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

- **Expensive**
  - Cost per line of code growing (unlike hardware)
  - More expensive than projected
- **Difficult to understand**
- **Missing features**
- **Too slow**
- **Frequently late**
  - Schedule overruns
  - Example: ARIS →$80 million data and information system for New York City public schools
    - http://www.nytimes.com/2008/10/24/education/24aris.html
  - Brooks's law: Adding manpower to a late software project makes it later
    - http://en.wikipedia.org/wiki/The_Mythical_Man-Month

Software Projects Fail

- Anywhere from 25-50% of custom software fails
- **Example (FBI Virtual Case File)**
  - Began Jan 2001 and 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…"
- **Reasons for failure of FBI Virtual Case File**
  - Poor specification → 800-page requirement document, repeated changes in specification
  - Poor management → Repeated management turnover, micromanagement of software developers, FBI personnel with no software experience

Impact of Software Failures Increasing

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

Software Contributes to Real Failures

- Bugs in software may cause real-world failures
- Example – Air Force F-22A Raptor
  - Stealth fighter costing $300 million each
  - 1.7 millions lines of code for plane's avionics

Software Contributes to Real Failures

- Air Force F-22A Raptor software fails midair
  - DefenseNews.com (March 5, 2007)
  - "When a **dozen Raptors** en route from Hawaii to Japan crossed the International Date Line for the first time, the jets' **Global Positioning System navigation avionics went haywire**, forcing the pilots to turn around."
  - GPS software unable to handle change in longitude from W179.99º to E180
  - Raptor pilots visually followed refueling tankers back to Hawaii

Software Contributes to Real Failures

- Happy ending for Raptor?
  - Lockheed-Martin provided software fix in 48 hours
  - For “operational security reasons” the USAF declined to elaborate, saying only that the F-22A “experienced a software problem involving the navigation system”
- Tough being a Raptor test pilot
  - DefenseNews.com (March 5, 2007)
  - “When the plane was in developmental stages … pilots flying the Raptor would often have to reboot the onboard computers that controlled the jet’s high-end functions”

Other Famous Software Failures

- 1985 Therac-25 Medical Accelerator
  - Therac-25 was a radiation therapy device.
  - Race condition lead patients receiving lethal or near lethal doses of radiation
- 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 number into a 16-bit number
- 1999 Mars Climate Orbiter Crashes on Mars
  - Missing conversion of English units to metric units
- Other Failures available at:
  - http://www.sundoginteractive.com/sunblog/posts/top-ten-most-infamous-software-bugs-of-all-time/
  - Top software failures 2015/2016
    - http://www.computerworlduk.com/galleries/infrastructure/top-10-software-failures-of-2014-3599618/

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 2000 – 29 million SLOC
  - Windows XP – 45 million SLOC
  - Windows Server 2003 – 50 million SLOC
  - Mac OS X 10.4 – 86 million SLOC
  - Linux Kernel pre-4.2 – 20 million SLOC
  - Debian 7.0 – 419 SLOC
  - Table available at http://en.wikipedia.org/wiki/Source_lines_of_code

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

Components of Software Life Cycle

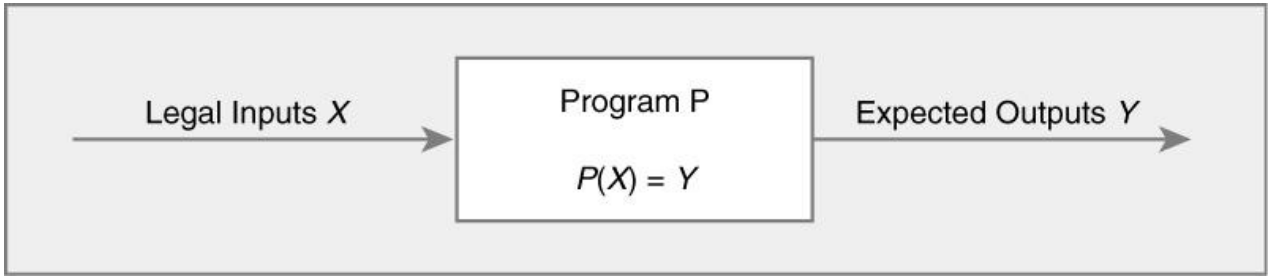
1. Problem specification
2. Program design
3. Algorithms and data structures
4. Coding and debugging
5. Testing and verification
6. Deployment
7. Documentation and support
8. Maintenance and Upgrades

Software Development

- Coding is small part of software development
- Estimated % of time
  - 35% Specification, design
  - 20% Coding, debugging
  - 30% Testing, reviewing, fixing
  - 15% Documentation, support

