SOFTWARE ENGINEERING
UNIT-II

REQUIREMENTS ANALYSIS AND SPECIFICATION

2.1 SOFTWARE REQUIREMENTS

- Requirements of a system are the descriptions of the services provided by the system and its operational constraints.
- The requirements reflect the needs of customers for a system that helps solve the problem
- Requirements specify what the system is supposed to do but not how the system is to accomplish the task.
- Requirements should be precise, complete, and consistent
    - Precise - They should state exactly what is desired of the system
    - Complete - They should include descriptions of all facilities required
    - Consistent - There should be no conflicts in the descriptions of the system facilities
- Requirement analysis
    - specifies software's operational characteristics
    - indicates software's interface with other system elements
    - establishes constraints that software must meet
- Requirements analysis allows the software engineer to:
    - elaborate on basic requirements established during earlier requirement engineering tasks
    - Build models that depict user scenarios, functional activities, problem classes and their relationships, system and class behavior, and the flow of data as it is transformed.
- Requirement can be
    - Functional
    - Non-functional
    - Domain

2.1.1 FUNCTIONAL REQUIREMENTS

- Functional requirements describe the services provided by the system, how the system should react to particular inputs and how the system should behave in particular situations.
- The functional requirements may also explicitly state what the system should not do.
- They depend on the type of the system being developed, expected user of the system and approach used for writing the requirements.
- Eg:
    - Compute the transactions (deposit, withdraw, print report etc) in a bank ATM system
    - Compute the salary of the employee in a payroll processing system etc
- The functional requirements specification of a system should be both complete and consistent. Completeness means that all services required by the user should be defined. Consistency means that requirements should not have contradictory definitions.

2.1.2 NON-FUNCTIONAL REQUIREMENTS

- Non functional requirements are the constraints on the services offered by the system such as timing constraints, constraints on the development process, standards, etc.
- Non functional requirements are not directly concerned with specific functions delivered by the system.
- They specify system performance, security, availability, and other emergent properties.

Types of non-functional requirements:

1. **Product requirements:**
    - These requirements specify the product behavior.
    - Ex: execution speed, memory requirement, reliability requirements that set out the acceptable failure rate; portability requirements; and usability requirements etc
2. **Organisational requirements**
    - These requirements are derived from policies and procedures of the organization.

- Ex: 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.

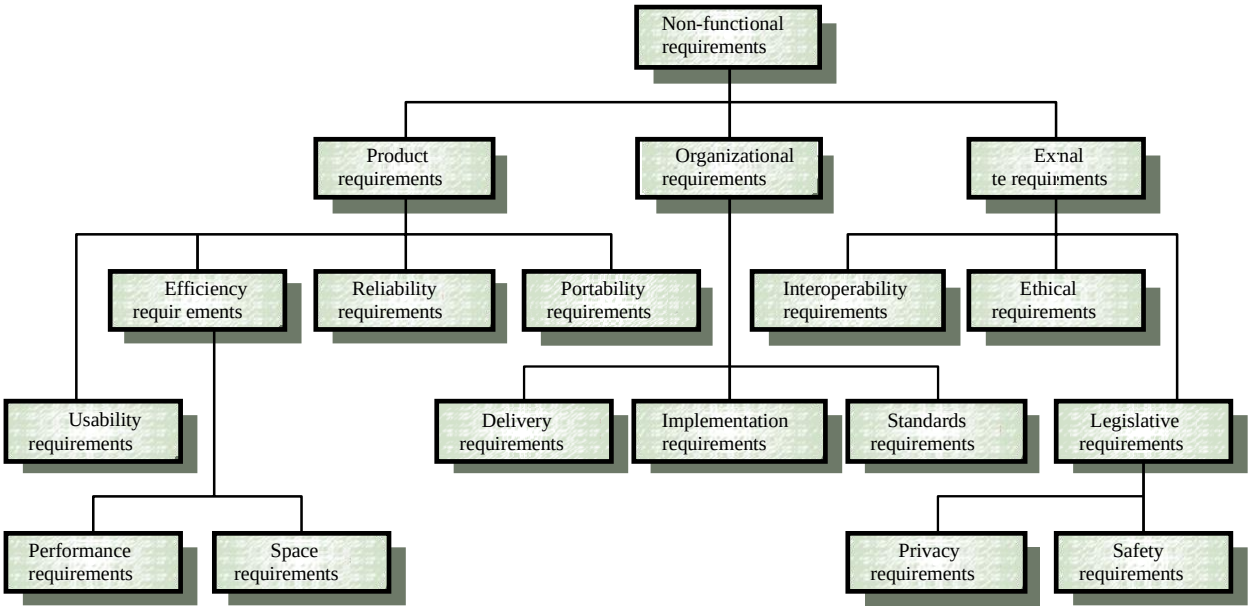


Fig: Non- functional requirements

3. External requirements

- These requirements are derived from factors external to the system and its development process.
- Eg: interoperability requirements that define how the system interacts with systems in other organizations.

- Non functional requirements are more critical than individual functional requirements. if these are not met, the system is useless
- Eg: if an aircraft system does not meet its reliability requirement, it will not be certified as safe for operation. If a real time control system fails to meet performance requirement, the control functions will not operate properly.
- Problem with non functional requirements is that they are difficult to verify.
- Non-functional requirements often conflict and interact with other functional or non-functional requirements.
- It's better to differentiate functional and non-functional requirements in the requirements document. But it's difficult to do so.

- Metrics used to specify the non functional requirements are:

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: Metrics for specifying non- functional requirements

- If the non-functional requirements are stated separately from the functional requirements, it is sometimes difficult to see the relationships between them.
- If they are stated with the functional requirements, then it may be difficult to separate functional and nonfunctional considerations and to identify requirements that relate to the system as a whole
- So the requirements that are clearly related to emergent system properties, such as performance or reliability can be explicitly highlighted by putting them in a separate section of the requirements document or distinguishing them from other system requirements

Domain requirements

- These requirements come from the application domain of the system.
- They reflect the characteristics and constraints of that domain
- They may be new functional requirement or non functional requirementss) or set out how particular computations must be carried out
- Eg: LIBSYS
  - There shall be a standard user interface to all databases that shall be based on the Z39.50 standards. This is a design constraint. So the developer has to find out about the standard before starting the interface design.
  - The deceleration of the train shall be computed as:

$$D_{train} = D_{control} + D_{gradient}$$

- This uses a domain specific terminology. To understand this you need understanding of operation of railway system and train characteristics

- Drawback: domain requirements are written in the language of application domain which is difficult for the software engineer to understand.

2.1.3 USER RQUIREMENTS

- User requirements describe what services the system is expected to provide and the constraints under which it must operate.
- These requirements should describe functional and non-functional requirements so that they are understandable by system users who don't have detailed technical knowledge
- They should specify only the external behaviour and avoid the system design characteristics
- User requirements are defined using natural language, tables, and diagrams
- Problems faced with natural language:
  - **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 requirements, non-functional requirements, system goals and design information may not be clearly distinguished.
  - **Requirements amalgamation:** Several different requirements may be expressed together as a single requirement.
- This requirement includes both conceptual and detailed information.
- The user requirement should simply focus on the key facilities to be provided.
- A rationale can be associated with each user requirement which explains why the requirement has been included and is particularly useful when requirements are changed
- Guidelines to be followed to minimize misunderstandings when writing user requirements:
  - Use a standard format to define all the requirement.
  - Use language consistently.
  - Use text highlighting (bold, italic or color) to pick out key parts of the requirement.
  - Avoid using computer jargon /technical terms

2.1.4 SYSTEM REQUIREMENTS

- System requirements describe the external behavior of the system and its operational constraints.
- This explains how the user requirements should be provided by the system
- System requirement should be a complete and consistent specification of the whole system
- Natural language is often used to write system requirements specifications
- Drawbacks of using natural language specifications for specifying system requirements:
  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.
  2. A natural language requirements specification is over flexible.
  3. It is difficult to modularize natural language requirements. It may be difficult to find all related requirements.

Notations used for System requirements specification

- **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 feature to specify the requirements by defining an operational model of the system
- **Graphical notations**
  - A graphical language, supplemented by text annotations is used to define the functional requirements for the system. Eg: SADT(structured analysis and design techniques), use case diagram, sequence diagrams etc
- **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

Structured natural language specifications

- Structured natural language is a way of writing system requirements in a standard format.
- Advantage of this approach is that it maintains most of the expressiveness and understandability of natural language but ensures that some degree of uniformity is imposed on the specification.
- Structured language notations limit the terminology that can be used and use templates to specify system requirements.
- They may include control constructs derived from programming languages and graphical highlighting to partition the specification.
- When a standard form is used for specifying functional requirements, the following information should be included:
    1. Description of the function or entity being specified
    2. Description of its inputs and where these come from
    3. Description of its outputs and where these go to
    4. Indication of what other entities are used (the *requires* part)
    5. Description of the action to be taken
    6. If a functional approach is used, a pre-condition setting out what must be true before the function is called and a post-condition specifying what is true after the function is called
    7. Description of the side effects (if any) of the operation.

