TEEGALA KRISHNA REDDY ENGINEERING COLLEGE
(UGC-Autonomous)
Approved by AICTE, Affiliated by JNTUH, Accreditated by NAAC- 'A' Grade
Medbowli, Meerpet, Balapur, Hyderabad, Telangana- 500097
Mob: 8498085218. principal@tkrec.ac.in, ace@tkrec.ac.in

Software Engineering
(22AI301PC)

YEAR & SEM	**:**	**B. Tech II YEAR I SEM**
SUBJECT	**:**	**Software Engineering**
REGULATION	**:**	**R22**
DEPORTMENT & SEC	**:**	**AIML** - C
NAME OF THE FACULTY	**:**	**Mr. N NARESH**

UNIT- II

Software Requirements

Process of finding out, analyzing and documenting the requirements is called requirements engineering

Types of requirements

User requirements

- Statements in natural language plus diagrams of the services the system provides and its operational constraints. Written for customers.

System requirements

- A structured document setting out detailed descriptions of the system's functions, services and operational constraints. Defines what should be implemented so may be part of a contract between client and contractor.

Functional and Non-Functional Requirements

Functional requirements

- Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.

Non-functional requirements

- Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.

Domain requirements

Requirements that come from the application domain of the system and that reflect characteristics of that domain

Functional Requirements

- Describe functionality or system services.
- Depend on the type of software, expected users and the type of system where the software is used.

- Functional user requirements may be high-level statements of what the system should do but functional system requirements should describe the system services in detail.

The LIBSYS system

- A library system that provides a single interface to a number of databases of articles in different libraries.
- Users can search for, download and print these articles for personal study.

Examples of functional requirements

- The user shall be able to search either all of the initial set of databases or select a subset from it.
- The system shall provide appropriate viewers for the user to read documents in the document store.
- Every order shall be allocated a unique identifier (ORDER_ID) which the user shall be able to copy to the account's permanent storage area.

In principle, requirements should be both complete and consistent.

Complete

They should include descriptions of all facilities required.

Consistent

There should be no conflicts or contradictions in the descriptions of the system facilities.

In practice, it is impossible to produce a complete and consistent requirements document.

Non-Functional Requirements

These define system properties and constraints e.g. reliability, response time and storage requirements. Constraints are I/O device capability, system representations, etc.

Non-functional requirements may be more critical than functional requirements. If these are not met, the system is useless.

Non-functional classifications

1. ***Product requirements*** These requirements specify product behaviour.Examples include performance requirements on how fast the system must execute and how much memory it requires; reliability requirements that set out the acceptable failure rate; portability requirements; and usability requirements.

2. **Organisational requirements** these requirements are derived from policies and procedures in the customer s and developer s organization. Examples include process standards that must be used; implementation requirements such as the programming language or design method used; and delivery requirements that specify when the product and: Its documentation are to be delivered.

3. **External requirements** This broad heading covers all requirements that are derived from factors external to the system and its development process. These may include interoperability requirements that define how the system interacts with systems in other organizations: legislative requirements that must be followed to ensure that the system operates within the law; and ethical requirements. Ethical requirements are requirements placed on a system to ensure that it will be acceptable to its users and the general public.

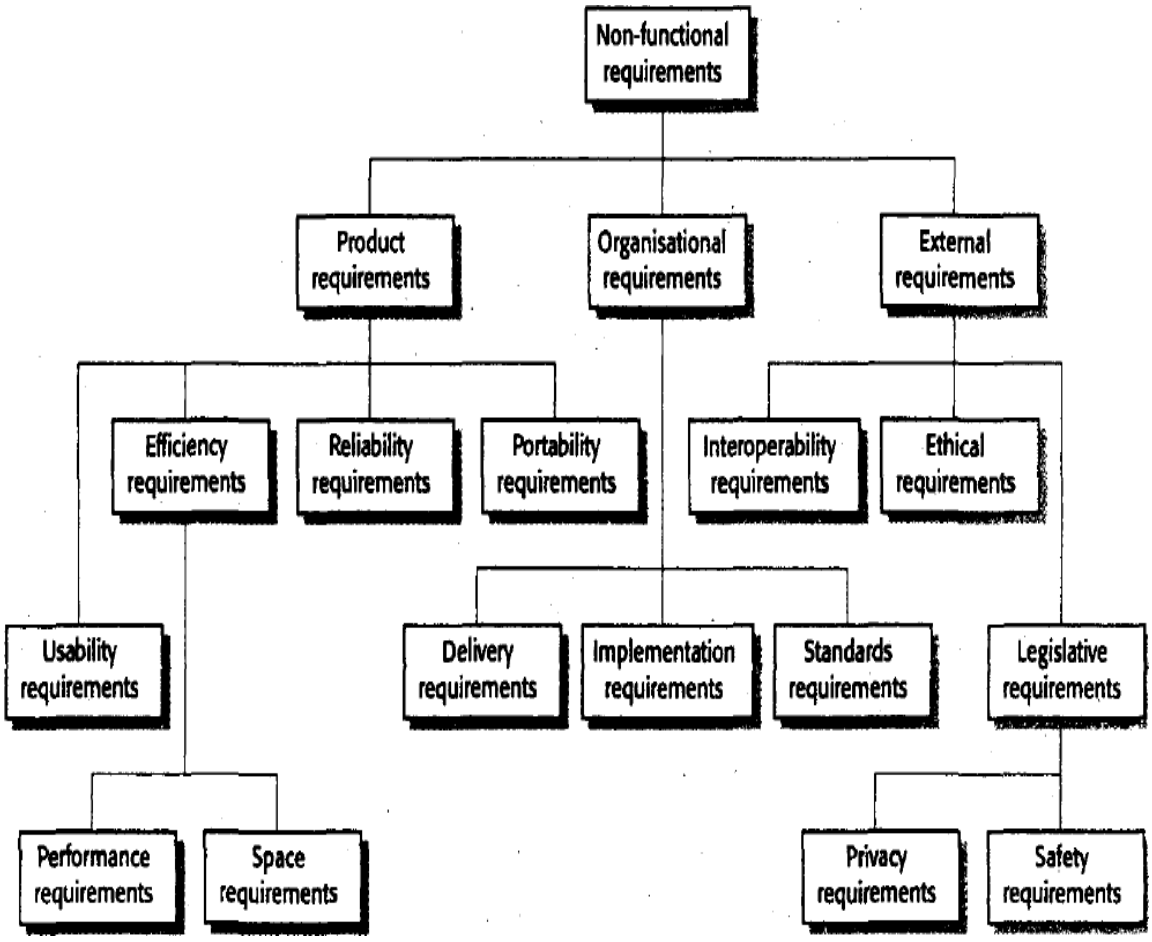


Fig. 2.1 Non – Functional Requirements

Non-functional requirements examples Product requirement

The user interface for LIBSYS shall be implemented as simple HTML without frames or Java applets**.**

Organisational requirement

The system development process and deliverable documents shall conform to the process and deliverables defined in XYZCo-SP-STAN-95.

External requirement

The system shall not disclose any personal information about customers apart from their name and reference number to the operators of the system.

A common problem with non-functional requirements is that they can be difficult to verify. Users or customers often state these requirements as general goals These vague goals cause problems for system developers as they leave scope for interpretation and subsequent dispute once the system is delivered. As an illustration of this problem, consider below This shows a system goal relating to the usability of a traffic control system and is typical of how a user might express usability requirements. I have rewritten it to show **how the goal can be expressed as a 'testable' non-functional requirement.**

Goal

- A general intention of the user such as ease of use.

Verifiable non-functional requirement

- A statement using some measure that can be objectively tested.

A system goal

- The system should be easy to use by experienced controllers and should be organised in such a way that user errors are minimised.

A verifiable non-functional requirement

- Experienced controllers shall be able to use all the system functions after a total of two hours training. After this training, the average number of errors made by experienced users shall not exceed two per day.

Whenever possible we should write non-functional requirements quantitatively so that they can be objectively tested.

You can measure these characteristics when the system is being tested to check whether or not the system has met its non-functional requirements.

Property	Measure
Speed	Processed transactions/second User/Event response time Screen refresh time
Size	K bytes Number of RAM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target-dependent statements Number of target systems

Fig. 2.2 Metrics for specifying non – functional requiements

In practice, however, customers for a system may find it practically impossible to translate their goals into quantitative requirements. For some goals, such as maintainability, there are no metrics that can be used.

Domain Requirements

Domain requirements are derived from the application domain of the system rather than from the specific needs of system users.
Domain requirements are important because they often reflect fundamentals of the application domain. If these requirements are not satisfied, it may be impossible to make the system work satisfactorily.

Library system domain requirements

1) There shall be a standard user interface to all databases which shall be based on the Z39.50 standard.

2) Because of copyright restrictions, some documents must be deleted immediately on arrival. Depending on the user's requirements, these documents will either be printed locally on the system server for manually forwarding to the user or routed to a network printer.

The first requirement is a design constraint. It specifies that the user interface to the database must be implemented according to a specific library standard. The developers therefore have to find out about that standard before starting the interface design. The second requirement has been introduced because of copyright laws that apply to material used in libraries. It specifies that the system must include an automatic delete-on-print facility for some classes of document. This means that users of the library system cannot have their own electronic copy of the document.

The deceleration of the train shall be computed as:

- $D_{train} = D_{control} + D_{gradient}$
  where $D_{gradient}$ is $9.81ms^2$ * compensated gradient/alpha and where the values of $9.81ms^2$ /alpha are known for different types of train.

This system automatically stops a train if it goes through a red signal. This requirement states how the train deceleration is computed by the system.It uses domain-specific terminology. To understand it, you need some understanding of the operation of railway systems and train characteristics.

Domain requirements problems

Requirements are expressed in the language of the application domain;
This is often not understood by software engineers developing the system.
Domain experts may leave information out of a requirement simply because it is so obvious to them. However, it may not be obvious to the developers of the system,and they may therefore implement the requirement in the wrong way.

User Requirements

Should describe functional and non-functional requirements in such a way that they are understandable by system users who don't have detailed technical knowledge.
User requirements are defined using natural language, tables and diagrams as these can be understood by all users.

However various **problems can arise when requirements are written in natural language** sentences in a text document:

Lack of clarity

It is sometimes difficult to use language in a precise and unambiguous way without making the document wordy and difficult to read.

Requirements confusion
Functional and non-functional requirements tend to be mixed-up.
Requirements amalgamation
Several different requirements may be expressed together as a single requirement.

EXAMPLE
Discover ambiguities and omissions for part of a ticket-issuing system

- An automated ticket-issuing system sells rail tickets. Users select their destination and input a credit card and a personal identification number. The rail ticket is issued and their credit card account charged. When the user presses the start button, a menu display of potential destinations is activated, along with a message to the user to select a destination. Once a destination has been selected, users are requested to input a personal identification number When the credit transaction has been validated, the ticket is issued.

