) open-ended and closed questions. Give an example of each.

Chapter 5: Functional Modeling I

At the end of this chapter you will be able to know about functional modeling of system. You will also be able to know about the design elements of the system.

5.1. Functional Requirements

After the thorough analysis of the system is done, the design process for the proposed system starts. Designing of the system includes making the conceptual layout of the system using various modeling techniques available and coding the proposed system for actual implementation.

The first step after the analysis of the system is making the conceptual design. Conceptual design consists of two elements: data model and functional models.

Data modeling is a technique for organizing and documenting system's data. Process model is a technique for organizing and documenting a system's processes, inputs, outputs and data stores. Generally, these models give the following information.

- What all processes make up a system?
- What data are used in each process?
- What data are stored?
- What data enter and leave the system?

Information is transformed as it flows through a computer-based system. As we already know, information transformation basically consists of: input, process and output. The system accepts input in variety of forms; applies hardware, software, and human elements to transform input into output; and produces output in a variety of forms. The transform(s) may comprise a single logical comparison, a complex numerical algorithm or rule-inference approach of an expert system. We can create a flow model for any computer-based system, regardless of size and complexity.

By flow models, we get to know the functionality of a system.

Data flow diagram is one of the tools used in the analysis phase. Data flow diagram is a graphical tool used to analyze the movement of data through a system-manual or automated-including the processes, stores of data, and delay in the system. The transformation of data from input to output, through processes, may be described logically and independently of the physical components (for example, computers, file cabinets, disk units, and word processors).

Structure chart is another tool used in designing phase of the life cycle.

Design Elements

This section describes the various design elements. These include

1. Modules
2. Processes
3. Input
4. Output
5. Files
6. Databases

5.1.1. Modules

As discussed in **lesson 1**, a large system actually consists of various small independent subsystems that combine together to build up the large systems. While designing the system too, the complete system is divided into small independent modules which may further be divided if the need be. Such independent modules are designed and coded separately and are later combined together to make the system functional.

For the better understanding and design of the system, it should be made as a hierarchy of modules. Lower level modules are generally smaller in scope and size compared to higher level modules and serve to partition processes into separate functions. Following factors should be considered while working on modules:

Size:

The number of instructions contained in a module should be limited so that module size is generally small.

Shared use:

Functions should not be duplicated in separate modules, but established in a single module that can be invoked by any other module when needed.

5.1.2. Processes

As already discussed, a system consists of many subsystems working in close coordination to achieve a specific objective. Each of these subsystems carries out a specific function and each of these functions may in turn be consisting of one or more processes.

Thus the system's functions can be subdivided into processes, as depicted by fig. 5.1. A process is a specific act that has definable beginning and ending points. A process has identifiable inputs and outputs. Create purchase requisition, follow up order etc are few examples of processes. For designing of any system, these processes need to be identified
as a part of functional modeling.

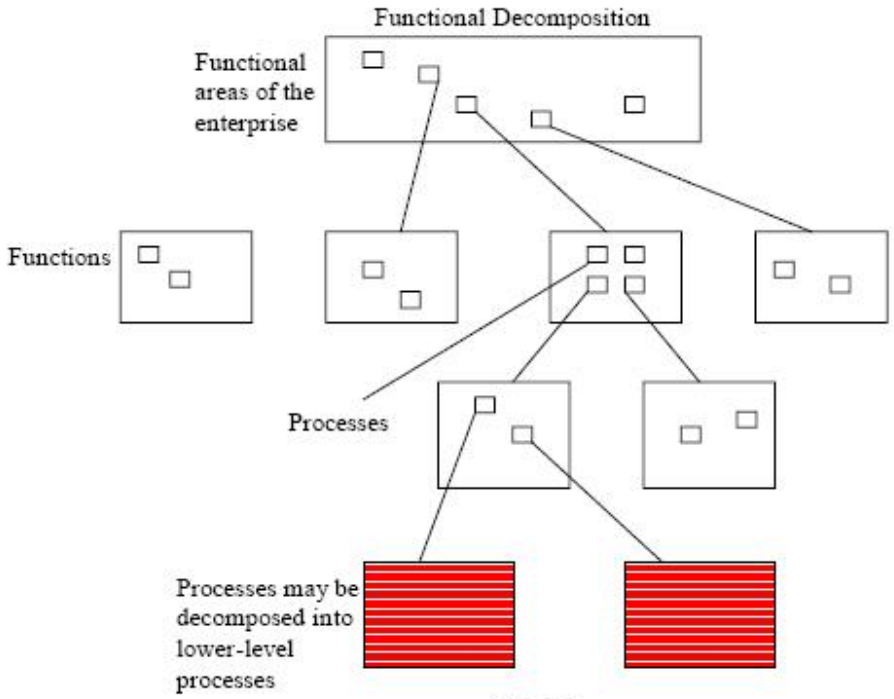


Fig 5.1
Functional Decomposition
Source : Information Engineering: Planning & Analysis by James Martin

Every process may be different from the other but each of them has certain common characteristics, as:

- A process is a specified activity in an enterprise that is executed repeatedly. This means that the processes are ongoing, for example, generation of bills may be labeled as a process for a warehouse as it is repeatedly carried out
- A process can be described in terms of inputs and outputs. Every process would have certain inputs required which are transformed into a certain output. For example, in case of a warehouse, information related to the sale of various items is required for generation of bills. This information is taken as input and the bills generated are the output of the process.
- A process has definable starting and ending points.
- A process is not based on organizational structures and is carried out irrespective of this structure.
- A process identifies what is done, not how.

5.1.3. Input(s) and Output(s)

As discussed earlier, inputs and outputs are an important part of any system, so while designing a system inputs and outputs of the system as a whole need to be identified and the inputs and outputs for the various processes of the system need to be listed down.

During design of input, the analyst should decide on the following details:

- What data to input
- What medium to use
- How data should be arranged
- How data should be coded i.e. data representation conventions
- The dialogue to guide users in providing input i.e. informative messages that should be provided when the user is entering data. Like saying, "It is required. Don't leave it blank."
- Data items and transactions needing validation to detect errors

- Methods for performing input validation and steps to follow when errors occur

The design decisions for handling input specify how data are accepted for computer processing. The design of inputs also includes specifying the means by which end-users and system operators direct the system in performing actions.

Output refers to the results and information that are generated by the system. In many cases, output is the main reason for developing the system and the basis on which the usefulness of the system is evaluated. Most end-users will not actually operate the information system or enter data through workstations, but they will use the output from the system.

While designing the output of system, the following factors should be considered:

- Determine what information to present
- Decide on the mode of output, i.e. whether to display, print, or "speak" the information and select the output medium
- Arrange the presentation of information in an acceptable format
- Decide how to distribute the output to intended recipients

These activities require specific decisions, such as whether to use preprinted forms when preparing reports and documents, how many lines to plan on a printed page, or whether to use graphics and color. The output design is specified on layout forms, sheets that describe the location characteristics (such as length and type), and format of the column heading, etc.

5.1.4. Design of Databases and Files

Once the analyst has decided onto the basic processes and inputs and outputs of the system, he also has to decide upon the data to be maintained by the system and for the system. The data is maintained in the form of data stores, which actually comprise of databases.

Each database may further be composed of several files where the data is actually stored. The analyst, during the design of the system, decides onto the various file-relating issues before the actual development of the system starts.

The design of files includes decisions about the nature and content of the file itself such as whether it is to be used for storing transaction details, historical data, or reference information.

Following decisions are made during file design:

- Which data items to include in a record format within the file?
- Length of each record, based on the characteristics of the data items
- The sequencing or arrangement of records within the file (the storage structure, such as sequential, indexed, or relative)

In database design, the analyst decides upon the database model to be implemented.Database model can be traditional file based, relational, network, hierarchical, or object oriented database model. These data models are discussed in detail in lesson 7 on Data Modeling.

5.1.5. Interface Designing

Systems are designed for human beings to make their work simpler and faster. Hence interaction of any system with the human being should be an important area of concern for any system analyst. The analyst should be careful enough to design the human element of the system in such a manner that the end user finds the system friendly to work with. Interface design implies deciding upon the human computer interfaces. How the end user or the operator will interact with the system. It includes designing screens,menus, etc.

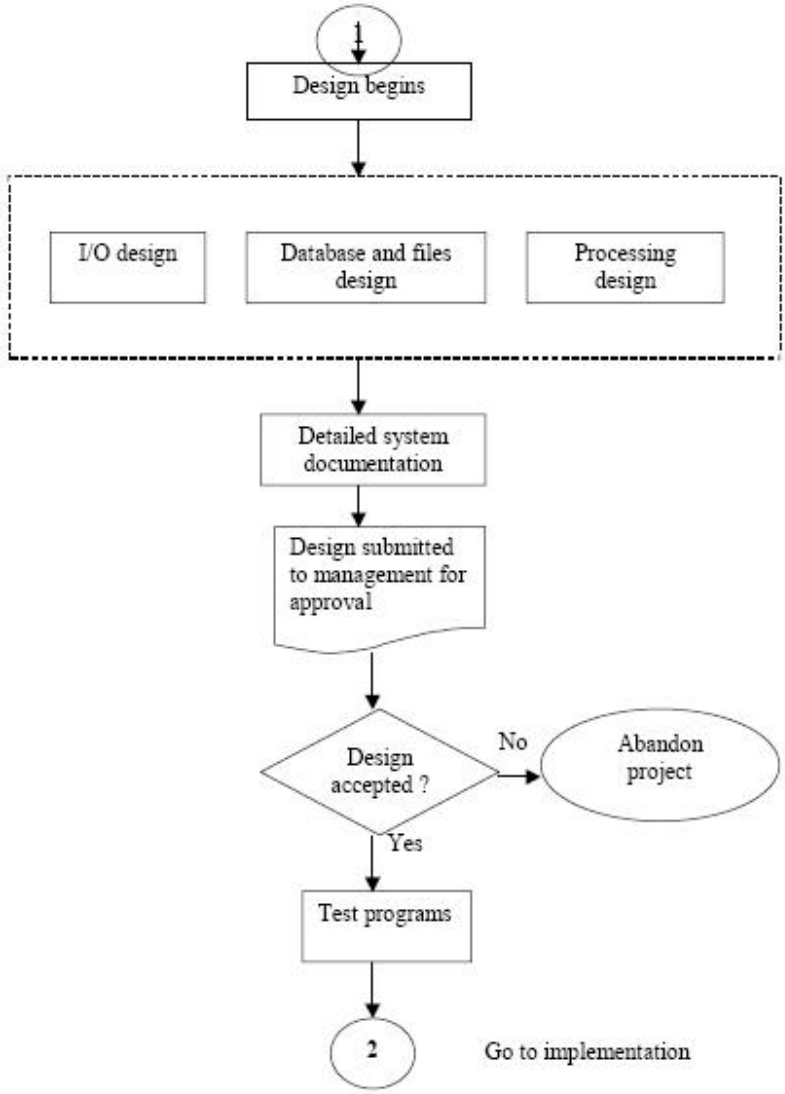


Fig 5.2 Basic Steps in system design

The following factors should be considered while working on interfaces.

Use of a consistent format for menu, command input, and data display.

Provide the user with visual and auditory feedback to ensure that two-way communication is established.

Provide undo or reversal functions.

Reduce the amount of information that must be memorized between actions.

Provide help facilities that are context sensitive.

Use simple action verbs or short verb phrases to name commands.

Display only that information that is relevant to the current context.

Produce meaningful error messages.

Use upper and lower case, indentation, and text grouping to aid in understanding.

Produce meaningful error messages.

Maintain consistency between information display and data input. The visual characteristics of the display (e.g., text size, color, and placement) should be carried over to the input domain.

Interaction should be flexible but also tuned to user's preferred mode of input.

Deactivate commands that are inappropriate in the context of current actions.

Provide help to assist with all input actions.

5.2. Functional Modeling Techniques

Now that we are familiar with the various design elements, let us take a look at the modeling techniques that are used for designing the systems. Data Flow Diagrams are used for functional modeling. As the name suggests, it is a diagram depicting the flow of data through the system. In the next section, we'll explore this technique in detail.

5.2.1. Data Flow Diagrams

A Data Flow Diagram (DFD) is used to describe the logical operation of a system, i.e. what a system does. DFD shows the flow of data through a system and the work or processing performed by that system.

DFDs only represent the flow of data through the system and do not describe the physical functioning of the system. DFDs are basically of 2 types: Physical and logical ones.

Physical DFDs are used in the analysis phase to study the functioning of the current system. Logical DFDs are used in the design phase for depicting the flow of data in proposed system.

5.2.2. Elements of Data Flow Diagrams

Data Flow Diagrams are composed of the four basic symbols shown below.

The External Entity symbol represents sources of data to the system or destinations of data from the system. Refer fig 5.11

The Data Flow symbol represents movement of data. Refer fig 5.10

The Data Store symbol represents data that is not moving (delayed data at rest). Refer fig 5.12

The Process symbol represents an activity that transforms or manipulates the data (combines, reorders, converts, etc.). Refer fig 5.9

Any system can be represented at any level of detail by these four symbols.

5.2.3. Processes

Processes are work or actions performed on incoming data flows to produce outgoing data flows. These show data transformation or change. Data coming into a process must be "worked on" or transformed in some way. Thus, all processes must have inputs and outputs. In some (rare) cases,

data inputs or outputs will only be shown at more detailed levels of the diagrams. Each process in always "running" and ready to accept data.

Major functions of processes are computations and making decisions. Each process may have dramatically different timing: yearly, weekly, daily.

Naming Processes

Processes are named with one carefully chosen verb and an object of the verb. There is no subject. Name is not to include the word "process". Each process should represent one function or action. If there is an "and" in the name, you likely have more than one function (and process). For example, get invoice ,update customer and create Order Processes are numbered within the diagram as convenient. Levels of detail are shown by decimal notation. For example, top level process would be Process 14, next level of detail Processes 14.1-14.4, and next level with Processes 14.3.1-14.3.6. Processes should generally move from top to bottom and left to right.

5.2.4. External Entities

External entities determine the system boundary. They are external to the system being studied. They are often beyond the area of influence of the developer.

These can represent another system or subsystem. These go on margins/edges of data flow diagram. External entities are named with appropriate name.

5.2.5. Data Flow

Data flow represents the input (or output) of data to (or from) a process ("data in motion"). Data flows only data, not control. Represent the minimum essential data the process needs. Using only the minimum essential data reduces the dependence between processes. Data flows must begin and/or end at a process.

Data flows are always named. Name is not to include the word "data". Should be given unique names. Names should be some identifying noun. For example, order, payment, complaint.

5.2.6. Data Stores

Data Stores are repository for data that are temporarily or permanently recorded within the system. It is an "inventory" of data. These are common link between data and process models. Only processes may connect with data stores.

There can be two or more systems that share a data store. This can occur in the case of one system updating the data store, while the other system only accesses the data.

Data stores are named with an appropriate name, not to include the word "file", Names should consist of plural nouns describing the collection of data. Like customers, orders, and products. These may be duplicated. These are detailed in the data dictionary or with data description diagrams.

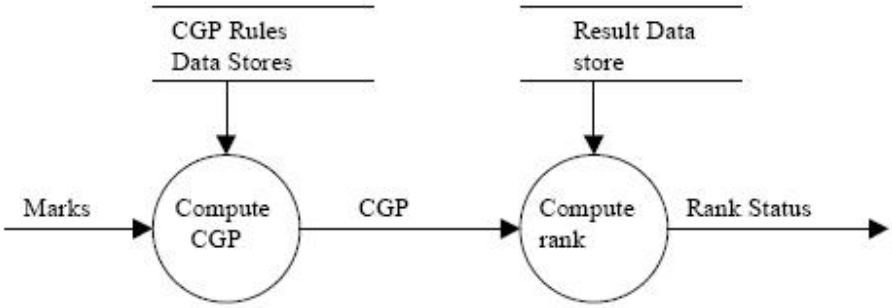


Fig: 5.8
DFD showing rank calculation process of a university

5.2.7. Different Levels of DFDs

The DFD may be used to represent a system or software at any level of abstraction. In fact, DFDs may be partitioned into levels that represent increasing information flow and functional details.

A 0 level DFD, also called a context model, represents the entire software elements as a single bubble with input and output data indicated by incoming and outgoing arrows, respectively. Additional processes and information flow paths are represented as the 0 level DFD is partitioned to reveal more information. For example, a 1 level DFD might contain five or six bubbles with interconnecting arrows.

Making DFDs (data Flow Diagrams)
Data Flow Diagramming is a means of representing a system at any level of detail with a graphic network of symbols showing data flows, data stores, data processes, and data sources/destinations.

The goal of data flow diagramming is to have a commonly understood model of a system. The diagrams are the basis of structured systems analysis. Data flow diagrams are supported by other techniques of structured systems analysis such as data structure diagrams, data dictionaries, and procedure-representing techniques such as decision
tables, decision trees, and structured English.

The purpose of data flow diagrams is to provide a semantic bridge between users and systems developers. The diagrams are graphical, eliminating thousands of words. These are logical representations, modeling what a system does, rather than physical models showing how it does it. DFDs are hierarchical, showing systems at any level of detail. Finally, it should be jargonless, allowing user understanding and reviewing.

Also, data flow diagrams have the objective of avoiding the cost of:

- user/developer misunderstanding of a system, resulting in a need to redo systems or in not using the system.
- having to start documentation from scratch when the physical system changes since the logical system, WHAT gets done, often remains the same when technology changes.
- systems inefficiencies because a system gets "computerized" before it gets "systematized".
- being unable to evaluate system project boundaries or degree of automation, resulting in a project of inappropriate scope.

Notation

Data flow analysis was developed and promoted simultaneously by two organizations. Yourdon,Inc., a consulting firm, has vigorously promoted the method publicly. Mc Donnell-Douglas, through the work and writings of Gane and Sarson, has also influenced the popularity of data flow analysis.

Data flow diagram can be completed using only four simple notations. The use of specific icons associated with each element depends on whether the Yourdon or Gane and Sarson approach is used.

1) People, procedures, or devices that use or produce(transform) data. The physical component is not identified Yourdon Gane and Sarson

Fig. 5.9
Process or transform

2) Data flow: Data move in a specific direction from origin to a destination in the form of a document, letter, telephone call, or virtually any other medium. The data flow is a "packet" of data.

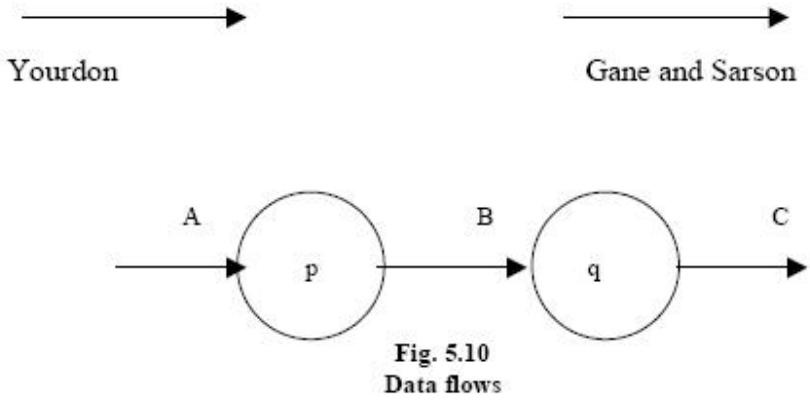


Fig. 5.10
Data flows

3) Source or destination of data: external sources or destination of data, which may be people, programs, organizations, or other entities, interact with the system but are outside its boundary. The term source and sink are interchangeable with origin and destination.

Fig. 5.11
Source or destination of data (external)

4) Here data are stored or referenced by a process in the system. The data store may represent computerized or non-computerized devices.

Fig. 5.12
Data stores

Each component in a data flow diagram is labeled with a descriptive name. Process names are further identified with a number that will be used for identification purposes. The number assigned to a specific process does not represent the sequence of processes.

Multiple input data streams and multiple data output streams are possible.

If two adjacently placed input are both required then a star sign * is placed between these.

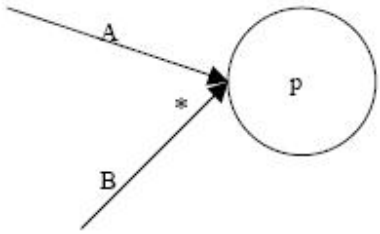


Fig. 5.13
When two inputs are required for a transform: star sign

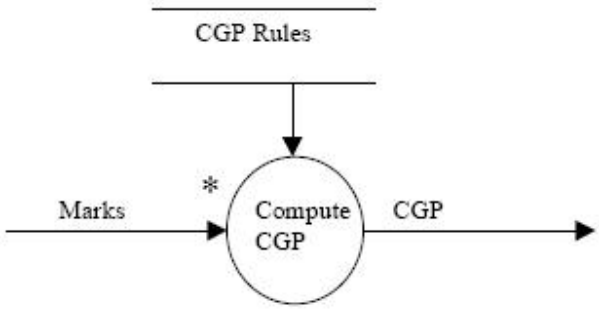


Fig 5.14
DFD showing when two inputs are both required.

If either of two adjacent placed inputs then a ringsum is placed between these.

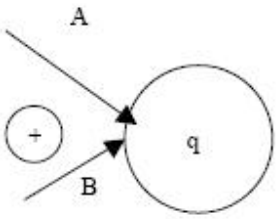


Fig. 5.15
When either of two inputs is required: Ring Sum

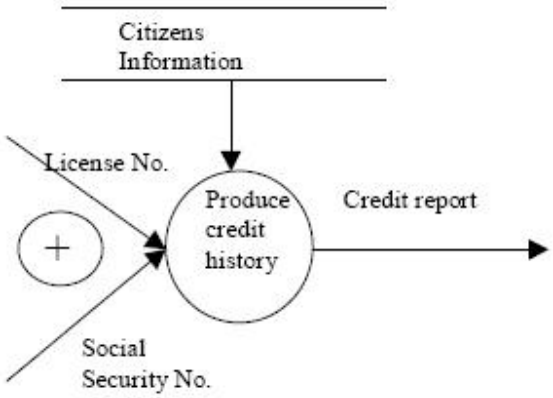


Fig 5.16
DFD showing when either of two inputs is required.

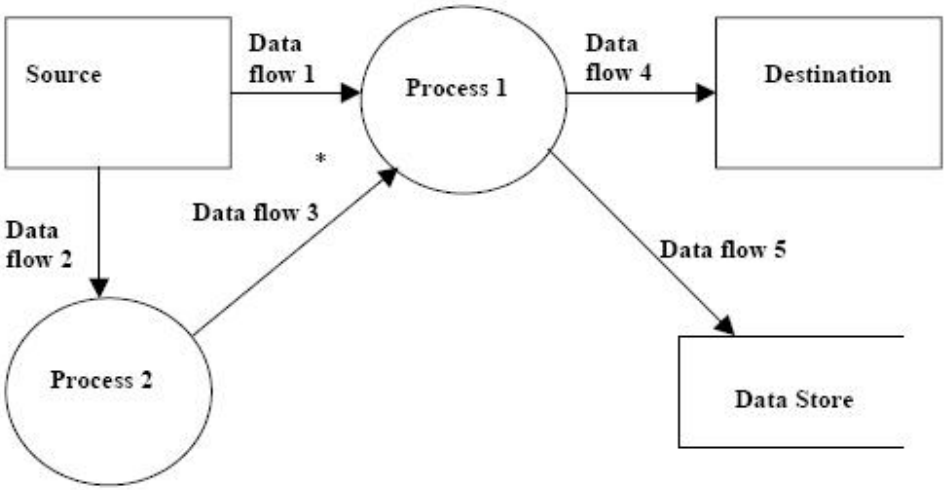


Fig. 5.17
Data Flow diagram using Yourdon notation

The procedure for producing a data flow diagram is to:

- identify and list external entities providing inputs/receiving outputs from system;

-identify and list inputs from/outputs to external entities;

Draw a context DFD

>> Defines the scope and boundary for the system and project

1. Think of the system as a container (black box)
2. Ignore the inner workings of the container
3. Ask end-users for the events the system must respond to
4. For each event, ask end-users what responses must be produced by the system
5. Identify any external data stores
6. Draw the context diagram

i. Use only one process
ii. Only show those data flows that represent the main objective or most common inputs/outputs

- identify the business functions included within the system boundary;
- identify the data connections between business functions;
- confirm through personal contact sent data is received and vice-versa;
- trace and record what happens to each of the data flows entering the system (data movement, data storage, data transformation/processing)
- Draw an overview DFD
  >>Shows the major subsystems and how they interact with one another
  >>Exploding processes should add detail while retaining the essence of the details from the more general diagram
  >>Consolidate all data stores into a composite data store
- Draw middle-level DFDs
  >>Explode the composite processes
- Draw primitive-level DFDs
  >>Detail the primitive processes
  >>Must show all appropriate primitive data stores and data flows
- verify all data flows have a source and destination;
- verify data coming out of a data store goes in;
- review with "informed";
- explode and repeat above steps as needed.

Balancing DFDs

- Balancing: child diagrams must maintain a balance in data content with their parent processes
- Can be achieved by either:
- exactly the same data flows of the parent process enter and leave the child diagram, or
- the same net contents from the parent process serve as the initial inputs and final outputs for the child diagram or
- the data in the parent diagram is split in the child diagram

Rules for Drawing DFDs

- A process must have at least one input and one output data flow
- A process begins to perform its tasks as soon as it receives the necessary input data flows
- A primitive process performs a single well-defined function
- Never label a process with an IF-THEN statement
- Never show time dependency directly on a DFD
- Be sure that data stores, data flows, data processes have descriptive titles. Processes should use imperative verbs to project action.
- All processes receive and generate at least one data flow.
- Begin/end data flows with a bubble.

Rules for Data Flows

1. A data store must always be connected to a process
2. Data flows must be named
3. Data flows are named using nouns
   " Customer ID, Student information
4. Data that travel together should be one data flow
5. Data should be sent only to the processes that need the data

Use the following additional guidelines when drawing DFDs.

Identify the key processing steps in a system. A processing step is an activity that transforms one piece of data into another form.

Process bubbles should be arranged from top left to bottom right of page.

Number each process (1.0, 2.0, etc). Also name the process with a verb that describes the information processing activity.

Name each data flow with a noun that describes the information going into and out of a process. What goes in should be different from what comes out.

Data stores, sources and destinations are also named with nouns.

Realize that the highest level DFD is the context diagram. It summarizes the entire system as one bubble and shows the inputs and outputs to a system

Each lower level DFD must balance with its higher level DFD. This means that no inputs and outputs are changed.

Think of data flow not control flow. Data flows are pathways for data. Think about what data is needed to perform a process or update a data store. A data flow diagram is not a flowchart and should not have loops or transfer of control. Think about the data flows, data processes, and data storage that are needed to move a data structure through a
system.

Do not try to put everything you know on the data flow diagram. The diagram should serve as index and outline. The index/outline will be "fleshed out" in the data dictionary, data structure diagrams, and procedure specification techniques.

Examples

1. Students wish to register for courses. Some courses have a prerequisite, which must be satisfied. A student must take the compulsory courses of her/his department and may opt for a couple of elective courses. Make a DFD.

