1. INTRODUCTION OF DESIGN AND MODELING

Objectives:

Understand the concept of design and modeling in a software development process.

1. Definitions

1.1. Model

A model is a simplification with a purpose. It uses a precisely define notation to describe and simplify a complex and interesting structure, phenomenon, or relationship. One of the reasons to start by building models is actually built to find out if something will work. This means, model verification and validation is very important.

1.2. Software system model

Model of software systems helps developer visualize, communicate and validate a system before significant amounts of money are spent. Software models also help structure and coordinate the efforts of a software development team.

Some Basic characteristics of software models

1.2.1. Simplification

- Models of software are far less complex; and therefore more accessible than the actual code and components that makes up the final system.
- It's easier for a developer to work to built, extern and evaluate a visual model than to work directly in the code. When decisions are made in code, they turn to stay made but if made at the level of the model, the code can easily be maintain.
- With modeling, and especially with a visual modeling tool, decisions can be made and revised quickly and efficiently.

1.2.2. Varying perspective

A single model of a software system can describe the system form different perspectives. One view may show how major parts of the system interact and corporate. Other views might zoom in a particular subsystem part. Having all the helps the developer manages complexity, as high level views provide context and navigation. This concept is use in maps, to zoom in and get the details.

1.2.3. Common notation

A precise software model in a common notation allows developers to combine their efforts to work in parallel. As long as each contribution fits the model, the parts can be combined into the final system. Modern manufacturing uses this technique to reduce cost of software production schedule.

1.3. Modeling

Modeling actually started years ago where early civilization such as the ancient Egypt, Rome and Greece model small scale plans in art and architecture. Today, modeling is use in sciences and engineering to provide abstractions of a system at some level of precision and detail. It consists of building an abstraction of reality. Abstraction are simplification that

- They ignore irrelevant details and
- They only represent the relevant details.
  What is relevant or irrelevant depends on the purpose of the model.

Models are built and well organize to better understand the system to be developed.

1.3.1. Software Modeling

In a model – base software design, modeling is use as an essential part of the software development process. Models are build and analysis prior to the implementation of the system and are used to direct the subsequent implementation. To understand the system, you need to understand the different perspectives (views). These perspectives may be requirement view, static view and dynamic views as models of the software. These can be done using tools. A graphical modeling language that helps to develop, understand and communicate the different views. E.g UML.

OMG (**Object Modeling Group**) define modeling as designing of software application before coding.

Software modeling addresses the entire software design including interfaces, interactions with other software, and all the software methods. Software models are ways of expressing a software design. For object oriented software, an object modeling language such as UML is used to develop and express the software design. Modeling languages allows the designer to try different designs and decide with will be best for the final solution. One of the benefits of designing your software using a modeling language is that you discover problems early and fix them without refactoring your code.

1.4. Software Models and modeling

Software models are ways of expressing a software design while Software modeling addresses the entire software design including interfaces, interactions with other software, and all the software methods.

In expressing software design, usually some sort of abstract language or pictures are used to express it. For object oriented software, an object modeling language such as UML is used to develop and express the software design.

1.5. Software Design

Software design is the process of defining software methods, functions, objects, and the overall structure and interaction of your code so that the resulting functionality will satisfy your user's requirements. It is a process to conceptualize the software requirements into software implementation.

It can also be referred to as "all the activity involved in conceptualizing, framing, implementing, commissioning, and ultimately modifying complex systems" or "the activity following requirements specification and before programming"

Types of design elements

1.5.1. Data design elements

- ❖ The data design element produced a model of data that represent a high level of abstraction.
- ❖ This model is then more refined into more implementation specific representation which is processed by the computer based system.
- ❖ The structure of data is the most important part of the software design.

1.5.2. Architectural design elements

- ❖ The architecture design elements provide us overall view of the system.
- ❖ The architectural design element is generally represented as a set of interconnected subsystem that are derived from analysis packages in the requirement model

1.5.3. Interface design elements

- ❖ The interface design elements for software represent the information flow within it and out of the system.
- ❖ They communicate between the components defined as part of architecture.

1.5.4. Component level diagram elements

- ❖ The component level design for software is similar to the set of detailed specification of each room in a house.
- ❖ The component level design for the software completely describes the internal details of the each software component.
- ❖ The processing of data structure occurs in a component and an interface which allows all the component operations.
- ❖ In a context of object-oriented software engineering, a component shown in a UML diagram.
- ❖ The UML diagram is used to represent the processing logic.

