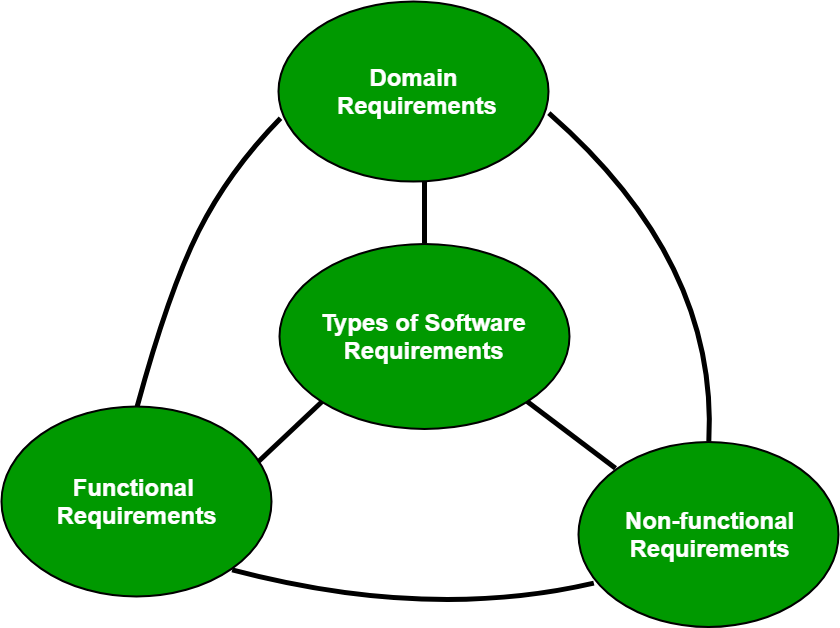
Software Requirements:

The software requirements are description of features and functionalities of the target system. Requirements convey the expectations of users from the software product. The requirements can be obvious or hidden, known or unknown, expected or unexpected from client’s point of view.

**Software requirement can be of 3 types:**

* Functional requirements
* Non-functional requirements
* Domain requirements



**Functional Requirements:** These are the requirements that the end user specifically demands as basic facilities that the system should offer. It can be a calculation, data manipulation, business process, user interaction, or any other specific functionality which defines what function a system is likely to perform. These are represented or stated in the form of input to be given to the system, the operation performed and the output expected. They are basically the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements.

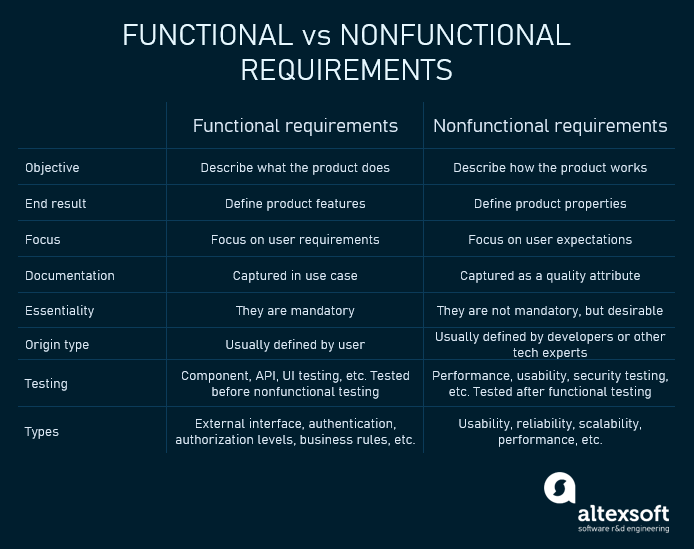
For example, in a hospital management system, a doctor should be able to retrieve the information of his patients.

**Non-functional requirements:** These are basically the quality constraints that the system must satisfy according to the project contract. Nonfunctional requirements, not related to the system functionality, rather define how the system should perform. The priority or extent to which these factors are implemented varies from one project to other. They are also called non-behavioral requirements. They basically deal with issues like:

* Portability
* Security
* Maintainability
* Reliability
* Scalability
* Performance
* Reusability
* Flexibility

**Domain requirements:** Domain requirements are the requirements which are characteristic of a particular category or domain of projects. Domain requirements can be functional or nonfunctional.

