MCSE 544 Software Design and Integration

Chapter 1: Software Design

Dr. Mehedi Hasan
Stamford University Bangladesh

Exam Related

- Midterm: 9th September
- Final Exam: 25th November

Outline

- Software Processes
- SDLC
- Software Design
- Modularization, concurrency, coupling and cohesion
- Design Verification
- Software Design Strategies
- Software Design Approaches
- UML

Overall Software Development Process and Phases

Software Development Life Cycle (SDLC)

A software development life cycle (SDLC) is a series of identifiable stages that a software product undergoes during its lifetime.

A software development life cycle model (also called process model) is a descriptive and diagrammatic representation of the software life cycle. A life cycle model represents all the activities required to make a software product transit through its life cycle phases. It also captures the order in which these activities are to be undertaken.

In other words, a software development life cycle model maps the different activities performed on a software product from its inception to retirement. Different life cycle models may map the basic development activities to phases in different ways. Thus, no matter which life cycle model is followed, the basic activities are included in all life cycle models though the activities may be carried out in different orders in different life cycle models.

SDLC Phases

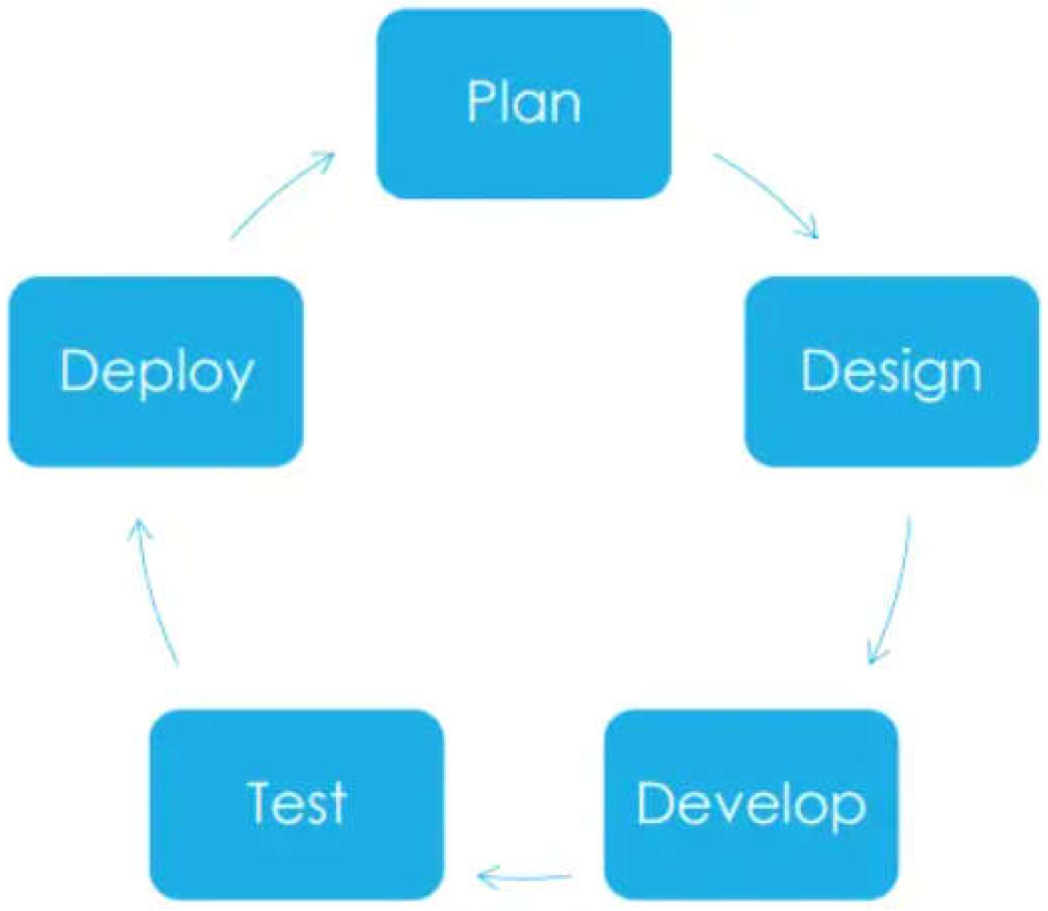


Figure 1 - Software Development Cycle

Why SDLC important?

- There is always a huge temptation to implementation of software without planning or designing specially for the small or medium-sized project.
- Programmers tend to argue that the time that spends on planning is already good enough for them to do the programming work and deliver the product.
- Management also tends to focus on efficiency and make use of the least amount of resource to get the same result however there is a certain reason to explain why we need software development life cycle
- The development team must identify a suitable software development life cycle model for the particular project and then adhere to it.
- When a software product is being developed by a team, there must be a clear understanding among team members about when and what to do.

Suppose a software development problem is divided into several parts and the parts are assigned to the team members. From then on, suppose the team members are allowed the freedom to develop the parts assigned to them in whatever way they like. It is possible that one member might start writing the code for his part, another might decide to prepare the test documents first, and some other engineer might begin with the design phase of the parts assigned to him. This would be one of the perfect recipes for project failure.

Importance of Software Development Life Cycle

•Without using a particular life cycle model, the development of a software product would **not be in a systematic and disciplined manner**.

•A software development life cycle model **defines entry and exit criteria for every phase**. A phase can start only if its phase-entry criteria have been satisfied. So without a software life cycle model, the entry and exit criteria for a phase cannot be recognized.

•Without software development life cycle models, it becomes **difficult** for software project managers to **monitor the progress** of the project.

Reason 1: Quality Assurance and Quality Control

•The definition of quality assurance is a set of activities for ensuring quality in the process of product development.

•Meanwhile, the definition of quality control is a set of activities for ensuring the quality of the developed product.

•While QA aims at prevention defect by focusing on the process of product development. QC is of identifying the defects by examining the finished product.

Reason 2:
Easier implementation control

•Within the five core stages in sdlc, multiple documentation should be prepared to give guidelines and instruction for the programmer and the tester to follow and for the management and approve words to be at least and approve on the activities and action were taken.

•Business case, software requirement specification, design documentation specification, functional specification, test plan, deploy deployment plan etc are all well defined at each stage.

Reason 3: Fulfill user requirements or even exceeding the expectation

Software 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.

Software Design Levels

- Software design yields three levels of results:

- **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.

- **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.

- **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.

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 rules of ‘divide and conquer’ problem-solving strategy this is because there are many other benefits attached with the modular design of a software.

Advantage of modularization:

- • Smaller components are easier to maintain
- • Program can be divided based on functional aspects
- • Desired level of abstraction ca n be brought in the program
- • Components with high cohesion can be re-used again.
- • Concurrent execution can be made possible
- • Desired from security aspect

Concurrency

- Back in time, all software's were 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.

Concurrency (Cont.)

- 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:
- The spell check feature in word processor is a module of software, which runs alongside the word processor itself.

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 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.

Cohesion and Types

- 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 –

- **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.

