**DECISION TREE**

A decision tree gives a graphic view of the processing logic involved in decision making and the corresponding actions taken. The edges of a decision tree represent conditions and the leaf nodes represent the actions to be performed depending on the outcome of testing the condition. Example: - Consider Library Membership Automation Software (LMS) where it should support the following three options:

* New member
* Renewal
* Cancel membership

**New member option**

**Decision:** When the 'new member' option is selected, the software asks details about the member like the member's name, address, phone number etc.

**Action:** If proper information is entered then a membership record for the member is created and a bill is printed for the annual membership charge plus the security deposit payable.

**Renewal option**

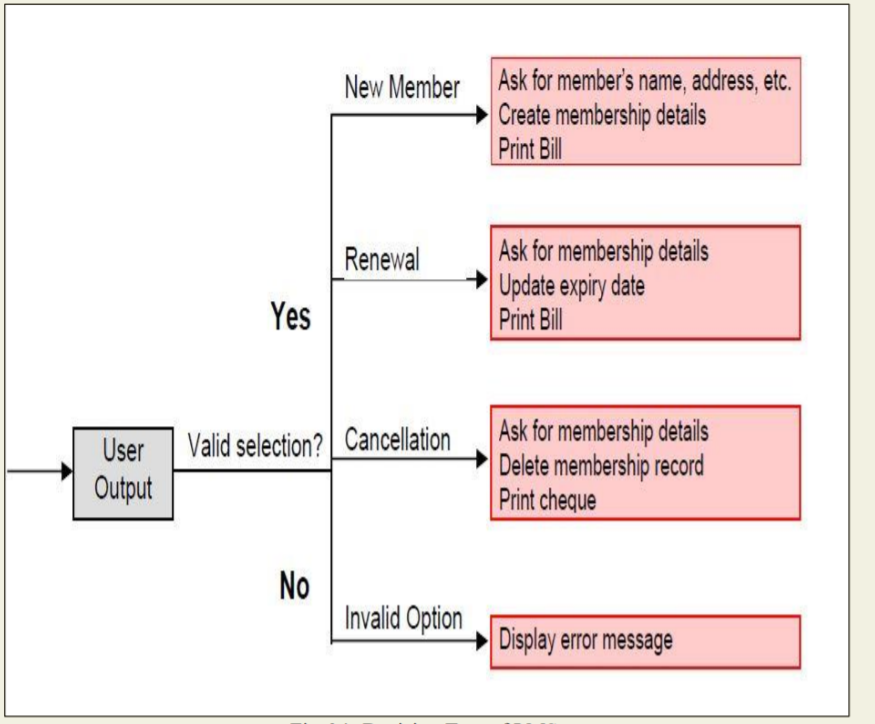
**Decision:** If the 'renewal' option is chosen, the LMS asks for the member's name and his membership number to check whether he is a valid member or not.

**Action:** If the membership is valid then membership expiry date is updated and the annual membership bill is printed, otherwise an error message is displayed.

**Cancel membership option**

**Decision:** If the 'cancel membership' option is selected, then the software asks for member's name and his membership number.

**Action:** The membership is cancelled, a cheque for the balance amount due to the member is printed and finally the membership record is deleted from the database.