For instance, in academic software that maintains records of a school or college, the functionality of being able to access the list of faculty and list of students of each grade is a domain requirement. These requirements are therefore identified from that domain model and are not user specific.



**Advantages of classifying software requirements include:**

1. **Better organization:** Classifying software requirements helps organize them into groups that are easier to manage, prioritize, and track throughout the development process.
2. **Improved communication:** Clear classification of requirements makes it easier to communicate them to stakeholders, developers, and other team members. It also ensures that everyone is on the same page about what is required.
3. **Increased quality:** By classifying requirements, potential conflicts or gaps can be identified early in the development process. This reduces the risk of errors, omissions, or misunderstandings, leading to higher quality software.
4. **Improved traceability:** Classifying requirements helps establish traceability, which is essential for demonstrating compliance with regulatory or quality standards.

**Disadvantages of classifying software requirements include:**

1. **Complexity**: Classifying software requirements can be complex, especially if there are many stakeholders with different needs or requirements. It can also be time-consuming to identify and classify all the requirements.
2. **Rigid structure**: A rigid classification structure may limit the ability to accommodate changes or evolving needs during the development process. It can also lead to a siloed approach that prevents the integration of new ideas or insights.
3. **Misclassification**: Misclassifying requirements can lead to errors or misunderstandings that can be costly to correct later in the development process.

**User Requirements:**

User requirements reflect the specific needs or expectations of the software's customers

These requirements describe what the end-user wants from the software system. User requirements are usually expressed in natural language and are typically gathered through interviews, surveys, or user feedback.

**System requirements:**

These requirements specify the technical characteristics of the software system, such as its architecture, hardware requirements, software components, and interfaces. System requirements are typically expressed in technical terms and are often used as a basis for system design.

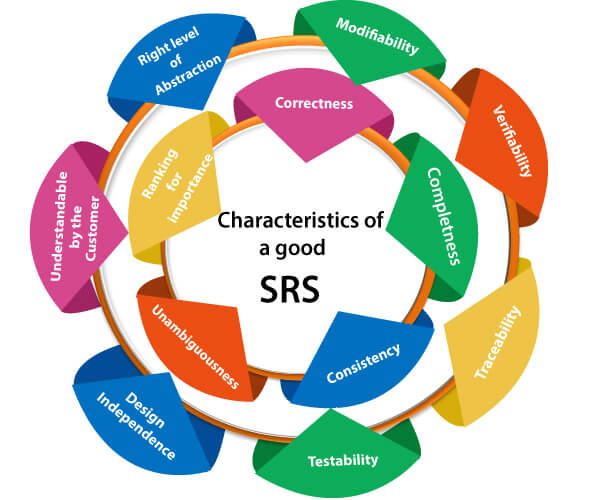
**Software Requirement Document**

A software requirements specification (SRS) is a document that describes what the software will do and how it will be expected to perform. It also describes the functionality the product needs to fulfill the needs of all stakeholders (business, users)

SRS is developed which describes requirements of software that may include changes and modifications that is needed to be done to increase quality of product and to satisfy customer’s demand.

