Software Engineering- CS6L034

Introduction

Acknowledgement: Fundamentals of Software Engineering
(**Prof. Rajib Mall**)

Logistics

- Instructor

  Srinivas Pinisetty,

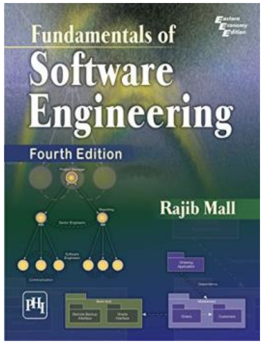
  School of Electrical Sciences,

  Room No: A- 205

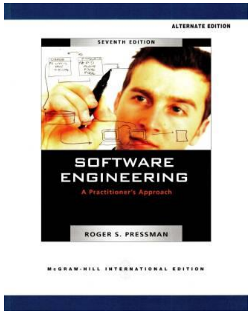
  E-mail: spinisetty@iitbbs.ac.in

Logistics: Contd.

- Reference books
  - Rajib Mall, “Fundamentals of software engineering”, 4th Edition.



  - Roger S. Pressman, “Software Engineering: A Practitioner's Approach ”, McGraw Hill Education. 7th Edition



Logistics: Contd.

- **Grading**:
  - Mid Semester : **25 to 30%**
  - End Semester : **45 to 50%**
  - Surprise Quizzes/ Assignments + Class Participation: **20 to 30%**

  Exact break up will be communicated !!

- **Attendance**

Logistics: Contd.

Join Google Classroom:

bykp3hk

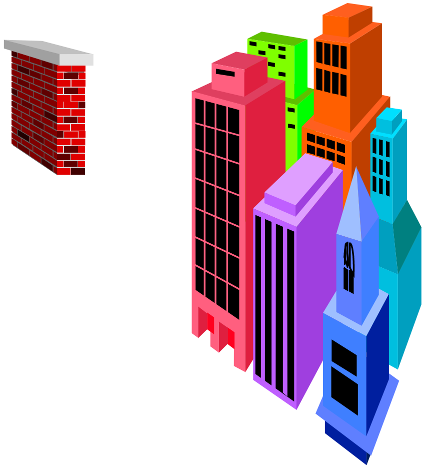
- Share lecture material
- Assignment questions/submissions
- Other course related announcements

Overview

- What is Software Engineering?
- Programs vs. Software Products
- Evolution of Software Engineering
- **Course Contents**

What is Software Engineering?

- Engineering approach to develop software.
  - Building Construction Analogy.



- Systematic collection of past experience:
  - Techniques,
  - Methodologies,
  - Guidelines.

Without using software engineering principles it would be difficult to develop a **large software product**

Software Engineering Principles

- Without using software engineering principles it would be difficult to develop large software products

- Software engineering principles use **two important techniques to reduce problem complexity**:
  - abstraction
  - decomposition

Abstraction

- **Simplify a problem by omitting unnecessary details**.
  - Consider aspects of the problem that are relevant for certain purpose (suppress other aspects)

  - Omitted details taken into account after the simpler problem is solved etc.

Decomposition

- **Decomposition** of a large problem into a set of smaller problems
- Complex problem divided into several smaller problems
- Smaller problems are solved one by one

- Problem to be decomposed such that
  - each component of the decomposed problem can be **solved independently**
  - solution of the different components can be **combined to get the full solution**

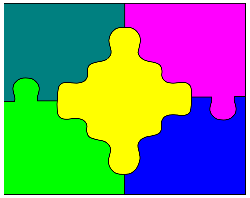
- ***Good decomposition*** should **minimize interaction among different components**

Why Study Software Engineering? (1)

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

Why Study Software Engineering? (2)

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



- Also learn techniques of:
  - Specification, design, testing, verification, project management, etc.

Software Crisis

- **Software products**:
  - Fail to meet user requirements.
  - Frequently crash.
  - Expensive.
  - Difficult to alter, debug, and enhance.
  - Often delivered late.
  - Use resources non-optimally.

Factors Contributing to the Software Crisis

- Larger problems,
- Lack of adequate training in software engineering,
- Increasing skill shortage,
- Low productivity

Programs Vs. Software Products

Programs	Software Products
Usually small in size	Large
Author himself is sole user	Large number of users
Single developer	Team of developers
Lacks proper user interface	Well-designed interface
Lacks proper documentation	Well documented & user-manual prepared
Ad hoc development.	Systematic development

Emergence of Software Engineering

- **Early Computer Programming (1950s):**
  - Programs were being written in assembly language.
  - Programs were limited to about a few hundreds of lines of assembly code.