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

Cohesion Types

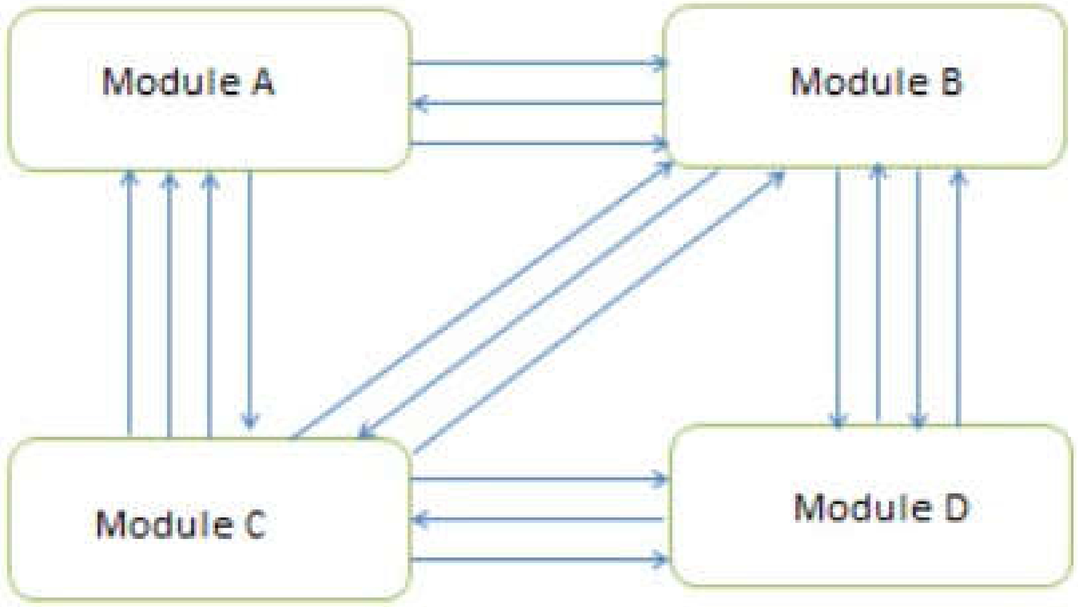
- **Temporal Cohesion -** When elements of module are organized such that they are processed at a similar point in time, it is called temporal cohesion.
- **Procedural cohesion -** When elements of module are grouped together, which are executed sequentially in order to perform a task, it is called procedural cohesion.
- **Communicational cohesion -** When elements of module are grouped together, which are executed sequentially and work on same data (information), it is called communicational cohesion.
- **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.
- **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.

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.
- Coupling is the measure of the independence of components. It defines the degree of dependency of each module of system development on the other. In practice, this means the stronger the coupling between the modules in a system, the more difficult it is to implement and maintain the system.
- Each module should have simple, clean interface with other modules, and that the minimum number of data elements should be shared between modules.

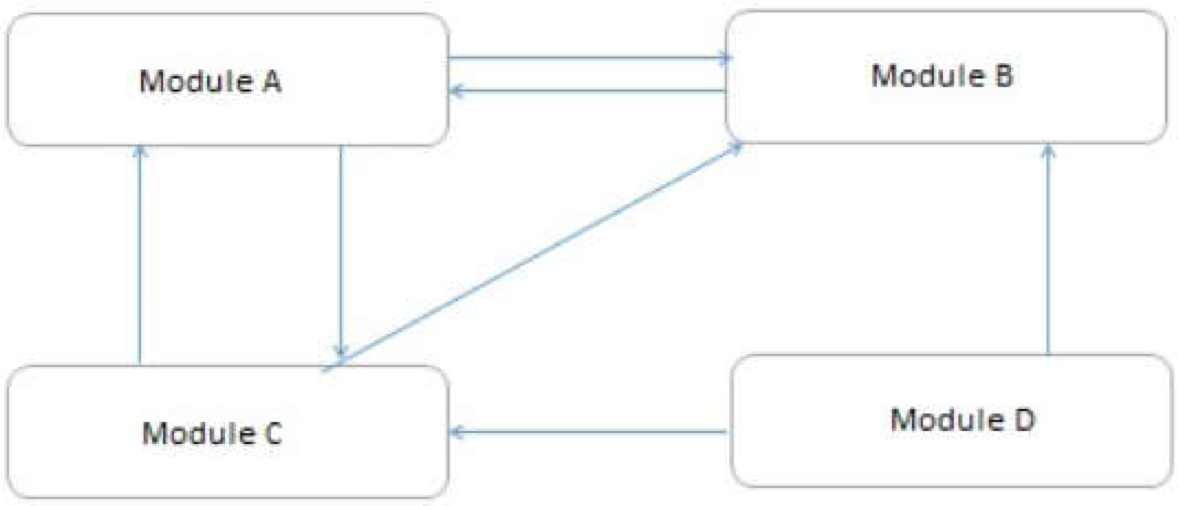
High Coupling

- These type of systems have interconnections with program units dependent on each other. Changes to one subsystem leads to high impact on the other subsystem.



Low Coupling

- These type of systems are made up of components which are independent or almost independent. A change in one subsystem does not affect any other subsystem.



Coupling Measure Types

- There are five levels of coupling, namely –
- **Content coupling** - When a module can directly access or modify or refer to the content of another module, it is called content level coupling.
- **Common coupling-** When multiple modules have read and write access to some global data, it is called common or global coupling.

