CMPSC 24: Lecture 1
Software Development Process

Divyakant Agrawal
Department of Computer Science
UC Santa Barbara

Syllabus and other course information

- http://www.cs.ucsb.edu/~agrawal/spring2010/cmpsc24.html

Announcements

- Revised curriculum for Lower Division courses:
  - Prepare you for the real-world
  - Based on the principle of "Learn by Practice"
  - Enable students to solve problem in language-agnostic manner
  - Hands-on Laboratory Sections
  - Extensive programming projects
  - Pair programming (i.e., team approach for software development)

Lecture outline

- Software Development processes and principles
- Abstraction, Encapsulation, and Information Hiding:
  - Abstract Data Types
  - Classes in C++

Software Process

- **Software Engineering**
  - A disciplined approach to the design, production, and maintenance of computer programs that are developed on time and within cost estimates, using tools that help to manage the size and complexity of the resulting software products
- **Software Process**
  - A standard, integrated set of software engineering tools and techniques used on a project or b an organization

5

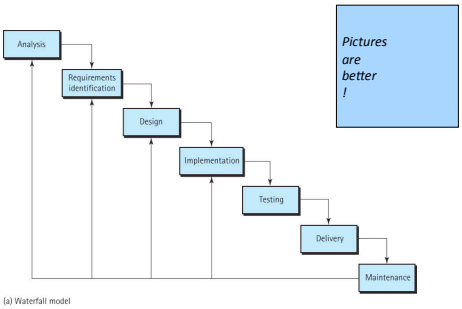
The Software Life Cycle

- Problem analysis
- Requirements identification
- Software specification
- High- and low-level design
- Implementation
- Testing and Verification
- Delivery
- Operation
- Maintenance

List are so dull !

6

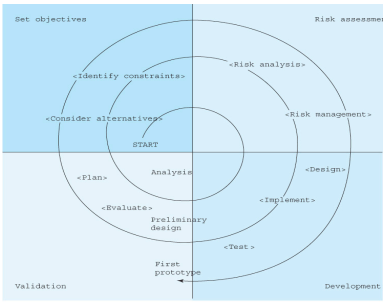
Software Life Cycle: Waterfall Model



(a) Waterfall model

7

Software Life Cycle: Spiral Model



(b) Spiral model

8

Agile Software Development

- Reduce the development cycle
- Short iterations
  - A working prototype at the end of each iteration
- In tune with user requirements
  - Minimize risks
- Minimal documentation
- Face to face communication among programmers
  - Pair programming
- Working software as the primary means of progress
- In wide use currently

Programmer's ToolBox

- **Hardware**
  - The computers and their peripheral devices
- **Software**
  - Operating systems, editors, compilers, interpreters, debugging systems, test-data generators, and so on
- **Ideaware**
  - Shared body of knowledge
  - Algorithms

Including all you learn in this course!

10

Goals of Quality Software

- It works.
- It can be modified without excessive time and effort.
- It is reusable.
- It is completed on time and within budget.

11

Goals of Quality Software

- **Requirements**
  - A statement of what is to be provided by a computer system or software product
- **Software specification**
  - A detailed description of the function, inputs, processing, outputs, and special requirements of a software product; it provides the information needed to design and implement the program

12

Program Design

- **Abstraction**
  - A model of a complex system that includes only the details essential to the perspective of the viewer of the system
- Programs are abstractions
- **Module**
  - A cohesive system subunit that performs a share of the work; an abstraction tool

13

Program Design

- **Information Hiding**
  - The practice of hiding the details of a function or data structure with the goal of controlling access to the details of a module or structure
- **Abstraction and information hiding are fundamental principles of software engineering**

14

Stepwise Refinement

- A problem-solving technique in which a problem is approached in stages and similar steps are followed during each stage, with the only difference being the level of detail involved
  - Top-down
  - Bottom-up
  - Functional decomposition
  - Round-trip gestalt design

15

Two Approaches to Building Manageable Modules

FUNCTIONAL DECOMPOSITION

Divides the problem into more easily handled subtasks, until the functional modules (subproblems) can be coded.

FOCUS ON: processes

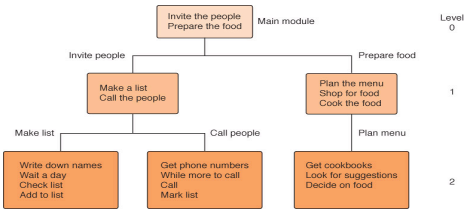
OBJECT-ORIENTED DESIGN

Identifies various objects composed of data and operations, that can be used together to solve the problem.

