

Year: 2015 (Spring) Date _____
Page _____

Q1) What is UML? Is UML an object oriented supported tool to design object oriented systems. Support your example with a suitable example.

Soln) Unified modelling language (UML) is a standardized modeling language enabling developers to specify, visualize, construct and document artifacts of a software system. Thus, UML makes these artifacts scalable, secure and robust for execution. So, UML is an important aspect involved in object oriented software development.

The UML is a tool for specifying SW systems. Standardized diagrams types to help us describe and visually map a software system design and structure. Many article discuss great feature of UML, but just a few of them describe experimental results with UML in real-life projects. In order to quantitatively present the advantages of UML over common way of software development.

A group of students who took, attended the course during their university education on C++, SQL basic, Java basic and C were divided into two subgroup. Among them, group (A) took UML courses of presentation while the group (B) didn't attend any UML sessions. Now, two systems were to be designed. the non-UML group or group (B), having no knowledge on UML nor experiences in the object oriented

System design and development had to design the systems using previous knowledge and sense. So, they just made few tough descriptions of the system operation and some flow-diagrams. On the flip side, UML group could use various tools of UML to understand and show the internal structure & architecture of the system. It created mostly use case sequence and activity diagrams and even class diagram that show classes, methods and attributes. Now, implementation as a final goal of every development process requires that many details got to be defined.

Q(b) What are the activities of Unified process? Describe in short.

The unified process is a popular iterative and incremental software development process framework. It's best known and extensively documented refinement of the Unified process. It's not just one process but rather an extensively used framework which should be customized for specific organizations or projects. The phases and activities of unified process are enlosed below:

i) Inception

ii) Elaboration

iii) Construction

iv) Transition

Inception

Inception is the smallest phase in project and ideally is short phase. In Inception phase idea is planned. The following are type or goals for the Inception phase:

- Establish.
- prepare preliminary project schedule and cost estimate
- Feasibility
- Buy or develop for.

The life cycle objective milestone marks the end of the Inception phase.

Elaboration

The goal of elaboration phase is to establish the ability to build new system given the financial constraints, and other kinds of constraints that the development project faces. The task in elaboration phase includes:

- capturing healthy majority of remaining functional requirement.
- Addressing significant risk on ongoing

basis.

- expanding the candidate architecture into full ~~base~~ architectural base line.

And major milestone associated with elaboration phase are:

- most of the functional requirements for new system have been captured in the use case model.

- architectural base line is complete which will serve as solid foundation for ongoing development.

A Construction

The primary goal of construction phase is to build a system capable of operating successfully in live customer environments. During construction, the team performs tasks that involve building the system iteratively and incrementally making sure that the viability of the system is always evident in a testable form.

A Transition

The primary goal of transition phase is to roll out the fully functional system to customers. During transition, the project team focuses on correcting defects and

modifying the system to correct previously unidentified problems.

Q@ What is business modelling? What is the relationship between Business modelling and requirement gathering in the development of a software? Take one example to prove your point.

The process of business model design is part of business strategy. Business model design and innovation refer to the way a firm (or a network of firms) defines its business logic at the strategic level. In contrast, firms implement their business much at the operational level, through their business operation.

For many projects, deciding what needs to be done is more difficult than getting it done. It does not matter how talented a development team is, if they are not sent out to accomplish the right goals, they can't be expected to find success. That's why proper requirements are essential to produce high quality software.

The requirement can be obvious or hidden or unknown, expected or unexpected from client point of view. Requirement gathering is one of the steps of requirement engineering process which includes: feasibility requirement study,

Software requirement specification and Software requirement validation. This

This implies firstly the feasibility study analyzes whether the software product can be practically materialized terms of implementation, contribution of project organization, cost constraints and as per values and objectives of the organization.

Today's SW development methodologies are equipped with a combination of methods and technologies for business process engineering and requirements.

