SOFTWARE ENGINEERING (15B11CI513)

Credits :- 4 **Contact Hours :-** 3-1-0

Cohesion and Coupling

Cohesion

- Cohesion refers to the degree to which the elements of a module belong together.

Types of Cohesion

- **Coincidental cohesion (worst)**: the parts of a component are not related but simply bundled into a single component. Harder to understand and not reusable. (e.g. a "Utilities" class).
- **Logical association**: similar functions such as input, error handling, etc. put together. Functions fall in same logical class (like input). May pass a flag to determine which ones executed. interface difficult to understand. Code for more than one function may be intertwined, leading to severe maintenance problems. Difficult to reuse.
- **Temporal cohesion**: Temporal cohesion is when parts of a module are grouped by when they are processed - the parts are processed at a particular time in program execution (e.g start up or shut down, are brought together. Initialization, clean up.)Functions weakly related to one another, but more strongly related to functions in other modules so may need to change lots of modules when do maintenance.
- **Procedural cohesion**: (e.g. a function which checks file permissions and then opens the file).

- **Communicational cohesion**: operate on same input data or produce same output data. May be performing more than one function. Generally acceptable if alternate structures with higher cohesion cannot be easily identified. Still problems with reusability.
- **Sequential cohesion**: output from one part serves as input for another part. May contain several functions or parts of different functions.
- **Functional cohesion (best)**: Functional cohesion is when parts of a module are grouped because they all contribute to a single well-defined task of the module. Each part necessary for execution of a single function. e.g., compute square root or sort the array.
- first two types of cohesion are inferior; communicational and sequential cohesion are very good; and functional cohesion is superior.

Coupling

- Coupling is a measure that defines the level of inter-dependability among modules of a program. It tells at what level the modules interfere and interact with each other. The lower the coupling, the better the program.

Types of Coupling for procedural languages

- Levels of coupling, namely –
  - **Content coupling (high) -** (also known as **Pathological coupling)** when one module modifies or relies on the internal workings of another module (e.g., accessing local data of another module). In this situation, a change in the way the second module produces data (location, type, timing) might also require a change in the dependent module.
  - **Common coupling-** When multiple modules have read and write access to some global data, it is called **common** or **global coupling**. Changing the shared resource might imply changing all the modules using it.
  - **External coupling-** occurs when two modules share an externally imposed data format, communication protocol, or device interface. This is basically related to the communication to external tools and devices.
  - **Control coupling-** Two modules are called control-coupled if one of them decides the function of the other module or changes its flow of execution or Control coupling is one module controlling the flow of another, by passing it information on what to do (e.g., passing a what-to-do flag).

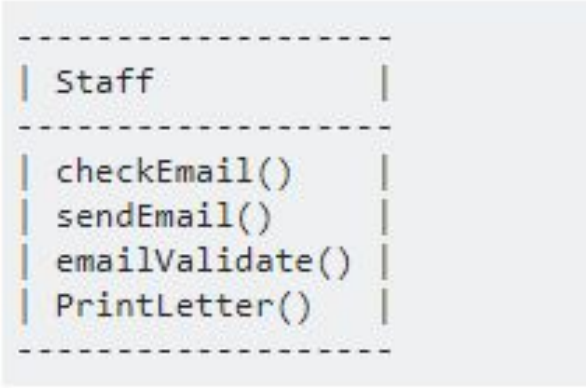
- **Stamp coupling (Data-structured coupling)-** When multiple modules share common data structure and work on different part of it, it is called stamp coupling (e.g., passing a whole record to a function that only needs one field of it).
- **Data coupling-** Data coupling is when two modules interact with each other by means of passing data (as parameter). If a module passes data structure as parameter, then the receiving module should use all its components. (e.g., passing an integer to a function that computes a square root).
- **Message coupling (low)-**This is the loosest type of coupling. It can be achieved by state decentralization (as in objects) and component communication is done via **parameters or message passing**.

Coupling in Object-oriented programming

- **Subclass Coupling-** Describes the relationship between a child and its parent. The child is connected to its parent, but the parent is not connected to the child.
- **Temporal coupling-** When two actions are bundled together into one module just because they happen to occur at the same time.
  - Ideally, highly cohesive and loosely coupled is considered to be the best.

- **Cohesion** refers to what the class (or module) can do. Low cohesion would mean that the class does a great variety of actions - it is broad, unfocused on what it should do. High cohesion means that the class is focused on what it should be doing, i.e. only methods relating to the intention of the class.
- As for **coupling**, it refers to how related or dependent two classes/modules are toward each other. For low coupled classes, changing something major in one class should not affect the other. High coupling would make it difficult to change and maintain your code; since classes are closely knit together, making a change could require an entire system revamp.
- Good software design has **high cohesion** and **low coupling**.

Example of Low Cohesion:



Example of High Cohesion: