# 

**Iqra National University Peshawar Pakistan**

**Department of Computer Science**

SpringSemester,Assignment,May-June2020

# 

**Rubina nizam**

**Roll no:12118**

|  |  |  |  |
| --- | --- | --- | --- |
| Q.1 |  | Study how many methods for designing of software have been found out in the literature and which can be chosen for designing? Explain  Answer  **A Brief Survey of Software Design Approaches**  When a software designer looks at software problems that need to be solved, he will group those that have similar characteristics together. This grouping is called a problem domain. And for each software design methodology there is a group of problems for which it is well-suited called an application domain. The application domain is ill-defined, as such, it is impossible to make comparison amongst the software design methodologies, comparing each against another. However, some scheme of classification will help in discussing particular methodologies.  A few criteria can be used to classify methodologies such as the characteristics of the systems to be designed, such as the type of software representation and how formal it is. Other types of criteria are the characteristics of the software structure, whether it is hierarchical or level or it is functionally modular. Another criterion that can be used is the characteristic of the design process. However, the basis of the methodology is often the best guide to classification. For example, a current classification scheme for real-time systems developed at the Software Engineering Institute (Ford, 1991, p53) is based on the three distinct views of a system. It can be summarized as follows:  "The basic view of the system taken by a design method, and hence captured by a design based on that method, can be functional, structural, or behavior. With the functional view, the system is considered to be a collection of components, each performing a specific function, and each function directly answering a part of the requirement. The design describes each functional component and the manner of its interaction with the other components. With the structural view, the system is considered to be a collection of components, each of a specific type, each independently buildable and testable, and able to be integrated into a working whole. Ideally, each structural component is also a functional component. With the behavioral view, the system is considered to be an active object exhibiting specific behaviors, containing internal state, changing state in response to inputs, and generating effects as a result of state changes."  Original detailed discussion was first published by Firth, Wood, Pethia, Roberts, Mosley, and Dolce (1987) and then Wood and Pethia (1988), another subsequent report was published by Wood (1989).  Dividing software design methodologies into classifications (called approaches) helps in the generalization, explanation and understanding of software design methodologies, and guide in the selection of the appropriate software design methodology to use. Details of software design approaches can vary greatly. Some consist of a set of guidelines, while others include strict rules and a set of coordinated diagrammatic representation. The approaches that have been proposed for software design are driver’s e. Many of these approaches are really full-fledged software design methods, in that they are composed of a set of techniques directed at and supporting a common, unifying rationale. The main design approaches (Pressman, 1992; Peters, 1981) are: the level oriented, data flow oriented, data structure oriented and the object oriented design approaches.  Different approaches have been taken to develop software solutions for different problems. However, in some problems, different approaches have been integrated or combined in a logical manner to derive a software solution. Thus, the different software design approaches are not necessarily mutually exclusive. For example, designers have used the leveling approach of top-down decomposition to break down a large complex system into smaller, more manageable modules and then use other approaches to design the software for each module. Depending on the criteria used, there are a number of ways to define the classification of software design approaches. The classification of software design approaches in this section is based on the basis of the approach, characteristics of the design process and the type of software constructs created to develop the solution.  **Level-Oriented Design**  In the level-oriented design approach, there are two general or broad strategies that can be used. The first strategy starts with a general definition of a solution to the problem then through a step-by-step process produce a detailed solution (this is called Stepwise Refinement). This is basically dependent on the system requirements and is a top-down process. The other strategy is to start with a basic solution to the problem and through a process of modeling the problem, build up or extend the solution by adding additional features (this is called design by composition).  The top-down process starts at the top level and by functional decomposition, breaks down the system into smaller functional modules. Smaller modules are more readily analyzed, easier to design and code. But, inherent in the top-down process is the requirement that there must be a complete understanding of the problem or system at hand. Otherwise, it could lead to extensive redesign later on. The top-down process also is dependent on decisions made at the early stages to determine the design structure. Different decisions made at the early stage will result in different design structures. Functional decomposition is an iterative "break down" process called stepwise refinement, where each level is decomposed to a more detailed lower level. Thus, at each decomposition, there have to be a way to determine if further decomposition is needed or necessary, that is, if the atomic level has been achieved. There are no inherent procedure or guidelines for this. There is also a possibility of duplication if stepwise refinement is not done carefully or "correctly"; this will occur toward the end of the process, that is, at the lower levels. This can be costly, especially if there are many different designers or programming teams working on a single system. As a result, the top-down process is often used in the initial phase of the design process to break down the different components or modules of a system. The top-down process has also been used as a preliminary step in the other design methodologies. Once the modules of the system have been determined, they can be divided amongst the different designers or design teams.  