Coupling Types

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

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.

Design Verification

- 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.

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. Let us study them briefly:

Variants of Software Design

A. Structured Design
B. Function Oriented Design
C. Object Oriented Design

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.

Structured Design Process

- 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.

Function Oriented Design

- In function-oriented design, the system is comprised 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.
  - 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.

FOD Process

- The whole system is seen as how data flows in the system by means of data flow diagram.
- DFD depicts how functions change the data and state of 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.

Object Oriented Design

- Object oriented design works around the entities and their characteristics instead of functions involved in the software system. This design strategy 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:
  - 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.
  - 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.

Pillars of OOD

- 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.
- 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.

Pillars of OOD

- 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.
- Data Abstraction - Abstraction is the process of hiding the internal details of an application from the outer world. Abstraction is used to describe things in simple terms. It’s used to create a boundary between the application and the client programs.

OOD 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 are defined.
  - Application framework is defined

Pillars of OOD

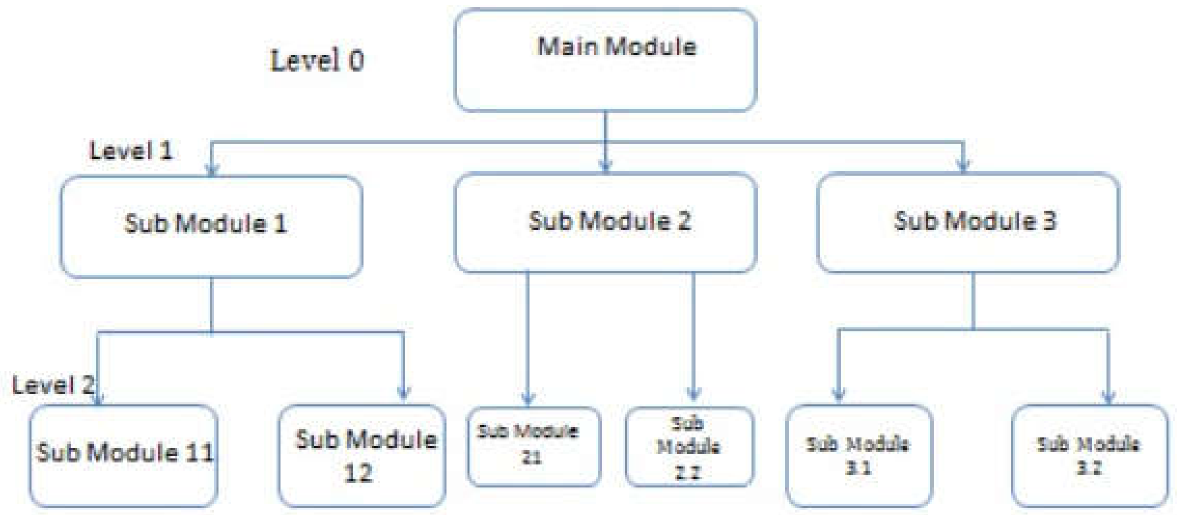
- Polymorphism - OOD languages provide a mechanism where methods performing similar tasks but vary in arguments, can be assigned same name. This is called polymorphism, which allows a single interface performing tasks for different types. Depending upon how the function is invoked, respective portion of the code gets executed.
- Data Abstraction - Abstraction is the process of hiding the internal details of an application from the outer world. Abstraction is used to describe things in simple terms. It’s used to create a boundary between the application and the client programs.