FOCUS ON: data objects

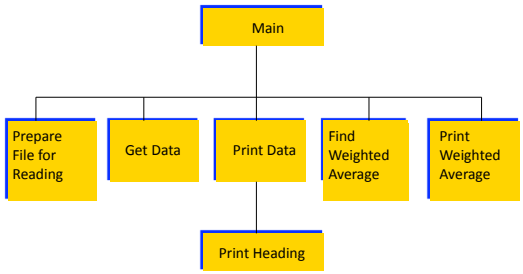
An Example of Functional Decomposition

- Planning a large party



17

Functional Design Modules



Object-Oriented Design

A problem-solving methodology that produces a solution to a problem in terms of self-contained entities called *objects*

Object

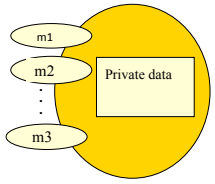
–A thing or entity that makes sense within the context of the problem

For example, a *student*, a *car*, *time*, *date*

19

Object-Oriented Design

A technique for developing a program in which the solution is expressed in terms of objects -- self-contained entities composed of data and operations on that data.



World View of OOD

Problems are solved by

- isolating the objects in a problem,
- determining their properties and actions (responsibilities), and
- letting the objects collaborate to solve a problem

21

Object-Oriented Design

An analogy: You and your friend fix dinner
- Objects: you, friend, dinner

- Class: you and friend are people
  - –People have name, eye color, …
  - –People can shop, cook, …

- Instance of a class: you and friend are instances of class People, you each have your own name and eye color, you each can shop and cook

- You collaborate to fix dinner

22

Object-Oriented Design

- **Class** (or object class)
  - –A description of a *group* of objects with similar properties and behaviors; a pattern for creating individual objects

- **Object** (instance of a class)
  - –A concrete example of the class

- **Classes** contain fields that represent the
  properties (name, eye color) and
  behaviors (responsibilities) (shop, cook) of the class
- **Method**
  - –A named algorithm that defines behavior (shop, cook)

23

Data Abstraction

Separation of a data type's logical properties from its implementation.

LOGICAL PROPERTIES	IMPLEMENTATION
What are the possible values? What operations will be needed?	How can this be done in C++? How can data types be used?

24

Data Abstraction

The separation of the representation of data from the applications that use the data at a logical level; a programming language feature that enforces information hiding.

APPLICATION

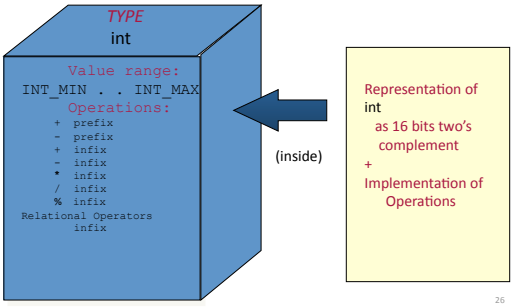
```
int  y;

y = 25;
```

REPRESENTATION

```
0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1
```

25

Abstraction of C++ Data Type int



26

Abstract Data Type (ADT)

- **A data type whose properties (domain and operations) are specified independently of any particular implementation.**

27

Data from 2 different levels

- ***Logical (or ADT) level:*** **abstract view of the domain and operations.** WHAT
- ***Implementation level:*** **specific representation of the structure to hold the data items, and the coding for operations.** HOW

28

C++ **class** data type

- **A class is an unstructured type that encapsulates a fixed number of data components (data members) with the functions (called member functions) that manipulate them.**
- **The predefined operations on an instance of a class are whole assignment and component access.**

29

An Example (High Level)

- Special type of a Queue:
  - Entities in the Queue have priority
  - Service the queue entities (people, files, forms, etc.) in the priority order
- WHAT concerns:
  - Add to the queue
  - Remove from the queue
- HOW concerns:
  - Internally: maintain the entities sorted? Unsorted?
  - Many choices at the design/implementation time

ADDITIONAL MATERIAL – FOR FURTHER READING/REVIEW

Verification of Software

- **Program verification**
  - The process of determining the degree to which a software project fulfills its specifications
- **Program validation**
  - the process of determining the degree to which software fulfills its intended purpose

32

Verification vs. Validation

Program verification asks,

"Are we doing the job right?"

