CPSC 131
Data Structures Concepts

Software Engineering Principles

Mr. Allen Holliday
aholliday@fullerton.edu

Based on Chapter 1 of:

C++ Plus Data Structures

Nell Dale, Chip Weems, Tim Richards

Jones & Bartlett Learning

The Software Process

Software Life Cycle

- Problem Analysis
- Requirements Specification
- Design, high-level and low-level
- Testing and verification
- Delivery
- Operation
- Maintenance: corrections and enhancements

An iterative process, multiple passes through the life cycle.

Goals of Quality Software

- It works
- It can be modified without excessive time and effort
- It is reusable, in whole or in part
- It’s completed on time and within budget

A Path To Failure

1. Google for something that looks like it might fit the problem
2. Randomly change parts until:
   - It works
   - You give up

- It’s tedious and frustrating
- It’s error-prone
- It’s usually unsuccessful (the random part)

Specification: Understanding the Problem

Approaching the Problem

- Class project common first steps:
  - Panic
  - Start typing code
  - Drop the course
  - Stop and think
- “Think before you code”—an excellent motto for this course.

Define the Problem

- Create a list of expected inputs and outputs.
- Outline the necessary processing to take those inputs and produce those outputs
- Consider error processing
  - There should be no unexpected inputs
  - Anticipate incorrect or missing data
- Create usage scenarios, sequences of steps
- State your assumptions clearly

Program Design

Basic Principles

- Abstraction: a simple model of a complex system that includes only essential details
- Information hiding: decompose system into modules with visible functionality and hidden implementation
- Stepwise refinement: further decomposition—modules into submodules, then subsubmodules, and so on… (divide and conquer)

Object-Oriented Design

- Don’t separate data from executable code
- Define *objects*, capsules of data and code
- Represent real-world things
- Be abstract
  - Define meaningful names of data
  - Define behavior, not algorithms

Contact List Example

- A *List* is an object that contains *contacts*
- *A List* has functions for adding *Contacts* to itself, finding a *Contact*, deleting a *Contact*.
- A *Contact* holds name, address, phone number
- A *Contact* has functions that modify its data

Two Design Levels; Two Important Words

- "What"
  - An abstract description of a program's behavior and appearance
  - The program or module as a black box
- "How"
  - The actual code
  - A clear box, showing its insides

Visual Tools

- Draw pictures—diagrams of data structures
- Outline algorithms—pseudocode
- Dream up scenarios—sequences of steps through the algorithms
- Execute scenarios—mark up the diagrams as you go through sequences.

Verification of Correctness

More Important Words

- Verify: “Did I do things right?”
  - Was the design and code correct?
- Validate: “Did I do the right thing?”
  - Was the problem understood?
  - Was the customer’s need met?

Find Errors Early

- The later an error is found, the more expensive it is to fix it.
- Later means more rework
- Fixing a requirements error? Rewrite some text, redraw a diagram
- Fixing an error found during code testing?
  - Maybe it’s a design flaw with wide ripple effects
  - Maybe it’s a misstated requirement with even wider ripple effects

Black-Box Testing

- Consider only inputs and outputs
- Provide a set of inputs
  - Use boundary cases: midrange, smallest, largest, just below smallest, just above largest
- Confirm that the outputs are as expected
- Include error and out-of-range cases

Clear-Box (White-Box) Testing

- Consider the internal logic
- Statement coverage; each one at least once
- Branch coverage; the true and false of an *if*, every *case* of a *switch* (including the default)
- Nesting multiplies the number of tests
- Loops: boundaries again
    - maximum count , minimum count, once, never

A Black-Box Example

- Divide (dividend, divisor, quotient, remainder, error)
- Input cases:
  - dividend and divisor are both positive
  - dividend and divisor are both negative
  - dividend is positive, divisor is negative
  - dividend is negative, divisor is positive
  - dividend is zero
  - divisor is zero

A Clear-Box Example

```
//      Open the input file
inputFile >> command
while (command != “Quit”)
{
    Function(command);
    inputFile >> command;
}
```

- There are several errors—find them

Remember The Motto.

Do you? It might be on a quiz.

“Think Before You Code”