Software Engineering Principles

WEEK 7

Software Design Principles

Software design principles are concerned with providing means to handle the complexity of the design process effectively. Effectively managing the complexity will not only reduce the effort needed for design but can also reduce the scope of introducing errors during design.

Divide and Conquer (Decomposition)

- Break problem into pieces to reduce complexity
- Finding the right piece is an art
- Without this two bad things can happen
- Trap 1
  - Solving the pieces may not solve the problem
- Trap 2
  - Decomposition of difficult part again may waste your time

Guidelines for seeking breakdowns

- Independent Parts
  - For concentration on one part without thinking on another
- 7 ± 2 Rule
  - Human keep intellectual control of collection best when the control five to nine parts
  - For better solution and command----Few items in decomposition
- Hierarchies
  - All parts must be simple and small
  - Problem decomposition can be drawn as a hierarchical tree
- Knowing when to stop

Rigor and Formality

- Creativity often leads to imprecision and inaccuracy
- Many people of differing skill-sets and interests are involved in the Software development process. Without set rules, each developer imposes his/her own interest on the project. When problems occur, they become difficult to resolve.
- The *rigor* of a software development project is achieved by setting rules into the process. Every person involved in the project has to observe the rules. With rigor, a project can carry on smoothly without hindrances.
- There are various degrees of rigor. The highest level of rigor is *formality* – a situation where software systems can be verified by mathematical laws.

Separation of concerns

- Separation of concerns is a software architecture design pattern/principle for separating an application into distinct sections, so each section addresses a separate concern.
- Decisions that must be made in software development
  - Features of product, special hardware software resources required
  - Development process, teams, scheduling etc.
  - Economic and financial methods
  - Some concerns are interrelated and interdependent
- Isolate issues/concerns that are not closely related to others
- Qualities may be consider as separate concern
  - Correctness and performance
- Separation of concerns has been known to enhance the visibility, understandability and maintainability of software systems.

Separation of concerns

Time

- Requirements, design, implementation, testing, ...

