

CS 350, slide set 5

---

M. Overstreet  
Old Dominion University  
Spring 2005

---

---

---

---

---

---

---

---

Reading

---

- PSP text, ch. 11, 12, 13, 14

---

---

---

---

---

---

---

---

Topics

---

- A software development process
- Measuring software quality
- Finding defects
- A code review checklist

---

---

---

---

---

---

---

---

## Process

- A sequence of steps to complete a task
  - Provides guidance
  - Can provide a record of what has been accomplished
  - Can provide an indication of percent of project which has been completed
    - After we've used 50% of time allocated, have we finished 50% of the work?

---

---

---

---

---

---

---

---

## Some basic definitions

- Recall defn's from Ch. 5
  - **Product**: something you produce, usually for someone else
  - **Project**: produces a product
  - **Task**: an element of work
  - **Process**: the steps to produce a product
  - **Plans**: the way a specific project is to be done; how, when, at what costs
  - **Job**: something you do; either a project or a task
- More
  - Processes can have various **phases**
  - Each phase can consist of several activities

---

---

---

---

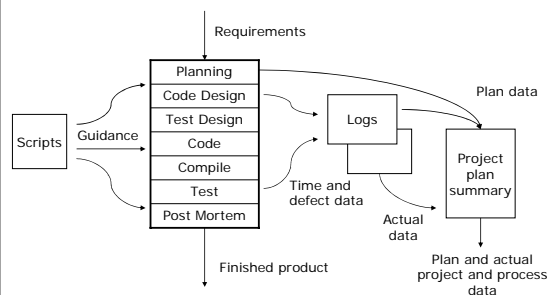
---

---

---

---

## PSP process



---

---

---

---

---

---

---

---

## PSP process script - 1

	Purpose	To guide you in developing programs
	Entry Criteria	Problem statement PSP Project Plan Summary form Actual size and time data from old projects Time Recording Log
1	Planning	Identify program functions <b>Identify test objectives</b> Estimate max, min, and total LOC Determine minutes/LOC Calculate max, min, and total time Enter data in Project Plan Summary Form Record planning time in Time Recording Log
2	Code design	Design the program Record design in specified format Enter code design time in Time Recording Log
3	Test design	<b>Prepare test plans and data</b> Enter test design time in Time Recording log

---

---

---

---

---

---

---

---

## PSP Process Script - 2

3	Code	Implement the design <b>Complete a Makefile</b> Use a standard format for entering code Record coding time in Time Recording Log
4	Compile	Compile the code Fix defects as discovered Record time in Time Recording Log
5	Test	Test the program Fix defects as discovered <b>Complete test reports</b> Record time in Time Recording Log
6	Postmortem	Complete the Project Plans Summary with actual time and size data Record postmortem time in Time Recording Log
	Exit criteria	A thoroughly tested program A properly documented design Complete source code <b>Documented and corrected test data</b> <b>A Makefile</b> <b>A completed set of test reports</b> A completed Project Plan Summary Completed Time Logs: <b>All required submitted.</b>

---

---

---

---

---

---

---

---

## Software Quality

- Complex topic
  - How do customers measure the quality of a product?
  - How do developers measure the quality of a product?
- Includes, among many possibilities:
  - How many problems does a typical user encounter when using the product?
  - Does the product provide key functionalities to the user?
  - How do features of this product compare with competing products?
  - How easy is it to enhance/modify/extend/revise/tailor the product?
  - How easily can components from this product be used in other products?
- We're going to keep it simple: defects/KLOC

---

---

---

---

---

---

---

---

## What's a defect in PSP?

- Anything wrong with a program:
  - Misspelling in comment
  - Bad punctuation, poor formatting
  - Incorrect logic
  - Missing functionality
  - Documentation, format does not conform to organizational standard
- In short, **any change** you make to code after you type it in and proofread it.
- **Errors** are mistakes people make. Some result in **defects** in code.