Sol:
Can a customer buy several tickets for the same destination together or must they be bought one at a time?

b) Can customers cancel a request if a mistake has been made?

c) How should the system respond if an invalid card is input?

d) What happens if customers try to put their card in before selecting a destination (as they would in ATM machines)?

e) Must the user press the start button again if they wish to buy another ticket to a different destination?

To minimise misunderstandings when writing user requirements, **following guidelines are recommended.**

1) Invent a standard format and use it for all requirements.
2) Use language in a consistent way. Use shall for mandatory requirements, should for desirable requirements.
3) Use text highlighting to identify key parts of the requirement.
4) Avoid the use of computer jargon.

EXAMPLE

1. **Using the technique suggested here, where natural language is presented in a standard way, write plausible user requirements for the following function**

   The cash dispensing function in a bank ATM.

SOL: The user shall enter their bank card in the slot provided. Following the appropriate prompts for a cash withdrawal, the user shall enter the requested amount. If the amount requested is not greater than the amount in the account, cash shall be dispensed, and the card shall be returned.

System Requirements

System requirements are expanded versions of the user requirements that are used by software engineers as the starting point for the system design. They add detail and explain how the user requirements should be provided by the system

Ideally, the system requirements should simply describe the external behavior of the system and its operational constraints. They should not be concerned with how the system should be designed or implemented

Natural language is often used to write system requirements specifications as well as user requirements. However, because system requirements are more detailed than user requirements, **natural language specifications can be confusing and hard to understand:**

1) Natural language understanding relies on the specification readers and writers using the same words for the same concept. This leads to misunderstandings because of the **ambiguity** of natural language ex shoes must be worn and dogs must be carried.

2) A natural language requirements specification is **over flexible.** You can say the same thing in completely different ways. It is up to the reader to find out when requirements are the same and when they are distinct

3) There is **no easy way to modularize** natural language requirements. It may be difficult to find all related requirements

Because of these problems, requirements specifications written in natural language are prone to misunderstandings. These are often not discovered until later phases of the software process and may then be very expensive to resolve.

Notation	Description
Structured natural language	This approach depends on defining standard forms or templates to express the requirements specification.
Design description languages	This approach uses a language like a programming language but with more abstract features to specify the requirements by defining an operational model of the system. This approach is not now widely used although it can be useful for interface specifications.
Graphical notations	A graphical language, supplemented by text annotations is used to define the functional requirements for the system. An early example of such a graphical language was SADT (Ross, 1977) (Schoman and Ross, 1977). Now, use-case descriptions (Jacobsen, et al., 1993) and sequence diagrams are commonly used (Stevens and Pooley, 1999).
Mathematical specifications	These are notations based on mathematical concepts such as finite-state machines or sets. These unambiguous specifications reduce the arguments between customer and contractor about system functionality. However, most customers don't understand formal specifications and are reluctant to accept it as a system contract.

Fig. 2.3 Notations for Requirements Specification

Structured Language Specifications

- The freedom of the requirements writer is limited by a predefined template for requirements.
- All requirements are written in a standard way.
- The terminology used in the description may be limited.

The advantage is that the most of the expressiveness of natural language is maintained but a degree of uniformity is imposed on the specification

When a standard fonn is used for specifying functional requirements, the following information should be included:

- Definition of the function or entity.
- Description of inputs and where they come from.
- Description of outputs and where they go to.
- Indication of other entities required(requires part)
- Description of the action to be taken.
- Pre and post conditions (if appropriate).
- The side effects (if any) of the function.

Function	Compute insulin dose: Safe sugar level
Description	Computes the dose of insulin to be delivered when the current measured sugar level is in the safe zone between 3 and 7 units
Inputs	Current sugar reading (r2), the previous two readings (r0 and r1)
Source	Current sugar reading from sensor. Other readings from memory.
Outputs	CompDose—the dose in insulin to be delivered
Destination	Main control loop
Action:	CompDose is zero if the sugar level is stable or falling or if the level is increasing but the rate of increase is decreasing. If the level is increasing and the rate of increase is increasing, then CompDose is computed by dividing the difference between the current sugar level and the previous level by 4 and rounding the result. If the result, is rounded to zero then CompDose is set to the minimum dose that can be delivered.
Requires	Two previous readings so that the rate of change of sugar level can be computed.
Pre-condition	The insulin reservoir contains at least the maximum allowed single dose of insulin.
Post-condition	r0 is replaced by r1 then r1 is replaced by r2
Side effects	None

Using formatted specifications removes some of the problems of natural language specification. Variability in the specification is reduced and requirements are organised more effectively. However, it is difficult to write requirements in an unambiguous ay, particularly when complex computations are required.

To address this problem, you can add extra information to natural language requirements using tables or graphical models of the system.

Tables are particularly useful when there are a number of possible alternative situations and you need to describe the actions to be taken for each of these.

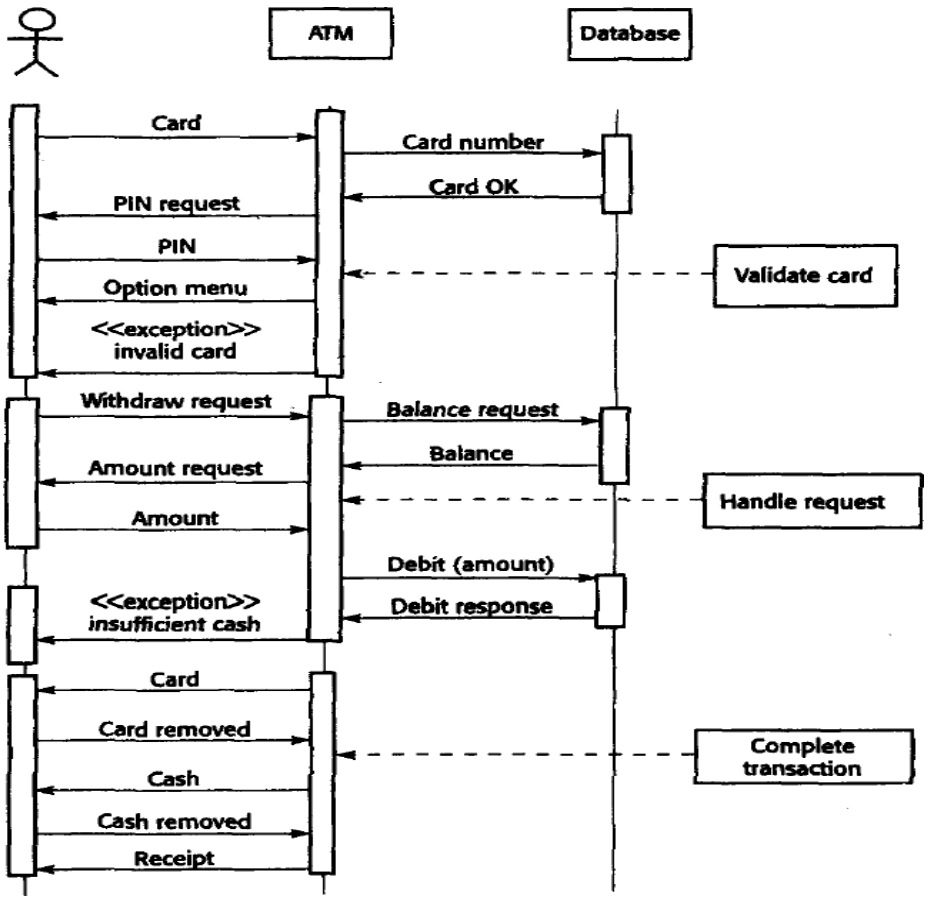
Condition	Action
Sugar level falling (r2 < r1)	CompDose = 0
Sugar level stable (r2 = r1)	CompDose = 0
Sugar level increasing and rate of increase decreasing ((r2 – r1) < (r1 – r0))	CompDose = 0
Sugar level increasing and rate of increase stable or increasing. ((r2 – r1) > (r1 – r0))	CompDose = round ((r2 – r1)/4) If rounded result = 0 then CompDose = MinimumDose

Fig. 2.4 Tabular Specification of computation

Graphical models are most useful when you need to show how state changes or where you need to describe a sequence of actions.

Sequence Diagrams

- These show the sequence of events that take place during some user interaction with a system.
- You read them from top to bottom to see the order of the actions that take place.
- Cash withdrawal from an ATM
    - Validate card;



- Handle request;
- Complete transaction.

Fig. 2.5Sequence Diagram for ATM

Interface Specifications

Almost all software systems must operate with existing systems that have already been implemented and installed in an environment. If the new system and the existing systems must work together, the: interfaces of existing systems have to be precisely specified.

There are three types of interface: that may have to be defined:

1. **Procedural interfaces** where existing programs or sub-systems offer a range of services that are accessed by calling interface procedures. These interfaces are sometimes called Application Programming Interfaces (APls).

2. **Data structures** that are passed from one sub-system to another. Graphical data models (described in Chapter 8) are the best notations for this type of description.

3. **Representations of data** (such as the ordering of bits) that have been established for an existing sub-system. the best way to describe these is probably to use a diagram of the structure with annotations explaining the function of each group of bits.

Figure 2.6 is an example of a procedural interface definition defined in Java.In this case, the interface is the procedural interface offered by a print server. This manages a queue of requests to print files on different printers. Users may examine the queue associated with a printer and may remove their print jobs from that queue. They may also switch jobs from one printer to another.

```
interface PrintServer {

// defines an abstract printer server
// requires: interface Printer, interface PrintDoc
// provides: initialize, print, displayPrintQueue, cancelPrintJob, switchPrinter

   void initialize ( Printer p ) ;
   void print ( Printer p, PrintDoc d ) ;
   void displayPrintQueue ( Printer p ) ;
   void cancelPrintJob (Printer p, PrintDoc d) ;
   void switchPrinter (Printer p1, Printer p2, PrintDoc d) ;
} //Pri  tServer
```

Fig. 2.6 The Java PDL description of a print server interface

The Software Requirements Document

The software requirements document (sometimes called the software requirements specification or SRS) is the official statement of what the system developers should implement. It should include both the user requirements for a system and a detailed specification of the system requirements.