Software Design Approaches

- Top Down Design
- Bottom Up Design

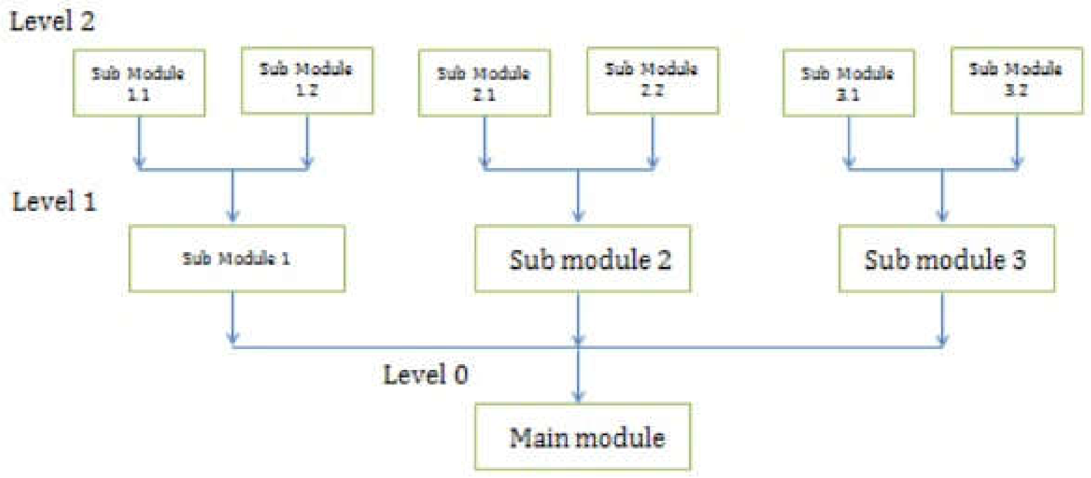
Top Down Design

- The top-down strategy uses the modular approach to develop the design of a system. It is called so because it starts from the top or the highest-level module and moves towards the lowest level modules.
- In this technique, the highest-level module or main module for developing the software is identified. The main module is divided into several smaller and simpler submodules or segments based on the task performed by each module. Then, each submodule is further subdivided into several submodules of next lower level. This process of dividing each module into several submodules continues until the lowest level modules, which cannot be further subdivided, are not identified.