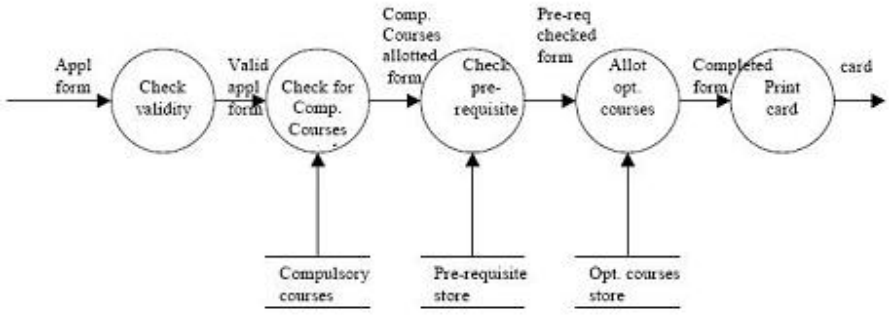


Fig. 5.18
DFD for a registration for university's new semester

2. A student can withdraw within 20 days of registration. Enhance the above DFD.

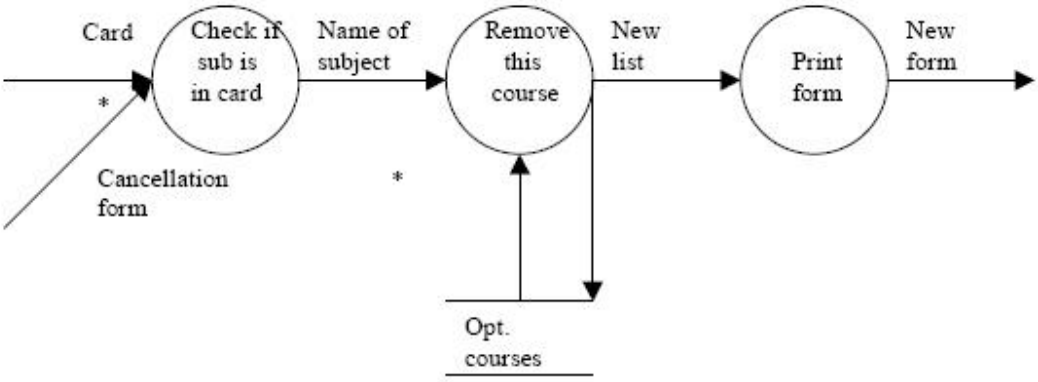


Fig. 5.19
DFD when the student has the option of withdrawing within 20 days

Exercise : Railway Reservation System

Railway caters to the need of passenger. Tickets can be booked for different classes. Some concessions are available for categories like students, old people, etc. there is a special category for reserving tickets for handicapped people, military, MPs. There is a certain surcharge levied for special trains. Fare computation depends on the class, category, train, and distance. The passenger is issued a confirmed ticket if seat is available. She/he gets a waitlisted/RAC ticket if she/he so desires.

Draw a DFD.

5.3. Functional Modeling I - Self Assessment

1. What is the difference between a context and leveled data flow diagrams?

2. What level number and name do we assign to the original data flow diagram?

3. If a process is numbered 1.5.3.1, what does this mean?

4. Match the following:

1. Circle	a. Source of data
2. Square	b. File or data store
3. Arrow	c. Conversion process
4. Parallel	d. Data flow

5. Draw a context data flow diagram of a payroll system.

5.4. Functional Modeling I - Summary

Key points covered in this chapter are:

- Data flow analysis concentrate on the following activities.

  What processes make up the system?
  What data are used in each process?
  What data are stored?
  What data enter and leave the system

- Tools for analysis and design

Data flow diagram: to show the flow of data through the system.
Data dictionary: to keep logical characteristics of the data of the system.

- Data stores
- Name
- Description
- Aliases
- Contents
- Types

5.5. Functional Modeling I - Exercises

1. Draw a context level model(level 0 DFD) for five systems with which you are familiar. The system need not be computer based.

2. Using the systems described in previous problem, refine each into a level 1 and level 2 DFDs.

3. What would a simple data flow diagram for a payroll system look like?

4. Discuss between logical and physical views of the system. Which view is included in a data flow diagram? Why?

Chapter 6: Functional Modeling II

At the end of this chapter you will able to know about the modular design of the system. You will also be able to know how to make structure charts.

DFDs pay a major role in designing of the software and also provide the basis for other design-related issues. Some of these issues are addressed in this chapter. All the basic elements of DFD are further addressed in the designing phase of the development procedure

6.1. Process Specification (PSPEC)

A process specification (PSPEC) can be used to specify the processing details implied by a bubble within a DFD.

The process specification describes the input to a function, the algorithm, the PSPEC indicates restrictions and limitations imposed on the process (function), performance characteristics that are relevant to the process, and design constraints that may influence the way in which the process will be implemented. In other words, process specification is used to describe the inner workings of a process represented in a flow diagram.

6.1.1. Control Flow Model

The Hatley and Pirbhai extensions focus on the representation and specification of the control-oriented aspects of the software. Moreover, there exists a large class of applications that are driven by events rather than data that produce control information rather than reports or displays, and that process information with heavy concern for time and performance.

Such an application requires the use of control flow modeling in addition to data flow modeling. For this purpose, control flow diagram is created. The CFD contain the same processes as the DFD, but shows control rather than data flow.

Control flow diagrams show how events flow among processes and illustrate those external events that cause various processes to be activated. The relationship between process and control model is shown in fig. 6.1in section 6.3.

Drawing a control flow model is similar to drawing a data flow diagram. A data flow model is stripped of all data flow arrows. Events and control items are then added to the diagram a "window" (a vertical bar) into the control specification is shown.

6.1.2. Control Specifications (CSPEC)

The CSPEC is used to indicate (1) how the software behaves when an event or control signal is sensed and (2) which processes are invoked as a consequence of the occurrence of the event. The control specification (CSPEC) contains a number of important modeling tools.

The control specification represents the behavior of the system in two ways. The CSPEC contains a state transition diagram that is sequential specification of behavior. It also contains a process activation table (PAT) -a combinatorial specification of behavior.

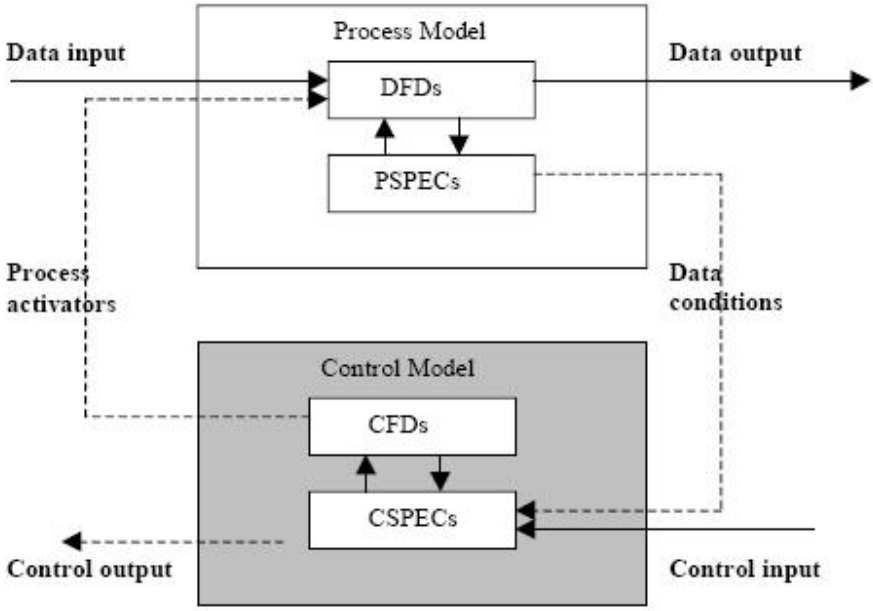


Fig. 6.1
The relationship between data and control models

6.1.3. Structure Charts

Once the flow of data and control in the system is decided using tools like DFDs and CFDs, the system is given shape through programming. Prior to this, the basic infrastructure of the program layout is prepared based on the concepts of modular programming.

In modular programming, the complete system is coded as small independent interacting modules. Each module is aimed at doing one specific task. The design for these modules is prepared in the form of structure charts.

A structure chart is a design tool that pictorially shows the relation between processing modules in computer software. Describes the hierarchy of components modules and the data are transmitted between them. Includes analysis of input-to-output transformations and analysis of transaction.

Structure charts show the relation of processing modules in computer software. It is a design tool that visually displays the relationships between program modules. It shows which module within a system interacts and graphically depicts the data that are communicated between various modules.

Structure charts are developed prior to the writing of program code. They identify the data passes existing between individual modules that interact with one another.

They are not intended to express procedural logic. This task is left to flowcharts and pseudocode. They don't describe the actual physical interface between processing functions.

Notation

Program modules are identified by rectangles with the module name written inside the rectangle.

Arrows indicate calls, which are any mechanism used to invoke a particular module.

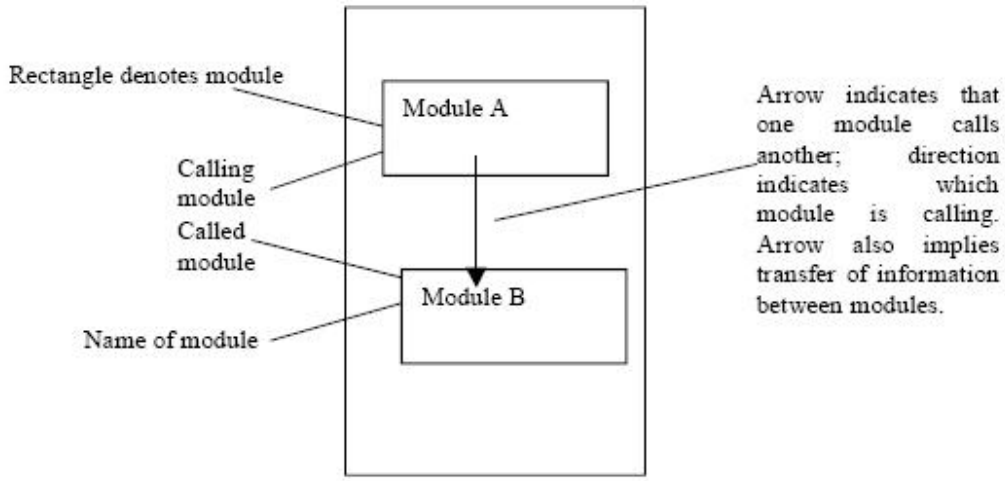


Fig. 6.2
Notation used in structure charts.

Annotations on the structure chart indicate the parameter that are passed and the direction of the data movement. In fig. 6.3, we see that modules A and B interact. Data identified as X and Y are passed to module B, which in turn passes back Z.

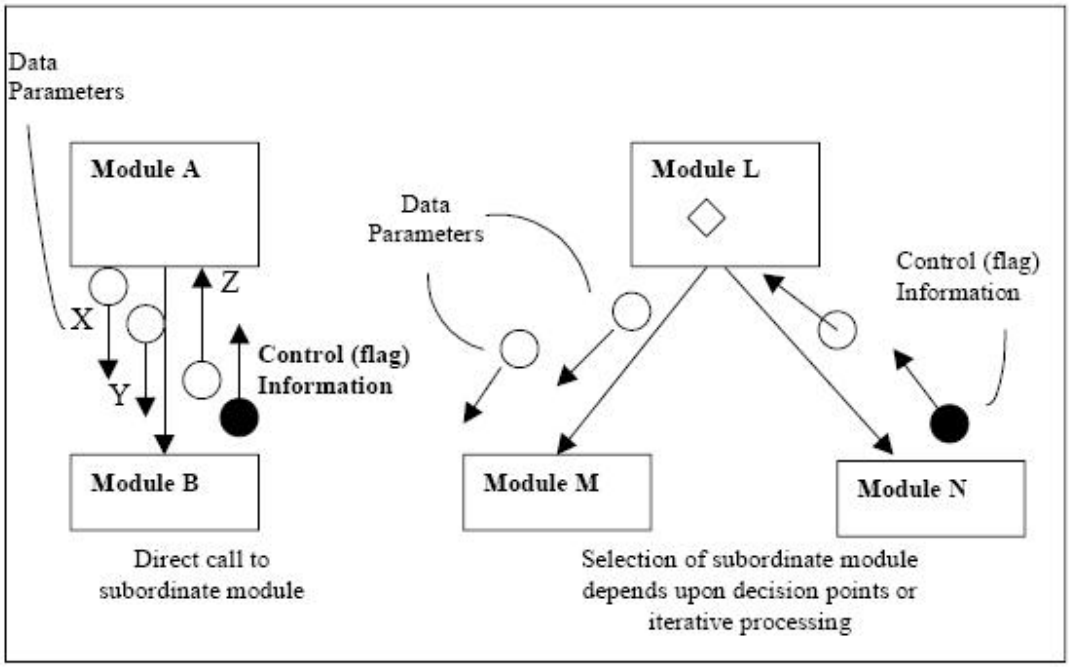


Fig. 6.3
Annotations and data passing in structure charts.

A calling module can interact with more than one subordinate module. Fig. 6.3 also shows module L calling subordinate modules M and N. M is called on the basis of a decision point in L (indicated by the diamond notation), while N is called on the basis of the iterative processing loop (noted by the arc at the start of the calling arrow.

Data passing

When one module calls another, the calling module can send data to the called module so that it can perform the function described in its name. The called module can produce data that are passed back to the calling module.

Two types of data are transmitted. The first, parameter data, are items of data needed in the called module to perform the necessary work. A small arrow with an open circle at the end is used to note the passing of data parameters. In addition, control information (flag data) is also passed. Its purpose is to assist in the control of processing by indicating the occurrence of, say, errors or end-of-conditions. A small arrow with a closed circle indicates the control information. A brief annotation describes the type of information passed.

Structure chart is a tool to assist the analyst in developing software that meets the objectives of good software design.

6.2. Structure of Modules

We have discussed in the previous chapter that a system may be seen as a combination of everal small independent units. So, while designing software also, it is designed as a collection of separately named and addressable components called modules.

This property of software is termed as modularity. Modularity is a very important feature of any software and allows a program to be intellectually manageable. For instance, while coding small programs in 'C' also, we make a program as a collection of small functions.

A program for finding average of three numbers may make use of a function for calculating the sum of the numbers. Each of these can be called as a separate module and may be written by a different programmer. But once such modules are created, different programs may use them. Thus modularity in software provides several advantages apart from making the program more manageable.

While designing the modular structure of the program, several issues are to be paid ention. The modular structure should reflect the structure of the problem. It should have the following properties.

1. Intra-module property: Cohesion
   Modules should be cohesive.

2. Inter module property: Coupling
   Modules should be as loosely interconnected as possible.

   - Highly coupled modules are strongly interconnected.
   - Loosely coupled modules are weakly connected.
   - De-coupled modules exhibit no interconnection.

3. A module should capture in it strongly related elements of the problem.

6.2.1. Cohesion

Cohesion, as the name suggests, is the adherence of the code statements within a module. It is a measure of how tightly the statements are related to each other in a module.Structures that tend to group highly related elements from the point of view of the problem tend to be more modular. Cohesion is a measure of the amount of such grouping.

Cohesion is the degree to which module serves a single purpose. Cohesion is a natural extension of the information-hiding concept. A cohesive module performs a single task within a software procedure, requiring little interaction with procedures being performed in other parts of a program. There should be always high cohesion.

Cohesion: modules should interact with and manage the functions of a limited number of lower-level modules.

There are various types of cohesion.

Functional Cohesion

A module performs just one function.

Examples:
1. READ-RECORD.
2. EDIT-TRANSACTION 13.

This is acceptable. But it breaks modules down into very small parts.

Sequential Cohesion

Module consisting of those processing elements, which has the output of one as the input of the next, is known to be sequentially cohesive.

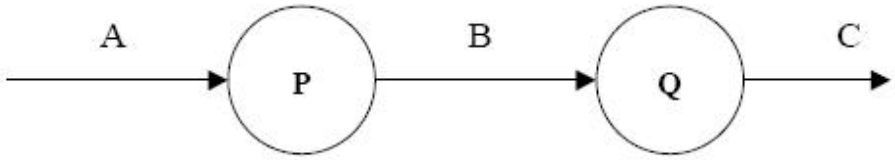


Fig. 6.4
P and Q modules show sequential cohesion

Module consisting of P and Q.

In terms of DFD, this combines a linear chain of successive transformations. This is acceptable.

Example:

1. READ-PROCESS; WRITE RECORD
2. Update the current inventory record and write it to disk.

Communicational Cohesion

Module consists of all processing elements, which act upon the same input data set and/or produce the same output data set.

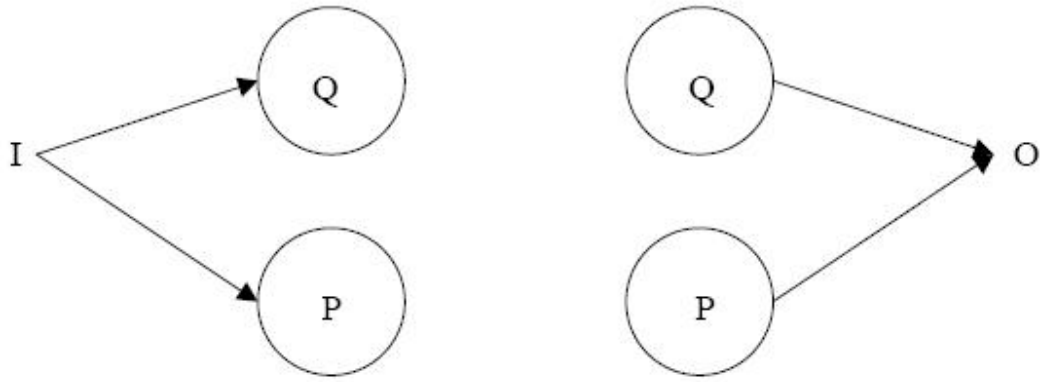


Fig. 6.5
Communicational cohesion

P and Q form a single module.

Module is defined in terms of the problem structure as captured in the DFD. It is commonly found in business or commercial applications where one asks what are the things that can be done with this data set.

This is acceptable.

Example:

1. Update master time clock record, the employee time and the current pay entry- all from same record.

Procedural Cohesion

Module formation associates processing elements together since these are found in the same procedural unit.

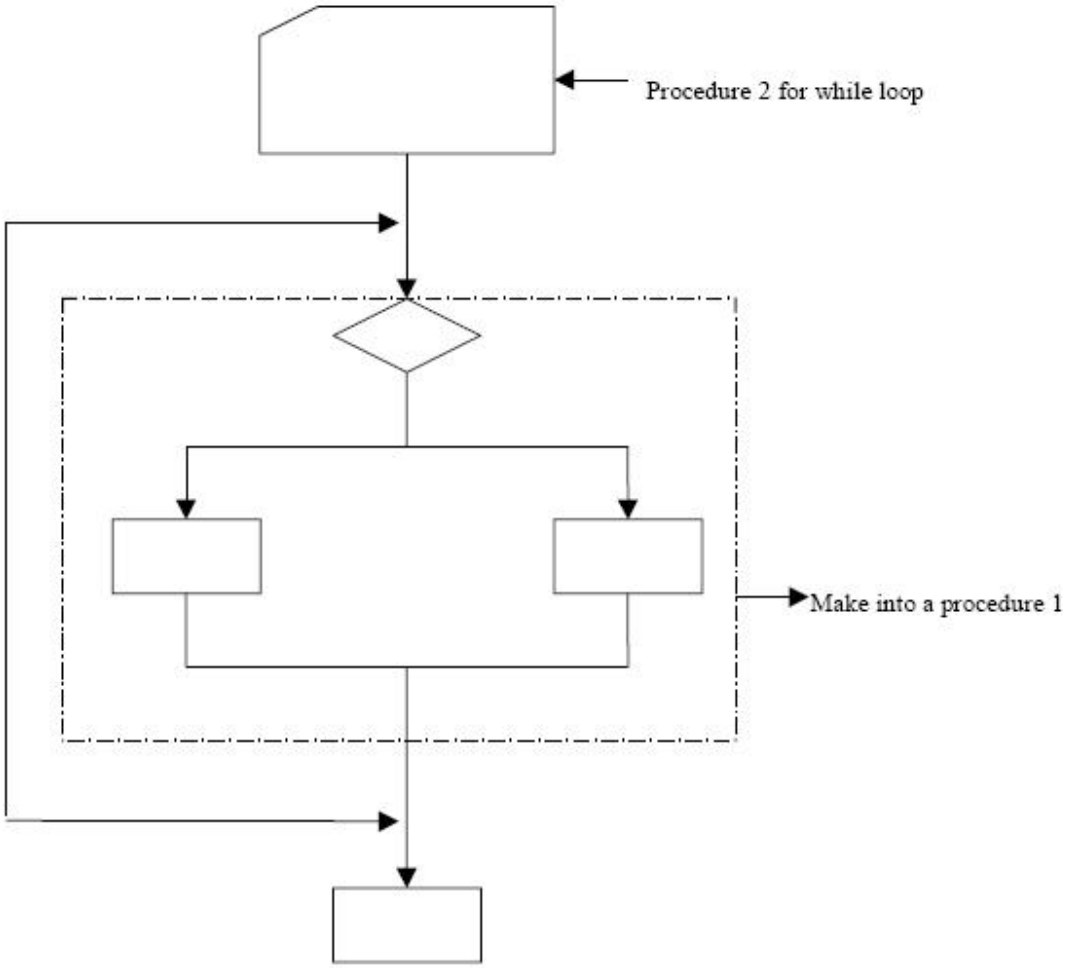


Fig. 6.6
Procedural cohesion

Often found when modules are defined by cutting up flowcharts or other procedural artifacts. There is no logical reasoning behind this. Fig 6.6 illustrates this. In this all the elements that are being used in the procedure 2 are put in the same module. It is not acceptable. Since elements of processing shall be found in various modules in a poorly structured way.

Temporal Cohesion

Temporal Cohesion is module formation by putting together all those functions, which happen at the same time.

Example:

Before sorting, write a proof tape and check totals. So put functions for writing proof tape and checking totals in the same module.

Logical Cohesion

Logical Cohesion is module formation by putting together a class of functions to be performed. It should be avoided.
For example, Display_error on file, terminal, printer, etc.

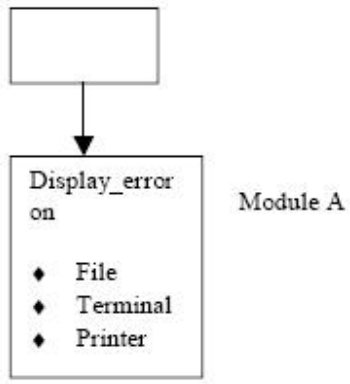


Fig. 6.7
Logical Cohesion

Fig 6.7 shows logical cohesion. Here function Display_error is for files, terminals, and printers. Since the function is to display error, all three functions are put into same modules.

Another example, produce job control reports, library file listings, and customer run support.

Coincidental Cohesion

Coincidental Cohesion is module formation by coincidence. Same code is recognized as occurring in some other module. Pull that code into a separate module. This type of cohesion should be avoided since it does not reflect problem structure

6.2.2. Coupling

Coupling is a measure of interconnection among modules in a program structure. Coupling depends on the interface complexity between modules, the point at which entry or reference is made to a module, and what data pass across the interface. Or, simply it is Strength of the connections between modules Coupling can be represented, as a spectrum as shown is figure below. As the number of different items being passed (not amount of data) rises the module interface complexity rises. It is number of different items that increase the complexity not the amount of data.

There are four types of coupling.

Data coupling

In this the argument list data that is passed to the module. In this type of coupling only data flows across modules. Data coupling is minimal.

Stamp coupling

In this type of coupling, a portion of the structure is argument.

Control coupling

If there is control flag, that is, control decisions in subordinate module then it is control coupling..

Common coupling

Common coupling occurs when there are common data areas. That is there are modules using data that are global. It should be avoided.

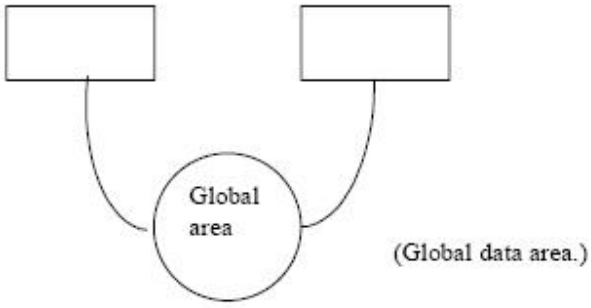


Fig. 6.8
Common coupling

Content coupling

if there is data access within the boundary of another. For example, passing pointer can be considered as content coupling or branch into the middle of a module.

6.3. Coding

Structure charts provide the framework for programmers to code the various modules of the system by providing all the necessary information about each module of the system. From here the system analyst takes a backseat and programmer comes into action. Programmer codes each and every module of the system, which gives a physical shape to the system.

Coding phase is aimed at converting the design of the system prepared during the design phase onto the code in a programming language, which performs the tasks as per the design requirements and is

executable by a computer. The programs should be written such that they are simple, easy to read, understand and modify.

Program should also include comment lines to increase the readability and modifiability of the program. Tools like flow charts and algorithms are used for coding the programs. In some cases, this phase may involve installation and modification of the purchased software packages according to the system requirements.

From here, the system goes for testing.

6.4. Data Dictionary

The analysis model encompasses representations of data objects, functions, and control. In each representation data objects and/or control items play a role. Therefore, it is necessary to provide an organized approach for representing the characteristics of each data objects and control items. This is accomplished with the data dictionary.

Formally, data dictionary can be defined as:

The data dictionary is an organized listing of all data elements that are pertinent to the system, with precise, rigorous definitions so that both user and system analyst will have a common understanding of inputs, outputs, components of stores and even intermediate calculations.

Data dictionary contains the following information:

Name-the primary name of data or control item, the data store, or an external entity.

Alias -other names used for first entry.

Where-used/how-used-a listing of the processes that use the data or control item and how it is used. For example, input to a process, output from the process, as a store, as an external entity.

Central description- a notation for representing content.

Supplementary information - other information about data types, preset values, restrictions or limitations, etc.

The logical characteristics of current data stores, including name, description, aliases, contents, and organization. Identifies processes where the data are used and where immediate access to information is needed. Serves as the basis for identifying database requirements during system design.

6.5. Functional Modeling II- Self Assessment

1. What is a module?

2. What are rules for writing a module?

3. What are module control structures?

4. What are the criteria for good module design.

5. What is the advantage of showing data flow graphically, rather than using narrative description to study or explain a system?

6.6. Functional Modeling II- Summary

Data flow analysis concentrate on the following activities.

- *What processes make up the system?*
- *What data are used in each process?*
- *What data are stored?*
- *What data enter and leave the system*

Tools for analysis and design

- *Data flow diagram: to show the flow of data through the system.*
- *Data dictionary: to keep logical characteristics of the data of the system.*
  - Data stores
  - Name
  - Description
  - Aliases
  - Contents
  - Types
- *Structure charts: tools for designing.*
  - To show the relationship between the modules of the system.
  - The hierarchy of modules.
  - Data transmitted between these modules.

Chapter 7: Data Modeling Techniques

At the end of this chapter you will be able to understand the concepts involve in the data modeling of a system. You will also be able to understand the Entity Relationship Model used for data modeling.

7.1. Data Requirements and Data modelings

Last chapter discusses about one part of the conceptual design process, the functional model. The other is the data model, which discusses the data related design issues of the system. See fig 7.1. The data model focuses on what data should be stored in the database while the function model deals with how the data is processed. In this chapter, we'll look into details of data modeling.

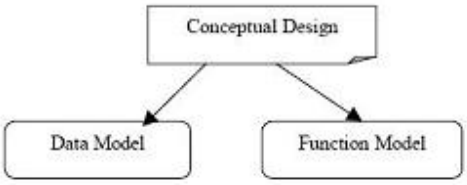


Fig 7.1 Elements of Conceptual Design

We have already discussed the Data Flow Diagrams, which make the foundation of the system under development. While the system is being studied, the physical DFDs are prepared whereas at the design phase, the basic layout of the proposed system is depicted in the form of a logical DFD. Taking this DFD as the basis the system is further developed. Even at the Data Modeling phase, the DFD can provide the basis in the form of the data flows and the Data Stores depicted in the DFD of the proposed system. The Data Stores from the DFD are picked up and based on the data being stored by them the Data Model of the system is prepared.

Prior to data modeling, we'll talk of basics of database design process. The database design process can be described as a set of following steps. (Also see figure 7.2)

- Requirement collection: Here the database designer interviews database users. By this process they are able to understand their data requirements. Results of this process are clearly documented. In addition to this, functional requirements are also specified. Functional requirements are user defined operations or transaction like retrievals, updates, etc, that are applied on the database.
- Conceptual schema: Conceptual schema is created. It is the description of data requirements of the users. It includes description of data types, relationships and constraints.
- Basic data model operations are used to specify user functional requirements.
- Actual implementation of database.
- Physical database design. It includes design of internal storage structures and files.

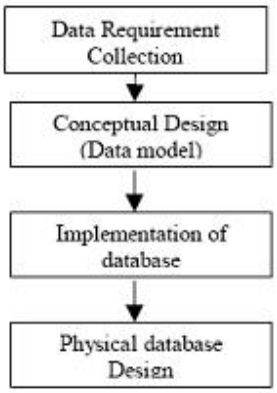


Fig 7.2 Overall database design process

In this chapter, our main concern is data model. There are various data models available. They fall in three different groups.

