# The Software Process - Part Two

Charles Carter

October 26, 2017

# Table of Contents

# Software Engineering Practice

# The Essence of Practice

- Understand the problem (communication and analysis)
- Plan the solution (modelinbg abd design)
- Execute the plan (construction and implementation)
- Test for accuracy (testing and quality assurance)

# Understand the problem

- Who has a stake in the solution to the problem?
- What are the unknowns?
- Can the problem be compartmentalized?
- Can the problem be represented graphically?

# Plan the solution

- Have you seen similar problems before?
- Has a similar problem been solved?
- Can subproblems be defined?
- Can a design model be created?

# Execute the plan

- Does the solution conform to the plan?
- Is each part of the solution correct?

# Test for accuracy

- Is it possible to test each component of the solution?
- Does the solution produce the correct results?

# Core Principles

# Core principles

- It exists for a reason.
- KISS - keep it simple, stupid!
- Maintain the vision.
- Others will consume what you produce.
- Be open to the future.
- Plan ahead, especially for reuse.
- Think

# It exists for a reason

There is one and only one reason for a software system: *to provide value to its users*!

# Keep it simple and sweet

All design should be as simple as possible but no simpler. *avoid needless complexity*!

# Maintain the vision

Without conceptual integrity, a system becomes a hash of parts, not held together by incompatible designs. *A clear vision is essential to success*!

# What you produce, others will consume

Always *specify, design, and implement* in such a manner that others can understand what you are doing. "Others" include yourself the next year, the next month, the next week, and even the next day.

# Be open to the future

*Mutatis mutandis*, nothing is permanent except change. Do not paint yourself in a corner — the system you are building may outlive you.

# Plan ahead

Why build something twice when you only have to build it once? Achieving a high degree of reuse is exceedingly difficult. *Plan ahead*!

# Think!

Ready, fire, aim? NO!!! Think before you act! *Think!*

# Communication Practices

# Communication practices

- Listen
- Prepare
- Facilitate
- Face to face is best
- Take notes
- Collaborate
- Focus and modularize
- Draw pictures
- Move on
- Win/win

# Listen

Focus on understanding others, and not your response to what they are saying. To be understood, first, understand.

# Prepare before you communicate

*Before you meet*, understand the problem, do research, understand specialized vocabularies, prepare an agenda so that you can stay on track.

# Someone should facilitate the discussionh

You must have a discussion leader. The leader's job is (a) keeping the discussion moving forward, (b) mediate the inevitable conflicts, and (c) ensure that the other principles are followed.

# Face to face communication is best

Make sure all interested parties are represented in the discussion. Everyone has a voice!

# Take notes and document decisions

Appoint a recorder. Make sure that all topics and all decisions and documented. Memories fail! Distribute the notes to all participants.

# Strive for collaboration

Building software is not a war! Collaboration builds trust, builds confidence, and builds the product.

# Stay focused, modularize the discussion

Don't chase rabbits. Discuss and decide one thing at a time. Conclude one topic before beginning another (if possible) — don't begin discussion of the next topic until the previous one has been resolved.

# Draw pictures

Don't depend only on verbal communication. If in doubt, *draw a picture*!

# Move on

Don't get bogged down. Don't chase rabbits. Don't revisit closed issues. Stay focused. If an issue cannot be resolved, defer it to a later discussion, and move on to the other points.

# Negotiation is not a game.

Negotiation works when all parties win. Everyone has the same goal, and evelyone has a stake in the outcome. Successful negotiation sometimes requires compromise.

# Planning Practices

# Planning practices

- Understand the scope
- Involve the customer
- Planning is iterative
- Estimate on what you know
- Consider the risk
- Be realistic
- Adjust granularity
- Intentionally ensure quality
- Describe change accomodation
- Track the plan

# Understand the scope of the project

Scope provides the team with a destination. If you don't know where you are going, you will not get there.

# Involve the customer in planning

The customer defines priorities and sets constraints. You need to establish these with the customer.

# Recognize that planning is iterative

Things change. You deal with change by adjusting the plan at intervals to accommodate change. Understand that the execution of the plan often results in changes to the plan.

# Estimate based on what you know

Don't guess. Estimates based on unreliable, vague, and uncertain will themselves be unreliable, vague, and uncertain.

# Consider the risk as you define the plan

The greater the risk, either in terms of likelihood or impact, the greater the need for planning with respect to the risk,

# Be realistic

Uncertainty, ambiguity, omissions, mistakes, and "noise" are facts of life. No one ever accomplishes every part of every plan. No one ever performs with absolute perfection. Stuff happens. Factor it in.

