CHAPTER 6

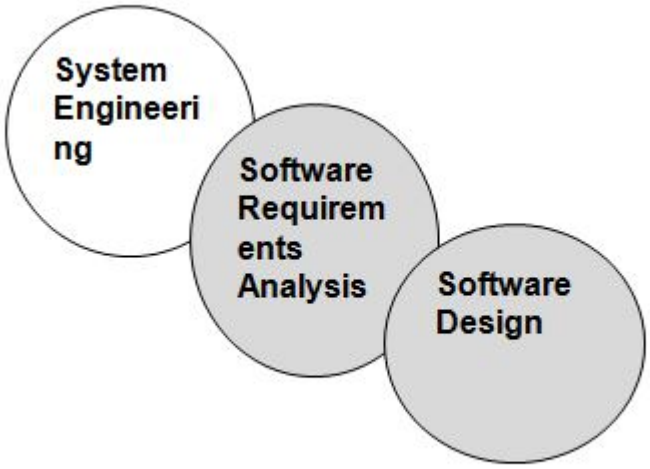
ANALYSIS PRINCIPLES AND CONCEPTS

- Software requirement analysis is a process of discovery, refinement, modeling and specification.
- Software **requirements analysis** is necessary to avoid creating a software product that fails to meet the customer's needs.
- Data, functional, and behavioral requirements are elicited from the customer and refined to create a specification that can be used to design the system.
- **Software requirements** work products must be reviewed for clarity, completeness, and consistency.
- Requirement Engineering is the systematic use of proven principles, techniques, language and tools for the cost and on-going evolution of user needs and the specification of the external behavior of a system to satisfy those user needs.

❖ REQUIREMENT ANALYSIS

- The set of activities to achieve the objective of software requirement engineering is often called analysis.
- Requirement analysis is a software engineering task that acts as a inter link or bridge between **system level requirements** engineering and **software design.**
- It provides software designer with a representation of system information, function, and behavior that can be translated to data, architectural, and component-level designs.
- Expect to do a little bit of design during analysis and a little bit of analysis during design.
- Requirement analysis helps in refinements of the software allocation and build models of data, function, behavioral and structural domains treated software.
- It also provides the software designer with representation of information, function and behavior. These entities are translated into data, architectural, interface and components level design.
- Finally quality in software is assessed from the phase of requirement analysis.
- Software Requirements Analysis Phases:
  - Problem recognition
  - Evaluation and synthesis (focus is on what not how)
  - Modeling
  - Specification
  - Review

- Requirement engineering activities results in:
  - Specification of software's operational characteristics like function, data and behavior.
  - Specification of software's interface with other system elements.
  - Specification of standards constraints that software must meet.



Analysis s as a bridge between system engineering and Software Design

❖ SOFTWARE REQUIREMENT ELICITATION

In requirements engineering, **requirements elicitation** is the practice of collecting the requirements of a system from users, customers and other stakeholders.[1] The practice is also sometimes referred to as "**requirement gathering**".

Customer meetings, interviews and seminars are the most commonly used technique. It **use context free** questions to find out customer's goals and benefits, identify stakeholders, gain understanding of problem, determine customer reactions to proposed solutions, and assess meeting effectiveness.

Example:

1. Who is behind the request for this work?
2. Who will use the solution?
3. What will be the economic benefit of a successful solution?
4. Is there another source for the solution that you need?

If many users are involved, be certain that a representative cross section of users is interviewed.

- Facilitated Action Specification Techniques (FAST):
  This is approach or method that encourages the creation of a joint team of customers and developers who work together to identify the problem, propose elements of the solution, negotiate different approaches and specify a preliminary set of solution requirements.
  Guidelines:
  - Meeting held at neutral site, attended by both software engineers and customers.
  - Rules established for preparation and participation.
  - Agenda suggested to cover important points and to allow for brainstorming.
  - Meeting controlled by facilitator (customer, developer, or outsider).
  - Definition mechanism (flip charts, stickers, electronic device, etc.) is used.
  - Goal is to identify problem, propose elements of solution, negotiate different approaches, and specify a preliminary set of solution requirement.