· Object-based logical models
· Records-based logical models
· Physical-models

7.1.1. Object-Based Logical Models

Object-based logical models are used in describing data at the logical and view levels. The main characteristic of these models is that they provide flexible structuring capabilities and allows data constraints to be specified explicitly. Many different models fall into this group. They are following.
· Entity-relationship model
· Object-oriented model

In this chapter, we'll discuss Entity-Relationship model in detail. The object-oriented model is covered in the next chapter.

7.1.2. Record-Based Logical Models

Records-based logical models are used in describing data at the logical and view levels. They are used to specify the overall logical structure of the database and to provide a higher-level description of the implementation.

In record-based models, the database is structured in fixed-format records of several types. Each record type defines a fixed number of fields, or attributes, and each field is usually of a fixed length. The use of fixed-length records simplifies the physical-level implementation of the database.

The following models fall in this group.
· Relational model
· Network model
· Hierarchical model

7.1.3. Relational Model

This model uses a collection of tables to represent both data and relationship among those data. Each table has multiple columns, and each column has a unique name. Figure shows a simple relational database.

EmpNo	EmpName	Age
Jyoti	1000	23
Saurabh	2000	21
Rajeev	3500	45
Abhay	4000	25

EmpNo	DOJ
1000	15-Jul-98
2000	1-May-97
3500	1-Jun-95
4000	15-Jun-94

Fig. 7.3 A sample relational model

7.1.4. Network Model

In network database, data is represented by collection of records, and relationships among data are represented by links. The records are organized as a collection of arbitrary graphs. Figure 7.4 represent a simple network database.

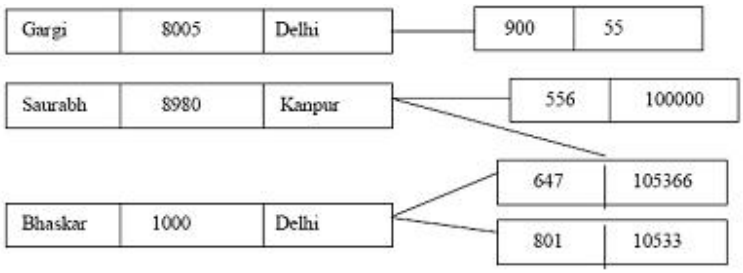


Fig. 7.4 sample network model

7.1.5. Hierarchical Model

The hierarchical model is similar to the network model. Like network model, records and links represent data and relationships among data respectively. It differs from the network model in that the records are organized as collections of trees rather than arbitrary graphs. Fig 7.5 represents a simple database.

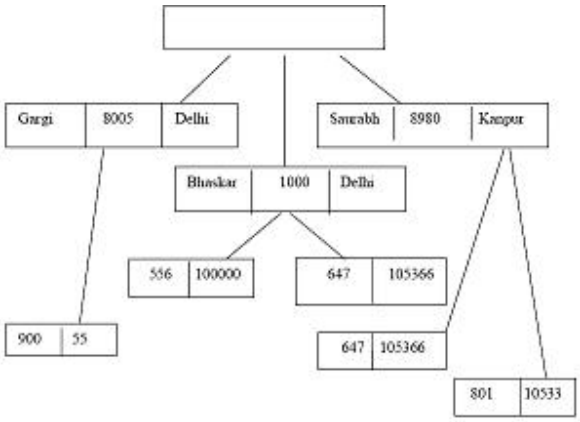


Fig. 7.5 A sample hierarchical database

7.1.6. Physical Data Models

Physical data models are used to describe data at the lowest level. A few physical data models are in use. Two such models are:

· Unifying model
· Frame-memory model

Physical data models capture aspects of database-system implementation.

7.2. E-R Data Modeling Technique

Now we know various data models available. To understand the process of data modeling we'll study Entity Relationship model. Peter P. Chen originally proposed the Entity- Relationship (ER) model in 1976. The ER model is a conceptual data model that views the real world as a construct of entities and associations or relationships between entities.

A basic component of the model is the Entity-Relationship diagram, which is used to visually represent data objects. The ER modeling technique is frequently used for the conceptual design of database applications and many database design tools employ its concepts.

ER model is easy to understand. Moreover it maps easily to relational model. The constructs used in ER model can easily be transformed into relational tables. We will look into relational model in the

next chapter, where other data models are discussed. In the following section, we'll look at E-R model concepts.

We can compare ER diagram with a flowchart for programs. Flow chart is a tool for designing a program; similarly ERD is a tool for designing databases. Also an ER diagram shows the kind and organization of the data that will be stored in the database in the same way a flowchart chose the way a program will run.

7.2.1. E-R Model concept

The ER data modeling techniques is based on the perception of a real world that consists of a set of basic objects called entities, and of relationships among these objects. In ER modeling, data is described as entities, relationships, and attributes. In the following section, entities and attributes are discussed. Later, entity types, their key attributes, relationship types, their structural constraints, and weak entity types are discussed. In the last, we will apply ER modeling to our case study problem "Library management system".

7.2.2. Entities and Attributes

One of the basic components of ER model is entity. An entity is any distinguishable object about which information is stored. These objects can be person, place, thing, event or a concept. Entities contain descriptive information. Each entity is distinct.

An entity may be physical or abstract. A person, a book, car, house, employee etc. are all physical entities whereas a company, job, or a university course, are abstract entities.

Fig. 7.6 Physical and Abstract Entity

Another classification of entities can be independent or dependent (strong or weak) entity.

ntities are classified as independent or dependent (in some methodologies, the terms used are strong and weak, respectively). An independent entity is one, which does not rely on another entity for identification. A dependent entity is one that relies on another entity for identification. An independent entity exists on its own whereas dependent entity exists on the existence of some other entity. For example take an organization scenario. Here department is independent entity. Department manager is a dependent entity. It exists for existing depts. There won't be any department manager for which there is no dept.

Some entity types may not have any key attributes of their own. These are called weak entity types. Entities belonging to a weak entity type are identified by being related to specific entities from another entity type in combination with some of their attribute values. For example, take the license entity. It can't exist unless it is related to a person entity.

7.2.3. Attributes

After you identify an entity, then you describe it in real terms, or through its attributes. Attributes are basically properties of entity. We can use attributes for identifying and expressing entities. For example, Dept entity can have DeptName, DeptId, and DeptManager as its attributes. A car entity can have modelno, brandname, and color as its attributes.

A particular instance of an attribute is a value. For example, "Bhaskar" is one value of the attribute Name. Employee number 8005 uniquely identifies an employee in a company.

The value of one or more attributes can uniquely identify an entity.

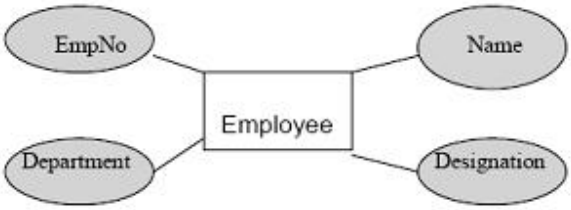


Fig 7.7 Entity and its attributes

In the above figure, employee is the entity. EmpNo, Name, Designation and Department are its attributes.

An entity set may have several attributes. Formally each entity can be described by set of <attribute, data value> pairs.

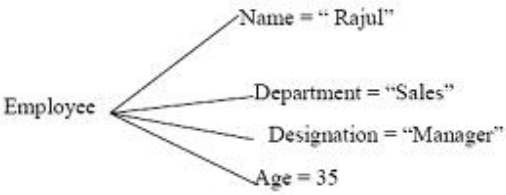


Fig. 7.8 Employee entity and its attribute values

7.3. Types of Attributes

Attributes can be of various types. In this section, we'll look at different types of attributes. Attributes can be categorized as:

7.3.1. Key or non-key attributes

Attributes can be classified as identifiers or descriptors. Identifiers, more commonly called keys or key attributes uniquely identify an instance of an entity. If such an attribute doesn't exist naturally, a new attribute is defined for that purpose, for example an ID number or code. A descriptor describes a non-unique characteristic of an entity instance.

An entity usually has an attribute whose values are distinct for each individual entity. This attribute uniquely identifies the individual entity. Such an attribute is called a key attribute. For example, in the Employee entity type, EmpNo is the key attribute since no two employees can have same employee number. Similarly, for Product entity type, ProdId is the key attribute.

There may be a case when one single attribute is not sufficient to identify entities. Then a combination of attributes can solve this purpose. We can form a group of more than one attribute and use this combination as a key attribute. That is known as a composite key attribute. When identifying attributes of entities, identifying key attribute is very important.

7.3.2. Required or optional Attributes

An attribute can be required or optional. When it's required, we must have a value for it, a value must be known for each entity occurrence. When it's optional, we could have a value for it, a value may be known for each entity occurrence. For example, there is an attribute EmpNo (for employee no.) of entity employee. This is required attribute since here would be no employee having no employee no. Employee's spouse is optional attribute because an employee may or may not have a spouse.

7.3.3. Simple and composite Attributes

Composite attributes can be divided into smaller subparts. These subparts represent basic attributes with independent meanings of their own. For example, take Name attributes. We can divide it into sub-parts like First_name, Middle_name, and Last_name.

Attributes that can't be divided into subparts are called Simple or Atomic attributes. For example, EmployeeNumber is a simple attribute. Age of a person is a simple attribute.

7.3.4. Single-valued and multi-valued Attributes

Attributes that can have single value at a particular instance of time are called singlevalued. A person can't have more than one age value. Therefore, age of a person is a single-values attribute. A multi-valued attribute can have more than one value at one time. For example, degree of a person is a multi-valued attribute since a person can have more than one degree. Where appropriate, upper and lower bounds may be placed on the number of values in a multi-valued attribute. For example, a bank may limit the number of addresses recorded for a single customer to two.

7.3.5. Stored, coded, or derived Attributes

There may be a case when two or more attributes values are related. Take the example of age. Age of a person can be can be calculated from person's date of birth and present date. Difference between the two gives the value of age. In this case, age is the derived attribute.

The attribute from which another attribute value is derived is called stored attribute. In the above example, date of birth is the stored attribute. Take another example, if we have to calculate the interest on some principal amount for a given time, and for a particular rate of interest, we can simply use the interest formula

Interest=NPR/100;

In this case, interest is the derived attribute whereas principal amount(P), time(N) and rate of interest(R) are all stored attributes.

Derived attributes are usually created by a formula or by a summary operation on other attributes.

A coded value uses one or more letters or numbers to represent a fact. For example, the value Gender might use the letters "M" and "F" as values rather than "Male" and "Female".

The attributes reflect the need for the information they provide. In the analysis meeting, the participants should list as many attributes as possible. Later they can weed out those that are not applicable to the application, or those clients are not prepared to spend the resources on to collect and maintain. The participants come to an agreement, on which attributes belong with an entity, as well as which attributes are required or optional.

7.3.6. Entity Types

An entity set is a set of entities of the same type that share the same properties, or attributes. For example, all software engineers working in the department involved in the Internet projects can be defined as the entity set InternetGroup. The individual entities that constitute a set are called extension of the entity set. Thus, all individual software engineers of in the Internet projects are the extensions of the entity set InternetGroup.

Entity sets don't need to be disjointed. For example, we can define an entity set Employee. An employee may or may not be working on some Internet projects. In InternetGroup we will have some entries that are there in Employee entity set. Therefore, entity sets Employee and InternetGroup are not disjoint.

A database usually contains groups of entities that are similar. For example, employees of a company share the same attributes. However, every employee entity has its own values for each attribute. An entity type defines a set of entities that have same attributes. A name and a list of attributes describe each entity type.

Fig. 7.10 shows two entity types Employee and Product. Their attribute list is also shown. A few members of each entity type are shown.

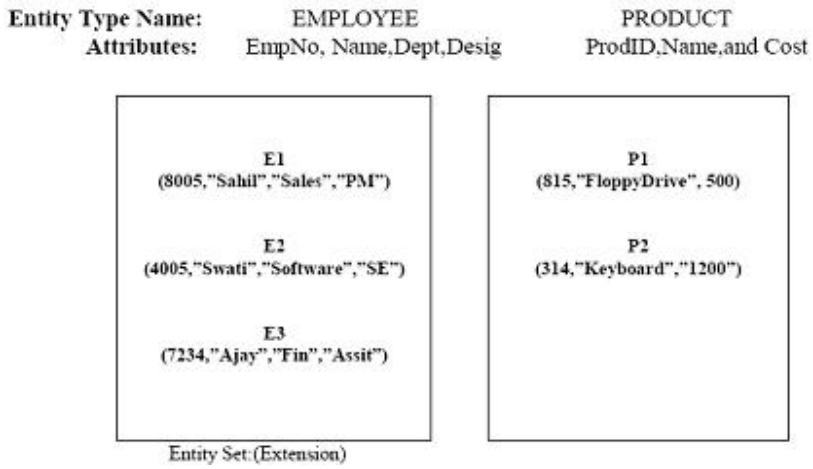


Fig. 7.10 Two entity types and some of the member entities of each

An entity type is represented in ER diagrams as rectangular box and the corresponding attributes are shown in ovals attached to the entity type by straight lines. See fig 7.7.

An entity type is basically the schema or intension or structure for the set of entities that share the same structure whereas the individual entities of a particular entity type are collectively called entity set. The entity set is also called the extension of the entity type.

7.3.7. Value Sets (domain) of Attributes

Each attribute of an entity type is associated with a value set. This value set is also called domain. The domain of an attribute is the collection of all possible values an attribute can have.

The value set specifies the set of values that may be assigned for each individual entity. For example, we can specify the value set for designation attribute as <“PM”, “Assit”, “DM”, “SE”>. We can specify “Name” attribute value set as <strings of alphabetic characters separated by blank characters>. The domain of Name is a character string.

7.4. Entity Relationships

After identification of entities and their attributes, the next stage in ER data modeling is to identify the relationships between these entities.

We can say a relationship is any association, linkage, or connection between the entities of interest to the business. Typically, a relationship is indicated by a verb connecting two or more entities. Suppose there are two entities of our library system, member and book, then the relationship between them can be “borrows”.

Member borrows book

Each relationship has a name, degree and cardinality. These concepts will be discussed next.

7.4.1. Degree of an Entity Relationship Type

Relationships exhibit certain characteristics like degree, connectivity, and cardinality. Once the relationships are identified their degree and cardinality are also specified.

Degree: The degree of a relationship is the number of entities associated with the relationship. The n-ary relationship is the general form for degree n. Special cases are the binary, and ternary, where the degree is 2, and 3, respectively.

Binary relationships, the association between two entities are the most common type in the real world.

Fig 7.11 shows a binary relationship between member and book entities of library system

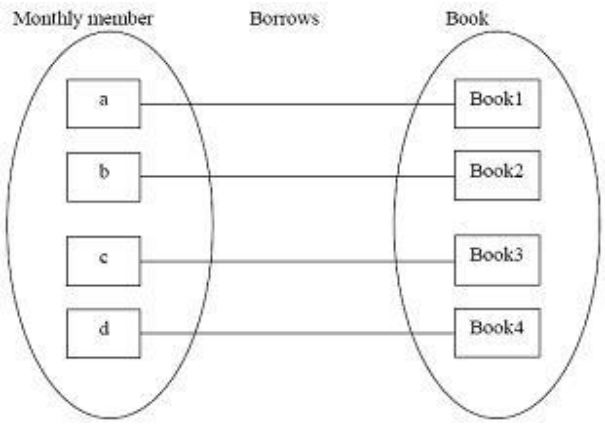


Fig. 7.11 Binary Relationship

A ternary relationship involves three entities and is used when a binary relationship is inadequate. Many modeling approaches recognize only binary relationships. Ternary or n-ary relationships are decomposed into two or more binary relationships.

7.4.2. Connectivity and Cardinality

By connectivity we mean how many instances of one entity are associated with how many instances of other entity in a relationship. Cardinality is used to specify such connectivity. The connectivity of a relationship describes the mapping of associated entity instances in the relationship. The values of connectivity are "one" or "many". The cardinality of a relationship is the actual number of related occurrences for each of the two entities. The basic types of connectivity for relations are: one-to-one, one-to-many, and many-tomany.

A **one-to-one (1:1)** relationship is when at most one instance of an entity A is associated with one instance of entity B. For example, each book in a library is issued to only one member at a particular time.

A **one-to-many (1:N)** relationship is when for one instance of entity A, there are zero, one, or many instances of entity B but for one instance of entity B, there is only one instance of entity A. An example of a 1:N relationships is

a department has many employees;
each employee is assigned to one department.

A **many-to-many (M:N)** relationship, sometimes called non-specific, is when for one instance of entity A, there are zero, one, or many instances of entity B and for one instance of entity B there are zero, one, or many instances of entity A. An example is employees may be assigned to no more than three projects at a time; every project has at least two employees assigned to it.

Here the cardinality of the relationship from employees to projects is three; from projects to employees, the cardinality is two. Therefore, this relationship can be classified as a many-to-many relationship.

If a relationship can have a cardinality of zero, it is an optional relationship. If it must have a cardinality of at least one, the relationship is mandatory. Optional relationships are typically indicated by the conditional tense. For example,

An employee may be assigned to a project.

Mandatory relationships, on the other hand, are indicated by words such as must have. For example, a student must register for at least three courses in each semester.

7.4.3. Designing basic model and E-R Diagrams

E-R diagrams represent the schemas or the overall organization of the system. In this section, we'll apply the concepts of E-R modeling to our "Library Management System" and draw its E-R diagram.

In order to begin constructing the basic model, the modeler must analyze the information gathered during the requirement analysis for the purpose of: and

- classifying data objects as either entities or attributes,
- identifying and defining relationships between entities,
- naming and defining identified entities, attributes, and relationships,
- documenting this information in the data document.
- Finally draw its ER diagram.

To accomplish these goals the modeler must analyze narratives from users, notes from meeting, policy and procedure documents, and, if lucky, design documents from the current information system.

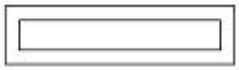
7.4.4. E-R diagrams constructs

In E-R diagrams, entity types are represented by squares. See the table below. Relationship types are shown in diamond shaped boxes attached to the participating entity types with straight lines. Attributes are shown in ovals, and each attribute is attached to its entity type or relationship type by a straight line. Multivalued attributes are shown in double ovals. Key attributes have their names underlined. Derived attributes are shown in dotted ovals.

Weak entity types are distinguished by being placed in double rectangles and by having their identifying relationship placed in double diamonds.

Attaching a 1, M, or N on each participating edge specifies cardinality ratio of each binary relationship type. The participation constraint is specified by a single line for partial participation and by double lines for total participation. The participation constraints specify whether the existence of an entity depends on its being related to another entity via the relationship type. If every entity of an entity set is related to some other entity set via a relationship type, then the participation of the first entity type is total. If only few member of an entity type is related to some entity type via a relationship type, the participation is partial.

ENTITY TYPE



WEAK ENTITY TYPE

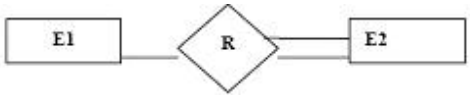
RELATIONSHIP TYPE

ATTRIBUTE

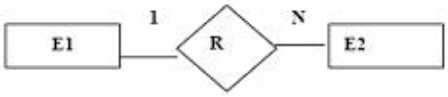
KEY ATTRIBUTE

MULTIVALUED ATTRIBUTE

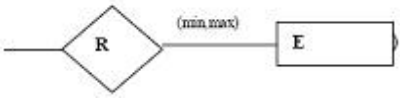
DERIVED ATTRIBUTE



TOTAL PARTICIPATION OF E2 IN R



Cardinality Ratio 1:N FOR E1:E2 IN R



Structural Constraint(Min,Max) On Participation Of E In R

7.4.5. Naming Data Objects

The names should have the following properties:

- unique,
- have meaning to the end-user.
- contain the minimum number of words needed to uniquely and accurately describe the object.

For entities and attributes, names are singular nouns while relationship names are typically verbs.

7.4.6. E-R Diagram for library management system

In the library Management system, the following entities and attributes can be identified.

- **Book** -the set all the books in the library. Each book has a Book-id, Title, Author, Price, and Available (y or n) as its attributes.
- **Member**-the set all the library members. The member is described by the attributes Member_id, Name, Street, City, Zip_code, Mem_type, Mem_date (date of membership), Expiry_date.
- **Publisher**-the set of all the publishers of the books. Attributes of this entity are Pub_id, Name, Street, City, and Zip_code.
- **Supplier**-the set of all the Suppliers of the books. Attributes of this entity are Sup_id, Name, Street, City, and Zip_code.

Assumptions: a publisher publishes a book. Supplier supplies book to library. Members borrow the book (only issue).

Return of book is not taken into account

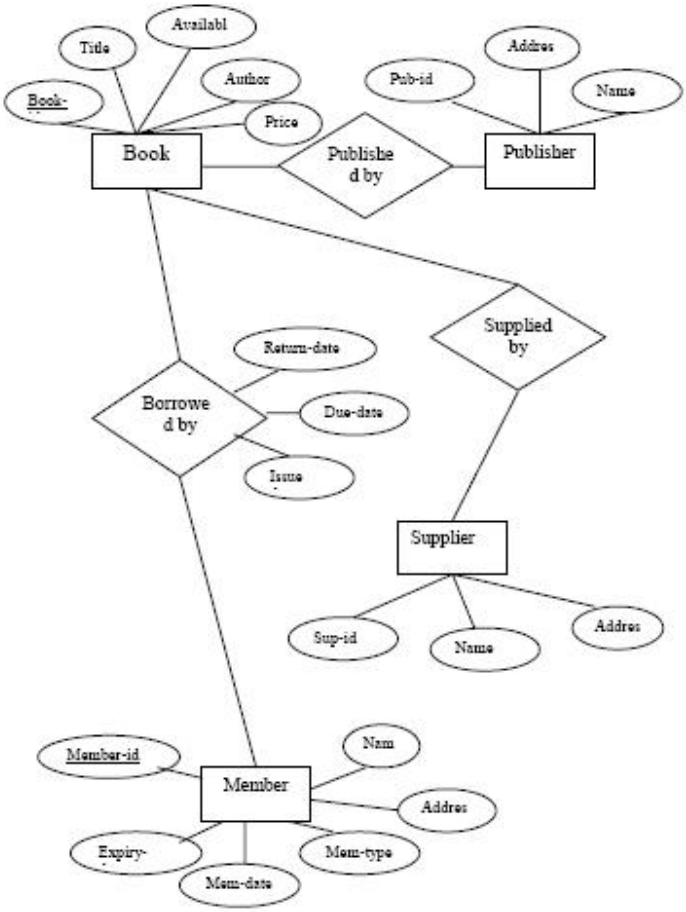


Fig. 7.13 E-R Diagram of Library Management System.

Chapter 8: Relational Data Modeling and Object Oriented Data Modeling Techniques

At the end of this chapter, you will be able to know about relational data modeling and Object Oriented data modeling techniques.In the last chapter we saw one data modeling technique, Entity Relationship data model. Having the basic knowledge of data modeling we can now explore other methods for data modeling. There are many data models available. Network, hierarchical, relational and object oriented databases. In this section, we'll take relational model and object oriented model.

8.1. Relational Database Model

E.F.Codd proposed this model in the year 1970. The relational database model is the most popular data model. It is very simple and easily understandable by information systems professionals and end users.

Understanding a relational model is very simple since it is very similar to Entity Relationship Model. In ER model data is represented as entities similarly here data in represented in the form of relations that are depicted by use of two-dimensional tables.

Also attributes are represented as columns of the table. These things are discussed in detail in the following section.

The basic concept in the relational model is that of a relation. In simple language, a relation is a two-dimensional table. Table can be used to represent some entity information or some relationship between them. Even the table for an entity information and table for relationship information are similar in form. Only from the type of information given in the table can tell if the table is for entity or relationship. The entities and relationships, which we studied in the ER model, are similar to relations in this model. In relational model, tables represent all the entities and relationships identified in ER model.

Rows in the table represent records; and columns show the attributes of the entity.

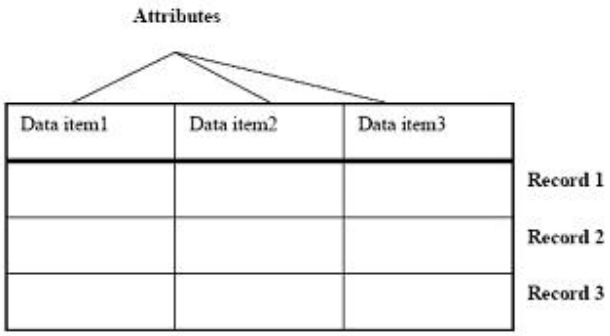


Fig. 8.1 Structure of a relation in a relational model.

Fig 8.1 shows structure of a relation in a relational model.

A table exhibits certain properties. It is a column homogeneous. That is, each item in a particular column is of same type. See fig 8.2. It shows two columns for EmpNo and Name. In the EmpNo column it has only employee numbers that is a numeric quantity. Similarly in Name column it has alphabetic entries. It is not possible for EmpNo to have some non-numeric value (like alphabetic value). Similarly for Name column only alphabetic values are allowed.

EmpNo	***Name***	***.....***	***.....***

1001	*Jason*		
1002	*William*		
1003	*Mary*		
1004	*Sarah*		

Fig. 8.2 Columns are homogeneous

Another important property of table is each item value is atomic. That is, item can't be further divided. For example, take a name item. It can have first name, middle name, or last name. Since these would be three different strings so they can't be placed under one column, say Name. All the three parts are placed in three different columns. In this we can place them under, FirstName, MiddleName, and LastName. See fig 8.3.

FirstName	***MiddleName***	***LastName***	***.....***
Jason	*Tony*	*White*	*.....*
William	*Bruce*	*Turner*	*.....*
Jack	*Pirate*	*Sparrow*	*.....*

Fig. 8.3 Table columns can have atomic values

Every table must have a primary key. Primary key is some column of the table whose values are used to distinguish the different records of the table. We'll take up primary key topic later in the session. There must be some column having distinct value in all rows by which one can identify all rows. That is, all rows should be unique. See fig. 8.4.

EmpNo	*EName*	*DOJ*
1001	*Jason*	*20-Jun-2007*
1002	*William*	*12-Jul-2007*
1002	*William*	*20-Jul-2007*
1010	*Smith*	*20-Jul-2007*

Fig. 8.4 Table with primary key "EmpNo" and degree "3"

In this table, EmpNo can be used as a primary key. Since it is the only column where the values are all distinct. Whereas in Ename there are two William and in DOJ column, 15-Jul- 1998 is same for three row no 1,3, and 4. If we use DOJ as primary key then there would be three records that have same DOJ so there won't be any way to distinguish these three records. For this DOJ can't be a primary key for this table. For similar reasons, Ename cannot be used as primary key.

Next property we are going to discuss is for ordering of rows and columns within a table. Ordering is immaterial for both rows and columns. See fig. 8.5. Table (a) and (b) represent the same table.

DName	***DeptID***	***Manager***
SD	*1*	*Smith*
HR	*2*	*William*
FIN	*3*	*Jonathan*
EDU	*4*	*Jason*

DName	***DeptID***	***Manager***
William	*HR*	*2*
Mary	*EDU*	*7*

Jason	*FIN*	*4*
Smith	*SD*	*1*

Fig. 8.5 Ordering of rows and columns in a table is immaterial

Names of columns are distinct. It is not possible to have two columns having same name in a table. Since a column specifies a attribute, having two columns with same name mean that we are specifying the same property in two columns, which is not acceptable. Total number of columns in a table specifies its degree. A table with n columns is said to have degree n. See fig. 8.1. Table represented there is of degree 3.

Domain in Relational Model

Domain is set of all possible values for an attribute. For example there is an Employee table in which there is a Designation attribute. Suppose, Designation attribute can take “PM”, “Trainee”, “AGM”, or “Developer”. Then we can say all these values make the domain for the attribute Designation. An attribute represents the use of a domain within a relation. Similarly for name attribute can take alphabetic strings. So domain for name attribute will be set of all possible valid alphabetic strings.

8.2. Different Types of Keys in Relational Database Model

Now we’ll take up another feature of relational tables. That is different type of keys. There are different types of keys, namely Primary keys, alternate keys, etc. The different types of keys are described below.