Advantages:

- Variability in the specification is reduced and requirements are organized more effectively.
- To avoid ambiguity, add extra information to natural language requirements using tables or graphical models of the system. These can show how computations proceed, how the system state changes, how users interact with the system and how sequences of actions are performed.

- Eg : **To compute the insulin dose for a person**

Insulin Pump/Control Software/SRS/3.3.2	
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

Fig: system requirements to compute the insulin dosage for a person

- **Tables** are useful when there are a *number of possible alternative situations and actions to be taken*.
- Eg:

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: sample table for system requirement specification

- **Graphical models** are most useful to show how state changes and to describe a sequence of actions
- Eg: sequence diagram for ATM withdrawal

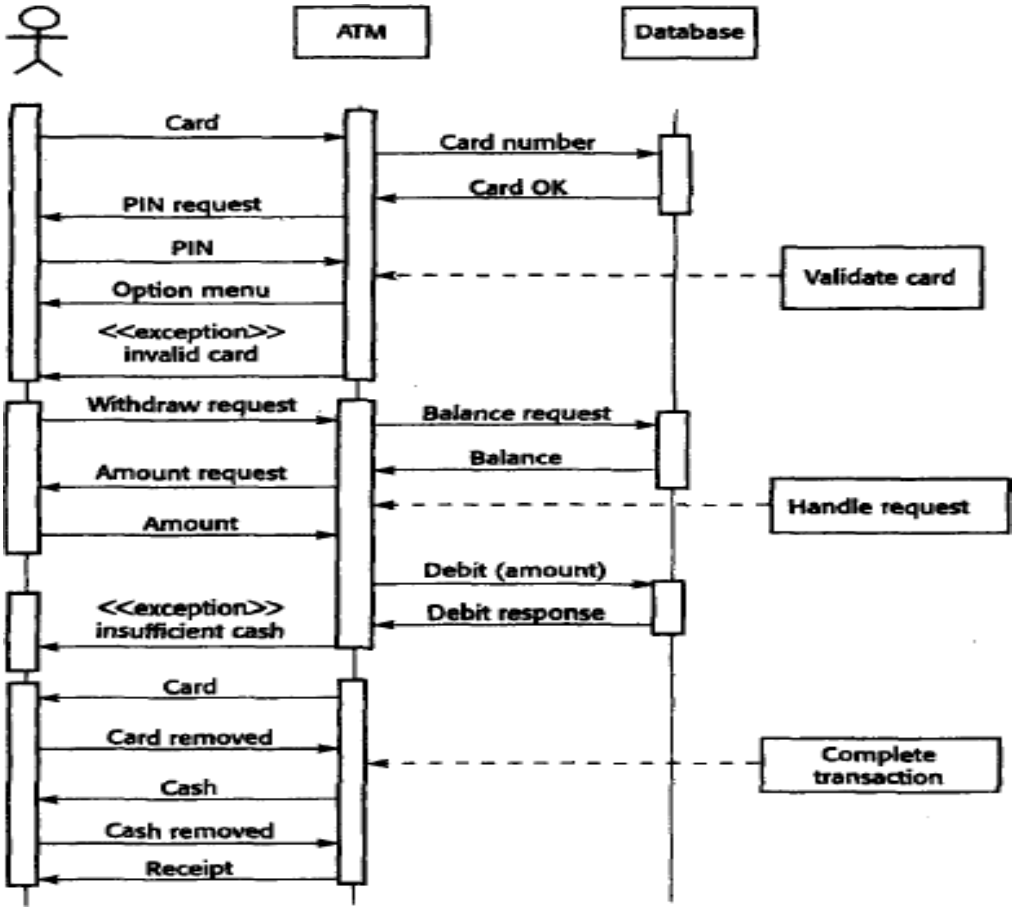


Fig: Sequence diagram for withdraw()

2.2 REQUIREMENTS ENGINEERING

- Requirements engineering (RE) is the process of identifying, analyzing, documenting and checking the services and constraints.
- Requirements engineering process is to create and maintain a system requirements document.
- Requirements engineering includes the task and techniques used to understand the basic requirements of the system.

- Process involved in Requirements engineering:

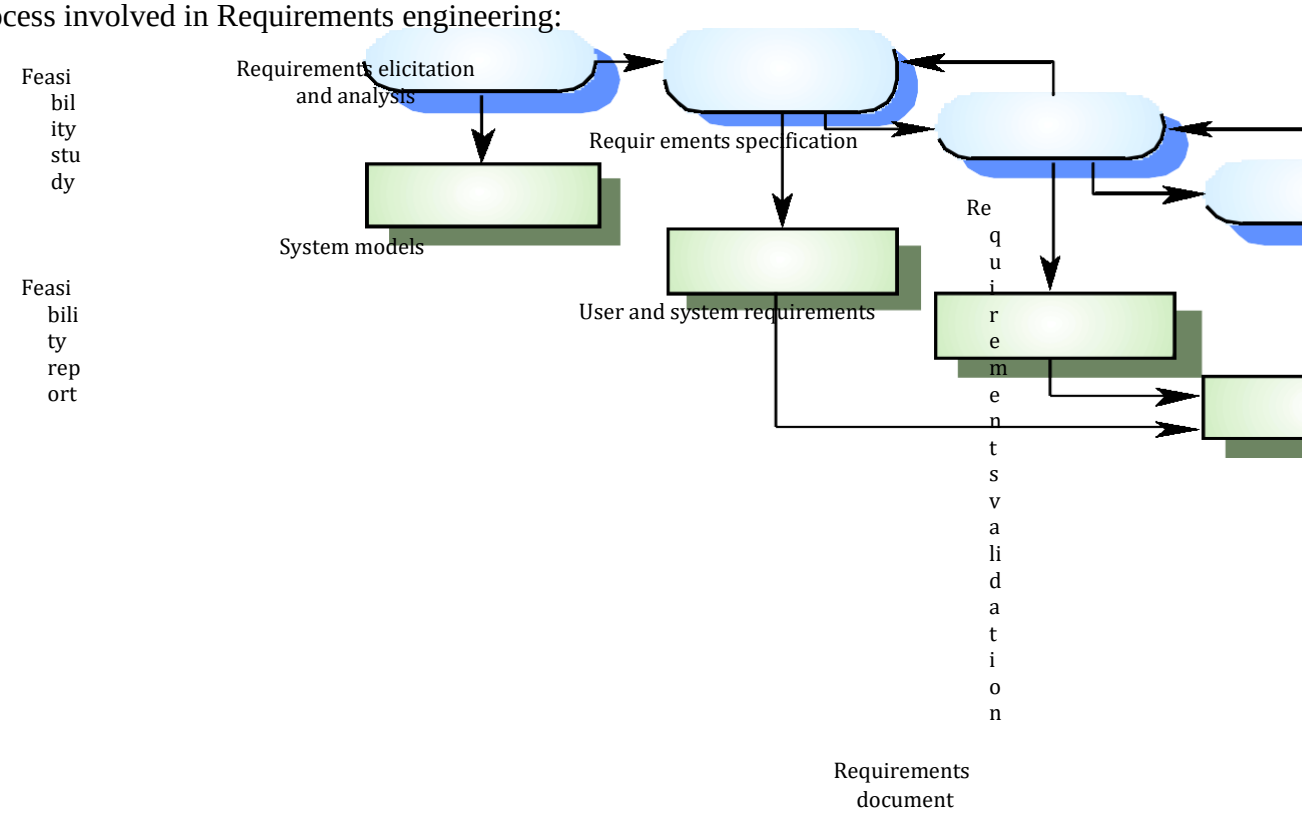


Fig: Requirements Engineering Process

1. **Feasibility Study**- concerned with assessing whether the system is useful to the business
2. **Requirements elicitation and analysis** - discovering requirements
3. **Requirements specification** - converting these requirements into some standard form
4. **Requirements Validation** - checking that the requirements actually define the system that the customer wants.

- The requirements engineering process is an iterative process around a spiral.

