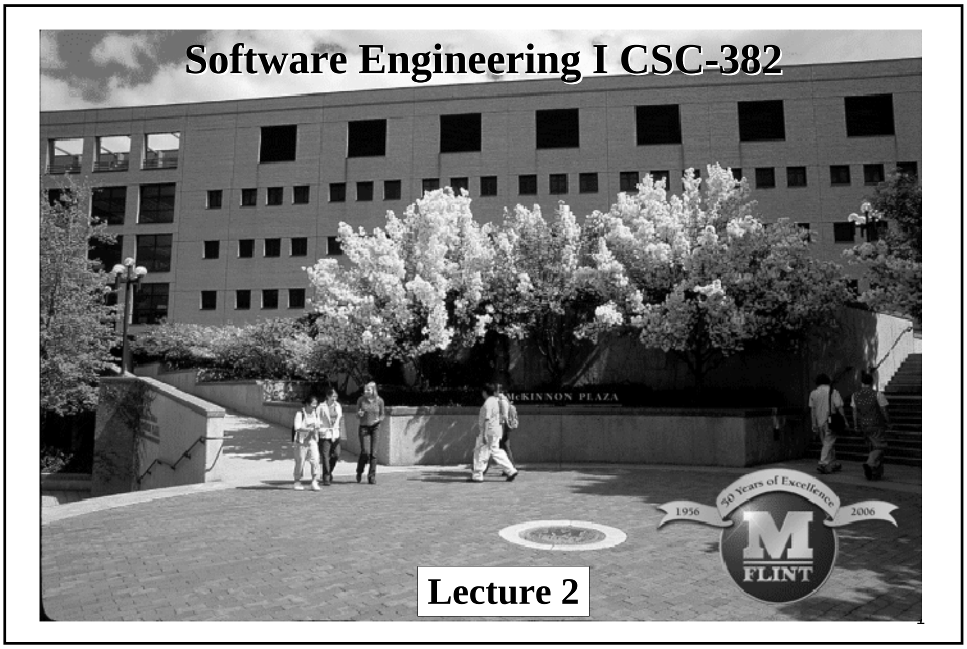
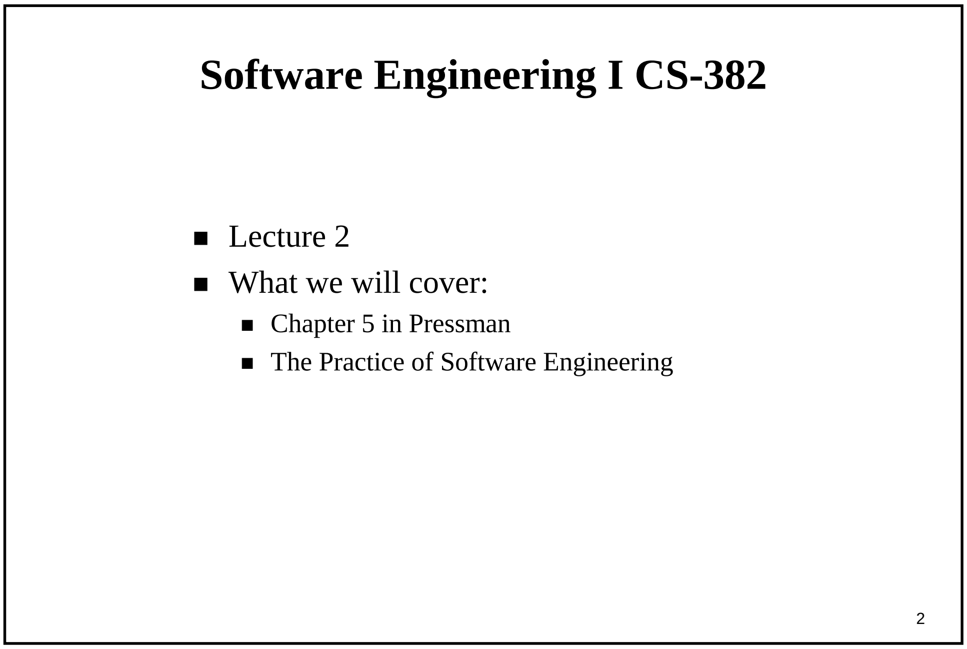


Software Engineering I CSC-382

Lecture 2



Software Engineering I CS-382

- Lecture 2
- What we will cover:
  - Chapter 5 in Pressman
  - The Practice of Software Engineering

Discussion of the Class Project

- Details are posted on the BB
- The deliverables are:
  1. Project proposal
  2. Report
  3. Presentation
- The dates are:

Deliverable	Date
Project Proposal	Jan 30th in class
Project Status Review	March 12h in class.
Project Write-up & Presentation	April 14th or 16th in class.

