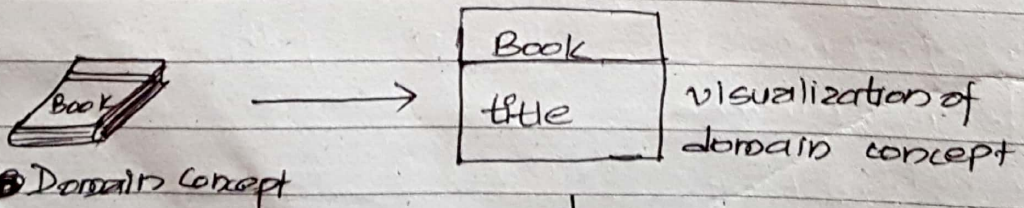
Q) What is object oriented Analysis and Design? Support your answer with suitable examples.

▷ During object oriented analysis, there is an emphasis on finding and describing the objects - or concepts - in the problem domain.
For example, in the case of the library information system, some of the concepts include Book, Library and Patron.

▷ During object oriented design, there is an emphasis on defining software objects and how they collaborate to fulfill the requirements.
For example, a Book software object may have a title attribute and a getChapter method.

▷ Finally during implementation of object oriented programming, design objects are implemented. such as a Book class in Java



Representation in an object-oriented programming language.

```
public class Book
{
    private String title;
    public Chapter getChapter(int n){...}
}
```

Iterative and Incremental Development

- The Unified Process is a approach to building, deploying and possibly maintaining software.

Unified Process

- The Unified Process is a software development process for building object-oriented system.
- The UP ~~combines~~ uses the software development practices such as Iterative and Incremental development.

Iterative and Incremental Development.

- The UP promotes Iterative and incremental development approach
- In this approach, development is organized into a series of short, fixed-length mini projects called Iterations.
- The outcome of each is a tested, integrated and executable system.
- Each Iteration includes its own requirements analysis, design, implementation and testing activities.
- The iterative lifecycle is based on the successive enlargement and refinement of a system through multiple iterations with cyclic feedback and adaptation as a core drivers to converge upon a suitable system.
- The system grows incrementally over time, Iteration by Iteration and thus this approach is also known as Iterative and Incremental development.