The design by composition strategy involves the evolution of a solution by building upon the solution from the previous stage. Using this technique, additional features are added as the solution evolves. This strategy uses as its origin, the basic or simple initial solution and through an iterative composition process add or expand the solution to include additional modules. This approach will also encompass the bottom-up design, where the lowest level solution is developed first and gradually builds up to the highest level. Freeman (1983) has added a few models, such as the outside-in model where what the end-users sees (external functions of the system) are defined as the top-level decisions and the implementation (the inside of the system) as the lower-level decisions. This model was created to overcome the tendency of designers to pay insufficient attention to the needs of end-users. The alternative to the outside-in model is the inside-out model, where decisions relating to the implementation (inside) of the system are made before the external function of the system. Another model is based on the most-critical-component-first approach, where one first design the components of the systems that are the most constrained so that these critical parameters are satisfied. Then the rest of the system components are designed. Often these models are conceptual, not to be rigorously enforced because in a real design effort, integration of models is often necessary.  **Data Flow-Oriented Design**  In the data flow-oriented design approach, which is often called Structured Design, information flow characteristic is used to derive program structure. In the data flow-oriented approach, emphasis is on the processing or operations performed on the da ta. Design is information driven. Information maybe represented as a continuous flow that is transformed, as it is processed from node to node in the input-output stream. As software can ideally be represented by a data flow diagram (DFD), a design mod el that uses a DFD can theoretically be applied in the software development project. The data flow-oriented approach is especially applicable when information is processed without hierarchical structure. A DFD can be mapped into the design structure by two means - transform analysis or transaction analysis. Transform analysis is applied when the data flow in the input-output stream has clear boundaries. The DFD is mapped into a structure that allocates control to three basic modules - input, process and output. Transaction analysis is applied when a single information item causes flow to branch along one of many paths. The DFD is mapped to a substructure that acquires and evaluates a transaction; another substructure controls all the data processing actions based on a transaction. A few examples of structured design or data flow-oriented design methodologies are Structured Analysis and Design Technique (SADT), Systematic Activity Modeling Method (SAMM) and Structured Design (SD).  **Data Structure-Oriented Design**  The data structure-oriented design approach utilizes the data structures of the input data, internal data (for example databases) and output data to develop software. In the data structure-oriented approach, the emphasis is on the object, which is the data. The structure of information, called data structure, has an important impact on the complexity and efficiency of algorithms designed to process information.  Software design is closely related to the data structure of the system, for example, alternative data will require a conditional processing element, repetitive data will require a control feature for repetition and a hierarchical data structure will require a hierarchical software structure. Data structure-oriented design is best utilized in applications that are a well-defined, hierarchical structure of information.  As both data flow and data structure oriented design approaches are based on considerations in the information domain, there are similarities between both approaches. Both depend on the analysis step to build the foundation for later steps. Both attempt to transform information into a software structure; both are driven by information. In data structure-oriented design information structure are represented using hierarchical diagrams; DFD has little relevance; transformation and transaction flows are not considered. Data structure-oriented design have a few tasks - evaluate the characteristics of the data structure, represent the data in its lowest form such as repetition, sequence or selection, map the data representation into a control hierarchy for software, refine the control hierarchy and then develop a procedural description of the software. Some examples of the data structure-oriented design approach are the Jackson System Development (JSD) and the Data Structured Systems Development (DDSD) which is also called the Warnier-Orr methodology.  **Object Oriented Design**  The object oriented design approach is unique in its usage of the three software design concepts: abstraction, information hiding and modularity. Objects are basically a producer or consumers of information or an information item. The object consists o f a private data structure and related operations that may transform the data structure. Operations contain procedural and control constructs that may be invoked by a message, that is, a request to the object to perform one of its operations. The object also has an interface where messages are passed to specify what operation on the object is desired. The object that receives a message will then determine how the requested operation is to be performed. By this means, information hiding (that is, the details of implementation are hidden form all the elements outside the object) is achieved. Also objects and their operations are inherently modular, that is, software elements (data and process) are grouped together with a well-defined interface mechanism (that is, messages).  Object oriented design is based on the concepts of: objects and attributes, classes and members, wholes and parts. All objects encapsulate data (the attribute values that define the data), other objects (composite objects can be defined), constants (set values), and other related information. Encapsulation means that all of this information is packaged into a single name and can be re-used. The object oriented design is rather new and as such it is still evolving even at this present moment. Object oriented design encompasses data design, architectural design and procedural design. By identifying classes and objects, data abstractions are created; by coupling operations to data, modules are specified and a structure for the software is established; by developing a mechanism for using objects (for example, passing of messages) interfaces are described. |  |

☺\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*☺