8.2.1. Primary key

Within a given relation, there can be one attribute with values that are unique within the relation that can be used to identify the tuples of that relation. That attribute is said to be primary key for that relation.

8.2.2. Composite primary key

Not every relation will have single-attribute primary key. There can be a possibility that some combination of attribute when taken together have the unique identification property. These attributes as a group is called composite primary key. A combination consisting of a single attribute is a special case.

Existence of such a combination is guaranteed by the fact that a relation is a set. Since sets don’t contain duplicate elements, each tuple of a relation is unique with respect to that relation. Hence, at least the combination of all attributes has the unique identification property.

In practice it is not usually necessary to involve all the attributes-some lesser combination is normally sufficient. Thus, every relation does have a primary (possibly composite) key.

Tuples represent entities in the real world. Primary key serves as a unique identifier for those entities.

8.2.3. Candidate key

In a relation, there can be more than one attribute combination possessing the unique identification property. These combinations, which can act as primary key, are called candidate keys.

EmpNo	*SocSecurityNo*	*Name*	*Age*
1011	*2364236*	*Harry*	*21*
1012	*1002365*	*Sympson*	*19*

1013	*1056300*	*Larry*	*24*

Fig. 8.6 Table having “EmpNo” and “SocSecurityNo” as candidate keys

8.2.4. Alternate key

A candidate key that is not a primary key is called an alternate key. In fig. 8.6 if EmpNo is primary key then SocSecurityNo is the alternate key

8.3. Integrity Rules in Relational Database Model

8.3.1. Integrity rule 1: Entity integrity

It says that no component of a primary key may be null.

All entities must be distinguishable. That is, they must have a unique identification of some kind. Primary keys perform unique identification function in a relational database. An identifier that was wholly null would be a contradiction in terms. It would be like there was some entity that did not have any unique identification. That is, it was not distinguishable from other entities. If two entities are not distinguishable from each other, then by definition there are not two entities but only one.

8.3.2. Integrity rule 2: Referential integrity

The referential integrity constraint is specified between two relations and is used to maintain the consistency among tuples of the two relations.

Suppose we wish to ensure that value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another. This is referential integrity.

The referential integrity constraint states that, a tuple in one relation that refers to another relation must refer to the existing tuple in that relation. This means that the referential integrity is a constraint specified on more than one relation. This ensures that the consistency is maintained across the relations.

Table A

DeptID	***DeptName***	***DeptManager***
F-1001	*Financial*	*Nathan*
S-2012	*Software*	*Martin*
H-0001	*HR*	*Jason*

TableB

EmpNo	***DeptID***	***EmpName***
1001	*F-1001*	*Tommy*
1002	*S-2012*	*Will*
1003	*H-0001*	*Jonathan*

8.3.3. Extensions and intensions in Relational Database Model

A relational in a relational database has two components, an extension and an intension.

8.3.3.1. Extension

The extension of a given relation is the set of tuples appearing in that relation at any given instance. The extension thus varies with time. It changes as tuples are created, destroyed, and updated.

Relation: Employee at time= t1

EmpNo	EmpName	Age	Dept
1001	Jason	23	SD
1002	William	24	HR
1003	Jonathan	28	Fin
1004	Harry	20	Fin

Relation: Employee at time= t2 after adding more records

EmpNo	EmpName	Age	Dept
1001	Jason	23	SD
1002	William	24	HR
1003	Jonathan	28	Fin
1004	Harry	20	Fin
1005	Smith	22	HR
1006	Mary	19	HR
1007	Sarah	23	SD

Relation: Employee at time= t2 after adding more records

EmpNo	EmpName	Age	Dept
1001	Jason	23	SD
1002	William	24	HR

8.3.3.2. Intension

The intension of a given relation is independent of time. It is the permanent part of the relation. It corresponds to what is specified in the relational schema. The intension thus defines all permissible extensions. The intension is a combination of two things : a structure and a set of integrity constraints.

- The naming structure consists of the relation name plus the names of the attributes (each with its associated domain name).
- The integrity constraints can be subdivided into key constraints, referential constraints, and other constraints.

For example,

Employee(EmpNo Number(4) Not NULL, EName Char(20), Age Number(2), Dept Char(4))

This is the intension of Employee relation.

8.4. Key Constraints in Relational Database Model

Key constraint is implied by the existence of candidate keys. The intension includes a specification of the attribute(s) consisting the primary key and specification of the attribute(s) consisting alternate keys, if any. Each of these specifications implies a uniqueness constraint(by definition of candidate key);in addition primary key specification implies a no-nulls constraint(by integrity rule 1).

8.4.1. Referential constraints

Referential constraints are constraints implied by the existence of foreign keys. The intension includes a specification of all foreign keys in the relation. Each of these specifications implies a referential constraint (by integrity rule 2).

8.4.2. Other constraints

Many other constraints are possible in theory.
Examples
salary >= 10000.

8.5. Relational Algebra

Once the relationships are identified, then operations that are applied on the relations are also identified. In relational model, the operations are performed with the help of relational algebra. Relational algebra is a collection of operations on relations.

Each operation takes one or more relations as its operand(s) and produces another relation as its result. We can compare relational algebra with traditional arithmetic algebra.

In arithmetic algebra , there are operators that operate on operands(data values) and produce some result. Similarly in relational algebra there are relational operators that operate upon relations and produce relations as results.

Relational algebra can be divided into two groups:

1. Traditional set operators that include union, intersection, difference, and Cartesian product.

2. Special relational operators that include selection, projection, and division.

8.5.1. Traditional set operators in Relational Database Model

8.5.1.1. Union:

The union of two relations A and B is the set of all tuples belonging to either A or B (or both).

Example:

A = The set of employees whose department is S/W Development
B = The set of employee whose age is less than 30 years.
A UNION B = The set of employees whose are either in S/W development department or having age less than 30 years.

8.5.1.2. Intersection:

The intersection of two relations A and B is the set of all tuples t belonging to both A and B.

Example:

A = The set of employees whose department is S/W Development
B = The set of employee whose age is less than 30 years.
A INTERSECTION B = The set of employees whose are in S/W development department having age less than 30 years.

8.5.1.3. Difference:

The difference between two relations A and B(in that order) is the set of all tuples belonging to A and not to B.

Example:

A = The set of employees whose department is S/W Development
B = The set of employee whose age is less than 30 years.
A MINUS B = The set of employees whose department is S/W development and not having age less than 30 years.

8.5.1.4. Cartesian Product:

The Cartesian product of two relations A and B is the set of all tuples t such that t is the concatenation of a tuple a belonging to A and a tuple b belonging to B. The concatenation of a tuple a = (a1, ……., am) and tuple b=(bm+1 , ……., bm+n)- in that order- is the tuple t =(a1, ….., am, bm+1, ……..bm+n).

Example:

A = The set of employees whose department is S/W Development
B = The set of employee whose age is less than 30 years.
A TIMES B = is the set of all possible employee no/department ids pairs.

8.5.2. Special Relational Operators in Relational Database Model

8.5.2.1. Selection:

The selection operator yields a 'horizontal' subset of a given relation - that is, the subset of tuples within the given relation for which a specified predicate is satisfied.

The predicate is expressed as a Boolean combination of terms, each term being a simple comparison that can be established as true or false for a given tuple by inspecting that tuple in isolation.

Book WHERE Author = ‘Kruse’

BookID	***BookName***	***Author***
A-112	*Algorithms*	*Jack*
C-12	*Data Mining*	*Jack*
F-348	*Software Engineer*	*Jack*

Employee WHERE Desig=’Manager’ AND Dept =’SD’

EmpNo	***EmpName***	***Designation***	***Depat***
2001	*Morrison*	*Manager*	*SD*
2002	*Steve*	*Manager*	*SD*
2003	*Fleming*	*Manager*	*SD*

8.5.2.2. Projection:

The projection yields a 'vertical' subset of a given relation- that is, the subset obtained by selecting specified attributes, in a specified left-to-right order, and then eliminating duplicate tuples within the attributes selected.

Example: Issue[BookId,ReturnDate]

BookID	***ReturnDate***
q-110	*20-May-2008*
w-990	*21-Jun-2008*
f-100	*23-Jun-2008*
r-800	*27-Jun-2008*
q-501	*15-Jul-2008*

Book[BookName]

BookName
Software Concepts
Data Structures
Programming
AssemblyLanguage
SSAD
PC-Troubleshooting
Compiler Design

Now we know about the constructs of relational data model. We also know how to specify constraints and how to use relational algebra for illustrating various functions. We now take up another data model that is entirely different from relational model.

8.5.2.3. Division:

The division operator divides a dividend relation A of degree m+n by a divisor relation B of degree n, and produces a result relation of degree m.

Let A be set of pairs of values <x, y> and B a set of single values, <y>. Then the result of dividing A by B - that is A DIVIDEDBY B- is the set of values x such that the pair <x, y> appears in A for all values y appearing in B.

8.6. Object Oriented Model

Nowadays people are moving towards the object oriented approach in many fields. These fields include programming, software engineering, development technologies, implementation of databases, etc. Object oriented concepts have their roots in the object oriented programming languages. Since programming languages were the first to use these concepts in practical sense. When these concepts became widely popular and accepted, other areas started to implement these ideas in them. It became possible due to the fact the object-oriented concepts try to model things as they are. So today it is a common practice to use object-oriented concepts in data modeling. In this session we'll try to look at the process of applying object oriented concepts to data modeling.

8.6.1. Object Oriented data Modeling Concepts

As discussed earlier Object oriented model has adopted many features that were developed for object oriented programming languages. These include objects, inheritance, polymorphism, and encapsulation.

In object-oriented model main construct is an object. As in E-R model we have entities and in relational model, there are relations similarly we have objects in OO data modeling. So first thing that is done in OO model is to identify the objects for the systems. Examining the problem statement can do it. Other important task is to identify the various operations for these objects. It is easy to relate the objects to the real world. In the section that follows we will try to understand the basic concepts of OO data model.

8.6.2. Objects and object identity:

In this model, everything is modeled as objects. An object can be any physical or abstract thing. It can be a person, place, thing, or a concept. An object can be used to model the overall structure not just a part of it. Also the behavior of the thing that is being modeled is also specified in the object. This feature is called encapsulation. Encapsulation is discussed later in the chapter. Only thing we need to know at this stage is object can store information and behavior in the same entity i.e. an object. Suppose a car is being modeled using this method. Then we can have an object 'Car' that has the following information.

Car:
Color, Brand, ModelNo, Gears, EngineCylinders, Capacity, No of gates. All this information is sufficient to model any car.

All the objects should be unique. For this purpose, every object is given an identity. Identity is the property of an object, which distinguishes it from all other object. In OO databases, each object has a unique identity. This unique identity is implemented via a unique, system generated object identifier (OID). OID is used internally by the system to identify each object uniquely and to create and manage inter-object references. But its value is not visible to the user.

The main properties of OID :

1. It is immutable: The value of OID for a particular object should not change. This preserves the identity of the real world object being represented.
2. It is used only once: Even if an object is removed its OID is not assigned to other objects.

The value of OID doesn't depend upon any attributes of the object since the values of attributes may change. OID should not be based on the physical address of the object in memory since physical reorganization of the database could change the OID. There should be some mechanism for generating OIDs.

Another feature of OO databases is that objects may have a complex structure. It is due to contain all of the significant information that describes the object. In contrast, in traditional database systems, information about a complex object is often scattered over many relations or records. See fig. 8.12. It leads to loss of direct correspondence between a real-world object and its database representation.

The internal structure of an object includes the specification of instance variables, which hold the values, that defines the internal state of the object.

In OO databases, the values (or states) of complex objects may be constructed from other objects. These objects may be represented as a triple t(i, c, v), where i is a unique object identifier, c is a constructor (that is, an indication of how the object value is constructed), and v is the object value (or state).

There can be several constructors, depending upon or the OO system. The basic constructors are the atom, tuple, set, list, and array constructors. There is also a domain D that contains all basic atomic values that are directly available in the system. These include integers, real numbers, character strings, Boolean, dates, and any other data types that the system supports directly.

An object value v is interpreted on the basis of the value of the constructor c in the triple (i,c,v) that represents the object.

If c = atom, the value is an atomic value from domain D of basic values supported by the system.

If c = set, the value v is a set of objects identifiers{i1,i2,……..,in), which are the identifiers(OIDs) for a set of objects that are typically of the same type.

If c= tuple, the value v is a tuple of the form <a1:i1,……., an:in), where each aj is an attribute name(sometimes called an instance variable name in OO terminology) and each ij is an object identifier(OID).

If c = list, the value v is an ordered list of object identifiers[i1,i2,…….., in] of the same type. For c = array, the value v, is an array of object identifier.

Consider the following example:

O1 = (i1, atom, Rohit)
O2 = (i2, atom, Jai)
O3 = (i3, atom, Gargi)
O4 = (i4, set, {i1,i2,i3})
O5 = (i5, atom, SE)
O6 = (i6, atom, NEPZ)
O7 = (i7,tuple,<DNAME:i5,DNUMBER:i8,LOCATION:i6,ENAME:i3>)
O8 = (i8,atom,1)

Here value of object 4 is constructed from object values of objects O1, O2, and O3. Similarly value of object 7 is constructed from the value of O1, O3, O6, and O8. These constructors can be used to define the data structures for an OO database schema.

8.6.3. Encapsulation of Operations, Methods, and Persistence

Encapsulation is related to the concepts of abstract data types and information hiding in programming languages. Here the main idea is to define the behavior of a type of object based on the operations that can be externally applied to objects of that type. The internal structure of the object is hidden, and the object is only accessible through a number of predefined operations. Some operations may be used to create or destroy objects; other operations may update the object value and other may be used to retrieve parts of the object value or to apply some calculations to the object value.

The external users of the object are only made aware of the interface of the object, which defines the names and arguments of each operation. The implementation of the object is hidden from the external users; it includes the definition of the internal data structure of the object and the implementation of the operations that access these structures.

In OO terminology, the interface part of each operation is called the signature, and the operation implementation is called a method. A method is invoked by sending a message to the object to execute the corresponding method.

Not all objects are meant to be stored permanently in the database. Transient objects exist in the executing program and disappear once the program terminates.

Persistent objects are stored in the database and persist after program terminates. The typical mechanism for persistence involves giving an object a unique persistent name through which it can be retrieved.

8.6.4. Inheritance

Inheritance is deriving objects from existing objects. The derived objects inherit properties from their parent object. Parent objects are those objects from which other objects are derived. Inheritance is a way of reusing the existing code.

8.6.5. Polymorphism

Polymorphism concept allows the same operator name or symbol to be bound to two or more different implementation of the operator, depending on the type of objects to which the operator is applied.

8.6.6. Major features of Object Oriented databases

Object Oriented databases store persistent objects permanently on secondary storage, and allow the sharing of these objects among multiple programs and applications.

Object Oriented databases provide a unique system-generated object identifier for each object. Object Oriented databases maintain a direct correspondence between real-world and database objects so that objects don't lose their integrity and identify and can be easily be identified and operated upon.

In Object Oriented databases, objects can be very complex in order to contain all significant information that may be required to describe the object completely.

Object Oriented databases allow us to store both the class and state of an object between programs. They take the responsibility for maintaining the links between stored object behavior and state away from the programmer, and manage objects outside of programs with their public and private elements intact. They also simplify the whole process of rendering objects persistent by performing such tasks invisibly.

Persistence has to do with time i.e. a persistent object can exist beyond the program that created it. It also has to do with space (the location of the object may vary between processors, and even change its representation in the process).

8.7. Comparison Between Relational Database Model and Object Oriented Model

Now we know about both relational and object oriented approach, we can now compare these two models. In this session, we compare the relational model and object oriented model. We compare model representation capabilities, languages, system storage structures, and integrity constraints.

8.7.1. Data Model Representation

Different database models differ in their representation of relationships. In relational model, connections between two relations are represented by foreign key attribute in one relation that reference the primary key of another relation. Individual tuples having same values in foreign and primary key attribute are logically related. They are physically not connected. Relational model uses logical references.

In object oriented model, relationships are represented by references via the object identifier (OID). This is in a way similar to foreign keys but internal system identifiers are used rather than user-defined attributes. The Object Oriented model supports complex object structures by using tuple, set,

list, and other constructors. It supports the specification of methods and the inheritance mechanism that permits creation of new class definitions from existing ones.

8.7.2. Storage Structures

In relational model, each base relation is implemented as a separate file. If the does not specify any storage structure, most RDBMS will store the tuples as unordered records in the file. It allows the user to specify dynamically on each file a single primary or clustering index and any number of secondary indexes. It is the responsibility of user to chObject Orientedse the attributes on which the indexes are set up. Some RDBMSs give the user the option of mixing records from several base relations together. It is useful when related records from more than one relation are often accessed together. This clustering of records physically places a record from one relation followed by the related records from another relation. In this way the related records may be retrieved in most efficient way possible.

Object Oriented systems provide persistent storage for complex-structured objects. They employ indexing techniques to locate disk pages that store the object. The objects are often stored as byte strings, and the object structure is reconstructed after copying the disk pages that contain the object into system buffers.

8.7.3. Integrity Constraints

Relational model has keys, entity integrity, and referential integrity. The constraints supported by Object Oriented systems vary from system to system. The inverse relationship mechanism supported by some Object Oriented systems provides some declarative constraints.

8.7.4. Data manipulation Languages

There are languages such as SQL, QUEL, and QBE are available for relational systems. These are based on the relational calculus.

In Object Oriented systems, the DML is typically incorporated into some programming language, such as C++. Hence, the structures of both stored persistent objects and programminglanguage transient objects are often compatible. Query languages have been developed for Object Oriented databases. Work on a standard Object Oriented model and language is progressing, but no complete detailed standard has emerged as yet.

Part 2
UML

UML 2 Tutorial

UML 2 defines thirteen basic diagram types, divided into two general sets:

1. Structural Modeling Diagrams

Structure diagrams define the static architecture of a model. They are used to model the 'things' that make up a model - the classes, objects, interfaces and physical components. In addition, they are used to model the relationships and dependencies between elements.

Package diagrams are used to divide the model into logical containers, or 'packages', and describe the interactions between them at a high level.

Class or Structural diagrams define the basic building blocks of a model: the types, classes and general materials used to construct a full model.

Object diagrams show how instances of structural elements are related and used at run-time.

Composite Structure diagrams provide a means of layering an element's structure and focusing on inner detail, construction and relationships.

Component diagrams are used to model higher level or more complex structures, usually built up from one or more classes, and providing a well defined interface.

Deployment diagrams show the physical disposition of significant artifacts within a real-world setting.

2. Behavioral Modeling Diagrams

Behavior diagrams capture the varieties of interaction and instantaneous states within a model as it 'executes' over time; tracking how the system will act in a real-world environment, and observing the effects of an operation or event, including its results.

Use Case diagrams are used to model user/system interactions. They define behavior, requirements and constraints in the form of scripts or scenarios.

Activity diagrams have a wide number of uses, from defining basic program flow, to capturing the decision points and actions within any generalized process.

State Machine diagrams are essential to understanding the instant to instant condition, or "run state" of a model when it executes.

Communication diagrams show the network, and sequence, of messages or communications between objects at run-time, during a collaboration instance.

Sequence diagrams are closely related to communication diagrams and show the sequence of messages passed between objects using a vertical timeline.

Timing diagrams fuse sequence and state diagrams to provide a view of an object's state over time, and messages which modify that state.

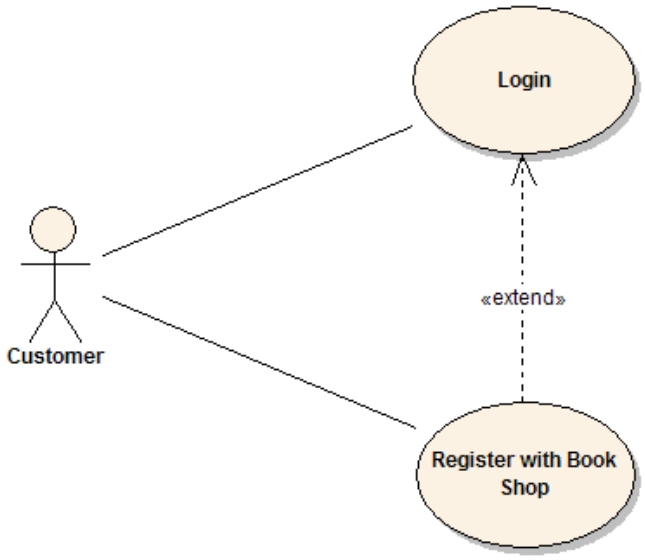
Interaction Overview diagrams fuse activity and sequence diagrams to allow interaction fragments to be easily combined with decision points and flows.

Table of contents

The Use Case Model

A Use Case Model describes the proposed functionality of a new system. A Use Case represents a discrete unit of interaction between a user (human or machine) and the system. This interaction is a single unit of meaningful work, such as Create Account or View Account Details.

Each Use Case describes the functionality to be built in the proposed system, which can include another Use Case's functionality or extend another Use Case with its own behavior.

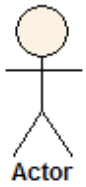


A Use Case description will generally includes:

- General comments and notes describing the use case.
- Requirements - The formal functional requirements of things that a Use Case must provide to the end user, such as <ability to update order>. These correspond to the functional specifications found in structured methodologies, and form a contract that the Use Case performs some action or provides some value to the system.
- Constraints - The formal rules and limitations a Use Case operates under, defining what can and cannot be done. These include:
- Pre-conditions that must have already occurred or be in place before the use case is run; for example, <create order> must precede <modify order>
- Post-conditions that must be true once the Use Case is complete; for example, <order is modified and consistent>
- Invariants that must always be true throughout the time the Use Case operates; for example, an order must always have a customer number.
- Scenarios – Formal, sequential descriptions of the steps taken to carry out the use case, or the flow of events that occur during a Use Case instance. These can include multiple scenarios, to cater for exceptional circumstances and alternative processing paths. These are usually created in text and correspond to a textual representation of the Sequence Diagram.
- Scenario diagrams - Sequence diagrams to depict the workflow; similar to Scenarios but graphically portrayed.
- Additional attributes, such as implementation phase, version number, complexity rating, stereotype and status.

Actors

Use Cases are typically related to 'actors', which are human or machine entities that use or interact with the system to perform a piece of meaningful work that helps them to achieve a goal. The set of Use Cases an actor has access to defines their overall role in the system and the scope of their action.



Includes and Extends relationships between Use Cases

One Use Case could include the functionality of another as part of its normal processing. Generally, it is assumed that the included Use Case is called every time the basic path is run. For example, when listing a set of customer orders to choose from before modifying a selected order, the <list orders> Use Case would be included every time the <modify order> Use Case is run.

A Use Case can be included by one or more other Use Cases, so it helps to reduce duplication of functionality by factoring out common behavior into Use Cases that are re-used many times.

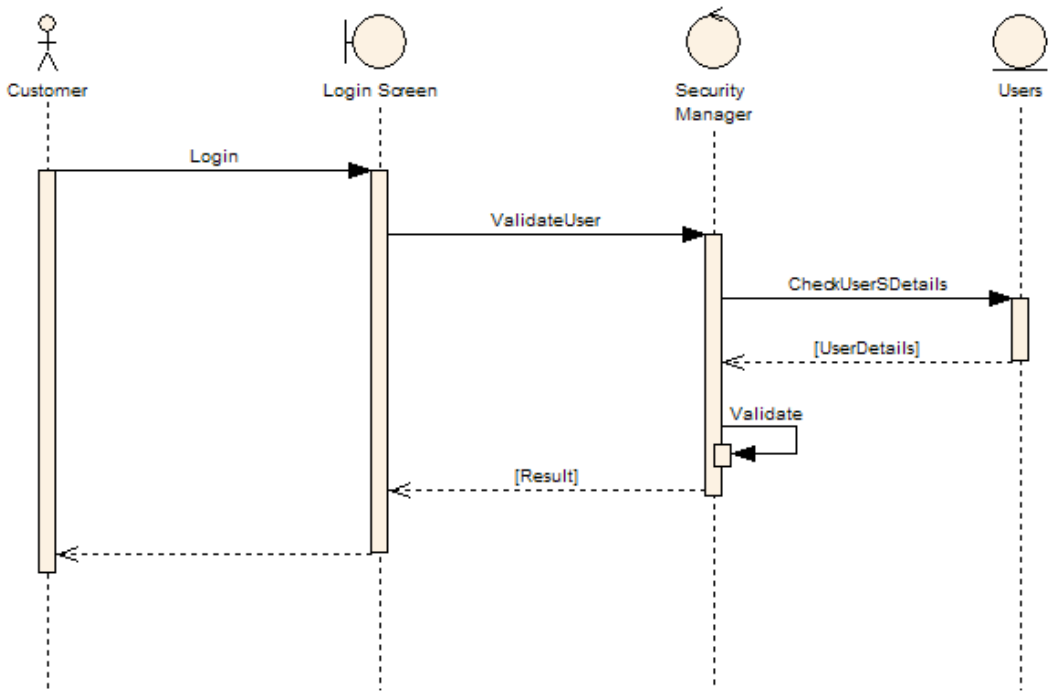
One Use Case can extend the behavior of another, typically when exceptional circumstances are encountered. For example, if a user must get approval from some higher authority before modifying a particular type of customer order, then the <get approval> Use Case could optionally extend the regular <modify order> Use Case.

1. Sequence Diagrams

Sequence diagrams provide a graphical representation of object interactions over time. These typically show a user or actor, and the objects and components they interact with in the execution of a use case. One sequence diagram typically represents a single Use Case 'scenario' or flow of events.

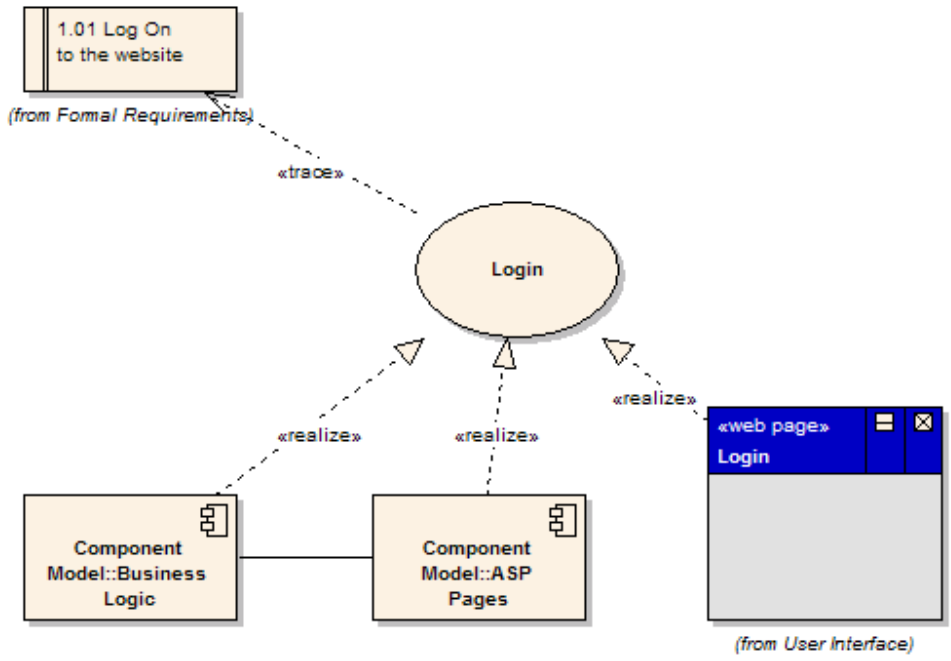
Sequence diagrams are an excellent way of documenting usage scenarios and both capturing required objects early in analysis and verifying object use later in design. The diagrams show the flow of messages from one object to another, and as such correspond to the methods and events supported by a class/object.

The following example of a sequence diagram shows the user or actor on the left initiating a flow of events and messages that correspond to the Use Case scenario. The messages that pass between objects become class operations in the final model.



2. Implementation Diagram

A Use Case is a formal description of functionality that the system will have when constructed. An implementation diagram is typically associated with a Use Case to document which design elements (for example, components and classes) implement the Use Case functionality in the new system. This provides a high level of traceability for the system designer, the customer and the team that will actually build the system. The list of Use Cases that a component or class is linked to documents the minimum functionality that must be implemented by the component.