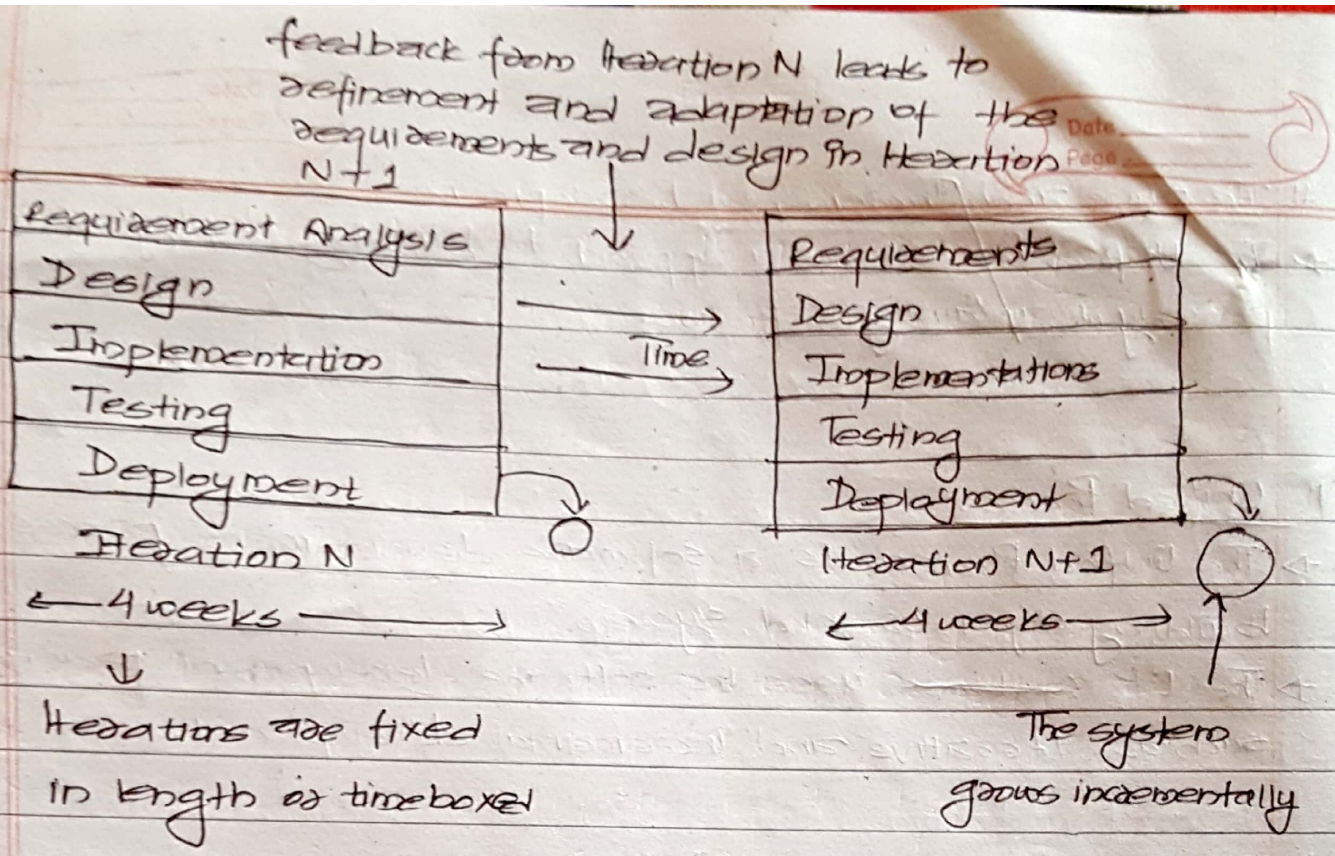


fig: Iterative and Incremental development

The phases of Unified Process are as follows:-

1) Inception
2) Elaboration
3) Construction
4) Transition.

1) Inception

- The vision of the end product is developed.
- The bussinesscase for the project is developed.
- The outline for the most crucial subsystems is designed.
- The most important risks are identified and prioritized.
- The elaboration phase is planned in detail and the whole project is roughly estimated.

2) Elaboration.

- Most of the project's use cases are specified in detail and the system architecture is designed
- The relationship between the architecture of a system and the system itself is developed.
- The architecture is expressed as views of all the models of the system, which together represent the whole system. The models include architectural views of the use-case model, the analysis model, the design model, the implementation model and deployment model.
- The most critical use cases are identified during the and realized.

Nikol

- Steps of building architecture
- Design principle.

Date
Page

→ At the end of elaboration phase, the project manager is in a position to plan the activities and estimate the resources required to complete the project.

Construction Phase:-

- The architecture baseline grows into a system.
→ Here, the product is fulfilled and made ready for transfer to the user.
→ Here, the developers may discover the better ways of structuring the system. So, they may suggest minor architectural changes to the architect.

At the end of this phase:-

- The product contains all the use cases that management and customer agreed to develop for this release.
- ~~Here~~ It may not be entirely free of defects, however.
~~Here~~ Here, the milestone is:
Does the product meet users needs for early delivery.

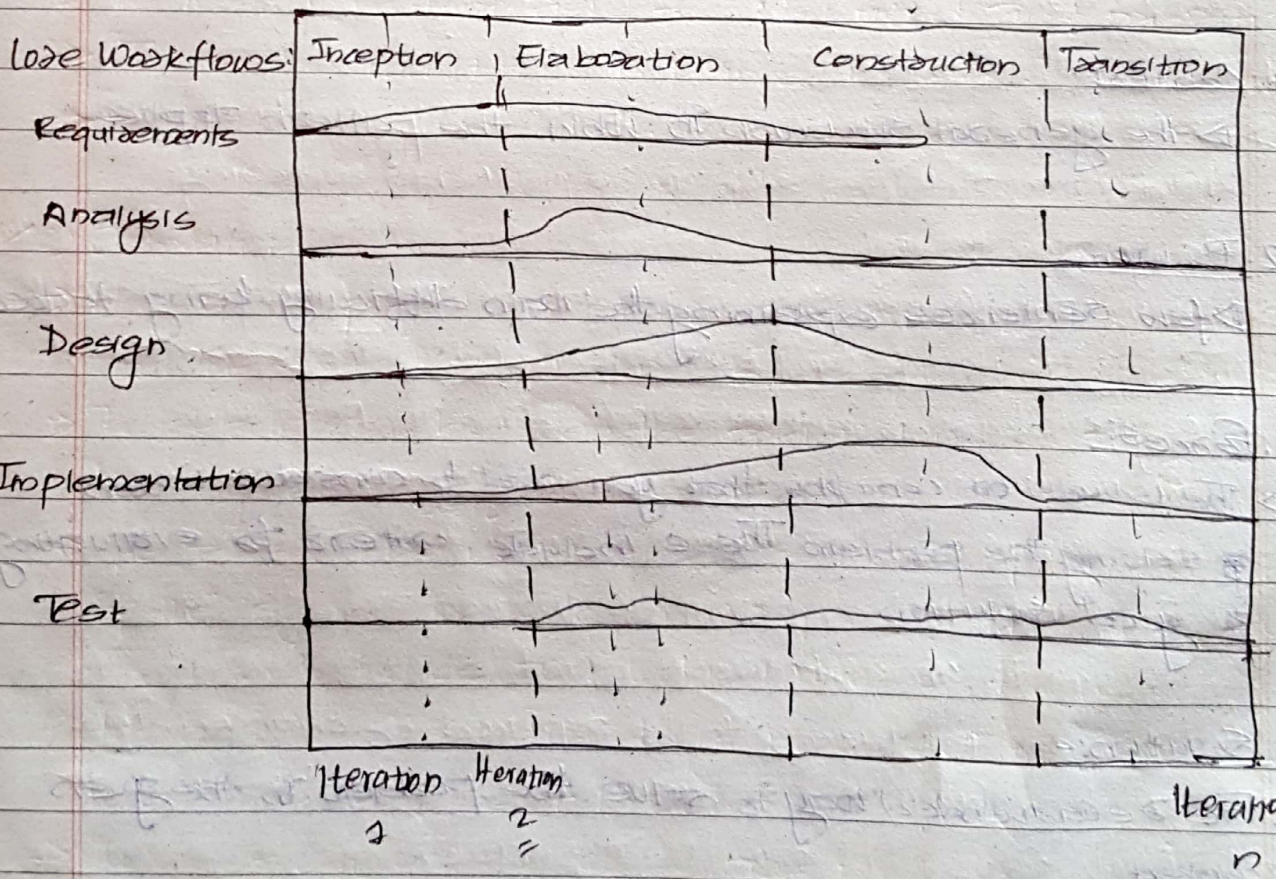
Transition Phase:-

→ In transition phase product moves into beta release.
→ In the beta release, a small number of experienced users tries the product and reports defects.
→ Developers then correct the reported problems and incorporate some of the suggested improvements into a general release for the larger user community

▷ This phase includes the activities such as reanufacturing, training customer personnel, providing help-line assistence and corresponding defects found after delivery.

▷ The maintenance team often divides these defects into two categories:-

- those with sufficient effect on operations to justify an immediate data release.
- those that can be corrected in the next regular release.



Design Pattern:-

▷ A pattern is the outline of a reuseable solution to a general problem encountered in a particular context.

A good pattern should be:-

- be as general as possible, containing a solution that has been proven to solve the problem effectively in the selected context
- easy-to-understand form

▷ Each pattern should have the following information.

1. Context
   ▷ the general situation in which the pattern applies
2. Problem:-
   ▷ few sentences explaining the main difficulty being tackled.
3. Forces:-
   ▷ The issues or concerns that you need to consider when solving the problem. These include criteria to evaluating a good solution.
4. Solution:-
   ▷ The recommended way to solve the problem in the given context
5. Antipatterns:- (optional)
   ▷ The solutions that don't work in this context.

Design Principles that lead to good Design

The overall goals we want to achieve when doing good design are

→ Increasing profit by reducing cost and increasing revenue.
→ Ensuring that all requirements of the customers were fulfilled.
→ Accelerating development. (reduce short-term cost,)
→ Increasing qualities such as usability, efficiency, reliability, maintainability and reusability.

The design principles are as follows.

