BACHELOR OF COMPUTER APPLICATIONS

SEMESTER 5

DCA3103

SOFTWARE ENGINEERING

Unit 4

Software Design Principles

Table of Contents

1. INTRODUCTION

Software design is an essential phase of the software engineering process for creating and evaluating software models that guide the construction effort for developing high-quality software systems on time and within budget. Design is an integral part of every engineering discipline. Airplanes, bridges, buildings, electronic devices, cars and many other products of similar complexity are all designed. In civil engineering, designs are used to specify detailed plans for developing physical and naturally built environments, such as bridges, roads, canals, dams, and buildings. In electrical engineering, designs are used to capture, evaluate, and specify the detailed qualitative and quantitative description of solutions for telecommunication systems, electrical systems and electronic devices. In mechanical engineering, designs are used for analysing, evaluating and specifying technical features required to construct machines and tools such as industrial equipment, heating and cooling systems, aircraft, robots and medical devices. In all other engineering disciplines, the design provides a systematic approach for creating products that meet their intended functions and users' expectations.

Software design, data design, architectural design, component level design, and user interface design fundamentals are covered in Unit 1. Additionally, essential design principles like module and modularization will be covered.

1.1 Learning Objectives:

At the end of the topic, students will be able to:

- ❖ *Explain the System models used in Software Design.*
- ❖ *Recall the fundamental principles of software design.*
- ❖ *Explain the importance of software design in the software development lifecycle.*
- ❖ *Apply the principles of software design to solve real-world software problems.*
- ❖ *Analyse the strengths and weaknesses of different software design approaches.*
- ❖ *Compare and contrast different design patterns for a given software problem.*

2. SYSTEM MODELS

A system model is a representation of a software system that includes its behaviour, structure, and non-functional requirements. Software systems are communicated, analysed, designed, and validated using system models.

Software engineering utilises a variety of system model types, each with a distinct function. *The following are a few of the most typical types of system models:*

- ***Functional models:*** These models describe the inputs, outputs, and processes of the software system.
- ***Structural models:*** These models show the components, modules, and connections between them that make up a software system like class diagrams, generalization and aggregation.
- ***Behavioral models:*** Use cases, scenarios, and state diagrams are examples of behavioural models that show how a software system responds to various inputs or events.
- ***Architectural models:*** Models describing the software system's architecture, including its high-level components, their interactions, and the mapping of these components to hardware and software platforms, are represented by these models.
- ***Data models:*** These models show the software system's data structures, including the entities, attributes, and relationships between them.
- ***Process models:*** These models describe the processes or workflows used in the development, deployment, and maintenance of the software system, including process flows, activity diagrams, and state transition diagrams.

3. BASICS OF SOFTWARE DESIGN

Software design is the process of transforming functional and non-functional requirements into models that describe the technical solution before development begins. To achieve this, the concept of software design, its activities and tasks must be well understood so that a problem-solving framework for designing quality software products can be established.

In today's modern software systems, there are numerous design principles, processes, strategies and other factors affecting how designers execute the software design phase.

A software design is a process of problem-solving and planning for a software solution. The design model resides at the core of the software engineering process consisting of several entities and relationships between these entities. The design is a baseline for any detailed implementation. It helps in the interoperability between the designers of subsystems.

It gives prior information on the system maintenance that the designers intend for.

Every phase of SDLC has a detailed document as an outcome. The software design document is also one such outcome that an architect designs after obtaining a detailed requirement specification from the customers.

There are primarily three main types of notations used in the design document:

a) ***Graphical Notations:*** This notation is used to represent co-relation among the entities involved in a design using certain modelling languages such as Unified Modeling Language (UML). It provides an abstract picture of the software at the early stages of the Software Development Life Cycle (SDLC).

b) ***Program Description Languages:*** Program Design Language (or PDL, for short) is a method for designing and documenting methods and procedures in software. It is related to pseudo-code, but unlike pseudocode, it is written in plain language without any terms that could suggest the use of any programming language or library.

c) ***Informal Text:*** The design explanation is expected to be more readable and hence the design is explained using natural language text called informal text.

Any design problem is solved using the following three approaches:

1) ***Detailed Study and Understanding of the Problem:*** The problem must be analyzed from different perspectives or views to check if all the requirements are met according to the design requirements.
2) ***Identify the Core Features of at Least One Possible Solution:*** It is useful to detect more solutions and to evaluate each of them. The choice of solution depends on the designer's view based on his/her experience, the availability of reusable components and the level of simplicity of the desired solutions.