Discussion of the Class Project - Proposal

- The project will be initiated with a high-level customer need statement
  - Each team will develop this for a software product that they will develop
- This statement should be roughly 1 page in length and describe the following:
  - the customer (who they are)
  - the problem being solved
  - the desired features of the solution
  - the economic benefit of the solution to the customer
  - what makes your team uniquely qualified to deliver the solution to the customer

Discussion of the Class Project - Report

- The report specifically will consist of the following document sections:
  1. Product Scope
  2. Systems Architecture
  3. Software Requirements
     - i. Scenario Models
     - ii. Data/Class Models
     - iii. Behavioral Models
     - iv. Data Flow Models

Discussion of the Class Project – Report II

4. Software Design
   - i. Detailed Class Models
   - ii. Detailed Behavioral Models
   - iii. Text descriptions (like P-SPECS or process narratives) for the various methods employed by the classes in the Class Model
   - iv. Deployment diagram (software onto the hardware)
5. Conclusions

Discussion of the Class Project - Presentation

- Presentation will be 20-30 minutes
  - Depends on the number of groups we have
- Presentation will be made on electronic media
  - Preferably PowerPoint
- Contents:
  - Brief narrative description of system
  - System hierarchy and context diagram
  - Software context diagram and top level Use Case diagram
  - …
  - Representative diagrams and text descriptions at various levels of abstraction will be presented

What is “Practice”?

- Practice is a broad array of concepts, principles, methods, and tools that you must consider as software is planned, developed, and delivered.
- It represents steps required to derive the details
  - The technical considerations and how to’s that are below the surface of the software process
  - The things that you’ll need to actually build high-quality computer software.

The Essence of the Engineering Practice

- George Polya, in a book written in 1945 (!), describes the essence of an engineering practice …
  - ***Understand the problem*** (communication and analysis).
  - ***Plan a solution*** (modeling and software design).
  - ***Carry out the plan*** (code generation).
  - ***Examine the result for accuracy*** (testing).

- At its core, good practice is ***common-sense problem solving***

Questions to Ask Based on These Steps

- Understanding the Problem
  - Who are the stakeholders?
  - What are the unknowns?
  - Can the problem be subdivided?
  - Can the problem be represented graphically?

Questions to Ask Based on These Steps II

- Plan the Solution
  - Have you seen similar problems before?
  - Has a similar problem been solved by someone else?
  - Can solutions for sub-problems be easily identified?
  - Can you represent the solution in a manner that eases implementation?

Questions to Ask Based on These Steps III

- Carry out the Plan
  - Does the solution match what was planned?
  - Is each component of the solution correct?

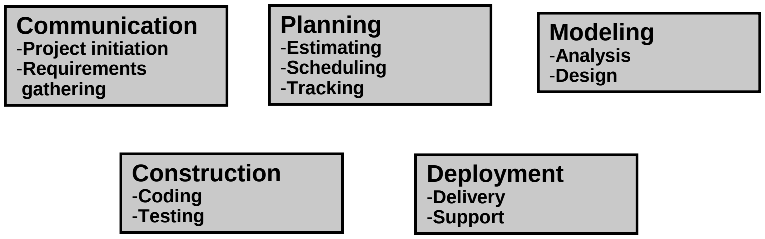
- Examine the Result
  - Is it possible to test each component of the solution?
  - Does the solution produce results that match what was required?

Core Software Engineering Principles

1) Provide value to the customer and the user
2) KIS—keep it simple!
3) Maintain the product and project “vision”
4) What you produce, others will consume
5) Be open to the future
6) Plan ahead for reuse
7) Think!