1. **Introduction :**
   * **(i) Purpose of this Document –** At first, main aim of why this document is necessary and what’s purpose of document is explained and described.
   * **(ii) Scope of this document –** In this, overall working and main objective of document and what value it will provide to customer is described and explained. It also includes a description of development cost and time required.
   * **(iii) Overview –** In this, description of product is explained. It’s simply summary or overall review of product.
2. **General description :** In this, general functions of product which includes objective of user, a user characteristic, features, benefits, about why its importance is mentioned. It also describes features of user community.
3. **Functional Requirements:** In this, possible outcome of software system which includes effects due to operation of program is fully explained. All functional requirements which may include calculations, data processing, etc. are placed in a ranked order.
4. **Interface Requirements:** In this, software interfaces which mean how software program communicates with each other or users either in form of any language, code, or message are fully described and explained. Examples can be shared memory, data streams, etc.
5. **Performance Requirements:** In this, how a software system performs desired functions under specific condition is explained. It also explains required time, required memory, maximum error rate, etc.
6. **Design Constraints:** In this, constraints which simply means limitation or restriction are specified and explained for design team. Examples may include use of a particular algorithm, hardware and software limitations, etc.
7. **Non-Functional Attributes:** In this, non-functional attributes are explained that are required by software system for better performance. An example may include Security, Portability, Reliability, Reusability, Application compatibility, Data integrity, Scalability capacity, etc.
8. **Preliminary Schedule and Budget:** In this, initial version and budget of project plan are explained which include overall time duration required and overall cost required for development of project.
9. **Appendices:** In this, additional information like references from where information is gathered, definitions of some specific terms, acronyms, abbreviations, etc. are given and explained.

Characteristics of good SRS



**Uses of SRS document:**

1. Development team requires it for developing product according to the need.
2. Test plans are generated by testing group based on the describe external behavior.
3. Maintenance and support staff needs it to understand what the software product is supposed to do.
4. Project manager base their plans and estimates of schedule, effort and resources on it.
5. Customers rely on it to know that product they can expect.
6. As a contract between developer and customer.
7. In documentation purpose.

# Requirements Engineering Process

**Requirements engineering (RE)** refers to the process of defining, documenting, and maintaining requirements in the engineering design process. Requirement engineering provides the appropriate mechanism to understand what the customer desires, analyzing the need, and assessing feasibility, negotiating a reasonable solution, specifying the solution clearly, validating the specifications and managing the requirements as they are transformed into a working system. Thus, requirement engineering is the disciplined application of proven principles, methods, tools, and notation to describe a proposed system's intended behavior and its associated constraints.

## Requirement Engineering Process

It is a four-step process, which includes -

1. Feasibility Study
2. Requirement Elicitation and Analysis
3. Software Requirement Specification
4. Software Requirement Validation
5. Software Requirement Management



### 1. Feasibility Study:

The objective behind the feasibility study is to create the reasons for developing the software that is acceptable to users, flexible to change and conformable to established standards.

**Types of Feasibility:**

1. **Technical Feasibility** - Technical feasibility evaluates the current technologies, which are needed to accomplish customer requirements within the time and budget.
2. **Operational Feasibility** - Operational feasibility assesses the range in which the required software performs a series of levels to solve business problems and customer requirements.
3. **Economic Feasibility** - Economic feasibility decides whether the necessary software can generate financial profits for an organization.

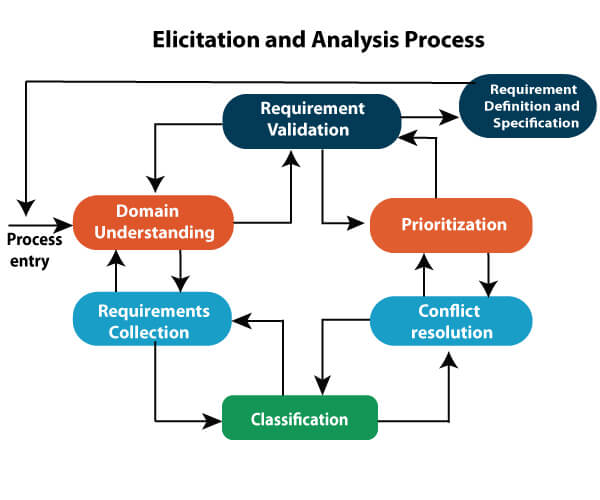
### 2. Requirement Elicitation and Analysis:

This is also known as the **gathering of requirements**. Here, requirements are identified with the help of customers and existing systems processes, if available.

Analysis of requirements starts with requirement elicitation. The requirements are analyzed to identify inconsistencies, defects, omission, etc. We describe requirements in terms of relationships and also resolve conflicts if any.

