Software Engineering (CSE3004)

Software Development life cycle (SDLC)

Puneet Kumar Jain

CSE Department

National Institute of Technology Rourkela

Reference

- Most of the slides belongs to the content prepared by Prof Rajib Mall, Fundamentals of Software Engineering, PHI, 2014

- Introduction to SDLC at wikipedia

  https://en.wikipedia.org/wiki/Systems_development_life_cycle

- Rest is mentioned on the particular slide

Software development process

- **Software Development Life Cycle (SDLC):** SDLC is the process of dividing software development work into distinct phases to improve design, product management, and project management..

- The main idea of the SDLC has been "to pursue the development of software systems in a very deliberate, structured and methodical way, requiring each stage of the life cycle—from inception of the idea to delivery of the final system—to be carried out rigidly and sequentially“

Ref: https://en.wikipedia.org/wiki/Systems_development_life_cycle

Software Development Activities

1. Stack Holder Analysis
2. System development requirements (Requirement analysis)
3. Feasibility study
4. System Design
5. Coding and Integration
6. Debugging and Testing
7. Implementation and post implementation

1. Stakeholder Analysis

Stakeholder Management is the process by which you identify your key stakeholders and win their support.

A stakeholder-based approach gives you four key benefits:

1. Getting Your Projects Into Shape:

Opinions of most powerful stakeholders help to define projects at an early stage.

2. Winning Resources

Gaining support from powerful stakeholders can help you to win more resources, such as people, time or money.

3. Building Understanding

Ensure that stockholders fully grasp what you're doing and understand the benefits of your project.

4. Getting Ahead of the Game

Understanding your stakeholders means that you can anticipate and predict their reactions to your project as it develops.

How to Conduct a Stakeholder Analysis

- There are three steps to follow in Stakeholder Analysis.
- **Step 1: Identify Your Stakeholders:**
  - Anyone who operates the system
  - Anyone who benefits from the system
  - Anyone involved in purchasing or procuring the system.
  - Organizations which regulate aspects of the system
  - People or organizations opposed to the system

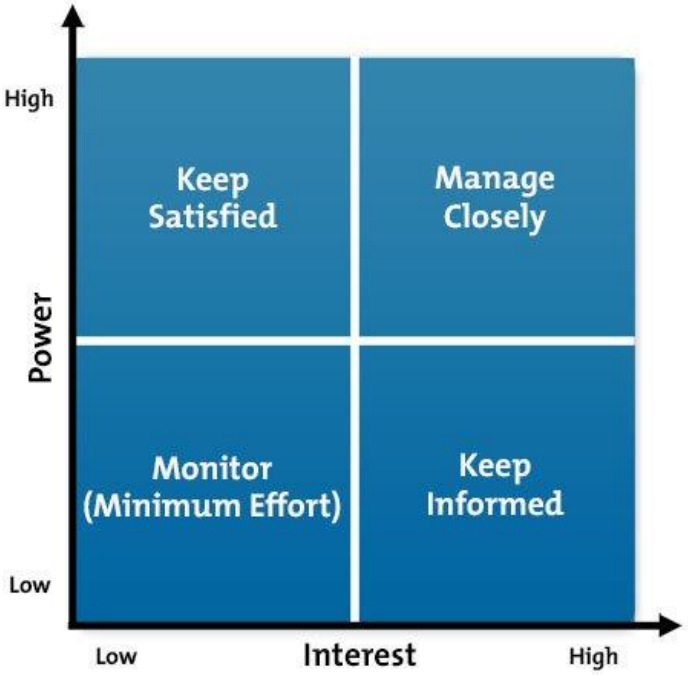
Types of stakeholders include:

- **Key stakeholders:** those with significant influence upon or importance within an organization
- **Primary stakeholders:** those ultimately most affected by an organization's actions
- **Secondary stakeholders:** persons or organizations who are indirectly affected by an organization's actions
- **Tertiary stakeholders:** those who will be impacted the least

How to Conduct a Stakeholder Analysis

- **Step 2: Prioritize Your Stakeholders**
  - Some of these may have the power either to block that work or to advance it.
  - Some may be interested in what you are doing,
  - while others may not care
- Classify according to their power and interest on a Power/Interest Grid.

Figure: Power/Interest Grid for Stakeholder Prioritization



Ref: Mendelow, A.L. (1981). 'Environmental Scanning - The Impact of the Stakeholder Concept,' ICIS 1981 Proceedings

How to Conduct a Stakeholder Analysis

- **Step 3: Understand Your Key Stakeholders**
  - How key stakeholders feel about the project?
  - Key questions that can help you understand your stakeholders include:
    - What financial or emotional interest?
    - What motivates them most of all?
    - What information do they want?
    - What is their current opinion of your work?
  - If they aren't likely to be positive, what will win them around to support your project?
  - If you think that you'll be not be able to win them around, how will you manage their opposition?

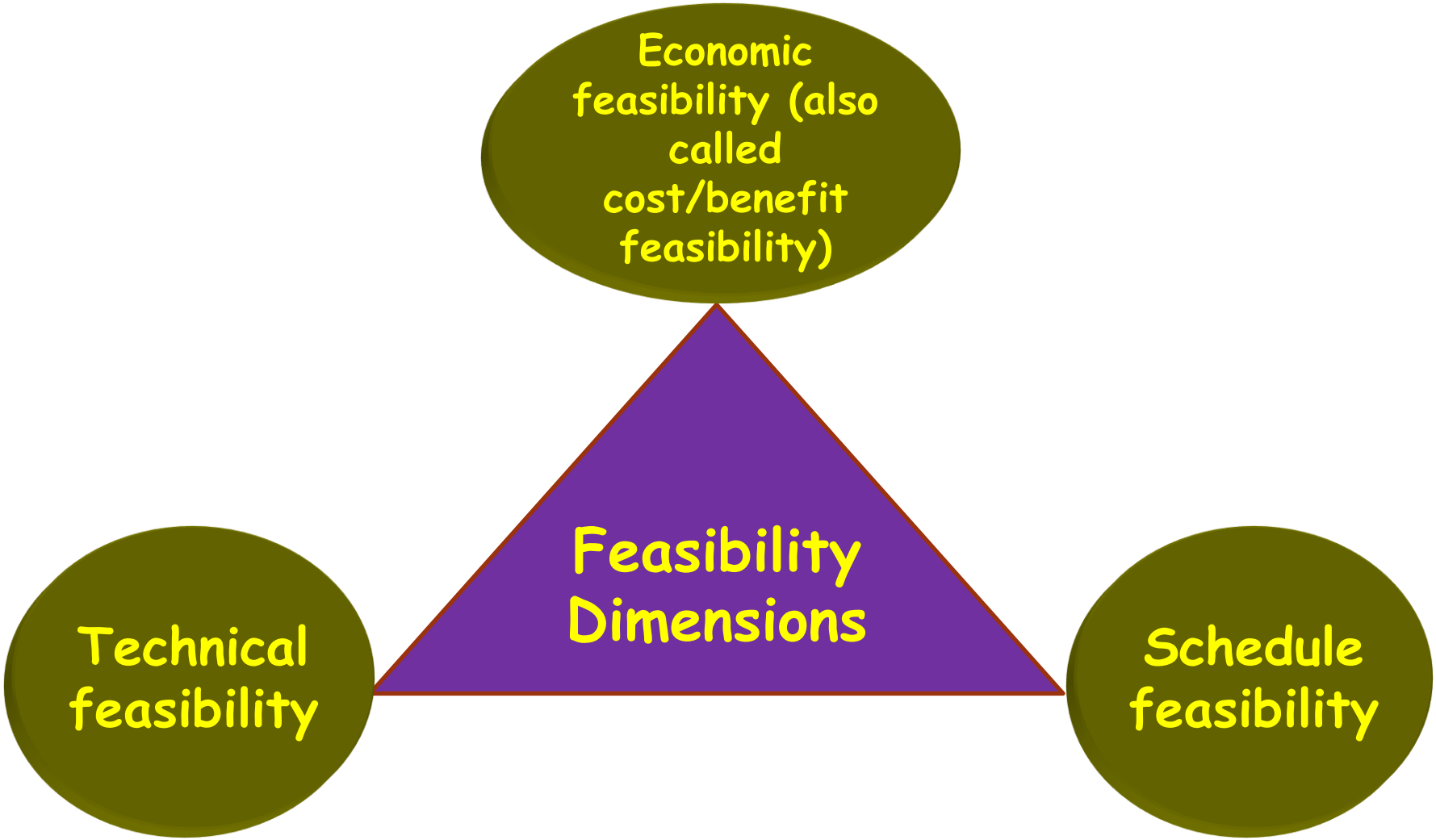
2. System development requirements

- **Requirements analysis**
  - Determining the needs or conditions to meet for a new or altered product/project.