Designers prefer to choose familiar solutions though they are not optimal as they have a better understanding of the pros and cons of using a familiar solution.

3) ***Describe all the Abstracted Information Used in the Solution:*** Before releasing any formal document, the designer must write an informal design description. This may be analyzed by developing the solution in detail. Errors and omissions in the high-level design will be detected during this analysis. Such errors are corrected before the design is documented

3.1 Design Process

A general model of a software design is a directed graph. The target of the design process is the creation of a graph without any inconsistencies. Nodes in this graph represent entities in the design entities such as process functions or types. The link represents a relation between these design entities such as calls, uses and so on. Software designers do not arrive at a finished design graph immediately but develop the design iteratively through several different versions. The design process involves adding formality and detail as the design is developed with constant backtracking to correct earlier, less formal, designs. The starting point is an informal design, which is refined by adding information to make it consistent and complete as shown in Figure 4.1 below.

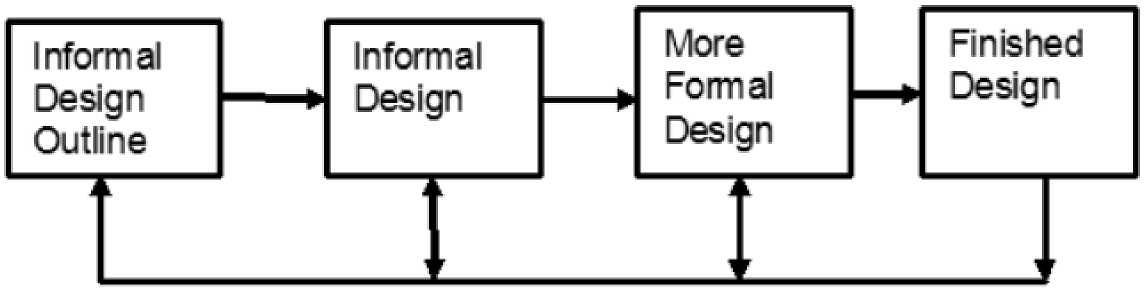

Fig 4.1: The progression from an informal to a detailed design

A general model of the design process shown in Figure 4.2 suggests that the design process stages are sequential in nature, but the activities of the design process will be proceeding in parallel.

The activities which are shown below are part of designing large software systems.

(1) ***Architectural Designs*** the sub-systems making up the system and their relationships are identified and documented.

(2) ***Abstract Specification*** for each sub-system, one to produce the abstract specification and the constraints.

(3) ***Interface Design*** for each sub-system, its interface with other sub-systems is designed and documented. This interface specification must be unambiguous as it allows the sub-system to be used without knowledge of the sub-system operation.

(4) ***Component Design*** Services are allocated to different components and the interfaces of these components are designed.

(5) ***Data Structure Design*** The data structures used in the system implementation is designed in detail and specified.

(6) ***Algorithm Design*** The algorithms used to provide services are designed in detail and specified.

This process is repeated for each sub-system until the components identified can be mapped directly into programming language components such as packages, procedures or functions.

