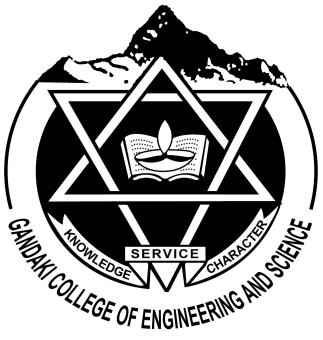
Gandaki College of Engineering and Science
Lamachaur-19,Pokhara



Submitted By
Pratigya Dhakal
Roll no :27

Submitted To
Er. Zena Poudel

1.

a) Describe the Object Oriented Analysis and Design process with example.

Ans:

Object Oriented Analysis (OOA) is the procedure of identifying software engineering requirements and development softwares specifications in terms of a software system's object model, which comprises interacting objects.

So, object oriented analysis and design(OOAD) is a popular technical approach for analysis and designing on application, as well as using visual modeling throughout the development life cycle to foster better stakeholder communication and product quality.

For example, consider a 'dice game' in which software simulates a player rolling a dice. If the total is seven, he wins and loses.

Define use case

Player requests to roll the dice. System presents results: If the dice face value totals seven, the player wins, otherwise, the player loses.

Define a domain model

So, domain models help in visiluation of the concept or mental models of the real world domain. So, it is also called conceptual objects model.

b) Change management is very important in Iterative and Incremental Development. Why and how is it done?

Ans:

The incremental and iterative development process is closely associated with Agile project management, most notably the Scrum methodology. This is because it aligns with one of the key pillars of Agile: responding to change over following a set plan.

Rather than adhering to a linear Waterfall method, software developers will react quickly to changes as their product evolves. They will build on previous versions to improve their product and repeat this process until the desired deliverables are achieved.

An example of iterative and incremental development in Agile could be the creation of a new e-commerce website. The project would be broken down into smaller increments, such as building a wireframe, uploading products, and creating advertising copy. As these steps are unfolding, the software development team would repeat the cycles of prototyping and testing to make improvements to the website with each iteration.

The incremental and iterative development process is integral to the field of Agile software development as it enables project managers to reap the benefits of both incremental and iterative approaches.

Incremental development ensures that developers can make changes early on in the process rather than waiting until the end when the allotted time has run out and the money has been spent.

Iterative development means improvements are made on an ongoing basis, so the end result is likely to be delivered on time and be of higher quality. This belief is echoed by CIO.com, which notes that short, iterative sprints can help teams to “deliver a better product, in a faster manner.”

2.

a) Explain 4 P’s of software development .

Ans:

Four p's in software engineering are:- People Product Process And Project

- "People" usually refer to all the people involved in the life cycle of a software.The most important component of a product and

its successful implementation is human resources. In building a proper product, a well-managed team with clear-cut roles defined for each person/team will lead to the success of the product. We need to have a good team in order to save our time, cost, and effort. Some assigned roles in software project planning are project manager, team leaders, stakeholders, analysts, and other IT professionals. Managing people successfully is a tricky process which a good project manager can do.

- "Product" refers to the estimation of the cost, time and effort required to produce the finished software product. As the name inferred, this is the deliverable or the result of the project.The project manager should clearly define the product scope to ensure a successful result, control the team members, as well technical hurdles that he or she may encounter during the building of a product. The product can consist of both tangible or intangible things such as shifting the company to a new place or getting new software in a company.

- "Process" means the various models and methodologies or sometimes even technologies used while going through the making of a software project. In every planning, a clearly defined process is the key to the success of any product. It regulates how the team will go about its development in the respective time period. The Process has several steps involved like, documentation phase, implementation phase, deployment phase, and interaction phase.

- "Project" means we have to take the right steps for the successful completion of the project. If something goes wrong, then the project manager should know what steps need to take in order to solve the problem. In this phase, the project manager

plays a critical role. They are responsible to guide the team members to achieve the project's target and objectives, helping & assisting them with issues, checking on cost and budget, and making

- sure that the project stays on track with the given deadlines.
-

b) What is Unified Process? Explain its characteristics.
Ans:
The Unified Process (UP), or Unified Software Development Process, is a iterative and incremental software development framework from which a customized process can be defined. The framework contains many components and has been modified a number of times to create several variations.

