

# Software Engineering Practice

- YOU practice software engineering
- Practice includes: concepts, methods, tools, and principles
- What is SE really? Problem solving

# Problem Solving

- George Polya
  - Understand the problem
    - Communication (planning?)
    - Analysis
  - Plan a solution
    - Planning
    - Modeling, design
  - Carry out the plan
    - Coding
  - Check results for accuracy
    - Testing and quality assurance

# Understanding Problem

- Who are the stakeholders?
- What are the unknowns?
- Can I break the problem down?  
(compartmentalization)
- Can I represent the problem graphically?

# Planning a solution

- Have I seen this before?
- Has this already been solved?
- What are the sub-problems?
- Can we get a solution that makes sense?

# Carry out the plan

- Does my solution conform to the plan?
- Is each piece correct?
  - Code review
  - Testing

# Examine Result

- Is it possible to test?
  - Do you have a test plan?
- Does the solution meet the standards?
  - Did we meet all requirements?

# Core SE Principles

1. The reason it all exists: provide value to users
2. KISS (Keep it simple, stupid!): make solution as simple as possible, but no simpler (iterative refinement)
3. Maintain the vision: everybody needs to row in the same direction
4. What you produce, others will consume: someone else will have to understand your work

# Principles cont.

١. Be open to the future: change will happen, successful systems start out with this in mind, “never design yourself into a corner”
٢. Plan for reuse: communication among teams (must know that something exists to reuse it)
٣. THINK! Clear, complete thought before action produces best results

# Communication

- **LISTEN:** focus on speaker, not your response, ask for clarification but don't interrupt too often, don't be contentious (rolling eyes, shaking head)
- **PREPARE BEFORE COMMUNICATING:** research client, learn terminology
- **USE A FACILITATOR:** keep the conversation focused, mediate conflict

# Comm Cont.

- **FACE TO FACE IS BEST:** but come with materials
- **TAKE NOTES:** sometimes things get lost, write down everything
- **STRIVE FOR COLLABORATION:** start small, small collaborations build trust
- **STAY FOCUSED:** more people = confusion

# More Comm

- DRAW PICTURES: “a picture is worth a thousand words”
- MOVE ON: 1) understood, agreed 2) disagree 3) too complex at this point
- NOT A CONTEST: negotiations occur, but should be mutually beneficial

# Planning

- Preparing for software development
- “minimalists” – light planning, they expect change
- “traditionalists” – heavy planning, they expect change too, but plan for it
- “agilists” – quick planning, they believe the plan will form as work happens

# Principles for planning

- Understand the scope
- Involve the customer in planning
- Planning is iterative
- Estimate based on your knowledge
- Consider risks (contingency planning)
- Be realistic
- Refine granularity as you define the plan
- Define quality (ensure quality)
- Describe processes for change
- Track the plan and make adjustments

# Modeling

- Some abstraction of a real “thing”
- 2 types of modeling
  - Analysis: customer requirements
    - Information
    - Functional
    - Behavioral
  - Design: for the practitioner
    - Software architecture
    - User interface
    - Component details

# Principles for Analysis

- Information domain must be represented and understood
- Functional domain must be represented
  - You must define the functionality of the system
- Behavior of the system must be represented (interaction design)
- Break down the problem into small pieces
- Move from essentials to implementation

# Principles for Design Modeling

- Design should be traceable to analysis
- Keep the architecture in mind
- Data design is just as important as functional design
- Interfaces must be designed with care
- User interface must address user needs
- Each component should be functionally independent

# More...

- Components should be loosely coupled
- Overall design should be understandable
- Design is an iterative process, always strive for simplicity

# Construction

- Coding and testing
- Coding
  - Source code in a programming language
  - Automatic code generation from high-level descriptions
  - Automatic code generation from 4GL (Visual Basic, Visual C++, etc)
- Testing
  - Unit testing
  - Integration testing
  - Validation testing
  - Acceptance testing

# Coding (preparation)

- Understand the problem
- Understand the design, and the design concepts
- Pick an appropriate programming language
- Select the right environment (Dev, Eclipse, .NET)
- Create unit tests

# Coding (actual coding)

- Constrain algorithms
- Select appropriate data structures
- Understand the software architecture, so you can create appropriate interfaces
- Keep logic as simple as possible
- Make sure nested loops are testable
- Select meaningful variable names, follow local standards
- Write self-documenting code
- Use a consistent visual layout

# Coding (validation)

- Conduct walkthroughs
- Perform unit tests
- Refactor the code

# Testing

- All tests should be traceable to requirements
- Tests should be planned long before testing starts
- Pareto principle applies to testing
  - 90-10 or 80-20, hard to isolate the problem parts
- Start by testing small, move to large scale
- Exhaustive testing is not possible

# Deployment

- Customer expectations should be managed
- A complete delivery package should be created and tested before given to customer
- Establish some support mechanism before delivery
- Provide appropriate instructional materials
- Buggy software should be fixed first, delivered later