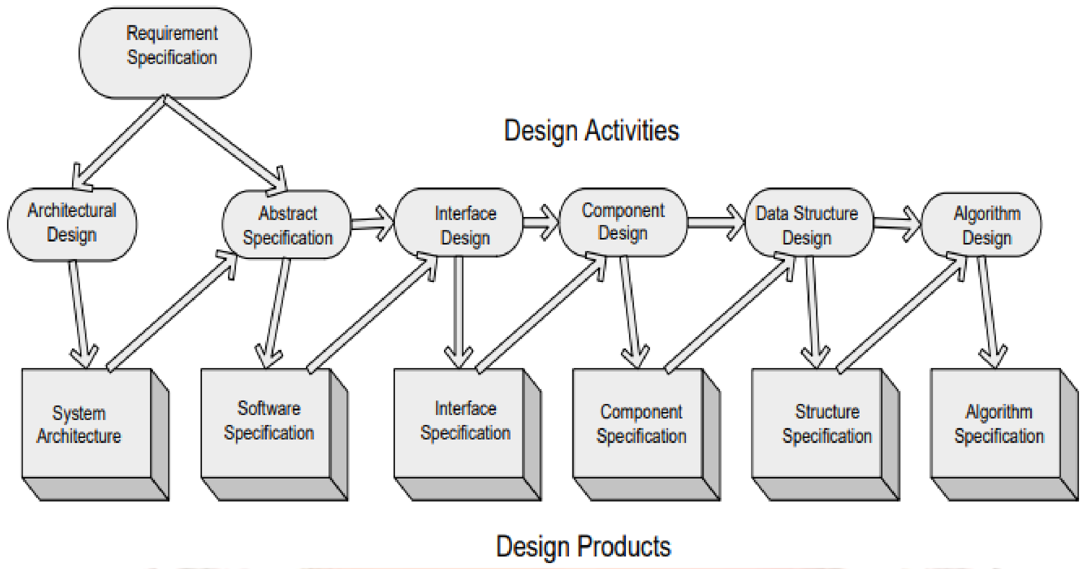

Fig 4.2: A general model of the design process

4. DATA DESIGN

The data design is the model of data that is represented at a high level of abstraction. The data design is then progressively refined to create implementation-specific representations. Various elements of data design are:

- ***Data Object* –** The data objects are identified and the relationships among various data objects can be represented using entity relationship diagrams or data dictionaries.
- Databases – using a software design model, the data models are translated into data structures and databases at the application level.
- Data Warehouses – at the business level useful information is identified from various databases and data warehouses are created. For extracting or navigating useful business information is stored in the data warehouse.

Guidelines for Data Design are listed below:

- ***Apply Systematic Analysis on Data:*** Represent data objects, relationships among them and data flow along with the contents.
- ***Identify Data Structures and Related Operations:*** For the design of efficient data structures, all the operations that will be performed on them should be considered.

- ***Establish Data Dictionary:*** The data dictionary explicitly represents various data objects, the relationships among them and the constraints on the elements of data structures.
- ***Defer the Low-Level Design Decisions until Late in the Design Process***: Major structural attributes are designed first to establish an architecture of data. And then low-level design attributes are established.
- ***Use Information Hiding in the Design of Data Structures:*** The use of information hiding helps in improving the quality of software design. It also helps in separating the logical and physical views.
- ***Apply a Library of Useful Data Structures and Operations:*** The data structures can be designed for reusability. The use of a library of data structure templates (called abstract data types) reduces the specification and design efforts for data.
- ***Use a Software Design and Programming Language to Support Data Specification and Abstraction:*** The implementation of data structures can be done by effective software design and by choosing a suitable programming language.

5. ARCHITECTURAL DESIGN

The requirements of the software should be transformed into an architecture that describes the software's top-level structure and identifies its components. This is accomplished through architectural design (also called system design), which is a preliminary 'blueprint' from which software can be developed.

Large systems can be decomposed into sub-system that provides some related set of services. The initial design process of identifying this sub-system and establishing a framework for sub-system control and communication is called Architectural design.

Architectural design comes before detailed system specification, it should not include any design information. Architectural design is necessary to structure and organize the specification. This model is the starting point for the specification of the various parts of the system.

An architectural design performs the following functions:

- It provides a level of abstraction at which the software designers can specify the system behaviour (such as function and performance).
- It serves as the conscience for a system as it evolves. By characterizing the crucial system design assumptions, a good architectural design guides the process of system enhancement indicating what aspects of the system can be easily changed without compromising system integrity.
- It evaluates all top-level designs.
- It develops and documents top-level design for the external and internal interfaces.
- It improves preliminary versions of user documentation.
- It describes and documents preliminary test requirements and the schedule for software integration.

Sources of Architectural Design:

- Information regarding the application domain for the software to be developed.
- Using data-flow diagrams.
- Availability of architectural patterns and architectural styles.

Architectural design occupies a pivotal position in software engineering. During architectural design, crucial requirements such as performance, reliability, costs, etc. are addressed. This job is clumsy as the software engineering model is shifting from huge, stand-alone, built-from-scratch systems to componentized, evolvable, standards-based and product line-oriented systems. Also, an important task for designers is to understand exactly how to continue from requirements to architectural design. To avoid these problems, designers accept strategies like reusability, componentization, platform-based, standards-based and many more.