- Conceptually, requirements analysis includes three types of activities:
  1. **Eliciting requirements:** This is sometimes also called requirements gathering or requirements discovery.
  2. **Analyzing requirements:** determining whether the stated requirements are clear, complete, consistent and unambiguous, and resolving any apparent conflicts.
  3. **Recording requirements (**Software Requirements Specification**):** Requirements may be documented in various forms, usually including a summary list and may include natural-language documents, use cases, user stories, process specifications and a variety of models including data models.

3. Feasibility Study

- **Feasibility Study** is an assessment of the practicality of a proposed project or system.



Areas of Feasibility study

A. Technical feasibility: determine whether the company has the technical expertise to handle completion of the project.

- **Method of production**
  - Availability of inputs or raw materials and their quality and prices.
  - Availability of markets for outputs
  - Various efficiency factors such as the expected increase in one of the additional production unit.
- **Production technique**
  - Tools and equipment needed for the project
  - Construction requirement such as buildings, storage, and roads …etc.
  - Requirements of skilled and unskilled labor and managerial and financial labor.
- **Project location**
  - Availability of land (proper acreage and reasonable costs).
  - The costs of transporting inputs and outputs to the project's location.
  - Availability of various related resources: water or electricity or good roads ...etc.

Areas of Feasibility study

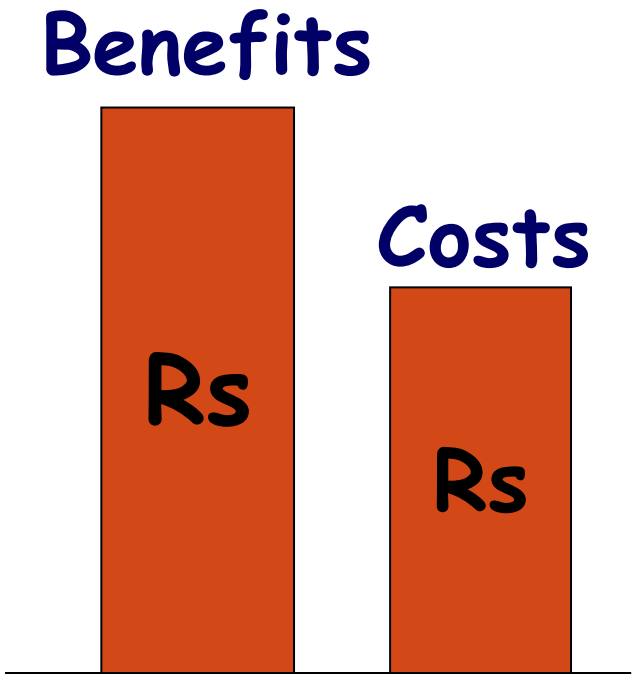
B. Financial feasibility

- In case of a new project, financial viability can be judged on the following parameters:
  - Total estimated cost of the project
  - Financing of the project in terms of its capital structure
  - Existing investment by the promoter in any other business
  - Projected cash flow and
  - **Profitability**

Cost benefit analysis (CBA)

- Need to identify all costs --- these could be:
  - **Development costs**
  - **Set-up**
  - **Operational costs**
- Identify the value of benefits
  - **Quantifiable**
  - **Non-quantifiable**
- Needs to take account of business risks
  - **Identify risks.**
  - **Explain how these will be managed.**

Check benefits are greater than costs



Areas of Feasibility study

C. Schedule feasibility

- Feasibility is a measure of how reasonable the project timetable is?
- Given our technical expertise, are the project deadlines reasonable?
- When it can be built?
- A project will fail if it takes too long to be completed before it is useful.
- It is necessary to determine whether the deadlines are mandatory or desirable.

Areas of Feasibility study

D. Operational feasibility

- Operational feasibility is the measure of
  - how well a proposed system solves the problems?
  - how it satisfies the requirements?
- Desired operational outcomes: reliability, maintainability, supportability, usability, producibility, disposability, sustainability, affordability and others.

E. Legal feasibility

- Determines whether the proposed system conflicts with legal requirements, e.g.,
  - if the proposed venture is acceptable in accordance to the laws of the land.
  - The impact of project on the environment and the approval of the concerned authority.

4. System Design

- Software design is a technical description about **how the system will implement the requirements**

- During design phase requirements specification is transformed into :
  - A form suitable for implementation in some programming language.

- The system architecture describes:
  - How the system will be decomposed into subsystems (modules)
  - Responsibilities of each module
  - Interaction between modules
  - Platforms and technologies

Ref: https://en.wikipedia.org/wiki/Systems_design

System Design

A. Logical design

- Abstract representation of the data flows, inputs and outputs of the system.
- This is often conducted via **modelling**
- Logical design includes entity-relationship diagrams(ER diagrams).

B. Physical design

- In physical design, the input, output, storage, processing, and recovery requirements about the system are decided.
  - User Interface Design
  - Data Design: concerned with how the data is represented and stored within the system.
  - Process Design: concerned with how data moves through the system (DFD)

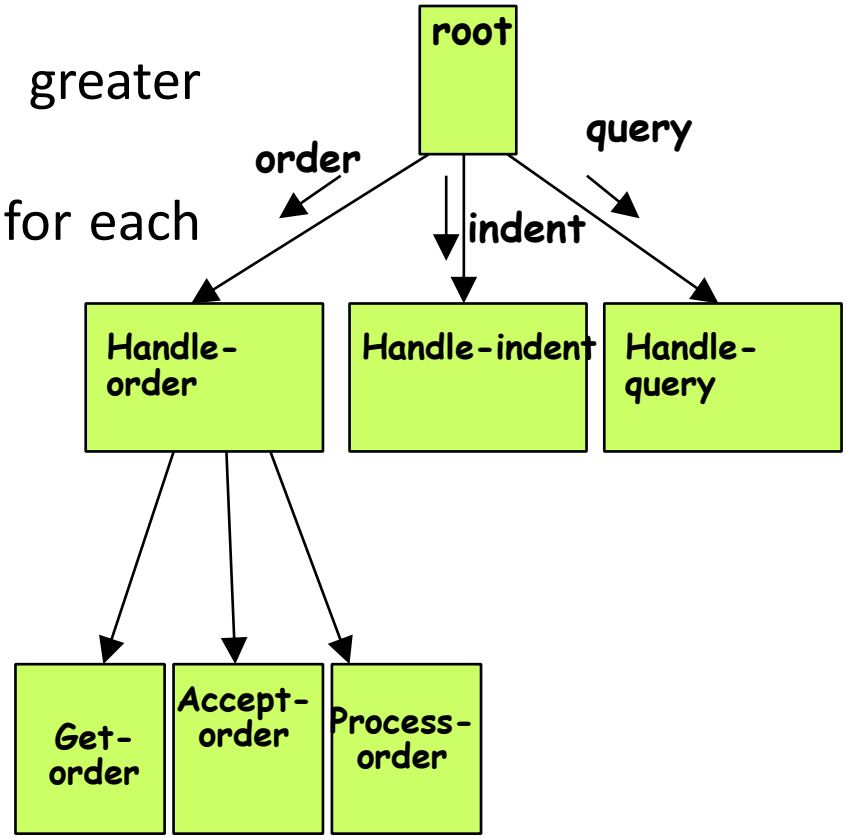
C. Architectural design

- Design of the system architecture that describes the structure, behavior and more views of that system and analysis.

Structured Design

- **High-level design:**
  - Decompose the system into modules,
  - Represent invocation relationships among modules.

- **Detailed design:**
  - Different modules designed in greater detail:
    - Data structures and algorithms for each module are designed.



Structured Design

- Two commonly used design approaches:
  - Traditional approach,
  - Object oriented approach

- Consists of two activities:
  - **Structured analysis:** typically carried out by using DFD, Data Dictionary, State Transition diagram and ER diagram.
  - **Structured design**: It uses Structure Chart

Object-Oriented Design

- First identify various objects (real world entities) occurring in the problem:
  - Identify the relationships among the objects.
  - For example, the objects in a pay-roll software may be:
    - employees,
    - managers,
    - pay-roll register,
    - Departments, etc.

5. Coding and Integration

- Coding is the process of writing the programming code (the source code)

- Inexperienced developers consider coding the core of development
  - In most projects, coding is only 20% of the project activities!
  - The important decisions are taken during the requirements analysis and design
  - Documentation, testing, integration, maintenance, etc. are often disparaged
- Software engineering is not just coding!

Programmer != software engineer

Quantifying program quality

- RUC (Readability, Understandability, and Comprehensibility)
- Logical structure : logical designing of modules
- Physical Layout: listing of the source code
- Robustness: how well the program can handle incorrect data
- CPU efficiency (execution time)
- Memory efficiency (Space consumption)
- Algorithmic and structural complexity
- Human Factors (Human-to-computer interface)
- System interface (system-to-environment interface)
- Reusable code

Integration

- **System integration** process of bringing together the component sub-systems into one system and ensuring that the subsystems function together as a system

6. Testing and debugging

- Testing:
  - A systematic attempt to find faults in planned way in the implemented software
  - A fault detection technique to create failures or erroneous states in a planned way

- Terminologies:
  - **Component**: part of system that can be isolated for testing
  - **Fault**: bug or defect, is a design or coding mistake that may cause abnormal component behavior
  - **Erroneous state**: manifestation of a fault during the execution of the system. Caused by one or more fault and lead to failure
  - **Failure**: deviation between the specification and the actual behavior
  - **Test case**: set of inputs and expected results

Ref: Object-Oriented Software Engineering Using UML, Patterns, and Java, Bernd Bruegge, Allen H. Dutoit 3rd ed. Prentice Hall Press

Software Testing Activities

- **Component Inspection: The goal of the inspection is to identify defects**. In an inspection, a source code is selected for review and a team is gathered for an inspection meeting to review the work product. (static verification)

- **Usability testing:** a small set of target end-users, of a software system, "use" it to expose usability defects. a **non-functional testing** technique that is a measure of how easily the system can be used by end users.

- **Unit testing:** Unit testing is the process of testing individual components in isolation. It is a defect testing process.

- **Integration testing:** individual software modules are integrated logically and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units.

- **System testing:** system testing is a level of software testing where a complete and integrated software is tested.

Testing methods

- **Static vs. dynamic testing**
  - **Static**: Reviews, walkthroughs, or inspections are referred to as static testing, when programming tools/text editors check source code structure or compilers (pre-compilers) check syntax and data flow.
  - **Dynamic:** executing programmed code with a given set of test cases is referred to as dynamic testing. Dynamic testing takes place when the program itself is run.

- **The "box" approach:** describe the point of view that the tester takes when designing test cases
  - **White box testing**: verifies the internal structures or workings of a program.
  - **Black box testing**: examining functionality without any knowledge of internal implementation, without seeing the source code. The testers are only aware of what the software is supposed to do, not how it does it
  - **Grey box testing**: design test cases based on knowledge of internal data structures and algorithms while executing those tests at the user, or black-box level.

https://en.wikipedia.org/wiki/Software_testing#Unit_testing

Unit testing

- The objective of Unit Testing is to isolate a section of code and verify its correctness.
- Motivations:
  - Reduces the complexity of overall test activities. Allowing us to focus on smaller units
  - Makes easier to pinpoint and correct faults
  - Allows parallelism in testing
- **Equivalence testing:** this blackbox testing minimizes the number of test cases. The possible inputs are classified into equivalence classes and a test case is selected for each class.
- **Boundary testing:** Special case of equivalence where boundary of the equivalence classes are tested. (0, empty string, year 2000 or Y2K bug)
- **Path testing:** this white box testing technique identifies faults in the implementation of the component by exercising all possible paths through the code at least once.
- **State-based testing:** recently developed for objected oriented systems. Compares the resulting state of the system with the expected state

Integration testing

- Integration testing is any type of software testing that seeks to verify the interfaces between components against a software design.
  - Big bang testing: assumes all components are tested individually and then tested together as a single system.
  - Bottom-up testing: first test each component of the bottom layer individually and then integrate them with components with next layer up.
  - Top-down testing: Unit test the component of the top layer and then integrate the components of the next layer down.