Early Computer Programming (50s)

- Every programmer developed his/her own style of writing programs:
  - According to his/her intuition (exploratory programming).

High-Level Language Programming (Early 60s)

- High-level languages such as FORTRAN, ALGOL, and COBOL were introduced:
  - This reduced software development efforts greatly.

High-Level Language Programming (Early 60s)

- **Software development style was still exploratory.**
  - **Typical program sizes were limited to a few thousands of lines of source code.**

Control Flow-Based Design (late 60s)

- Size and complexity of programs increased further:
  - Exploratory programming style proved to be insufficient.

- Programmers found:
  - Very difficult to write cost-effective and correct programs.

Control Flow-Based Design (late 60s)

- Programmers found:
  - **programs written by others** very difficult to understand and maintain.

- To cope up with this problem, experienced programmers advised:

``Pay particular attention to the design of the **program's control structure**."

Control Flow-Based Design (late 60s)

- A program's control structure indicates:
  - The sequence in which the program's instructions are executed
- To help design programs having good control structure:
  - Flow charting technique was developed.

Control Flow-Based Design (late 60s)

- Using flow charting technique:
  - One can represent and design a program's control structure.

- Usually one understands a program:
  - By mentally simulating the program's execution sequence.

Control Flow-Based Design (Late 60s)

- A program having a messy flow chart representation:
  - Difficult to understand and debug.

- **GOTO** statements makes control structure of a program messy.

Control-flow Based Design (Late 60s)

- Everyone accepted:
  - It is possible to solve any programming problem without using GOTO statements.

  - This formed the basis of Structured Programming methodology.

Structured Programming

- A program is called structured
  - When it uses only the following types of constructs:
    - sequence,
    - selection,
    - iteration

Structured Programs

- Unstructured control flows are avoided.
- Consist of a neat set of modules.

Object-Oriented Design (80s)

- Object-Oriented Techniques have gained wide acceptance:
  - Simplicity
  - Reuse possibilities
  - Lower development time and cost
  - More robust code
  - Easy maintenance

Evolution of Other Software Engineering Techniques

- In addition to the software design techniques:
  - Several other techniques evolved.

Evolution of Other Software Engineering Techniques

- life cycle models,
- specification techniques,
- project management techniques,
- testing techniques,
- debugging techniques,
- quality assurance techniques,
- software measurement techniques,
- CASE tools, etc.

Differences between the exploratory style and modern software development practices

- Use of Life Cycle Models

- Software is developed through several **well-defined stages**:
  - requirements analysis and specification,
  - design,
  - coding,
  - testing, etc.

Differences between the exploratory style and modern software development practices

- Emphasis has shifted
  - from error correction to error prevention.

- Modern practices emphasize:
  - detection of errors as close to their point of introduction as possible.

Differences between the exploratory style and modern software development practices (CONT.)

- In exploratory style,
  - errors are detected only during testing,

- Now,
  - focus is on detecting as many errors as possible in each phase of development.

Differences between the exploratory style and modern software development practices (CONT.)

- In exploratory style,
  - coding is synonymous with program development.
- Now,
  - coding is considered only a small part of program development effort.

Differences between the exploratory style and modern software development practices (CONT.)

- A lot of effort and attention is now being paid to:
  - Requirements specification.
- Also, now there is a distinct design phase:
  - Standard design techniques are being used.

Differences between the exploratory style and modern software development practices (CONT.)

- During all stages of development process:
  - **Periodic reviews** are being carried out

- Software testing has become systematic:
  - **Standard testing techniques** are available.

Differences between the exploratory style and modern software development practices (CONT.)

- There is better visibility of design and code:
  - **Visibility** means production of good quality, consistent and standard documents.

  - In the past, very little attention was being given to producing good quality and consistent documents.

  - We will see later that increased visibility makes software project management easier.

Differences between the exploratory style and modern software development practices (CONT.)

- Because of good documentation:
  - fault diagnosis and maintenance are smoother now.

- Several metrics are being used:
  - help in software project management, quality assurance, etc.

Differences between the exploratory style and modern software development practices (CONT.)

- Projects are being thoroughly planned:
  - estimation,
  - scheduling,
  - monitoring mechanisms.
- Use of CASE tools.

Software Life Cycle

- Software life cycle (or software process):
  - Series of identifiable stages that a software product undergoes during its life time:
    - Feasibility study
    - Requirements analysis and specification,
    - Design,
    - Coding,
    - Testing
    - maintenance.

