# Assignment 1
# Group 4

Members:
Ayushman Tripathy - 19114018
Piyush Dagdiya - 19114025
Jitesh Jain - 19114039
Shashank Aital - 19114076
Shreyas Dodamani - 19114079

# Answer 1

## Perspective 1

In this we focus on what the design mainly contains. Whether it relies on data or control flow or whether some parts are grouped together as per the similarities between them. We don't define the way of implementation here, like for a certain type here, we may start developing the model from top to bottom or from bottom to top.

### 1. Exploratory Programming Style

- Every programmer developed his/her own style of writing programs according to his/her intuition.
- Typical program sizes were limited to a few thousands of lines of source code.
- There was no proper solution, strategy, plan or design for development.
- Coding was started immediately after hearing the problem

### 2. Control Flow-Based Design

- This method came up as size and complexity of programs increased. Hence, exploratory style proved to be insufficient. Also, programmers found it very difficult to write cost-effective and correct programs.
- In this technique, the main focus is on the control flow of the program.
- Control flow represents the sequence in which the program's instructions are executed.
- This method makes use of flow charting techniques to represent design algorithms
- This helps the programmer to easily trace and understand the processing that takes place along all the paths of the program.

- The control flow is made as simple as possible, thus making the debugging process easier and easy to trace errors in the control flow path.
- The problem with this design methodology is that , if the complexity of the software increases by a large amount, then it would cause the design to become messy and difficult to understand and debug.

## 3. Data Structure-Oriented Design

- Data structure-oriented design utilizes the data structures of the input data, internal data and output data to develop software.
- Hence, the emphasis is on the object, which is the data.
- The structure of information has an important part in the efficiency and complexity of algorithms designed to process information.
- Similar to the Data Flow Oriented Design, which also depends upon the information domain, it depends upon the analysis step to build the foundation for later steps.
- Both attempt to transform information into a software structure; and are represented by hierarchical diagrams.
- Examples of the Data Structure-Oriented Design approach:
    - Jackson System Development (JSD)
    - Data Structured Systems Development (DSSD)

## 4. Dataflow Oriented Design:

- This type of design methodology places more emphasis on the points of data in the program than the functions performed in the program.
- In this technique, we represent the system with a ‘data flow diagram’
- The diagram is a network of nodes (representing the various parts of the system) interconnected by arrows representing the flow of data from one part of the system to the other

- Advantages of the Data Flow design methodology:
    - They give a functional overview of the boundaries of the system.
    - Communicate the existing information to the stakeholders in a clear visual representation
    - Provide a functional breakdown of the system
- Drawbacks of this methodology:
    - It takes a long time to create, so, the analyst / designer might not receive support from the management to create it
    - If we want to know all the functions of a specific piece of data, then we would need to travel through all the nodes of the system. This would also take a long time.

## 5. Object Oriented Design

- Object oriented design works around the entities and their characteristics instead of functions involved in the software system.
- 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.
- Object Oriented Design is based on the concepts of:
    - Objects and attributes
    - Classes and members
    - Wholes and Parts
- All objects encapsulate data, other objects, constants, and other related information. **Encapsulation** means that this information is packaged into a single name and can be reused.

- Data abstractions are created by identifying classes and objects.
- Modules are specified by coupling operations and data and structure for the software is established.
- Interfaces are described by developing a mechanism for using objects (Ex. passing of messages).

---

## Perspective 2

In this perspective we focus on the way to design the software model rather than on what the model should contain. We define and design different hierarchies pertaining to which we develop our model.

The different design techniques are as follows:

1. **Stepwise Refinement**
2. **Levels of Abstraction**
3. **Structured Design**
4. **Integrated Top-Down Development**

### 1. Stepwise Refinement-

- It is a top-down technique for decomposing a system from high-level specifications into more elementary levels. It involves the following activities:
    - Isolating design aspects that are not truly interdependent.
    - Postponing decisions concerning representation details as long as possible.
    - Carefully demonstrating that each successive step in the refinement process is a faithful expansion of previous steps.
- **Advantages**:
    - Top down decomposition.

- Incremental addition of detail.
- Postponement of design decisions.
- Continual Verification of consistency.

## 2. Levels of Abstraction Technique

- Levels of abstraction was originally described by Dijkstra as a bottom-up design technique in which an operating system was designed as a layering of hierarchical levels starting at the lowest level.
- Internal functions are hidden from other levels, that can only be invoked by functions on the same level.
- Each level of abstraction performs a set of services for the next higher level of abstraction.
- Thus, a file manipulation system might be layered as a set of routines to manipulate fields.
- Higher level functions can invoke functions on lowest levels, but lower-level functions cannot use higher-level functions.

