

# CMSC 132: Object-Oriented Programming II

---



## Software Development

Department of Computer Science  
University of Maryland, College Park

# Modern Software Development

## ■ Why do we want to study the software development process?

### ■ To understand

- Software development problems
- Why software projects fail
- Impact of software failures
- How to develop better software

# Software Engineering

## ■ Definition from Wikipedia

- Field that creates and maintains software applications by applying technologies and practices from computer science, project management, engineering, application domains, and other fields.

# Software Development Problems

- **Software is**
  - **Expensive**
    - **Cost per line of code growing (unlike hardware)**
  - **Frequently late**
    - **Schedule overruns**
  - **More expensive than projected**
    - **Cost overruns**
  - **Difficult to use & understand**
  - **Missing features**
  - **Too slow**

# Software Projects Fail

- Anywhere from 25-50% of custom software fail
- Example – FBI Virtual Case File
  - Began Jan 2001
  - Officially scrapped Jan 2005
  - LA Times (Jan 13, 2005)

“A new FBI computer program designed to help agents share information to ward off terrorist attacks may have to be scrapped... Sources said about **\$100 million would be essentially lost** if the FBI were to scrap the software...”

# Software Projects Fail

- **Reasons for failure of FBI Virtual Case File**
  - **Poor specification**
    - 800-page requirement document
    - Repeated changes in specification
    - New requirements continually added
  - **Poor management**
    - Repeated management turnover
    - Micromanagement of software developers
    - FBI personnel with no software experience

# Software Contributing to Real Failures

- **1990 AT&T long distance calls fail for 9 hours**
  - **Wrong location for C break statement**
- **1996 Ariane rocket explodes on launch**
  - **Overflow converting 64-bit float to 16-bit integer**
- **1999 Mars Climate Orbiter crashes on Mars**
  - **Missing conversion of English units to metric units**



# Impact of Software Failures Increasing

- **Software becoming part of basic infrastructure**
  - **Software in cars, appliances**
  - **Business transactions moving online**
- **Computers becoming increasingly connected**
  - **Failures can propagate through internet**
    - **Internet worms**
  - **Failures can be exploited by others**
    - **Viruses**
    - **Spyware**

# Why Is Software So Difficult?

## ■ Complexity

- Software becoming much larger
  - Millions of line of code
  - Hundreds of developers
- Many more interacting pieces

## ■ Length of use

- Software stays in use longer
  - Features & requirements change
  - Data sets increase
  - Can outlast its creators

# Software Size

## ■ Small

- 1-2 programmers, < 3000 lines of code

## ■ Medium

- 2-5 programmers, < 20,000 lines of code

## ■ Large

- 5-20 programmers, < 100,000 lines of code

## ■ Very large

- 20-200 programmers, < 1 million lines of code

## ■ Extremely large

- > 200 programmers, > 1 million lines of code

# Source Lines of Code

## ■ Source lines of code

- Software metric
- Measures the amount of code in a program
- Abbreviated as **SLOC**

## ■ Example software sizes

- Windows 95 – 15 million SLOC
- Windows 98 – 18 million SLOC
- Windows XP – 40 million SLOC
- Windows Vista – 50 million SLOC (estimated)

# Software Size

- **Small software projects**
  - Can keep track of details in head
  - Last for short periods
  - What students learn in school
  
- **Large projects**
  - Much more complex
  - Commonly found in real world
  - Why we try to teach you
    - Software engineering
    - Object-oriented programming

# Software Life Cycle

- Coding is only part of software development
- Software engineering requires
  - Preparation before writing code
  - Follow-up work after coding is complete
- Software life cycle
  - List of essential operations / tasks
    - Needed for developing good software
  - No universal agreement on details
  - Also known as **software development process**

# Components of Software Life Cycle

- 1. Problem specification**
- 2. Program design**
- 3. Algorithms and data structures**
- 4. Coding and debugging**
- 5. Testing and verification**
- 6. Documentation and support**
- 7. Maintenance**