**Problems of Elicitation and Analysis**

* Getting all, and only, the right people involved.
* Stakeholders often don't know what they want
* Stakeholders express requirements in their terms.
* Stakeholders may have conflicting requirements.
* Requirement change during the analysis process.
* Organizational and political factors may influence system requirements.



### 3. Software Requirement Specification:

Software requirement specification is a kind of document which is created by a software analyst after the requirements collected from the various sources - the requirement received by the customer written in ordinary language. It is the job of the analyst to write the requirement in technical language so that they can be understood and beneficial by the development team.

The models used at this stage include ER diagrams, data flow diagrams (DFDs), function decomposition diagrams (FDDs), data dictionaries, etc.

* **Data Flow Diagrams:** Data Flow Diagrams (DFDs) are used widely for modeling the requirements. DFD shows the flow of data through a system. The system may be a company, an organization, a set of procedures, a computer hardware system, a software system, or any combination of the preceding. The DFD is also known as a data flow graph or bubble chart.
* **Data Dictionaries:** Data Dictionaries are simply repositories to store information about all data items defined in DFDs. At the requirements stage, the data dictionary should at least define customer data items, to ensure that the customer and developers use the same definition and terminologies.
* **Entity-Relationship Diagrams:** Another tool for requirement specification is the entity-relationship diagram, often called an "**E-R diagram**." It is a detailed logical representation of the data for the organization and uses three main constructs i.e. data entities, relationships, and their associated attributes.

### 4. Software Requirement Validation:

After requirement specifications developed, the requirements discussed in this document are validated. The user might demand illegal, impossible solution or experts may misinterpret the needs. Requirements can be the check against the following conditions -

* If they can practically implement
* If they are correct and as per the functionality and specially of software
* If there are any ambiguities
* If they are full
* If they can describe

**Requirements Validation Techniques**

* **Requirements reviews/inspections:** systematic manual analysis of the requirements.
* **Prototyping:** Using an executable model of the system to check requirements.
* **Test-case generation:** Developing tests for requirements to check testability.
* **Automated consistency analysis:** checking for the consistency of structured requirements descriptions.

### Software Requirement Management:

Requirement management is the process of managing changing requirements during the requirements engineering process and system development.

New requirements emerge during the process as business needs a change, and a better understanding of the system is developed.

The priority of requirements from different viewpoints changes during development process.

The business and technical environment of the system changes during the development.

## Prerequisite of Software requirements

Collection of software requirements is the basis of the entire software development project. Hence they should be clear, correct, and well-defined.

A complete Software Requirement Specifications should be:

* Clear
* Correct
* Consistent
* Coherent
* Comprehensible
* Modifiable
* Verifiable
* Prioritized
* Unambiguous
* Traceable
* Credible source

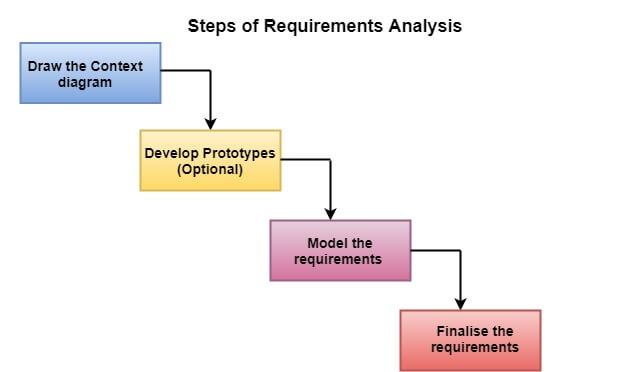
**Software Requirements:** Largely software requirements must be categorized into two categories:

1. **Functional Requirements:** Functional requirements define a function that a system or system element must be qualified to perform and must be documented in different forms. The functional requirements are describing the behavior of the system as it correlates to the system's functionality.
2. **Non-functional Requirements:** This can be the necessities that specify the criteria that can be used to decide the operation instead of specific behaviors of the system.  
   Non-functional requirements are divided into two main categories:
   * **Execution qualities** like security and usability, which are observable at run time.
   * **Evolution qualities** like testability, maintainability, extensibility, and scalability that embodied in the static structure of the software system.