## 3. Structured Design

- Structured design was developed as a top-down technique for architectural design of software systems.
- The basic approach in structured design is systematic conversion of data flow diagrams into structure charts.
- Design heuristics such as coupling and cohesion are used to guide the design process.
- **First step:** Review and refine the data flow diagram developed during requirements definition and external design.
- **Second Step:** Determine whether the system is transform centered or transaction driven, and derive a high level structure chart based on this determination.
- **Third Step:** Decomposition of each subsystem using guidelines such as coupling, cohesion, information hiding and levels of abstraction, data abstraction and the other decomposition criteria.

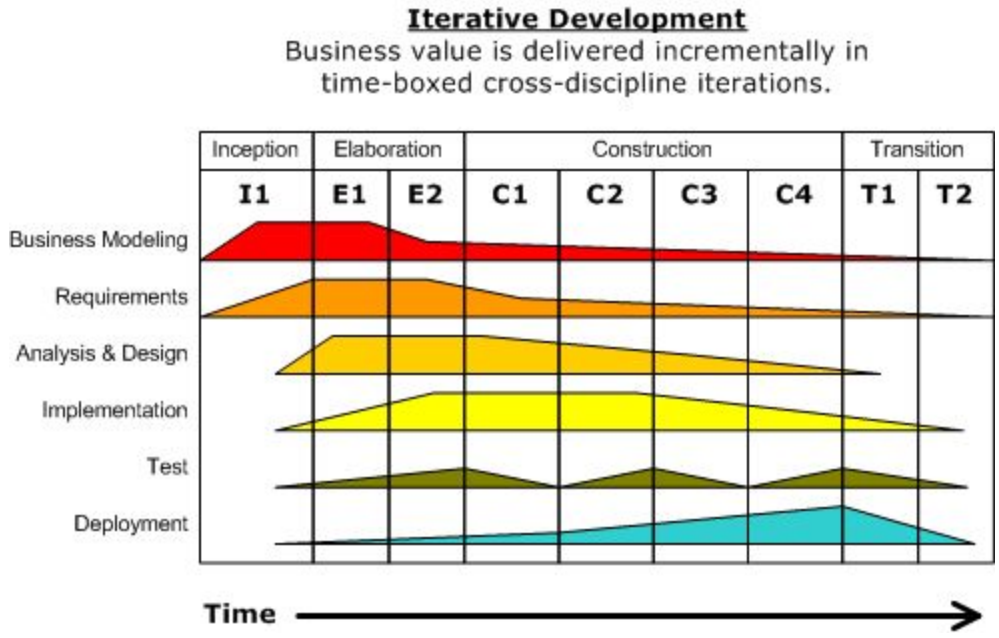
- The primary strength of structure design is provision of a systematic method for converting data flow diagrams into top-level structure charts.

## 4. Integrated Top-Down Development

- Integrated top-down development integrates design, implementation, and testing.
- Using integrated top-down development, design precedes top-down from the highest-level routines. They have the primary function of coordinating and sequencing the lower-level routines.
- Lower level routines may be implementation of elementary functions or they may in turn invoke more primitive routines.
- There is thus, a hierarchical structure to a top-down system in which routines can invoke lower-level routines but cannot invoke routines on a higher level.

# Answer 2

The Unified Software Development Process or **Unified Process** is a popular ***iterative and incremental software development process framework.*** It is not simply a process, but rather an extensible design framework which should be customized for specific organizations or projects.



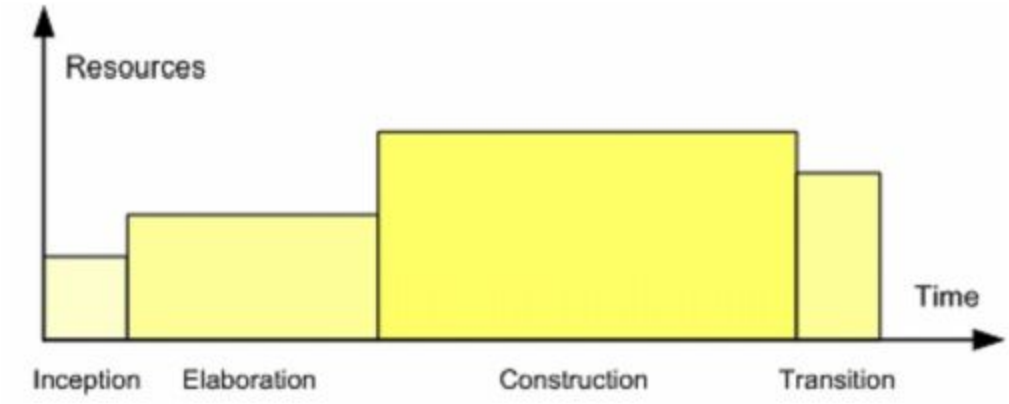

## Project Lifecycle

I. **Inception Phase:** Inception is the smallest phase in the project, and ideally, it should be quite short. This is the beginning of our project, where we plan out everything.
   A. Establish a justification or business case for the project
   B. Establish the project scope and boundary conditions
   C. Outline the use cases and key requirements that will drive the design tradeoffs
   D. Outline one or more candidate architectures and identify the risks
   E. Prepare a preliminary project schedule and cost estimate

II. **Elaboration Phase:** During the Elaboration phase, the project team is expected to capture a healthy majority of the system requirements. However, the primary goals of Elaboration are to address known risk factors and to establish and validate the system architecture. An example implementation is carried out to come up with a solid plan at the end.