System testing

Ensures that the complete system compiles with functional and non-functional requirements

- **Functional testing**: find difference between the functional requirement and the system. (Black box testing)
- **Performance testing:** check the design goals
  - Stress testing how system respond to multiple requests
  - Volume testing: fault associated with large amount of data
  - Security testing
  - Timing testing
  - Recovery testing
- **Pilot testing** (field test): system is deployed and used by selected users
  - Alpha test: in development environment
  - Beta test: in the target environment
- **Acceptance testing**: performed by the end user
- **Installation testing**: after installation in target environment
- System testing is done by QA engineers
  - Unit testing is done by developers

Debugging

- Debugging aims to find the source of already identified defect and to fix it
  - Performed by developers

- Steps in debugging:
  - Attempt to reproduce the problem.
  - After the bug is reproduced, the input of the program may need to be simplified to make it easier to debug.
  - After the test case is sufficiently simplified, a programmer can use a debugger tool to examine program states (values of variables, plus the call stack) and track down the origin of the problem(s).
  - Fix the defect
  - Test to check if the fix is correct

7. Implementation and Maintenance

- **Software implement** is all of the activities that make a software system available for use.

- Deployment activities
  - Release
  - Installation and activation
  - Deactivation
  - Uninstallation
  - Update
  - Version tracking

Ref: https://en.wikipedia.org/wiki/Software_deployment

Software Implementation Challenges

- **Code-reuse** - There are huge issues faced by programmers for compatibility checks and deciding how much code to re-use.

- **Version Management** - Every time a new software is issued to the customer, developers have to maintain version and configuration related documentation. This documentation needs to be highly accurate and available on time.

- **Target-Host** - The software program, which is being developed in the organization, needs to be designed for host machines at the customers end. But at times, it is impossible to design a software that works on the target machines.

https://www.tutorialspoint.com/software_engineering/software_testing_overview.htm

Project maintenance

- Reasons for which modifications are required:
  - **Market Conditions** - Change in policies, such as taxation and new constraints
  - **Client Requirements** - Customer may ask for new features or functions.
  - **Host Modifications** - If any of the hardware or platform of the target host change.
  - **Organization Changes** - If there is any business level change at client end, such as reduction of organization strength, acquiring another company, organization venturing into new business.

Types of maintenance

- **Corrective Maintenance** - to correct or fix problems.
- **Adaptive Maintenance** - to keep the software product up-to date.
- **Perfective Maintenance** - to keep the software usable over long period of time. It includes new features, new user requirements for refining the software and improve its reliability and performance.
- **Preventive Maintenance** - to prevent future problems of the software. It aims to attend problems, which are not significant at this moment but may cause serious issues in future.

Ref: https://www.tutorialspoint.com/software_engineering/software_maintenance_overview.htm

- **End of Chapter**

- A software lifecycle model is a process for developing software.
- **The IEEE 1074 Framework**
- The IEEE 1074 framework is a 1997 IEEE standard for defining software lifecycle models.
- IEEE 1074 doesn't attempt to define a lifecycle model, rather it describes what sub-processes should be included in a lifecycle model.
- These sub-processes are organized into six process groups:
- Lifecycle Modeling
  - Selection of a lifecycle model

Project Management
- Project Initiation
- Project Monitoring and Control
- Software Quality Management

Pre-development
- Concept Exploration
- System Allocation

Development
- Requirements
- Design
- Implementation

Post-Development
- Installation
- Operation and Support
- Maintenance
- Retirement

Integral Processes
- Verification and Validation
- Software Configuration Management
- Documentation Development
- Training

http://www.cs.sjsu.edu/faculty/pearce/modules/lectures/se/index.htm