Unit 3 Notes

Software design is a mechanism to transform user requirements into some suitable form, which helps the programmer in software coding and implementation.

The design phase of software development deals with transforming the customer requirements as described in the SRS documents into a form implementable using a programming language.

In software design, we consider the system to be a set of components or modules with clearly defined behaviors & boundaries.

Software Design Principles

Software design principles are concerned with providing means to handle the complexity of the design process effectively. Effectively managing the complexity will not only reduce the effort needed for design but can also reduce the scope of introducing errors during design.

1. Problem Partitioning

For small problem, we can handle the entire problem at once but for the significant problem, divide the problems and conquer the problem it means to divide the problem into smaller pieces so that each piece can be captured separately.

For software design, the goal is to divide the problem into manageable pieces.

Benefits of Problem Partitioning

1. Software is easy to understand
2. Software becomes simple
3. Software is easy to test
4. Software is easy to modify
5. Software is easy to maintain
6. Software is easy to expand

2. Abstraction

An abstraction is a tool that enables a designer to consider a component at an abstract level without bothering about the internal details of the implementation. Abstraction can be used for existing element as well as the component being designed.

Here, there are two common abstraction mechanisms

1. Functional Abstraction
2. Data Abstraction

3. Modularity

Modularity specifies to the division of software into separate modules which are differently named and addressed and are integrated later on in to obtain the completely functional software. It is the only property that allows a program to be intellectually manageable. Single large programs are difficult to understand and read due to a large number of reference variables, control paths, global variables, etc.

Modular Design

Modular design reduces the design complexity and results in easier and faster implementation by allowing parallel development of various parts of a system. We discuss a different section of modular design in detail in this section:

1. Functional Independence

2. Information hiding

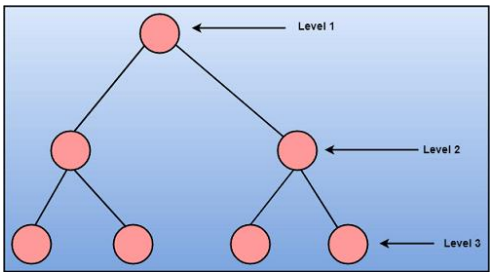
4. Strategy of Design

A good system design strategy is to organize the program modules in such a method that are easy to develop and latter too, change. Structured design methods help developers to deal with the size and complexity of programs. Analysts generate instructions for the developers about how code should be composed and how pieces of code should fit together to form a program.

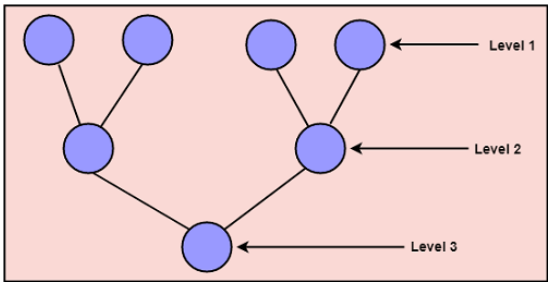
To design a system, there are two possible approaches:

1. Top-down Approach
2. Bottom-up Approach

1. Top-down Approach: This approach starts with the identification of the main components and then decomposing them into their more detailed sub-components.

3. 

2. Bottom-up Approach: A bottom-up approach begins with the lower details and moves towards up the hierarchy, as shown in fig. This approach is suitable in case of an existing system.

4. 

High-level design

High-level design or **HLD** refers to the overall system, a design that consists description of the system architecture and design and is a generic system design that includes:

1. System architecture
2. Database design
3. Brief description of systems, services, platforms, and relationships among modules.

Low-level design

LLD, or Low-Level Design, is a phase in the software development process where detailed system components and their interactions are specified.

- It describes detailed description of each and every module means it includes actual logic for every system component and it goes deep into each modules specification.
- It is also known as micro level/detailed design.
- It is created by designers and developers.