The key characteristics of the Unified Process are:

- It is an iterative and incremental development framework
- It is architecture-centric with major work being done to define and validate an architectural design for most coding is done
- It is risk-focused and emphasizes that highest-risk factors be addressed in the earliest deliverables possible
- It is use-case and UML model driven with nearly all requirements being documented in one of those forms

The characteristics are :
It is use-case driven
It is architecture-centric
It is risk focused
It is iterative and incremental

1)Use Case Driven

A use case is a sequence of actions, performed by one or more actors (people or non-human entities outside of the system) and by the system itself, that produces one or more results of value to one or more of the actors. One of the key aspects of the Unified Process is its use of use cases as a driving force for development. The phrase use case driven refers to the fact that the project team uses the use cases to drive all development work, from initial gathering and negotiation of requirements through code. (See "Requirements" later in this chapter for more on this subject.

2) Architecture Centric

The architecture-centric design method (ACDM) is an iterative process used to design software architectures. It is a lightweight method with a product focus and seeks to ensure that the software architecture maintains a balance between business and technical concerns. It attempts to make the software architecture the intersection between requirements and the solution.

3) Iterative and Incremental

Incremental: An incremental approach breaks the software development process down into small, manageable portions known as increments. Each increment builds on the previous version so that improvements are made step by step.

Iterative: An iterative model means software development activities are systematically repeated in cycles known as iterations. A new version of the software is produced after each iteration until the optimal product is achieved.

3.

a) Define software quality. How can you assure quality in software you develop? Give your answer from the prospect of software System Analysist.

Ans:

Software quality is defined as a field of study and practice that describes the desirable attributes of software products.

Assuring quality in software:

1) Create a quality culture

This is all about fostering the correct environment that will produce quality software. It's not just about the final source code - it's about the culture and the team that built it.

2) Put the user first

"Quality is essentially about meeting user expectations. Consistently. Reliably. Confidently."

We know that the end user rules in software development. No matter how great your software system, if your UX isn't up to scratch, your customers will simply go elsewhere. The average app lost about $33,000 a month in 2019 due to uninstalls, so a dissatisfied user truly can directly impact your ROI.

3) Test early and often

Even if you don't use the agile methodology, you shouldn't leave testing until the last minute. Regular testing and a structured QA strategy is a surefire way to ensure software quality. Multiple testing cycles combined with automation will catch the bugs that thinner testing strategies may have missed.

4) Don't just have one software testing method in your toolkit

"Correct understanding and scoping of testing requirements and identification of the testing types required" are both of vital importance to ensuring quality in the software you create.

5) Slow down the process
If your dev and QA teams work under intense pressure to release by tight deadlines, the likelihood of technical debt will increase, and the quality of your products may decrease as a result.
Technical debt is often inevitable, unfortunately, but slowing down the release process is a great way to reduce it. Take your time to put best practices in place, and reduce the likelihood of costly fixes in the future.

b) What do you mean by Creational Patterns? Explain any one of its type with DCD and source code snippet.
Ans:
Creational design patterns are design patterns that deal with object creation mechanisms, trying to create objects in a manner suitable to the situation. The basic form of object creation could result in design problems or in added complexity to the design. Creational design patterns solve this problem by somehow controlling this object creation.Creational design patterns provide various object creation mechanisms, which increase flexibility and reuse of existing code.
Types of creational pattern:

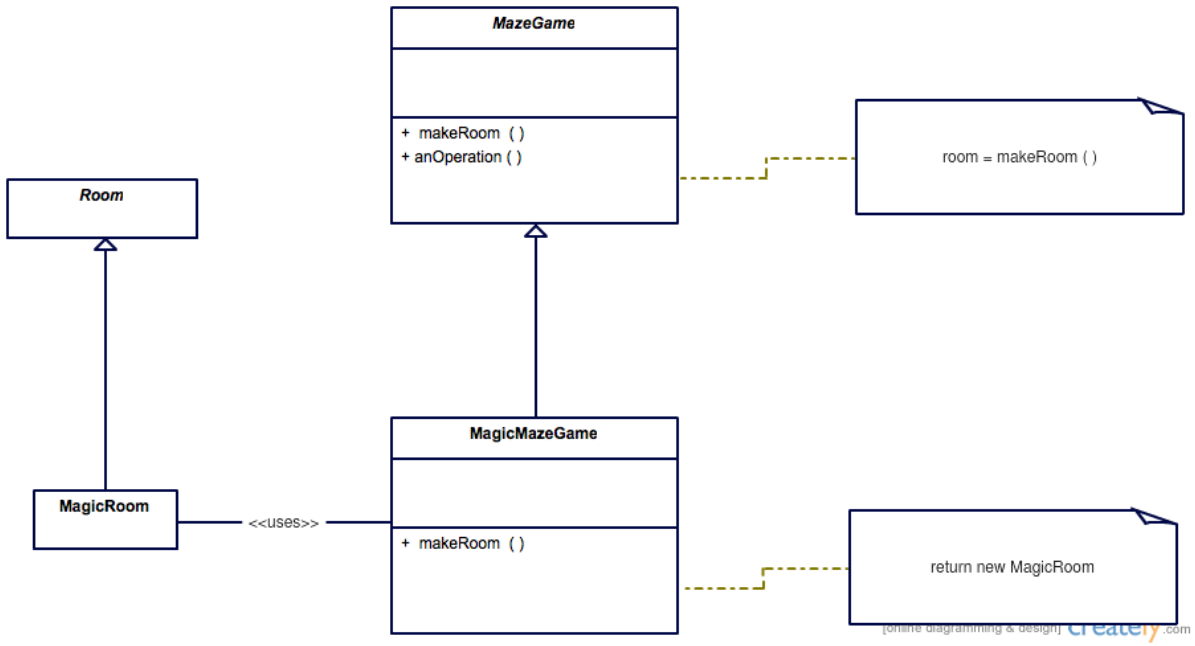
- Abstract Factory
  Creates an instance of several families of classes
- Builder
  Separates object construction from its representation
- Factory Method
  Creates an instance of several derived classes
- Object Pool

Avoid expensive acquisition and release of resources by recycling objects that are no longer in use

- Prototype
  A fully initialized instance to be copied or cloned
- Singleton
  A class of which only a single instance can exist

Explanation:
Factory Method:
Factory Method is a creational design pattern that provides an interface for creating objects in a superclass, but allows subclasses to alter the type of objects that will be created.
This is done by creating objects by calling a factory method—either specified in an interface and implemented by child classes, or implemented in a base class and optionally overridden by derived classes—rather than by calling a constructor.

//DCD



//CODE
This example is similar to one in the book Design Patterns.

The MazeGame uses Rooms but it puts the responsibility of creating Rooms to its subclasses which create the concrete classes. The regular game mode could use this template method:

```
public abstract class Room {
   abstract void connect(Room room);
}

public class MagicRoom extends Room {
   public void connect(Room room) {}
}

public class OrdinaryRoom extends Room {
   public void connect(Room room) {}
}

public abstract class MazeGame {
    private final List<Room> rooms = new ArrayList<>();

    public MazeGame() {
        Room room1 = makeRoom();
        Room room2 = makeRoom();
        room1.connect(room2);
        rooms.add(room1);
        rooms.add(room2);
    }

    abstract protected Room makeRoom();
}
```

In the above snippet, the MazeGame constructor is a template method that makes some common logic. It refers to the makeRoom factory method that encapsulates the creation of rooms such that other rooms

can be used in a subclass. To implement the other game mode that has magic rooms, it suffices to override the makeRoom method:

```
public class MagicMazeGame extends MazeGame {
    @Override
    protected MagicRoom makeRoom() {
        return new MagicRoom();
    }
}

public class OrdinaryMazeGame extends MazeGame {
    @Override
    protected OrdinaryRoom makeRoom() {
        return new OrdinaryRoom();
    }
}

MazeGame ordinaryGame = new OrdinaryMazeGame();
MazeGame magicGame = new MagicMazeGame();
```

4.

a) What is Design Pattern? Write about the importance of design patterns.