Problem Specification

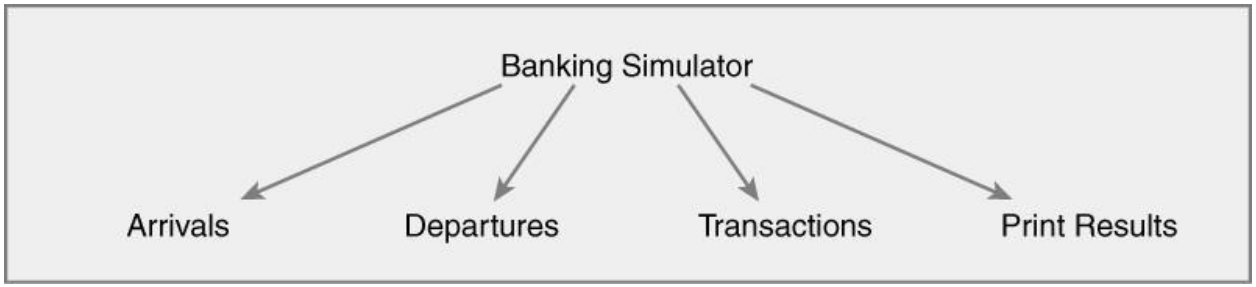
- Goal
  - Create complete, accurate, and unambiguous statement of problem to be solved
- Example
  - Specification of input & output of program



- 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



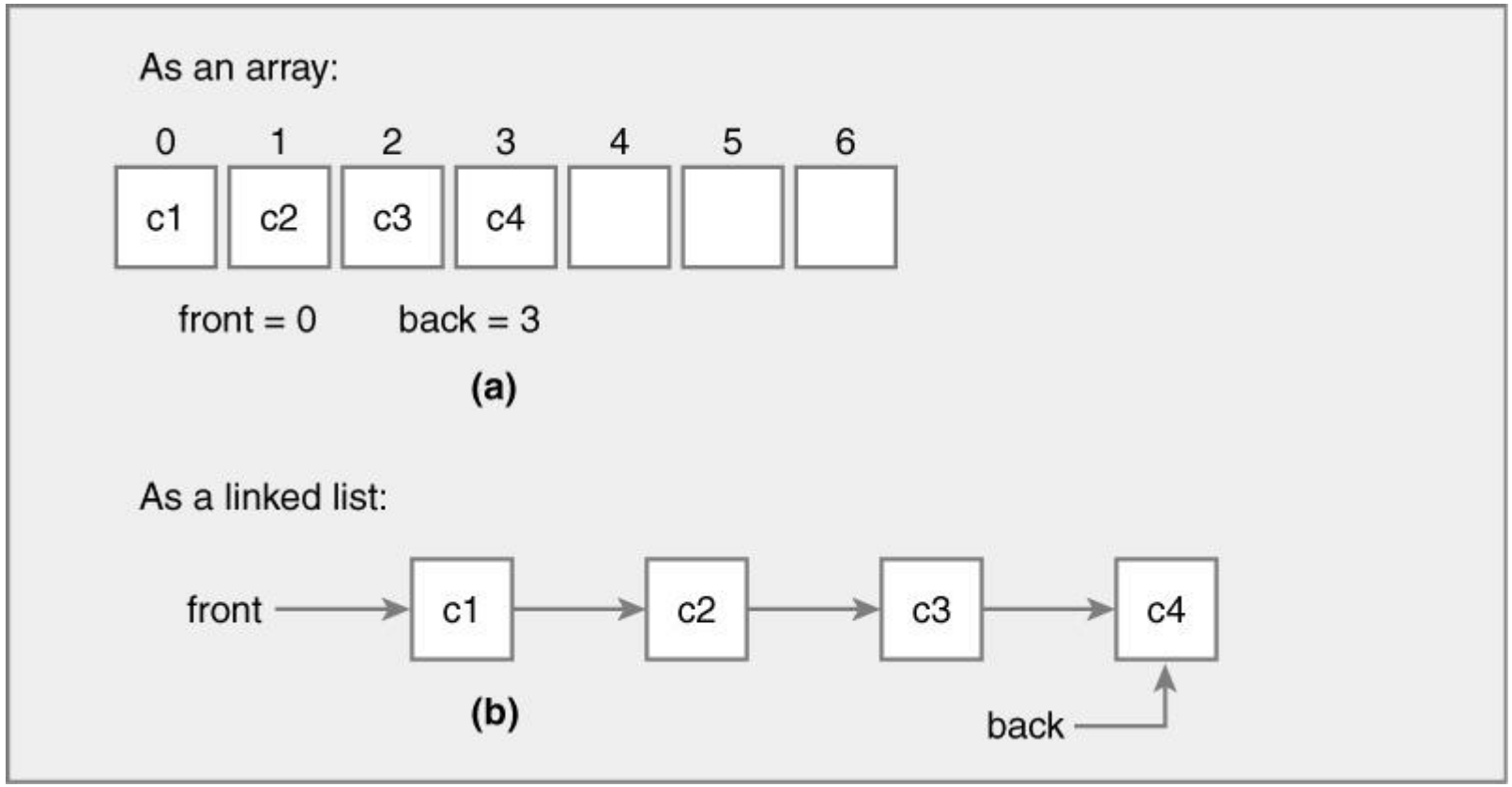
- Problems
  - Methods for decomposing problem
  - How components work together

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



Coding and Debugging

- Goal
  - Write actual code and ensure code works
- Problems
  - Choosing programming language
    - Procedural 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, but if you can prove you should, since the guarantees are so much stronger than testing
  - **Empirical testing**
    - Verify using test cases
      - Unit tests, integration tests, alpha / beta tests
    - Used in majority of cases in practice
    - You don’t know what may happen for tests you did not run

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