However, experience has shown that the methods and techniques deliver disappointing results when ~~not~~ being applied independent of each other. There are two main obstacles to effective alignment of business process and the information system (IS) that support them:

i) The IS are not developed with correct understanding of the business they are supposed to support.

ii) Business process are not linked to the system requirements and thus evolve independent from the IS.

Hence, unless the two process, i.e. business modelling and requirement gathering go side-by-side or aligned, the development of a software is heavily affected after math.

Q1) Discuss the strength and weakness of Object Oriented and procedural programming with the help of point of sale system.

Ans) Object oriented programming is problem solving technique which uses classes and objects to create models based on the real world environment.

An OOP application may use a collection of objects which will pass message when called upon to request a specific service or information.

The main advantage of OOP programming is it makes it easy to maintain and modify existing code as new objects are created inheriting characteristics from existing ones. This makes adjusting program much simpler. But the major disadvantage of OOP is increase in size of program, effort speed.

Similarly, procedural programming which at time has been preferred to as fine programming takes more top-down approach to programming. They takes an applications by solving problems from the top to the codes down to the bottom. They are faster

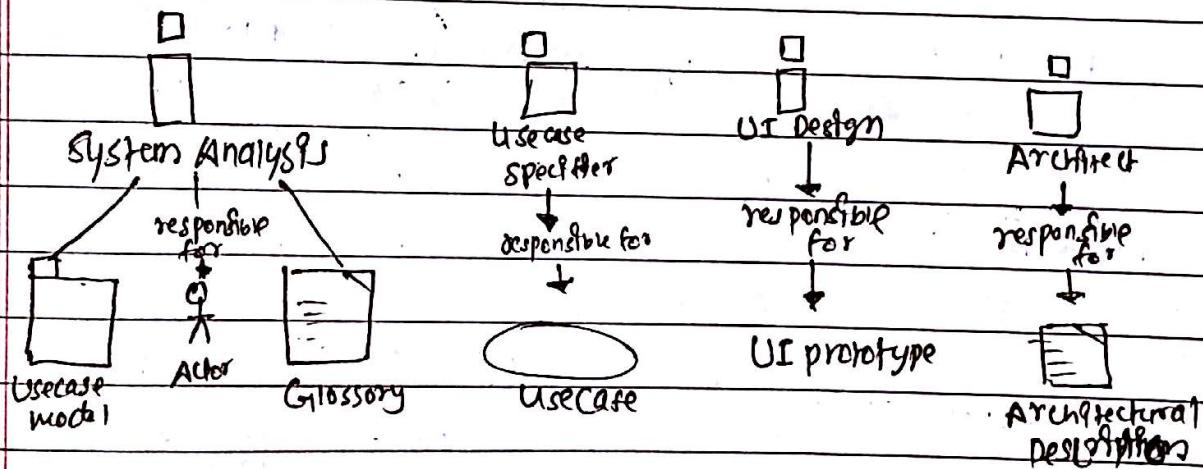
and size efficient but developer must edit every line of code that corresponds to the original change in code. For example: let us take a point of sales system which comprises network operated by a main computer and linked to several check-out terminals with pos system.

- One can analyse sales data, figure out how well the item on your shelves sell and adjust purchasing levels accordingly.
- One can maintain a sales history to help adjust buying decisions for seasonal purchasing trends.
- One can improve pricing accuracy by integrating bar-code scanner and credit card authorisation ability with pos system.

Q 3(a) What are use cases? Define workflow for capturing requirement as use cases, including the participating workers and their activities.

Soln In software and system engineering, a user-case is a list of actions or events steps typically defining the interactions between a role (known in UML as actor) and a system to achieve a goal. The actor can be a human or other external

System workflow within the unstaffed process, have come across the set of four phases: Requirements, analysis, design, implementation, and testing. Each workflow is a set of activities that varies project workers perform. The primary activities of the requirement work are aimed at building the use case model, which captures the functional requirements of the system being defined - the model helps the project stakeholders to reach agreement on the capabilities of the system and the conditions to which it must conform.