Program validation asks,

"Are we doing the right job?"

B.W. Boehm, Software Engineering Economics, 1981

33

Bugs!

- Dijkstra decries the term "bugs"

• Ever since the moth was found in the hardware, computer errors have been called *bugs*. Edsger Dijkstra chides us for the use of this terminology. He says that it can foster the image that errors are beyond the control of the programmer—that a bug might maliciously creep into a program when no one is looking. He contends that this is intellectually dishonest because it disguises that the error is the programmer's own creation.

34

Verification of Software

Where do you begin to look for errors?

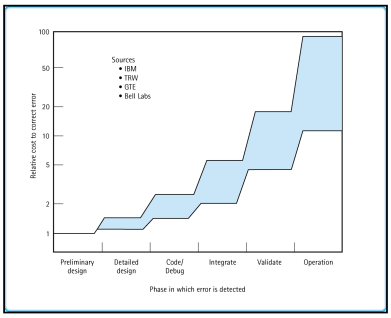
35

Kinds of errors

- Specification and design errors
  - *Can you give an example?*
- Compile-time errors
  - *Can you give an example*?
- Run-time errors (data)
  - *Can you give an example?*
- **Robustness**
  - The ability of a program to recover following an error; the ability of a program to continue to operate within its environment

36

Cost of an Error Based on When It Is Discovered



Verification of Software

- Testing
  - The process of executing a program with data sets designed to discover errors
- Formal verification
- Other aspects

38

Formal Verification

- **Pre-conditions**
  - Assumptions that must be true on entry into an operation or function for the postconditions to be guaranteed.
- **Post-conditions**
  - Statements that describe what results are to be expected at the exit of an operation or function, assuming that the preconditions are true
- **Loop invariants**
- **Logics & theorem proving**
- **Model checking**

39

Other Aspects of Verification

- **Desk checking**
  - –Tracing an execution of a design or program on paper
- **Walk-through**
  - –A verification method in which a team performs a manual simulation of the program or design
- **Inspection**
  - –A verification method in which one member of a team reads the program or design line by line and the other members point out errors

40

Exceptions

An unusual, generally unpredictable event, detectable by software or hardware, that requires special processing; the event may or may note be erroneous

Three parts to an exception mechanism:

- **Define** the exception
- **Generate** (raising) the exception
- **Handling** the exception

Try, throw, and catch statements in C++

41

Program Testing

- For each data set
  - Determine inputs that demonstrate the goal of test case
  - Determine the expected behavior of the program
  - Run the program and observe the resulting behavior
  - Compare the expected behavior and the actual behavior

42

Types of Testing

Unit testing

Testing a module, class, or function by itself

Black-box testing

Testing a program or function based on the possible input values, treating the code as a "black box"

Clear (white) box testing

Testing a program or function based on covering all of the statements, branches, or paths of the code

43

More kinds of testing

- Acceptance test
  - The process of testing the system in its real environment with real data
- Regression testing
  - Re-execution of program tests after modifications have been made to ensure that the program still works

44

Program Testing

Statement coverage

Every statement in the program or function is executed at least once

Branch

A code segment that is not always executed; for example, a *switch* statement has as many branches as there are case labels

Path

A combination of branches that might be traversed when a program or function is executed

Path testing

A testing technique whereby the tester tries to execute all possible paths in a program or function

45

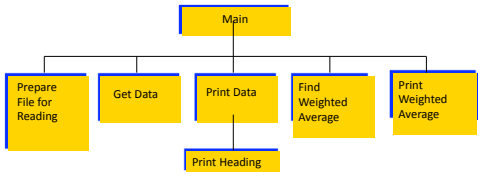
Program Testing

- For program testing to be effective, it must be planned
- Start planning for testing before writing a single line of code
- A program is not complete until it has been thoroughly tested
- Your test plan should convince a reader that the program is correct

46

Integration Testing

- Is performed to integrate program modules that have already been independently unit tested.



Integration Testing Approaches

TOP-DOWN

Ensures correct overall design logic.

USES: placeholder module "stubs" to test the order of calls.

BOTTOM-UP

Ensures individual modules work together correctly, beginning with the lowest level.

USES: a test driver to call the functions being tested.

Test plan

- Document showing the test cases planned for a program or module, their purposes, inputs, expected outputs, and criteria for success
  - Goals
  - Data to test goals
  - Expected output
- **Implementing a test plan**
  - Running the program with the test cases listed in the test plan

49