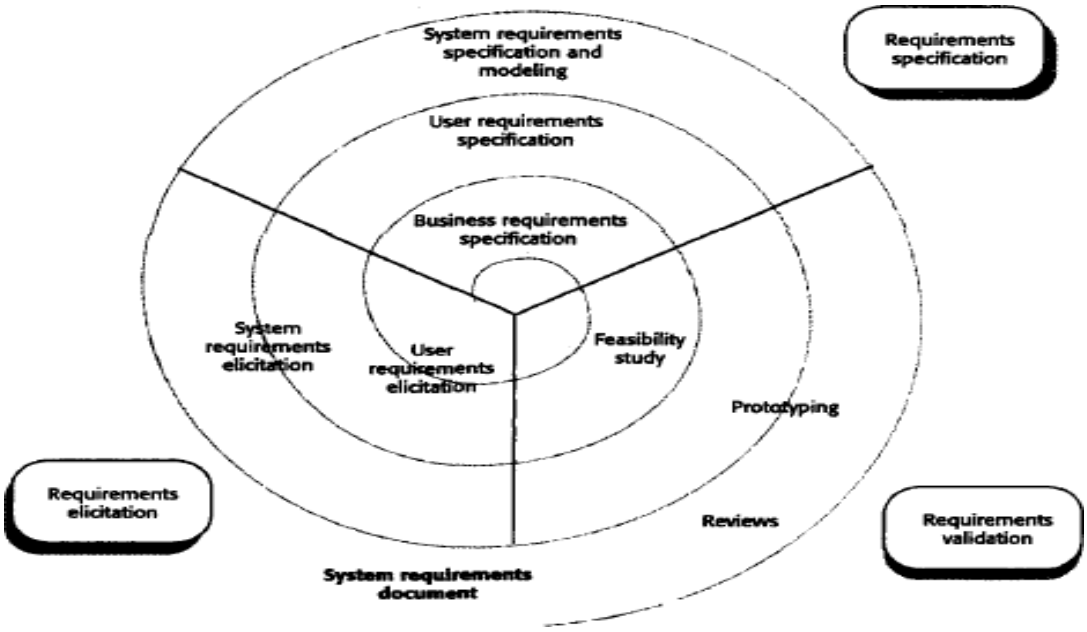


Fig: Requirements Engineering

- The amount of time and effort for each activity in the iteration depends on the stage of the overall process and the type of system being developed.
- Early stage includes understanding the non-functional requirements and the user requirements. Later stages are devoted to system requirements engineering and system modeling.
- The number of iterations around the spiral can vary
- Requirements engineering is the process of applying a structured analysis method such as object-oriented analysis
- This involves analyzing the system and developing a set of graphical system models, such as use-case models, that then serve as a system specification.

2.2.1 FEASIBILITY STUDY

- Feasibility study is used to determine if the user needs can be satisfied with the available technology and budget
- Feasibility study checks the following:
  - Does the system contribute to organisational objectives?
  - Can the system can be implemented using current technology and within budget
  - Can the system can be integrated with other systems that are used
- If a system does not support these objectives, it has no real value to the business.
- Feasibility study is based on the information assessment, information collection and report writing.
- Sample Questions that may be asked for information collection are:
  1. What if the system wasn't implemented?
  2. What are current process problems?
  3. How will the proposed system help?
  4. What will be the integration problems?
  5. Is new technology needed? What skills?
  6. What facilities must be supported by the proposed system?
- Information sources are the managers of the departments, software engineers, technology experts and end-users of the system.
- The feasibility study should be completed in two or three weeks.
- After collecting the information, the feasibility report is created.

- In the report, changes to the scope, budget and schedule of the system can be proposed and also suggest further high-level requirements for the system.

1.2.2 REQUIREMENTS ELICITATION AND ANALYSIS

- Requirements elicitation is nothing but identifying the application domain, the services that the system should provide and the constraints.
- Requirements are gathered from the *stakeholders*
- Stakeholders are any person or group who will be affected by the system, directly or indirectly. Eg: end-users, managers, engineers, domain experts etc
- Drawbacks of Eliciting and understanding stakeholder requirements:
    1. Stakeholders don't know what they really want.
    2. Stakeholders express requirements in their own terms.
    3. Different stakeholders may have conflicting requirements.
    4. Organisational and political factors may influence the system requirements.
    5. The requirements change during the analysis process. New stakeholders may emerge and the business environment change.

Steps involved in requirements elicitation and analysis

1. *Requirements discovery*

   This is the process of interacting with stakeholders in the system to collect their requirements. Domain requirements are also identified.

2. *Requirements classification and organisation*

   This activity takes the unstructured collection of requirements, groups related requirements and organises them into coherent clusters.

3. *Requirements prioritisation and negotiation*

   Since multiple stakeholders are involved, requirements will conflict. This activity is concerned with prioritizing the 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 for further requirements discovery. Formal or informal requirements documents may be produced.

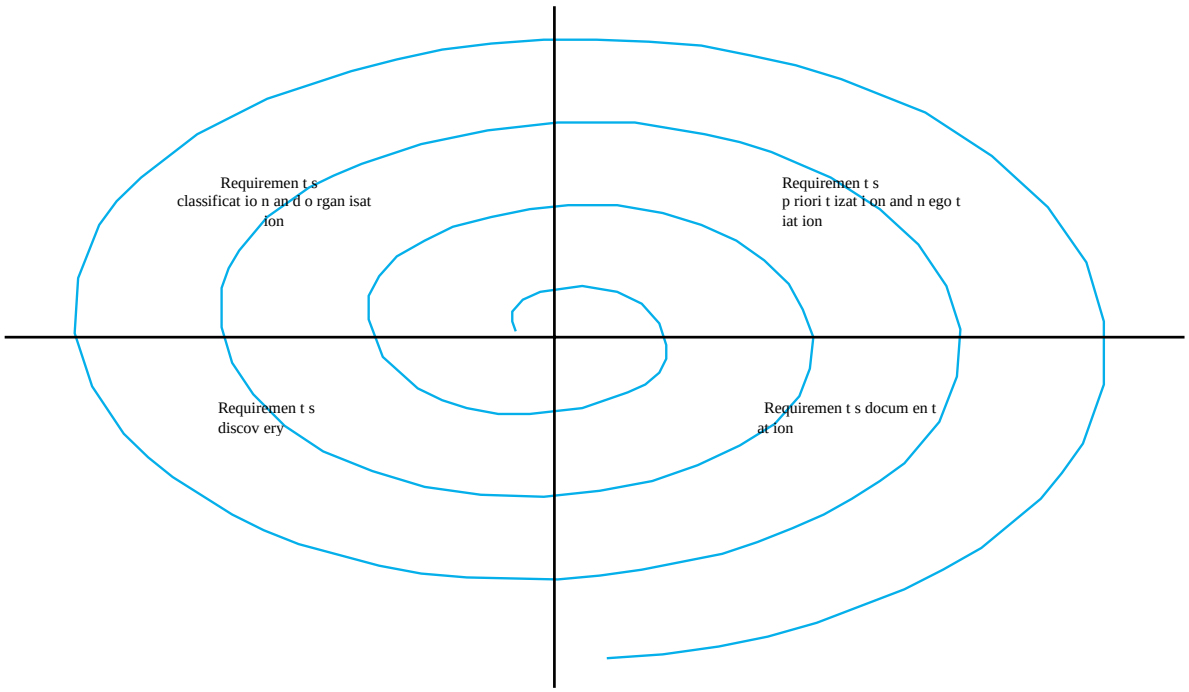


Fig: Requirement Elicitation and Analysis

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.
- Sources of information during the requirements discovery phase include documentation, system stakeholders and specifications of similar systems.
- Stakeholders are interacted through interviews and observation, and may use scenarios and prototypes to help with the requirements discovery.
- Stakeholders range from system end-users through managers and external stakeholders such as regulators who certify the acceptability of the system.
- For example,

  System stakeholders for a bank ATM include:

  Current bank customers, Representatives from other banks, Managers of bank branches, Counter staff at bank branches, Database administrators, Bank security managers, bank's marketing department, Hardware and software maintenance engineers, National banking regulators etc
- The requirements may also come from the application domain and from other systems that interact with the system being specified.

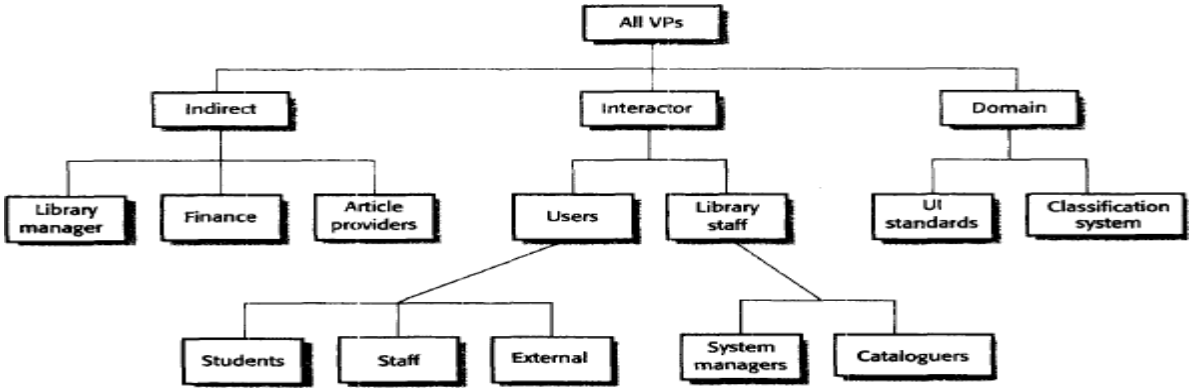
- These requirements sources (stakeholders, domain, systems) can all be represented as **system viewpoints**
- Each viewpoint presents a sub-set of the requirements for the system. Each viewpoint provides a fresh perspective on the system, but these perspectives are not completely independent--they usually overlap so that they have common requirements.