3(b) Design patterns have to be applied using some programming language. Discuss the statement in the light of programming paradigm versus design patterns. Take any two patterns with an example to prove your point.

Design pattern are as much lower level tool, providing proven models on how to organize code to gain specific functionality while not compromising the overall structure. In other words, a design pattern is a general, reusable solution to a commonly occurring problem in software design. It's not finished design that can be transformed directly into code. It's a descriptive template for how to solve a problem that can be used in many different situations.

Besides, paradigms are other extreme, guiding the principles on how code is actually laid out, and they each require quite different mind sets to apply. For instance, procedural programming is mainly concerned about dividing the program logic into functions and bundling those functions into modules. Object-oriented programming aims to encapsulate the data and the operations that manipulate the data into objects. Functional programming emphasizes the use of functions instead of separating statements following one another, following side effects and state changes. Some instances of programming paradigm includes :- imperative - which allow side effect.

- declarative - which does not state the order in which operations execute.
- functional - which allows side effect.
- Object oriented - which groups code together with state. the code modifier.
- procedural - which groups code into functions and so on.

So design patterns are easier to understand, they are template of solution, some common design patterns are :-

- Factory method
- Abstract factory. ~~Abstract~~

Factory method:

Creates an object from a set of similar classes, or some parameter, usually a string. Example is: creation of a message object in Java.

S1 Q1) What is the role of design pattern? How is design pattern important? Highlight the role of software development with respect to the design pattern?

Soln In software engineering, a design pattern is a general repeatable solution to a commonly occurring problem. A design pattern isn't a finished design that can be transformed directly into code.

This is a common method of mathematical pattern analysis and such analysis is important for the following reasons:

- Understanding patterns allows examine to identify such pattern when they first appear.
- It allows someone to make educated guesses.
- Understanding patterns aid in developing mental skill.

To understand the role of design patterns, let's take an example of an adapter. Adapter pattern is one among those 23 patterns described in the book "Design pattern" by gang of four. Adapter provides a solution to the scenario in which a client and server need to interact with one another, but cannot because their interfaces are incompatible.

To implement an adapter, we create a custom class, that has the interface provided by the server and defines the server operations in terms the client expect.

Hence, the role of design pattern in software development is very crucial to make timely and effective response to design issue and to build a better quality software. Design patterns have two major benefits:

- ① They provide you with a way to ~~get~~ some issues related to software development

Using a proven solution.

- (b) They make communication between designers efficient. Software professionals can head when they refer the name of the pattern used to solve the particular issue when discussing system design.

Q(b) Write about structure and documentation at pattern.

Design pattern documentation is highly structured. The patterns are documented from template that identifies the information needed to understand the software problem and the solution in terms of the relationship between the classes and objects necessary to implement the solution. There's no uniformity in the design pattern template decision. I.e., the authors prefer different style for their pattern templates. Some want to be more expressive less structural while the rest prefer their pattern to be more precise and high gain instruction. An example of template from the authors of the first book "Design pattern" is as follows:

Term	Description
pattern-name	→ Describes the short expressive name
Intent	→ what the pattern does.
Also known as	→ List any synonym for the pattern
Applicability	→ List the situations where pattern is applicable.
Structure	→ Set of diagrams of the classes and objects that depict pattern.
Participants	→ the classes and object that participate in design pattern.
Collaborations	→ Describes how the participants collaborate to carry out their responsibility.

The documentation of design patterns describes about the context in which the pattern is used, therefore the faces within the context that the pattern seeks to resolve and the suggested solution.

5(a)

Define design principles, design concept and design patterns. What are the dis-advantages of design patterns?

5(b)

Software design principles represent a set of guidelines that help us to avoid having bad design.

According to Robert Martin there are 3 important characteristics of a bad design that should be

avoided:

- (i) Rigidity
- (ii) Fragility
- (iii) Immobility

Other major principles includes

- * DRY (Don't Repeat Yourself)
 - Try to avoid duplicates. For instance, imagine you have copied and pasted blocks of code in different part of your system, then there are lots of same codes.