- QULAITY FUNCTION DEPLOYMENT (QFD):

  Quality Function Deployment (QFD) is a structured approach to defining customer needs or requirements and translating them into specific plans to produce products to meet those needs. The "voice of the customer" is the term to describe these stated and unstated customer needs or requirements. The voice of the customer is captured in a variety of ways: direct discussion or interviews, surveys, focus groups, customer specifications, observation, warranty data, field reports, etc. This understanding of the customer needs is then summarized in a product planning matrix or "house of quality". These matrices are used to translate higher level "what's" or needs into lower level "how's" – product requirements or technical characteristics to satisfy these needs.
  It translates customer needs into technical software requirements and uses customer interviews, observation, surveys, and historical data for requirements gathering. It consits of Customer voice table (contains summary of requirements).
  QFD is oriented toward involving a team of people representing the various functional departments that have involvement in product development: Marketing, Design Engineering, Quality Assurance, Manufacturing/ Manufacturing Engineering, Test Engineering, Finance, Product Support, etc.

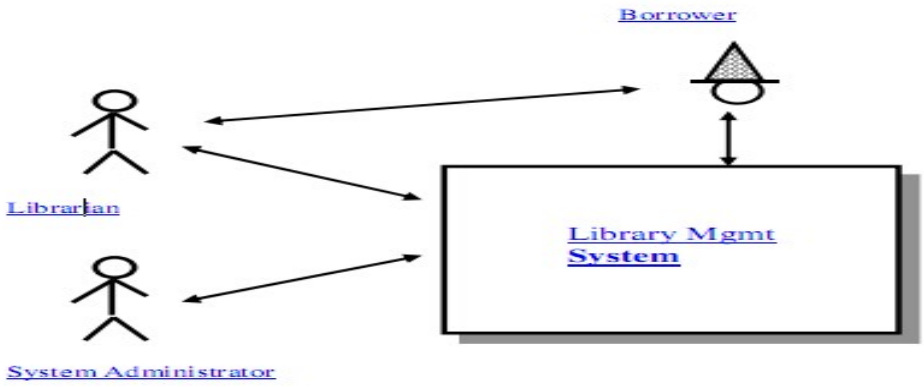
❖ USE CASE:

- Scenarios that describe how the product will be used in specific situations.
- Written narratives that describe the role of an actor (user or device) as interaction with the system occurs.
- Use-cases are designed from the actor's point of view.
- Example:

  Library Management System

  Consider a **library database** with the following transactions:

  1. Check out a copy of a book / Return a copy of a book.
  2. Add a copy of a book to / Remove a copy of a book from the library.
  3. Get the list of books by a particular author or in a particular subject area;
  4. Find out the list of books currently checked out by a particular borrower;
  5. Find out what borrower last checked out a particular copy of a book.
- Not all actors can be identified during the first iteration of requirements elicitation, but it is important to identify the primary actors before developing the use-cases.



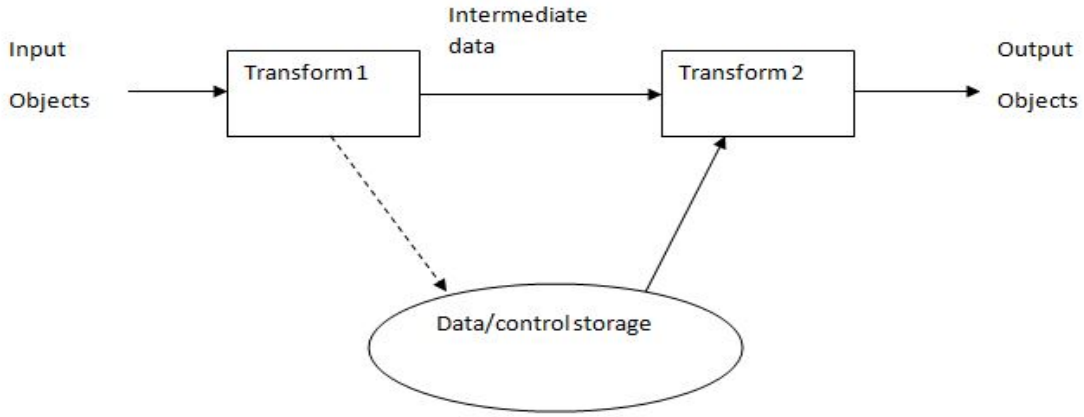
❖ ANALYSIS PRINCIPLES:

- There are many methods of analysizing the requirement. These pronciple or focus driven methods for analysis of requirements are analysis principles.
- Eventhough different, all the analysis priciples are guided by same set of operational priciples are:
  - Representation and understanding the problem information.
  - Objective and functions that software need to fulfil.
  - Behaviour of the software is to be represented.

- The models should be partitioned in such a way that inner details that are layered should be uncovered.
- The analysis should consider implementation of the information as requirement.
- Davis gave additional 6 guiding principles:

1. Understand the problem before you begin the analysis model.
2. Develop a prototype that helps user to understand human-computer interaction.
3. Record the origin of reason for every requirement.
4. Use multiple views of requirements.
5. Rank requirements.
6. Work to eliminate ambiguity.

1. INFORMATION DOMAIN:
   - Encompasses all data objects that contain numbers, text, images, audio, or video.
   - Information content or data model (shows the relationships among the data and control objects that make up the system)
   - Information flow (represents the manner in which data and control objects change as each moves through the system)
   - Information domain consists of three views of data and control.
     1. Information content and relationship
     2. Information flow
     3. Information structure(representations of the internal organizations of various data and control items)



2. MODELLING:
   - To get better understanding of the working of s/w and its requirements models are to be developed.

- In s/w development representation of the information transforms, the functions and the behavior of the system transformation takes place is modeling.
- Analysis principle focuses of modeling of function and behavior.
- During requirement analysis, following modeling or models are created:
    a. Data Modeling
    b. Functional modeling and information flow
    c. Behavioral modeling

IMPORTANCE OF MODELLING:

1) It helps in understanding information, function and behavior making the requirements analysis task easier and more systematic.
2) Helps in review and determination of the completeness, consistency and accuracy of specification.
3) Acts as foundation for design, helping designer to implement the specification of requirements.

❖ DATA MODELLING:

Data modeling answers a set of specific questions that are relevant to any data processing application. Example:

- What are the primary data objects to be processed by the system?
- What is the composition of each data object and what attributes describe the object?
- What are the relationships between each objects and other objects?