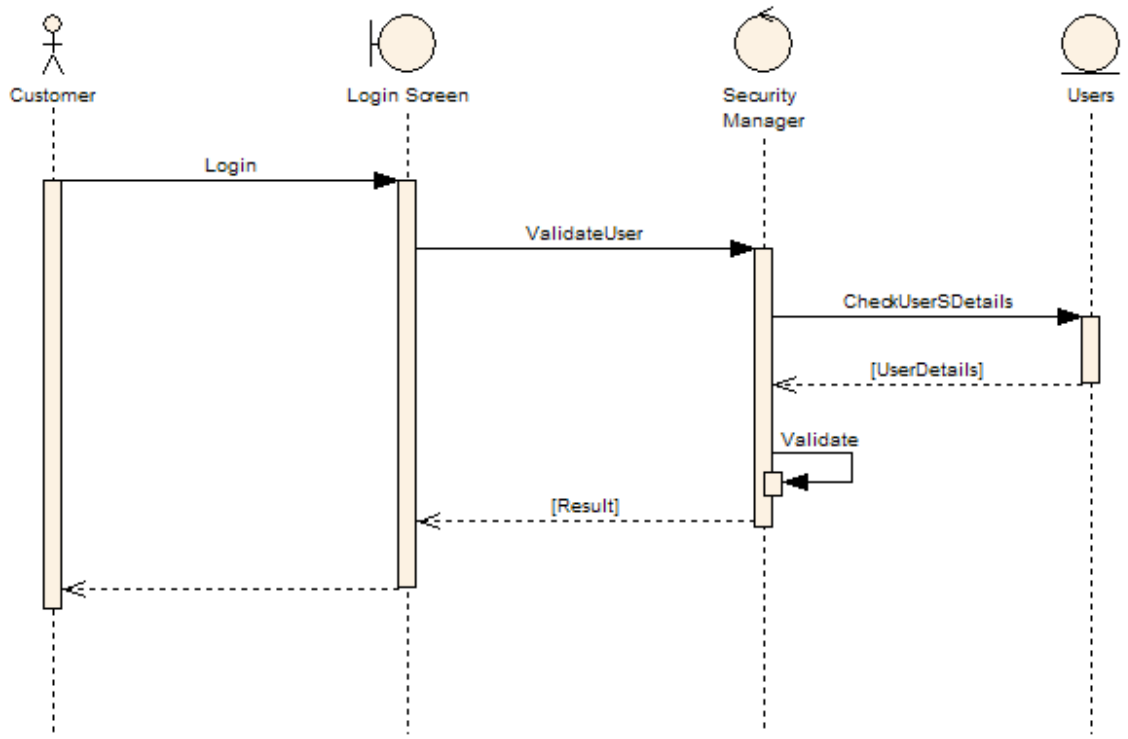
The example above shows that the use case 'Login' implements the formal requirement '1.01 Log On to the website'. It also shows that the 'Business Logic' component and 'ASP Pages' component implement some or all of the 'Login' functionality. A further refinement is to show the 'Login' screen (a web page) as implementing the 'Login' use case. These implementation or realization links define the traceability from the formal requirements, through use cases on to components and screens.

3. The Dynamic Model

The dynamic model is used to express and model the behaviour of the system over time. It includes support for activity diagrams, state diagrams, sequence diagrams and extensions including business process modelling.

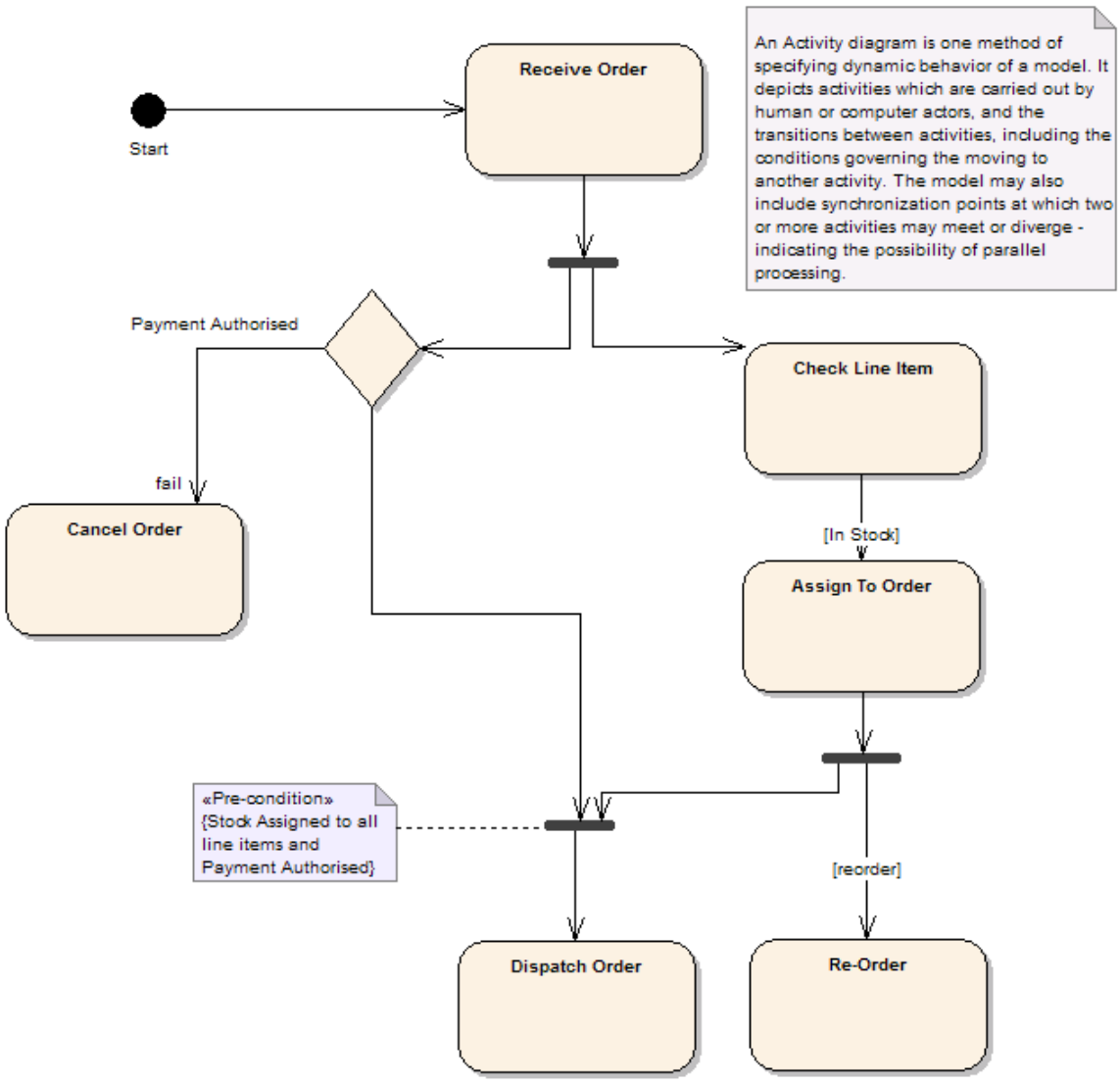
Sequence Diagrams

Sequence diagrams are used to display the interaction between users, screens, objects and entities within the system. It provides a sequential map of message passing between objects over time. Frequently these diagrams are placed under Use Cases in the model to illustrate the use case scenario - how a user will interact with the system and what happens internally to get the work done. Often, the objects are represented using special stereotyped icons, as in the example below. The object labelled Login Screen is shown using the User Interface icon. The object labelled SecurityManager is shown using the Controller icon. The Object labelled users is shown using the Entity icon.



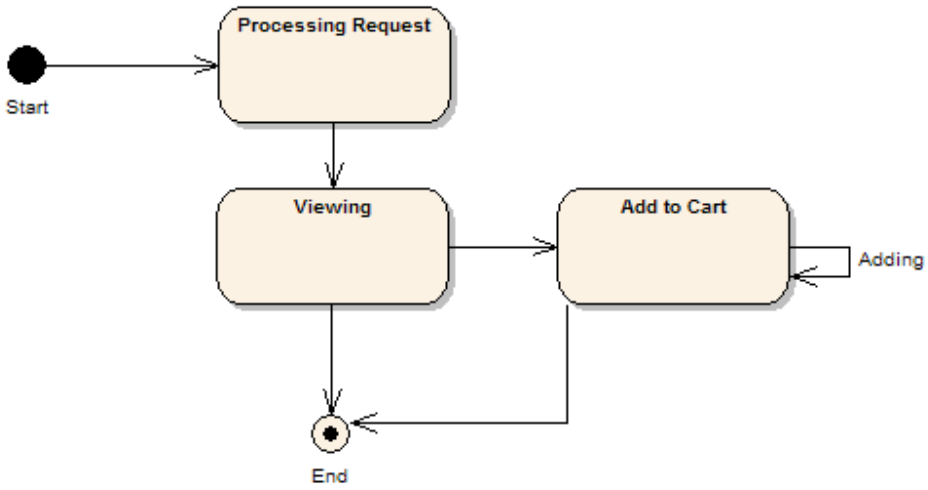
4. Activity Diagrams

Activity diagrams are used to show how different workflows in the system are constructed, how they start and the possibly many decision paths that can be taken from start to finish. They may also illustrate the where parallel processing may occur in the execution of some activities.



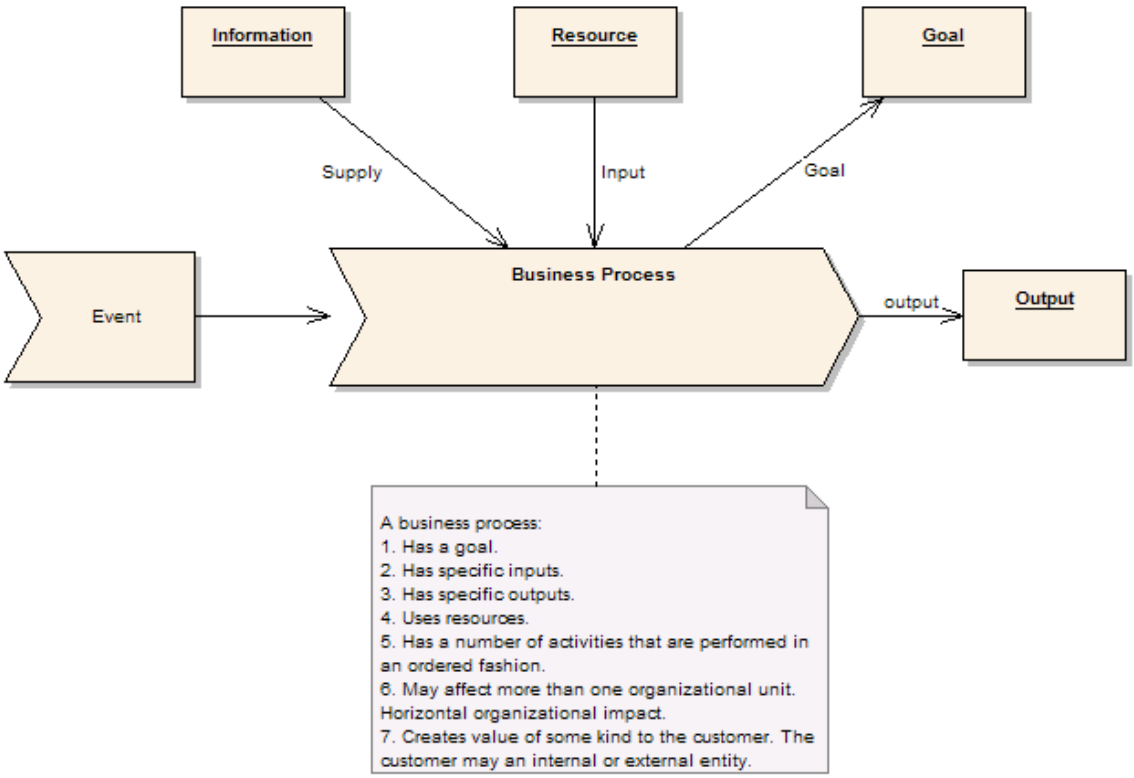
State Charts

State charts are used to detail the transitions or changes of state an object can go through in the system. They show how an object moves from one state to another and the rules that govern that change. State charts typically have a start and end condition.



Process Model

A process model is a UML extension of an activity diagram used to model a business process - this diagram shows what goal the process has, the inputs, outputs, events and information that are involved in the process.



The Logical Model

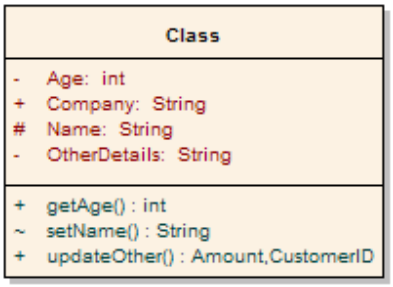
A logical model is a static view of the objects and classes that make up the design/analysis space. Typically, a Domain Model is a looser, high level view of Business Objects and entities, while the Class Model is a more rigorous and design focused model. This discussion relates mainly to the Class Model

The Class Model

A Class is a standard UML construct used to detail the pattern from which objects will be produced at run-time. A class is a specification - an object an instance of a class. Classes may be inherited from other classes (that is they inherit all the behavior and state of their parent and add new functionality of their own), have other classes as attributes, delegate responsibilities to other classes and implement abstract interfaces.

The Class Model is at the core of object-oriented development and design - it expresses both the persistent state of the system and the behavior of the system. A class encapsulates state (attributes) and offers services to manipulate that state (behavior). Good object-oriented design limits direct access to class attributes and offers services which manipulate attributes on behalf of the caller. This hiding of data and exposing of services ensures data updates are only done in one place and according to specific rules - for large systems the maintenance burden of code which has direct access to data elements in many places is extremely high.

The class is represented as below:



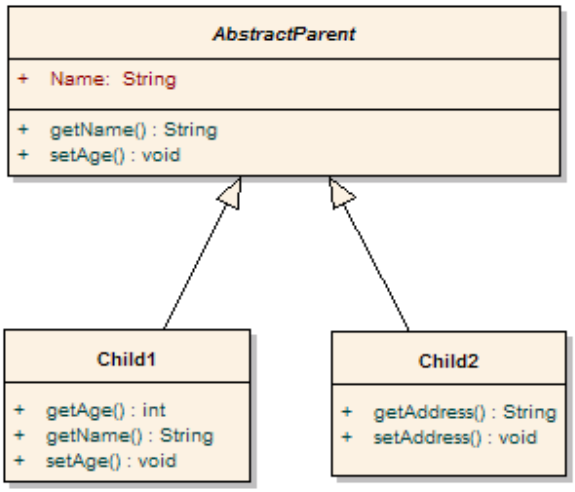
Note that the class has three distinct areas:

1. The class name (and stereotype if applied)
2. The class attributes area (that is internal data elements)
3. The behavior - both private and public

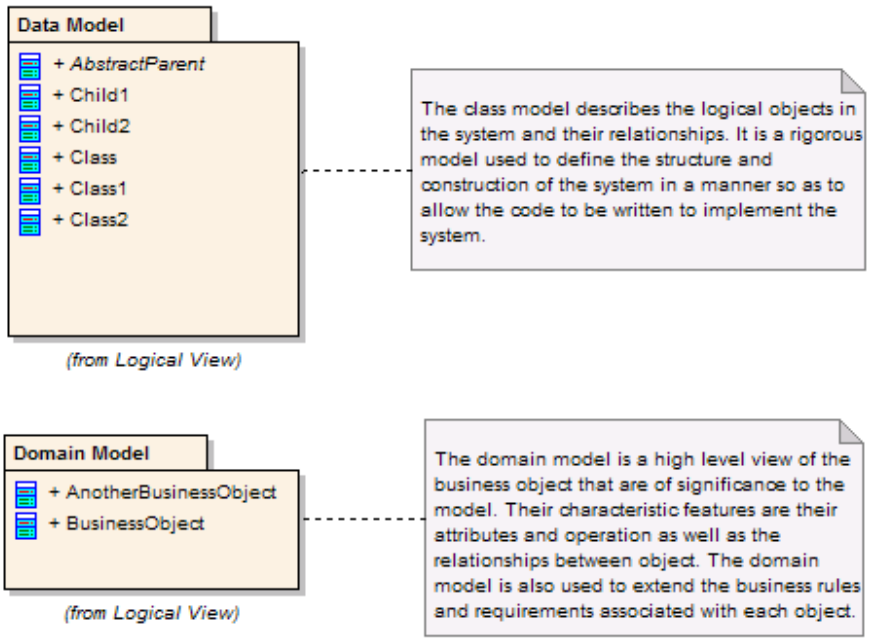
Attributes and methods may be marked as

- Private, indicating they are not visible to callers outside the class
- Protected, they are only visible to children of the class
- Public, they are visible to all

Class inheritance is shown as below: an abstract class in this case, is the parent of two children, each of which inherits the base class features and extends it with their own behavior.



Class models may be collected into packages of related behavior and state. The diagram below illustrates this.

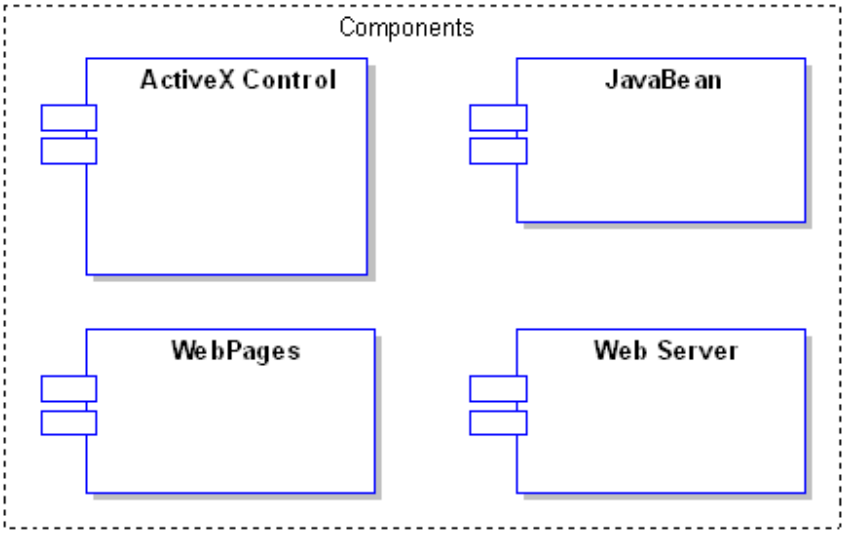


The Component Model

The component model illustrates the software components that will be used to build the system. These may be built up from the class model and written from scratch for the new system, or may be brought in from other projects and 3rd party vendors. Components are high level aggregations of smaller software pieces, and provide a 'black box' building block approach to software construction.

Component Notation

A component may be something like an ActiveX control - either a user interface control or a business rules server. Components are drawn as the following diagram shows:

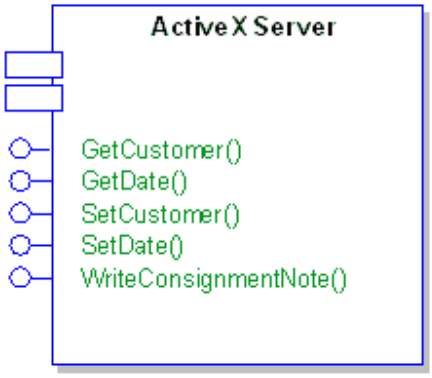


5. The Component Diagram

The component diagram shows the relationship between software components, their dependencies, communication, location and other conditions.

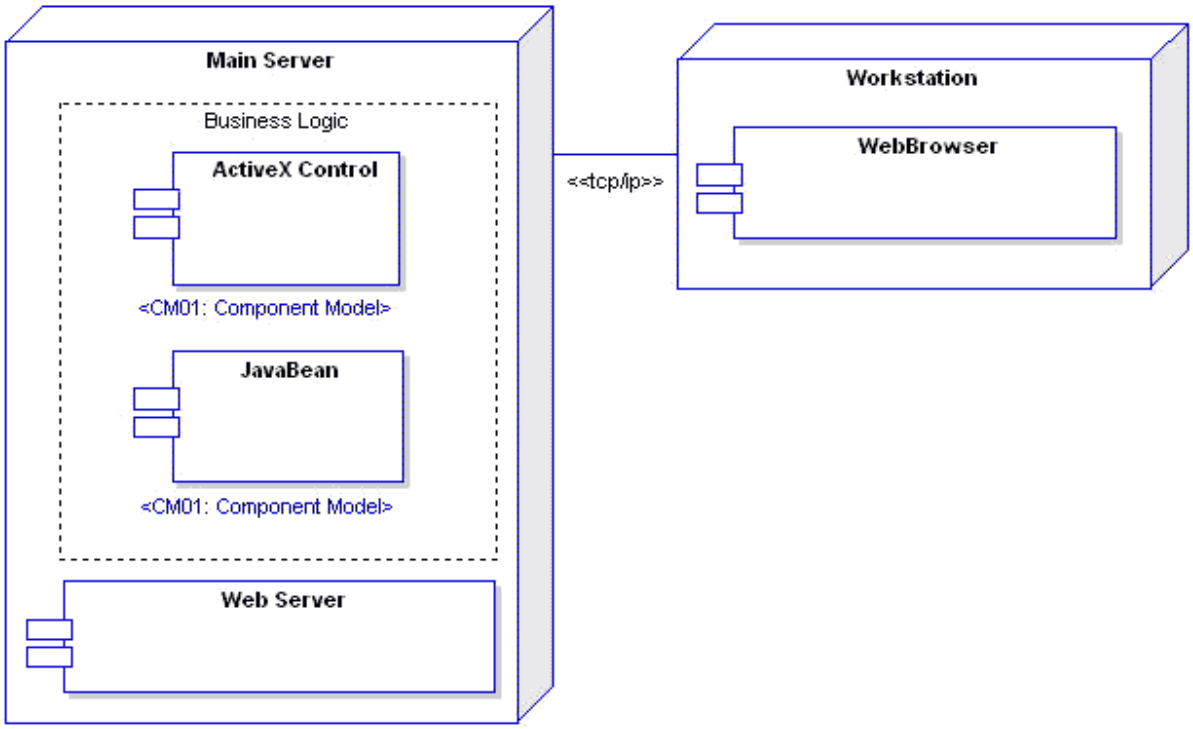
Interfaces

Components may also expose interfaces. These are the visible entry points or services that a component is advertising and making available to other software components and classes. Typically a component is made up of many internal classes and packages of classes. It may even be assembled from a collection of smaller components.



Components and Nodes

A deployment diagram illustrates the physical deployment of the system into a production (or test) environment. It shows where components will be located, on what servers, machines or hardware. It may illustrate network links, LAN bandwidth & etc.



Requirements
Components may have requirements attached to indicate their contractual obligations - that is, what service they will provide in the model. Requirements help document the functional behaviour of software elements.

Constraints
Components may have constraints attached which indicate the environment in which they operate. Pre-conditions specify what must be true before a component can perform some function; post-conditions indicate what will be true after a component has done some work and Invariants specify what must remain true for the duration of the components lifetime.

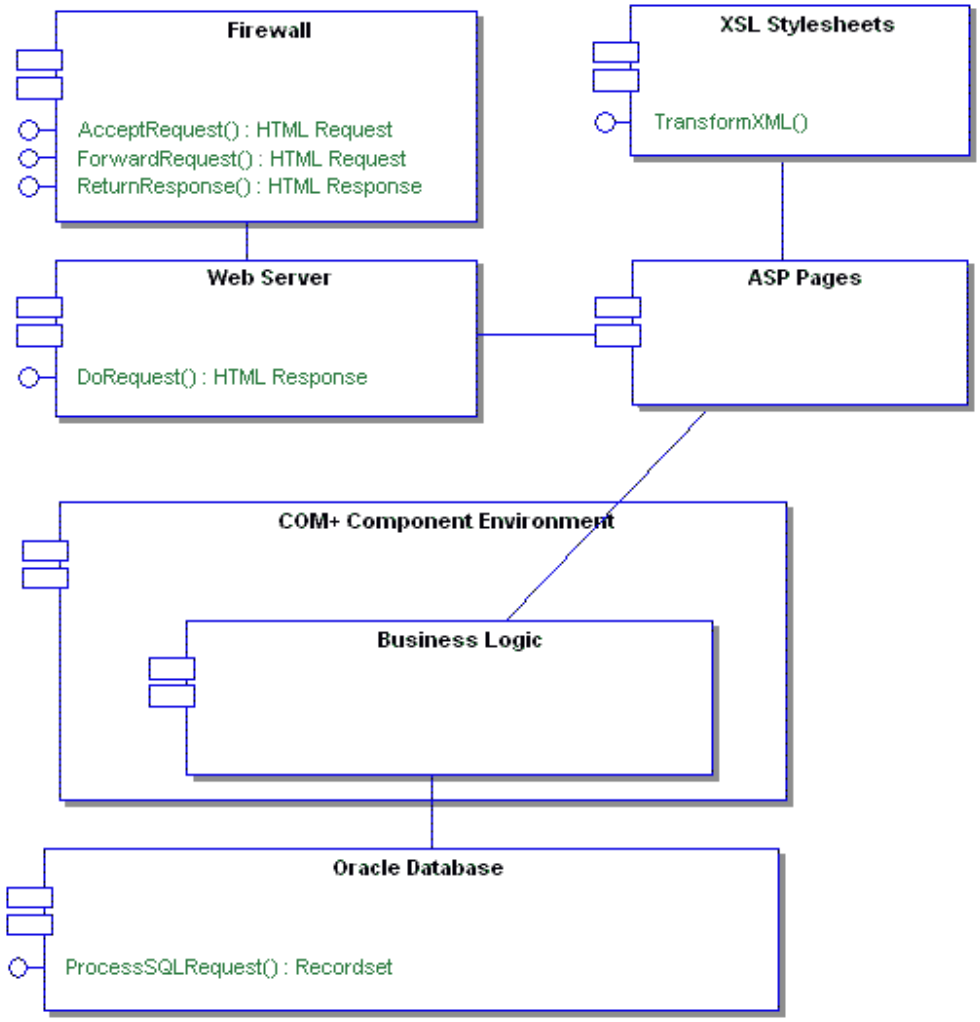
Scenarios
Scenarios are textual/procedural descriptions of an object's actions over time and describe the way in which a component works. Multiple scenarios may be created to describe the basic path (a perfect run through) as well as exceptions, errors and other conditions.

Traceability
You may indicate traceability through realisation links. A component may implement another model element (eg. a use case) or a component may be implemented by another element (eg. a package of classes). By providing realisation links to and from components you can map the dependencies amongst model elements and the traceability from the initial requirements to the final implementation.

An Example
The following example shows how components may be linked to provide a conceptual/logical view of a systems construction. This example is concerned with the server and security elements of an on-line book store. It includes such elements as the web server, firewall, ASP pages & etc.