1. Divide and Conquer:-

→ In software engineering, the process of development is divided into activities such as requirements gathering, design and testing.
→ Dividing a software system into pieces has many advantages
(i) Separate people can work on each part. Therefore, the original development work can done in parallel.
(ii) An individual software engineer can specialize in his or her component, becoming expert at it.
(iii) Each individual component is smaller, and therefore easier to understand.
(iv) When one part needs to be replaced or changed, this can hopefully be done without having to replace.
(v) Each component can be reuseable.

A software system can be divided in many ways.
- A distributed system is divided up into clients & services.
- A system is divided up into subsystems.
- A subsystem can be divided up into one or more packages.
- A package is composed of classes.
- A class is composed of methods.

2. Increase cohesion when possible:-
→ The cohesion principle is an extension of divide and conquer principle. It says, divide things up, but keep things together that belongs together.
→ A subsystem has high cohesion if it keeps things together that are related to each other, and keeps out other things.
→ This makes system as a whole easier to understand and change.
→ There are different types of cohesion
3) - functional cohesion
- layer cohesion
- communicational cohesion.
- procedural cohesion

4)

3) Reduce Coupling where possible:-
→ Coupling occurs when there are interdependencies between one module and another.

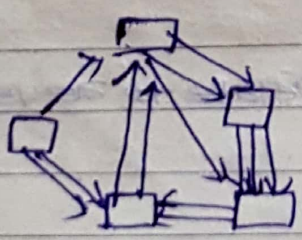


fig: highly coupled system

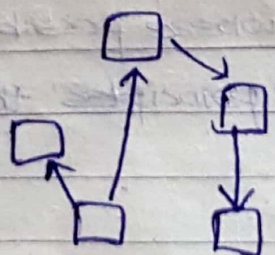


fig: low coupled system

The importance of low coupling are as follows:-
when/independence exist

1) Requiring changes to made in more than one place is problematic since it is time consuming to find the different places that need changing and it is likely that errors will be made.

2) A network of interdependencies make it hard to see how component works.

→ To reduce coupling, you have to reduce the number of connections between modules and the strength of the connections.

These are following types of coupling. They are:-
- Content coupling
- Common coupling
- Data coupling

4) Keep the level of abstraction as high as possible:-

→ We should ensure that our designs allow us to hide consideration of details, thus reducing complexity.

→ Abstraction allow work by allowing us to understand the essence of significant features of something and make important decisions without knowing unnecessary details.

5) Increase reusability where possible

→ The two complementary principles that relate to reuse are
- design for reuse
- design with reuse.

Date
Page

▷ Designing for reusability means designing various aspects of our system so that they can be used again in other contexts, both in your system and in other system.

Important strategies for increasing reusability are as follows.

→ Generalize your design as much as possible.
→ Follow the preceeding three design principles.
→ Design your system to contain hooks. Hook allow other designers to add functionality.
→ Simplify your design as much as possible.

6) Reuse existing designs and code where possible:-

▷ Reusing designs & code allows us to take advantage of ~~their~~ reuse our or others made reusable components

▷ Cloning should normally not be seen as effective form of reuse. Cloning involves copying code from one place to another. It should be avoided since, when there are two or more occurances of the same or similar code in the system any changes made will have to be made in all 'clones'.

◆ So, to clone more than a couple of lines of code, it is normally best to encapsulate the code in a separate method and call it from all the places it is needed.

7) Design for flexibility. (aka adaptability)

▷ It means actively anticipating changes that a design have to undergo in the future and preparing for them.

It can be achieved by:-

- Reducing coupling and increasing cohesion
- Creating abstraction
- Not hard-coding anything
- Leaving all options open
- Using reuseable code and making code reuseable

8) Anticipate Obsolescence:-

▷ It is the special case of design for flexibility.

▷ Changes will inevitably occur in the technology a software system uses and in the environment in which it runs.

▷ Anticipating obsolescence means planning for evolution of the technology or environment so that the software will continue to run or can be easily changes.

The rules that designers can use to better anticipate obsolescence.

- Avoid using early releases of technology
- Avoid using software libraries that are specific to the particular environment.
- Avoid using little-used features of software libraries
- Avoid using reusable software or special hardware from small companies or from those that are less likely to provide long term support
- Use standard languages and technologies that are

supported by multiple versions.

9) Design for portability

→ Portability is ability to have the software run on as many platforms as possible although sometimes this might also be a necessary for survival.