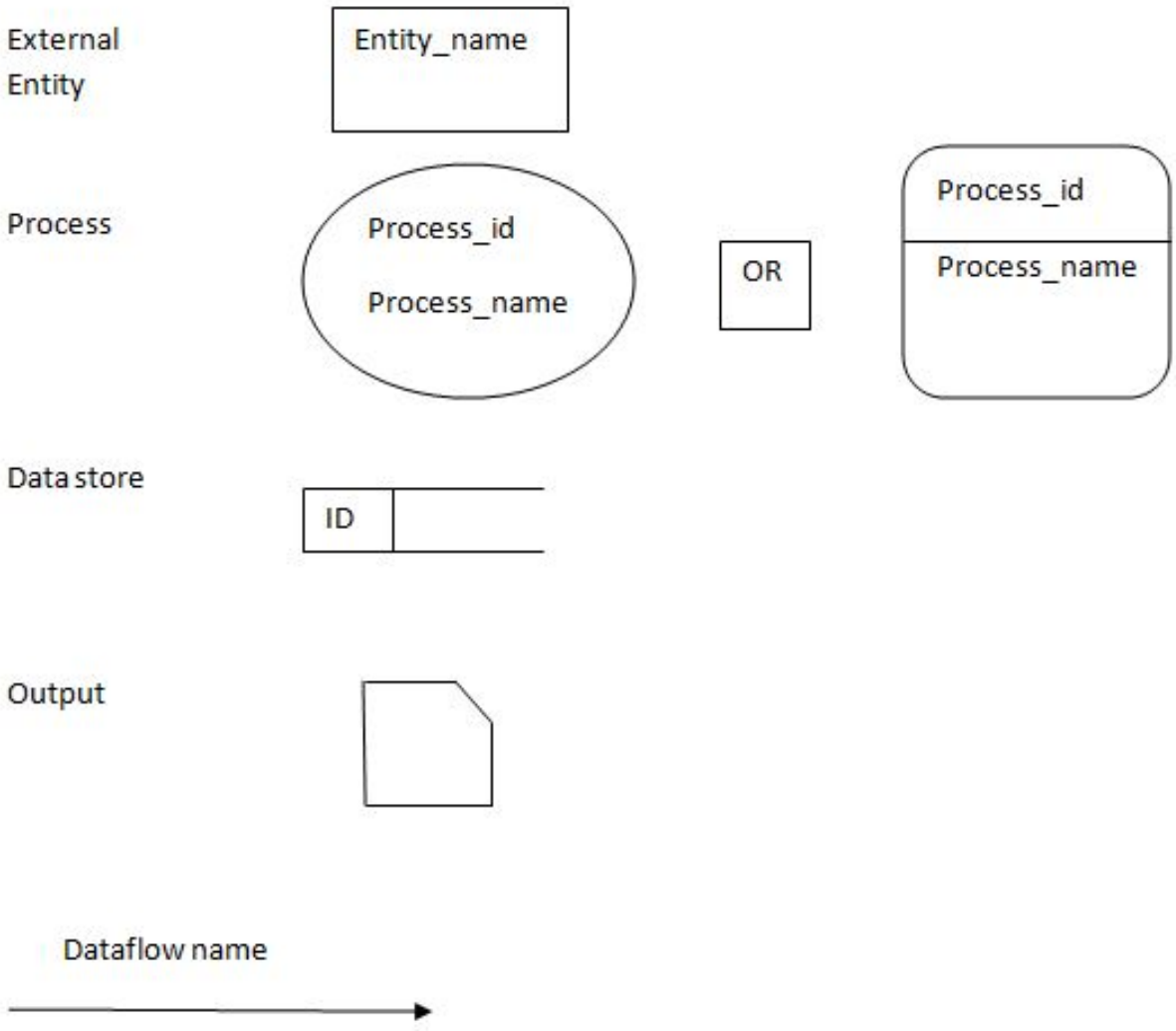
To answer above questions, data modeling methods make use of the entity relationship diagram. The Entity Relationship Diagram (ERD) enables a software engineer to identify data objects and the relationships using a notation. The ERD solely focuses on data.
The data model consists of **3 interrelated** pieces of information: **data object,** the **attributes** that describes the data object and **the relationships** that connect data objects to one another.

❖ FUNCTIONAL MODELLING:

- Software transforms information and in order to accomplish this, it must perform at least three functions: input, processing and output.
- When functional models of an application are created, the software engineers focuses on problem specific functions. Information flow can be represented by a DFD (Data flow diagram).

- DFDs can be used not only to model information-processing systems, but also as a way of modeling whole organizations, that is, as a tool for business planning and strategic planning.
- Data flow diagrams illustrate how data is processed by a system in terms of inputs and outputs.
- Context diagram or level 0:
  Most abstract data flow representation of a system. It represents the entire system as a single bubble. Bubble is labeled as main function or objective of the system. Various external entities with which the system interacts are also represented.