Software Engineering Practices

- Recall the generic process framework elements:



Communication
-Project initiation
-Requirements gathering

Planning
-Estimating
-Scheduling
-Tracking

Modeling
-Analysis
-Design

Construction
-Coding
-Testing

Deployment
-Delivery
-Support

- Here, we’ll identify
  - Underlying principles for each
  - A typical task set for each element

Communication Practices - Principles

1) Listen
2) Prepare before you communicate
3) Facilitate the communication
4) Face-to-face is best
5) Take notes and document decisions

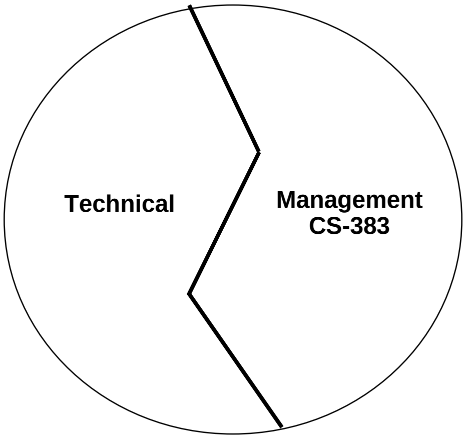
Communication Practices – Principles II

6) Collaborate with the customer
7) Stay focused
8) Draw pictures when things are unclear
9) Know when to move on in discussions
10) Negotiation works best when both parties win.

Communication – Generic Task Set

1. Identify primary customer
2. Meet with and discuss business/end-user issues
3. Develop project scope statement
4. Review and modify project scope statement
   - ■ Sometimes called Operational Concept Description (OCD)
5. Get more detailed:
   - ■ Define customer usage scenarios, system I/O, major functions/features
6. Document data from #5
7. Iterate upon data from #5
8. Prioritize data from #5
9. Review the results with all stakeholders

Planning Practices



Planning Practices - Principles

1) Understand the project scope
2) Involve the customer (and other stakeholders)
3) Recognize that planning is iterative
4) Estimate based on what you know
5) Consider risk

Planning Practices – Principles II

6) Be realistic
7) Adjust granularity as you plan
8) Define how quality will be achieved
9) Define how you’ll accommodate changes
10) Track what you’ve planned

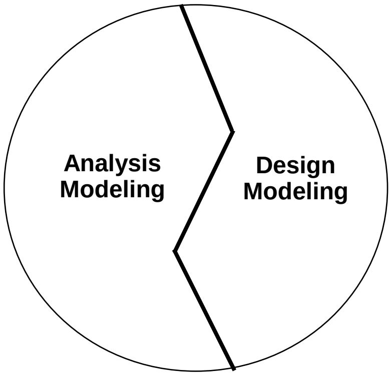
Planning – Generic Task Set

1. Re-assess project scope
2. *Assess risks*
3. Develop/Modify Usage Scenarios
4. Derive functions/features
5. Consider infrastructure functions/features
6. Prioritize features
7. *Create a coarse granularity plan*
8. *Create fine granularity plan for current increment*
9. *Track progress*

***** *CS-383 topics*

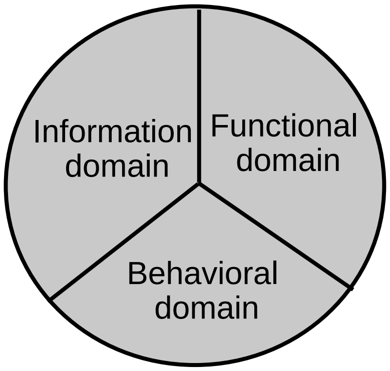
Modeling Practices

- We create models to gain a better understanding of the actual entity to be built:



Analysis Modeling

- ***Analysis models*** represent the customer requirements by depicting the software in three different domains:



Analysis Modeling Practices - Principles

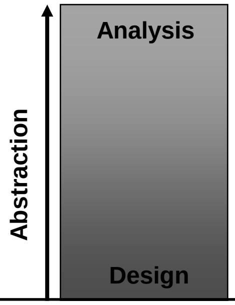
1) The **information** domain must be represented and understood.
2) The **functions** that the software performs must be defined.
3) The **behavior** of the software as a result of external events must be represented.
4) The models that define #1-#3 must be partitioned to uncover detail in a hierarchical manner.
5) The analysis task moves from essence toward implementation.