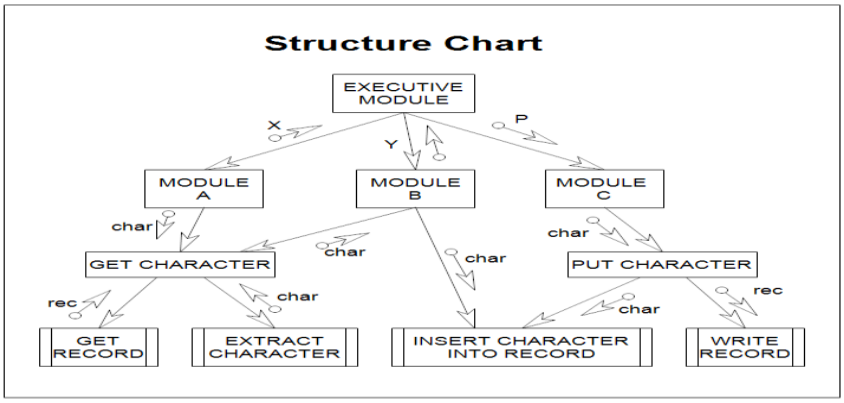
Modularization: Modularization is the process of dividing a software system into multiple independent modules where each module works independently. There are many advantages of Modularization in software engineering.
Some of these are given below:

- Easy to understand the system.
- System maintenance is easy.
- A module can be used many times as their requirements. No need to write it again and again.

Structure Chart

A structure chart (SC) in software engineering and organizational theory is a chart which shows the breakdown of a system to its lowest manageable levels. A classic "organization chart" for a company is an example of a structure chart. The top of the chart is a box representing the entire problem, the bottom of the chart shows a number of boxes representing the less complicated subproblems. (Left-right on the chart is irrelevant.) A structure chart is NOT a flowchart

.



Module coupling and cohesion

Coupling describes the relationships between modules, and **cohesion** describes the relationships within them. A reduction in interconnectedness between modules (or classes) is therefore achieved via a reduction in coupling.

Coupling refers to the degree of interdependence between software modules. High coupling means that modules are closely connected and changes in one module may affect other modules. Low coupling means that modules are independent and changes in one module have little impact on other modules.

1. No Direct Coupling: There is no direct coupling between M1 and M2.

In this case, modules are subordinates to different modules. Therefore, no direct coupling.

2. Data Coupling: When data of one module is passed to another module, this is called data coupling.

3. Stamp Coupling: Two modules are stamp coupled if they communicate using composite data items such as structure, objects, etc. When the module passes non-global data structure or entire structure to another module, they are said to be stamp coupled. For example, passing structure variable in C or object in C++ language to a module.

4. Control Coupling: Control Coupling exists among two modules if data from one module is used to direct the structure of instruction execution in another.

5. External Coupling: External Coupling arises when two modules share an externally imposed data format, communication protocols, or device interface. This is related to communication to external tools and devices.

6. Common Coupling: Two modules are common coupled if they share information through some global data items.

7. Content Coupling: Content Coupling exists among two modules if they share code, e.g., a branch from one module into another module.

Cohesion defines to the degree to which the elements of a module belong together. Thus, cohesion measures the strength of relationships between pieces of functionality within a given module.

1. **Functional Cohesion:** Functional Cohesion is said to exist if the different elements of a module, cooperate to achieve a single function.
2. **Sequential Cohesion:** A module is said to possess sequential cohesion if the element of a module form the components of the sequence, where the output from one component of the sequence is input to the next.
3. **Communicational Cohesion:** A module is said to have communicational cohesion, if all tasks of the module refer to or update the same data structure, e.g., the set of functions defined on an array or a stack.
4. **Procedural Cohesion:** A module is said to be procedural cohesion if the set of purpose of the module are all parts of a procedure in which particular sequence of steps has to be carried out for achieving a goal, e.g., the algorithm for decoding a message.
5. **Temporal Cohesion:** When a module includes functions that are associated by the fact that all the methods must be executed in the same time, the module is said to exhibit temporal cohesion.
6. **Logical Cohesion:** A module is said to be logically cohesive if all the elements of the module perform a similar operation. For example Error handling, data input and data output, etc.
7. **Coincidental Cohesion:** A module is said to have coincidental cohesion if it performs a set of tasks that are associated with each other very loosely, if at all.

Function Oriented Design

Function Oriented design is a method to software design where the model is decomposed into a set of interacting units or modules where each unit or module has a clearly defined function. Thus, the system is designed from a functional viewpoint.

Object-Oriented Design

In the object-oriented design method, the system is viewed as a collection of objects (i.e., entities). The state is distributed among the objects, and each object handles its state data. For example, in a Library Automation Software, each library representative may be a separate object with its data and functions to operate on these data. The tasks defined for one purpose cannot refer or change data of other objects. Objects have their internal data which represent their state. Similar objects create a class. In other words, each object is a member of some class. Classes may inherit features from the superclass.