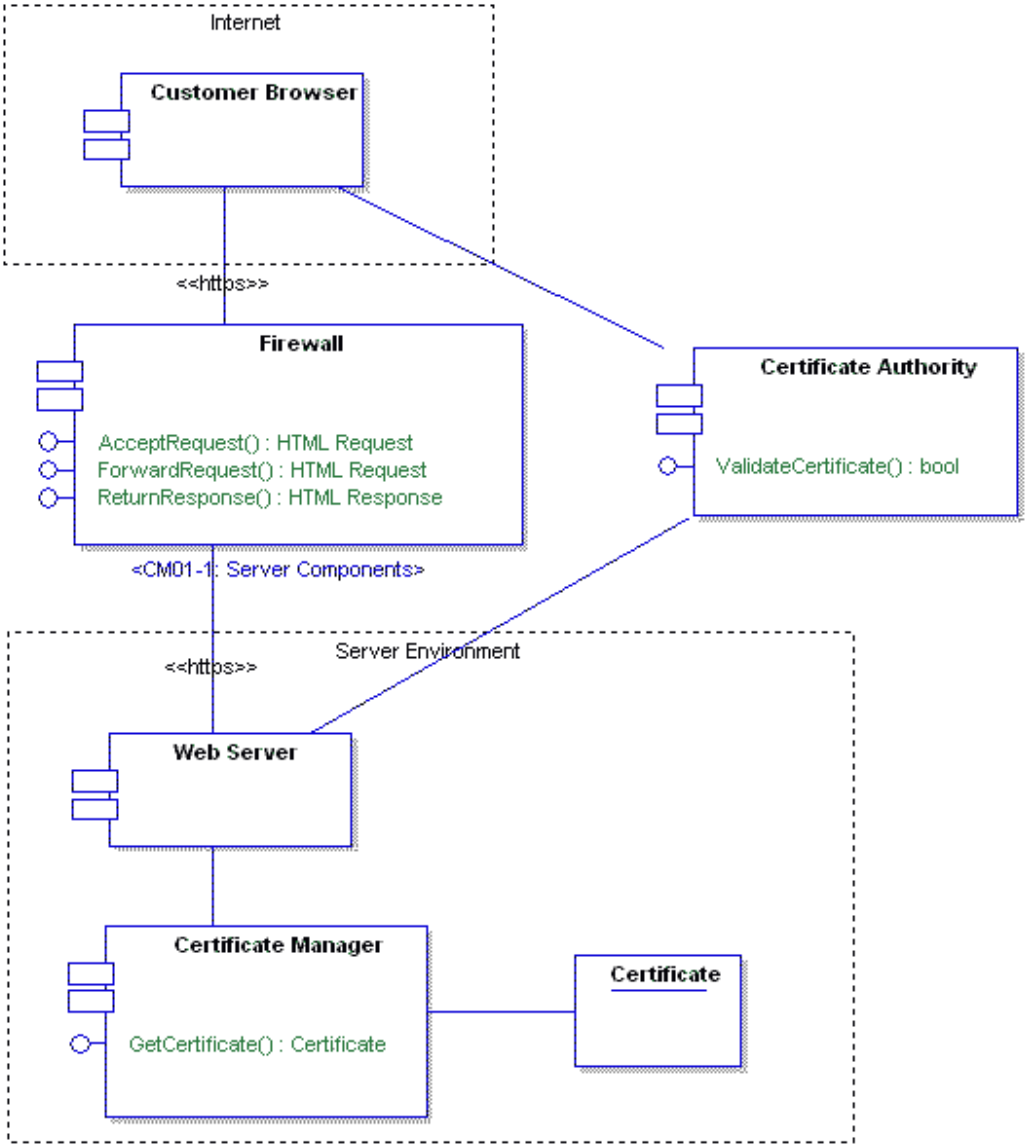
Server Components

This diagram illustrates the layout of the main server side components that will require building for an on-line book store. These components are a mixture of custom built and purchased items which will be assembled to provide the required functionality.



Security Components

The security components diagram shows how security software such as the Certificate Authority, Browser, Web server and other model elements work together to assure security provisions in the proposed system.

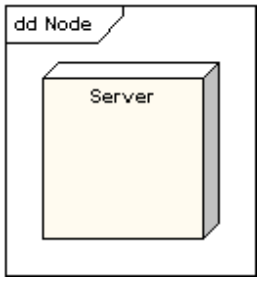


6. Deployment Diagram

Deployment Diagrams

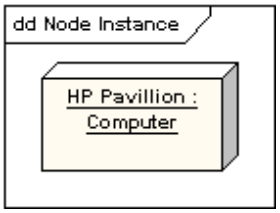
A deployment diagram models the run-time architecture of a system. It shows the configuration of the hardware elements (nodes) and shows how software elements and artifacts are mapped onto those nodes.

Node
A Node is either a hardware or software element. It is shown as a three-dimensional box shape, as shown below.



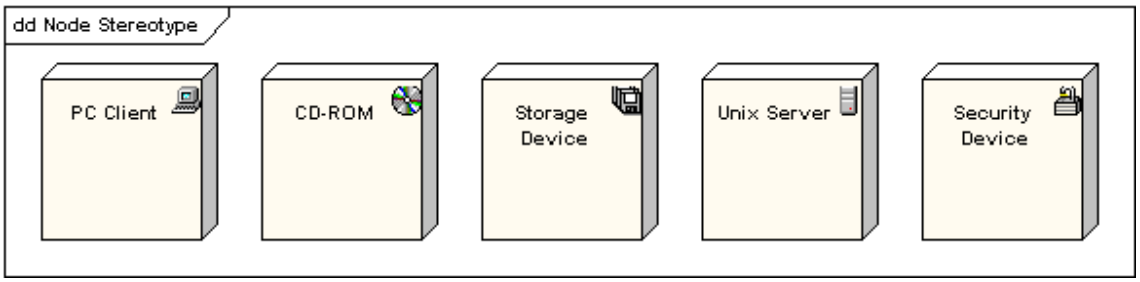
Node Instance

A node instance can be shown on a diagram. An instance can be distinguished from a node by the fact that its name is underlined and has a colon before its base node type. An instance may or may not have a name before the colon. The following diagram shows a named instance of a computer.



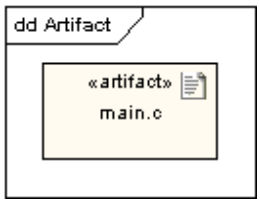
Node Stereotypes

A number of standard stereotypes are provided for nodes, namely «cdrom», «cd-rom», «computer», «disk array», «pc», «pc client», «pc server», «secure», «server», «storage», «unix server», «user pc». These will display an appropriate icon in the top right corner of the node symbol

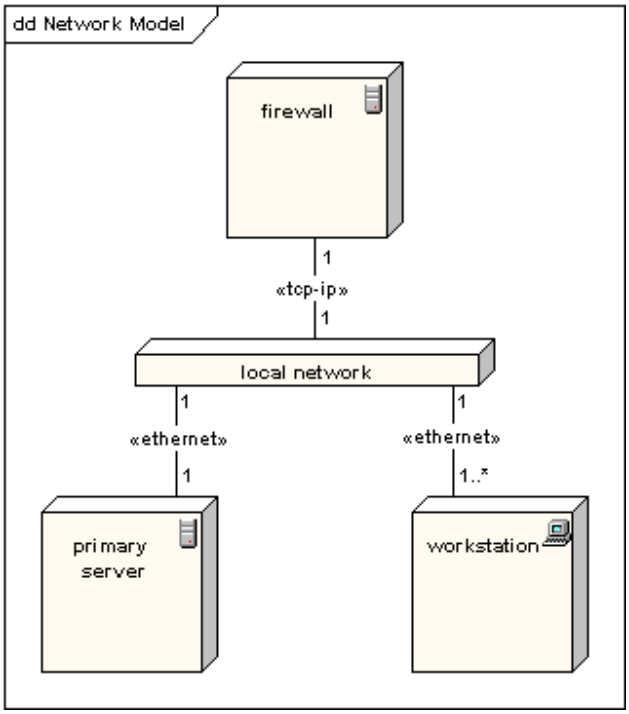


Artifact
An artifact is a product of the software development process. That may include process models (e.g. use case models, design models etc), source files, executables, design documents, test reports, prototypes, user manuals, etc.

An artifact is denoted by a rectangle showing the artifact name, the «artifact» keyword and a document icon, as shown below.

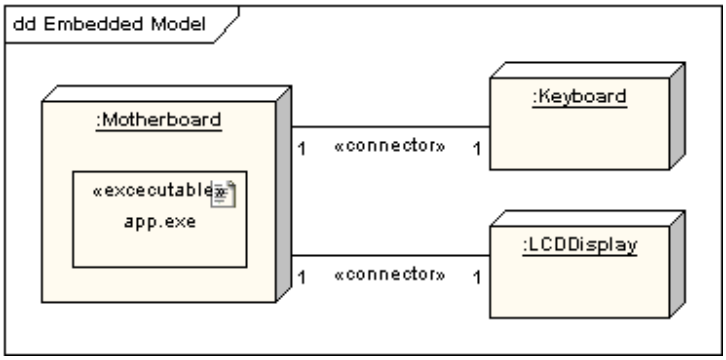


Association
In the context of a deployment diagram, an association represents a communication path between nodes. The following diagram shows a deployment diagram for a network, depicting network protocols as stereotypes, and multiplicities at the association ends.



Node as Container

A node can contain other elements, such as components or artifacts. The following diagram shows a deployment diagram for part of an embedded system, depicting an executable artifact as being contained by the motherboard node.

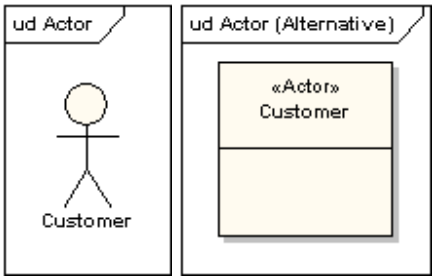


7. Use Case Diagram

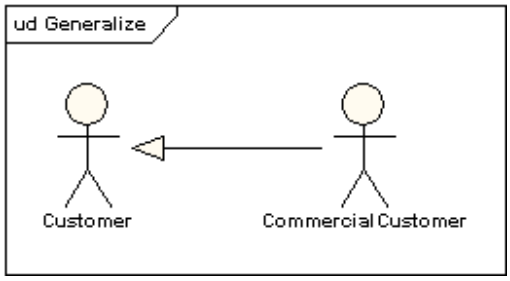
Use Case Model

The use case model captures the requirements of a system. Use cases are a means of communicating with users and other stakeholders what the system is intended to do.

Actors
A use case diagram shows the interaction between the system and entities external to the system. These external entities are referred to as actors. Actors represent roles which may include human users, external hardware or other systems. An actor is usually drawn as a named stick figure, or alternatively as a class rectangle with the «actor» keyword.

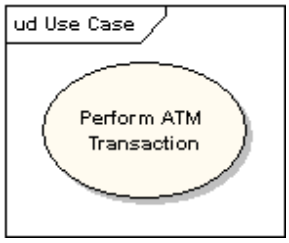


Actors can generalize other actors as detailed in the following diagram:

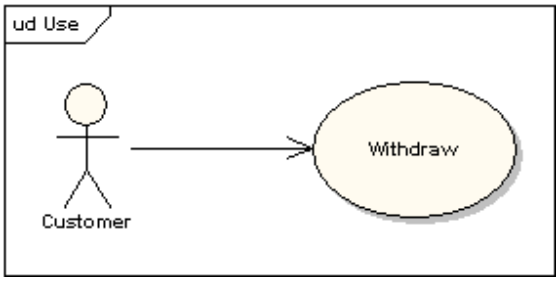


Use Cases

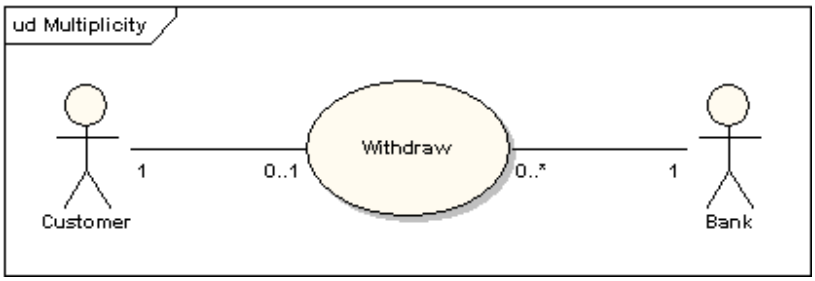
A use case is a single unit of meaningful work. It provides a high-level view of behavior observable to someone or something outside the system. The notation for a use case is an ellipse.



The notation for using a use case is a connecting line with an optional arrowhead showing the direction of control. The following diagram indicates that the actor "Customer" uses the "Withdraw" use case.



The uses connector can optionally have multiplicity values at each end, as in the following diagram, which shows a customer may only have one withdrawal session at a time, but a bank may have any number of customers making withdrawals concurrently.



Use Case Definition
A use case typically Includes:

- Name and description
- Requirements
- Constraints
- Scenarios
- Scenario diagrams
- Additional information.

Name and Description

A use case is normally named as a verb-phrase and given a brief informal textual description.

Requirements
The requirements define the formal functional requirements that a use case must supply to the end user. They correspond to the functional specifications found in structured methodologies. A

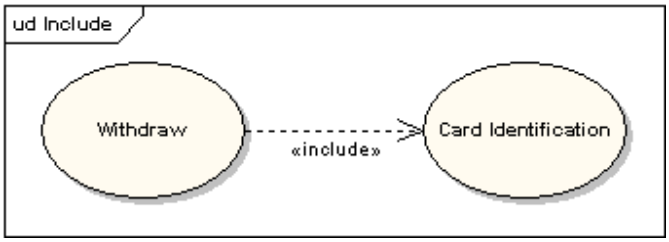
requirement is a contract or promise that the use case will perform an action or provide some value to the system.

Constraints
A constraint is a condition or restriction that a use case operates under and includes pre-, post- and invariant conditions. A precondition specifies the conditions that need to be met before the use case can proceed. A post-condition is used to document the change in conditions that must be true after the execution of the use case. An invariant condition specifies the conditions that are true throughout the execution of the use case.

Scenarios
A Scenario is a formal description of the flow of events that occur during the execution of a use case instance. It defines the specific sequence of events between the system and the external actors. It is normally described in text and corresponds to the textual representation of the sequence diagram.

Including Use Cases

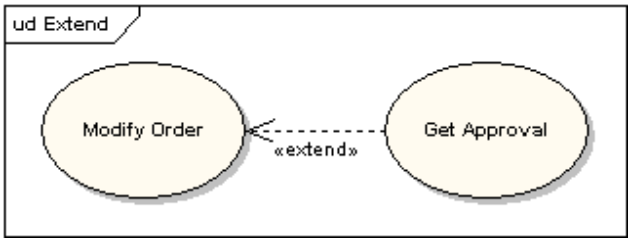
Use cases may contain the functionality of another use case as part of their normal processing. In general it is assumed that any included use case will be called every time the basic path is run. An example of this is to have the execution of the use case <Card Identification> to be run as part of a use case <Withdraw>.



Use Cases may be included by one or more Use Case, helping to reduce the level of duplication of functionality by factoring out common behavior into Use Cases that are re-used many times.

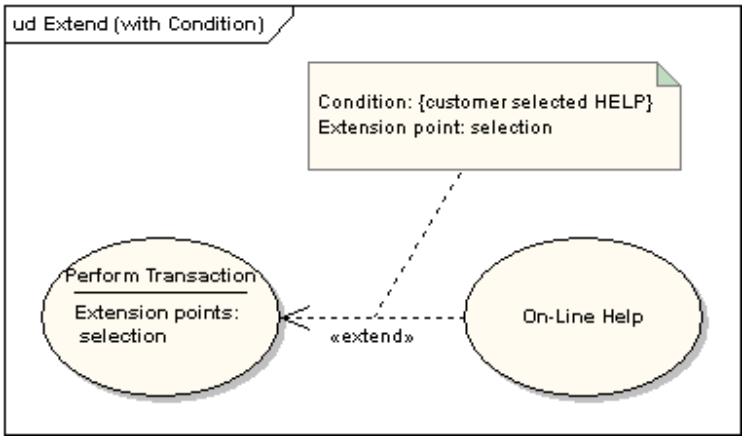
Extending Use Cases

One use case may be used to extend the behavior of another; this is typically used in exceptional circumstances. For example, if before modifying a particular type of customer order, a user must get approval from some higher authority, then the <Get Approval> use case may optionally extend the regular <Modify Order> use case.



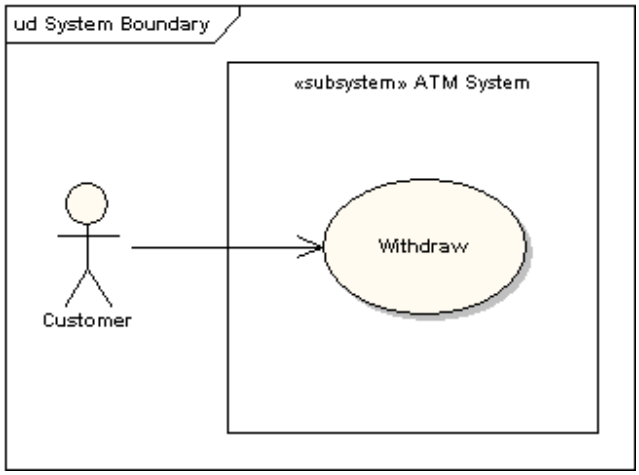
Extension Points

The point at which an extending use case is added can be defined by means of an extension point.



System Boundary

It is usual to display use cases as being inside the system and actors as being outside the system.

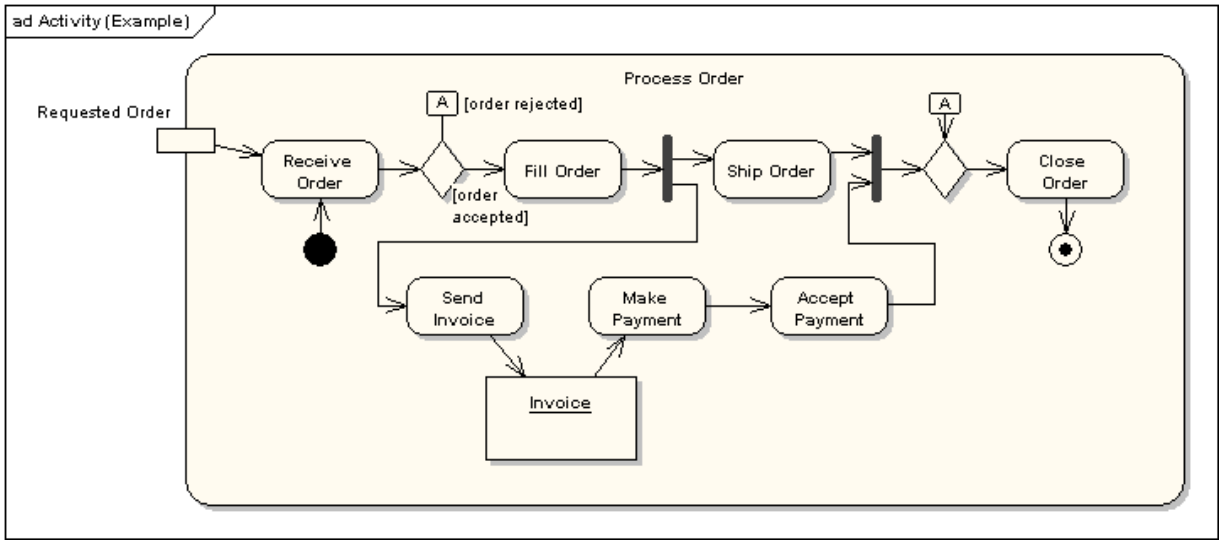


8. Activity Diagram

Activity Diagrams

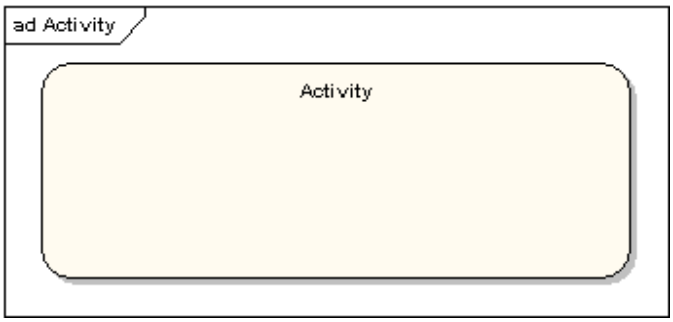
In UML, an activity diagram is used to display the sequence of activities. Activity diagrams show the workflow from a start point to the finish point detailing the many decision paths that exist in the progression of events contained in the activity. They may be used to detail situations where parallel processing may occur in the execution of some activities. Activity diagrams are useful for business modelling where they are used for detailing the processes involved in business activities.

An Example of an activity diagram is shown below.

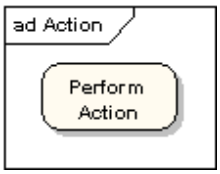


The following sections describe the elements that constitute an activity diagram.

Activities
An activity is the specification of a parameterized sequence of behaviour. An activity is shown as a round-cornered rectangle enclosing all the actions, control flows and other elements that make up the activity.

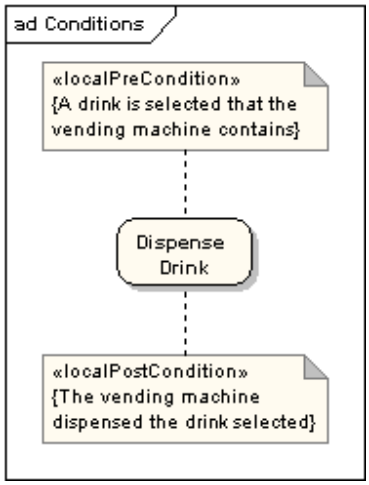


Actions
An action represents a single step within an activity. Actions are denoted by round-cornered rectangles.



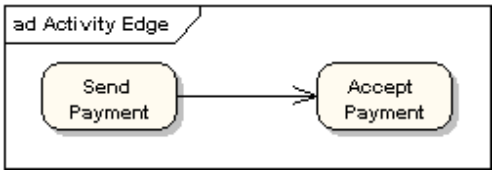
Action Constraints

Constraints can be attached to an action. The following diagram shows an action with local pre- and post-conditions.



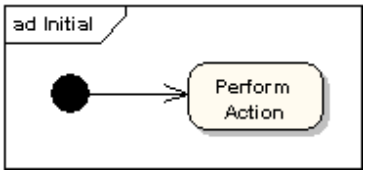
Control Flow

A control flow shows the flow of control from one action to the next. Its notation is a line with an arrowhead.



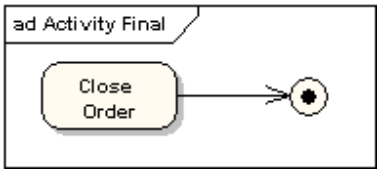
Initial Node

An initial or start node is depicted by a large black spot, as shown below.

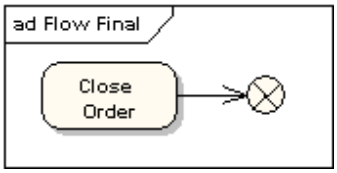


Final Node

There are two types of final node: activity and flow final nodes. The activity final node is depicted as a circle with a dot inside.



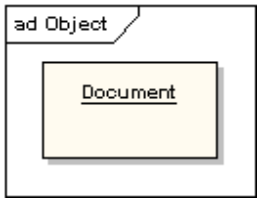
The flow final node is depicted as a circle with a cross inside.



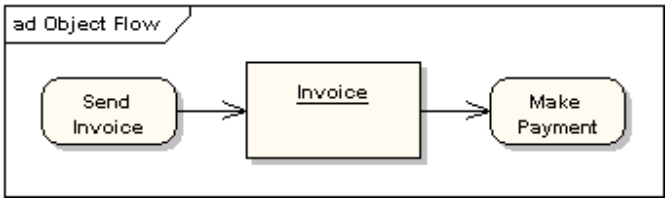
The difference between the two node types is that the flow final node denotes the end of a single control flow; the activity final node denotes the end of all control flows within the activity.

Objects and Object Flows

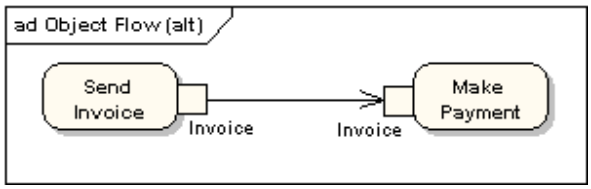
An object flow is a path along which objects or data can pass. An object is shown as a rectangle.



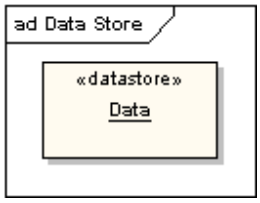
An object flow is shown as a connector with an arrowhead denoting the direction the object is being passed.



An object flow must have an object on at least one of its ends. A shorthand notation for the above diagram would be to use input and output pins.



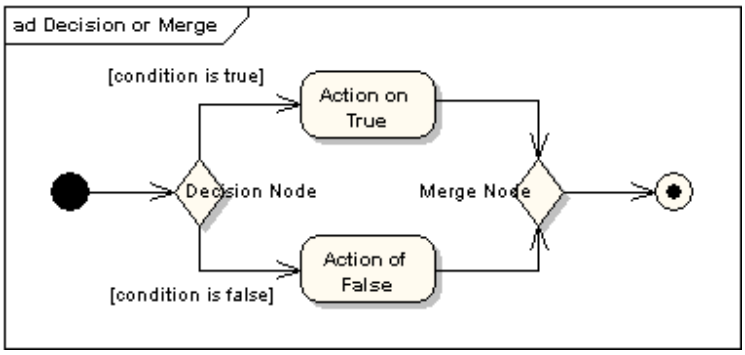
A data store is shown as an object with the «datastore» keyword.



Decision and Merge Nodes

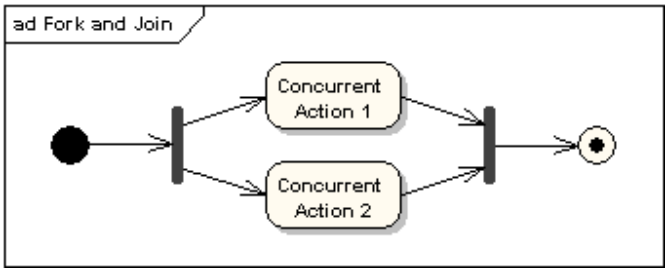
Decision nodes and merge nodes have the same notation: a diamond shape. They can both be named. The control flows coming away from a decision node will have guard conditions which will

allow control to flow if the guard condition is met. The following diagram shows use of a decision node and a merge node.



Fork and Join Nodes

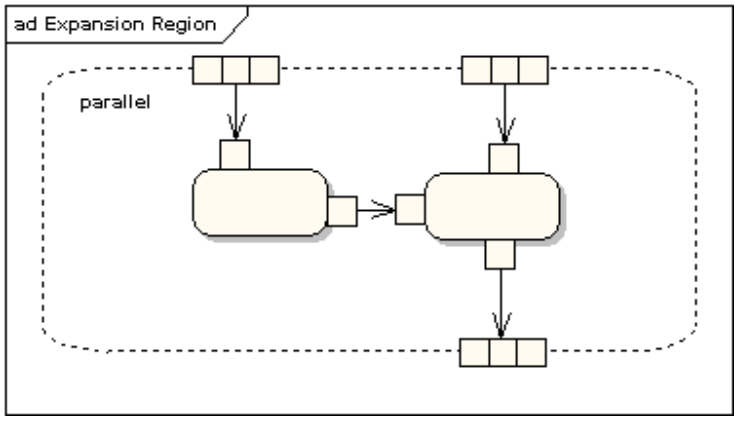
Forks and joins have the same notation: either a horizontal or vertical bar (the orientation is dependent on whether the control flow is running left to right or top to bottom). They indicate the start and end of concurrent threads of control. The following diagram shows an example of their use.



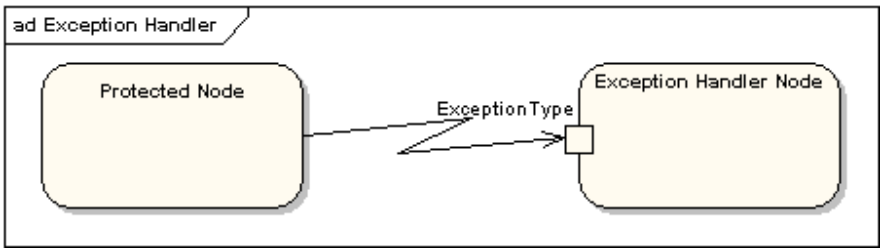
A join is different from a merge in that the join synchronizes two inflows and produces a single outflow. The outflow from a join cannot execute until all inflows have been received. A merge passes any control flows straight through it. If two or more inflows are received by a merge symbol, the action pointed to by its outflow is executed two or more times.

Expansion Region

An expansion region is a structured activity region that executes multiple times. Input and output expansion nodes are drawn as a group of three boxes representing a multiple selection of items. The keyword "iterative", "parallel" or "stream" is shown in the top left corner of the region.

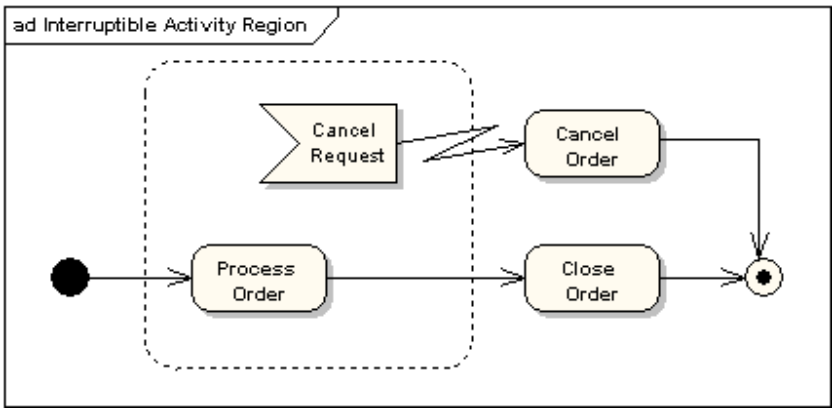


Exception Handlers
Exception Handlers can be modelled on activity diagrams as in the example below.