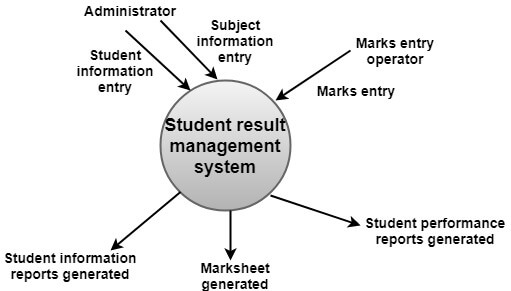
# Requirements Analysis

Requirement analysis is significant and essential activity after elicitation. We analyze, refine, and scrutinize the gathered requirements to make consistent and unambiguous requirements. This activity reviews all requirements and may provide a graphical view of the entire system. After the completion of the analysis, it is expected that the understandability of the project may improve significantly. Here, we may also use the interaction with the customer to clarify points of confusion and to understand which requirements are more important than others.

**The various steps of requirement analysis are shown in fig:**



**(i) Draw the context diagram:** The context diagram is a simple model that defines the boundaries and interfaces of the proposed systems with the external world. It identifies the entities outside the proposed system that interact with the system. The context diagram of student result management system is given below:



**(ii) Development of a Prototype (optional):** One effective way to find out what the customer wants is to construct a prototype, something that looks and preferably acts as part of the system they say they want.

**(iii) Model the requirements:** This process usually consists of various graphical representations of the functions, data entities, external entities, and the relationships between them. The graphical view may help to find incorrect, inconsistent, missing, and superfluous requirements. Such models include the Data Flow diagram, Entity-Relationship diagram, Data Dictionaries, State-transition diagrams, etc.

**(iv) Finalise the requirements:** After modeling the requirements, we will have a better understanding of the system behavior. The inconsistencies and ambiguities have been identified and corrected. The flow of data amongst various modules has been analyzed. Elicitation and analyze activities have provided better insight into the system.

## What is Structured Analysis?

Structured Analysis is a development method that allows the analyst to understand the system and its activities in a logical way.

It is a systematic approach, which uses graphical tools that analyze and refine the objectives of an existing system and develop a new system specification which can be easily understandable by user.

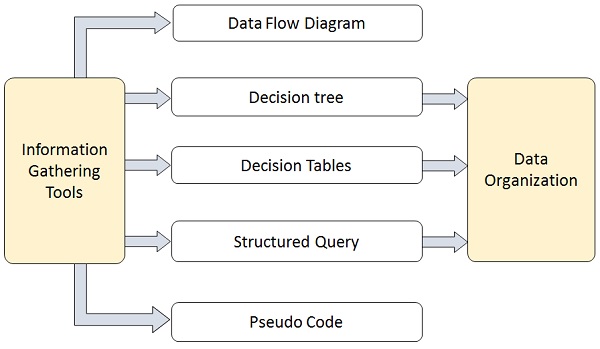
It has following attributes −

* It is graphic which specifies the presentation of application.
* It divides the processes so that it gives a clear picture of system flow.
* It is logical rather than physical i.e., the elements of system do not depend on vendor or hardware.
* It is an approach that works from high-level overviews to lower-level details.

## Structured Analysis Tools

During Structured Analysis, various tools and techniques are used for system development. They are −

* Data Flow Diagrams
* Data Dictionary
* Decision Trees
* Decision Tables
* Structured English
* Pseudocode



## Data Flow Diagrams (DFD) or Bubble Chart

It is a technique developed by Larry Constantine to express the requirements of system in a graphical form.

* It shows the flow of data between various functions of system and specifies how the current system is implemented.
* It is an initial stage of design phase that functionally divides the requirement specifications down to the lowest level of detail.
* Its graphical nature makes it a good communication tool between user and analyst or analyst and system designer.
* It gives an overview of what data a system processes, what transformations are performed, what data are stored, what results are produced and where they flow.

