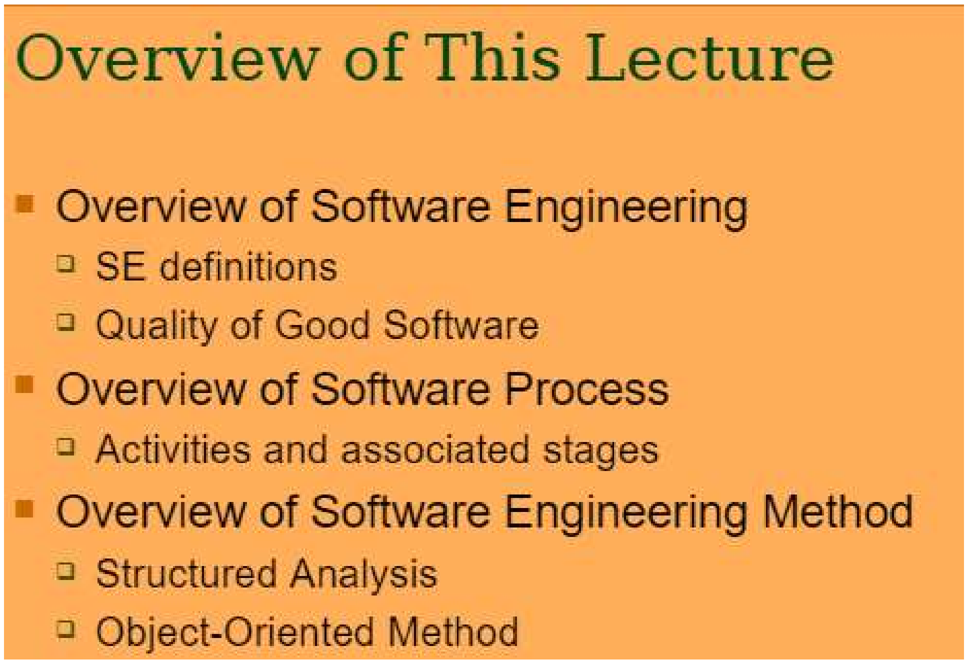
Lecture 1

Wednesday, January 18, 2023 6:00 PM



Overview of This Lecture

- Overview of Software Engineering
  - SE definitions
  - Quality of Good Software
- Overview of Software Process
  - Activities and associated stages
- Overview of Software Engineering Method
  - Structured Analysis
  - Object-Oriented Method

- Overview of Software Engineering
  - What is "Software Engineering"? (slide 5)
    - Large + high quality software, teamwork and coord
  - Software?(slide 6)
    - Collection of pgrms, docs, and config data, 3 main types
      - Generic Product: Stand alone, sold on open market
      - Customized Product: Specific for customer
      - Embedded Product: Built-in hardware
  - Natures of Software:
    - Intangible- hard to understand and dev process
    - Easy to Reproduce- costly design, but cheap manufacturing
    - Malleable- easy to change (even w/out full understanding, 'hack' something)
  - Probs
    - Unbound complexity
    - Difficult understanding & managing complexity causes
      - Known software disasters – 10-12
  - Engineering? - Systemically id, understand, & integrate constraints on design for successful results
  - Quality of Good Software:
    - Usability- easy to learn/use
    - Efficiency- no waste, like CPU time and mem
    - Dependability- reliable, secure, safe
    - Maintainability- easy modifiable, meet changing req
    - Reusability- reusable pts, no mods
- Overview of Software Process
  - Software Process- 'Software Engineering Method'
    - Set of activities and associated results, make software product