With the architectural design the responsibility of developers, participants in the architectural design phase should also include user representatives, systems engineers, hardware engineers, and operational personnel. In reviewing the architectural design, the project management should ensure that all parties are consulted to minimize the risk of incompleteness and error.

Architectural Design Representation

Architectural design can be represented using various models, such as:

- ***Structural Model:*** This shows architecture as a systematic collection of program components.
- ***Framework Model:*** This tries to recognize repeatable architectural design patterns met in similar types of applications which leads to a rise in the level of abstraction.
- ***Dynamic Model:*** This states the behavioural features of the software architecture and shows how the structure or system configuration changes as the function change due to changes in the external environment.
- ***Process Model:*** This specifies the design of the business or technical process, which must be implemented in the system.
- ***Functional Model:*** This denotes the function hierarchy of the system.

There is no generally accepted process model for architectural design. The process depends on application knowledge and the skill and intuition of the system architect. For this process, the following activities are usually necessary:

(1) ***System Structuring:*** The system is structured into several principal sub-systems where a sub-system is an independent software unit. Communications between sub-systems are identified.

(2) ***Control Modeling:*** A general model of the control relationships between the parts of the system is established.

(3) ***Modular Decomposition:*** Each identified sub-system is decomposed into modules. The architect must decide on the types of the module and their interconnections.

During any of these process stages, it may be necessary to develop the design in more detail to find out if the architectural design decision allows the system to meet its requirements. The output of the architectural design process is an architectural design document. This consists of several graphical representations of the system models along with associated descriptive text. It should describe how the system is structured into sub-systems and how each sub-system is structured into modules.

6. COMPONENT-LEVEL DESIGN

As quickly as the first iteration of architectural design is complete, component-level design takes place. Component-level design is created by transforming the structural elements defined by the software architecture into procedural descriptions of software components. These components are derived from the analysis model where the data-flow-oriented element (present in the analysis model) serves as the base for the derivation.

A component, also known as a module, resides within the software architecture and serves one of the following three roles:

- ***A Control Component,*** which coordinates the invocation of all other components present in the problem domain.
- ***A Problem Domain Component,*** which implements a complete or partial function as required by the user.
- ***An Infrastructure Component*** supports functions, which in turn support the processing required in the problem domain.

Component-level design is used to define the data structures, algorithms, interface description and communication mechanisms allocated to each module. The module or component can be defined as a modular building block for the software. Though, the importance of components varies according to how software engineers use it.

The modular design of the software should exhibit the following sets of properties:

- ***Provide Simple Interfaces:*** Simple interfaces reduce the number of interactions that must be considered when verifying that a system performs its intended function. Simple interfaces also make it easier to reuse components in different circumstances. Reuse is a major cost saver. Not only does it reduce the time spent in coding, designing and testing but also allows development costs to be amortized over many projects.
- ***Ensure Information Hiding:*** the benefits of modularity automatically do not follow the act of subdividing a program. Each module should encapsulate information that is not available to the rest of the program. This reduces the cost of subsequent design changes. For example, a module may encapsulate related functions which can benefit from a common implementation, or which are used in many parts of a system.

Modularity has become an accepted method in each engineering discipline. With the overview of modular design, the difficulty of software design has significantly reduced, and change in the program is facilitated which has encouraged parallel development of systems. To attain effective modularity, design thoughts such as functional independence are reflected to be very important.

7. USER INTERFACE DESIGN

The user interface determines how users interact with the software. The user interface design creates an effective communication medium between a human and a computer machine. It provides easy and intuitive access to information as well as efficient interaction and control of software functionality. For this, the designers must understand what the user requires from the interface. Since the user is central while developing the software, the user interface must also be central while designing the software. It is important to first know the user for whom the user interface is being designed before designing the user interface. Direct contact between end-users and developers often improves the user interface design. The result of this communication helps the designers to know the user's goals and needs.

While designing the user interface software engineer communicates with the user and according to his thinking about the interface, the software engineer draws the sketches. Get it approved by the user and then work on defining objects and corresponding actions.

7.1 User Interface Rules

Designing a good and efficient user interface is a common objective among software designers. But what makes a user interface look good.

Software engineering strives to achieve a good user interface by following three rules namely ease of learning, efficiency of use and aesthetic appeal.