1.5.5. Deployment level design elements

- ❖ The deployment level design element shows the software functionality and subsystem that allocated in the physical computing environment which support the software.

2. The importance of design and modeling in a software development process

3. Software design and architecture

A system's software architecture is the set of significant design decisions about how the software is organized to promote desired quality attributes and other properties.

A design decision might be significant for any number of reasons.

- ❖ It might represent a point of no return or influence quality attributes, schedule, or costs.

A Significant decision might be one that affects many people or forces other software systems to change. In any case, significant design decisions are costly to change later if you get them wrong.

- ❖ To promote a quality attribute means to encourage it to appear in the software system. When the architecture is well organized, it will boost the quality attributes stakeholders want and downplay or eliminate the quality attributes stakeholders don't want. One of the main components of software design is the software requirements analysis (SRA)

3.1. The Basics of Software Design

- Abstraction.
- Patterns.
- Separation of Data.
- Modularity.
- Data Hiding.
- Functional Independence.
- Refactoring.

The inputs from users and information gathered in requirement gathering phase are the inputs of this step. The output of this step comes in the form of two designs

- Logical design
- Physical 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.

4. Software Design Levels

Software design yields three levels of results: **Architectural design, High level design and Detailed design.**

4.1. **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.

4.2. **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.

4.3. **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.

5. 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 rule of divide and conquer problem solving strategy, this is because there are many other benefits attached with the modular design of a software.

5.1. Advantages of modularization

a) Smaller components are easier to maintain
b) Program can be divided based on functional aspects
c) Desired level of abstraction can be brought in the program
d) Components with high cohesion can be re-used again
e) Concurrent execution can be made possible
f) Desired from security aspects

6. Concurrency

Back in time, all software are 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 is the spell check feature in word processor is a module of software, which runs along side the word processor itself.

7. 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 a 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.

7.1. 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 –

7.1.1. **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.

7.1.2. **Logical cohesion -** When logically categorized elements are put together into a module, it is called logical cohesion.

7.1.3. **Temporal Cohesion -** When elements of module are organized such that they are processed at a similar point of time, it is called temporal cohesion.

7.1.4. **Procedural cohesion -** When elements of module are grouped together, which are executed sequentially in order to perform a task, it is called procedural cohesion.

7.1.5. **Communicational cohesion -** When elements of module are grouped together, which are executed sequentially and work on same data (information), it is called communicational cohesion.

7.1.6. **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.

7.1.7. **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.

7.2. **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 -

7.2.1. **Content coupling -** When a module can directly access or modify or refer to the content of another module, it is called content level coupling.

7.2.2. **Common coupling-** When multiple modules have read and write access to some global data, it is called common or global coupling.

7.2.3. **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.

7.2.4. **Stamp coupling-** When multiple modules share common data structure and work on different part of it, it is called stamp coupling.

7.2.5. **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.

8. Design Verification

The output of software design process is design documentation, pseudo codes, detailed logic diagrams, process diagrams, and detailed description of all functional or non-functional requirements.

The next phase, which is the implementation of software, depends on all outputs mentioned above.

It is then becomes necessary to verify the output before proceeding to the next phase. The early any mistake is detected, the better it is or it might not be detected until testing of the product. If the outputs of design phase are in formal notation form, then their associated tools for verification should be used otherwise a thorough design review can be used for verification and validation.
By structured verification approach, reviewers can detect defects that might be caused by overlooking some conditions. A good design review is important for good software design, accuracy, and quality.

9. Software design strategies

Software design is a process to conceptualize the software requirements into software implementation. Software design takes the user requirements as challenges and tries to find optimum solution. While the software is being conceptualized, a plan is chalked out to find the best possible design for Implementing the intended solution.

There are multiple variants of software design.

9.1. Structured Design

Structured design is a conceptualization of problem into several well-organized elements of solution. It is basically concerned with the solution design. Benefit of structured design is, it gives better understanding of how the problem is being solved. Structured design also makes it simpler for designer to concentrate on the problem more accurately. Structured design is mostly based on 'divide and conquer' strategy where a problem is broken into several small problems and each small problem is individually solved until the whole problem is solved. The small pieces of problem are solved by means of solution modules. Structured design emphasis that these modules be well organized in order to achieve precise solution.
These modules are arranged in hierarchy. They communicate with each other.