# Adjust granularity as you define the plan

Take tine and distance into account. Detailed planning is appropriate for tasks that are near in terms of time and completion. For tasks that are distant in time and completion cannot be planned with a fine degree of granularity. You can plan tomorrow hour by hour, but you cannot plan next year even week by week.

# Define how you intend to ensure quality

Make quality assurance a specific part of your plan. If you don't plan for it, you won't do it. Also, plan for particular measures of product quality.

# Describe how you intend to accommodate change

Uncontrolled change can wreck the best plan. Make plans to change the plan. Intentionally plan on how you will deal with changes to the plan.

# Track the plan and make adjustments as required

How are you following the plan? Are you on schedule? Are you within your budget? Have you reached the planned milestone? Part of the plan should address progress of plan execution and adjustments to be made if the plan goes off track.

# Analysis Modeling

# Analysis modeling

- Understand the knowledge domain
- Define the functionality
- Represent the behavior responsive to external stimuli
- Partition models to uncover details
- Move from the abstract to the concrete

# The knowledge domain must be represented and understood

"Knowledge domain" describes all the inputs to the system, all the outputs, and all the internal processing variables. Understanding and representing the data is essential.

# The functions that the software performs must be defined

If you can not model the behavior of the software, you cannot build it. You canmot implement the software unless and until you understand fully what it does.

# The behavior of the software, in response to external events, must be represented

All software exists in sonme external environment. All software responds in some manner to stimuli from the external environment. Every model must represent the response of the software product to external events.

# Models must be partitioned to uncover details

Complex problems are difficult to solve. Accordingly, we break the problem into subdomains that are easier to solve. Break the problem into modules, partitions, or subproblems that reveal details that lend themselves to solution.

# Move from the abstract to the concrete

Address the "essence" of the problem first, and attempt abstract solutions. Then, piece by piece, work toward more specific and particular solutions that can be submitted to a machine for solution.

# Design Modeling

- Design traceable to analysis
- Take architecture into account
- Design data as well as processing
- Carefully design interfaces
- Tune user interface to the needs of users
- Design independent components
- Loosely couple components
- Easily understood design representations
- Develop design iteratively

# Traceable Design

Design should be traceable to the analysis model

# Consider the Architecture

Always consider the architecture of the system you are building

# Design the Data

Design of the data is as important as the design of the processing functions

# Design Interfaces

Interfaces, bothg external and internal, must be carefully designed

# Consider Users

User interface design should be tuned to the needs of the users

# "Single minded" Components, High Cohesion

Component level design should be independent

# Loose Coupling

Components should be loosely coupled to one another and the external environment

# Understandable Models

Design models (representations) should be easily understood

# Iterative Development

Thge design should be developed iteratively, striving for greater simplicity with each iteration

# Coding Principles

# Preparation

- Understand the problem you are trying to solve
- Understand basic design concepts
- Choose the correct programming language
- Choose the correct programming environment
- Createye automated unit tests

# Coding

- Use structured programming
- Use data structures that meet the needs of the design
- Understand the architecture, create consistent interfaces
- Simplify your logic, keep it as simple as possible
- Nested loops should be easily testable
- Select meaningful names
- Write self documenting code, **Write self documenting code**
- Create a visual layout that aids understanding

# Validation

- Conduct code walk-throughs
- Perform unit tests
- Refactor the code

# Testing Principles

# Testing Principles

- Test against requirements
- Pre-plan testing
- Recognize the Pareto principle
- Test from small to large
- Exhaustive testing isn't possible

# Test against requirements

The objective of testing is to uncover defects. From a customer's point of view, software that does not meet requirements is defective.

# Pre-plan testing

Plan for testing when you analyze the requirements. Use the tests to verify the design.

# Recognize the Pareto principle

The 80/20 rule. Eighty percent of errors will be found in twenty percent of the code.

# Test from small to large

Test smallest components first: assignments, expressions, functions. Move toward larger components. Test the largest components last.

# Exhaustive testing isn't possible

Adequately test the software, recognizing the impossibility of testing every case. Strive toward comprehensive coverage.

# Deployment

- Manage customer expectations
- Assemble and test complete packages
- Plan support before delivery
- Properly instruct end users
- Fix first, deliver later

# Manage customer expectations

If the customer expects you to deliver more than you contract for, he will be disappointed and you will fail.

# Assemble and test complete packages

Don't deliver piecemeal. Test your complete package in beta before delivers, don't assemble on the customer's time.

# Plan support before delivery

You *will* have to support your work. Agree before delivery as to how, when, and what that support will consist of.

# Properly instruct end users

You are not delivering only software, but knowledge-ware. If your customer can't use it, you might as well have not written it.

# Fix first, deliver later

Don't deliver low quality increments, expecting to fix them later. Don't put yourself in the position of telling the customer that the bugs will be fixed in the next release.