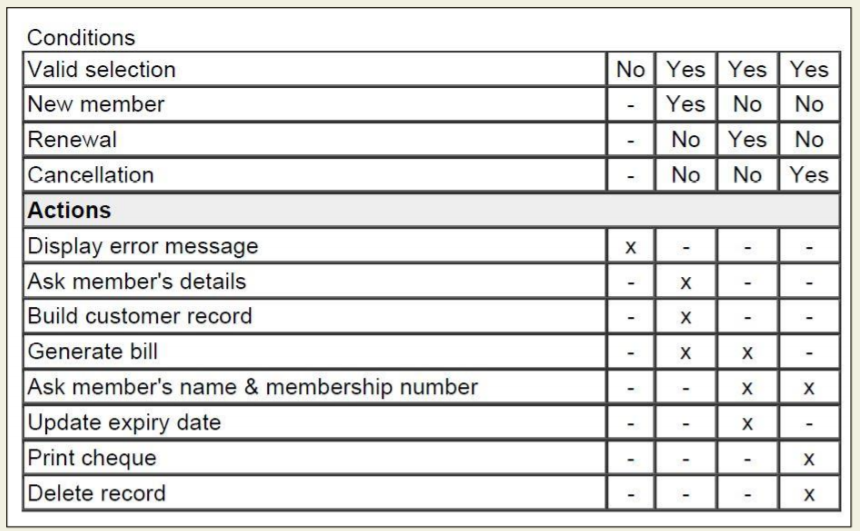


**Decision Tree for LMS**

**DECISION TABLE**

A decision table is used to represent the complex processing logic in a tabular or a matrix form. The upper rows of the table specify the variables or conditions to be evaluated. The lower rows of the table specify the actions to be taken when the corresponding conditions are satisfied. A column in a table is called a rule. A rule implies that if a condition is true, then the corresponding action is to be executed.

Example: - Consider the previously discussed LMS example. The following decision table shows how to represent the LMS problem in a tabular form. Here the table is divided into two parts, the upper part shows the conditions and the lower part shows what actions are taken. Each column of the table is a rule.

****

**Decision Table of LMS**

From the above table you can easily understand that, if the valid selection condition is false then the action taken for this condition is 'display error message'. Similarly, the actions taken for other conditions can be inferred from the table.

**SOFTWARE DESIGN**

Software design is a process to transform user requirements into some suitable form, which helps the programmer in software coding and implementation. For assessing user requirements, an SRS (Software Requirement Specification) document is created whereas for coding and implementation, there is a need of more specific and detailed requirements in software terms. The output of this process can directly be used into implementation in programming languages.

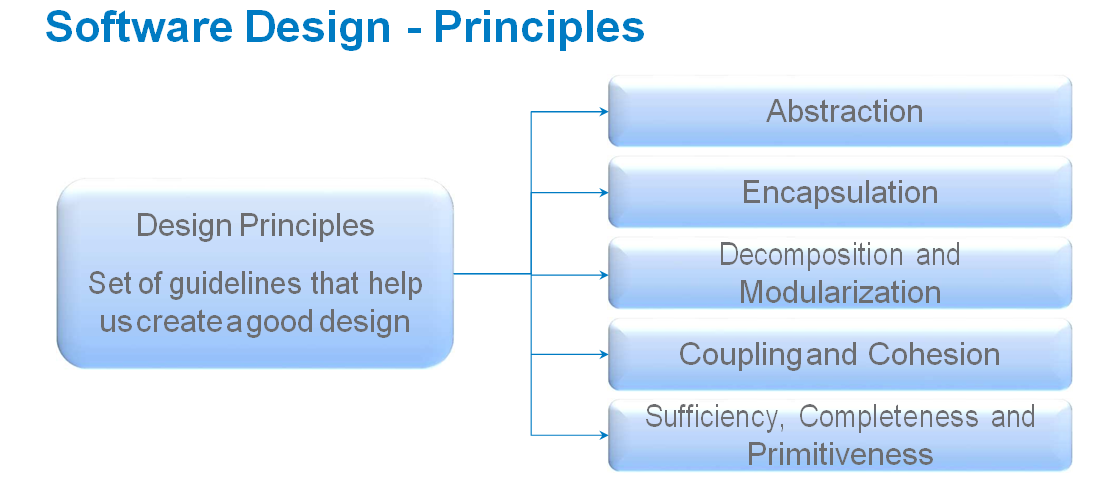
Software design is the first step in SDLC (Software Design Life Cycle), which moves the concentration from problem domain to solution domain. It tries to specify how to fulfill the requirements mentioned in SRS.

**Software Design Levels Software design yields three levels of results:**

**Architectural Design -** The architectural design is the highest abstract version of the system. It identifies the software as a system with many components interacting with each other. At this level, the designers get the idea of proposed solution domain.