- * keep it simple.

Most systems works best if they are kept simple rather than making them complex, therefore, simplicity should be a key goal in design and unnecessary complexity should be avoided. Software design is the process of implementing software solutions to one or more sets of problems. One of the main components of a software design is the requirements analysis (SRA). SRA is a part of the software development process that defines specifications used in software engineering. The sets of fundamentals software design concepts includes the following:

- (i) Abstraction

(i) Architecture

(ii) patterns

(iii) modularity

(iv) Information hiding

(v) Functional Independence

(vi) Refinement

(vii) Refactoring

The disadvantages of design patterns are as follows:

- Design patterns may increase or decrease the understandability of a design and implementation.

They can decrease by adding condition or increasing the amount of code.

- It does not lead to direct code reuse.

- Complex fn. nature

- they are deceptively simple.

- They are validated by experience and decisions.

- Teams may suffer from pattern overload.

- Integrating patterns into a software development process is a human-intensive activity.

5(b) What is software architecture? Why do we need it?

Ans → Software architecture is the process of designing the global organization of a 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 engineers discuss all aspects of systems design in terms of the architectural model. The architectural model will often consider the overall efficiency, reusability, and maintainability of the system. There are four main reasons why we need to develop an architectural model:

- (i) To enable everyone to better understand the system.
- (ii) To allow people to work on individual pieces of the system in isolation.
- (iii) To prepare for extension of the system.
- (iv) To facilitate reuse and reusability.

6(a) Describe Service oriented architecture and design principles to help to adhere.

Soln A service-oriented architecture (SOA) is a style of software design where services are provided to the other components by application components through a communication protocol over a network. The basic principles of service-oriented architecture are independent of vendors, products, and technologies. A service is a discrete unit of functionality that can be accessed remotely and acted upon and updated independently, such as retrieving a credit card statement online.

The principles of SOA are:

- ① Explicit boundaries
- ② Shared contract and Schema, not class.
- ③ Policy-driven
- ④ Autonomous
- ⑤ wire formats, not programming languages APIs
- ⑥ Document-oriented
- ⑦ Loosely coupled
- ⑧ Standards - compliant
- ⑨ Vendor independent
- ⑩ Metadata - driven

Q6(b) Discuss about Client Server and Distributed architecture. Are they similar? Justify. What is the role of distributed architecture in today's world of automation? Explain.

SOL Client-Server architecture is a network architecture in which each computer or process on the network is either a client or a server. Servers are powerful computers or processes dedicated to managing disk drivers (file servers), printers (print servers) or network traffic (network servers).

On the other hand, distributed system architecture (DSA) is the ideal solution for integrating processes where there are multiple units, control rooms or geographically distributed locations. In distributed architecture components are hosted on different platforms.

