PROBLEMS THAT MAY OCCUR IN OUR DESIGN

VI. DESIGN PRINCIPLES AND PATTERNS

SYMPTOMS OF ROTTING DESIGN.

These four symptoms (Rigidity, Fragility, immobility and viscosity) are the signs of poor architecture.

Any application that exhibit them is suffering from a design that is rotting from the inside.

SYMPTOMS OF ROTTING DESIGN.

These four symptoms
(Rigidity, Fragility, immobility and viscosity) are the signs of poor architecture.

But what causes the rot to take place?

VI. DESIGN PRINCIPLES AND PATTERNS

SYMPTOMS OF ROTTING DESIGN.

These four symptoms (**Rigidity, Fragility, immobility and viscosity**) are the signs of poor architecture.

But what causes that rot to take place?

- Changing requirements

VI. DESIGN PRINCIPLES AND PATTERNS

SYMPTOMS OF ROTTING DESIGN.

1. Changing requirements problem

- Modification may be requested in a way that the requirement or initial design did not anticipate the changes.

Who should be blame?

- Requirements should not be blamed, as an engineer , you know requirements will change. We must find a way to make our design resilient to such changes and protect it from rotting.

VI. DESIGN PRINCIPLES AND PATTERNS

SYMPTOMS OF ROTTING DESIGN.

2. Dependency Management problem

- Changes that introduced new and unplanned dependencies degrades the architecture.
- In other to solve this, the dependencies in an architecture must be managed.

VI. DESIGN PRINCIPLES AND PATTERNS

SYMPTOMS OF ROTTING DESIGN.

❖ How can we Manage the Dependencies ?

- Can be manage by the creation of dependency firewalls such that across such firewalls, the dependencies do not propagates.

Example

- Object oriented design is known to have:-

➢ Principles and techniques for building such firewalls and managing modules dependencies.

VI. DESIGN PRINCIPLES AND PATTERNS

SYMPTOMS OF ROTTING DESIGN.

- How can we Manage the Dependencies ?
- The **Principles** and **Techniques** that helps maintain the dependency architecture of an application is called the ***Design Principles and Patterns***.

(***A Principle is a fundamental truth that serves as a foundation…)***

VI. DESIGN PRINCIPLES AND PATTERNS

PRINCIPLES OF OBJECT ORIENTED DESIGN.

- (SOLID Principle)

S: Single Responsibility Principle(SRP)

O: Open Close principle(OCP)

L: Liskov's Substitution Principle(LSP)

I: Interface Segregation Principle(ISP)

D: Dependency Inversion Principle(DIP)

PRINCIPLES OF OBJECT ORIENTED DESIGN.

❖ The Open close Principle (OCP)

A **module, Classes, functions**, etc. should be open for extension but close for modification.

- Meaning modules should be written in such a way that it can be extended, without requiring them to be modified.

Example : Extending an already tested working class for it will be very costly to modify

VI. DESIGN PRINCIPLES AND PATTERNS

PRINCIPLES OF OBJECT ORIENTED DESIGN.

❖ The Liskov Substitution Principle (LSP)

- Subclasses should be substitutable for their base classes.
- Functions that use references to base classes must be able to use objects of derived classes without knowing it.

The principle by Barbar Liskov.

She meant that a user of a base class should continue to function properly if a derivation of the base class is passed to it.

VI. DESIGN PRINCIPLES AND PATTERNS

PRINCIPLES OF OBJECT ORIENTED DESIGN.

- **The Dependency Inversion Principle (DIP)**
- This principle depends on abstraction.
- It does not depend upon concretions.
- It is a strategy of depending upon interface or abstract functions and classes, rather than concert functions and classes.
- Depending upon abstractions, the design should target an interface, or an abstract class and not concrete class targeted.

VI. DESIGN PRINCIPLES AND PATTERNS

PRINCIPLES OF OBJECT ORIENTED DESIGN.

- The Dependency Inversion Principle (DIP)

What is the different between a concrete and an abstract class?

VI. DESIGN PRINCIPLES AND PATTERNS

PRINCIPLES OF OBJECT ORIENTED DESIGN.

❖ The interface segregation Principle (ISP)

- Many client specific interfaces are better than one general purpose interface.
- If you have a class that have several clients, rather than loading the class with all the methods, that the client needs, create specific interfaces for each client and multiply inherit them into the class.

PRINCIPLES OF OBJECT ORIENTED DESIGN.

- Principle of Package Architecture
- Classes are necessary but insufficient means of organizing a design.
- Packages are needed to help bring order.

How do we chose which classes belong to which packages?

The are 3 principles that attempt to help the software architect known as **Package cohesion Principle.**

VI. DESIGN PRINCIPLES AND PATTERNS

PRINCIPLES OF OBJECT ORIENTED DESIGN.

❖ Package cohesion Principle

1. The release reuse equivalency Principle (REP)

The granule of reuse is the granule of release

2. The common closure Principle (CCP)

Classes that change together , belong together

3. The common reuse principle (CRP)

Classes that aren't reused together should not be grouped together.

VI. DESIGN PRINCIPLES AND PATTERNS

PRINCIPLES OF OBJECT ORIENTED DESIGN.

❖ Package cohesion Principle

1. The release reuse equivalency Principle (REP)

The granule of reuse is the granule of release.

VI. DESIGN PRINCIPLES AND PATTERNS

PRINCIPLES OF OBJECT ORIENTED DESIGN.

❖ Package cohesion Principle

2. The common closure Principle (CCP)

Classes that change together , belong together.

A large development project is subdivided into a large network of interrelated packages.

The more packages that change in any given release, the greater the work to rebuild, test and deploy the release.

PRINCIPLES OF OBJECT ORIENTED DESIGN.

❖ Package cohesion Principle

2. The common closure Principle (CCP)

Classes that change together , belong together.

For a release

More packages that change $\rightarrow$ The greater the work (To rebuild, test, deploy)

Therefore, it would be necessary to minimize the number of packages that are changed in any given release cycle of the product.

VI. DESIGN PRINCIPLES AND PATTERNS

PRINCIPLES OF OBJECT ORIENTED DESIGN.

❖ Package cohesion Principle

2. The common closure Principle (CCP)

Classes that change together , belong together.

How do we minimize the number of packages that change in any given release cycle?

We group together classes that we think will change together in anticipation.

VI. DESIGN PRINCIPLES AND PATTERNS

PRINCIPLES OF OBJECT ORIENTED DESIGN.

❖ Package cohesion Principle

2. The common closure Principle (CCP)

Classes that change together , belong together.

When we group classes that change together into the same package, the package impact from release to release will be minimized.

VI. DESIGN PRINCIPLES AND PATTERNS

PRINCIPLES OF OBJECT ORIENTED DESIGN.

❖ Package cohesion Principle

3. **The common reuse principle (CRP)**

Classes that aren't reused together should not be grouped together.

- A dependency upon a package is a dependency upon everything within the package.
- When a package changes, and its release is done, all clients of that package must verify they work with the new package- even if nothing they used within the package changed.

VI. DESIGN PRINCIPLES AND PATTERNS

PRINCIPLES OF OBJECT ORIENTED DESIGN.

- Package cohesion Principle

3. **The common reuse principle (CRP)**

Classes that aren't reused together should not be grouped together.

- When a package changes, and its release is done, all clients of that package must verify they work with the new package- even if nothing they used within the package changed.

VI. DESIGN PRINCIPLES AND PATTERNS

PRINCIPLES OF OBJECT ORIENTED DESIGN.

❖ Package cohesion Principle

3. **The common reuse principle (CRP)**

Classes that aren't reused together should not be grouped together.

- When a package changes, and its release is done, all clients of that package must verify they work with the new package- (even if nothing they used within the package changed)because the vendor will not support the older version for ever.

PRINCIPLES OF OBJECT ORIENTED DESIGN.

❖ Package cohesion Principle

3. The common reuse principle (CRP)

Classes that aren’t reused together should not be grouped together.

- Change to a class that I don’t care about will force a new release of the package, and still cause the going through the effort of upgrading and revalidating.