---

---

---

---

---

---

---

---

## Why record defect types?

- To have real data on what is causing problems
- To help direct possible changes to software processes
- Pointless unless defect types, frequencies, and times-to-fix are periodically analyzed and acted upon

---

---

---

---

---

---

---

---

## Goals of defect form

- Improve your programming
- Reduce number of defects in your code
- Save you time
- To do you job responsibly: you find your own mistakes rather than someone else finding them

---

---

---

---

---

---

---

---

## Fixing defects

- Recognize defect symptoms
- From symptoms, deduce likely locations of defects
  - Sometimes hard to do, sometimes not
- Identify source
- Fix it
- Verify that the fix:
  - Resolved the problem
  - Did not introduce new problems

---

---

---

---

---

---

---

---

## Ways to find code defects

- Let others find them (**not** recommended)
  - This approach has some disadvantages
- Use testing to locate defects
  - Widely used; very expensive; misses many
- Use mathematical methods (proofs of correctness) to locate defects
  - Rarely used; very expensive, maybe infeasible in most cases
- Use Code Reviews
- Use Code Inspections (will discuss later)

---

---

---

---

---

---

---

---

## Why Code Reviews?

- Compilers miss lots of typo/syntax errors (probably 10% -- depends on language)
- Testing misses lots of defects (some assert 50%)
  - No matter how many defects are in the code to begin with!
- Testing and code reviews are probably complimentary activities
  - Code reviews can find some defects not easily found in testing
  - Testing finds can some defects not easily found by code reviews

---

---

---

---

---

---

---

---

## Review **before** compiling!

- Takes as long to do review before as after
- Saves some compile time
  - Typically 10% to 15% if done without review
  - Typically ~3% if done with review
- Compiles finds errors equally well with or without review
- Reviewers don't find errors as effectively if they know the code has compiled cleanly. Probably human nature:
  - if we don't really expect to find things, we don't look as hard.
  - if we expect to find things and don't, we probably look again.

---

---

---

---

---

---

---

---

## Additional comments

- In PSP, you do your own reviews.
  - Goal is to save you time, improve your code quality
- In TSP (remember the next textbook?), this is supplemented by inspections.
- Since fixing code is so expensive, we have much industrial data that shows reviews pay!

---

---

---

---

---

---

---

---

## Code review script - 1

	Entry criteria	Check that the following are on hand: 1. The requirements statement 2. The program design 3. The program source code 4. The coding standards 5. A Defect Recording Log
1	Review procedure	First produce the finished program source code. Before compiling, print a source listing Next, do the code review During the code review, carefully check every line of source code to find and fix as many defects as possible
2	Fix the defects	Fix all defects found Check the fixes to ensure they are correct Record the defects in the Defect Recording Log
3	Review for coverage	Verify that the program design fulfills all of the functionality described in the requirements Verify that the source code implements the design

---

---

---

---

---

---

---

---

## Code Review Script - 2

4	Review the program logic	Verify that the design logic is correct Verify that the source code correctly implements the design logic
5	Check names and types	Verify that all names and types are correctly declared and used Check for proper declaration of integer, long int., and floating point data types
6	Check all variables	Ensure that every variable is initialized Check for overflow, underflow, and out-of-range problems
7	Check program syntax	Verify that the source code properly follows the language specifications
	Exit criteria	At completion you must have: 1. The completed and corrected source code 2. Completed Time Recording Log 3. Completed Defect Recording Log

---

---

---

---

---

---

---

---

## Code Review Checklist

- See text, table 14.1, pg. 177
- Use different checklists for different languages
  - Different languages have different shortcomings
    - While some mistakes are common-place in many languages, features of a particular language often cause people to make similar mistakes
    - Your checklist should evolve and be based on what you discover about your own behavior from your own code reviews!

---

---

---

---

---

---

---

---

## C++ Coding Standard

- See text, Table 14.9, pg. 189
- Starting with assignment 4, this is what the grader will use for future assignments.
- Follow formatting rules.
  - Some editors automatically indent. Take advantage of this.
- Grading rules:
  - Any submitted code that does not fully comply with the coding standard from the text will not be graded.

---

---

---

---

---

---

---

---