Viewpoints

- Viewpoint-oriented approach recognizes multiple perspectives of the stakeholders and provides a framework for discovering conflicts in the requirements proposed by different stakeholders.
- Types of viewpoint:

1. *Interactor viewpoints*
    - This viewpoint represents people or other systems that interact directly with the system.Eg: In bank ATM system, the interactor viewpoints are the bank's customers and the bank's account database.
    - Interactor viewpoints provide detailed system requirements covering the system features and interfaces.
2. *Indirect viewpoints*
    - This represents the views of stakeholders who do not use the system themselves but who influence the requirements in some way.
    - Eg: In the bank ATM system, the indirect viewpoints are the management of the bank and the bank security staff.
    - Indirect viewpoints are more likely to provide higher-level organisational requirements and constraints
3. *Domain viewpoints*
    - *This represents* the domain characteristics and constraints that influence the system requirements.
    - *Eg:* In the bank ATM system, the domain viewpoint would be the standards that have been developed for interbank communications.
    - Domain viewpoints normally provide domain constraints that apply to the system

- Almost all organisational systems must interoperate with other systems in the organisation. When a new system is planned, the interactions with other systems must be planned which may place requirements and constraints on the new system.
- Finally organise and structure the viewpoints into a hierarchy.
- Eg: viewpoint hierarchy for LIBSYS



- Once viewpoints have been identified and structured, identify the most important viewpoints and start with them to discover the system requirements.

Interviewing

- RE team asks questions to stakeholders about the system they are using and the system to be developed and derive the requirements from their answers.
- 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 and a range of issues are explored with stakeholders.

- Completely open-ended discussions rarely work well; most interviews require some questions to get started and to keep the interview focused on the system to be developed.
- Interviews are good for getting an overall understanding of what stakeholders do, how they might interact with the system and the difficulties that they face with current systems.
- Interviews are not good for understanding the domain requirements
- Interviews are not an effective technique for eliciting knowledge about organisational requirements and constraints
- It is hard to elicit domain knowledge during interviews for two reasons:
  - Requirements engineers cannot understand specific domain specific terminology;

- Some domain knowledge is so familiar that people find to explain or they think it is so fundamental that it isn't worth mentioning.

Scenarios

- Scenarios are real-life examples of how a system can be used.
- Scenarios are useful for adding detail to an outline requirements description. They are descriptions of example interaction sessions.
- Each scenario covers one or more possible interactions.
- The scenario starts with an outline of the interaction, and, during elicitation, details are added to create a complete description of that interaction.
- Scenarios should include
    - A description of the starting situation;
    - A description of the normal flow of events;
    - A description of what can go wrong and how it is handled;
    - Information about other concurrent activities;
    - A description of the state when the scenario finishes.
- Scenarios may be written as text, diagrams, screen shots etc
- Alternatively, a more structured approach such as event scenarios or usecases may be adopted

Use cases

- Use-cases are a scenario based technique in the UML (Unified Modelling Language) notation for describing object oriented system model.
- Use cases identify the actors involved and the type of interaction between them.
- Eg:

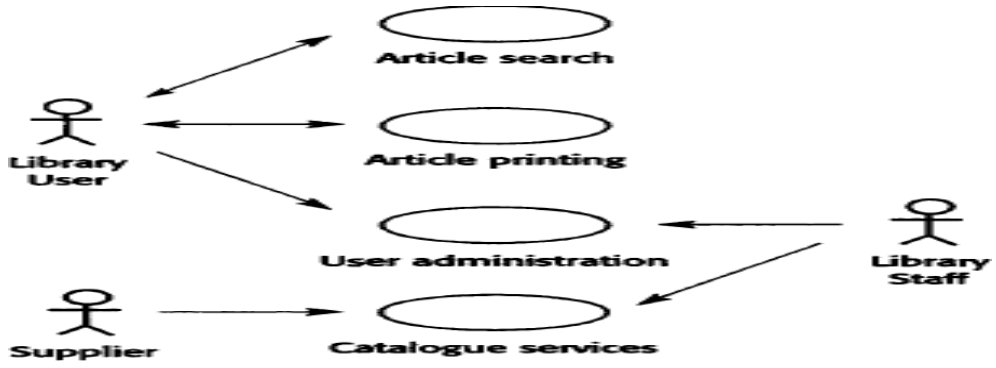


Fig: Use case diagram for article printing

- A set of use cases should describe all possible interactions with the system.
- Actors in the process are represented as stick figures, and each class of interaction is represented as a named ellipse.

Sequence diagrams

- Sequence diagrams are used to add detail to use-cases by showing the sequence of event processing in the system.
- Eg:

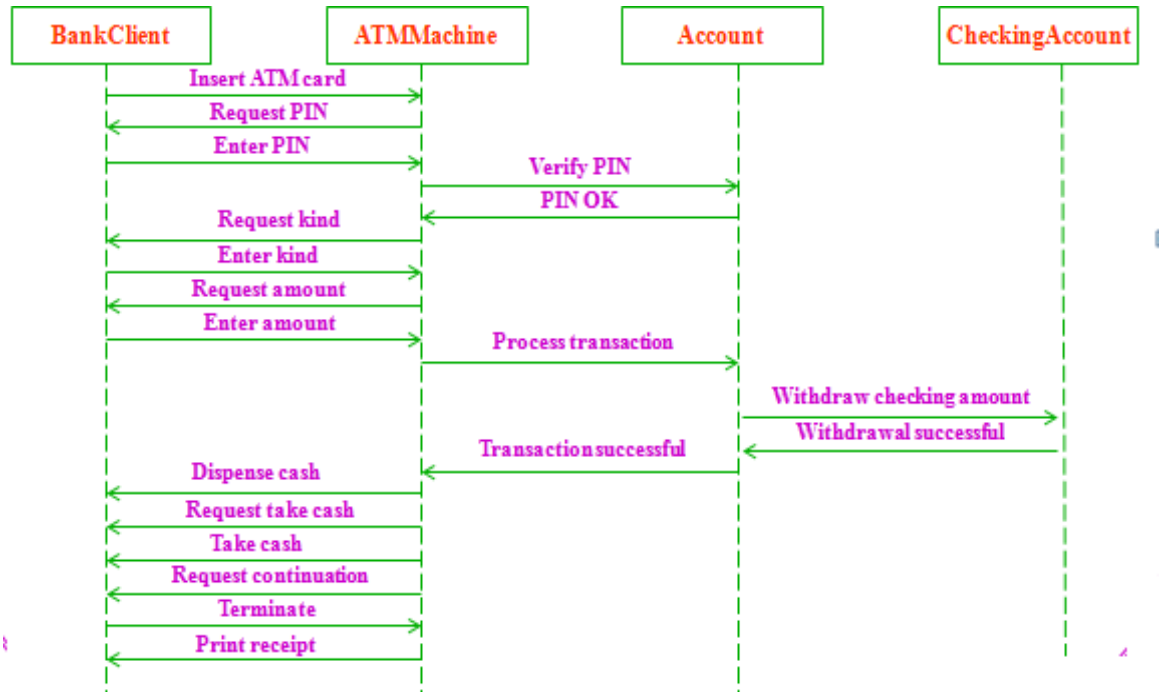


Fig: Sequence diagram for withdraw from ATM

Ethnography

- *Ethnography* is an observational technique that can be used to understand social and organisational requirements.
- Ethnography helps analysts to discover implicit system requirements that reflect the actual rather than the formal processes in which people are involved.
- Ethnography is particularly effective at discovering two types of requirements:
    - Requirements that are derived from the way that people actually work rather than the way in which process definitions suggest that they ought to work.
    - Requirements that are derived from cooperation and awareness of other people's activities

- Ethnography may be combined with prototyping.

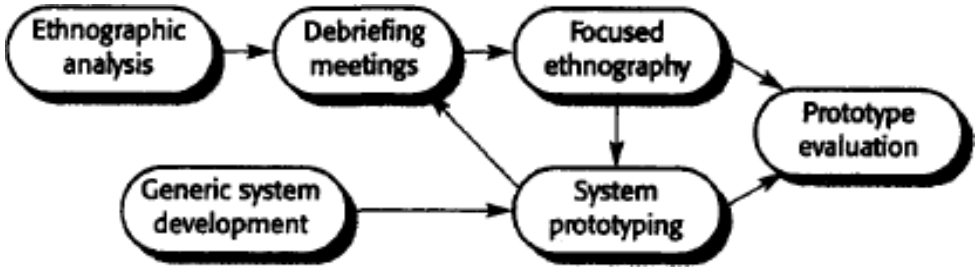


Fig: Ethnography process

- The ethnography informs the development of the prototype so that fewer prototype refinement cycles are required.
- Furthermore, the prototyping focuses the ethnography by identifying problems and questions that can then be discussed with the ethnographer.
- Ethnographic studies can reveal critical process details that are often missed by other requirements elicitation techniques.
- This approach is not appropriate for discovering organisational or domain requirements as it focuses on the end users.

2.2.3 SOFTWARE REQUIREMENTS DOCUMENT/ SOFTWARE SPECIFICATION

- Also referred as software requirements specification or SRS
- The software requirements document 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.
- The user and system requirements may be integrated into a single description.
- 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.