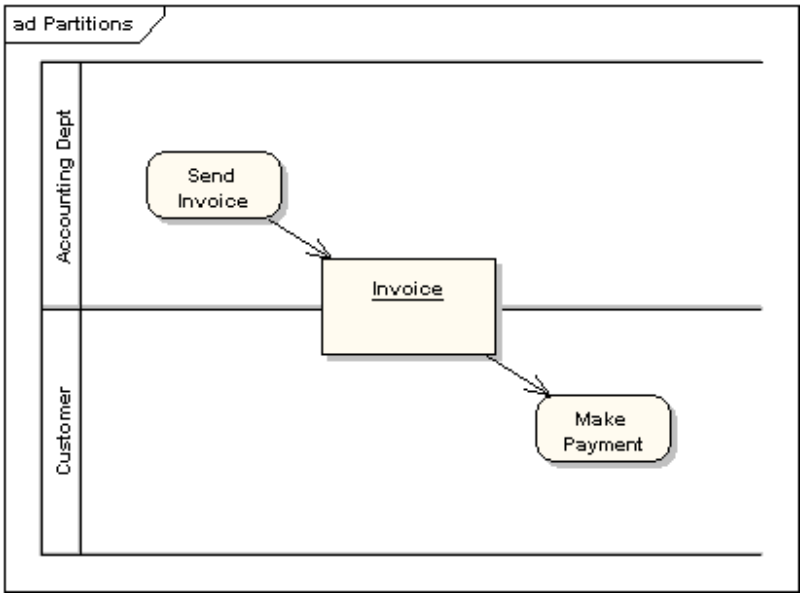


Interruptible Activity Region

An interruptible activity region surrounds a group of actions that can be interrupted. In the very simple example below, the "Process Order" action will execute until completion, when it will pass control to the "Close Order" action, unless a "Cancel Request" interrupt is received, which will pass control to the "Cancel Order" action.



Partition
An activity partition is shown as either a horizontal or vertical swimlane. In the following diagram, the partitions are used to separate actions within an activity into those performed by the accounting department and those performed by the customer.

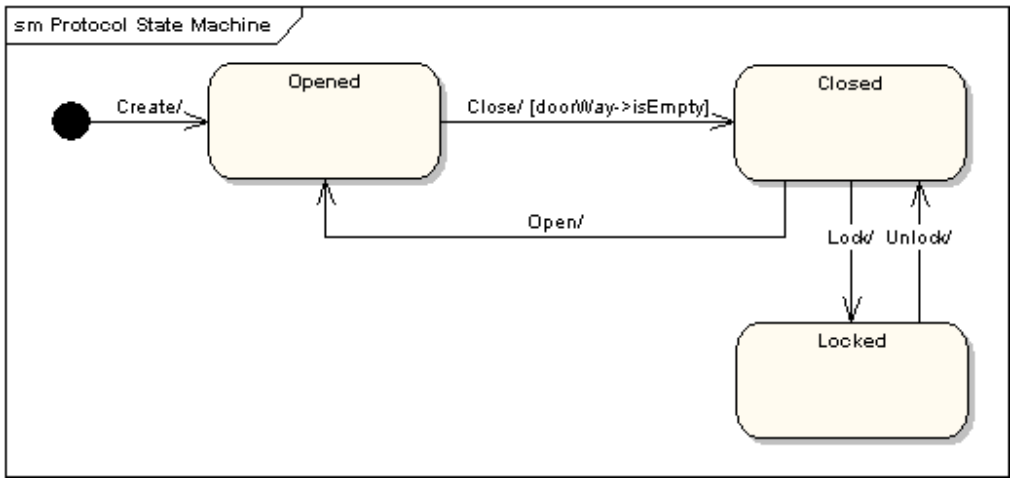


9. State Machine Diagram

State Machine Diagrams

A state machine diagram models the behaviour of a single object, specifying the sequence of events that an object goes through during its lifetime in response to events.

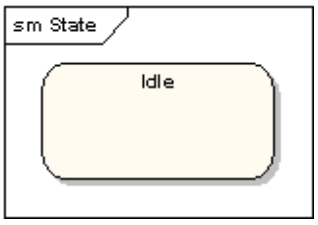
As an example, the following state machine diagram shows the states that a door goes through during its lifetime.



The door can be in one of three states: "Opened", "Closed" or "Locked". It can respond to the events Open, Close, Lock and Unlock. Notice that not all events are valid in all states; for example, if a door is opened, you cannot lock it until you close it. Also notice that a state transition can have a guard condition attached: if the door is Opened, it can only respond to the Close event if the condition doorWay->isEmpty is fulfilled. The syntax and conventions used in state machine diagrams will be discussed in full in the following sections.

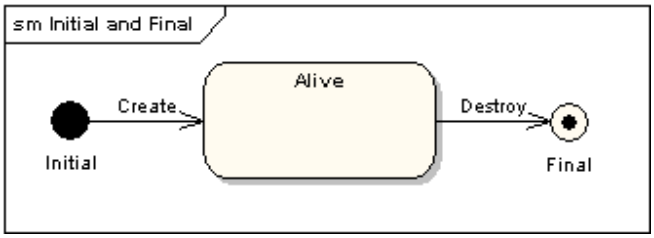
States

A state is denoted by a round-cornered rectangle with the name of the state written inside it.



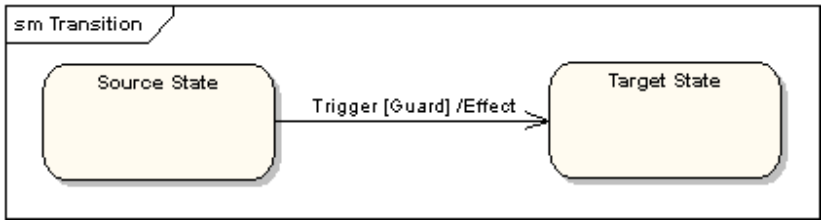
Initial and Final States

The initial state is denoted by a filled black circle and may be labeled with a name. The final state is denoted by a circle with a dot inside and may also be labeled with a name.



Transitions

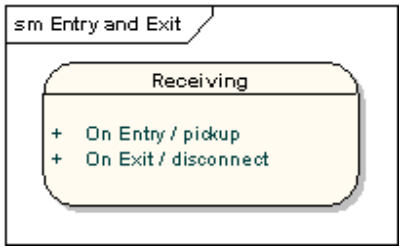
Transitions from one state to the next are denoted by lines with arrowheads. A transition may have a trigger, a guard and an effect, as below.



"Trigger" is the cause of the transition, which could be a signal, an event, a change in some condition, or the passage of time. "Guard" is a condition which must be true in order for the trigger to cause the transition. "Effect" is an action which will be invoked directly on the object that owns the state machine as a result of the transition.

State Actions

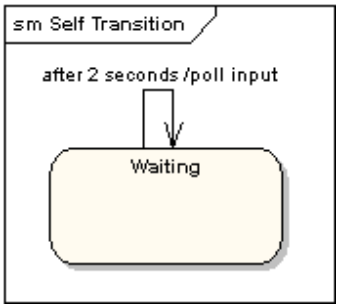
In the transition example above, an effect was associated with the transition. If the target state had many transitions arriving at it, and each transition had the same effect associated with it, it would be better to associate the effect with the target state rather than the transitions. This can be done by defining an entry action for the state. The diagram below shows a state with an entry action and an exit action.



It is also possible to define actions that occur on events, or actions that always occur. It is possible to define any number of actions of each type.

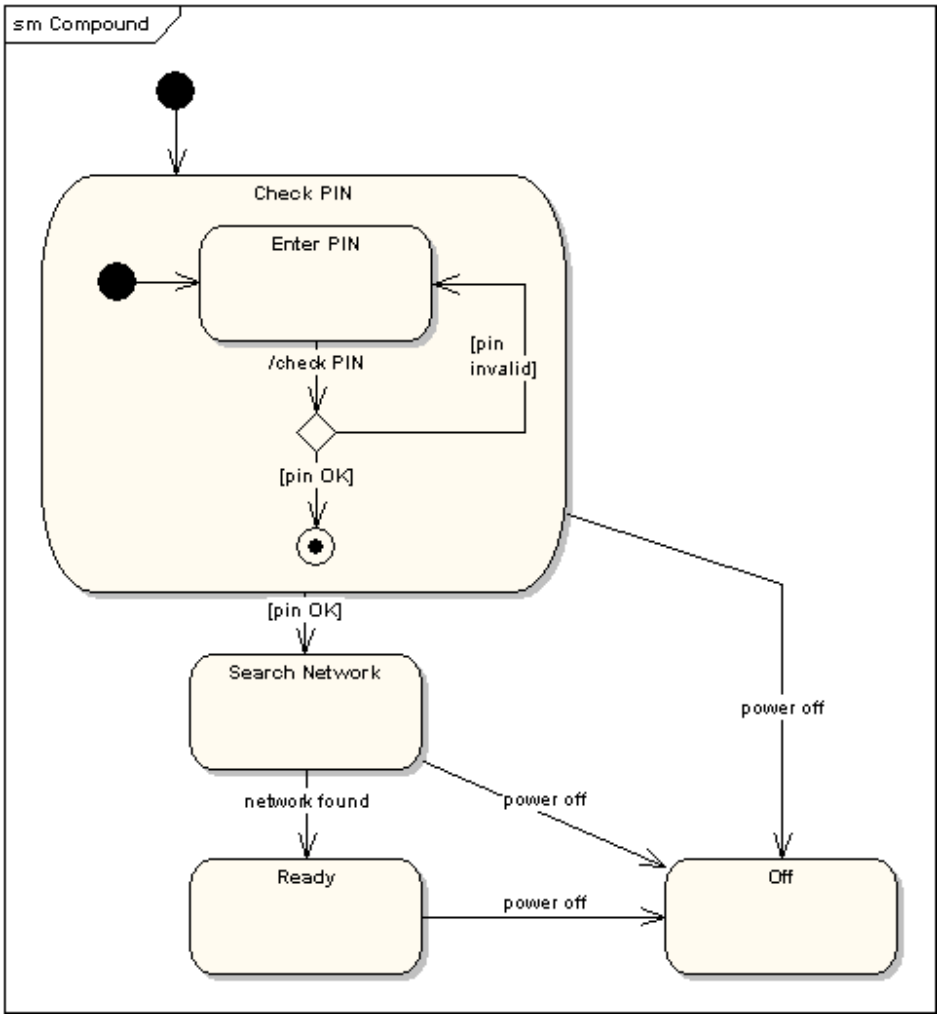
Self-Transitions

A state can have a transition that returns to itself, as in the following diagram. This is most useful when an effect is associated with the transition.

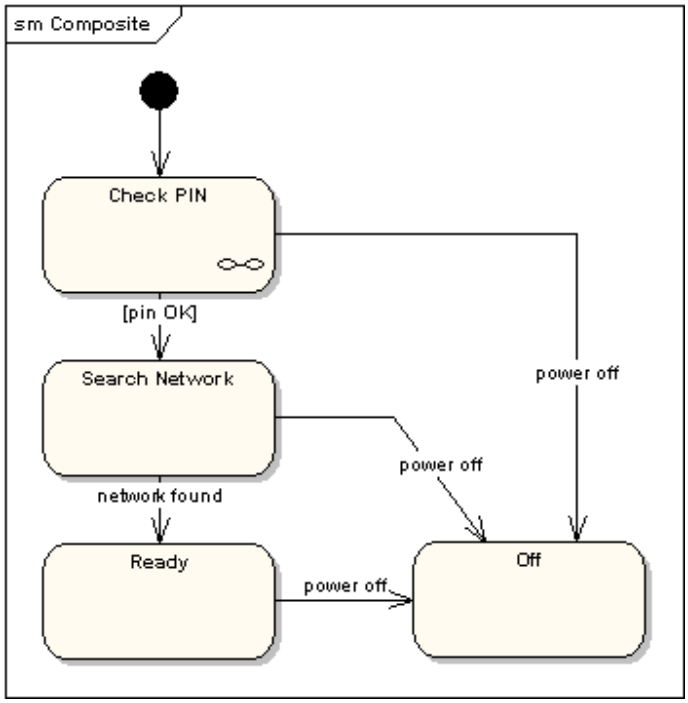


Compound States

A state machine diagram may include sub-machine diagrams, as in the example below.



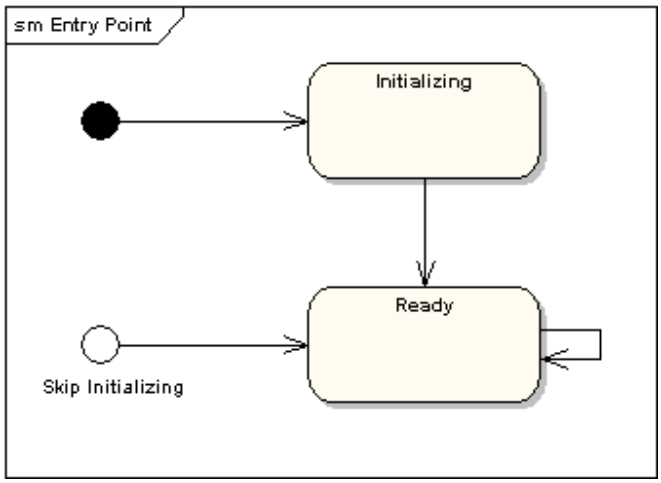
The alternative way to show the same information is as follows.



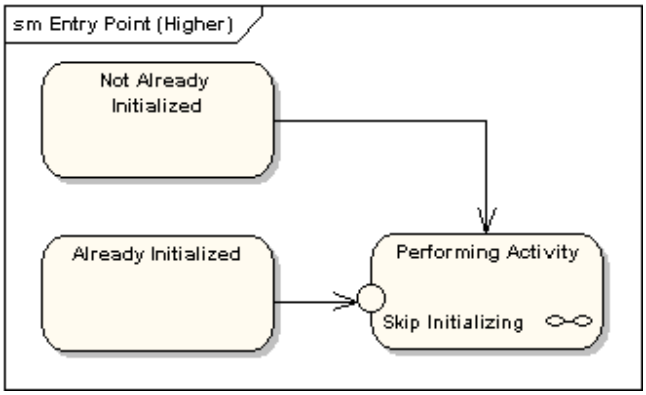
The notation in the above version indicates that the details of the Check PIN sub-machine are shown in a separate diagram.

Entry Point

Sometimes you won't want to enter a sub-machine at the normal initial state. For example, in the following sub-machine it would be normal to begin in the "Initializing" state, but if for some reason it wasn't necessary to perform the initialization, it would be possible to begin in the "Ready" state by transitioning to the named entry point.

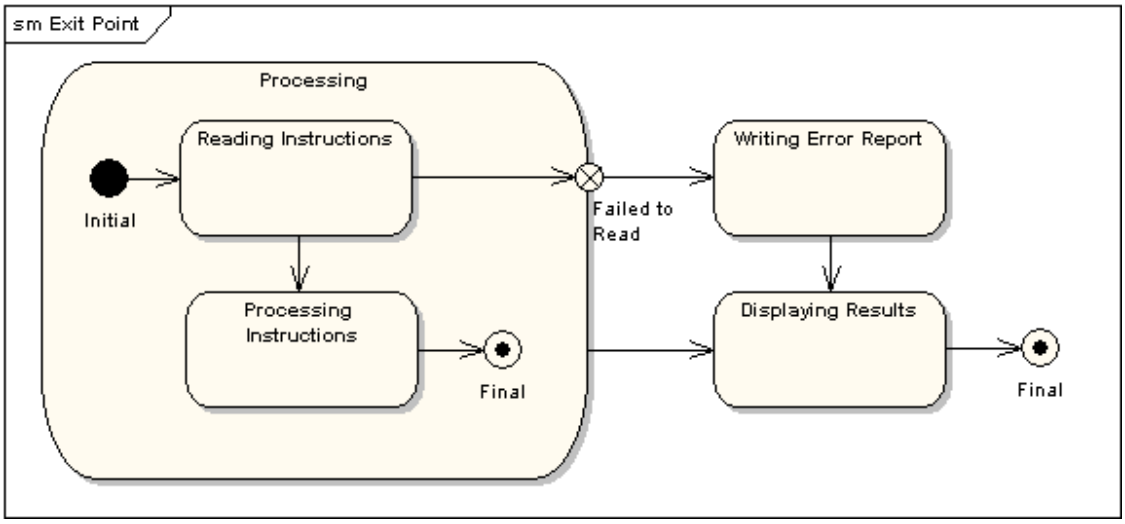


The following diagram shows the state machine one level up.



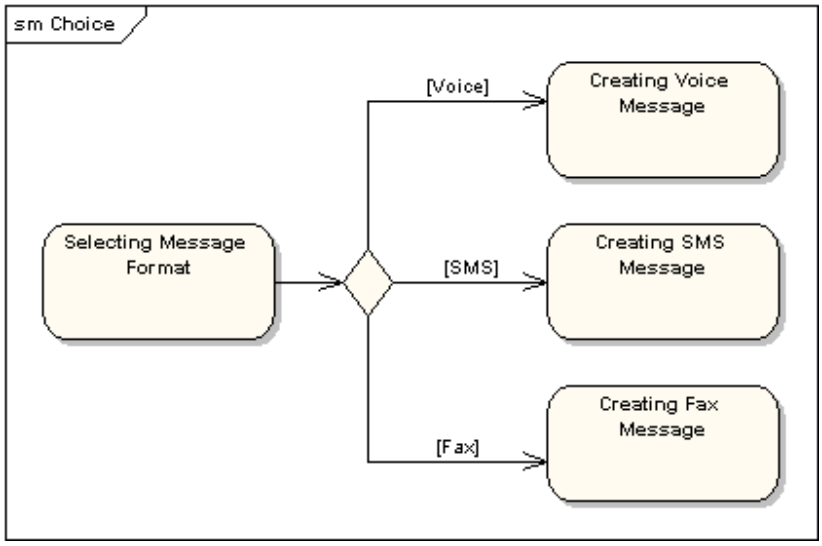
Exit Point

In a similar manner to entry points, it is possible to have named alternative exit points. The following diagram gives an example where the state executed after the main processing state depends on which route is used to transition out of the state.



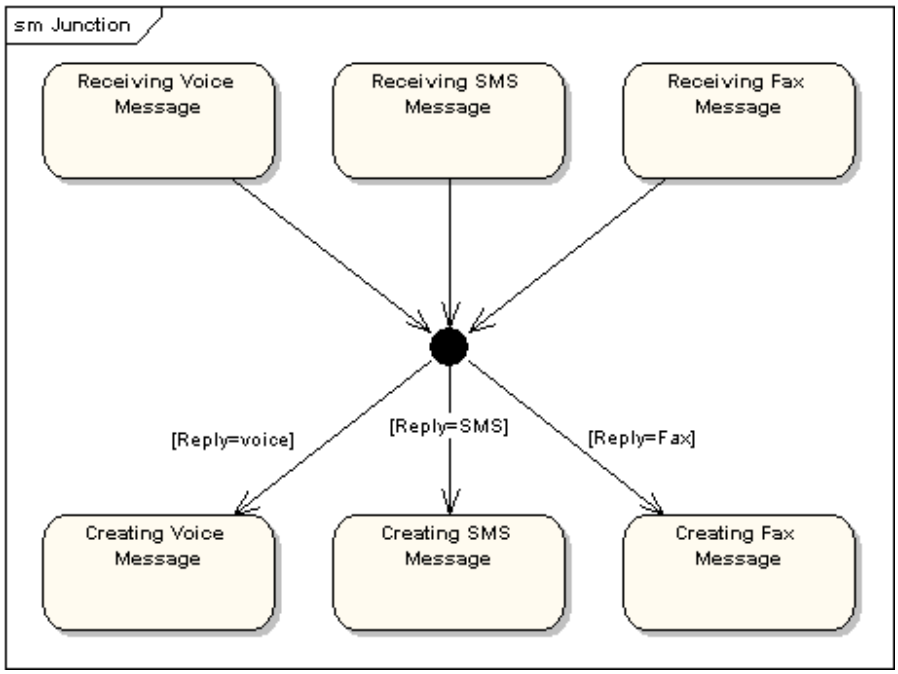
Choice Pseudo-State

A choice pseudo-state is shown as a diamond with one transition arriving and two or more transitions leaving. The following diagram shows that whichever state is arrived at, after the choice pseudo-state, is dependent on the message format selected during execution of the previous state.



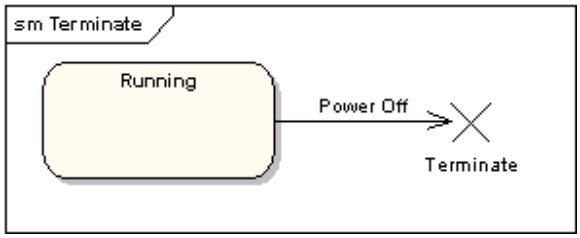
Junction Pseudo-State

Junction pseudo-states are used to chain together multiple transitions. A single junction can have one or more incoming, and one or more outgoing, transitions; a guard can be applied to each transition. Junctions are semantic-free. A junction which splits an incoming transition into multiple outgoing transitions realizes a static conditional branch, as opposed to a choice pseudo-state which realizes a dynamic conditional branch.



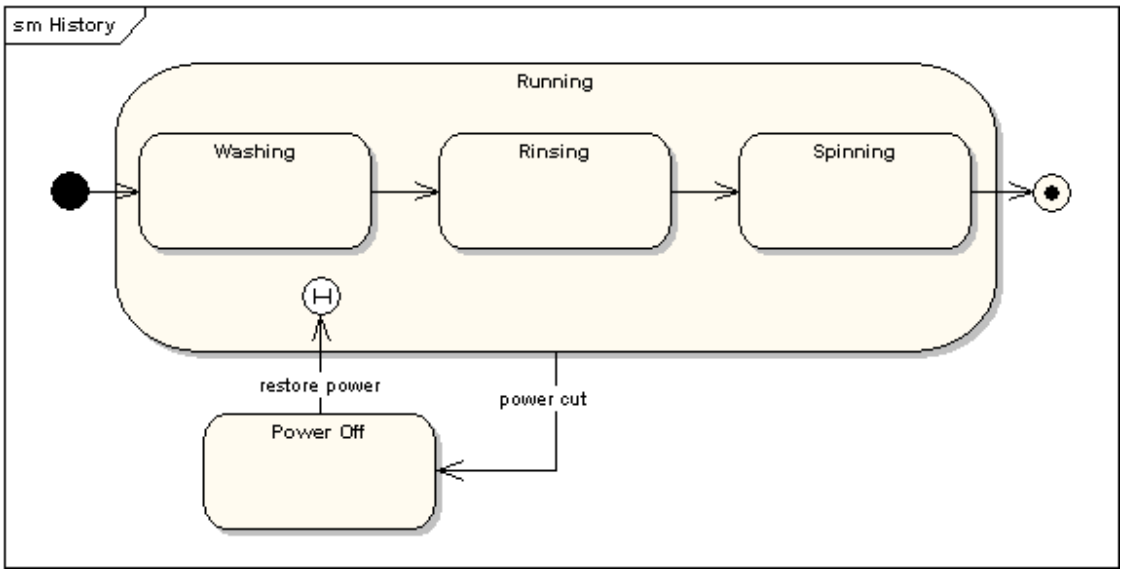
Terminate Pseudo-State

Entering a terminate pseudo-state indicates that the lifeline of the state machine has ended. A terminate pseudo-state is notated as a cross.



History States

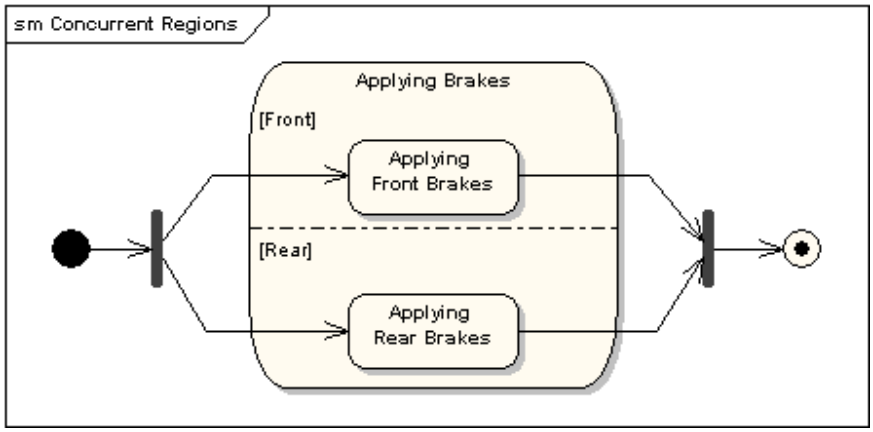
A history state is used to remember the previous state of a state machine when it was interrupted. The following diagram illustrates the use of history states. The example is a state machine belonging to a washing machine.



In this state machine, when a washing machine is running, it will progress from "Washing" through "Rinsing" to "Spinning". If there is a power cut, the washing machine will stop running and will go to the "Power Off" state. Then when the power is restored, the Running state is entered at the "History State" symbol meaning that it should resume where it last left-off.

Concurrent Regions

A state may be divided into regions containing sub-states that exist and execute concurrently. The example below shows that within the state "Applying Brakes", the front and rear brakes will be operating simultaneously and independently. Notice the use of fork and join pseudo-states, rather than choice and merge pseudo-states. These symbols are used to synchronize the concurrent threads.

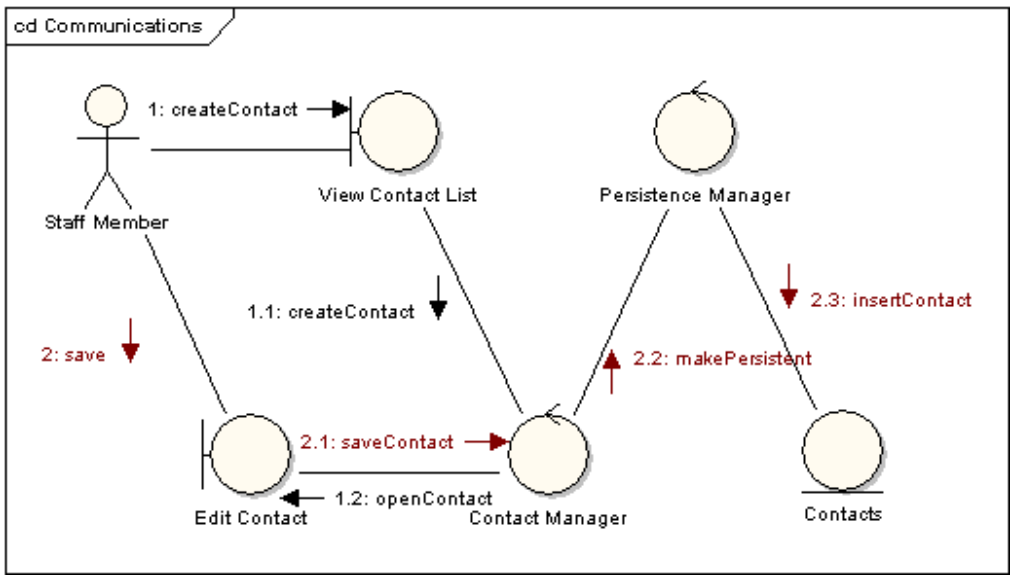


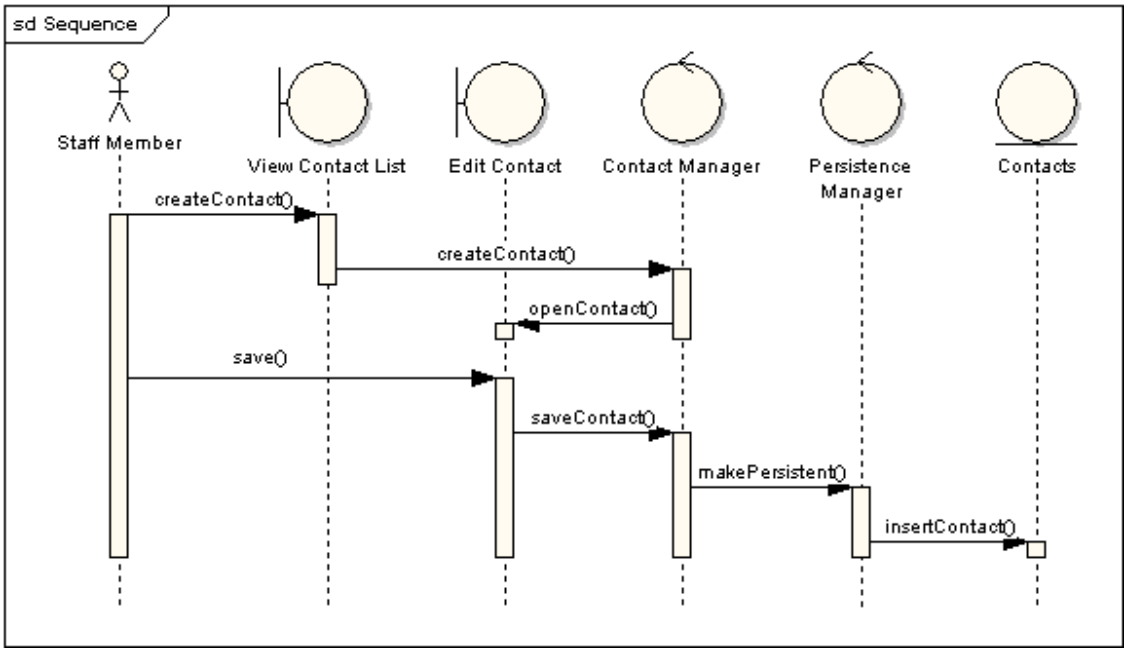
10. Communication Diagram

Communication Diagrams
A communication diagram, formerly called a collaboration diagram, is an interaction diagram that shows similar information to sequence diagrams but its primary focus is on object relationships.