A good structured design always follows some rules for communication among multiple modules, namely -

- ❖ **Cohesion** - grouping of all functionally related elements.
- ❖ **Coupling** - communication between different modules. A good structured design has high cohesion and low coupling arrangements.

9.2. Function Oriented Design

In function-oriented design, the system comprises of many smaller sub-systems known as functions. These functions are capable of performing significant task in the system. The system is considered as top view of all functions. Function oriented design inherits some properties of structured design where **divide and conquer methodology is used**. This design mechanism divides the whole system into smaller functions, which provides means of abstraction by concealing the information and their operation.
These functional modules can share information among themselves by means of information passing and using information available globally.
Another characteristic of functions is that when a program calls a function, the function changes the state of the program, which sometimes is not acceptable by other modules.
Function oriented design works well where the system state does not matter and program/functions work on input rather than on a state.

9.2.1. Design Process

- ❖ The whole system is seen as how data flows in the system by means of data flow diagram.
- ❖ DFD depicts how functions change data and state of the entire system.
- ❖ The entire system is logically broken down into smaller units known as functions on the basis of their operation in the system.
- ❖ Each function is then described at large.

9.3. Object Oriented Design

Object Oriented Design (OOD) works around the entities and their characteristics instead of functions involved in the software system. This design strategies focuses on entities and its characteristics. The whole concept of software solution revolves around the engaged entities.
Let us see the important concepts of Object Oriented Design:

a) **Objects -** All entities involved in the solution design are known as objects.

For example, person, banks, company, and customers are treated as objects. Every entity has some attributes associated to it and has some methods to perform on the attributes.

b) **Classes -** A class is a generalized description of an object. An object is an instance of a class. Class defines all the attributes, which an object can have and methods, which defines the functionality of the object. In the solution design, attributes are stored as variables and functionalities are defined by means of methods or procedures.

c) **Encapsulation -** In OOD, the attributes (data variables) and methods (operation on the data) are bundled together is called encapsulation. Encapsulation not only bundles important information of an object together, but also restricts access of the data and methods from the outside world.

This is called information hiding.

d) **Inheritance -** OOD allows similar classes to stack up in hierarchical manner where the lower or sub-classes can import, implement and re-use allowed variables and methods from their immediate super classes. This property of OOD is known as inheritance. This makes it easier to define specific class and to create generalized classes from specific ones.

e) **Polymorphism -** OOD languages provide a mechanism where methods performing similar tasks but vary in arguments, can be assigned same name. This is called polymorphism, which allows a single interface performing tasks for different types. Depending upon how the function is invoked, respective portion of the code gets executed.

10. Design Process

Software design process can be perceived as series of well-defined steps. Though it varies according to design approach (function oriented or object oriented, yet It may have the following steps involved:

- ❖ A solution design is created from requirement or previous used system And /or system sequence diagram.
- ❖ Objects are identified and grouped into classes on behalf of similarity in attribute characteristics.
- ❖ Class hierarchy and relation among them is defined. Application framework is defined.

11. Software Design Approaches

Here are two generic approaches for software designing:

11.1. Top Down Design

We know that a system is composed of more than one sub-systems and it contains a number of components. Further, these sub-systems and components may have their own set of sub-systems and components, and creates hierarchical structure in the system.

Top-down design takes the whole software system as one entity and then decomposes it to achieve more than one sub-system or component based on some characteristics. Each sub-system or component is then treated as a system and decomposed further. This process keeps on running until the lowest level of System in the top-down hierarchy is achieved.

Top-down design starts with a generalized model of system and keeps on defining the more specific part of it. When all the components are composed the whole system comes into existence.

Top-down design is more suitable when the software solution needs to be designed from scratch and specific details are unknown.

11.2. Bottom-up Design

The bottom up design model starts with most specific and basic components. It proceeds with composing higher level of components by using basic or lower level components. It keeps creating higher level components until the desired system is not evolved as one single component. With each higher level, the amount of abstraction is increased.

Bottom-up strategy is more suitable when a system needs to be created from some existing system, where the basic primitives can be used in the newer system.

Both, top-down and bottom-up approaches are not practical individually. Instead,
a good combination of both is used.