- Users of the requirements document are:

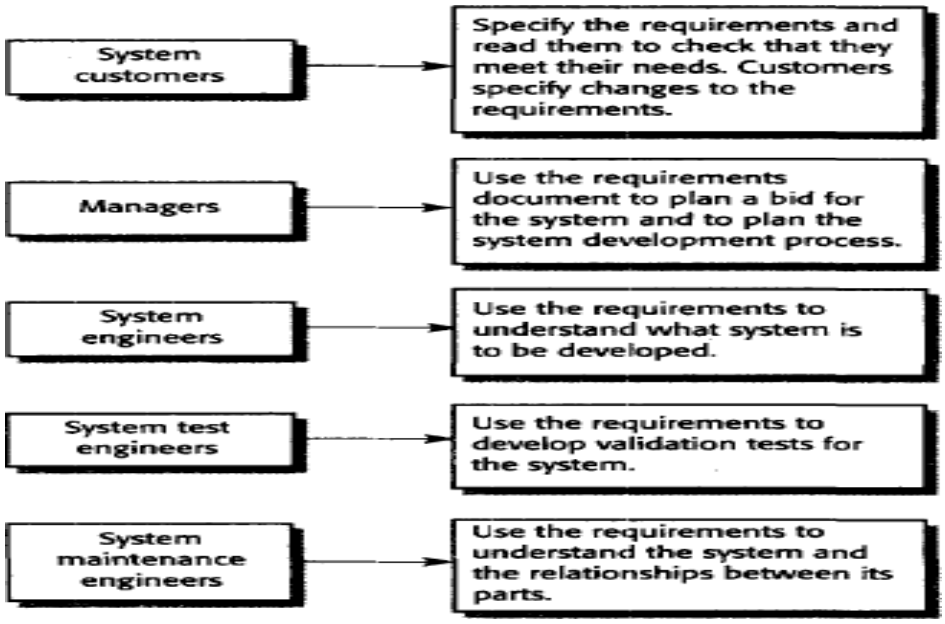


Fig: Users of SRS

- Requirements document helps in communicating the requirements to customers, defining the requirements in detail for developers and testers, and including information about possible system evolution.
- Requirements information can help system designers to avoid restrictive design decisions and help system maintenance engineers who have to adapt the system to new requirements.
- The level of detail to be included in a requirements document depends on the type of system that is being developed and the development process used.
- When the system will be developed by an external contractor, the system specifications need to be precise and very detailed.
- When there is more flexibility in the requirements and an iterative development process is used, the requirements document can be much less detailed
- **IEEE standard suggests the following structure for requirements documents:**

1. Introduction

1.1 Purpose of the requirements 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

- It is a general framework that can be tailored and adapted to define a standard to the needs of a particular organization
- Requirements documents are essential when an outside contractor is developing the software system.
- The focus will be on defining the user requirements and high-level, non-functional system requirements.
- When the software is part of a large system engineering project that includes interacting hardware and software systems, it is essential to define the requirements to a fine level of detail.

2.2.4 REQUIREMENTS VALIDATION

- Requirements validation is concerned with showing that the requirements actually define the system that the customer wants.
- Requirements error costs are high so validation is very important
- When the requirements change that the system design and implementation must also be changed and then the system must be tested again. So the cost of fixing a requirement problem is greater than repairing the design or coding errors

- Requirements checking:

1. *Validity checks*
    - A user needs the system to perform certain functions.
    - Does the system provide the functions which best support the customer's needs?
2. *Consistency checks*
    - Are there any requirements conflicts?
3. *Completeness checks*
    - The requirements document should include requirements, which define all functions, and constraints intended by the system user.
    - Are all functions required by the customer included?
4. *Realism checks*
    - Can the requirements be implemented given available budget and technology?
5. *Verifiability*
    - To reduce the potential for dispute between customer and contractor, system requirements should always be written so that they are verifiable.
    - Can the requirements be checked?

Requirements validation techniques

1. *Requirements reviews*

   The requirements are analysed systematically by a team of reviewers.
2. *Prototyping*

   *A*n executable model of the system is demonstrated to end-users and customers to see if it meets their real needs
3. *Test-case generation*

   Requirements should be testable. This approach is for developing tests for requirements to check testability.

   If the tests for the requirements are devised as part of the validation process, this often reveals requirements problems. If a test is difficult or impossible to design, then the requirements will be difficult to implement and should be reconsidered.

Requirements reviews

- A requirements review is a manual process that involves people from both client and contractor organisations to check the requirements document for anomalies and omissions.
- Requirements reviews can be informal or formal.
- Informal reviews involve contractors discussing requirements with the system stakeholders. Good communications can help to resolve problems at an early stage.
- In a formal requirements review, the development team explains the implications of each requirement to the client. The review team should check each requirement for consistency as well as completeness.
- Reviewers may also check for:
  1. **Verifiability**. Is the requirement realistically testable?
  2. **Comprehensibility**. Is the requirement properly understood?
  3. **Traceability**. Is the origin of the requirement clearly stated? Traceability is important as it allows the impact of change on the rest of the system to be assessed
  4. **Adaptability**. Can the requirement be changed without a large impact on other requirements?
- Conflicts, contradictions, errors and omissions in the requirements should be pointed out by reviewers and formally recorded in the review report.
- It is then up to the users, the system procurer and the system developer to negotiate a solution to these identified problems.

1.3 REQUIREMENTS MANAGEMENT

- Requirements management is the process of managing changing requirements during the requirements engineering process and system development.
- Requirements management is the process of understanding and controlling changes to system requirements.
- Requirements are incomplete and inconsistent because:
  - New requirements arise 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.

- Requirements may change over time for the following reasons:
  - The priority of requirements from different viewpoints changes during the development process.
  - System customers may specify requirements from a business perspective that conflict with end-user requirements.
  - The business and technical environment of the system changes during its development.

Types of requirements:

- Enduring requirements.

These are 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.

These are requirements which change during development or when the system is in use. In a hospital, requirements derived from health-care policy

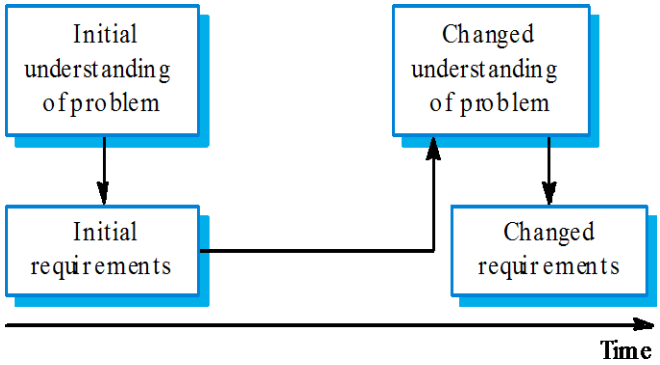


Fig: Requirement Evolution

- **Mutable requirements**

Requirements that 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 that 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 that result from the introduction of the computer system. Introducing the computer system may change the organisations processes and open up new ways of working which generate new system requirements

- **Compatibility requirements**

  Requirements that 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.

Requirements management planning:

- During the requirements engineering process, you have to plan:
  - Requirements identification
    - Each requirement must be uniquely identified so that it can be cross-referenced by other requirements and so that it may be used in traceability assessments.
  - 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
- **Types of traceability information**:
  - **Source traceability**
    - Links from requirements to stakeholders who proposed these requirements;
  - **Requirements traceability**
    - Links between dependent requirements;

- This information helps to assess how many requirements are likely to be affected by a proposed change

- **Design traceability**
  - Links from the requirements to the design;
  - This information helps to assess the impact of proposed requirements changes on the system design and implementation.

➢ **Fig: Traceability Matrix**

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			D		D
1.3	R			R				
2.1			R		D			D
2.2								D
2.3		R		D				
3.1								R
3.2							R	

CASE tools support for:

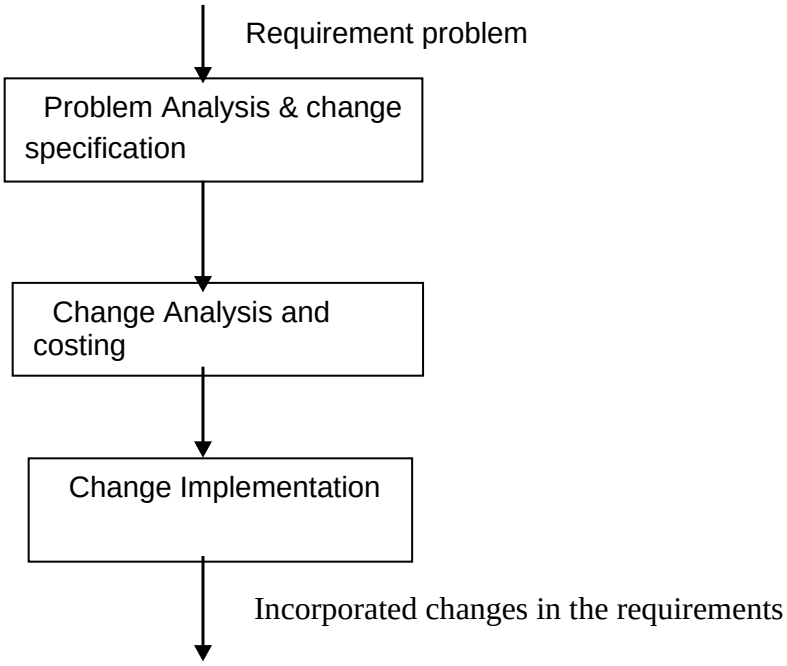
- Requirements storage
  - Requirements should be managed in a secure, managed data store.
- Change management
  - The process of change management is simplified using CASE tools.
- Traceability management
  - Tools help to discover possible relationship between the requirements.