Life Cycle Model

- A software life cycle model (or process model):
  - a descriptive and diagrammatic model of software life cycle:
  - identifies all the activities required for product development,
  - establishes a precedence ordering among the different activities,
  - Divides life cycle into phases.
    - Phase entry & exit criteria

Life Cycle Model (CONT.)

- Several different activities may be carried out in each life cycle phase.
  - For example, the design stage might consist of:
    - structured analysis activity followed by
    - structured design activity.

Why Model Life Cycle?

- A written description:
  - Forms a common understanding of activities among the software developers.
  - Helps in identifying inconsistencies, redundancies, and omissions in the development process.
  - Helps in tailoring a process model for specific projects.

Why Model Life Cycle ?

- Processes are tailored for special projects.
  - A documented process model
    - Helps to identify where the tailoring is to occur.

Life Cycle Model (CONT.)

- The development team must identify a suitable life cycle model:
  - and then adhere to it.
  - Primary advantage of adhering to a life cycle model:
    - Helps development of software in a systematic and disciplined manner.

Life Cycle Model (CONT.)

- When a program is developed by a single programmer ---
  - he has the freedom to decide his exact steps.

Life Cycle Model (CONT.)

- When a software product is being **developed by a team**:
  - there must be a precise understanding among team members as to when to do what,
  - otherwise it would lead to chaos and project failure.

Life Cycle Model (CONT.)

- A software project will never succeed if:
  - one engineer starts writing code,
  - another concentrates on writing the test document first,
  - yet another engineer first defines the file structure
  - another defines the I/O for his portion first.

Life Cycle Model (CONT.)

- A life cycle model:
  - defines ***entry*** and ***exit*** criteria for **every phase**.
  - A phase is considered to be ***complete***:
    - only when all its exit criteria are satisfied.

Life Cycle Model (CONT.)

- The phase exit criteria for the software requirements specification phase:
  - Software Requirements Specification (SRS) document is complete, reviewed, and approved by the customer.
- A phase can start:
  - only if its phase-entry criteria have been satisfied.

Life Cycle Model (CONT.)

- It becomes easier for software project managers:
  - to monitor the progress of the project.

Life Cycle Model (CONT.)

- When a life cycle model is adhered to,
  - the project manager can at any time fairly accurately tell,
    - at which stage (e.g., design, code, test, etc.) of the project is.
  - Otherwise, it becomes very difficult to track the progress of the project
    - the project manager would have to depend on the guesses of the team members.

Life Cycle Model (CONT.)

- This usually leads to a problem:
  - known as the 99% complete syndrome.

Life Cycle Model (CONT.)

- Many life cycle models have been proposed.
- We will confine our attention to a few important and commonly used models.
  - Classical waterfall model
  - Iterative waterfall,
  - Evolutionary,
  - Prototyping, and
  - Spiral model

Summary

- Software engineering is:
  - Systematic collection of decades of programming experience
  - Together with the innovations made by researchers.

Summary

- A fundamental necessity while developing any large software product:
  - Adoption of a life cycle model.

Summary

- Adherence to a software life cycle model:
  - Helps to do various development activities in a systematic and disciplined manner.
  - Also makes it easier to manage a software development effort.

Course Contents

Course Contents- Module 1

S.No	Topic
1.	Introduction to software engineering
2.	Software life cycle models
3.	Agile software development
4.	Requirements analysis and specification
5.	Structured analysis and design

Course Contents- Module 2

S.No	Object oriented software design/development
1.	Object-oriented Analysis and Design
2.	UML views and models
3.	Design patterns (GRASP and GoF Patterns)

Course Contents- Module 3

S.No	Testing
1.	Levels of testing
2.	**Unit testing**: Coverage criteria, black-box/ white-box testing, unit testing tools

Course Contents- Module 4

Introduces practically and theoretically the few important styles of formal methods for reasoning about software

- Modeling and modeling languages
- Specification and specification languages
- In depth analysis of possible system behavior
- Reason about system (mis)behavior

S.No	Formal Methods for Software Development
1.	Formal specification- Importance/significance, LTL
2.	Ideas of model-driven software engineering
3.	Motivation: need for automated formal verification approaches, (e.g. model-checking approach)
4.	Modeling and automated verification/analysis of distributed systems (using Promela/Spin)

Other Topics

- **Coding**: Coding standards and guidelines
- Software project management
- Software Reliability, Software Project Monitoring & Control
- Software Maintenance & Reuse

All the best !!