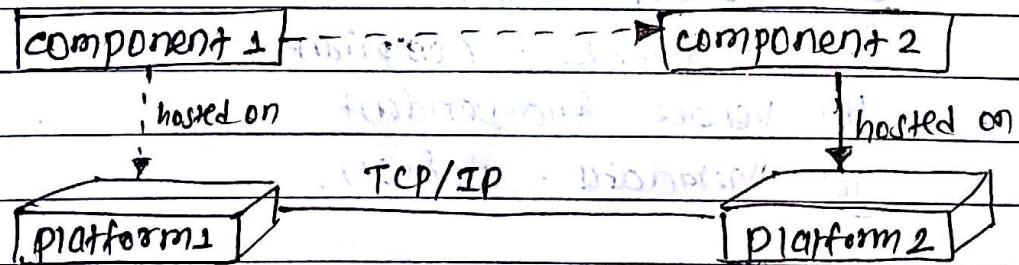


Figure: Distributed system Architecture

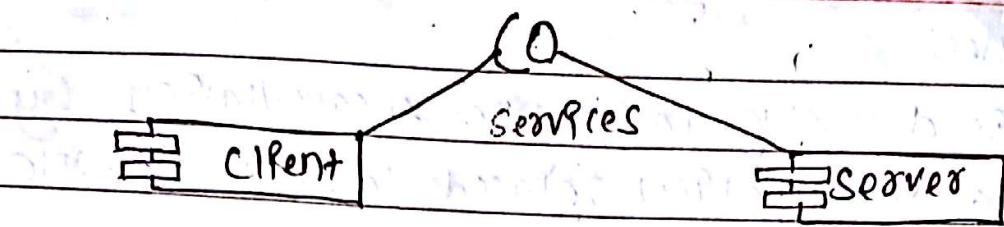


Figure: 1

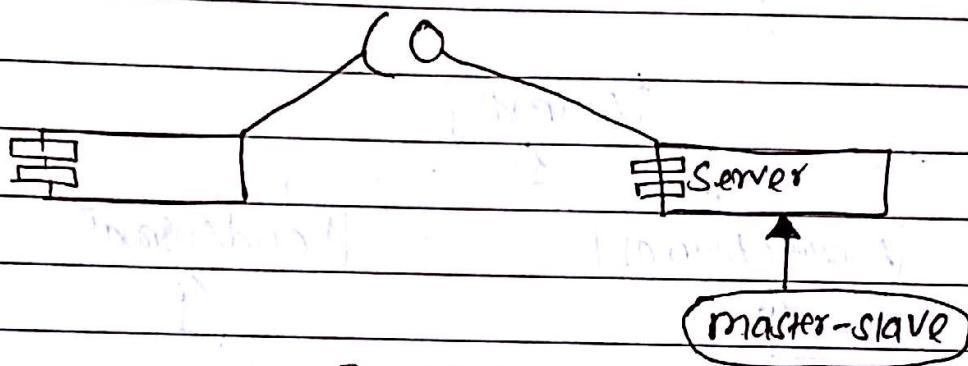


Figure: 2

7 @ player - ROK design pattern

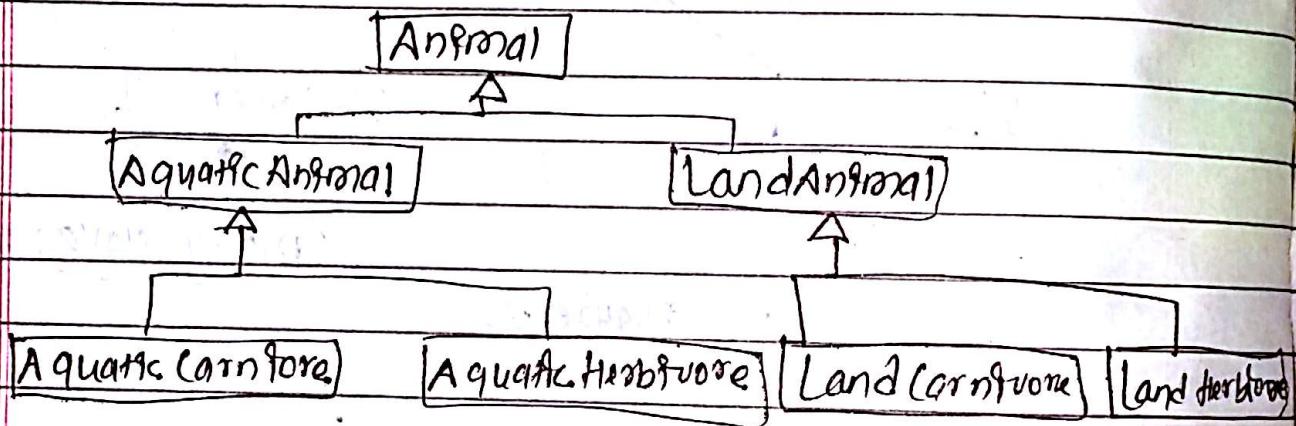
Goal: "studying software design patterns is an effective way to learn from the experience of others".

