**Pattern based design:**

Create a new application by finding proven solutions for existing problems

Each software solution has different problems, but often problems have been encountered already

It's important to consider context - the environment we are in - to determine most applicable solution

**(Must be very specific, know considerations for design patterns + architectural styles [Theres a difference] for the exam]**

**Considerations**

-when choosing a "**DESIGN PATTERN**", not an "Architectural style before..."

**Economy** - abstract, not related to money: Choose an uncluttered design pattern. Simplistic, should unclutter your solution. Economy because it's based on your situation - in financial economy you won't go to nandos, you'll go to chicken licken if you only have R50. Don't want to get in a further bad situation

**Visibility** - Pick the clear, obvious solutions when you can. Don't go inventing a whole maps system for your project, incorporate google maps. Would just bring in problems and make the system unclear if you don't do that

**Spacing** - separation of concerns, as much as you can.

**Symmetry** - consistency and balancing. Choose a solution that will balance itself out the most. Eg uber is having a problem with security, they must find a solution for security. But don't forget about eg reliability, bringing in problems there. Compensating, not overcompensating

**Emergence** - flexible system that can handle changing behaviour. Will the solution be able to adapt to/cater for changes you can foresee?

**Design Patterns**

Context, Problem, solution are the building blocks of design patterns. All work with one another

**Context** and the problem and the solution are all related.

**Problem** must be solved within the context.

**Context** must be considered within/around the problem.

**Solution** must be considered in context, etc.

Think of parents breaking up a fight between siblings.

The parent solves the problem using a proven concept - their past knowledge, how they've solved previous fights

Solution isn't instantly obvious, need to understand context first - what started the fight etc.

In order to find most applicable solution

**Design patterns** describe a relationship between classes to solve the problem.

Design pattern you choose should be an elegant solution to the issue - work well, not messy, "seamless/effortless"

In your project you have main problems and subproblems.

Can solve subproblems by solving main problem, or vice versa, depending on context.

-Generative Patterns: We know what the problem is, now must find a concrete solution  
Predefined/specified solution: We will know exactly what the solution will be to solve the problem.

-Non-generative patterns: This is where we can describe the problem, but there is no solution for it.

-eg Kinda of like traffic, if theres no other routes you can take, theres nothing you can do. You just have to sit in the traffic. You know the problem:traffic, you know the context: many cars at 8am on busy road, but no solution.

**Abstraction and application (Design Patterns)**

**-Below 5 when looking at above.**

**Architectural Patterns:** Are for architectural problems whenever there is a structure involved.

**Component patterns:** Address problems when components of systems or subsystems involved. How they work ect. (the way the **communicate** with one another. **Not their development**)  
  
**Interface Design Patters:** Common user interface problems, and what are the solutions for those interfaces. (Anything interface related) (User related or component level and their **type** of communication with one another)

**WebApp patterns:** When developing WebApp

**Mobile Patterns**: When developing mobile apps

(The below 3 in exam **NB – You don’t want 5 above in exam** – probs asked to give a pattern)

**Creational Patterns:** Object creation, object orientation. Creating objects that can be used to solve a specific problem.  
Composed of 2 main ideas:   
-Incapsulation: OO approach  
-Hiding: How instances of concrete classes are defined and hid

**Structural Patterns:** Design patterns that ease the design by relising. How classes and object talk to one another.  
  
**Behavior** **Patterns**: Identifies common communication between objects. Increase the flexibility in carrying out communication.

**Design Patterns (In exam: Given case study, what design pattern would u choose for this specific case study: Then u apply one of above 3, then one of the below eg Visitor (which is part of behavioral pattern). You describe behavioral + visitor, then what visitor does. Relate to case study)  
-We will need to draw pattern  
   
Bevaioral Patterns:  
  
-Visitor**

A way to separate algorithms from the specific object.  
Allows adding new virtual functions without modifying specific classes.  
Adding behavior to existing classes without changing existing classes.

**Problems can solve: When Classes make up ob**

**-Observer:** here objects are named subject. Notifies any change in state  
-When constant updates and notifications are required  
-Mainly used for event handling system.

**-Command**

**Structural Patterns:**

**-Proxy:** Class that functions as an interface to something else. A wrapper agent that is called by client to access real service object behind the scenes.

**Façade:** Software design where an object serves as the front, hiding the complex system.

**Creational Patterns:**

**Abstract Factory:** Allows encapsulating group of factories, without specifying concrete class itself.  
  
**Factory Design:**

**Singleton Design: Wont really use this, as nt applicable in Use Cases.**

**--------------------------------------------------------------**

**For exam, must know: Generative, Non-generative patterns, types of design patterns (3), and Visitor ect.  
--------------------------------------------------------------**

**Framework:**

Where patterns are not sufficient.

Requires implementation-specific skeletal infrastructure: Reusable architecture that give some type of generic behavior for a family of abstractions for a specific context.  
-Not a pattern: But rather a collection of hooks and slots.  
  
vs Design Patterns  
- Design Patterns are More abstract: More broad.  
- Design Patterns are Smaller architectural elements.  
- Design Patterns are Less specialized.

**Pareidolia**: The tendency for perception to impose some type of meaning/pattern, where there is none.

**Pattern Based Software Design**

When looking at patterns:  
Understand the big picture.   
Examine big picture and extract patterns.  
Begin with big picture  
Work inwards  
Refine solution

**Pattern Based Software Design Tasks**

-Went through quickly (text)

**Pattern-Organizing Table**

-Enough time taken to understand the underlying problem and its context, is how you can select the most appropriate pattern.  
-Common mistake: Don’t give it enough time to understand problem  
  
  
  
**Patterns in all places**

**User Management  
-**Registration  
-Log in  
-Forgot password  
-User Settings  
  
**User Interface Design Patterns**

**-Kinda skipper through rest**