Software Engineering - Final Notes

Software Desing

- Unified Modeling Language (***UML***): The standard tool for visualizing, specifying, and documenting the artifacts of an object-oriented software.
  - Not a programming language, just a visual design notation
  - Independent of implementation language
  - **Use Case Diagrams** (User View)
    - The ***behavior*** of the system, during requirements capture and analysis
    - Provide a way for ***developers***, ***domain experts*** and ***end-users*** to communicate
  - **Class Diagrams** (Structural View)
  - **Interaction Diagrams**
    - *Sequence Diagram*
    - Collaboration Diagram (Behavioral View)
  - **State Transition Diagram** (Behavioral View)
  - **Activity Diagram** (Behavioral View)
  - **Implementation Diagrams**
    - Component Diagram (Implementation View)
    - Deployment Diagram (Enviroment View)
- Software Design Contents
  - Component
    - Any piece of software or hardware that has a clear role
  - Module
    - A component that is defined at the programming language level
  - Modularity
    - A system is composed of modules is called modular.
    - When dealing with a module we can ignore details of other modules.

- Abstraction
  - A means of achieving stepwise refinement by suppressing unnecessary details.
  - Code Abstraction
  - Procedural Abstraction
- Layers of software design
  - Architectural Design
  - Modular Design
    - Data Design
    - Behavioral Design
  - User Interface Design

- Object Oriented Design
  - Steps
    - Complete the class diagram
    - Perform the detailed design
  - Principles
    - Software should be ***modular***.
    - Modules should be in a ***hirerchical*** organization.
    - Contain both ***data*** and ***procedural abstractions.***
    - Low coupling, high cohesion.
    - Must be an understandable guide for coders, testers, and maintainers.
  - Strategies
    - Stepwise refinement
    - Top-down design
      - First design very high level structures of system
      - Then gradually work down to detailed decisions about low-level constructs
    - Divide and conquer
      - Separate people can work on each part.

- Cohesion
  - A measure of dependencies within a module
  - If a module contains many closely related functions, its cohesion is high.
  - The designer should aim ***high*** cohesion
- Coupling
  - A measure of dependencies between modules
  - If two modules are strongly coupled, it is hard to modify one without modifying the other.
  - The designer should aim ***low*** coupling

Implementation and Testing

- Implementation / Programming Guide
  - Good Programming
    - Use ***consistent*** and ***meaningful*** variable names
  - ***Tags*** in Document Comments
    - Class and interface descriptions
      - @author KEO
      - @version 1.0.0
    - Method descriptions
      - @param name description
      - @return
      - @exception e end of file error
  - ***Nesting of if*** statements should not exceed a depth of 3, except with prior approval from the team leader
  - ***Modules*** should consist of between 35 and 50 statements, except with prior approval from the team leader
  - Use of "***goto***" should be avoided, except with prior approval from the team leader, may be used for error handling

- Software Testing Concepts
  - ***Definition***: Testing is the process of executing a program to find errors.
  - ***Test case***: a collection of input data and expected output.
  - ***Unit Testing***
    - Does each module do what it supposed to do?
    - Environment
      - ***Stub*** modules
        - Prints a message or
        - Returns pre-determined values from pre-planned test cases
      - ***Driver*** modules
        - Calls its stub modules and checks the results
    - ***Verification***: Are we building the product right?
    - ***Validation***: Are we building the right product?
    - Static Analysis
      - Hand execution (reading source code)
      - Code inspection
      - Automatic tools checking for syntactic and semantic errors (like PyCharm)
    - Dynamic Analysis
      - ***Black-Box*** Testing
        - Test the input output behavior
        - Used for both ***verification*** and ***validation***
        - Focuses on ***functional requirements***
        - Test center values and boundary values
      - ***White-Box*** Testing
        - Test the internal logic
        - Used for ***verification***
        - Ensure that all statements and conditions have been executed at least once.

    - ***Code coverage***: At a minimum, every line of code should be executed by at least one test case.
    - ***Cyclomatic Complexity***
        - Number of regions in the flow chart
        - V(G) = E - N + 2 (#Edges - #Nodes + 2)

- ***Integration Testing***
    - Do you get the expected results when the parts are out together?
- ***System Testing***
    - Does it work within the overall system?
    - Does the program satisfy the requirements?
- ***Integration and Strategies***
    - ***Bigbang Approach***
        - All modules are fully implemented and combined as a whole, then tested as a whole.
        - Not practical
    - ***Incremental Approach*** (Top-Down or Buttom-Up)
        - Program is constructed and tested in small clusters
- Other Tests
    - Functional Testing
    - Regression Testing
    - Alpha-Beta Testing
    - Performance Testing
    - Stress Testing
    - Volume Testing
    - Security Testing
    - Acceptance Testing