In some cases, the user and system requirements may be integrated into a single description. In other cases, the user requirements are defined in an introduction to the system requirements specification. If there are a large number of requirements, the detailed system requirements may be presented in a separate document

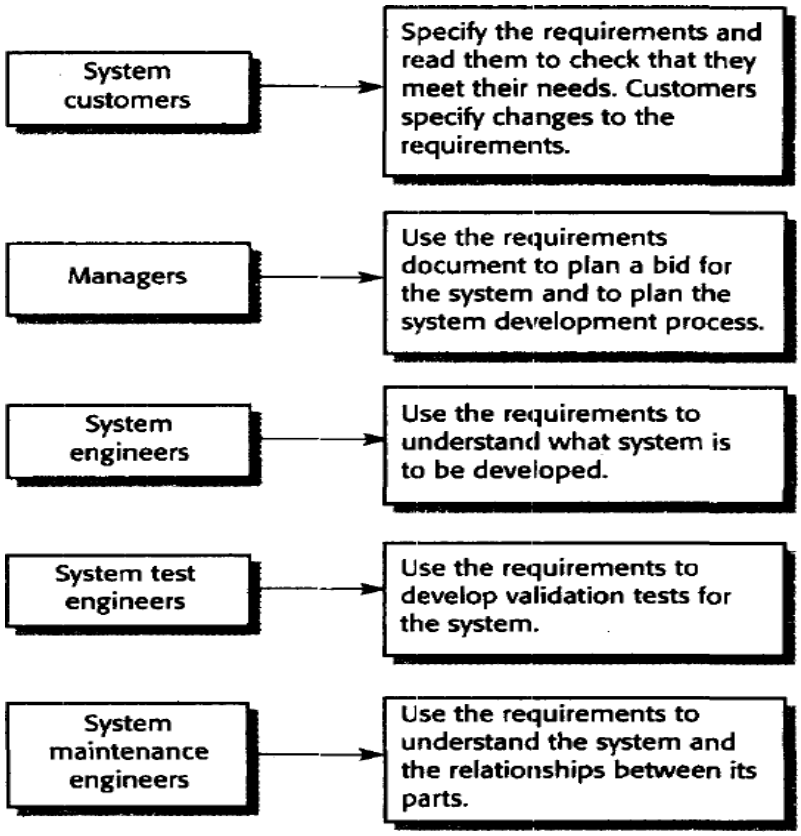


Fig. 2.7 Users of a requirements document

A number of large organisations, such as the US Department of Defense and the IEEE, have defined standards for requirements documents.

IEEE standard suggests the following structure for requirememts documents:

I. Introduction

1.1 Purpose of the requiremt:nts document
1.2 Scope of the product
1.3 Definitions, acronyms and abbreviations
1.4 References
1.5 Overview of the remainder of the document

2. General description

2.1 Product perspective
2.2 Product functions

2.3 User characteristics
2.4 General constraints
2.5 Assumptions and dependencies

3. **Specific requirements** cover functional, non··functional and interface requirements. The requirements may document external interfaces, describe system functionality and performance, specify logical database requirements, design constraints, emergent system properties and quality characteristics.

4. Appendices

5. Index

Although the IEEE standard is not ideal, it contains a great deal of good advice on how to write requirements and how to avoid problems. It is too general to be an organizational standard in its own right.

Figure 2.7 illustrates a possible organization for a requirements document that is based on the IEEE standard. However, the below fig have extended this to include information about predicted system evolution. This was first proposed by Heninge

The information that is included in a requirements document must depend on the type of software being developed and the approach to development that is used. If an evolutionary approach is adopted for a software product (say), the requirements document will leave out many of detailed chapters suggested above. The focus will be on defining the user requirements and high-level, non-functional system requirements. In this case, the designers and programmers use their judgment to decide how to meet the outline user requirements for the system

For long documents, it is particularly important to include a comprehensive table of contents and document index so that readers can find the information that they need.

Chapter	Description
Preface	This should define the expected readership of the document and describe its version history, including a rationale for the creation of a new version and a summary of the changes made in each version.
Introduction	This should describe the need for the system. It should briefly describe its functions and explain how it will work with other systems. It should describe how the system fits into the overall business or strategic objectives of the organisation commissioning the software.
Glossary	This should define the technical terms used in the document. You should not make assumptions about the experience or expertise of the reader.
User requirements definition	The services provided for the user and the non-functional system requirements should be described in this section. This description may use natural language, diagrams or other notations that are understandable by customers. Product and process standards which must be followed should be specified.
System architecture	This chapter should present a high-level overview of the anticipated system architecture showing the distribution of functions across system modules. Architectural components that are reused should be highlighted.
System requirements specification	This should describe the functional and non-functional requirements in more detail. If necessary, further detail may also be added to the non-functional requirements, e.g. interfaces to other systems may be defined.
System models	This should set out one or more system models showing the relationships between the system components and the system and its environment. These might be object models, data-flow models and semantic data models.
System evolution	This should describe the fundamental assumptions on which the system is based and anticipated changes due to hardware evolution, changing user needs, etc.
Appendices	These should provide detailed, specific information which is related to the application which is being developed. Examples of appendices that may be included are hardware and database descriptions. Hardware requirements define the minimal and optimal configurations for the system. Database requirements define the logical organisation of the data used by the system and the relationships between data.
Index	Several indexes to the document may be included. As well as a normal alphabetic index, there may be an index of diagrams, an index of functions, etc.

Fig. 2.8 The Structure of the requirements document

Requirements Engineering Process

- The goal of the requirements engineering process is to create and maintain a system requirements document.
- The overall process includes four high-level requirements engineering sub-processes.
  1) These are concerned with assessing whether the system is useful to the business **(feasibility study);**
  2) discovering requirements **(elicitation and analysis);**
  3) converting these requirements into some standard form **(specification);**
  4) and checking that the requirements actually define the system that the customer wants **(validation).**

- Figure 2.9 illustrates the relationship between these activities. It also shows the documents produced at each stage of the requirements engineering process.

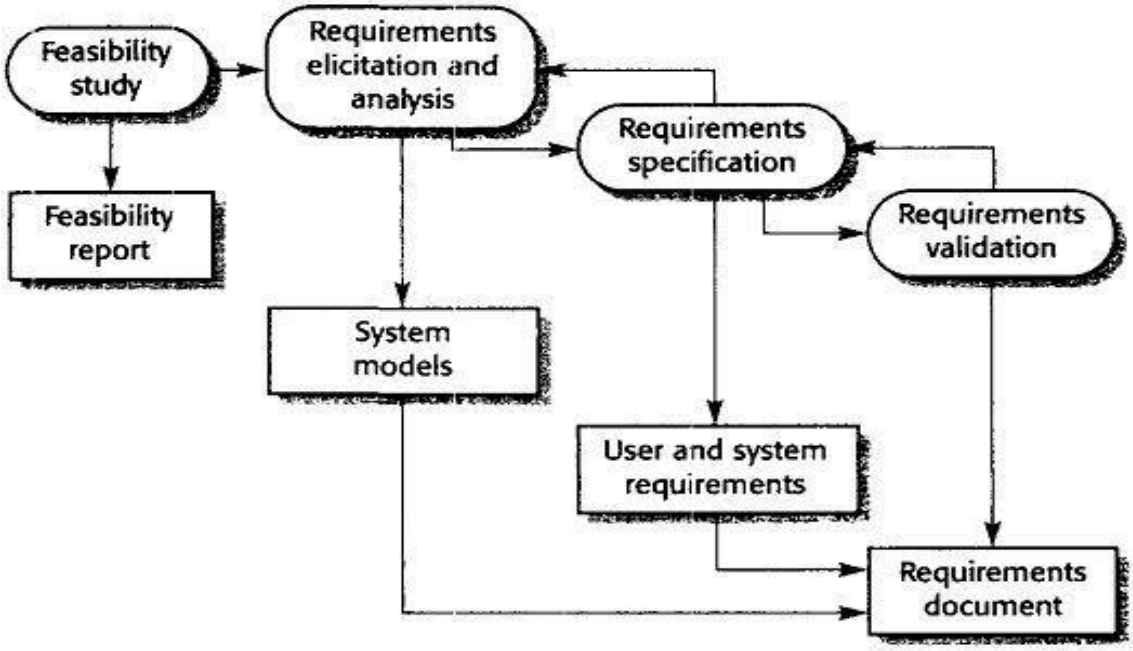


Fig. 2.9 The requirements engineering process

Feasibility Studies

- For all new systems, the requirements engineering process should start with a feasibility study. The input to the feasibility study is a set of preliminary business requirements, an outline description of the system and how the system is intended to support business processes.
- The results of the feasibility study should be a report that recommends whether or not it is worth carrying on with the requirements engineering and system development process
- **Technical feasibility:** It is carried out to determine whether the company has the capability in terms of software, hardware & personnel expertise to handle completion of project.
- **Economic feasibility:** I f benefits outweigh costs then decision is made to implement.
- **Operational feasibility:** How well a proposed system solves problems?
- In a feasibility study, you may consult information sources such as the managers of the departments where the system will be used, software engineers who are familiar with the type of system that is proposed, technology experts and end-users of the system. Normally, you should try to complete a feasibility study in two or three weeks.

Requirements Elicitation and Analysis

- The next stage of the requirements engineering process is requirements elicitation and analysis. In this activity, software engineers work with customers and system end-users to find out about the application domain, what services the system should provide, the required performance of the system, hardware constraints, and so on.
- Requirements elicitation and analysis may involve a variety of people in an organization. The term **stakeholder** is used to refer to any person or group who will be affected by the system, directly or indirectly. **Stakeholders** include end-users who interact with the system and everyone else in an organization that may be affected by its installation

Eliciting and understanding stakeholder requirements is difficult for several reasons:

- Stakeholders often don't know what they want from the computer system except in the most general terms.
- Stakeholders naturally express requirements in their own terms and with implicit knowledge of their own work. Requirements engineers, without experience in the customer's domain, must understand these requirements.
- Different stakeholders have different requirements, which they may express in different ways.
- Political factors may influence the requirements of the system. For example, managers may demand specific system requirements that will increase their influence in the organization.
- New requirements may emerge from new stakeholders who were not originally consulted.

System Stakeholders For A Bank ATM includes:

1) **Current bank customers** who receive services from the system
2) **Representatives from other banks** who have reciprocal agreements that allow each other's ATMs to be used
3) **Managers of bank branches** who obtain management information from the system
4) **Counter staff** at bank branches who are involved in the day-to-day running of the system
5) **Database administrators** who are responsible for integrating the system with the bank's customer database
6) **Bank security managers** who must ensure that the system will not pose a security hazard
7) **The bank's marketing department** who are likely be interested in using the system as a means of marketing the bank
8) **Hardware and software maintenance engineers** who are responsible for maintaining and upgrading the hardware and software.
9) **National banking regulators** who are responsible for ensuring that the system conforms to banking regulations

The process activities in requirements elicitation and analysis phase are:

1. **Requirements discovery:** This is the process of interacting with stakeholders in the system to collect their requirements. Domain requirements from stakeholders and documentation are also discovered during this activity.
2. **Requirements classification and organization**: This activity takes the unstructured collection of requirements, groups related requirements and organizes them into coherent clusters.
3. **Requirements prioritization and negotiation** Inevitably, where multiple stakeholders are involved, requirements will conflict. This activity is concerned with prioritizing requirements, and finding and resolving requirements conflicts through negotiation.
4. **Requirements documentation** the requirements are documented and input into the next round of the spiral.

Requirements Discovery

- Requirements discovery is the process of gathering information about the proposed and existing systems and distilling the user and system requirements from this information.
- Techniques of requirements discovery include viewpoints, interviewing, scenarios and ethnography.

Viewpoints

- These requirements sources (stakeholders, domain, and systems) can all be represented as system viewpoints, where each viewpoint presents a sub-set of the requirements for the system.
- **View Point Identification**
  - Providers of services to the system and receivers of system services
  - Systems that should interface directly with the system being specified
  - Regulations and standards that apply to the system
  - The sources of system business and non-functional requirements
  - Engineering viewpoints
  - Marketing and other viewpoints that generate requirements on the product features expected by customers and how the system should reflect the external image of the organization

- **Viewpoints can be used as a way of classifying stakeholders**

Interactor viewpoints represent people or other systems that interact directly with the system. In the bank ATM system, examples of interactor viewpoints are the bank's customers and the bank's account database.
Indirect viewpoints represent stakeholders who do not use the system themselves but who influence the requirements in some way. In the bank ATM system, examples of indirect viewpoints are the management of the bank and the bank security staff.
Domain viewpoints represent domain characteristics and constraints that influence the system requirements. In the bank ATM system, an example of a domain viewpoint would be the standards that have been developed for interbank communications.

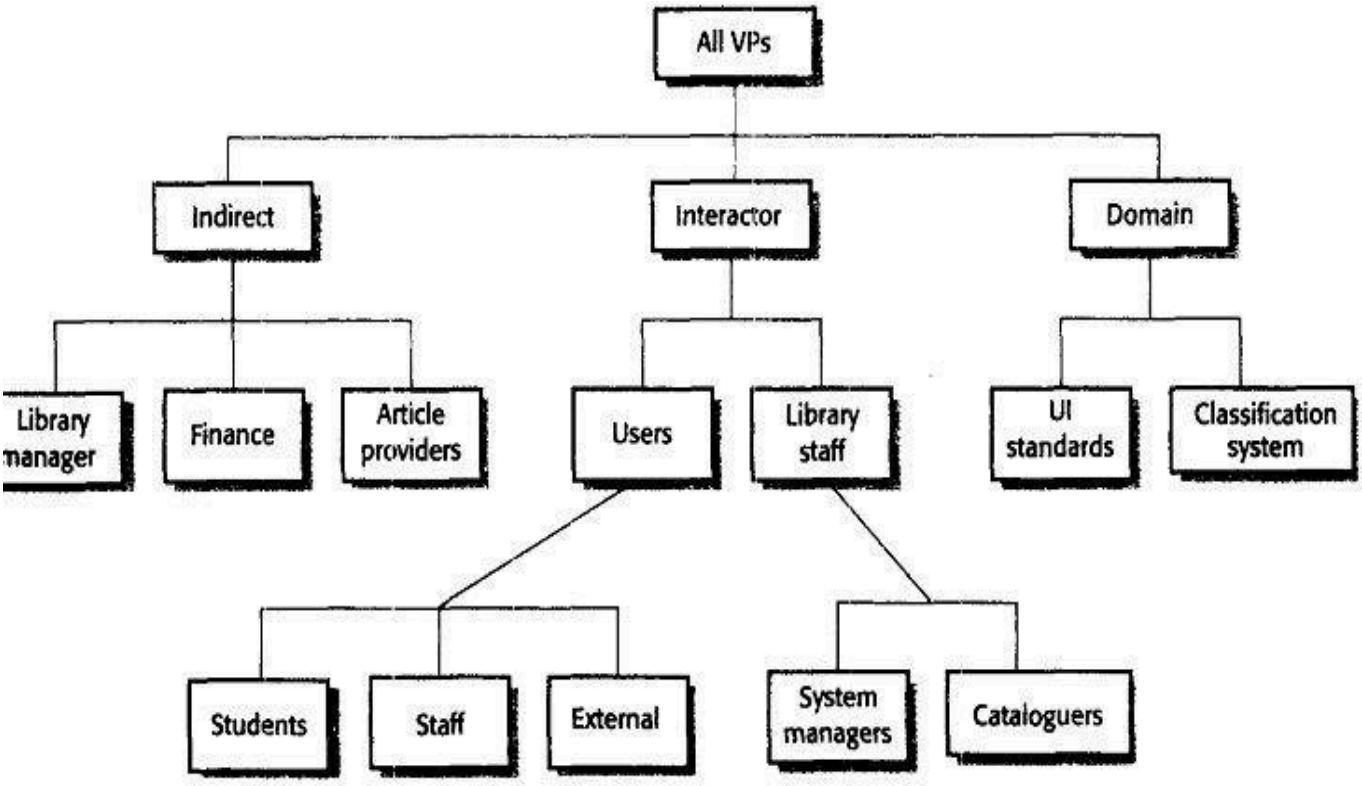


Fig. 2.10 Viewpoint of LIBSYS

Interviewing

Interviews may be of two types:

1. **Closed interviews** where the stakeholder answers a predefined set of questions.

2. **Open interviews** where there is no predefined agenda. The requirements engineering team explores a range of issues with system stakeholders and hence develops a better understanding of their needs.

However, interviews are not so good for understanding the requirements from the application domain.

It is hard to elicit domain knowledge during interviews for two reasons:

1. All application specialists use terminology and jargon that is specific to a domain. It is impossible for them to discuss domain requirements without using this terminology. They normally use terminology in a precise and subtle way that is easy for requirements engineers to misunderstand.

2. Some domain knowledge is so familiar to stakeholders that either they find it difficult to explain or they think it is so fundamental that it is not worth mentioning. For example, for a librarian, it is understood that all acquisitions are catalogued before they are added to the library.

However, this may not be obvious to the interviewer so it is not taken into account in the requirements.

Effective interviewers have two characteristics:

1) They are open-minded, avoid preconceived ideas about the requirements and are willing to listen to stakeholders. If the stakeholder comes up with surprising requirements, they are willing to change their mind about the system.
2) They prompt the interviewee to start discussions with a question.

Scenarios

- Scenarios can be particularly useful for adding detail to an outline requirements description. They are descriptions of example interaction sessions. Each scenario covers one or more possible interactions.
- **scenario may include:**

  I. A description of what the system and users expect when the scenario starts
  2. A description of the normal flow of events in the scenario
  3, A description of what can go wrong and how this is handled
  4. Information about other activities that might be going on at the same time
  5. A description of the system state when the scenario finishes.

- Scenarios may be written as text, supplemented by diagrams, screen shots, and so on.
- Simple text scenario, consider how a user of the LIBSYS library system may use the system. This scenario is shown in Figure 2.11.

Initial assumption: The user has logged on to the LIBSYS system and has located the journal containing the copy of the article.

Normal: The user selects the article to be copied. The system prompts the user to provide subscriber information for the journal or to indicate a method of payment for the article. Payment can be made by credit card or by quoting an organisational account number.

The user is then asked to fill in a copyright form that maintains details of the transaction and submit it to the LIBSYS system.

The copyright form is checked and, if it is approved, the PDF version of the article is downloaded to the LIBSYS working area on the user's computer and the user is informed that it is available. The user is asked to select a printer and a copy of the article is printed. If the article has been flagged as 'print-only' it is deleted from the user's system once the user has confirmed that printing is complete.

What can go wrong: The user may fail to fill in the copyright form correctly. In this case, the form should be re-presented to the user for correction. If the resubmitted form is still incorrect, then the user's request for the article is rejected.

The payment may be rejected by the system, in which case the user's request for the article is rejected.

The article download may fail, causing the system to retry until successful or the user terminates the session.

It may not be possible to print the article. If the article is not flagged as 'print-only' it is held in the LIBSYS workspace. Otherwise, the article is deleted and the user's account credited with the cost of the article.

Other activities: Simultaneous downloads of other articles.

System state on completion: User is logged on. The downloaded article has been deleted from LIBSYS workspace if it has been flagged as print-only.

Fig. 2.11 Scenario for article downloading in LIBSYS

Use-Cases

- Use-cases are a scenario-based technique for requirements elicitation which were first introduced in the Objectory method
- Use-case identifies the type of interaction and the actors involved.
- Use-cases identify the individual interactions with the system.
- Actors in the process are represented as stick figures, and each class of interaction is represented as a named ellipse. The set of use-cases represents all of the possible interactions to be represented in the system requirements.
- Figure 2.13 develops the LIBSYS example and shows other use-cases in that environment.

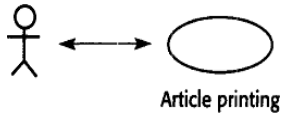


Fig. 2. 12 A simple use-case for article printing

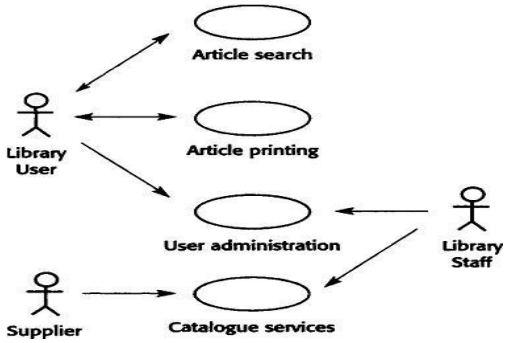


Fig. 2. 13 Use-Case for the library system

ETHNOGRAPHY

Ethnography is an observational technique that can be used to understand social and organizational requirements. An analyst immerses him or herself in the working environment where the system will be used. He or she observes the day-to-day work and notes made of the actual tasks in which participants are involved.

Ethnography is particularly effective at discovering two types of requirements:

1) Requirements that are derived from the way in which people actually work
2) Requirements that are derived from cooperation and awareness of other people's activities.

Requirements Validation

Concerned with demonstrating that the requirements define the system that the customer really wants.

- Requirements error costs are high so validation is very important
    - Fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error.

During the requirements validation process, checks should be carried out on the requirements in the requirements document. These checks include:

- **Validity.** Does the system provide the functions which best support the customer's needs?
- **Consistency**. Are there any requirements conflicts?
- **Completeness.** Are all functions required by the customer included?
- **Realism.** Can the requirements be implemented given available budget and technology
- **Verifiability.** Can the requirements be checked?

Requirements Validation Techniques

- **Requirements reviews** Systematic manual analysis of the requirements.
- **Prototyping** Using an executable model of the system to check requirements. Covered
- **Test-case generation** Developing tests for requirements to check testability.

Requirements Reviews

- Regular reviews should be held while the requirements definition is being formulated.
- Both client and contractor staff should be involved in reviews.
- Requirements reviews can be informal or formal.
- Informal reviews simply involve contractors discussing requirements with as many system stakeholders as possible.
- In a formal requirements review, the development team should 'walk' the client through the system requirements, explaining the implications of each requirement.
- Reviewers may also check for:
  - **Verifiability.** Is the requirement realistically testable?
  - **Comprehensibility**. Is the requirement properly understood?
  - **Traceability.** Is the origin of the requirement clearly stated?
  - **Adaptability.** Can the requirement be changed without a large impact on other requirements?

Requirements Management

- The requirements for large software systems are always changing.
- Requirements management is the process of understanding and controlling changes to system requirements. You need to keep track of individual requirements
- Requirements are inevitably incomplete and inconsistent
  - New requirements emerge during the process as business needs change and a better understanding of the system is developed;
  - Different viewpoints have different requirements and these are often contradictory.

Enduring and volatile requirements

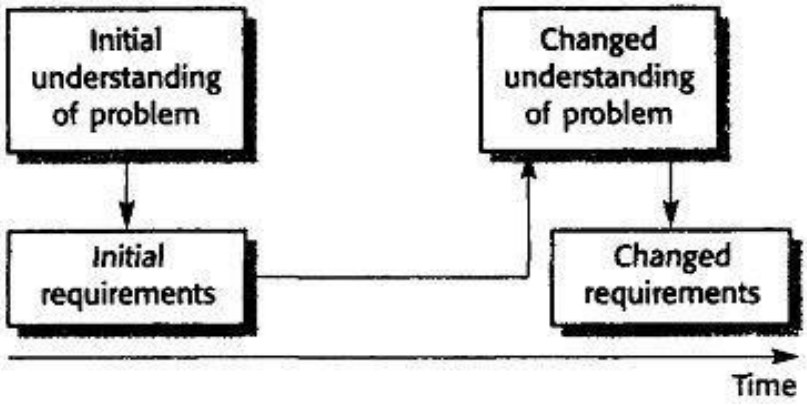


Fig. 2. 14 Requirements Evolution

- As the requirements definition is developed, you normally develop a better understanding of users needs.This feeds information back to the user, who may then propose a change to the requirements (Figure 2.14). Furthermore, it may take several years to specify and develop a large system. Over that time, the system's environment and the business objectives may change
- From an evolution perspective, requirements fall into two classes:
- **Enduring requirements**. Stable requirements derived from the core activity of the customer organisation. E.g. a hospital will always have doctors, nurses, etc. May be derived from domain models
- **Volatile requirements**. Requirements which change during development or when the system is in use.

Requirement Type	Description
Mutable requirements	Requirements which change because of changes to the environment in which the organisation is operating. For example, in hospital systems, the funding of patient care may change and thus require different treatment information to be collected.
Emergent requirements	Requirements which emerge as the customer's understanding of the system develops during the system development. The design process may reveal new emergent requirements.
Consequential requirements	Requirements which result from the introduction of the computer system. Introducing the computer system may change the organisation's processes and open up new ways of working which generate new system requirements.
Compatibility requirements	Requirements which depend on the particular systems or business processes within an organisation. As these change, the compatibility requirements on the commissioned or delivered system may also have to evolve.

Fig. 2. 15 Classification of volatile requirements

Requirements Management Planning

Planning is an essential first stage in the requirements management process. Requirements management is very expensive. During the requirements management stage, you have to decide on

Requirements identification: How requirements are uniquely identified;

A change management process: This is the set of activities that assess the impact and cost of changes.

Traceability policies these policies define the relationships between requirements, and between the requirements and the system design that should be recorded and how these records should be maintained

CASE tool support The tool support required to help manage requirements change**;**

Traceability

Traceability is concerned with the relationships between requirements, their sources and the system design

Source traceability Links from requirements to stakeholders who proposed these requirements;

Requirements traceability Links between dependent requirements; You use this infonnation to assess how many requirements are likely to be affected by a proposed change

Design traceability Links from the requirements to the design; You use this infonnation to assess the impact of proposed requirements changes on the system design and implementation

Traceability information is often represented using traceability matrices

Figure 2.16 shows a simple traceability matrix that records the dependencies between requirements. A 'D' in the row/column intersection illustrates that the requirement in the row depends on the requirement named in the column; an 'R' means that there is some other, weaker relationship between the requirements. For example, they may both define the requirements for parts of the same subsystem.

Traceability matrices can be generated automatically from the database

Req. id	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2
1.1		D	R					
1.2			D			R		D
1.3	R			R				
2.1			R		D			D
2.2								D
2.3		R		D				
3.1								R
3.2							R	

Fig. 2. 16 A Traceability matrix

Case Tool Support

Requirements management needs automated support; the CASE tools for this should be chosen during the planning phase. You need tool support for:

Requirements storage The requirements should be maintained in a secure, managed data store that is accessible to everyone involved in the requirements engineering process.

Change management The process of change management (Figure 7.13) is simplified if active tool support is available.

Traceability management As discussed above, tool support for traceability allows related requirements to be discovered.

Requirements Change Management

Requirements change management (Figure 2.17) should be applied to all proposed changes to the requirements.

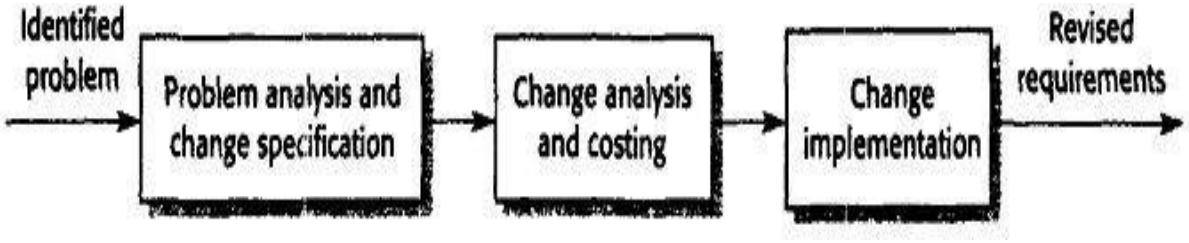


Fig. 2. 17 Requirements change management

Problem analysis and change specification The process starts with an identified requirements problem or, sometimes, with a specific change proposal. During this stage, the problem or the change proposal is analysed to check that it is valid.

Change analysis and costing The effect of the proposed change is assessed using traceability information and general knowledge of the system requirements.

Change implementation The requirements document and, where necessary, the system design and implementation are modified.

System Models

- These models are graphical representations that describe business processes, the problem to be solved and the system that is to be developed.
- System modelling helps the analyst to understand the functionality of the system and models are used to communicate with customers.
- Different models present the system from **different perspectives**

    - **External perspective** showing the system's context or environment;
    - **Behavioural perspective** showing the behaviour of the system;
    - **Structural perspective** showing the system or data architecture.
- A system model is an abstraction of the system being studied
- Examples of the types **of system models that you might create during the analysis process are:**

    1. **A data- flow model** Data-flow models show how data is processed at different Stages in the system.
    2. **A composition model** A composition or *aggregation* model shows how entities in the system are composed of other entities.
    3. **An architectural model** Architectural models show the principal sub-systems that make up a system.
    4. **A classification model** Object class/inheritance diagrams show how entities have common characteristics.
    5. **A stimulus-response model** A stimulus-response model, or *state transition diagram,* shows how the system reacts to internal and external events.

Context Models

- At an early stage in the requirements elicitation and analysis process we should decide 0n the boundaries of the system. This involves working with system stakeholders to distinguish what is the system and what is the system's environment.
- Social and organisational concerns may affect the decision on where to position system boundaries.
- Once some decisions on the boundaries of the system have been made, part of the analysis activity is the definition of that context and the dependencies that a system has on its environment. Normally, producing a simple architectural model is the first step in this activity
- **Architectural models** show the system and its relationship with other systems.
- High-level architectural models are usually expressed as simple block diagrams where each sub-system is represented by a named rectangle, and lines indicate associations between sub-systems
- From Figure 2.18, we see that each ATM is connected to an account database, a local branch accounting system, a security system and a system to support machine maintenance. The system is also connected to a usage database that monitors how the network of ATMs is used and to a local branch counter system. This counter system provides services such as backup and printing.
- Architectural models describe the environment of a system. However, they do not show the relationships between the other systems in the environment and the system that is

being specified. External systems might produce data for or consume data from the system.

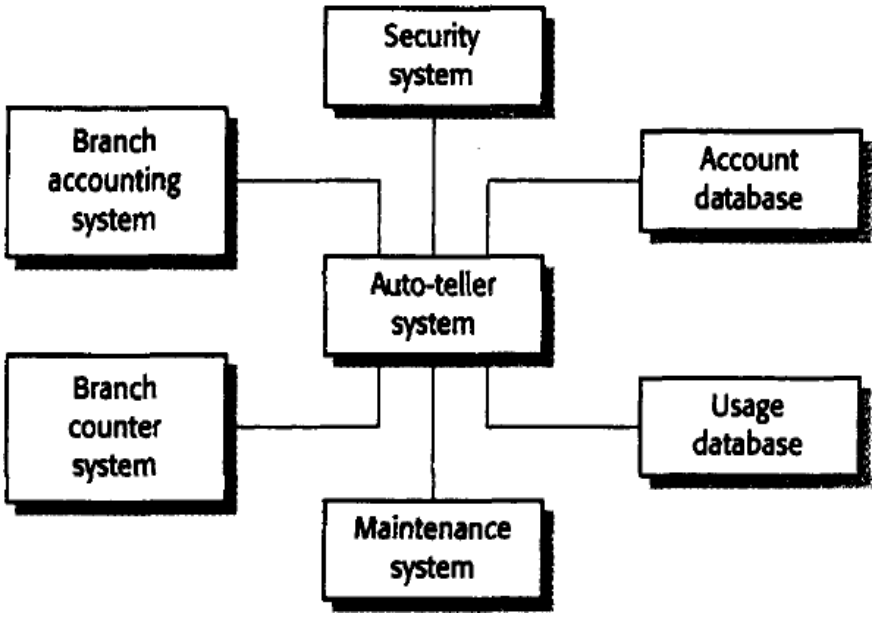


Fig: 2.18 Architectural model of an ATM

- Simple architectural models are normally supplemented by other models, such as process models, that show the process activities supported by the system.
- Data-flow models may also be used to show the data that is transferred between the system and other systems in its environment

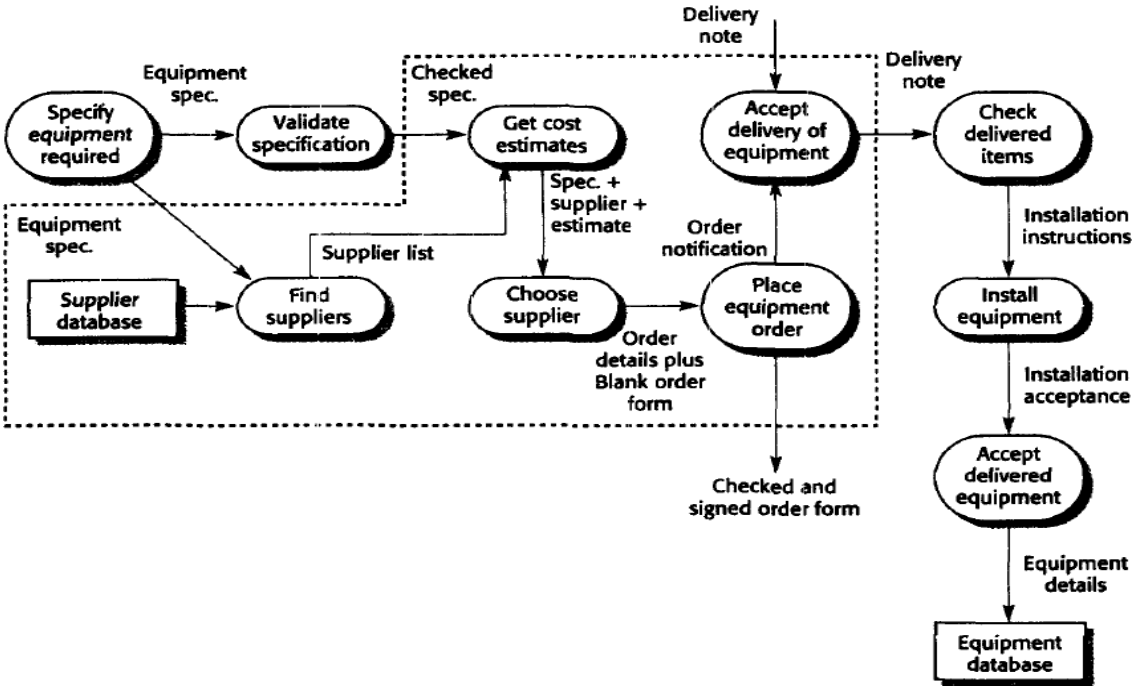


Fig: 2.19 Process Model of Equipment Procurement

Figure 2.19 illustrates a process model for the process of procuring equipment in an organization. This involves specifying the equipment required, finding and choosing suppliers, ordering the equipment, taking delivery of the equipment and testing it after delivery. When specifying computer support for this process, we have to decide which of these activities will actually be supported. The other activities are outside the boundary of the system. In Figure 2. 19, the dotted line encloses the activities that are within the system boundary.

Behavioural Models

Be**h**avioural models are used to describe the overall behaviour of a system.

Two types of behavioural model are:

- **Data processing models** that show how data is processed as it moves through the system
- **State machine models** that show the systems response to events

Data flow models

- Data flow diagrams (DFDs) may be used to model the system's data processing.
- These show the processing steps as data flows through a system.
- DFDs are an intrinsic part of many analysis methods.
- Simple and intuitive notation that customers can understand.
- The notation used in these models represents functional processing (rounded rectangles), data stores (rectangles) and data movementsbetween functions (labelled arrows).

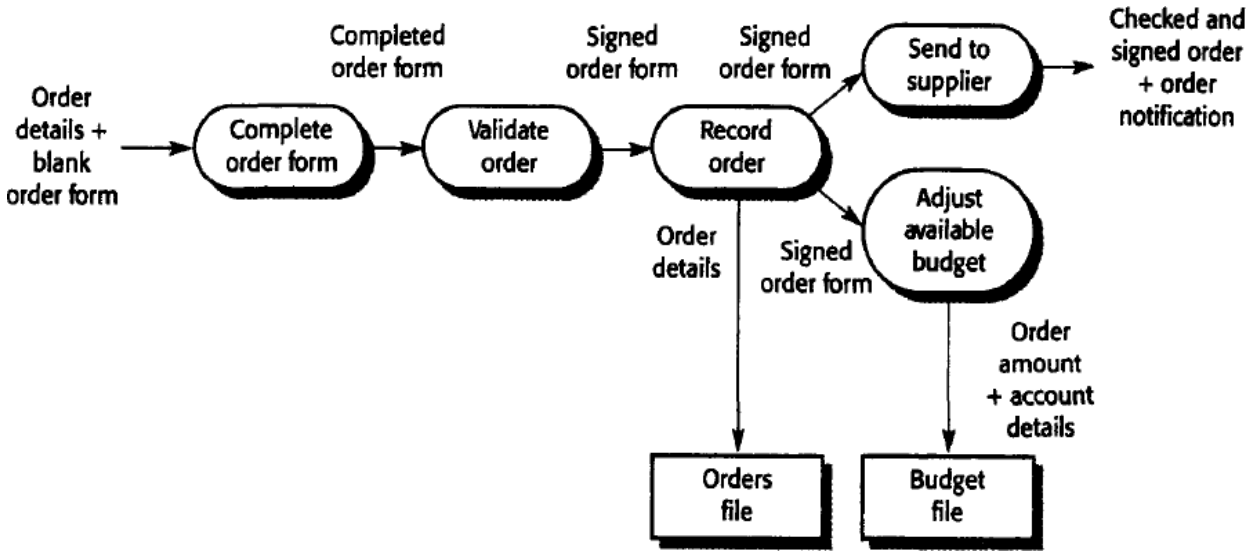


Fig. 2.20 Data flow diagram of order processing

A data-flow model, which shows the steps involved in processing an order for goods (such as computer equipment) in an organization, is illustrated in Figure 2.20.This particular model describes the data processing in the Place equipment order activity in the overall process model shown in Figure 2.19. The model shows how the order for the goods moves from process to process. It also shows the data stores (Orders file and Budget file) that are involved in this process.

- Data-flow models are valuable because tracking and documenting how the data associated with a particular process moves through the system helps analysts understand what is going on
- Data-flow models show a functional perspective where each transformation represents a single function or process. They are particularly useful during the analysis of requirements
- That is, they show the entire sequence of actions that take place from an input being processed to the corresponding output that is the system's response. Figure 8.4 illustrates this use of data flow diagrams.

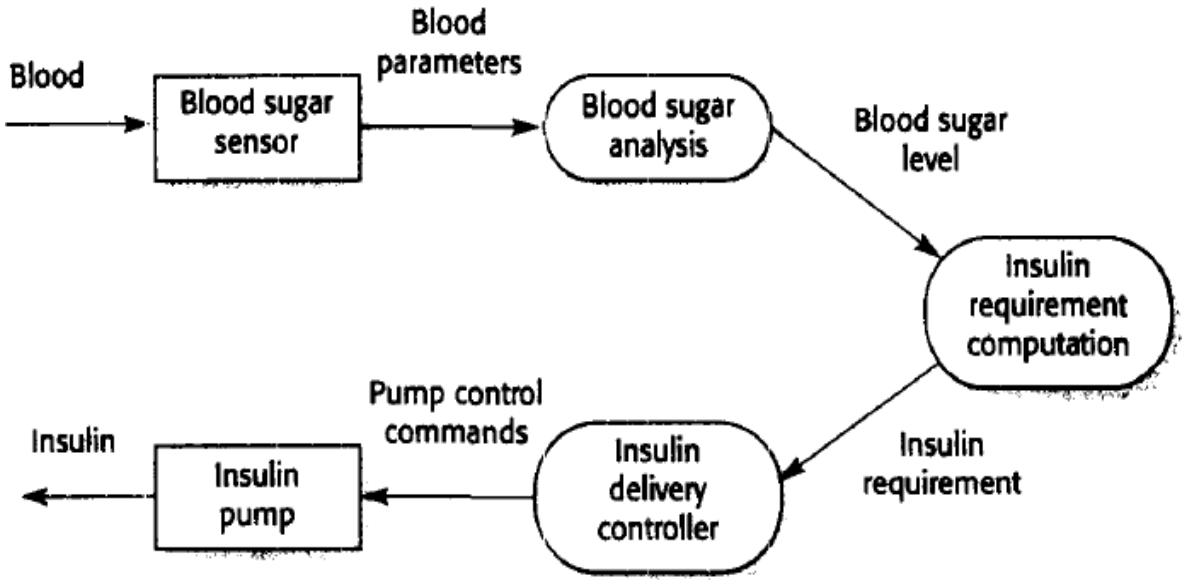


Fig. 2.21 DFD for insulin pump

State-Machine Models

These model the behaviour of the system in response to external and internal events.
They show the system's responses to stimuli so are often used for modelling real-time systems.
State machine models show system states as nodes and events as arcs between these odes. When an event occurs, the system moves from one state to another.

Statecharts are an integral part of the UML and are used to represent state machine models

State-charts allow the decomposition of a model into sub-models.A brief description of the actions is included following the 'do' in each state.

This approach to system modelling is illustrated in Figure 2.22. This diagram shows a state machine model of a simple microwave oven equipped with buttons to set the power and the timer and to start the system.

Sequence of actions in using the microwave is:

1. Select the power level (either half-power or full-power).

2. Input the cooking time.

3. Press Start, and the food is cooked for the given time.

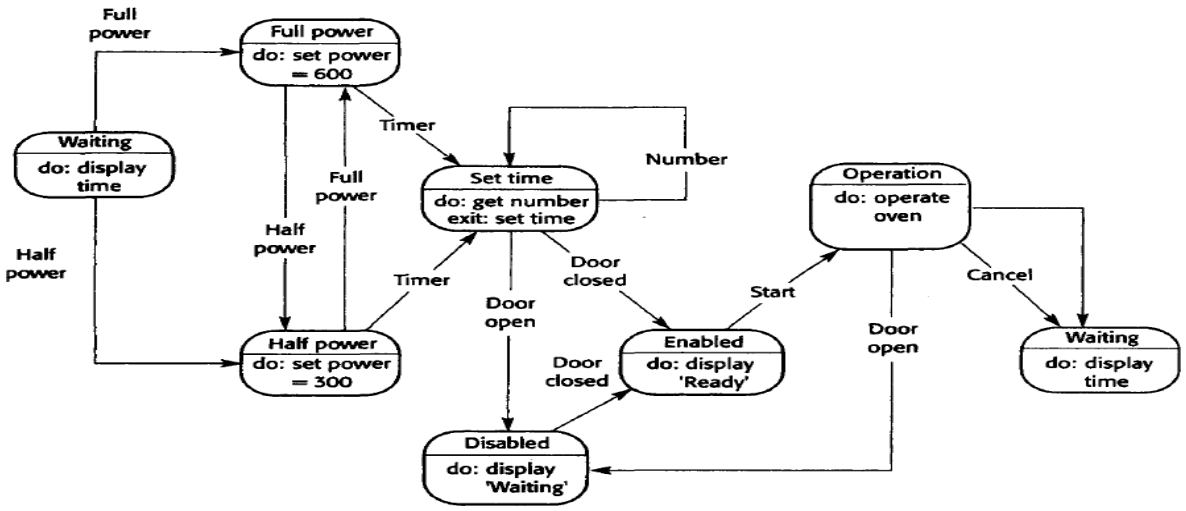


Fig. 2.22 State-Machine Model of a MicroWave Oven

For safety reasons, the oven should not operate when the door is open and, on completion of cooking, a buzzer is sounded. we can see that the system responds initially to either the full-power or the half-power button. Users can change their mind after selecting one of these and press the other button. The time is set and, if the door is closed,the Start button is enabled. Pushing this button starts the oven operation and cooking takes place for the specified time.

In a detailed system specification, you have to provide more detail about both the stimuli and the system states (Figure 2.23). This information may be maintained in a data dictionary or encyclopaedia .

State	Description
Waiting	The oven is waiting for input. The display shows the current time.
Half power	The oven power is set to 300 watts. The display shows 'Half power .
Full power	The oven power is set to 600 watts. The display shows 'Full power .
Set time	The cooking time is set to the user's input value. The display shows the cooking time selected and is updated as the time is set.
Disabled	Oven operation is disabled for safety. Interior oven light is on. Display shows 'Not ready'.
Enabled	Oven operation is enabled. Interior oven light is off. Display shows 'Ready to cook'.
Operation	Oven in operation. Interior oven light is on. Display shows the timer countdown. On completion of cooking, the buzzer is sounded for 5 seconds. Oven light is on. Display shows 'Cooking complete' while buzzer is sounding.