**High-level Design-** The high-level design breaks the ‘single entity-multiple component’ concept of architectural design into less-abstracted view of sub-systems and modules and depicts their interaction with each other. High-level design focuses on how the system along with all of its components can be implemented in forms of modules. It recognizes modular structure of each sub-system and their relation and interaction among each other.

**Detailed Design-** Detailed design deals with the implementation part of what is seen as a system and its sub-systems in the previous two designs. It is more detailed towards modules and their implementations. It defines logical structure of each module and their interfaces to communicate with other modules.

****

**Modularization**

Modularization is a technique to divide a software system into multiple discrete and independent modules, which are expected to be capable of carrying out task(s) independently. These modules may work as basic constructs for the entire software. Designers tend to design modules such that they can be executed and/or compiled separately and independently.

Modular design unintentionally follows the rules of ‘divide and conquer’ problem-solving strategy this is because there are many other benefits attached with the modular design of a software.

**Advantage of modularization:**

* Smaller components are easier to maintain
* Program can be divided based on functional aspects
* Desired level of abstraction ca n be brought in the program
* Components with high cohesion can be re-used again.
* Concurrent execution can be made possible
* Desired from security aspect

Concurrency Back in time, all softwares were meant to be executed sequentially. By sequential execution we mean that the coded instruction will be executed one after another implying only one portion of program being activated at any given time. Say, a software has multiple modules, then only one of all the modules can be found active at any time of execution.

In software design, concurrency is implemented by splitting the software into multiple independent units of execution, like modules and executing them in parallel. In other words, concurrency provides capability to the software to execute more than one part of code in parallel to each other.

It is necessary for the programmers and designers to recognize those modules, which can be made parallel execution.

**Example The spell check feature in word processor is a module of software, which runs alongside the word processor itself.**

**Coupling and Cohesion**

When a software program is modularized, its tasks are divided into several modules based on some characteristics. As we know, modules are set of instructions put together in order to achieve some tasks. They are though, considered as single entity but may refer to each other to work together. There are measures by which the quality of a design of modules and their interaction among them can be measured. These measures are called coupling and cohesion.

**Cohesion**

Cohesion is a measure that defines the degree of intra-dependability within elements of a module. The greater the cohesion, the better is the program design.

There are seven types of cohesion, namely –

1. **Co-incidental cohesion -** It is unplanned and random cohesion, which might be the result of breaking the program into smaller modules for the sake of modularization. Because it is unplanned, it may serve confusion to the programmers and is generally not-accepted.
2. **Logical cohesion -** When logically categorized elements are put together into a module, it is called logical cohesion.
3. **Temporal Cohesion -** When elements of module are organized such that they are processed at a similar point in time, it is called temporal cohesion.
4. **Procedural cohesion -** When elements of module are grouped together, which are executed sequentially in order to perform a task, it is called procedural cohesion.
5. **Communicational cohesion -** When elements of module are grouped together, which are executed sequentially and work on same data (information), it is called communicational cohesion.
6. **Sequential cohesion -** When elements of module are grouped because the output of one element serves as input to another and so on, it is called sequential cohesion.
7. **Functional cohesion -** It is considered to be the highest degree of cohesion, and it is highly expected. Elements of module in functional cohesion are grouped because they all contribute to a single well-defined function. It can also be reused.

**Coupling**

Coupling is a measure that defines the level of inter-dependability among modules of a program. It tells at what level the modules interfere and interact with each other. The lower the coupling, the better the program.

There are five levels of coupling, namely -

1. **Content coupling -** When a module can directly access or modify or refer to the content of another module, it is called content level coupling.
2. Common coupling- When multiple modules have read and write access to some global• data, it is called common or global coupling.
3. Control coupling- Two modules are called control-coupled if one of them decides the• function of the other module or changes its flow of execution.
4. Stamp coupling- When multiple modules share common data structure and work on different part of it, it is called stamp coupling.
5. Data coupling- Data coupling is when two modules interact with each other by means of passing data (as parameter). If a module passes data structure as parameter, then the receiving module should use all its components.

**Ideally, no coupling is considered to be the best.**