Chapter 5
Practice: A Generic View

Moonzoo Kim
CS Division of EECS Dept.
KAIST
moonzoo@cs.kaist.ac.kr
http://pswlab.kaist.ac.kr/courses/cs350-07

What is “Practice”?

- Practice is a broad array of concepts, principles, methods, and tools that you must consider as software is planned and developed.
- It represents 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 Practice

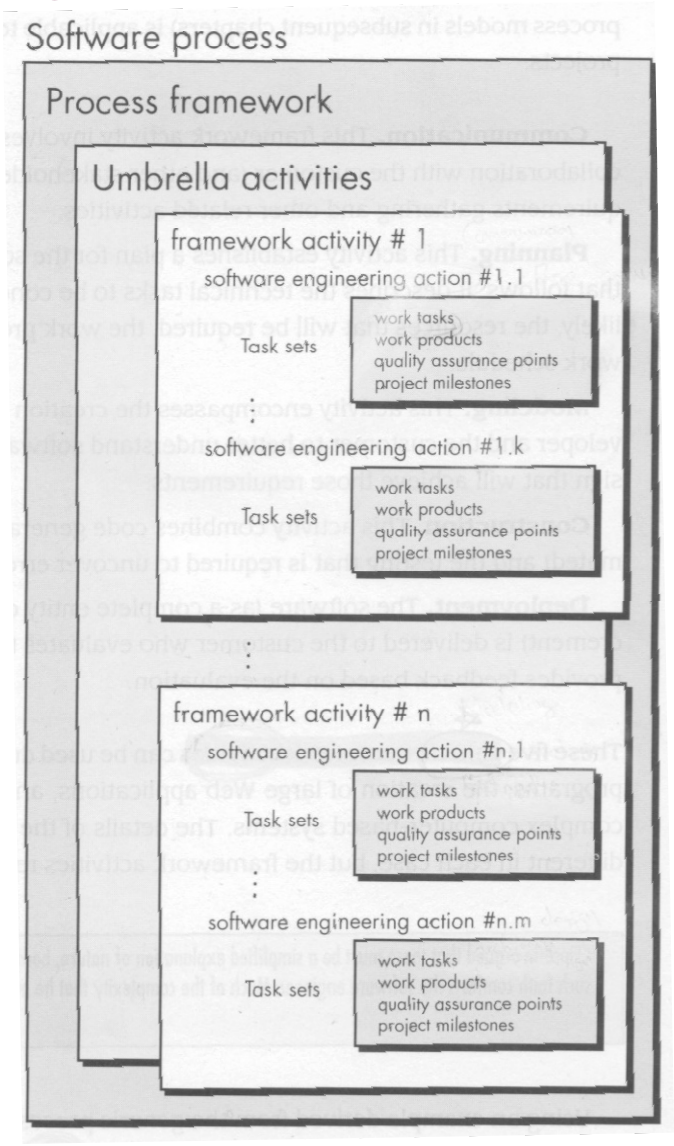
- George Polya, in a book written in 1945 (!), describes the essence of software engineering practice …
  - *Understand the problem* (communication and analysis).
  - *Plan a solution* (modeling and software design).
  - *Carry out the plan* (code generation).
  - *Examine the result for accuracy* (testing and quality assurance).
- At its core, good practice is common-sense problem solving

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/use/maintain
5. Be open to the future
6. Plan ahead for reuse
7. Think before action

Software Engineering Practices

- Consider the generic process framework
  - Communication
  - Planning
  - Modeling
  - Construction
  - Deployment
- Here, we will identify
  - Underlying principles
  - How to initiate the practice
  - An abbreviated task set



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
  6. Collaborate with the customer
  7. Stay focused
  8. Draw pictures when things are unclear
  9. Move on anyway
  10. Negotiation works best when both parties win.

Communication Practices

- Initiation
  - The parties should be physically close to one another
  - Make sure communication is interactive
- An abbreviated task set
  - Identify who it is you need to speak with
  - Define the best mechanism for communication
  - Establish overall goals and objectives and define the scope
  - Get more detailed
    - Have stakeholders define scenarios for usage
    - Extract major functions/features
  - Review the results with all stakeholders

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
  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 Practices

- Initiation
  - Ask Boehm's questions (W^5HH questions)
    - Why is the system begin developed? (business reason)
    - What will be done? (functionality)
    - When will it be accomplished? (timeline)
    - Who is responsible? (work assignment)
    - Where are they located (organizationally)?
    - How will the job be done technically and managerially?
    - How much of each resource is needed?

Planning Practices

- An abbreviated task set
  - Re-assess project scope
  - Assess risks
  - Evaluate functions/features
  - Consider infrastructure functions/features
  - Create a coarse granularity plan
    - Number of software increments
    - Overall schedule
    - Delivery dates for increments
  - Create fine granularity plan for first increment
  - Track progress

Modeling Practices

- We create models to gain a better understanding of the actual entity to be built
- *Analysis models* represent the customer requirements by depicting the software in three different domains (multi-view points)
  - the information domain
  - the functional domain
  - the behavioral domain.
- *Design models* represent characteristics of the software that help practitioners to construct it effectively:
  - the architecture
  - the user interface
  - component-level detail.

Analysis Modeling Practices

- Analysis modeling principles
  1. Represent the information domain
     - input,output, internal data storage
  2. Represent software functions
     - Various features of SW
  3. Represent software behavior
     - In terms of a consequence of external events
  4. Partition these representations
  5. Move from essence toward implementation
- Elements of the analysis model (Chapter 8)
  - Behavior model (sequence diagram in UML)
  - Class model (class diagram in UML)
  - Data model
  - Flow model

Design Modeling Practices

- Principles
  1. Design must be traceable to the analysis model
  2. Always consider architecture
  3. Focus on the design of data
  4. Interfaces (both user and internal) must be designed
  5. Components should exhibit functional independence
  6. Components should be loosely coupled
  7. Design representation should be easily understood
  8. The design model should be developed iteratively
- Elements of the design model
  - Data design
  - Architectural design
  - Component design
  - Interface design

Construction Practices

- Preparation principles: *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.

Construction Practices

- Coding principles: *As you begin writing code, be sure you:*
  1. Constrain your algorithms by following structured programming 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.
  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.

Construction Practices

- Validation Principles: *After you've completed your first coding pass, be sure you:*
  1. Conduct a code walkthrough when appropriate.
  2. Perform unit tests and correct errors you've uncovered.
  3. Refactor the code.

Construction Practices

- Testing Principles
  1. All tests should be traceable to requirements
  2. Tests should be planned
  3. The Pareto Principle applies to testing
  4. Testing begins “in the small” and moves toward “in the large”
  5. Exhaustive testing is not possible

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