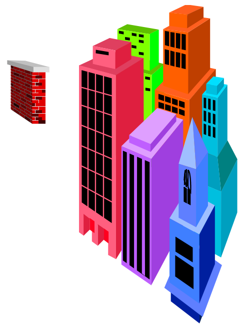
INTRODUCTION

What is Software Engineering?

The action of working artfully to bring something about.

- Engineering approach to develop software.
- Systematic collection of past experience:
  - Techniques
  - Methodologies
  - Guidelines

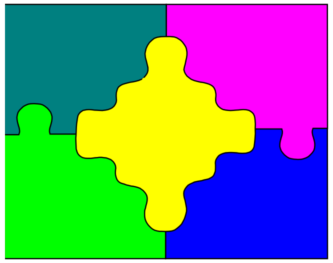


Why Study Software Engineering?

- To acquire skills to develop large programs.
  - Exponential growth in complexity and difficulty level with size.
  - The ad-hoc approach breaks down when size of software increases.

Why Study Software Engineering?

- Ability to solve complex programming problems:
  - How to break large projects into smaller and manageable parts?



- Learn techniques of:
  - specification, design, interface development, testing, project management, etc.

Why Study Software Engineering?

- To acquire skills to be a better programmer:
  - Higher Productivity
  - Better Quality Programs

SOFTWARE DEVELOPMENT LIFECYCLE (SDLC)

- It is a **systematic process** for building software that ensures the **quality and correctness** of the software built.
- It aims to produce **high-quality software** that meets **customer expectations**.

SOFTWARE DEVELOPMENT LIFECYCLE (SDLC)

- Phases of SDLC
  - Feasibility Study
  - Requirements Analysis and Specification
  - Design
  - Implementation
  - Testing
  - Maintenance

Phase 1: Feasibility Study

- Main aim of feasibility study:
  - Determine whether developing the product
  - Financially worthwhile
  - Technically feasible
- First roughly understand what the customer wants:
  - different data which would be **input** to the system
  - **processing** needed on data
  - **output data** to be produced by the system
  - **various constraints** on the behaviour of the system

Activities during Feasibility Study

- Work out an **overall understanding** of the problem.
- Formulate different **solution strategies**.
- Examine **alternate solution strategies** in terms of:
  - Resources required
  - Cost of development
  - Development time
- **Perform a cost/benefit analysis:**
  - To determine which solution is the **best**.

Phase 2: Requirements Analysis and Specification

- <u>Aim of this phase:</u>
  - understand the <u>exact requirements</u> of the customer
  - document them properly.
- Consists of two distinct activities:
  - requirements gathering and analysis
  - requirements specification.

Requirements Gathering

- Gathering relevant data:
  - usually collected from the end-users through interviews and discussions.
  - For example, for a business accounting software:
    - interview all the accountants of the organization to find out their requirements.

Goals of Requirements Analysis

- Collect all related data from the customer:
  - Analyze the collected data to clearly understand what the customer wants
  - Find out any **inconsistencies and incompleteness** in the requirements
  - **Resolve** all inconsistencies and incompleteness.

Requirements Analysis (CONT.)

- Ambiguities and contradictions:
  - must be identified
  - resolved by discussions with the customers.
- Next, requirements are organized:
  - into a Software Requirements Specification (SRS) document.

Requirements Analysis (CONT.)

- Engineers doing requirements analysis and specification:
  - are designated as <u>**analysts**</u>.

Phase 3:Design

- Design phase transforms requirements specification:
  - into a form suitable for implementation in some programming language.

Design

- In technical terms:
  - during design phase, software architecture is derived from the SRS document.
- Two design approaches:
  - traditional approach
  - object oriented approach

Traditional Design Approach

- Consists of two activities:
  - Structured analysis
  - Structured design

Structured Analysis Activity

- Identify all the **functions** to be performed.
- Identify **data flow among the functions**.
- Decompose each function recursively into sub-functions.
  - Identify **data flow among the subfunctions** as well.

Structured Analysis (CONT.)

- Carried out using Data flow diagrams (DFDs).
- After structured analysis, carry out structured design:
  - architectural design (or high-level design)
  - detailed design (or low-level design).

Structured Design

- High-level design:
  - decompose the system into *modules*,
  - represent relationships among the modules.
- Detailed design:
  - different modules designed in greater detail:
    - data structures and algorithms for each module are designed.

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.

Phase 4: Implementation

- Purpose of implementation phase (coding and unit testing phase):
  - translate software design into source code.

Implementation

- During the implementation phase:
  - each module of the design is **coded**
  - each module is unit **tested**
    - tested independently as a stand alone unit
  - each module is **documented**.

Implementation (CONT.)

- The purpose of unit testing:
  - test if individual modules work correctly.
- The end product of implementation phase:
  - a set of program modules that have been tested individually.

Phase 5: Integration and System Testing

- Different modules are integrated in a planned manner:
  - Normally integration is carried out through a number of steps.
- During each integration step,
  - the partially integrated system is tested.

System Testing

- After all the modules have been successfully integrated and tested:
  - system testing is carried out.
- <u>Goal of system testing:</u>
  - To ensure that the developed system functions according to its requirements as specified in the SRS document.

Phase 6:Maintenance

- Maintenance of any software product:
  - requires much more effort than the effort to develop the product itself.
  - development effort to maintenance effort is typically 40:60.

Maintenance (CONT.)

- **Corrective maintenance:**
  - Correct errors which were not discovered during the product development phases.
- **Perfective maintenance:**
  - Improve implementation of the system
  - Enhance functionalities of the system.
- **Adaptive maintenance:**
  - Port software to a new environment,
    - e.g. to a new computer or to a new operating system.