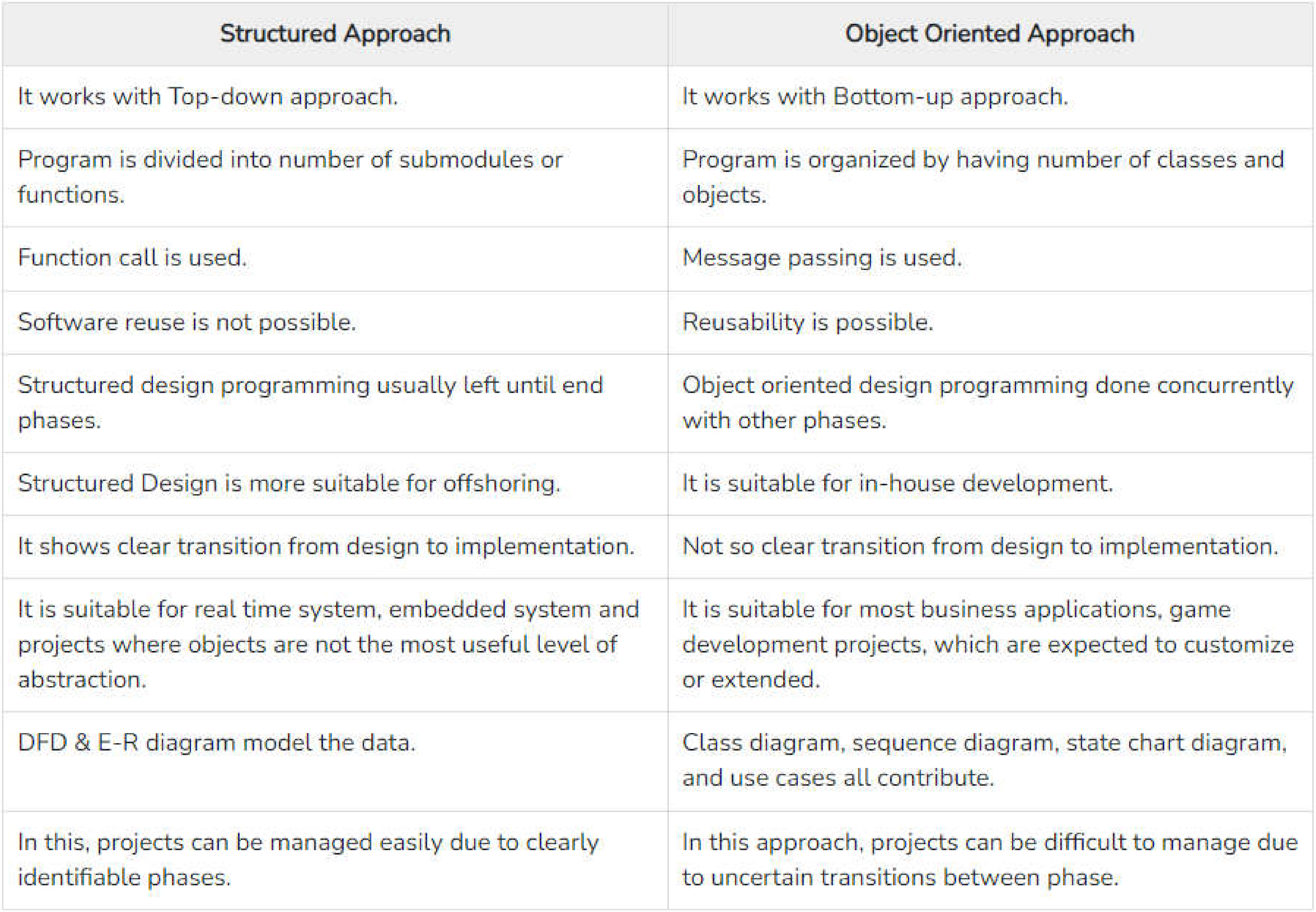


Bottom Up Design

- Bottom-Up Strategy follows the modular approach to develop the design of the system. It is called so because it starts from the bottom or the most basic level modules and moves towards the highest level modules.
  - In this technique, The modules at the most basic or the lowest level are identified.
  - These modules are then grouped together based on the function performed by each module to form the next higher-level modules.
  - Then, these modules are further combined to form the next higher-level modules.
  - This process of grouping several simpler modules to form higher level modules continues until the main module of system development process is achieved.



Structured Approach Vs. Object-Oriented Approach



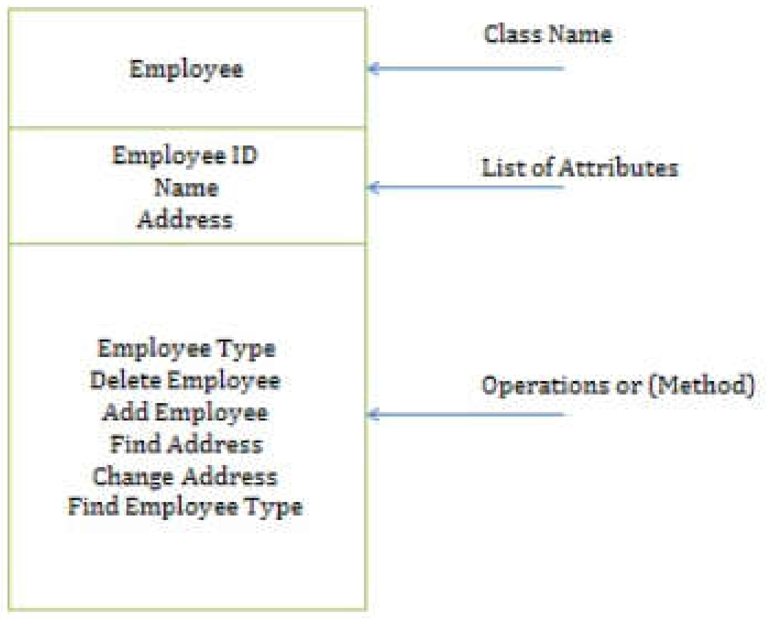
Structured Approach	Object Oriented Approach
It works with Top-down approach.	It works with Bottom-up approach.
Program is divided into number of submodules or functions.	Program is organized by having number of classes and objects.
Function call is used.	Message passing is used.
Software reuse is not possible.	Reusability is possible.
Structured design programming usually left until end phases.	Object oriented design programming done concurrently with other phases.
Structured Design is more suitable for offshoring.	It is suitable for in-house development.
It shows clear transition from design to implementation.	Not so clear transition from design to implementation.
It is suitable for real time system, embedded system and projects where objects are not the most useful level of abstraction.	It is suitable for most business applications, game development projects, which are expected to customize or extended.
DFD & E-R diagram model the data.	Class diagram, sequence diagram, state chart diagram, and use cases all contribute.
In this, projects can be managed easily due to clearly identifiable phases.	In this approach, projects can be difficult to manage due to uncertain transitions between phase.