III. **Construction Phase:** Construction is the largest phase of the project.
   A. System features are implemented in a series of short, timeboxed iterations.
   B. Each iteration results in an executable release of the software.

IV. **Transition Phase:** The final project phase is Transition.
   A. In this phase, the system is deployed (released) to the target users.
   B. Feedback received from an initial release (or initial releases) may result in further refinements to be incorporated over the course of several Transition phase iterations.




The characteristics of the Unified Process Model are:

1. **Iterative and Incremental:**
   a. The Unified Process is an iterative and incremental development process.

b. Each iteration results in an increment, which is a release of the system that contains added or improved functionality compared with the previous release.
c. Although most iterations will include work in most of the process disciplines (e.g. Requirements, Design, Implementation, Testing) the relative effort and emphasis will change over the course of the project.

2. **Use Case Driven**
    a. In the Unified Process, use cases are used to capture the functional requirements and to define the contents of the iterations.
    b. Each iteration takes a set of use cases or scenarios from requirements all the way through implementation, test, and deployment.

3. **Architecture Centric**
    a. The Unified Process insists that architecture sits at the heart of the project team's efforts to shape the system.
    b. One of the most important deliverables of the process is the executable architecture baseline which is created during the Elaboration phase. This partial implementation of the system serves and validates the architecture and acts as a foundation for remaining development.
4. **Risk Focused**
    a. The Unified Process requires the project team to focus on addressing the most critical risks early in the project life cycle.
    b. The deliverables of each iteration, especially in the Elaboration phase, must be selected in order to ensure that the greatest risks are addressed first.

# Answer 3

We believe that the Unified Process would be the best choice for designing the accounting framework of a small scale organisation. We could develop our software using Data Flow Design Methodology.

## Problems:

1. As the organisation is a small scale organisation, having generic structures might increase the cost of design.
2. It might cause unnecessary slowing down of the software due to a large number of unnecessary parts in it. So, our design strategy needs to be simple and with minimal complexity.
3. We might have to scale up on some parts of the design later or would have to add some new features to our design. So, we require our design to be adaptable to allow space for changes later.

Proposed So
In the accounting framework, there would be a lot of movement of money. Funds would come from outside, there would be expenses of the firm itself, as well as the handling of the payroll of the employees.

This description clearly indicates the *flow* of money from one part of the firm to another. Hence, the data flow model design best serves as the solution to this problem.

We are not relying on the object oriented model because it will increase the number of unnecessary elements in the system. Our organisation is also small in size, we don't need a complex program to manage it. The data flow design model would better cater to our needs as all the elements would be just sufficient and not superfluous.

In the unified process, we design the model in such a way that we only consider our current requirements without thinking much of the future changes that our design might encounter. When new changes come up, we try to accommodate them in our design, in such a way that the use-case model is still satisfied.
Since this is a small scale business, it will be easy to add the new (nodes) in the system, with less number of reconnections being required at each stage of addition of new features to our software.

# Answer 4

## Design Requirements-

Our framework would keep track of the transactions made by a shop and the business exchanges done with others, including dealers. It would also keep track of the stocks remaining in the store.

1. The design should be able to accommodate the addition of new stores easily
2. The chain may consist of stores selling various types of products, thus it must be able to accommodate for any new type of store or any new item in a given product chain.
3. Making changes to a specific store should be easy and not affect any other store.

## Proposed Solution-

We would suggest an **Object-Oriented Design** methodology for implementing the framework. We can define various classes to represent the different entities like the store, stock, transaction, etc. We can also generalise the store as a class and create subclasses based on the characteristics of the store like location, size, etc.

The object oriented design should be implemented in a top-down method. For this, we would suggest the **Integrated Top Down Technique**, as it would easily cater to the requirements of stores. We could have a branching of the classes at the top which would be according to the type of store. Then further specificity would be according to the magnitude of stores of each type. Then based on the amount of stocks further classification would be

done. Finally for implementing a particular store objects would be made which would correspond to each store. These objects would have all the parameters to keep record and have a function to calculate the total stocks for that store. The function would be very generic as it would be a result of inheritance from various levels and would do the work we required it to perform.

The company can easily monitor the total stocks by simply calling the stock function for a particular store and can even fetch specific details if it requires, as it would have access to various data of the store in the chain.

This would be an efficient way to handle the stocks of stores in the whole chain, in which there would be a lot of similarities and generic parts have a single parent from which they inherited, thus allowing for uniformity and also having space for addition of new stores.