Ans:

Design patterns are typical solutions to commonly occurring problems in software design. They are like pre-made blueprints that you can customize to solve a recurring design problem in your code.These solutions were obtained by trial and error by numerous software developers over a quite substantial period of time.

Benefits of design patterns.

- Robust Cost

- Code Reusability
- Highly Maintainability
- Reduce Total Cost of Ownership(TCO)
- Loosely coupled application

The design pattern are important because:

- They can speed up the development process by providing tested, proven development paradigms.
- Reusing it helps to prevent subtle issues that can cause major problems and improve code readability for coders .
- They provide general solutions, documented in a format that doesn't require a particular problem.
- Allows developers to communicate using well-known, well understanding names for software interactions.

b) What are the difficulties and risk while using design pattern? Discuss in brief.

Ans:

Design patterns have both good and bad leading to design problems. The patterns are no magic but they are some best practices identified by some experts who solving some repetitive similar problems.

By applying the design patterns to some identified problems, it is almost guaranteed that you will have a better-than-average solution that is flawless.

On the other hand, design patterns discourage our thinking of solving problems in a different way. It is like you already know how to cook something which fulfills your hunger, then you will not spend more time exploring and inventing better recipes.

1) Patterns are not a panacea : Whenever you see an indication that a pattern
should be applied , you might be tempted to apply the pattern blindly.
2) Developing patterns is hard : Writing a good pattern takes considerable
work. A poor pattern can be hard for other people to apply correctly, and can
lead them to make incorrect decisions.
3) Leads to inefficient solutions : The idea of design pattern is an attempt to
standardize what are already accepted best practices.Just barely good enough pattern.
4) Targets the wrong problem : The need for patterns results from using
computer language or techniques with insufficient abstraction ability.
5) Lack of formal foundations : The study of design patterns has been excessively ad hoc, and some have argued that the concept sorely needs to
be put on a more formal footing.THey were “convicted by ⅔ of the jurors
who attended the trial.

5.
a) Explain about concurrency pattern.
Ans:
Concurrency patterns describe ways to deal with multithreading and concurrent executing code.
Some of the concurrency patterns are

- Active Object Concurrency Pattern
  The active object design pattern decouples method execution from method invocation for objects that each live in their thread

of control. The goal is to introduce concurrency, by using asynchronous method invocation and a scheduler for handling requests.

- Active Record Concurrency Pattern
  An object that encapsulates a row from a database table or view, by providing access to the database and adding domain logic. Because the data managing operations are implemented directly on the object, it is tightly coupled to the storage technology.
- Monitor Object Concurrency Pattern
  The Monitor Object design pattern synchronizes concurrent method execution to ensure that only one method at a time runs within an object. It also allows an object's methods to cooperatively schedule their execution sequences.
- Thread-Specific Storage Concurrency Pattern
  The thread-specific Storage design pattern allows multiple threads to use one `logically global' access point to retrieve an object that is local to a thread, without incurring locking overhead on each object access.

b) What is Software Architecture ? Describe an architecture using UML.

Ans:

Software Architecture is the process of designing the global organization of the software system, including dividing software into subsystems, deciding how these will interact and determining their interfaces. The term “Software Architecture” is also applied to the documentation produced as a result of the process.

Software architecture involves the high level structure of software system abstraction, by using decomposition and composition, with architectural style and quality attributes. A software architecture design must conform to the major functionality and performance

requirements of the system, as well as satisfy the non-functional requirements such as
reliability, scalability, portability, and availability.
A software architecture must describe its group of components, their connections, interactions among them and deployment configuration of all components. The center is the Use Case view which connects all these four. A Use Case represents the functionality of the system. Hence, other perspectives are connected with use case. Design of a system consists of classes, interfaces, and collaboration.
UML provides class diagram, object diagram to support this.Implementation defines the components assembled together to make a complete physical system. UML component diagram is used to support he implementation perspective.
Process defines the flow of the system. Hence, the same elements as used in Design are also used to support this perspective.
Deployment represents the physical nodes of the system that forms the hardware. UML deployment diagram is used to support this perspective.