            1. Software Specification
                - Define software to be made, and constraints on operation
                - Stages:
                1. Feasibility Study: possible w/ current tech & budget?
                2. Domain Analysis: background for software?
                3. Req Gathering & Analysis: what user wants?
                4. Req Specification: doc on User and Sys req
                5. Req Validation: realism, consistency, and completeness
            2. Software Development
                - Design and pgrm
                - Sys analysts, decide how req implemented
                - Pgrmers, translate into code and languages
                - Stages:
                1. Architectural Design: split into subsys
                2. Abstract Specification: high lvl specs on services & constraints
                3. Interface Design: interact w/ other sys
                4. Component Design: Components in subsys
                5. Data Structure Design: data structure
                6. Algorithm Design: design and specify algorithm for service
                7. Data Structure & Algorithm Design: for pgrmer
            3. Software Validation
                - Check software, make sure meet customer's req
                - Stages:
                1. Component Testing: testing of each part in subsys
                2. Sys Testing: test integrated parts, multilevel
                3. Acceptance Test: test w/ customer supplied data
            4. Software Evolution
                - Define changing req, mod software to adapt
                - Usually sys updates
    - Simple and Complex Software Process (slide 28)
        - Complex- better to design b4 code, on big projects, mid-pt pieces of doc made
        - Complex has 4 steps:
            - Software spec- do req analysis and make req docs
            - Software dev- design(structure chart of functions/modules/classes), & programming (code into src code, debugging)
            - Software Validation- compare against sample outputs
            - Software Evo- applicable
    - Modeling the Sys
        - Structure Chart is ex of sys model in complex software process, easier to understand and manipulate
- Overview of Software Engineering Method
    - Software Engineering Method
        - Strat for success dev software, principle thru software process, no best method
    - Structured Analysis
        - Earliest method, ex is DFD (data flow diagram)
        - Basically: Function Oriented- Id process (function) that transform data
        - Good match for procedural lang, like C, Pascal, Fortran, etc.
    - Structured methods (analysis and design) refer to software sys where data processed by functions outside data, so sys data stored sep, functions stored sep
    - Object-Oriented Methods
        - Basically object oriented: id entity (object) that has collection of both data and process (function) that operate them

    - Most op use small fraction of tot data of sys, most parts of data got by small ops
    - Need to split data repo and integrate parts of data together w/ op that direct change that data, main principle
- Compare Structured and Object-Oriented Methods:
    - Op localized together with data affected, in object oriented, which is easier to understand and maintain
    - Structured, each op can get data from central repo

Assessment 1

Saturday, January 21, 2023 10:03 AM

Q1:

(**Software Engineering Myths)** Explain whether the following software engineering myths hold and why:

1.A general statement of objectives is sufficient to begin writing programs - we can fill in the details later.

A: This myth does not hold because software has unbounded complexity. Though the main objective of the code may be written, additional elements might majorly affect the functioning of the program. Aside from this, badly defined objectives will lead to worse interpretations by the programmers, which will then lead to failure.

2.If we get behind schedule, we can add more programmers and catch up.

Project requirements continually change, but change can be easily accommodated because software is flexible.

A: This myth does not hold because software is already complicated. Adding more programmers to the project would mean that they all would have to understand what the program already contains. This would lead to confusion, and the project getting delayed even more.

Q2:
(Having a meal, [Priestley, 2004], page 13) A description of a methodology for having a meal might include the following steps: plan the menu; do the shopping; cook the meal; eat the meal; wash the dishes. Define a suitable 'deliverable' (artifact) for each step in the process.

A: Artifacts are documentation or by-products made at stages of the development process. These are the steps of having a meal along with suitable artifacts:

1. Plan the menu
   - Document of menu, including ingredients required and process of making
2. Do the shopping
   - Make sure that all ingredients listed are easily accessible (note where all ingredients can be found) and go buy them.
   - All ingredients should be ready to assemble.
3. Cook the meal
   - Follow the recipe
   - Final product should look, smell, and taste the same
4. Eat the meal
   - Full stomach
   - Empty plate/dish
5. Wash the dishes
   - Dishes should be cleaned, dried, and kept away

Q3:
What are the differences between object-oriented approach and traditional approach? Give your own example and explain it.

A: In the traditional approach, operations choose to modify the data specified. In the object-oriented approach, operations affect sets of data that are connected together (they make up an object).
An example of this would be birds. All bird are endothermic, have two feet, wings, feathers, a beak, and lay

eggs. The traditional approach will just say there is a hawk, penguin, and a pigeon instead of saying there are birds.

Q4:

Which of the following statements regarding generic and custom software product development in general is false?

The product specification is owned by the product developer in case of the generic product development, and by the client in case of the custom product development;

The client is the end user in case of the custom product development, and on the contrary in case of the generic product development;

The generic product development involves the client in testing;

None of the above.

Q5:

Which of the following statements regarding software quality is true?

User and customer/client perceive software quality in the same way;

For critical system (e.g., flight control software), reusability is more important than dependability;

Different software qualities always complement each other;

Usability is only important for generic software but not for customized software;