2.4 REQUIREMENTS CHANGE MANAGEMENT

- Requirements change management should be applied to all proposed changes to the requirements
- Stages
  - **Problem analysis**. Identifying the requirements problem and propose change;
  - **Change analysis and costing**. Assess effects of change on other requirements and estimate the cost of the impact using the traceability information

- **Change implementation**. Modify requirements document and other documents to reflect change

Fig: Requirements change management process



2.5 SYSTEM MODELING

2.5.1 STRUCTURED SYSTEM ANALYSIS

- System models are graphical representations that describe business processes, the problem to be solved and the system that is to be developed.
- Graphical representations are often more understandable than detailed natural language descriptions of the system requirements
- **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

Types of analysis models:

- Structured analysis
- Object-oriented analysis

Three primary objectives of the analysis model:

- to describe what the customer requires

- to establish a base for the creation of a software design
- to define a set of requirements that can be validated

Elements of Analysis model:

- **The core - data dictionary**

  - this contains the descriptions of all data objects in the software

- **The entity-relationship diagram (ERD)**

  - ERD specifies the attributes of data objects and also depicts the relationships between data objects.

- **The data flow diagram (DFD)**

  - DFD provides an indication of how data are transformed as they move through the system.

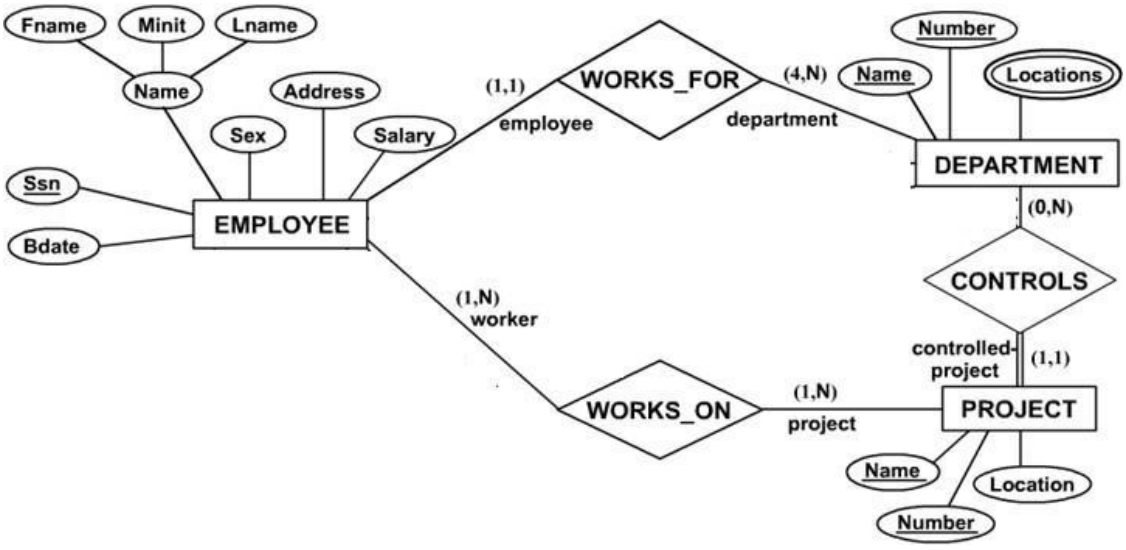
  - DFD depicts the functions that transform the data flow.

- **The state-transition diagram(STD)**

  -This indicate how the system behaves as a consequence of external events

DATA MODELLING:

- Entity-Relationship Diagram is a very useful method for data modeling.
- It represents:
  - data objects, object attributes, and relationships between objects
- **Eg:**



PROCESS MODELLING

- Process model represents the system's function.
- This graphically represent the process that capture, manipulate, store and distribute data between the system and its environment
- Eg: Data Flow Diagram

DATA FLOW DIAGRAM

- A Data Flow Diagram (DFD) is a graphical representation of the flow of data through an information system, modeling its *process* aspects.
- This is used to create an overview of the system

Elements of DFD:

- **external entity** - people or organisations that send data into the system or receive data from the system
- **process** - models what happens to the data i.e. transforms incoming data into outgoing data
- **data store** - represents permanent data that is used by the system
- **data flow** - models the actual flow of the data between the other elements

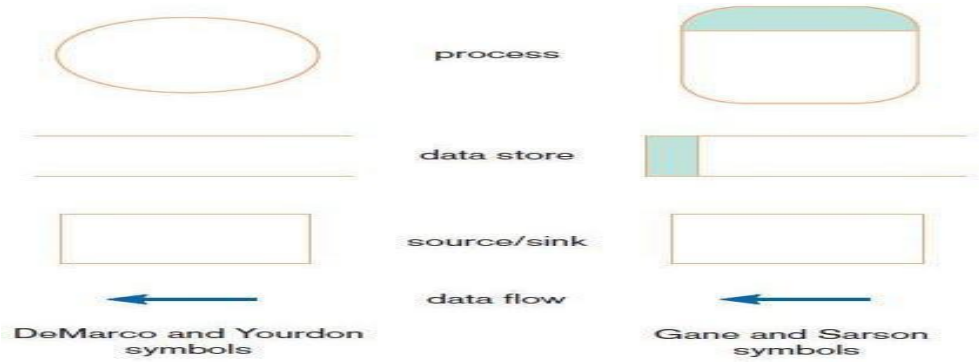


Fig: Symbols used in DFD

DFD Diagramming rules:

- Data flow can take place between processes from
  - data store to a process and a process to a data store
  - external entity to a process and from process to an external entity
- Data flows can't take place between 2 data stores or two external entities
- Data flows are unidirectional
- A process must have at least one input and one output

Developing DFDs

- **Context diagram** is an overview of an organizational system that shows:
  - The system boundaries.
  - External entities that interact with the system.
  - Major information flows between the entities and the system.
- Note: only one process symbol, and no data stores shown

Eg:

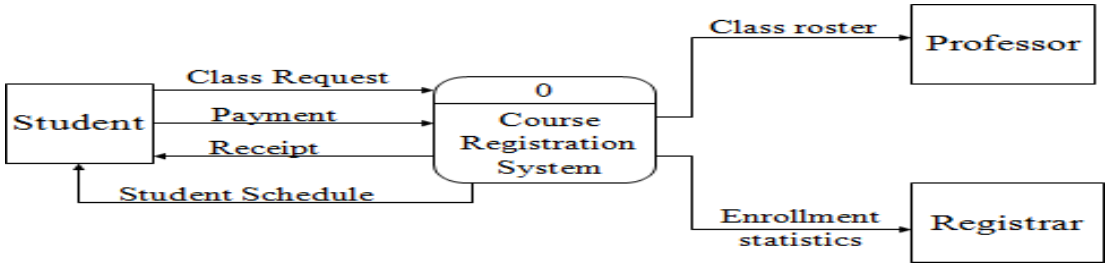


Fig: Context level DFD for course registration system

- **Level-1 diagram** is a data flow diagram that represents a system's major processes, data flows, and data stores at a high level of detail.