- Level 1 DFD:
  Examine the high level functional requirement. Generally 5 to 7 functions or process are used to decompose the context diagram. If more functions are to be defined then related one should be combined as one and again be decomposed in next level diagram and so on. Id_ name is done using special convention. At 0 level the process if is 0. In next level 1,2,…….n or 0.1,0.2,….., and so on in preceding level.



❖ BEHAVIOURAL MODELLING:

It is an operational principle for all requirements analysis methods. The state transition diagram represents the behavior of the system by depicting its state and the events that cause the system to change state.

A state transition diagram indicates how the system moves from one state to another. Most s/w responds to events from outside world. This stimulus/response characteristic forms the base for behavioral model. A computer always exists in some externally observable mode of behavior that is changed only when some events occurs called state. A behavioral model creates a representation of the states of the software and the events that cause an s/w to change.

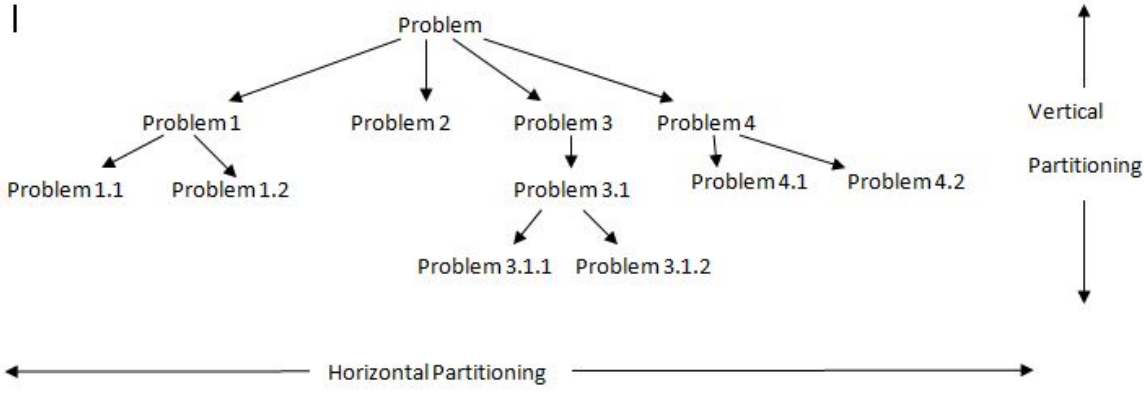
3. PARTITIONING

Partitioning is the process that results in the elaboration of data, function, or behavior. It is also defined a process of decomposition of a problem into its constituent's parts. Partitioning may be done in two ways.

a. Horizontal partitioning
b. Vertical partitioning

a. HORIZONTAL PARTITIONING:
Horizontal partitioning is a breadth-first decomposition of the system function, behavior, or information, one level at a time.

b. VERTICAL PARTITIONING:
Vertical partitioning is a depth-first elaboration of the system function, behavior, or information, one subsystem at a time.



❖ SOFTWARE PROTOTYPING

For any software development paradigm analysis is to be done, but the form may be different. Operational analysis principles may be applied and different models may be derived or requirement may be gathered, analysed and then models may be derived or built for customer and developer assessment called prototyping.

Prototyping may be:

- Close ended or throwaway prototyping – A prototype serves as a rough demonstration of requirements. It is then discarded and the s/w is engineered using a different paradigm.
- Open ended or evolutionary prototyping - Uses the prototype as the first part of an analysis activity that will be continued into design and construction.
- Customers are generally actively involved in development for getting feedback for the improvement of the s/w in prototyping.