- **Ease of learning:** Ease of learning describes how quickly and effortlessly users learn to use the software. Ease of learning is especially important for new users. However, even experienced users face a learning experience problem while attempting to expand their usage of the product or when using a new version of the software. Here, the principle of state visualization is applied, which states that each change in the behaviour of the software should be accompanied by a corresponding change in the appearance of the interface.

Generally, to ease the task of learning, designers make use of the following tools:

a. ***Affordance:*** provides clues that suggest what a machine or tool can do and how to use it. For example, the style of the door handle on the doors of many departmental stores, offices and shops suggests whether to pull a door or push a door to open it. If the wrong style door handle is used, people struggle with the door. In this case, the door handle is more than just a tool for physically helping you to open the door, it is also an affordance showing you how the door opens. Similarly, software designers while developing user interfaces should offer hints as to what each part does and how it functions.

b. ***Consistency:*** Designers strive to maintain consistency within the interface. Every aspect of the interface, including minor details, such as font usage and colours is kept consistent when the behaviour is consistent. Here the principle of coherence (i.e. the behaviour of the program should be internally and externally consistent) is applied. Internal consistency means that the behaviour of the program must make 'sense' concerning other parts of the program. For example, if one attribute of an object (for example colour) is modified using a pop-up menu, then it is expected that other attributes of the object will also be edited similarly. External consistency means that the program is consistent with the environment in which it runs. This includes consistency with both the operating system and another suite of applications that run within that operating system.

- **Efficiency of Use:** once a user knows how to perform tasks, then efficiency can be evaluated reasonably only if users are no longer engaged in learning how to do the task and are rather engaged in performing the task. A few guidelines help in designing an efficient interface.

a) The task should require minimal physical actions. The desire of experienced users for hotkeys and shortcuts to pull-down menu actions is a well-known example of reducing the number of actions required to perform a task

b) The task should require minimal effort as well.

- **Aesthetically Pleasing:** Today, look and feel is one of the biggest USPs (Unique Selling Points) while designing software. An attractive user interface improves sales because a buyer likes to have things that look attractive. An attractive user interface makes the user feel better (as it provides ease of use) while using the product. Many software organizations focus specifically on designing software, which has an attractive look and feel so that they can attract customers/users towards their products.

7.2 User Interface Design Process

User interface design is an iterative process in which each design process occurs more than once. Each time the design step gets elaborated in more detail. Following are the commonly used interface design steps.

- During the analysis step define interface objects and corresponding actions or operations.
- Define the major events in the interface. These events depict the user's actions. Finally, model these events.
- Analyse how the interface looks from the user's point of view.
- Identify how the user understands the interface with the information provided along with it.

The user interface design activity starts with the identification of the user, task and environmental requirements. After this, user states are created and analysed to define a set of interface objects and actions. These objects then form the basis for the creation of screen layouts, menus, icons and much more.

While designing the user interface, the following points must be remembered:

- Follow the rules stated in section 4.7.1 Any interface that fails to achieve any of these rules to a reasonable degree needs to be redesigned.
- Determine how the interface will be implemented.
- Consider the environment (like operating system, development tools, display properties and so on).

8. FUNDAMENTAL DESIGN CONCEPTS

In today's modern software systems, software design plays a key role in the development of software products; however, it is only one phase of the complete software engineering life cycle. Software design was introduced as a systematic and intelligent process for generating, evaluating and specifying designs for devices, systems or processes. Software designs provide blueprints that capture how software systems meet their required functions and how they are shaped to meet their planned quality. Formally, software engineering design is defined as the process of identifying, evaluating, validating and specifying the architectural, detailed, and construction models required to build software that meets its intended functional and non-functional requirements. The term software design is used interchangeably in practice as a means to describe both the process and product of software design. From a process perspective, software design is used to identify the phase, activities, tasks and interrelationship between them required to model the software's structure and behavior before construction begins. From a product development perspective, software design is used to identify the design artifacts that result from the identified phase, activities, and tasks, therefore, these products by themselves or collectively, are referred to as a software design. Design products vary according to several factors, including design perspective, language, purpose and their capabilities for evaluation and analysis. For example, designs can be in architectural form, using architectural notations targeted at specific stakeholders. These types of design can be presented using block diagrams, Unified Modeling Language (UML) diagram. In other cases, design can be in detailed form, where the system is used to model structural and behavioural aspects. These can include software models that contain class diagrams, object diagrams, sequence diagrams or activity diagrams. Other design products include models that represent interfaces, data or user interface designs. During the software design phase, the system is decomposed to allow optimum development of the software, requirements are mapped to conceptual models of the operational software, roles are assigned to software teams on the same or remote sites, well-known interfaces for software components are created, quality attributes are addressed and incorporated into the design of the system; the user interface is created, the software capability is analyzed, functions and variable names are identified, design documentation goals are established and the foundation for the rest of the software engineering life cycle is also established.