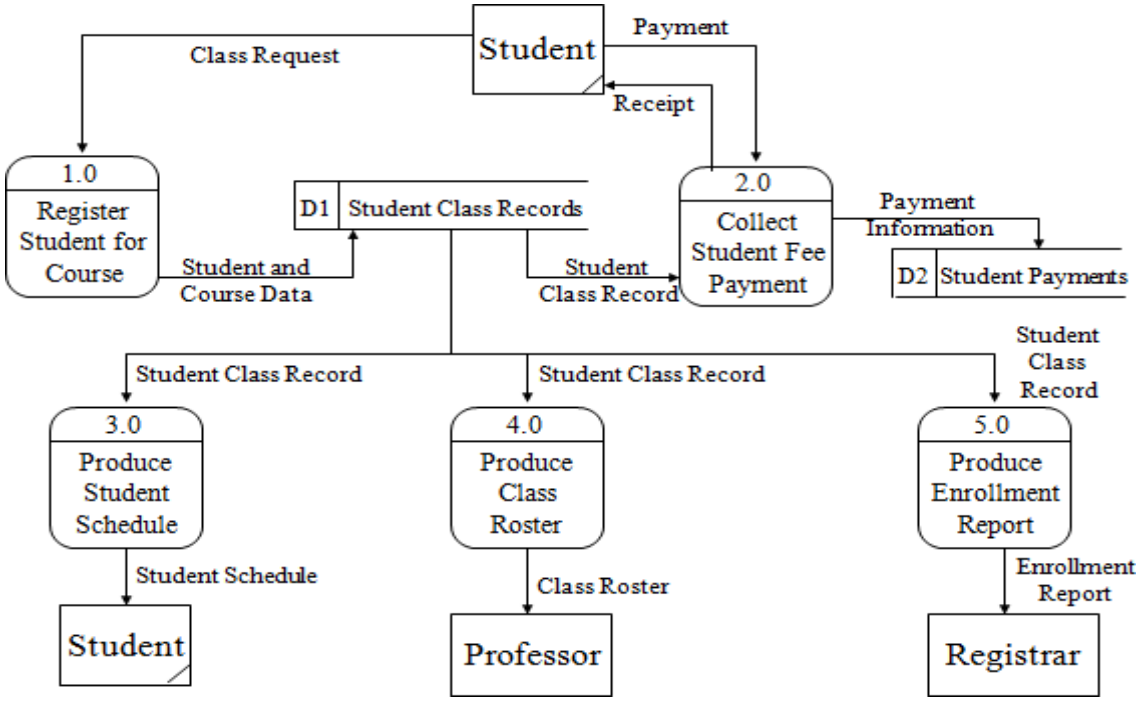


Fig: Level – 1 DFD

- **Level-n Diagram** shows how the system is divided into sub-systems (processes), each of which deals with one or more of the data flows to or from an external agent, and which together provide all of the functionality of the system as a whole.
- **Eg:**

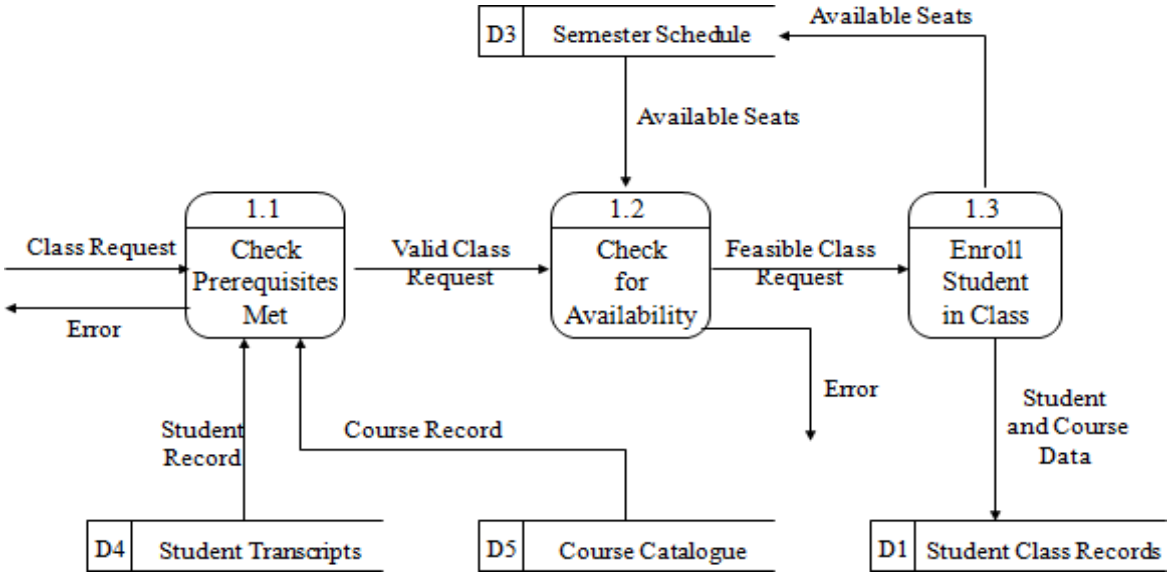


Fig: **level 2 DFD for Process 1: Register student for the course**

Class Diagram

- A class is a collection of objects with common structure, common behavior, common relationships and common semantics
- Classes are found by examining the objects in sequence and collaboration diagram
- A class is drawn as a rectangle with three compartments
- A class diagram shows the existence of classes and their relationships in the logical view of a system
- UML modeling elements in class diagrams
    - Classes and their structure and behavior
    - Association, aggregation, dependency, and inheritance relationships
    - Multiplicity and navigation indicators
    - Role names

- Eg: ATM system

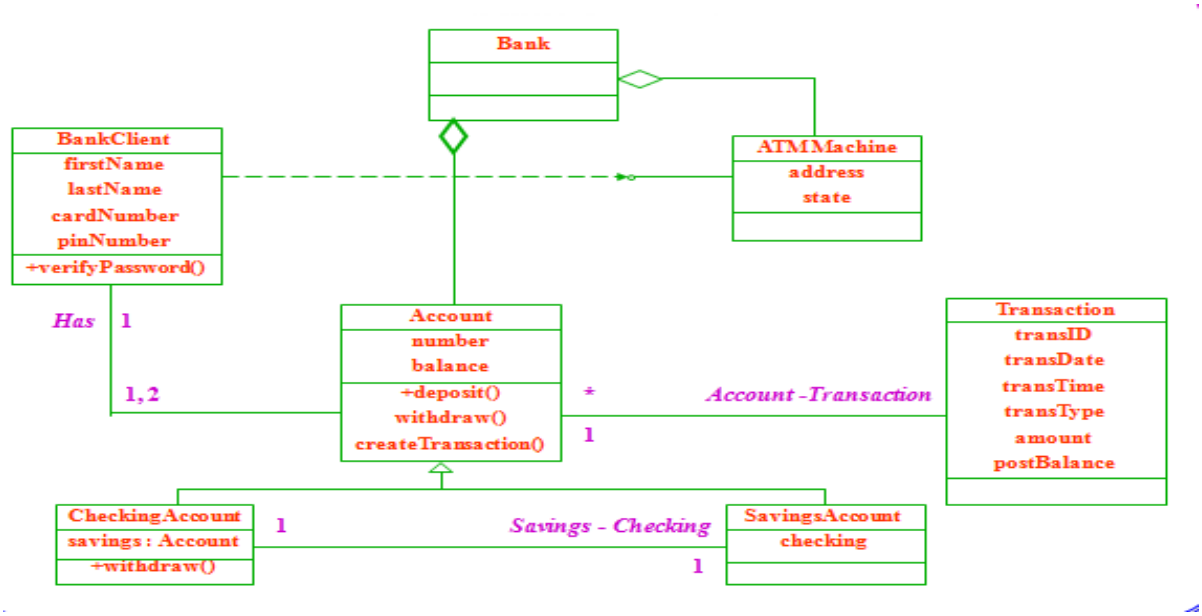


Fig: Class diagram for ATM System

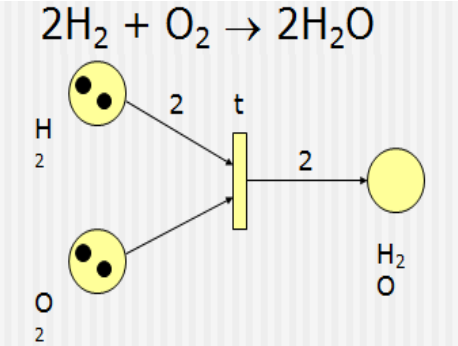
2.5.2 PETRI NETS

- A **Petri net** is also known as a **place/transition net**
- It is a diagrammatic tool to model concurrency and synchronization in distributed systems.
- It is similar to State Transition Diagrams.
- This is used as a visual communication aid to model the system behaviour.
- This is a mathematical modeling language for the description of distributed systems
- Petri nets can be used to model complex processes
- Petri nets can be simulated in order to illustrate and test system behaviour, benchmark its speed etc.

Components:

- **Places** (circles): Places represent possible states of the system;
- **Transitions** (rectangles): events or actions which cause the change of state;
- **Arcs** (arrows): Used to connect a place with a transition or a transition with a place.
- Change of state is denoted by a movement of ***tokens*** (black dots) from place to place and is caused by the ***firing*** of a transition.
- The firing represents an occurrence of the event or an action taken.
- The firing is subject to the input conditions, denoted by token availability.

