Iqra National University Peshawar Pakistan

Department of Computer Science

Spring Semester, Assignment, May/June 2020

Subject:	**Software Design**	Issue Date:	**15/May/2020**
Program:	**MS (CS & SE)**	Submission Date:	**10/June/2020**
Teacher Name:	**Dr. Fazal-e-Malik**		
Student Name	**Rooh Ullah Jan**	ID	**6611**

The Software Design Methodology

Many software development projects have been known to incur extensive and costly design errors. The most expansive errors are often introduced early in the development process. This underscores the need for better requirement definition and software de sign methodology. Software design is an important activity as it determines how the whole software development task would proceed including the system maintenance. The design of software is essentially a skill, but it usually requires a structure which will provide a guide or a methodology for this task. A methodology can be defined as the underlying principles and rules that govern a system. A method can be defined as a systematic procedure for a set of activities. Thus, from these definitions, a methodology will encompass the methods used within the methodology. Different methodologies can support work in different phases of the system life cycle, for example, planning, analysis, design and programming, testing and implementation. Svoboda (1990) developed the idea of a methodology further by proposing that there should be at least four components:

1. a conceptual model of constructs essential to the problem,
2. a set of procedure suggesting the direction and order to proceed,
3. a series of guidelines identifying things to be avoided, and
4. a collection of evaluation criteria for assessing the quality of the product.

The conceptual model is needed to direct or guide the designers to the relevant aspects of the system. The set of procedure provides the designer a systematic and logical set of activities to begin the design task. The evaluation criteria provide an objective measurement of the work done against some established standard or specifications.

A software design methodology can be structured as comprising of the software design process component and the software design representation or diagrammatic component. The process component is based on the basic principles established in the methodology while the representation component is the "blueprint" from which the code for the software will be built. It should be noted, that in practice, the design methodology is often constrained by existing hardware configuration, the implementation language, the existing file and data structures and the existing company practices, all of which would limit the solution space available to develop the software. The evolution of each software design needs to be meticulously recorded or diagramed, including the basis for choices made, for future walk-throughs and maintenance.

The Design Process

Design is a formation of a plan of activities to accomplish a recognized need. The need may be well defined or ill defined. When needs are ill-defined, it is likely due to the fact that neither the need nor problem has been identified. The design process is a process of creative invention and definition, it involves synthesis and analysis, and thus, is difficult to summarize in a simple design formula. Design is an applied science. In a software design problem, a number of solutions exist. The designer (each word "designer" can also refer to "designers") must plan and execute the design strategy taking into account certain established design practices. The designer often has to fall back on previous experience gained and has to study the existing software methodologies designed by others, to analyze their advantages and disadvantages. It is useful also to review the basic parameters, especially the requirements and system specifications. A designer must constantly improve and enrich his store of design solution. The development of design alternatives should be a regular design activity aimed at seeing the most rational solution. In the design of software, often there are different design methodologies that can be used to derive a software solution, this is called the design degree of freedom. A design solution can also have several degrees of freedom, which implies that there is possibly at least one solution in the solution space, often there is more than one solution. It must be noted that there are no unique answers for design. However, that does not imply that any answer will do. Some solutions are more optimal than others. A design is subject to certain problem-solving constraints, for example, in a vacation design problem, the constraints are money and time. Note also that there are constraints on the solutions, for example, users must be satisfied. Then in synthesis, it is necessary to begin with the solution of the main design problems and separate secondary items from the main ones. Successful design often begins with a clear definition of the design objectives. While in analysis and evaluation, the designer using these knowledge and simple diagrams, makes a first draft of the design in the form of diagrams. This has to be thoroughly analyzed to verify that it meets the design objectives and that it is adequate but not over designed. A systematic approach is useful only to the extent that the designer is presented with a strategy that he can use as a base for planning the required design strategy for the problem at hand. A design problem is not a theoretical construct. Design has an authentic purpose - the creation of an end result by taking definite action or the creation of something having physical reality. The design process is by necessity an iterative one. The software design should describe a system that meet the requirements. It should be thorough, in-depth, complete and use standardized notations to depict the structure and systems. According to Ford (1991, p. 52):

"The process of design consists of taking a specification (a description of what is required) and converting it into a design (a description of what is to be built). A good design is complete (it will build everything required), economical (it will not build what is not required), constructive (it says how to build the product), uniform (it uses the same building techniques throughout), and testable (it can be shown to work)."

Most software engineers agreed that software design integrates and utilizes a great deal of the basics of computer science and information systems; unless one can apply the knowledge gained to the design of quality software, one cannot truly be considered to have mastered software design.

Software design methodology provides a logical and systematic means of proceeding with the design process as well as a set of guidelines for decision-making. The design methodology provides a sequence of activities, and often uses a set of notations or diagrams. The design methodology is especially important for large complex projects that involve programming-in-the-large (where many designers are involved); the use of a methodology establishes a set of common communication channels for translating design to codes and a set of common objectives. In addition, there must be an objective match between the overall character of the problem and the features of the solution approach, in order for a methodology to be efficient.

In all the different models, design plays a central role in the models. Software design is thus a major phase in the software system development. In this research project, the design process of the SDLC will be considered, which includes requirement definition or system analysis, system or requirement specifications, logical or system design, and detailed or program design and development. Even though pure software design consists of architectural and detailed design. Pure software design cannot proceed without the requirement definition and specification stages. Architectural design deals with the general structure of a software system. It involves identifying and decomposing the software into smaller models or components, determining data structures and specifying the relationship among the modules. According to a Software Engineering Institute report (Budgen, 1989, p6), detailed design involves the "formulation of blueprints for the particular solution and with modeling the detailed interaction between its components." It is concerned with the detailed "how to" for packaging all the components together.

Design Approaches

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 diverse. 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 or Structured Design

In the data flow-oriented design approach, which is often called Structured Design, information flow characteristic is used to derived 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 may be 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 a nd 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 d ata. 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 have 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 of 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.

Component- Based Design

Component-based architecture focuses on the decomposition of the design into individual functional or logical components that represent well-defined communication interfaces containing methods, events, and properties. It provides a higher level of abstraction and divides the problem into sub-problems, each associated with component partitions.

The primary objective of component-based architecture is to ensure component reusability. A component encapsulates functionality and behaviors of a software element into a reusable and self-deployable binary unit. There are many standard component frameworks such as COM/DCOM, JavaBean, EJB, CORBA, .NET, web services, and grid services. These technologies are widely used in local desktop GUI application design such as graphic JavaBean components, MS ActiveX components, and COM components which can be reused by simply drag and drop operation.

Component-oriented software design has many advantages over the traditional object-oriented approaches such as −

- Reduced time in market and the development cost by reusing existing components.
- Increased reliability with the reuse of the existing components.

Formal Method Or Design

The mathematical methods are aimed at specification activities. They are intended for use in determining the system specification and component specification. This method Provide a mathematical specification notations and use the classic methods of mathematical proof to verify that a program is consistent with its specification.

The mathematical model performs well with respect to providing an unambiguous notation and the possibility of verification. However they do not offer many rules or heuristics for the creation of specification.

Which method is Chosen for Designing?

The structural method is best for designing because.

The structural methods are particularly useful for the specification of system structure in terms of the criteria for design method, the structural vary. The structured design approach of Yourden and Constantine (1978) offers a number of heuristics with which to Assesses a particular design based on the principles of coupling and cohesion (strength).The coupling principle Advise designer to minimize the links among modules by restricting them, whenever possible to direct calling links, and minimizing the use of common data structure.

It is interesting to note that strengths of structural methods complement the weaknesses of formal methods and vice versa.