### Basic Elements of DFD

DFD is easy to understand and quite effective when the required design is not clear and the user wants a notational language for communication. However, it requires a large number of iterations for obtaining the most accurate and complete solution.

The following table shows the symbols used in designing a DFD and their significance −

|  |  |  |
| --- | --- | --- |
| **Symbol Name** | **Symbol** | **Meaning** |
| Square | Square | Source or Destination of Data |
| Arrow | Arrow | Data flow |
| Circle | Circle | Process transforming data flow |
| Open Rectangle | Rectangle | Data Store |

### Types of DFD

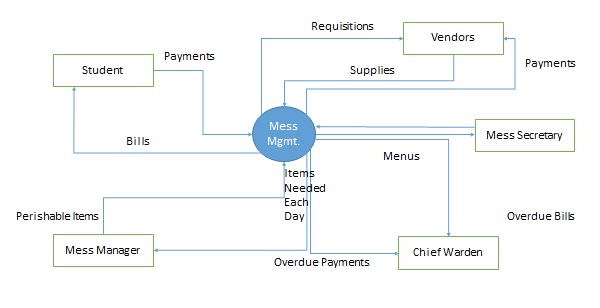
DFDs are of two types: Physical DFD and Logical DFD. The following table lists the points that differentiate a physical DFD from a logical DFD.

|  |  |
| --- | --- |
| **Physical DFD** | **Logical DFD** |
| It is implementation dependent. It shows which functions are performed. | It is implementation independent. It focuses only on the flow of data between processes. |
| It provides low level details of hardware, software, files, and people. | It explains events of systems and data required by each event. |
| It depicts how the current system operates and how a system will be implemented. | It shows how business operates; not how the system can be implemented. |

### Context Diagram

A context diagram helps in understanding the entire system by one DFD which gives the overview of a system. It starts with mentioning major processes with little details and then goes onto giving more details of the processes with the top-down approach.

The context diagram of mess management is shown below.



## Data Dictionary

A data dictionary is a structured repository of data elements in the system. It stores the descriptions of all DFD data elements that is, details and definitions of data flows, data stores, data stored in data stores, and the processes.

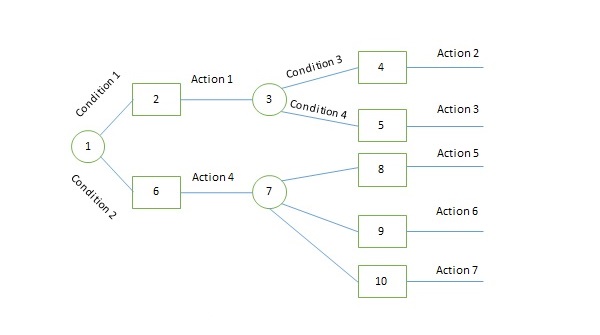
A data dictionary improves the communication between the analyst and the user. It plays an important role in building a database. Most DBMSs have a data dictionary as a standard feature. For example, refer the following table −

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr.No.** | **Data Name** | **Description** | **No. of Characters** |
| 1 | ISBN | ISBN Number | 10 |
| 2 | TITLE | Title | 60 |
| 3 | SUB | Book Subjects | 80 |
| 4 | ANAME | Author Name | 15 |

## Decision Trees

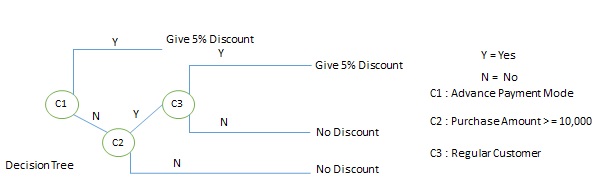
Decision trees are a method for defining complex relationships by describing decisions and avoiding the problems in communication. A decision tree is a diagram that shows alternative actions and conditions within horizontal tree framework. Thus, it depicts which conditions to consider first, second, and so on.