8.1 Module

Software architecture and design patterns represent modularity, that is software is divided into named and addressable components, sometimes called modules that are integrated to satisfy problem requirements. It has been stated that "modularity is the single attribute of software that allows a program to be intellectually manageable". Monolithic software (i.e., a large program composed of a single module) cannot be easily grasped by a software engineer. The number of control paths, span of reference, number of variables and overall complexity would make understanding close to impossible. To illustrate this point, consider the following argument based on observations of human problem-solving. Consider two problems, p1 and p2. If the perceived complexity of p1 is greater than the perceived complexity of p2, it follows that the effort required to solve p1 is greater than the effort required to solve p2. As a general case, this result is intuitively obvious. It does take more time to solve a difficult problem. It also follows that the perceived complexity of two problems when they are combined is often greater than the sum of the perceived complexity when each is taken separately. This leads to a "divide and conquer" strategy – it's easier to solve a complex problem when you break it into manageable pieces. This has important implications with regard to modularity and software. It is possible to conclude that, if we subdivide software indefinitely, the effort required to develop it will become negligibly small. The effort (cost) to develop an individual software module does decrease as the total number of modules increases. Given the same set of requirements, more modules mean smaller individual sizes. However, as the number of modules grows, the effort (cost) associated with integrating the modules also grows. We can modularize a design (and the resulting program) so that development can be more easily planned, software increments can be defined and delivered, changes can be more easily accommodated, testing and debugging can be conducted more efficiently and long–term maintenance can be conducted without serious side effects.

8.2 Modularization

Modularization is one of the most important design principles in software design. Modularity allows software systems to be manageable at all levels of the development life cycle. That is, the work products of the requirements, design, construction and testing efforts can all be modularized to efficiently carry out the operations. In the design phase, modularization is the principle that drives the continuous decomposition of the software system until fine-grained components are created. Modularization plays a key role during all design activities, including software architecture and detailed construction design, when applied effectively, it provides a roadmap for software development starting from coarse-grained components that are further modularized into fine-grained components directly related to code. If applied properly, modularization can lead to designs that are easier to understand, resulting in systems that are easier to develop and maintain. Efficient modularization can be achieved by following and applying the principles of abstraction and encapsulation. With proper modularization, software systems can be decomposed into modules that allow the complexity of the system to be manageable and allow the system to be efficiently built, maintained and reused.

9. DESIGN TECHNIQUES

A more methodical approach to software design is purposed by structured methods, which are sets of notations and guidelines for software design. Budgen (1993) describes some of the most commonly used methods such as structured design, structured systems analysis, Jackson System Development and various approaches to object-oriented design. The use of structured methods involves producing large amounts of diagrammatic design documentation. CASE tools have been developed to support particular methods. In many projects, the structured methods are applied successfully. As they use standard notations and produce standard design documents, they can reduce the cost. Irrespective of who applies the mathematical method, they always produce the same results. The term structured methods suggest that from the specification, the designers should generate designs which are similar to each other. Notations set of activities, rules, guidelines and report formats are included by the structured method.

The following models are supported by the structured method:

(1) **A Data-Flow Model**: In this model, by using the transformations in the data the system will be modelled.

(2) **An Entity-Relation Model:** The logical data and structures used are described by this model.

(3) **A Structural Model:** Documents the components of the system and interactions between them.

(4) If the method is **Object-Oriented** it will include an inheritance model of the system, a model of how objects are composed of other objects and, usually, an object-use model which shows how objects are used by other objects. The other system models like state transition diagrams, and entity life histories enhance the above models. Most methods suggest a centralized repository for system information or a data dictionary should be used. No one method is better or worse than other methods: the success or other methods often depends on their suitability for an application domain.