- A transition is *firable* or *enabled* when there are sufficient tokens in its input places.
- After firing, tokens will be transferred from the input places (old state) to the output places, denoting the new state.
- Eg: firing event



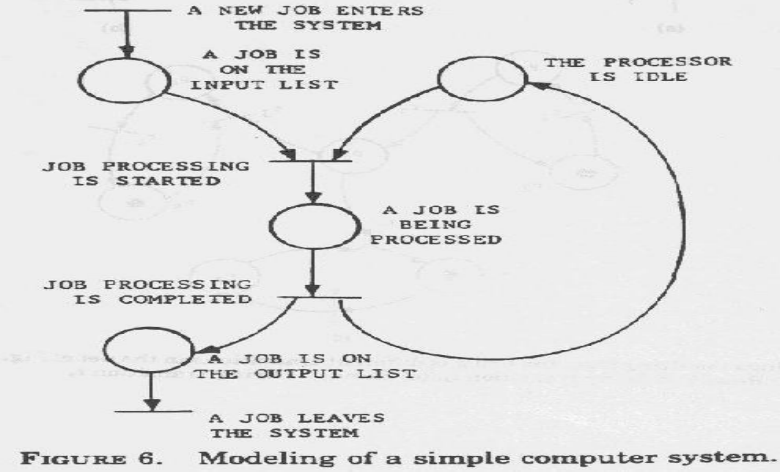
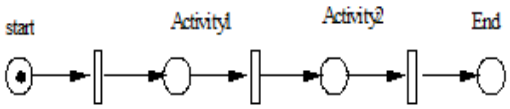


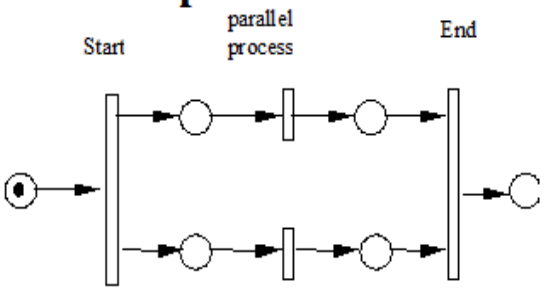
FIGURE 6. Modeling of a simple computer system.

Primitive structures that occur in real systems:

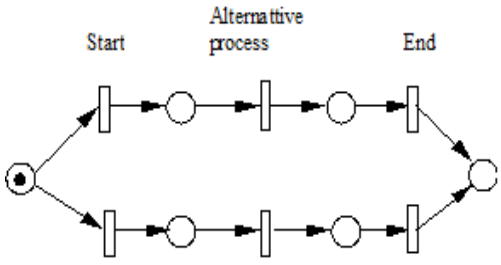
Precedence relation:



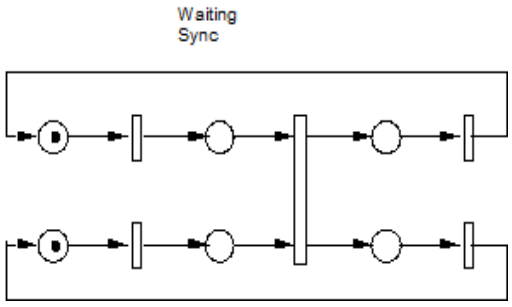
Parallel processes:



Alternative processes:



Synchronization:



2.3 DATA DICTIONARIES

- Data dictionaries are lists of all of the names used in the system models. Descriptions of the entities, relationships and attributes are also included
- Collects and coordinates data terms, and confirms what each term means to different people in the organization
- Many CASE workbenches support data dictionaries
- The data dictionary may be used for the following reasons:
    - Provide documentation
    - Eliminate redundancy
    - Validate the data flow diagram
    - Provide a starting point for developing screens and reports
    - To develop the logic for DFD processes
- Data dictionaries contain
    - Data flow
    - Data structures
    - Elements
    - Data stores

Defining Data Flow

- Each data flow should be defined with descriptive information and it's composite structure or elements

Eg:

Name	Customer Order
Description	Contains customer order information and is used to update the customer master and item files and to produce an order record.
Source	Customer External Entity
Destination	Process 1, Add Customer Order
Type	Screen
Data Structure	Order Information
Volume/Time	10/hour
Comments	An order record contains information for one customer order. The order may be received by mail, fax, or by telephone.

Fig: Data flow definition for customer order

- Include the following information:
    - ID - identification number

- Label, the text that should appear on the diagram
- A general description of the data flow
- The source and destination of the data flow
- Type of data flow
- The name of the data structure describing the elements
- The volume per unit time
- An area for further comments and notations

Defining Data Structures

- Data structures are a group of smaller structures and elements
- An algebraic notation is used to represent the data structure
- The symbols used are
  - Equal sign, meaning “consists of”
  - Plus sign, meaning "and”
  - Braces {} meaning repetitive elements, a repeating element or group of elements
  - Brackets [] for an either/or situation- The elements listed inside are mutually exclusive
  - Parentheses () for an optional element
- A repeating group may be
  - A sub-form
  - A screen or form table
  - A program table, matrix, or array
- There may be one repeating element or several within the group
- Data structure may be logical or physical data structure
  - **Logical:** Show what data the business needs for its day-to-day operations
  - **Physical**: Include additional elements necessary for implementing the system

Eg: Data structure for customer details

Customer name = First name + (middle name) + last name

Address= Street + (Apartment) + City + State +Zip

Data Element

Data element includes the following:

- Element ID

- The name of the element
- Aliases -Synonyms or other names for the element
- A short description of the element
    - Element is base or derived
        - A base element is one that has been initially keyed into the system
        - A derived element is one that is created by a process, usually as the result of a calculation or a series of decision-making statements
- Element length
- Type of data - Alphanumeric or text data
- Input and output formats - using coding symbols:
    - Z - Zero suppress
    - 9 - Number
    - X - Character
    - X(8) - 8 characters
    - . , - Comma, decimal point, hyphen
- Validation criteria - Ensure that accurate data are captured by the system
- Default value - Include any default value the element may have. The default value is displayed on entry screens
- An additional comment or remark area

Data Stores

- Data stores are created for each different data entity being stored
- When data flow base elements are grouped together to form a structural record, a data store is created for each unique structural record
- Because a given data flow may only show part of the collective data that a structural record contains, many different data flow structures may need to be examined to arrive at a complete data store description
- A data store has:
    - The data store ID
    - The data store name
    - An alias for the table
    - A short description of the data store

- The file type
- File format

Eg:

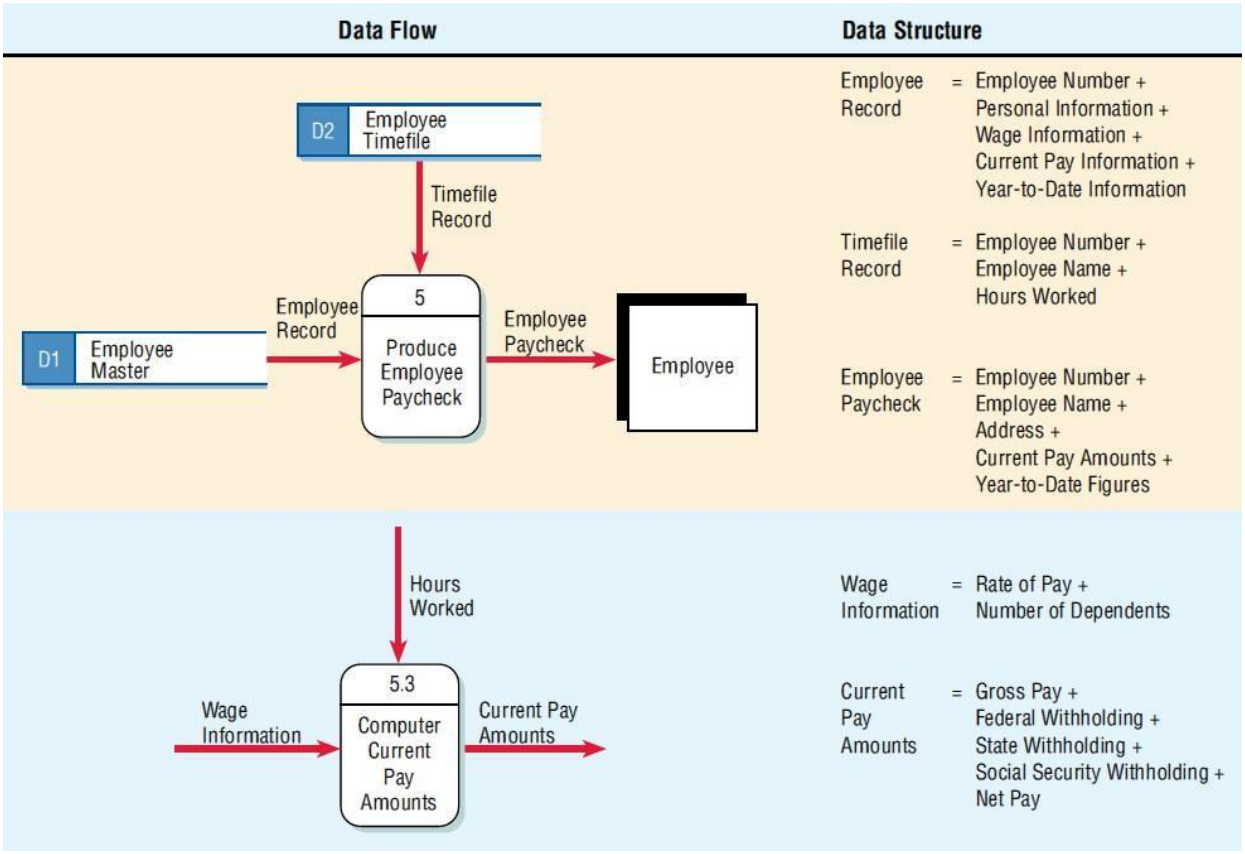


Fig: Sample data dictionary for employee database