→ It is achieved by avoiding the use of facilities that are specific to one particular environment

10) Design for testability

→ It focuses to design a system so that automatic testing is made easy.

→ It is achieved by ensuring all the functionality of the code without going through the graphical user interface.
i.e separating UI layer from the functional layer

11) Design defensively:

→ It focuses to design a system without defects as we never know how the user uses the component we are designing.

→ It is achieved by checking all the inputs to our component are valid.

(9) Design for portability

-> Portability is ability to have the software run on as many platforms as possible although sometimes this might also be a necessary for survival.

-> It is achieved by avoiding the use of facilities that are specific to are particular environment

(10) Design for testability

-> It focuses to design a system so that automatic testing is made easy

-> It is achieved by ensuring all the functionality of the code without going through the graphical user interface

ie seperating UI layer from the functional layer

(11) Design defensively.

- It focuses to design a system without defects as we never know how the user uses the component we are designing.

- It is achieved by checking all the inputs to our component are valid.

-> Extreme reliability and safety: real-time systems typically control the environment in which they operate; failure to control can result in loss of life, damage to environment or economic loss

-> Guaranteed response time: we need to be able to predict with confidence the worst case response for systems; efficiency is important but predictability is essential.

Real-Time System

→ A. RTS is any information processing system that has to respond to externally generated signal with a finite and specified period.

→ RTS systems are those which must produce correct responses within a definite time limit.

→ A RTS is a computer system where the correct functioning of the system depends on the results produced and the time at which they are produced

Classification of RTSs:

1) HARD REAL-TIME SYSTEM

- A system whose operation is degraded if results are not proceeded according to specified timing requirements.
- System response occur within a specified deadline. failure to meet such a timing requirement can have catastrophic consequences.
- Systems where it is absolutely imperative that responses occur within the required deadline

Example: Flight control systems, automative system robotics etc.

2. SOFT REAL-TIME SYSTEMS

- A system whose operation is incorrect if results are not produced according to the timing constraints. Catastrophic results will happen then.
- The responses times are important but not critical to the operation of the system. failure to meet the timing requirements would not impair/effect the system.

- Systems where deadlines are important, but which will still function correctly if deadlines are occasionally missed. Example: Banking systems, multimedia etc

3) FIRM REALTIME SYSTEMS:

There is no value for a response that occurs past a specific deadline. The computation is obsolete if the job is not finished on time. Failure to meet the timing requirements is undesirable.

Notes:

- A single system may have both hard and soft real-time subsystems.
- In reality many systems will have a cost function associated with missing each deadline.

Characteristics of a RTS:

- Large and complex: vary from a few hundred lines of assembler or C up to 2 millions lines of Ada estimated for the space station freedom.
- Concurrent control of seperate system components: devices operate in parallel in the real-world, better to model this parallelism by concurrent entities in the program.
- Facilities to interact with special purpose hardware: need to be able to program devices in a reliable and abstract way.
- Mixture of hardware/software: Some modules implemented in hardware even whole systems. Such

- Is it true that a software product can always be developed faster by having a larger development team of competent software engineers? Justify your name.
- Object Oriented Software Engineering makes easy working with model and program." Explain it
- Briefly Explain the characteristics of a system development
- classify the real time system and define why it is difficult for testing and verification of real time system.
- How does a software system change during its life cycle. Explain clearly how changes are accommodated.
- Explain the primary objectives of any system development activity.
- Describe the phases followed in waterfall model and also mention its weakness
- Define software engineering and describe its relationship with other areas of computer science and other disciplines.

a "Models represent the abstract of software development". Explain it.

b) Briefly explain the characteristics of a system development

c) Classify the real time system and define why it is difficult for testing & verification of real time system.

- Explain Agility principles of software development. which software development model is best suited for a risk driven software development.
- Can spiral model be used for all types of project? Given an example of development project for which spiral model is not appropriate.
- Object oriented software development life cycle is different from other software engineering." Explain it
- Explain the process and models during a software development?
- Differentiate testing and verification.

- Object oriented software Engineering makes easy working with model and program". Justify
- Explain the characteristics of a system development.
- Classify the real time system and define why it is difficult for testing and verification of real ~~life~~ time system.
- "With the knowledge and implementation of software engineering techniques we can build software that is reliable, efficient and time relevant." Explain.