Coding Standards and Guidelines

Good software development organizations want their programmers to maintain to some well-defined and standard style of coding called coding standards. They usually make their own coding standards and guidelines depending on what suits their organization best and based on the types of software they develop. It is very important for the programmers to maintain the coding standards otherwise the code will be rejected during code review.

Purpose of Having Coding Standards:

- A coding standard gives a uniform appearance to the codes written by different engineers.
- It improves readability, and maintainability of the code and it reduces complexity also.
- It helps in code reuse and helps to detect error easily.
- It promotes sound programming practices and increases efficiency of the programmers.

Some of the coding standards are given below:

1. **Limited use of globals:** These rules tell about which types of data that can be declared global and the data that can't be.

2. **Standard headers for different modules:** For better understanding and maintenance of the code, the header of different modules should follow some standard format and information. The header format must contain below things that is being used in various companies:
   - Name of the module
   - Date of module creation
   - Author of the module
   - Modification history
   - Synopsis of the module about what the module does
   - Different functions supported in the module along with their input output parameters
   - Global variables accessed or modified by the module

3. **Naming conventions for local variables, global variables, constants and functions:** Some of the naming conventions are given below:
   - Meaningful and understandable variables name helps anyone to understand the reason of using it.
   - Local variables should be named using camel case lettering starting with small letter (e.g. **local Data**) whereas Global variables names should start with a capital letter (e.g. **Global Data**). Constant names should be formed using capital letters only (e.g. **CONSDATA**).

- It is better to avoid the use of digits in variable names.
- The names of the function should be written in camel case starting with small letters.
- The name of the function must describe the reason of using the function clearly and briefly.

4. **Indentation:** Proper indentation is very important to increase the readability of the code. For making the code readable, programmers should use White spaces properly. Some of the spacing conventions are given below:
   - There must be a space after giving a comma between two function arguments.
   - Each nested block should be properly indented and spaced.
   - Proper Indentation should be there at the beginning and at the end of each block in the program.
   - All braces should start from a new line and the code following the end of braces also start from a new line.

5. **Error return values and exception handling conventions:** All functions that encountering an error condition should either return a 0 or 1 for simplifying the debugging. On the other hand, Coding guidelines give some general suggestions regarding the coding style that to be followed for the betterment of understand ability and readability of the code. Some of the coding guidelines are given below :

6. **Avoid using a coding style that is too difficult to understand:** Code should be easily understandable. The complex code makes maintenance and debugging difficult and expensive.

7. **Avoid using an identifier for multiple purposes:** Each variable should be given a descriptive and meaningful name indicating the reason behind using it. This is not possible if an identifier is used for multiple purposes and thus it can lead to confusion to the reader. Moreover, it leads to more difficulty during future enhancements.

8. **Code should be well documented:** The code should be properly commented for understanding easily. Comments regarding the statements increase the understand ability of the code.

9. **Length of functions should not be very large:** Lengthy functions are very difficult to understand. That's why functions should be small enough to carry out small work and lengthy functions should be broken into small ones for completing small tasks.

Top down and Bottom up approach in programming

A program is instructions that the computer executes that perform some meaningful work. Top down design starts with the general concept and repeatedly breaks it down into its component parts. Bottom up program design starts with component parts and repeatedly merges them into the general concept.

The structured program consists of well structured and separated modules. But the entry and exit in a Structured program is a single-time event. It means that the program uses single-entry and single-exit elements. Therefore a structured program is well maintained, neat and clean program. This is the reason why the Structured Programming Approach is well accepted in the programming world.

Advantages of Structured Programming Approach:

1. Easier to read and understand
2. User Friendly

3. Easier to Maintain
4. Mainly problem based instead of being machine based
5. Development is easier as it requires less effort and time
6. Easier to Debug
7. Machine-Independent, mostly.

Disadvantages of Structured Programming Approach:

1. Since it is Machine-Independent, So it takes time to convert into machine code.
2. The converted machine code is not the same as for assembly language.
3. The program depends upon changeable factors like data-types. Therefore it needs to be updated with the need on the go.
4. Usually the development in this approach takes longer time as it is language-dependent. Whereas in the case of assembly language, the development takes lesser time as it is fixed for the machine.

References

- Software Engineering: A Practitioner's Approach | 9th Edition By Roger S. Pressman, Bruce R. Maxim
- Software Engineering 9th Edition by Ian Sommerville.
- https://www.geeksforgeeks.org/software-engineering-introduction-to-software-engineering/
- https://www.coursera.org/learn/introduction-to-software-engineering
- https://en.wikipedia.org/wiki/Software_engineering