For effective prototyping, a prototype must be developed rapidly so that customer may assess results and recommend changes. For rapid prototyping, three generic classes of methods and tools are available as follows:

- Fourth Generation Techniques (4GT) – 4GT encompasses a broad array of database query and reporting languages, program and application generators and other very high level non procedural langugaes. 4GT enables engineeers to generate executable code quickly hence ideal for rapid prototyping.
- Reusable s/w components- Another approach focuses on use of existing components to built s/w. Entire or improved components or even off-shelf components may be used to develop rapid prototyped s/w.
- Formal Specification and prototyping enviroments: Replacing natural langugages specification techniques, many formal specification and prototyping langugages and tools have been developed. The developers are even improving these tools inorder to develop interactive enviroments. These environments will enable an analyst to interactively create langugae-based applications of system or s/w and invoke automated tools that translates the language- base specification into executable code. It would also enable customer to use the prototype executable code to refine formal requirements.

❖ SPECIFICATION

Specification affects quality. Incomplete, inconsistent and misleading specification creates frustration and confused developer. There has been multiple principles for specification Balzer and Goodman propsed:

1. Separate functionality from implementation.
2. Develop behavioural model showing data and functional reponses of a systemto various stimuli from the enviroments.

3. Establish the context in which s/w operates by specifying the manner of others.
4. Define the enviroment where system operates and how interwined collection of agents react to stimuli in this environment.
5. Create a congnitive model rather than design or implementation model. Congnitive model focuses on how system is perceived by the environment.
6. Specification must be complete and detailed.
7. Establish the content and structure of a specification in a way that will enable it to be amenable to change.

These principles are the base for specification or representation of s/w requirements. Now these specification are to be represented in paper. While representing few guidelines are to be followed as:

- Representation format and content must be problen relevant and specific.
- Information within the specification must be nested or layered depending upon details.
- Diagrams and other notational forms should be restricted in number and comsistent in use.
- Reperesenatation should be revisable.

❖ SPECIFICATION REVIEW

A review of the SRS is conducted by both the s/w developer and the customer. Review should be done carefully as specification forms are the foundation of s/w development. Review is started by examining completeness, consistency and accuracy of overall information functional and behavioural domain. Review is continued in more details in every domain vertically. Once review is completed SRS is signed off and transformed into contract.

Post requiremental changes by customer may change the cost which is to be considered later. During reviews SRS may face changes and it should be considered and addressed positively. Case tool helps in reviewing and changing.

❖ ANALYSIS MODELLING

Software engineering begins with a series of modelling tasks that leads to a complete specification of requirements and a comprehensive design representation for the s/w to built. The analysis model is the first technical representatio of a system. It is a set of different models. Itbasically focuses on data and flow modelling.Over the time many models have been developed for the analysis but two most used models are:

1. Structured analysis
2. Object oriented analysis

3. Data structure system development (DSSD)
4. Jackson system development (JSD)
5. Structured analysis and design techniques (SADT)

- THE ELEMENT OF THE ANALYSIS MODEL

  Primary objective of analysis model are:

  1. To describe what the customer requires
  2. To establish a basic for the creation of a software design
  3. To define a set of requirements that can be validated once the s/w is built

  For achievement of these objectives, the structure of the analysis model must be as shown in the figure below:

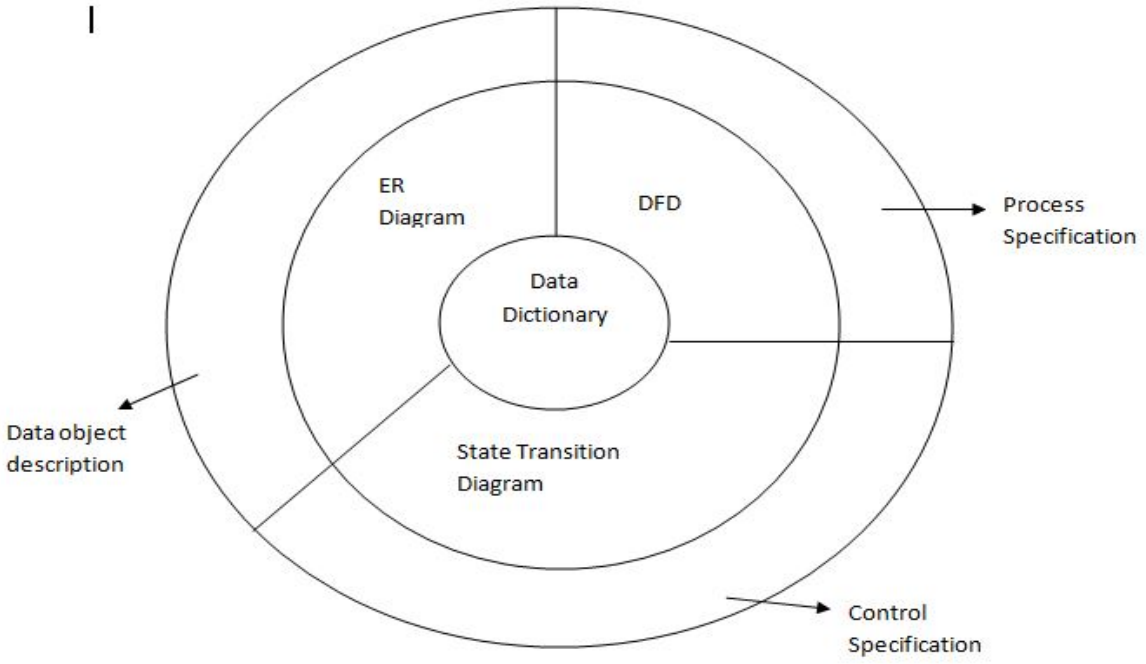


Fig: The structure of Analysis Model

1. Data Dictionary: A repository that contains description of all data objects consumed or produced by the s/w
2. ERD: It shows the relationship between data objects.
3. DFD: Provides an indication of how data are transformed as they move through the system to depict the functions that are transformed the data flow.
4. STD: Indicates how the system behaves as a consequence of external events.

Elements of the analysis model may be scenario based elements, flow oriented elements, class based elements and behavioral elements. The elements are the views of representation to depict the analysis model.

Scenario based elements is the described from user's point of view using scenario based approach. Use case text, Use case diagram, Activity diagram are used for scenario based representation of the system.

Class based elements uses scenario implies a set of objects that are manipulated as an actor interacting with the system. These objects are categorized into classes. Class diagrams, analysis packages, CRC model, collaboration diagram are used for class based representation of the system.

Behavioral elements or state transformations are based in behavioral elements. Analysis model must provide modeling elements that's depicts behavior. State and sequence diagrams are used for representation of behavioral elements in the system.

Flow oriented elements are defined as information and control flow as the system is realized. These data, information and control elements are flow elements. DFD, CFDS and Processing narratives are used to represent flow elements of the system.

❖ STRUCTURED ANALYSIS

It is one of the classical modeling methods and most used modeling technique. Structured analysis considers data and the processes that transform the data as separate entities. Data objects are modeled in a way that defines their attributes and relationships. Processes that manipulate data objects are modeled in a manner that shows hoe they transform data as data objects flow through the system. Structured analysis used different graphical symbols/ notations to model the system.

- DATA DICTIONARY

A data dictionary is an organized listing of all data elements that are pertinent to the system, with precise, rigorous definitions so that user and system analyst will have a common understanding of inputs, outputs, and processing. Most of the data dictionary contains following information;

i. ***Name-*** the primary name of the data or control item or external entity.
ii. ***Alias*** – other name used for the entity.
iii. ***Where-used/how–used*** – a listing of the processes that use the data or control item and how it is used (e.g. input to the process, output from the process, as an external entity.
iv. ***Content description*** – a notation for representing content.
v. ***Supplementary information-*** other information or restrictions or limitations.

Example:

Name:

Aliases:

Where-used/how-used:

Description:

Telephone number = [local number| long distance number]
Local number = prefix + access number
Long distance number = 1 + area code + local number
Area code = [800 | 888 | 768]
Prefix = "a three digit number that never starts with 0 or 1"
Access number = "any six number string"