Stimulus	Description
Half power	The user has pressed the half power button.
Full power	The user has pressed the full power button.
Timer	The user has pressed one of the timer buttons.
Number	The user has pressed a numeric key.
Door open	The oven door switch is not closed.
Door closed	The oven door switch is closed.
Start	The user has pressed the start button.
Cancel	The user has pressed the cancel button.

Fig. 2.23 State and Stimulus description for the microwave oven

The **problem with the state machine approach** is that the number of possible states increases rapidly. For large system models, therefore, some structuring of these state models is necessary. One way to do this is by using the notion of a superstate that encapsulates a number of separate states. This superstate looks like a single state on a high-level model but is then expanded in more detail on a separate diagram.To illustrate this concept, consider the Operation state in Figure 2.22. This is a superstate that can be expanded, as illustrated in Fig 2.24.

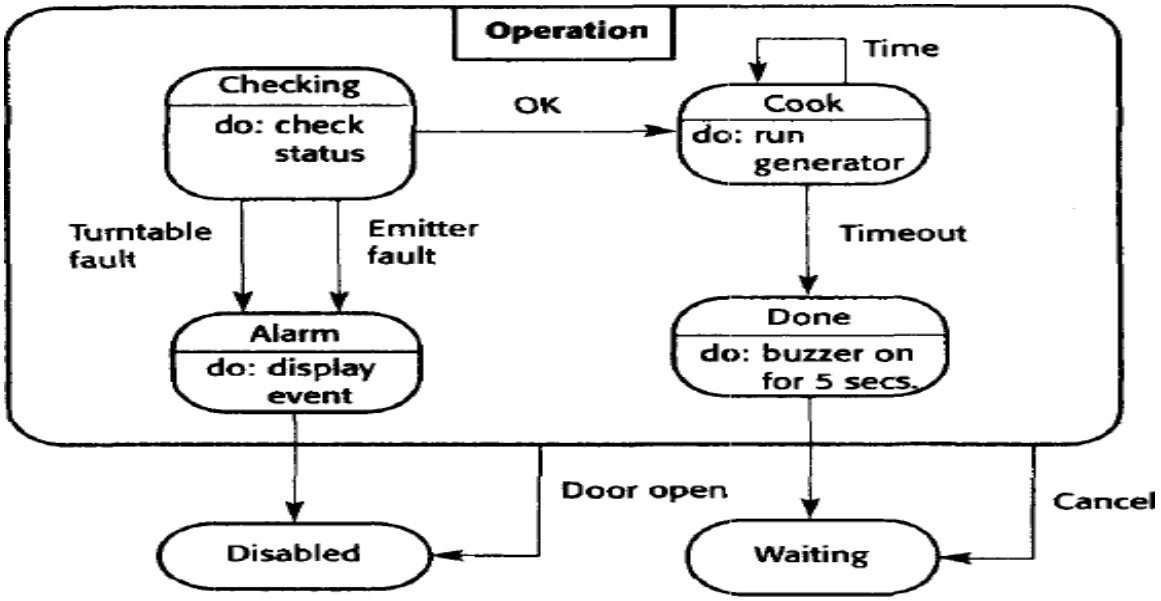


Fig 2.24 Microwave oven operation

- The Operation state includes a number of sub-states. It shows that operation starts with a status check, and that if any problems are discovered, an alarm is indicated and operation is disabled. Cooking involves running the microwave generator for the specified time; on completion, a buzzer is sounded. If the door is opened during operation, the system moves to the disabled state

Semantic Data Models

- ***o*** Used to describe the logical structure of data processed by the system. These are sometimes called ***semantic data models.***
- o An entity-relation-attribute model sets out the entities in the system, the relationships between these entities and the entity attributes
- o Widely used in database design. Can readily be implemented using relational databases.
- o No specific notation provided in the UML but objects and associations can be used.
- o Figure 2.25 is an example of a data model that is part of the library system LIBSYS introduced in earlier chapters. Recall that LIBSYS is designed to deliver copies of copyrighted articles that have been published in magazines and journals and to collect payments for these articles. Therefore, the data model must include information about the article, the copyright holder and the buyer of the article. I have assumed that payments for articles are not made directly but through national copyright agencies.
- o Figure 2.25 shows that an Article has attributes representing the title, the authors,the name of the PDF file of the article and the fee payable. This is linked to the Source, where the article was published, and to the Copyright Agency for the country of publication. Both Copyright Agency and Source are linked to

Country. The country of publication is important because copyright laws vary by country. The diagram also shows that Buyers place Orders for Articles.

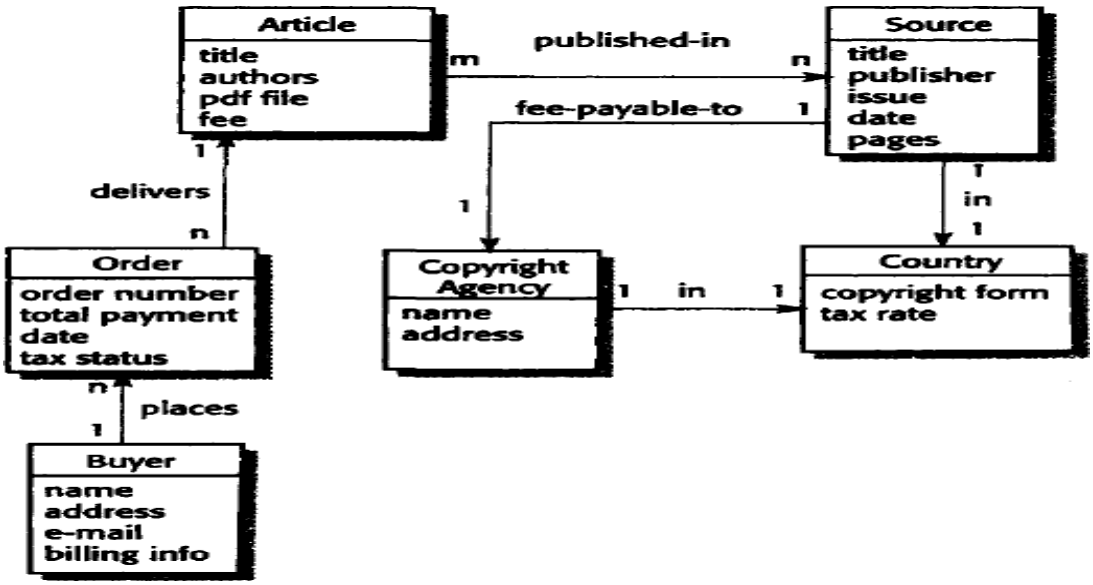


Fig 2.24 Semantic Model for LIBSYS

- Like all graphical models, data models lack detail, and you should maintain more detailed descriptions of the entities, relationships and attributes that are included in the model. **We may collect these more detailed descriptions in a repository or data dictionary.**

- Data dictionaries are generally useful when developing system models
- They may be used to manage all information from all types of system models.
- A data dictionary is. simplistically, an alphabetic list of the names included in the system models. As well as the name, the dictionary should include an associated description of the named entity and, if the name represents a composite object,a description of the composition. Other information such as the date of creation.the creator and the representation of the entity may also be included depending on the type of model being developed.
- **The advantages of using a data dictionary are:**
  1. ***It is a mechanism for name management.*** Many people may have to invent names for entities and relationships when developing a large system model. These names should be used consistently and should not clash. The data dictionary software can check for ame uniqueness where necessary and warn requirements analysts of name duplications.
  2. ***It serves as a store of organisational information.*** As the system is developed,information that can link analysis, design, implementation and evolution is added to the data dictionary, so that all information about an entity is in one place.

Name	Description	Type	Date
Article	Details of the published article that may be ordered by people using LIBSYS.	Entity	30.12.2002
authors	The names of the authors of the article who may be due a share of the fee.	Attribute	30.12.2002
Buyer	The person or organisation that orders a copy of the article.	Entity	30.12.2002
fee-payable-to	A 1:1 relationship between Article and the Copyright Agency who should be paid the copyright fee.	Relation	29.12.2002
Address (Buyer)	The address of the buyer. This is used to any paper billing information that is required.	Attribute	31.12.2002

Fig 2.25 Examples of Data Dictionary Entries

Object Models

- Object models describe the system in terms of object classes and their ssociations.
- An object class is an abstraction over a set of objects with common attributes and the services (operations) provided by each object.
- Objects are executable entities with the attributes and services of the object class. Objects are instantiations of the object class, and many objects may be created from a class.
- In object-oriented requirements analysis, we should model real-world entities using object classes.
- Various object models may be produced
    - Inheritance models;
    - Aggregation models;
    - Interaction models.

- An object class in UML, as illustrated in the examples in Figure 8.10, is represented as a vertically oriented rectangle with three sections:
    I. The name of the object class is in the top section.
    2. The class attributes are in the middle section.
    3. The operations associated with the object class are in the lower section of the rectangle.

- Object class identification is recognised as a difficult process requiring a deep understanding of the application domain

Inheritance Models

Object-oriented modelling involves identifying the classes of object that are important in the domain being studied. These are then organised into a taxonomy. A taxonomy is a classification scheme that shows how all object class is related to other classes through common attributes and services.

the classes are organised into an inheritance hierarchy with the most general object classes at the top of the hierarchy. More specialized objects inherit their attributes and services. These specialized objects may have their own attributes and services.

- Figure 2.26 illustrates part of a simplified class hierarchy for a model of a library.
- This hierarchy gives information about the items held in the library. The library holds various items, such as books, music, recordings of films, magazines and newspapers. In Figure 2.26, the most general item is at the top of the tree and has a set of attributes and services that are common to all library items. These are inherited by the classes Published item and Recorded item, which add their own attributes that are then inherited by lower-level items.

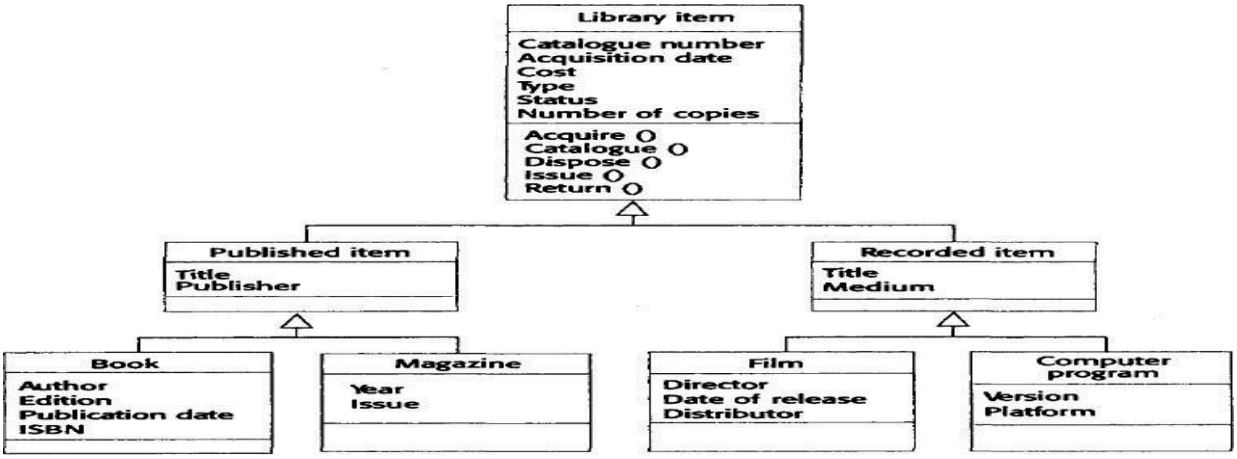


Fig. 2. 26 Part of a class hierarchy for a library

Multiple inheritance models may also be constructed where a class has several parents. Its inherited attributes and services are a conjunction of those inherited from each super-class. Figure 2.27 shows an example of a multiple inheritance model that may also be part of the library model.