On communication diagrams, objects are shown with association connectors between them. Messages are added to the associations and show as short arrows pointing in the direction of the message flow. The sequence of messages is shown through a numbering scheme.

The following two diagrams show a communication diagram and the sequence diagram that shows the same information. Although it is possible to derive the sequencing of messages in the communication diagram from the numbering scheme, it isn't immediately visible. What the communication diagram does show quite clearly though, is the full set of messages passed between adjacent objects.



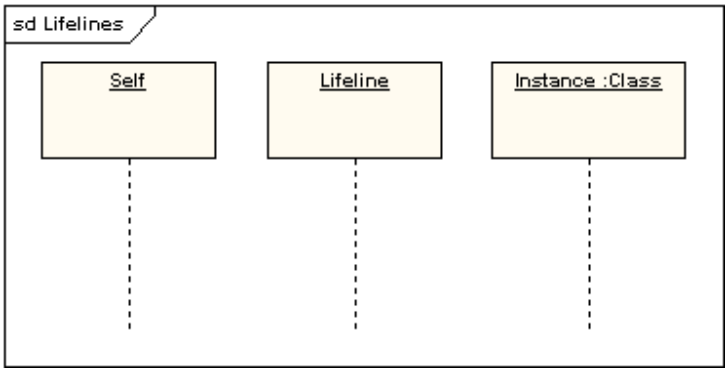


11. Sequence Diagram

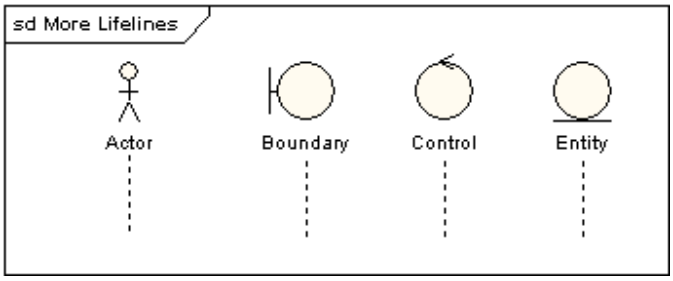
Sequence Diagrams

A sequence diagram is a form of interaction diagram which shows objects as lifelines running down the page, with their interactions over time represented as messages drawn as arrows from the source lifeline to the target lifeline. Sequence diagrams are good at showing which objects communicate with which other objects; and what messages trigger those communications. Sequence diagrams are not intended for showing complex procedural logic.

Lifelines
A lifeline represents an individual participant in a sequence diagram. A lifeline will usually have a rectangle containing its object name. If its name is "self", that indicates that the lifeline represents the classifier which owns the sequence diagram.

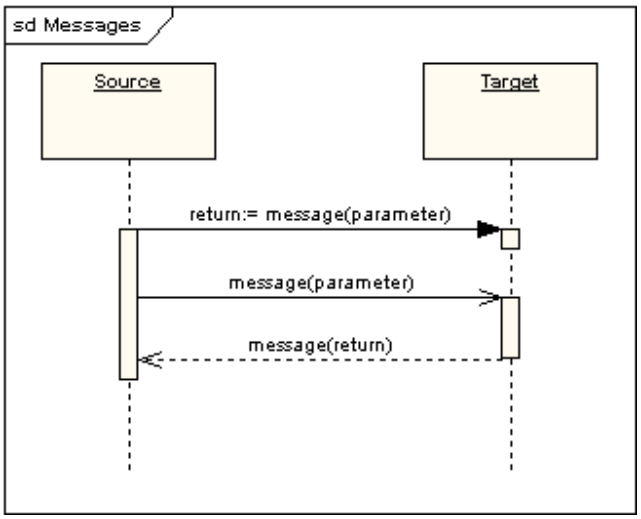


Sometimes a sequence diagram will have a lifeline with an actor element symbol at its head. This will usually be the case if the sequence diagram is owned by a use case. Boundary, control and entity elements from robustness diagrams can also own lifelines.



Messages

Messages are displayed as arrows. Messages can be complete, lost or found; synchronous or asynchronous; call or signal. In the following diagram, the first message is a synchronous message (denoted by the solid arrowhead) complete with an implicit return message; the second message is asynchronous (denoted by line arrowhead), and the third is the asynchronous return message (denoted by the dashed line).

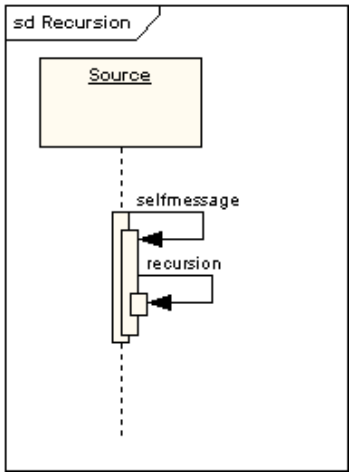


Execution Occurrence

A thin rectangle running down the lifeline denotes the execution occurrence, or activation of a focus of control. In the previous diagram, there are three execution occurrences. The first is the source object sending two messages and receiving two replies; the second is the target object receiving a synchronous message and returning a reply; and the third is the target object receiving an asynchronous message and returning a reply.

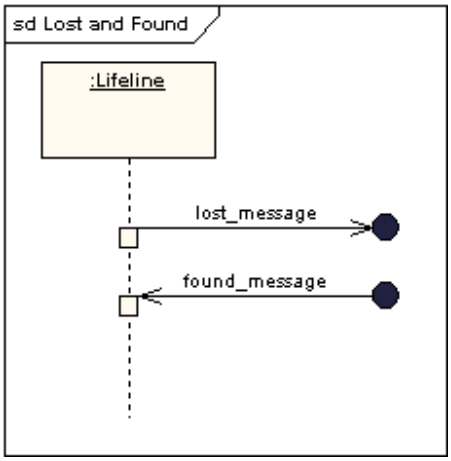
Self Message

A self message can represent a recursive call of an operation, or one method calling another method belonging to the same object. It is shown as creating a nested focus of control in the lifeline's execution occurrence.



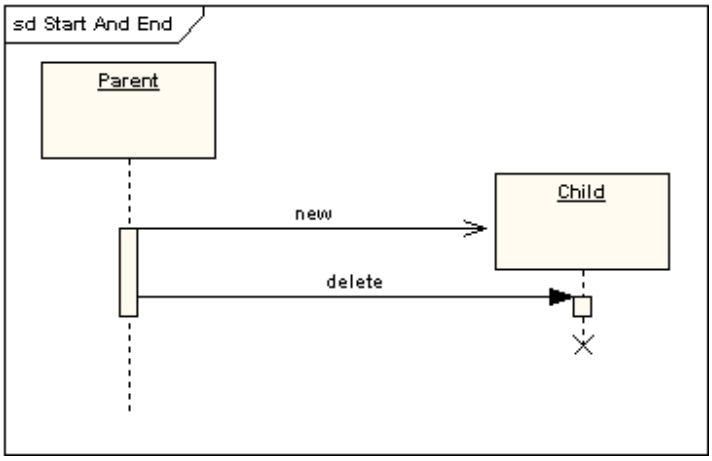
Lost and Found Messages

Lost messages are those that are either sent but do not arrive at the intended recipient, or which go to a recipient not shown on the current diagram. Found messages are those that arrive from an unknown sender, or from a sender not shown on the current diagram. They are denoted going to or coming from an endpoint element.



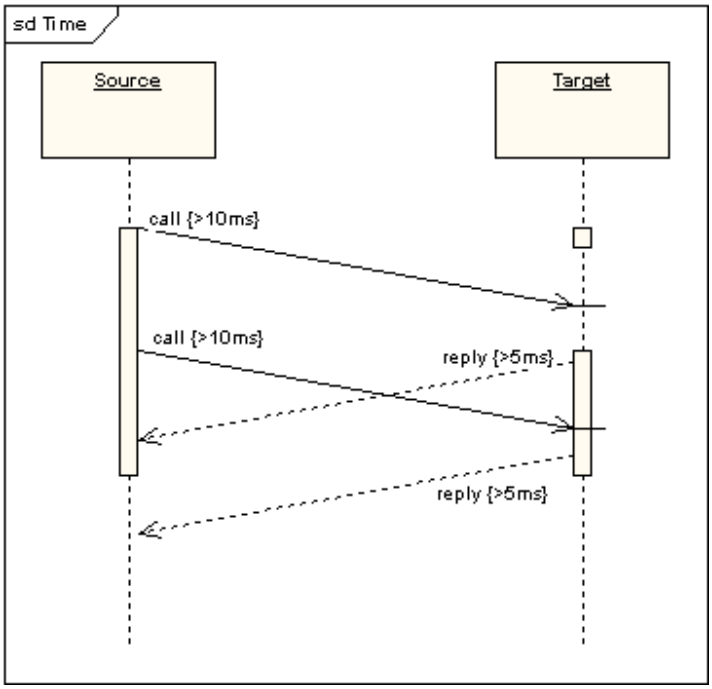
Lifeline Start and End

A lifeline may be created or destroyed during the timescale represented by a sequence diagram. In the latter case, the lifeline is terminated by a stop symbol, represented as a cross. In the former case, the symbol at the head of the lifeline is shown at a lower level down the page than the symbol of the object that caused the creation. The following diagram shows an object being created and destroyed.



Duration and Time Constraints

By default, a message is shown as a horizontal line. Since the lifeline represents the passage of time down the screen, when modelling a real-time system, or even a time-bound business process, it can be important to consider the length of time it takes to perform actions. By setting a duration constraint for a message, the message will be shown as a sloping line.



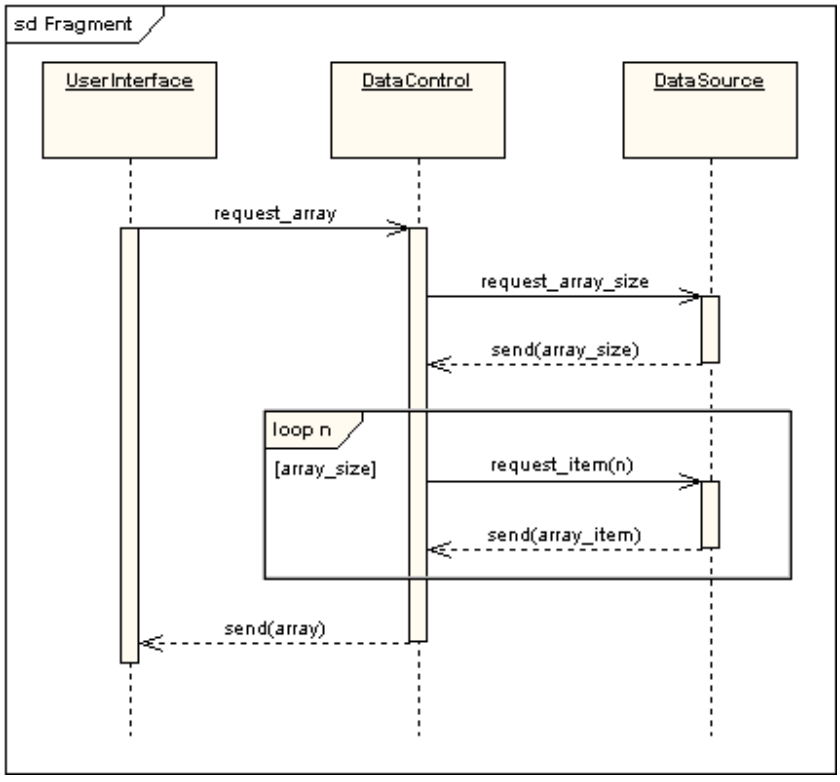
Combined Fragments

It was stated earlier that sequence diagrams are not intended for showing complex procedural logic. While this is the case, there are a number of mechanisms that do allow for adding a degree of procedural logic to diagrams and which come under the heading of combined fragments. A combined fragment is one or more processing sequence enclosed in a frame and executed under specific named circumstances. The fragments available are:

- Alternative fragment (denoted “alt”) models if…then…else constructs.
- Option fragment (denoted “opt”) models switch constructs.

- Break fragment models an alternative sequence of events that is processed instead of the whole of the rest of the diagram.
- Parallel fragment (denoted “par”) models concurrent processing.
- Weak sequencing fragment (denoted “seq”) encloses a number of sequences for which all the messages must be processed in a preceding segment before the following segment can start, but which does not impose any sequencing within a segment on messages that don’t share a lifeline.
- Strict sequencing fragment (denoted “strict”) encloses a series of messages which must be processed in the given order.
- Negative fragment (denoted “neg”) encloses an invalid series of messages.
- Critical fragment encloses a critical section.
- Ignore fragment declares a message or message to be of no interest if it appears in the current context.
- Consider fragment is in effect the opposite of the ignore fragment: any message not included in the consider fragment should be ignored.
- Assertion fragment (denoted “assert”) designates that any sequence not shown as an operand of the assertion is invalid.
- Loop fragment encloses a series of messages which are repeated.

The following diagram shows a loop fragment.

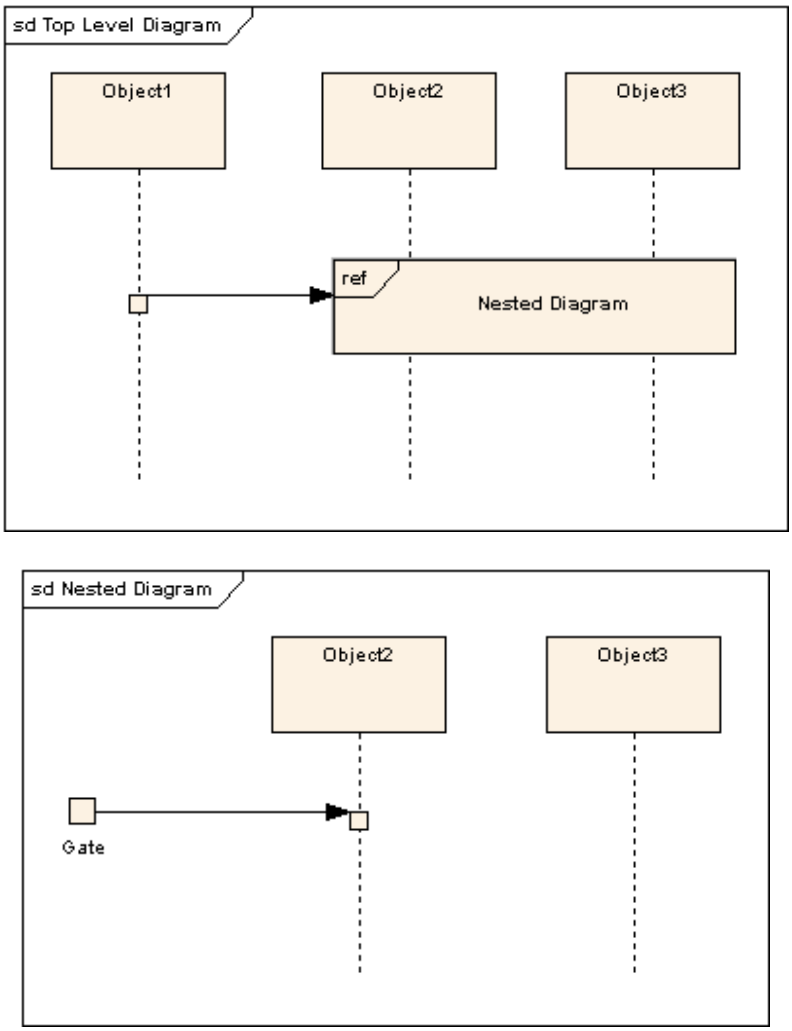


There is also an interaction occurrence, which is similar to a combined fragment. An interaction occurrence is a reference to another diagram which has the word "ref" in the top left corner of the frame, and has the name of the referenced diagram shown in the middle of the frame.

Gate

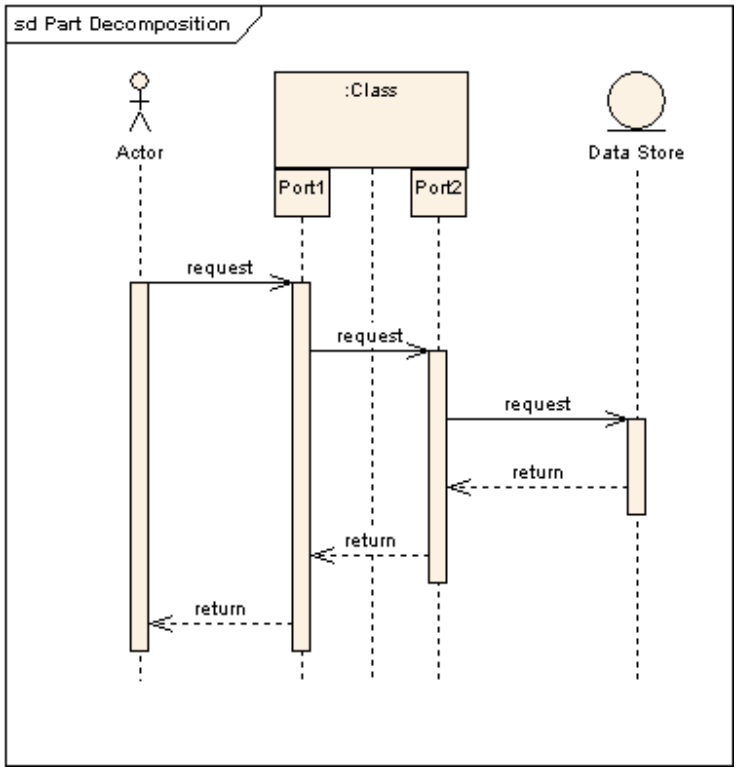
A gate is a connection point for connecting a message inside a fragment with a message outside a fragment. EA shows a gate as a small square on a fragment frame. Diagram gates act as off-page connectors for sequence diagrams, representing the source of incoming messages or the target of outgoing messages. The following two diagrams show how they might be used in

practice. Note that the gate on the top level diagram is the point at which the message arrowhead touches the reference fragment - there is no need to render it as a box shape.



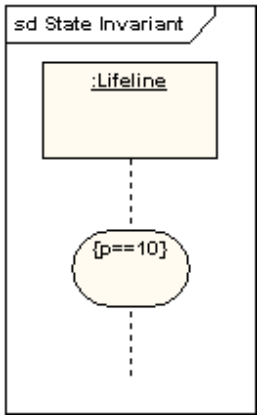
Part Decomposition

An object can have more than one lifeline coming from it. This allows for inter- and intra-object messages to be displayed on the same diagram.



State Invariant / Continuations

A state invariant is a constraint placed on a lifeline that must be true at run-time. It is shown as a rectangle with semi-circular ends.



A continuation has the same notation as a state invariant, but is used in combined fragments and can stretch across more than one lifeline.

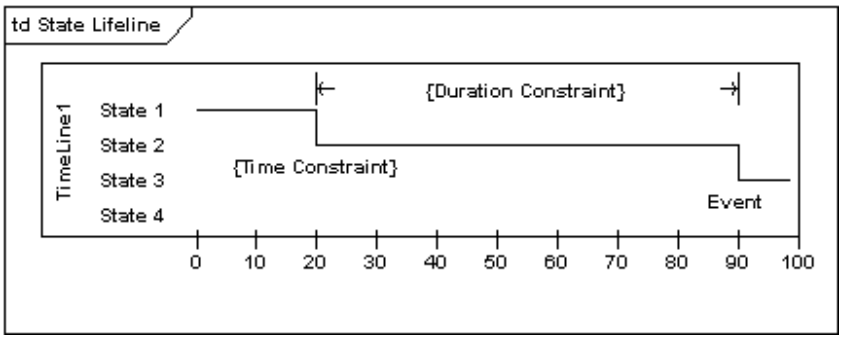
12. Timing Diagram

Timing Diagrams

UML timing diagrams are used to display the change in state or value of one or more elements over time. It can also show the interaction between timed events and the time and duration constraints that govern them.

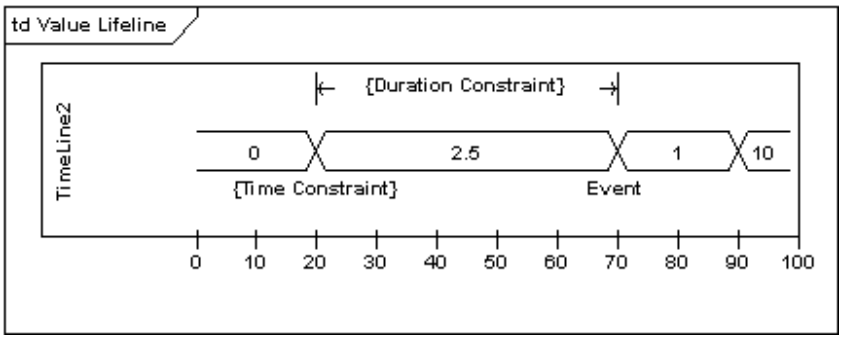
State Lifeline

A state lifeline shows the change of state of an item over time. The X-axis displays elapsed time in whatever units are chosen, while the Y-axis is labelled with a given list of states. A state lifeline is shown below.



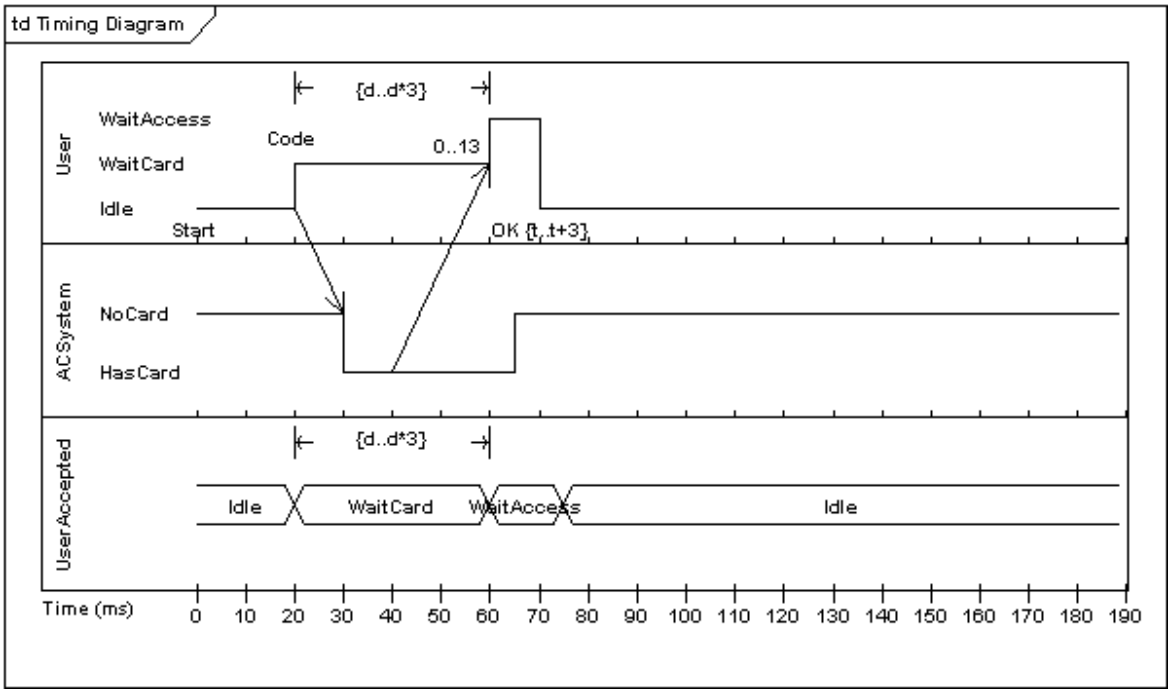
Value Lifeline

A value lifeline shows the change of value of an item over time. The X-axis displays elapsed time in whatever units are chosen, the same as for the state lifeline. The value is shown between the pair of horizontal lines which cross over at each change in value. A value lifeline is shown below.



Putting it all Together

State and value Lifelines can be stacked one on top of another in any combination. They must have the same X-axis. Messages can be passed from one lifeline to another. Each state or value transition can have a defined event, a time constraint which indicates when an event must occur, and a duration constraint which indicates how long a state or value must be in effect for. Once these have all been applied, a timing diagram may look like the following.



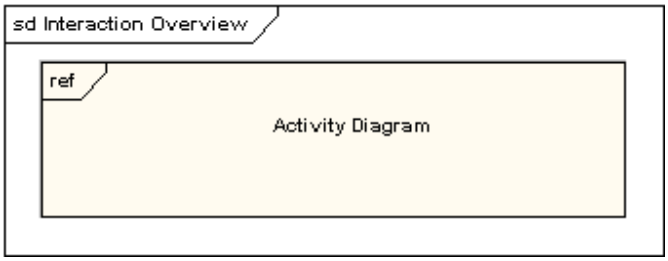
13. Interaction Overview Diagram

Interaction Overview Diagrams

An interaction overview diagram is a form of activity diagram in which the nodes represent interaction diagrams. Interaction diagrams can include sequence, communication, interaction overview and timing diagrams. Most of the notation for interaction overview diagrams is the same for activity diagrams. For example, initial, final, decision, merge, fork and join nodes are all the same. However, interaction overview diagrams introduce two new elements: interaction occurrences and interaction elements.

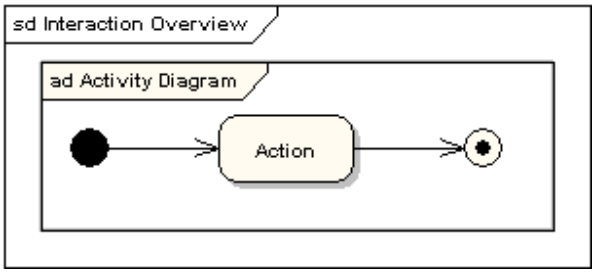
Interaction Occurrence

Interaction occurrences are references to existing interaction diagrams. An interaction occurrence is shown as a reference frame; that is, a frame with "ref" in the top-left corner. The name of the diagram being referenced is shown in the center of the frame.



Interaction Element

Interaction elements are similar to interaction occurrences, in that they display a representation of existing interaction diagrams within a rectangular frame. They differ in that they display the contents of the references diagram inline.



Putting it all Together

All the same controls from activity diagrams (fork, join, merge, etc.) can be used on interaction overview diagrams to put the control logic around the lower level diagrams. The following example depicts a sample sale process, with sub-processes abstracted within interaction occurrences.