Design concepts

A set of fundamental software design concepts has evolved over the history of software engineering are as follows.

1. Abstraction

- Solution to any software problem represented with many levels of abstraction.
- At Highest level of abstraction state the solution in broad terms.
- At lower level of abstraction more detailed descriptions are added to solution.
- At lowest level of abstraction solution is stated in a manner, that can be directly implemented.
- We can create procedural abstraction and data abstraction with different levels.

- Procedural abstraction: it is a specific sequence of instructions with limited function. For example, open a door is a procedural abstraction, the word open implies long sequence of instructions.
- Data abstraction: it is a collection of data that describes data object. In case of procedural abstraction open the door, we can define data abstraction for door, that contains various properties of door.

2. Architecture

- Software architecture gives us “the overall structure of the software”.
- In simple terms it is the organization of components, interaction among the components and the structure of data used by those components.
- Architecture gives us framework from which more design details are added.
- Properties are to be considered as part of architectural design:

    - Structural properties define the components of the systems, and how these components are connected and interact with one another.
    - Extra-functional properties address how the architecture meets non-functional requirements.
    - Families of related systems addresses the ability to reuse architectural building blocks.

3. Patterns

- Design pattern is a general repeatable solution to a common problems in software design.
- It is a description or template for how to solve a problem that can be used in many different situations.

4. Separation of Concerns

- It is suggested that any complex problem can be easily handled by dividing it into pieces and solve them independently.
- A concern is a feature or behaviour that is specified in the requirements model.

5. Modularity

- Software is divided into separately named and addressable components, sometimes called modules.
- Modules are to be integrated later to satisfy system requirements.
- Modularity is the single attribute of a software that permits a program to be managed easily.
- Modularity makes understanding of design modules easier, as a result it reduces the cost of the software to be built.

6. Information hiding

- The basic principle of information hiding is thet modules must hide from one another.
- Effective modularity can be achieved by defining modules as much as independent as possible.
- Access constraints are enforced on both procedural details and local data structure of a module.
- Information hiding provides the greatest benefits when modifications are required in software as part of maintenance.

7. Functional independence

- To achieve functional independence modules are to be developed with “single minded” function and little interaction with other modules.
- In simple terms, modules should be designed in a manner that they should address specific requirement and has simple interface with other nodules.
- Independent modules are easy to maintain as modifications and error propagations are limited, and can be reusable.

The functional independence is accessed using two criteria: Cohesion and coupling.

i. Cohesion

- it is an indication of relative strength of a module and is natural extension to information hiding.
- A cohesive module performs a single task and has less interaction with the other modules.
- A good design should always strive to achieve high cohesion.

ii. Coupling

- Coupling is an indication of interconnection between modules in a structure of software and depends on interface complexity.
- A good design should always strive to achieve low coupling.

8. Refinement

- Refinement is a process of elaboration.
- Abstraction and Refinement are complementary concepts. Where abstraction suppresses internal details and refinement reveals internal details of modules.

9. Aspects

- During the requirement model each requirement is considered independently, but in practice requirements cannot isolated easily.
- An aspect is a representation of a cross-cutting concern. It means that when A and B are two requirements, B cannot be satisfied without considering A.
- During the design process requirements, A and B are refined into A* and B*.
- Here the design concern is that B* cross-cuts A*.

10. Refactoring

- It is a reorganization technique which simplifies the design of components without changing its function behaviour.
- Refactoring is the process of changing the software system in a way that it does not change the external behaviour of the code still improves its internal structure.

11 Object-Oriented Design Concepts

- The object-oriented (OO) paradigm is widely used in modern software engineering.
- The object-oriented concepts are classes and objects, inheritance, messages, and polymorphism and other.

12. Design classes

- The design model evolves, we shroud define set of design classes such as user interface classes, business domain classes, process classes, persistent classes, system classes.
- Design classes provide more technical detail and they act as a guide for implementation.

Text Books

1. Roger Pressman S., “Software Engineering: A Practitioner's Approach”, 7th Edition, McGraw Hill, 2010.
2. Sommerville, “Software Engineering”, Eighth Edition, Pearson Education, 2007

Web Links:

1. https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-355j-software-engineering-concepts-fall-2005/lecture-notes/cnotes4.pdf
2. https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-355j-software-engineering-concepts-fall-2005/lecture-notes/cnotes5.pdf
3. https://drive.google.com/file/d/1-e8kYCqYRhk1Dg_JKdSXbcWNNZXxf632/view
4. https://cdn.shopify.com/s/files/1/0457/4009/7694/files/software_engineering_pdf_pressman_7th_edition.pdf