6.
a) What do you mean by architectural pattern? Explain about Model View Controller(MVC) architectural pattern.
Ans:
Architectural patterns are a method of arranging blocks of functionality to address a need. Patterns can be used at the software, system, or enterprise levels. Good pattern expressions tell you how to use them, and when, why, and what trade-offs to make in doing so. Patterns can be characterized according to the type of solution they are addressing (e.g., structural or behavioral).

Model View Controller (MVC) architecture is software architectural pattern for implementing good Software on computers. It divides a given application into three interconnected parts. This is done to separate internal representation of information is presented to and accepted from the user the MVC design pattern decouples the major components allowing for efficient code reuse and parallel development.

Traditionally used for desktop GUIs, this architecture has become popular for designers in web applications and even mobile, desktop and other clients. Popular programming language like java, C#, Ruby , PHP and others have popular MVC framework that are currently being used in web application development straight out of the box

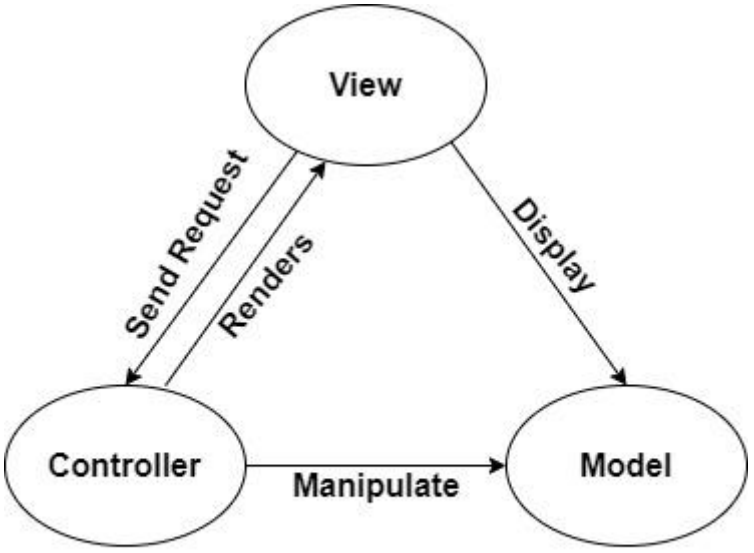


Fig: MVC Architecture

The MOdel stores data that is retrieved according to commands from the controller and displayed in the view. A view generates new output to the user based in changes in the model. A controller can send commands to the model's state . It can send command to its associated view to change the view's presentation of the model.

b) Explain Message Oriented Architecture pattern with design principles.

Ans:

A Message-Oriented Architecture is sometimes called Message-Oriented Middleware or MOM because there's a middle tier or middleware that acts as the broker to relay messages from producers to consumers. At the most succinct definition, MOM is simply an architecture that supports sending and receiving messages.

Some of the terminologies related are

- Producer
  Something that produces a message. A message could be a command sent to a single command handler or an event published to zero or more event handlers.
- Consumer
  Something that consumes a message
- Handler
  Synonym for consumer
- Message Broker
  A component, often in an ESB, that transforms/translates, and/or routes messages from one subsystem to another.
- Queue
  A storage mechanism for messages. Depending on the implementation, a queue may be where messages are received from. Generally, a separate process that receives messages through a known protocol like HTTP or TCP. Generally used to implement load balancing semantics.
- Aggregator
  Something that consumes messages and contains them in some way and directs the message to another consumer. Used, partially, to decouple producers from consumers. A Queue is a form of the aggregator. Often lives within the process. Sometimes used to merge pipelines.
- Topic
  Metadata associated with one or more messages that is generally used for publishing and subscribe semantics. Something

publishes to or subscribes to topics rather than having to process all messages.

- Exchange
  In messaging systems that implement exchanges, exchanges are where messages are sent (compared to a queue, which is technically where messages are received from).
- Service Bus
  An integration platform that generally combines web services, messaging, transformations, and routing, to transactionally integrate many disparate applications or systems. Often referred to as an Enterprise Service Bus or ESB.
- Pipeline
  Often used in the context of messaging to describe a particular stream of messages or the fact that messages arrive or are processed one by one in a particular order.
- Event
  A message that informs subscribers of something that occurred in the past (albeit possibly not in the very distant past).