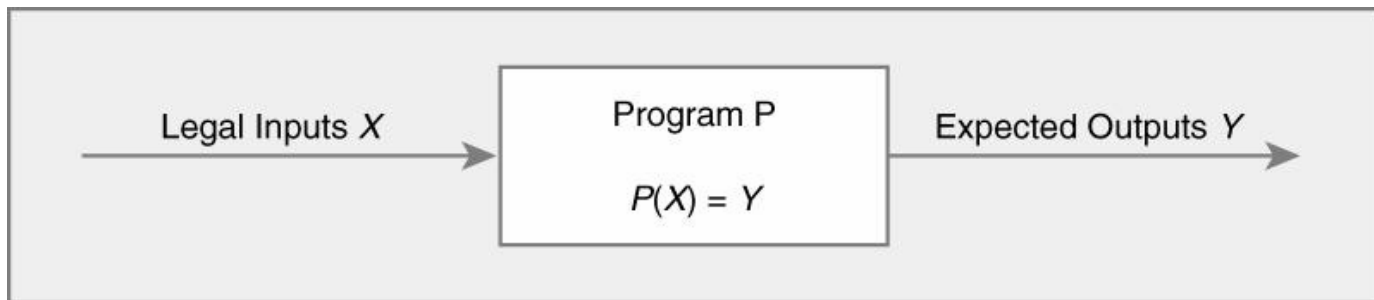
# Problem Specification

## ■ Goal

- Create complete, accurate, and unambiguous statement of problem to be solved

## ■ Example

- Specification of input & output of program



## ■ Problems

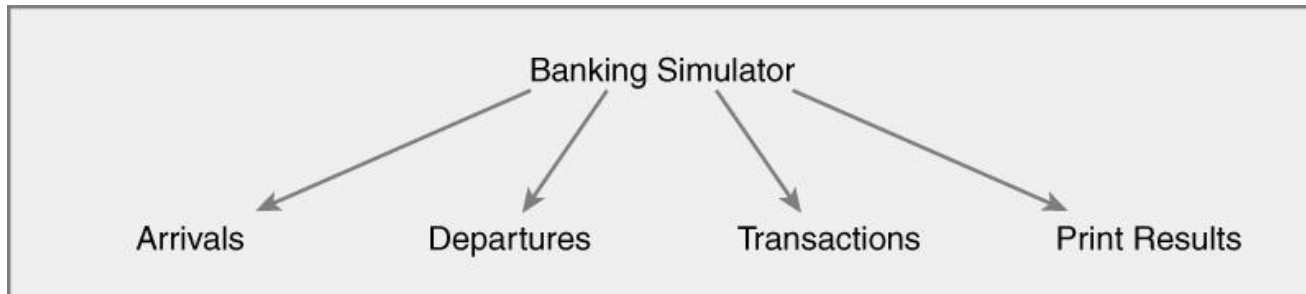
- Description may be inaccurate or change over time
- Difficult to specify behavior for all inputs

# Program Design

## ■ Goal

- Break software into integrated set of **components** that work together to solve problem specification

## ■ Example



## ■ Problems

- Methods for decomposing problem
- How components work together

# Algorithms and Data Structures

## ■ Goal

- Select algorithms and data structures to implement each component

## ■ Problems

### ■ Functionality

- Provides desired abilities

### ■ Efficiency

- Provides desired performance

### ■ Correctness

- Provides desired results

# Algorithms and Data Structures

## ■ Example

### ■ Implement list as array or linked list

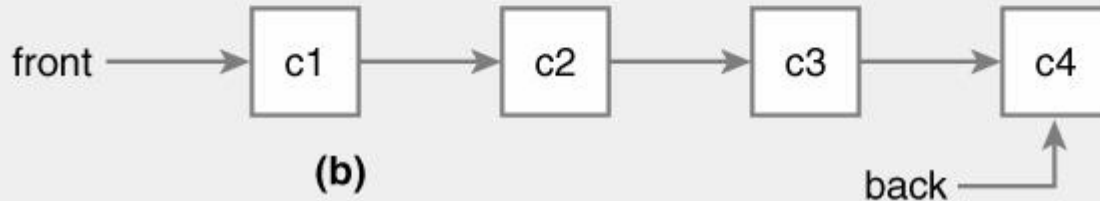
As an array:



front = 0      back = 3

(a)

As a linked list:



(b)

back

# Coding and Debugging

## ■ Goal

- Write actual code and ensure code works

## ■ Problems

### ■ Choosing programming language

#### ■ Functional design

- Fortran, BASIC, Pascal, C

#### ■ Object-oriented design

- Smalltalk, C++, Java

### ■ Using language features

- Exceptions, streams, threads

# Testing and Verification

## ■ Goal

- Demonstrate software correctly match specification

## ■ Problem

### ■ Program verification

- Formal proof of correctness
- Difficult / impossible for large programs

### ■ Empirical testing

- Verify using test cases
  - Unit tests, integration tests, alpha / beta tests
- Used in majority of cases in practice

# Documentation and Support

## ■ Goal

- Provide information needed by users and technical maintenance

## ■ Problems

### ■ User documentation

- Help users understand how to use software

### ■ Technical documentation

- Help coders understand how to modify, maintain software

# Maintenance

## ■ Goal

- Keep software working over time

## ■ Problems

- Fix errors
- Improve features
- Meet changing specification
- Add new functionality