10. SUMMARY

- Software design is the process of transforming functional and non-functional requirements into models that describe the technical solution before development begins.
- There are primarily three main types of notations used in the design document: graphical notations, program description languages, and informal text
- The data design is the model of data that is represented at a high level of abstraction. Various elements of data design are data objects, databases and data warehouses.
- Architectural design comes before detailed system specification, it should not include any design information. Architectural design is necessary to structure and organize the specification.
- Architectural design can be represented using various models, like a structural model, framework model, dynamic model, process model and functional model.
- Component-level design is created by transforming the structural elements defined by the software architecture into procedural descriptions of software components.
- A component, also known as a module, resides within the software architecture and serves one of the following three roles: A control component, A problem domain component and an infrastructure component.
- Component-level design is used to define the data structures, algorithms, interface description and communication mechanisms allocated to each module.
- User interface determines how users interact with the software. The user interface design creates an effective communication medium between a human and a computer machine.
- In the design phase, modularization is the principle that drives the continuous decomposition of the software system until fine-grained components are created.
- Various models that are supported by the structured method are A data-flow model, an entity-relation model, structural model, If the method is object-oriented it will include an inheritance model of the system

11. SELF-ASSESSMENT QUESTIONS

SELF-ASSESSMENT QUESTIONS – 1

1. ___________ is the process of transforming functional and non-functional requirements into models that describe the technical solution before development begins.
2. Mention various types of notations used in the design document.
3. Mention any one approach in which the software design problem can be solved.
4. A general model of a software design is a ________.
5. The data design is the model of data that is represented at the high level of_____________.
6. Mention any two elements of data design.
7. The ________ explicitly represents various data objects, the relationships among them and the constraints on the elements of data structures.
8. _________ is necessary to structure and organize the specification.
9. Mention any two functions of architectural design.
10. ________ model shows architecture as a systematic collection of program components. 11. A component is also called as ________
11. ________ design is used to define the data structures, algorithms, interface description and communication mechanisms allocated to each module.
12. ________determine the way in which users interact with the software.
13. ________, ________ and ________ are the rules for software engineering to achieve a good user interface.
14. User interface design is an ________ process in which each design process occurs more than once.
15. __________ plays a key role in the development of software products.
16. _________ allows software systems to be manageable at all levels of the development life cycle.

SELF-ASSESSMENT QUESTIONS – 1

17. In the design phase, ___________ principle drives the continuous decomposition of the software system until fine-grained components are created.
18. Mention any two models that are supported by structured methods.
19. The transformations in the data the system will be modelled by _________ model.
20. The logical data and structures used are described by __________ model.

12. SELF-ASSESSMENT ANSWERS

1. Software design
2. Graphical notations, Program Description Languages, Informal Text
3. Detailed study and understanding of the problem
4. Directed graph.
5. Abstraction
6. Data objects, databases
7. data dictionary
8. Architectural design
9. It provides a level of abstraction at which the software designers can specify the system behaviour (such as function and performance). It evaluates all top-level designs.
10. Structural
11. Module
12. Component-level
13. User interface
14. Ease of learning, efficiency of use and aesthetic appeal
15. Iterative
16. software design
17. Modularity
18. Modularization
19. A data-flow model and structural model
20. Data-flow

21. Entity-relation

13. TERMINAL QUESTIONS

1. Explain the design process with the help of a diagram.
2. List the various guidelines for data design.
3. List various functions of architectural design.
4. Explain architectural design representation.
5. Write a note on:
   a) Component-level design
   b) User Interface Design
   c) Design Techniques

14. TERMINAL ANSWERS

1. A general model of a software design is a directed graph. The target of the design process is the creation of a graph without any inconsistencies. (Refer to section 4.3)
2. Guidelines for data design are: Apply systematic analysis on data: represent data objects, relationships among them and data flow along with the contents. (Refer to section 4.4)
3. Various functions of architectural design are: It provides a level of abstraction at which the software designers can specify the system behaviour (such as function and performance). (Refer to section 4.5)
4. Architectural design can be represented using various models, such as the Structural model: Which shows architecture as a systematic collection of program components. (Refer to section 4.5)
5. a) Component-level design is created by transforming the structural elements defined by the software architecture into procedural descriptions of software components. (Refer to section 4.6)

   b) User interface determines the way in which users interact with the software. (Refer to section 4.7)

   c) A more methodical approach to software design is purposed by structured methods, which are sets of notations and guidelines for software design. (Refer to section 4.9)