Analysis Modeling – Generic Task Set

1. Review requirements from planning stage
2. Expand/refine usage scenarios
3. Model the **information** domain
4. Model the **functional** domain
5. Model the **behavioral** domain
6. Analyze and model the user interface
7. Review all models for completeness, consistency, and **correctness** (missed in book – most subtle and critical errors. E.g “That’s nice but I wanted a Bud Light”)

Design Modeling

- ***Design models*** represent characteristics of the software that help practitioners to construct it effectively
  - It refines the models begun in the analysis to the point where the software can be constructed



- Elements of the **design** model
  - Data/Class design
  - Architectural design
  - Interface design
  - Component design

Design Modeling Practices - Principles

1) Design must be traceable to the analysis model

2) Always consider architecture

3) Design of data as critical as design of functionality

4) Interfaces (external and internal) must be designed

5) User interface should be designed towards **end-**user

Design Modeling Practices – Principles II

6) Components should exhibit functional independence

7) Components should be loosely coupled

8) Design representation should be easily understood

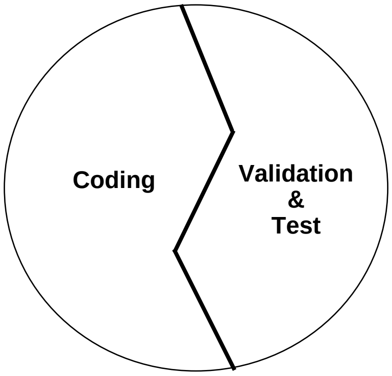
9) The design model should be developed iteratively

Design Modeling – Generic Task Set

1. Select an architectural style
2. Partition the analysis model into subsystems and allocate across the architecture
3. Design the user interface
4. Conduct component-level design
5. Develop a deployment model

Construction Practices

- Two tasks associated with the software construction:



Coding Practices – Preparation

- ***Before** you write one line of code*, *be sure you:*
  1. Understand of the problem you're trying to solve (see communication and modeling)
  2. Understand basic design principles and concepts.
  3. Pick a programming language that meets the needs of the software to be built and the environment in which it will operate.
  4. Select a programming environment that provides tools that will make your work easier.
  5. Create a set of unit tests that will be applied once the component you code is completed**.**

Coding Practices – Principles

1. Constrain your algorithms by following structured programming [BOH00] practice.
2. Select data structures that will meet the needs of the design.
3. Understand the software architecture and create interfaces that are consistent with it.
4. Keep conditional logic as simple as possible.

Coding Practices – Principles II

5. Create nested loops in a way that makes them easily testable.
6. Select meaningful variable names and follow other local coding standards.
7. Write code that is self-documenting.
8. Create a visual layout (e.g., indentation and blank lines) that aids understanding.

Validation & Testing Practices – Principles

0. After you’ve completed your first coding pass:
   - Conduct a code walkthrough (can remove many errors early)
   - Perform unit tests and correct errors you’ve uncovered.
   - Re-factor the code.
1. All tests should be traceable to requirements.
2. Tests should be planned.

Validation & Testing Practices – Principles II

3. The Pareto Principle applies to testing (80/20 rule).
    1. Don’t be afraid to re-design & re-write the problem areas.
4. Testing begins “in the small” and moves toward “in the large”.
5. Exhaustive testing is not possible.

Coding – Generic Task Set

1. Build the architectural infrastructure
    - Code the overall system top-level structure that will be accepting the components
2. Build the software components that reside within this architecture
3. Unit test the components as they are completed
4. Integrate tested components into the architectural infrastructure

Validation & Test – Generic Task Set

1. Design unit tests for each component
2. Develop an integration strategy
   - E.g. define builds and possibly incremental releases to customer for early acceptance testing
3. Develop a validation strategy
4. Conduct integration and validation tests
5. Conduct high-level tests
6. Coordinate acceptance tests with customer
   - Critical for managing customer expectations

Deployment Practices - Principles

1. Manage customer expectations for each increment
2. A complete delivery package should be assembled and tested
3. A support regime should be established
4. Instructional materials must be provided to end-users
5. Buggy software should be fixed first, delivered later

Deployment – Generic Task Set

1. Create the delivery media
   - This contains all deliverables including documentation and help files
2. Establish the support infrastructure
3. Establish user feedback mechanisms
4. Disseminate delivery media to all users
5. Conduct ongoing support
6. Collect and review user feedback

For Next Class

- Read Chapter 6 (Systems Engineering)
  - 6.1
  - 6.2 (skip the subsections 6.2.1 and 6.2.2)
  - 6.3
  - 6.4
- Start Chapter 7 (Requirements Engineering)