Software Engineering Introduction

- What is Software Engineering and SE as a profession?
  - Software Engineering is the **process** of solving customers' problems by developing software systems **within cost, time** and other constraints
  - SE as a profession requires:
    - A relevant education and continued learning
    - Have a holistic or system life-cycle focus
    - Work to industry standards
    - Have a balanced approach to technical task
- Importance of Software Engineering in the IT industry
  - The economies of nations are dependent on software
  - More and more systems are software controlled
  - SE represents a significant fraction in all developed countries

Software Development Process

- What is software process?
  - A series of steps, road maps, that helps us create a timely and high-quality result.
- Software process model
  - An abstract representation of a process: it describes a process and guides the software development team
- Four types of process model
  - Linear: Waterfall
  - Incremental: Incremental model, RAD(Rapid Application Development)
  - Evolutionary: Spiral, Prototyping Model
  - Other: CBSE

Waterfall model (i.e. linear)

- Advantages:
  - Easy
  - Structured
- Disadvantages:
  - Sequential, does not reflect reality
  - Does not produce a prototype: Little feedback from the user, takes a long time to develop
- Definition: Follows a systematic, sequential approach to SW development that begins at the system level and progresses through analysis, design, code and testing.

Incremental

- Requirements are well defined first, software is completed in small increments
- Combines the characteristics of waterfall and the iterative nature of the prototyping model

Evolutionary Process Models

- Development and validation activities are carried out concurrently with rapid feedback
- Core requirements are well understood, but additional requirements are evolving and changing fast
- Advantages:
    - Iterative
    - Allows feedback
    - Provide steady, visible progess to customer
- Disadvantages:
    - Time estimation is difficult
    - Project completion date may be unknown

Prototype Model:

- A quick design, developed very quickly, but disadvantages include performance, reliability

Spiral Model:

- Iterative (like prototype) and controlled (waterfall)
- Advantages:
    - Can be combined with other models
    - Provides early warning of problems
- Disadvantages:
    - More complex
    - Requires more management

Concurrent Model:

- Explicitly use the **divide and conquer** principle

CBSE

- Reuse is a development objective, CBSE emphases on composing solutions from prepackaged software components or classes
- CBSE components are independent so that they do not interfere with one another
- Components implementations are hidden
- Reducing development efforts / costs

Object Oriented Programming

Advantages of OO Paradigm

- Easier to maintain

- Objects are potentially reusable components

Inheritance

Advantages:
- Abstraction mechanism to classify entities
- Supports reuse
- provides a mechanism to extend or refine a class

Polymorphism

- An OO feature that allows subclass to provide a specific implementation of a method that is already provided by one of its superclass (method overriding)

Roles of Variables

- Constant / Fixed value
  - Write once and read many
  - A variable initialized without any calculation and whose value does not change thereafter
- Stepper (incrementor, i++)
  - A variable stepping through values that can be predicted
- Most-Recent Holder (latest input value)
  - Most recent member of a group, or simply latest input value
- Gatherer (accumulator)
  - Purpose: Accumulates values seen so far
  - A variable accumulating the effect of individual values in going through a succession of values
- Transformation
  - Purpose: Compute value in new unit
  - A variable that always gets its new value from the same calculation
- One-way Flag (Boolean, True/False)
- Temporary (tmp value variable)
  - e.g. temp in swap(a, b)
- Organizer (Data Container, Array, List)

Class Association

- Composition (Black Diamond)
  - A has B
  - Must / Compulsory
- Aggregation (white Diamond)

    - A has B
    - Optional
- Association (Just a line)
    - A uses B

Use Case Diagram

Sequence Diagram

Class Diagram

Design Principles

- Single Responsibility Principle (SRP)
    - Object classes should have a singlek responsibility (single purpose)
    - If you have a class that does two things, you should split it
    - There should never be more than one reason for a class to change
- Open-Closed Principle (OCP)
    - open for extension, closed for modification
- Liskov Substitution Principle (LSP)
    - Derived subclasses (sub-types) must be completely substitutable for their parent class (base type)
- Interface Segregation Principle (ISP)
    - Many client-specific interfaces are better than one general-purpose interface
    - Avoid classes with too many responsibilities
    - Divide a large interface into a set of smaller interfaces, but make sure that each interface should be self-contained
- Dependency Inversion Principle (DIP)
    - A super class in an inheritance hierarchy should not know any information about its subclasses
    - Classes with detailed implementations must depend upon their abstractions, but not vice versa
    - DIP using abstract/virtual classes is a way to achieve OCP
    - High level modules should not depend on low level ones (button and lamp)
    - Use interface/abstract for better extension
- Law of Demeter (LoD)
    - Each unit should only have limited knowledge about other units: only about units "closely" related to the current unit.

Design Patterns

- State Pattern
  - Ability to change to another state with ease
  - According to different classifications at run-time
- Strategy Pattern
  - Similar to State Pattern, but with only 1 function
- Singleton Pattern
  - Need only one instance for a class, prevents creating more than one object
- Factory Pattern
  - Defines an interface for creating object
  - Leaves the "choice" of its type to the subclasses
  - creation of object at run-time
- Facade Pattern
  - Provide a simplified interface to the other code
  - Reduce dependencies of outside code
- Observer Pattern
  - The current state/change of the objects need to be informed
  - A subject has to be observed by one or more objects
- Command Pattern
  - Encapsulate commands(method calls) in objects
  - Allows saving requests in a queue for reference (log)

Professional Ethics

- four virtual
  - Prudence
  - Temperance
  - Fortitude
  - Justice
- five P's
  - Purpose
  - Pride
  - Patience
  - Persistence
  - **Perspective** is the inner guidance gained from the other P's that allows us to see the issue more clearly

Eight code of Ethics

- Public interest
- Client and employer: best interest
- Product: highest standards
- Judgement: integrity and independence
- Management: ethical (e.g. equal opportunity and management)
- Profession
- Colleagues: be fair and supportive
- Self: lifelong learning

Why is code of ethics important to software engineers

- Powerful instruments for professionalism and safeguards
- Educate and inspire profession members
- Inform the public about the responsibilities
- Instruct the engineers about the standards of the society
- offer practical advise about issues that matter to professionals

Todo:

- 1. Use case specifications
  - Alternative course of events (Check your assignment)
  - Pre and Post conditions
  - Actions for actors and system
- 2. Sequence Diagram
- 3. Roles of Variables
- 4. Design Patterns and Design Principles
- 5. Check the final slides again for the asterisk.