Decision trees depict the relationship of each condition and their permissible actions. A square node indicates an action and a circle indicates a condition. It forces analysts to consider the sequence of decisions and identifies the actual decision that must be made.



The major limitation of a decision tree is that it lacks information in its format to describe what other combinations of conditions you can take for testing. It is a single representation of the relationships between conditions and actions.

For example, refer the following decision tree −



## Decision Tables

Decision tables are a method of describing the complex logical relationship in a precise manner which is easily understandable.

* It is useful in situations where the resulting actions depend on the occurrence of one or several combinations of independent conditions.
* It is a matrix containing row or columns for defining a problem and the actions.

### Components of a Decision Table

* **Condition Stub** − It is in the upper left quadrant which lists all the condition to be checked.
* **Action Stub** − It is in the lower left quadrant which outlines all the action to be carried out to meet such condition.
* **Condition Entry** − It is in upper right quadrant which provides answers to questions asked in condition stub quadrant.
* **Action Entry** − It is in lower right quadrant which indicates the appropriate action resulting from the answers to the conditions in the condition entry quadrant.

The entries in decision table are given by Decision Rules which define the relationships between combinations of conditions and courses of action. In rules section,

* Y shows the existence of a condition.
* N represents the condition, which is not satisfied.
* A blank - against action states it is to be ignored.
* X (or a check mark will do) against action states it is to be carried out.

For example, refer the following table –

The condition states that if the user provides the correct username and password the user will be redirected to the homepage. If any of the input is wrong, an error message will be displayed.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CONDITIONS** | **Rule 1** | **Rule 2** | **Rule 3** | **Rule 4** |
| Username | F | T | F | T |
| Password | F | F | T | T |
| **ACTIONS** |  |  |  |  |
| Output | E | E | E | H |

In the above example,

* **T** – Correct username/password
* **F** – Wrong username/password
* **E** – Error message is displayed
* **H** – Home screen is displayed

Now let’s understand the interpretation of the above cases:

* **Case 1** – Username and password both were wrong. The user is shown an error message.
* **Case 2** – Username was correct, but the password was wrong. The user is shown an error message.
* **Case 3** – Username was wrong, but the password was correct. The user is shown an error message.
* **Case 4** – Username and password both were correct, and the user is navigated to the homepage.

**Why is Decision Table Important?**

A decision table is an outstanding technique used for testing and requirements management. Some of the reasons why the decision table is important include:

* Decision tables are very much helpful in test design technique.
* It helps testers to search the effects of combinations of different inputs and other software states that implement business rules.
* It provides a regular way of stating complex business rules which benefits the developers as well as the testers.
* It assists in the development process with the developer to do a better job. Testing with all combination might be impractical.
* It the most preferable choice for testing and requirements management.
* It is a structured exercise to prepare requirements when dealing with complex business rules.
* It is also used in model complicated logic.

## ****Advantages of Decision Table in Software Testing****

There are different advantages of using the decision table in software testing such as:

* Any complex business flow can be easily converted into the test scenarios & test cases using this technique.
* Decision tables work iteratively. Therefore, the table created at the first iteration is used as the input table for the next tables. The iteration is done only if the initial table is not satisfactory.
* Simple to understand and everyone can use this method to design the test scenarios & test cases
* It provides complete coverage of test cases which help to reduce the rework on writing test scenarios & test cases.
* These tables guarantee that we consider every possible combination of condition values. This is known as its completeness property

## Structured English

Structure English is derived from structured programming language which gives more understandable and precise description of process. It is based on procedural logic that uses construction and imperative sentences designed to perform operation for action.

* It is best used when sequences and loops in a program must be considered and the problem needs sequences of actions with decisions.
* It does not have strict syntax rule. It expresses all logic in terms of sequential decision structures and iterations.

For example, see the following sequence of actions −

if customer pays advance

then

Give 5% Discount

else

if purchase amount >=10,000

then

if the customer is a regular customer

then Give 5% Discount

else No Discount

end if

else No Discount

end if

end if

## Pseudocode

A pseudocode does not conform to any programming language and expresses logic in plain English.

* It may specify the physical programming logic without actual coding during and after the physical design.
* It is used in conjunction with structured programming.
* It replaces the flowcharts of a program.

# Petri Nets-Data Dictionary

Petri nets — Formal technique for describing concurrent interrelated activities

Consists of four parts

(1)             A set of places

(2)             A set of transitions

(3)             An input function

(4)             An output function

# Data Dictionaries in Software Engineering

Data Dictionary is the major component in the structured analysis model of the system. It lists all the data items appearing in DFD. A data dictionary in Software Engineering means a file or a set of files that includes a database’s metadata (hold records about other objects in the database), like data ownership, relationships of the data to another object, and some other data**.**

**Example** a data dictionary entry**:**GrossPay = regular pay + overtime pay

Case Tools is used to maintain data dictionary as it captures the data items appearing in a DFD automatically to generate the data dictionary.

### Components of Data Dictionary:

In Software Engineering, the data dictionary contains the following information:

* **Name of the item:**It can be your choice.
* **Aliases:**It represents another name.
* **Description:**Description of what the actual text is all about.
* **Related data items:**with other data items.
* **Range of values:**It will represent all possible answers.
* The Notations used within the data dictionary are given in the table below as follows:

| **Notations** | **Meaning** |
| --- | --- |
| **X = a+b** | X consists data elements a and b. |
| **X = [a/b]** | X consists of either elements a or b. |
| **X = a X** | X consists of optimal data elements a. |
| **X = y[a]** | X consists of y or more events of data element a |
| **X = [a] z** | X consists of z or less events of data element a |
| **X = y [a] z** | X consists of some events of data elements between y and z. |

### Features of Data Dictionary :

Here, we will discuss some features of the data dictionary as follows.

* It helps in designing test cases and designing the software.
* It is very important for creating an order list from a subset of the items list.
* It is very important for creating an order list from a complete items list.
* The data dictionary is also important to find the specific data item object from the list.

### Uses of Data Dictionary:

Here, we will discuss some use cases of the data dictionary as follows.

* Used for creating theordered listof data items
* Used for creating the ordered list of a subset of the data items
* Used for Designing and testing software in Software Engineering
* Used for finding data items from a description in Software Engineering

### Importance of Data Dictionary:

* It provides developers with standard terminology for all data.
* It provides developers to use different terms to refer to the same data.
* It provides definitions for different data
* Query handling is facilitated if a data dictionary is used in RDMS.

**Advantages of Data Dictionary:**

* **Consistency and Standardization:** A data dictionary helps to ensure that all data elements and attributes are consistently defined and named across the organization, promoting standardization and consistency in data management practices.
* **Data Quality:**A data dictionary can help improve data quality by providing a single source of truth for data definitions, allowing users to easily verify the accuracy and completeness of data.
* **Data Integration:** A data dictionary can facilitate data integration efforts by providing a common language and framework for understanding data elements and their relationships across different systems.
* **Improved Collaboration:** A data dictionary can help promote collaboration between business and technical teams by providing a shared understanding of data definitions and structures, reducing misunderstandings and communication gaps.
* **Improved Efficiency:** A data dictionary can help improve efficiency by reducing the time and effort required to define, document, and manage data elements and attributes.

**Disadvantages of Data Dictionary:**

* **Implementation and Maintenance Costs:**Implementing and maintaining a data dictionary can be costly, requiring significant resources in terms of time, money, and personnel.
* **Complexity:**A data dictionary can be complex and difficult to manage, particularly in large organizations with multiple systems and data sources.
* **Resistance to Change:** Some stakeholders may be resistant to using a data dictionary, either due to a lack of understanding or because they prefer to use their own terminology or definitions.
* **Data Security:**A data dictionary can contain sensitive information, and therefore, proper security measures must be in place to ensure that unauthorized users do not access or modify the data.
* **Data Governance:**A data dictionary requires strong data governance practices to ensure that data elements and attributes are managed effectively and consistently across the organization.