Some features are:

- Very loosely coupled
  Consumers and producers are separated by messages. These messages are simply data. They could be a binary serialized blob of data that a producer needs to be able to understand how to deserialize (often limiting producer/consumer to the same platform), or the message could be in the form of XML; where the consumer need only know how to process XML (not limiting consumer/producer to the same platform).
- Platform agnostic

Messages can be sent and received on any platform. Much in the same way an HTML page can be downloaded and viewed on any platform. There are degrees to which a message-oriented system can support multiplatform (see above w.r.t. binary serialization), but the architecture itself imposes no real platform specifics.

- Asynchronous
  Consumers give messages to a broker, aggregator, or queue of some sort and that message is processed and delivered asynchronously. The consumer is free to go about its business before effectively before the message has even left the broker. Of course, when consumers and producers live in the same memory space using certain types of aggregators, this may not be the case; but typical message-oriented architectures use middleware that processes messages asynchronously.

7.

a) Architecture centric process

Ans:

Aspects of an architecture include static elements,dynamic elements, how those elements work together, and the overall architectural style that guides the organization of the system.

Architecture also addresses issues such as performance,scalability,reuse and economic and technological constraints.

The Unified Process specifies that the architecture of the system being built ,as the fundamental foundation on which that system will rest, must sit at the heart of project team's efforts to shape the system/

Need of the architecture centric:

- Understanding the big picture
- Organizing development effort
- Facilitating the possibilities for reuse

- Guiding the use cases
- Evolving the system

b) Interaction Diagram

Ans:

Interaction diagrams describe how groups of objects collaborate in some behavior. The UML defines several forms of interaction diagrams, of which the most common is the sequence diagram.

Interaction diagrams give the designer a unique ability to see the entire sequence in a use case at an overview level.

For each use case we draw an interaction diagram.The interaction takes place as the blocks sends stimuli between one another.As we draw interaction diagrams,we aslo define stimuli, including their parameters.The main purpose of the use case design is thus to define the protocols of the blocks.In the design model we refine the description of the use cases by showing in the interaction diagram how the objects behaves in the use cases.

The interaction diagrams are controlled by events.A new event gives rise to a new operation.These events are stimuli that are sent from one object to another to initiate an operation.

The interaction diagram is a type of diagram used for a long time in the world of telecommunications.

Each participating block is represented by blocks .These bars are drawn as vertical lines in the diagram .The order between the bars is insignificant and should be selected to give greater clarity.

c) Message Oriented Architecture

Ans:

A Message-Oriented Architecture is sometimes called Message-Oriented Middleware or MOM because there's a middle tier or middleware that acts as the broker to relay messages from

producers to consumers. At the most succinct definition, MOM is simply an architecture that supports sending and receiving messages. Some of the terminologies related are

- Producer
  Something that produces a message. A message could be a command sent to a single command handler or an event published to zero or more event handlers.
- Consumer
  Something that consumes a message
- Handler
  Synonym for consumer
- Message Broker
  A component, often in an ESB, that transforms/translates, and/or routes messages from one subsystem to another.
- Queue
  A storage mechanism for messages. Depending on the implementation, a queue may be where messages are received from. Generally, a separate process that receives messages through a known protocol like HTTP or TCP. Generally used to implement load balancing semantics.
- Aggregator
  Something that consumes messages and contains them in some way and directs the message to another consumer. Used, partially, to decouple producers from consumers. A Queue is a form of the aggregator. Often lives within the process. Sometimes used to merge pipelines.
- Topic
  Metadata associated with one or more messages that is generally used for publishing and subscribing semantics. Something publishes to or subscribes to topics rather than having to process all messages.
- Exchange

In messaging systems that implement exchanges, exchanges are where messages are sent (compared to a queue, which is technically where messages are received from).

- Service Bus
  An integration platform that generally combines web services, messaging, transformations, and routing, to transactionally integrate many disparate applications or systems. Often referred to as an Enterprise Service Bus or ESB.
- Pipeline
  Often used in the context of messaging to describe a particular stream of messages or the fact that messages arrive or are processed one by one in a particular order.
- Event
  A message that informs subscribers of something that occurred in the past (albeit possibly not in the very distant past).