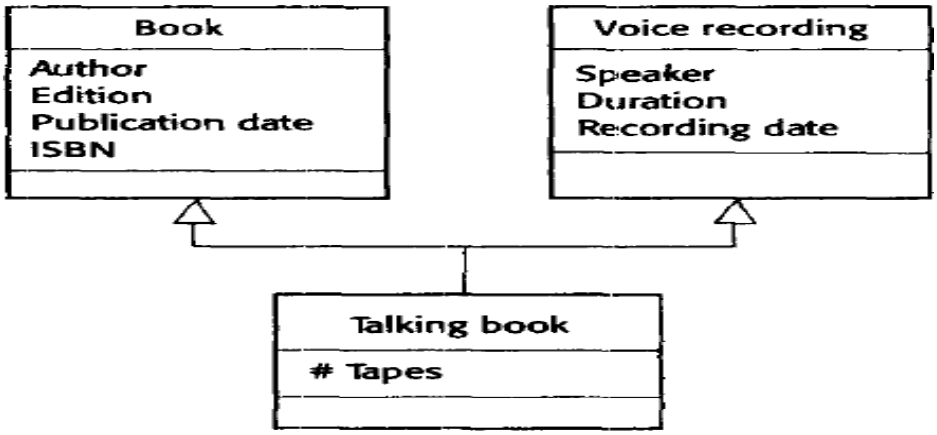


Fig. 2. 27 Multiple Inheritance

Object Aggregation

- An aggregation model shows how classes that are collections are composed of other classes.
- Aggregation models are similar to the part-of relationship in semantic data models.
- in Figure 2.28. I have modelled a library item, which is a study pack for a university course. This study pack includes lecture notes, exercises, sample solutions, copies of transparencies used in lectures, and videotapes.
- The UML notation for aggregation is to represent the composition by including a diamond shape on the source of the link. Therefore, Figure 2.28 can be read as 'A study pack is composed of one of more assignments, OHP slide packages, lecture notes and videotapes

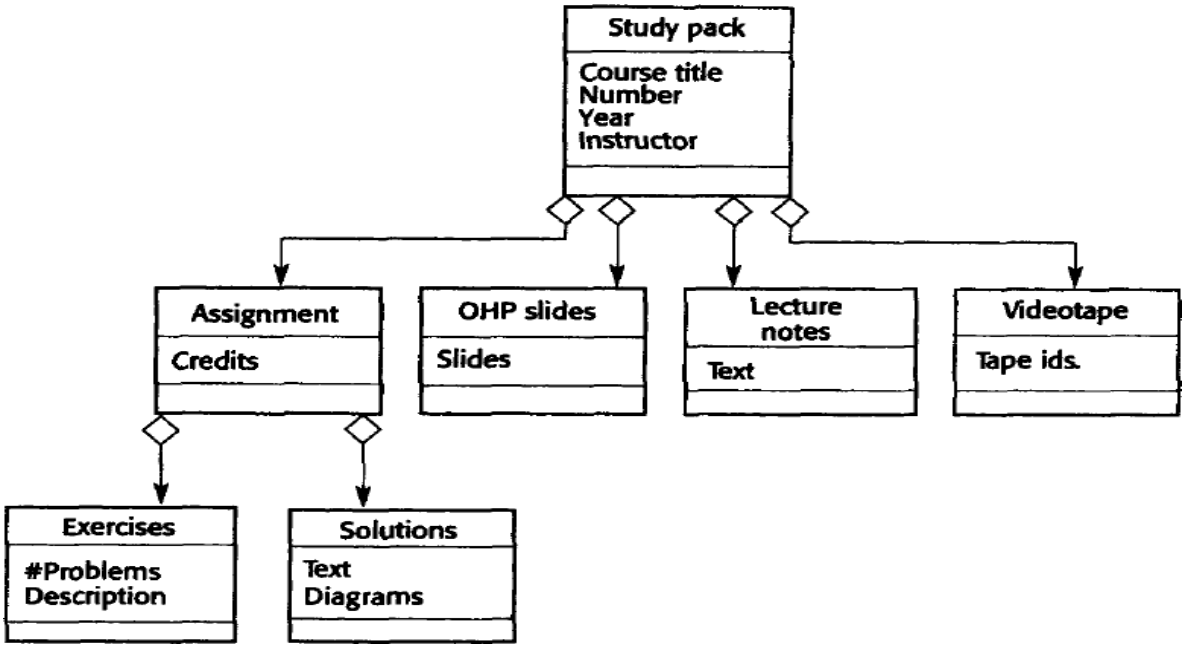


Fig 2.28 Aggregation Object representing a course

Object Behaviour Modelling

- To model the behaviour of objects, you have to show how the operations provided by the objects are used. In the UML, you model behaviours using scenarios that are represented as UML use-cases
- Sequence diagrams (or collaboration diagrams) in the UML are used to model interaction between objects.
- For example, imagine a situation where the study packs shown in Figure 2.28. could be maintained electronically and downloaded to the student's computer.

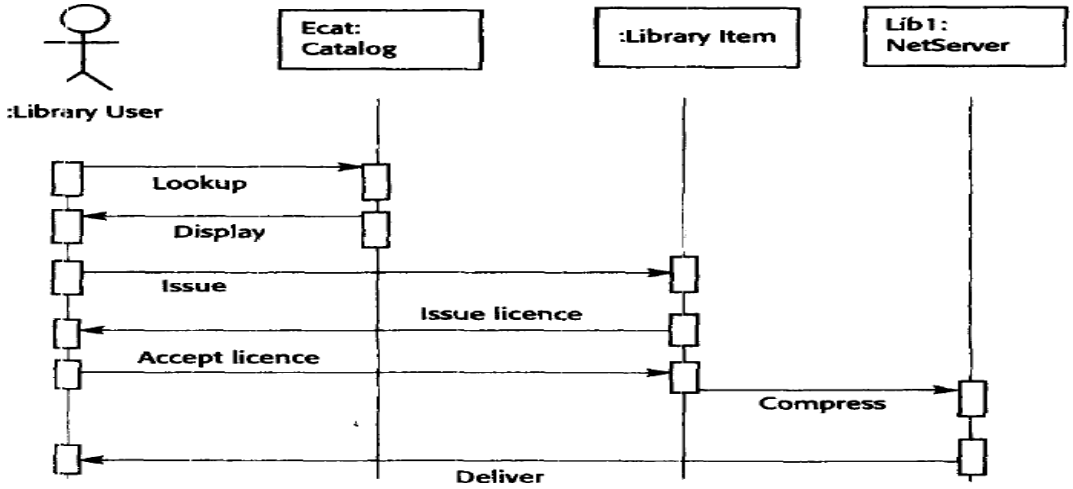


Fig 2.29 The issue of electronic items

In a sequence diagram, objects and actors are aligned along the top of the diagram.Labelled arrows indicate operations; the sequence of operations is from top to bottom. In this scenario, the library user accesses the catalogue to see whether the item required is available electronically; if it is, the user requests the electronic issue of that item. For copyright reasons, this must be licensed so there is a transaction between the item and the user where the license is agreed. The item to be issued is then sent to a network server object for compression before being sent to the library user.

Structured Methods

- A structured method is a systematic way of producing models of an existing system or of a system that is to be built. They were first developed in the 1970s to support software analysis and design
- Methods define a set of models, a process for deriving these models and rules and guidelines that should apply to the models.
- CASE tools support system modelling as part of a structured method.
- Structured methods have been applied successfully in many large projects. However, structured methods suffer from a number of weaknesses:
    - They do not model non-functional system requirements.
    - They do not usually include information about whether a method is appropriate for a given problem.
    - They may produce too much documentation.
    - The system models are sometimes too detailed and difficult for users to understand

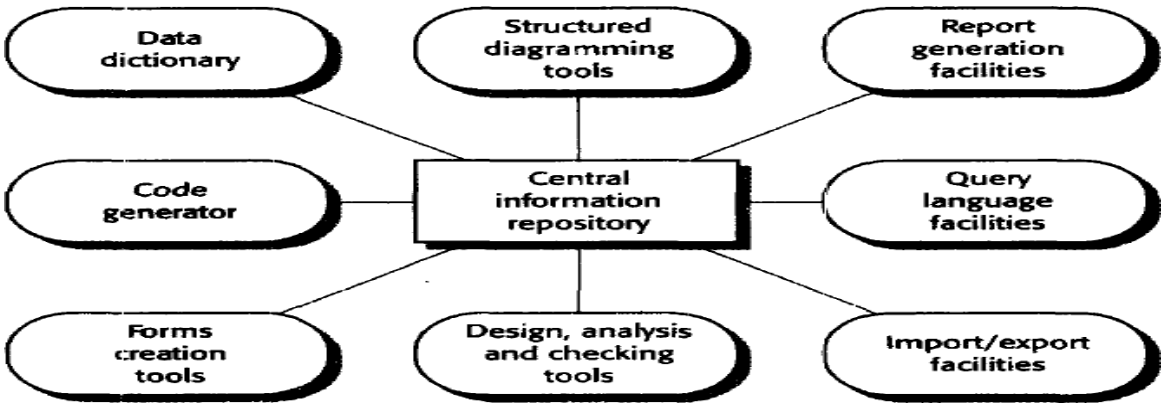


Fig 2.30 The Components of the CASE Tool for structured method support

- A coherent set of tools that is designed to support related software process activities such as analysis, design or testing.

1. **Diagram editors** used to create object models, data models, behavioral models, and so on. These editors are not just drawing tools but are aware of the types of entities in the diagram. They capture information about these entities and save this information in the central repository.

2. **Design analysis and checking tools** that process; the design and report on error and anomalies. These may be integrated with the editing system so that user errors are trapped at an early stage in the process.

3. **Repository query languages** that allow the designer to find designs and associated design information in the repository.

4. **A data dictionary** that maintains information about the entities used in a system design.

5. **Report definition and generation tools** that take information from the central store and automatically generate system documentation.

6. **Forms definition tools** that allow screen and document formats to be specified.

7. **Import/export facilities** that allow the interchange of information from the central repository with other development tools.

8. **Code generators** that generate code or code skeletons automatically from the design captured in the central store.