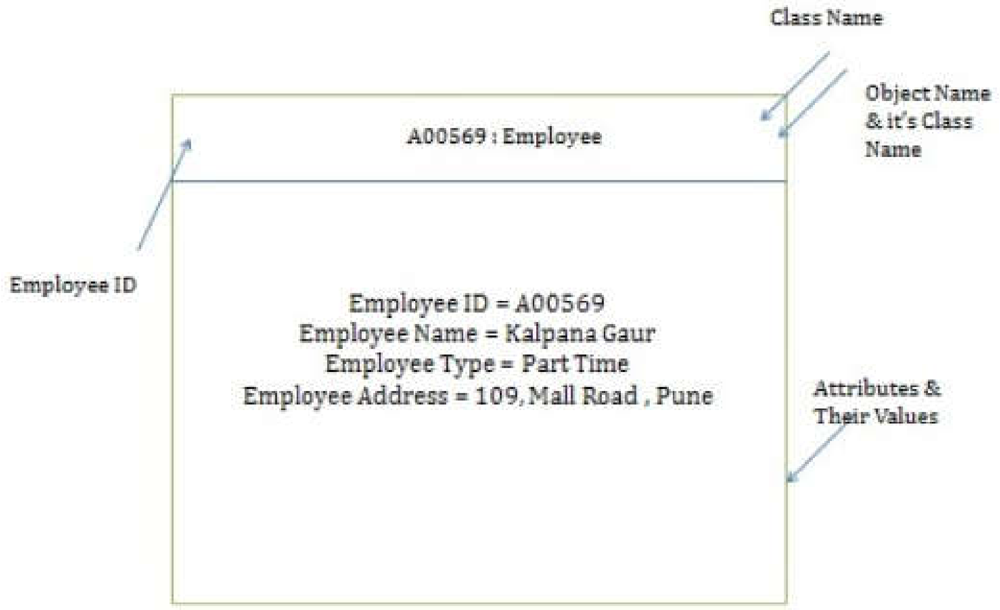
Unified Modeling Language (UML)

- UML is a visual language that lets you to model processes, software, and systems to express the design of system architecture. It is a standard language for designing and documenting a system in an object oriented manner that allow technical architects to communicate with developer.
- It is defined as set of specifications created and distributed by Object Management Group. UML is extensible and scalable.
- The objective of UML is to provide a common vocabulary of object-oriented terms and diagramming techniques that is rich enough to model any systems development project from analysis through implementation.
- UML is made up of –
  - **Diagrams** – It is a pictorial representations of process, system, or some part of it.
  - **Notations** – It consists of elements that work together in a diagram such as connectors, symbols, notes, etc.

UML Notation

- UML Notation for Class:
- Instance Diagram:




Class Name
Object Name & it's Class Name
A00569 : Employee
Employee ID
Employee ID = A00569
Employee Name = Kalpana Gaur
Employee Type = Part Time
Employee Address = 109, Mall Road , Pune
Attributes & Their Values

Uses of UML

- UML is quite useful for the following purposes −
  - Modeling the business process
  - Describing the system architecture
  - Showing the application structure
  - Capturing the system behavior
  - Modeling the data structure
  - Building the detailed specifications of the system
  - Sketching the ideas
  - Generating the program code

Static Models

Static models show the structural characteristics of a system, describe its system structure, and emphasize on the parts that make up the system.

- They are used to define class names, attributes, methods, signature, and packages.
- UML diagrams that represent static model include class diagram, object diagram, and use case diagram.

Dynamic Models

Dynamic models show the behavioral characteristics of a system, i.e., how the system behaves in response to external events.

- Dynamic models identify the object needed and how they work together through methods and messages.
- They are used to design the logic and behavior of system.
- UML diagrams represent dynamic model include sequence diagram, communication diagram, state diagram, activity diagram.

Thank You