Context:

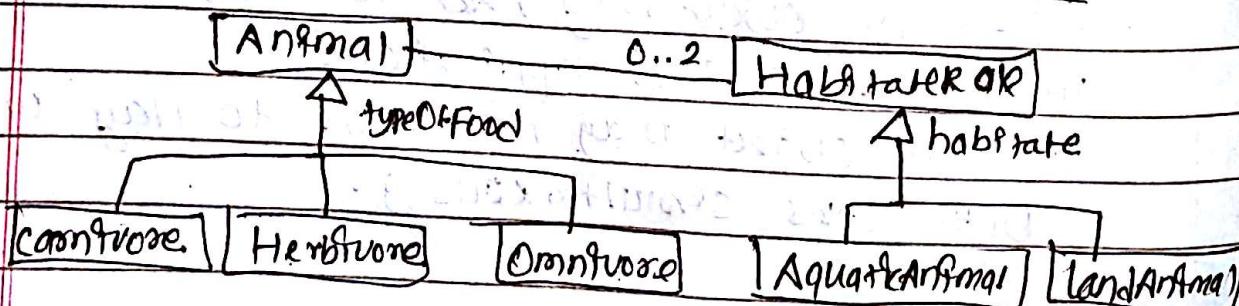
- an object may play different roles in different contexts of an application.
- an object may need to change roles throughout the course of the application.
- an object may not need to play one or both roles simultaneously.

ISSUES:

- It is desirable to improve encapsulation by keeping information related to only one role in one class.
- Making two classes to describe the same object is not good OO design.
Example of class design of animals

problem:

TWO or more objects are required if the animal needs to play different roles, like aquatic and land carnivore, which is not good OO design.

Player-role solution to animals

Solutions :



An animal can be:

- Carnivore
- Herbivore
- Omnivore

An animal can have:

- No habitat role
- an aquatic habitat role
- a land habitat role
- an aquatic and land habitat roles.

7(b)

SOM

Reuse Vs Reusability

Code reuse is called software reuse. Software reuse is the process of creating new software systems from existing software components. Reuse has an enormous impact on productivity. The following elements of software reused are, software specifications, designs, test cases, data, prototypes, plans, documentation, frameworks and templates.

Reusability is the degree to which the artifacts can be reused. The ability to use all or the greater part of the same programming code or system design in another application.

Reusability is the segment of source code that can be used to add new functionalities with slight

or localizes code modifications when a change in implementation is ~~not~~ required. Reusability is the extent to which a software component is able to be reused.

Reusable modules and classes

Increase the performance, reduce implementation time and use has to have appropriate design and coding standards. The types of software reuse

Software Reuse

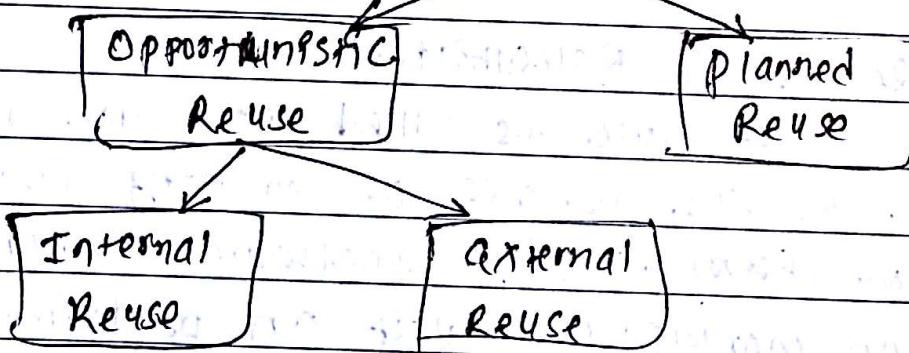


Fig: TYPES OF REUSE.