Qualities

- Efficiency and user friendliness
- Correctness and portability

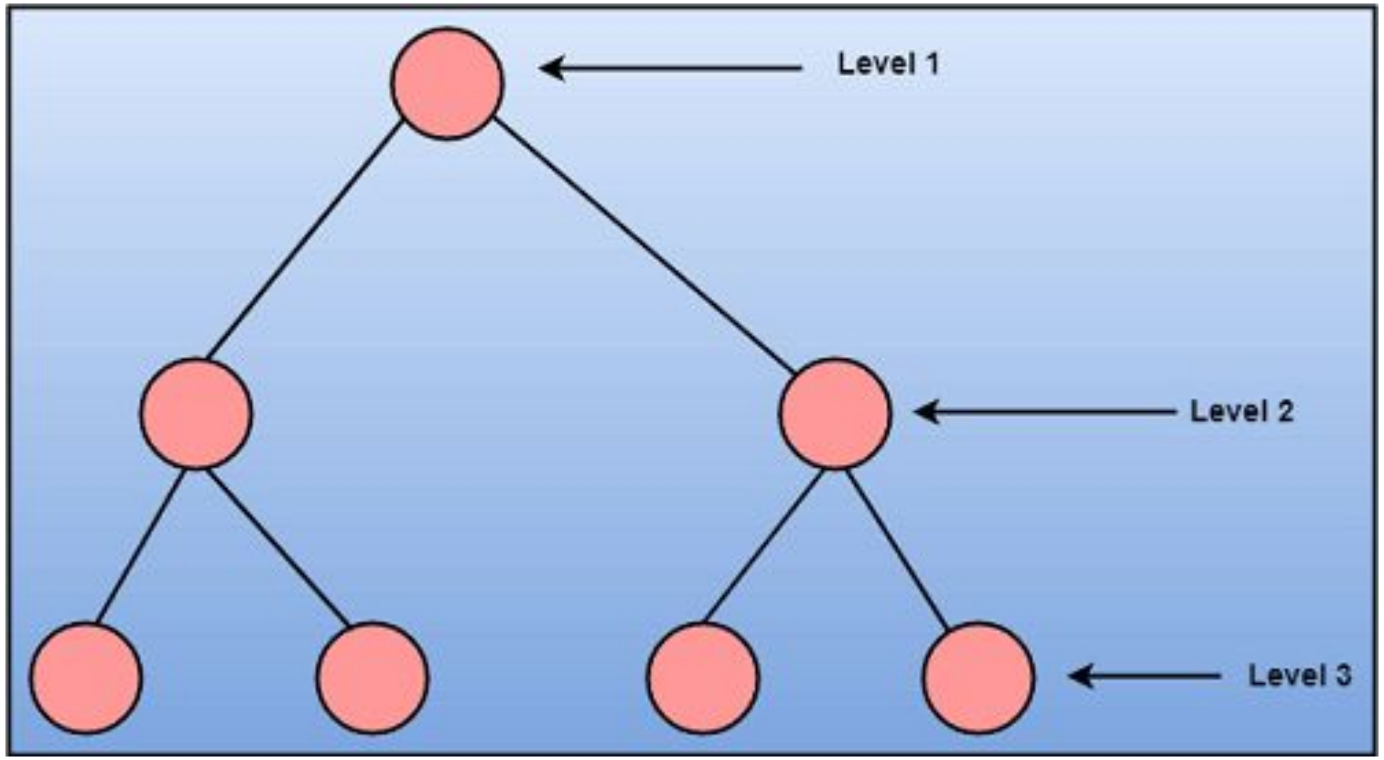
Views

- Data flow and control flow
- Management and development

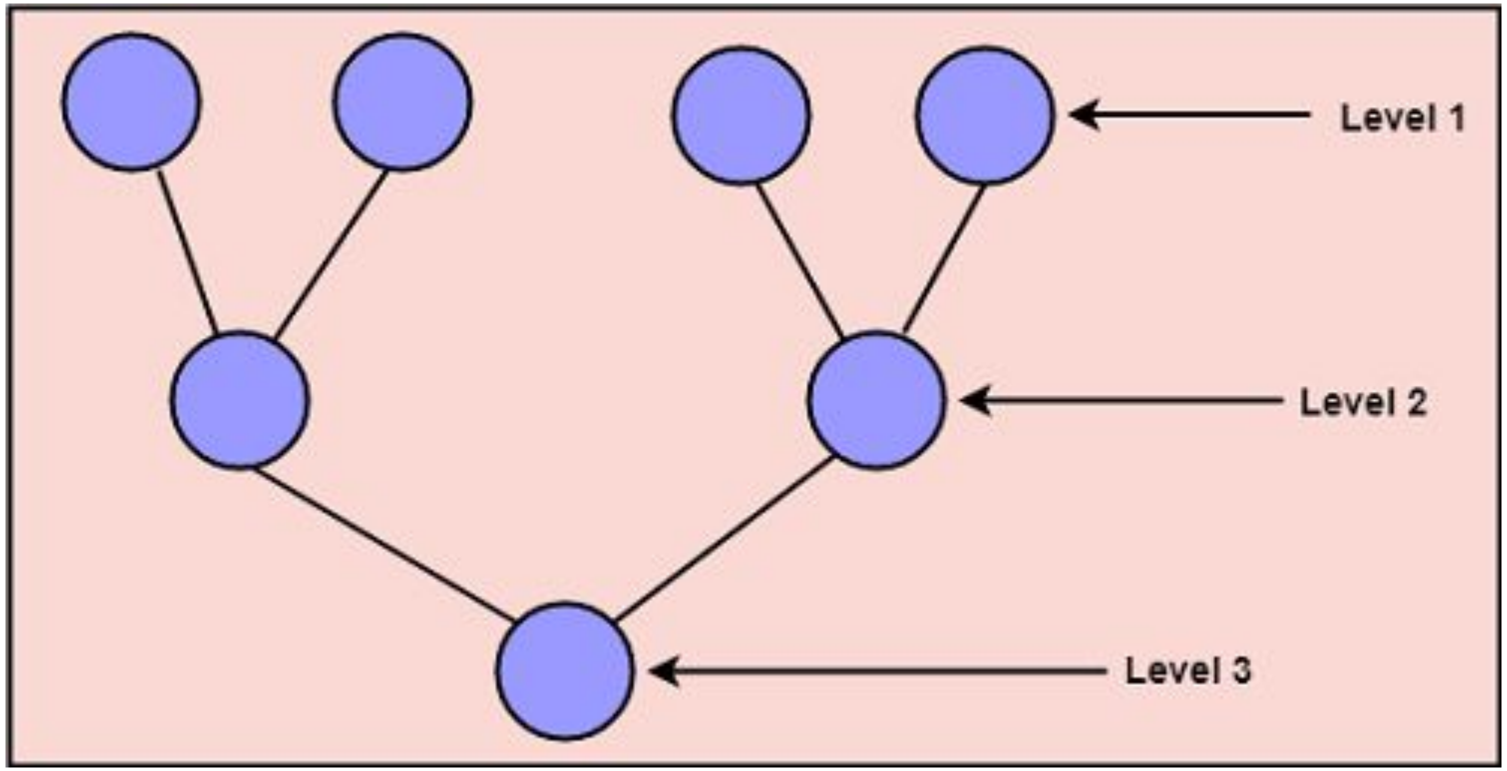
Modularity

- A complex system may be divided into simpler pieces called modules
- A system that is composed of modules is called modular Support application of separation of concerns
- When dealing with a module we can ignore details of other modules
  - Top-down
    Decompose the whole design into modules first and then concentrate on individual module design
  - Bottom-up
    Concentrate on modules and then on their composition

Top-down



Bottom-up Approach



Modularity

Three goals with modularity

- **Decomposability**
- Break the system down into understandable modules
- Divide and conquer
- **Composability**
- Construct a system from smaller pieces
- Reuse, ease of maintenance
- **Ease of understanding**
- The system will be changed; we must understand it
- Understand in pieces versus understanding the whole

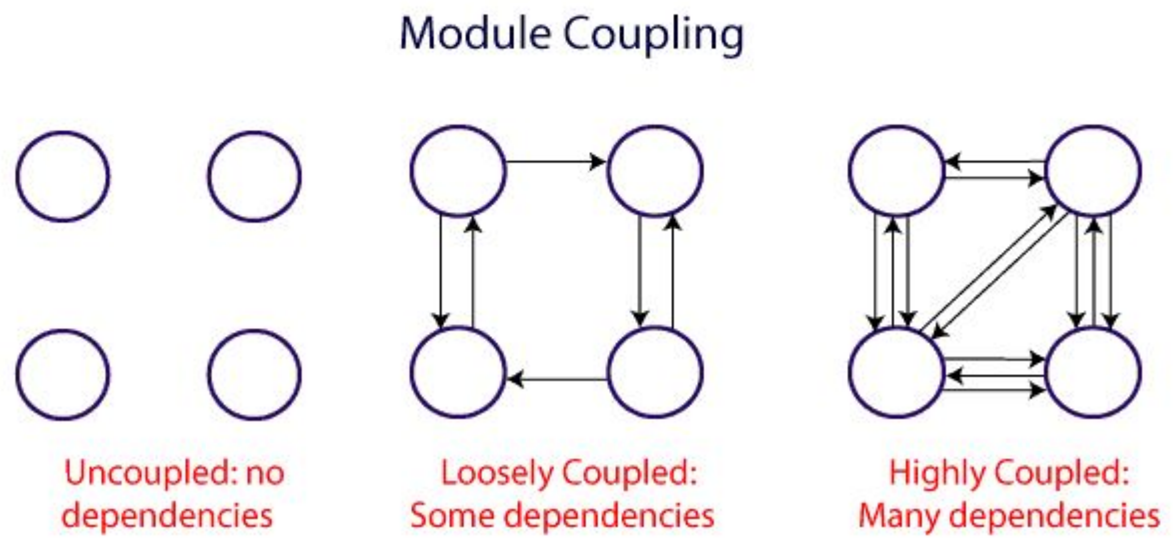
Abstraction

- The principle of abstraction is another specialization of the principle of separation of concerns
- principle of abstraction implies separating the behavior of software components from their implementation. It requires learning to look at software and software components from two points of view: what it does, and how it does it.
- Identify the important aspects and ignore the details
  - Must have different abstractions of the same reality
  - Provide different views
  - **Examples of abstraction**
  - Design notations
  - Project planning
  - **Two key concepts**
  - Information hiding and data encapsulation

Coupling

In software engineering, the coupling is the degree of interdependence between software modules. Two modules that are tightly coupled are strongly dependent on each other. However, two modules that are loosely coupled are not dependent on each other. **Uncoupled modules** have no interdependence at all within them.

A good design is the one that has low coupling. Coupling is measured by the number of relations between the modules. That is, the coupling increases as the number of calls between modules increase or the amount of shared data is large. Thus, it can be said that a design with high coupling will have more errors.


Module Coupling
Uncoupled: no dependencies
Loosely Coupled: Some dependencies
Highly Coupled: Many dependencies

Module Cohesion

In computer programming, cohesion defines to the degree to which the elements of a module belong together. Thus, cohesion measures the strength of relationships between pieces of functionality within a given module. For example, in highly cohesive systems, functionality is strongly related.

Cohesion is an **ordinal** type of measurement and is generally described as "high cohesion" or "low cohesion."