

Modern Software Development



Fawzi Emad
Chau-Wen Tseng

Department of Computer Science
University of Maryland, College Park

Overview

- **Motivation for modern software development**
- **Software life cycle**

Motivation

- Software development problems
- Software projects fail
- Software contributing to real failures
- Impact of software failures increasing

Software Development Problems

- Software is expensive
 - Cost per line of code increasing
(while hardware costs drop)
- Software is late (schedule overruns)
- Software cost more (cost overruns)
- Software is difficult to use
- Software is difficult to understand
- Software is missing features
- Software is too slow

Software Projects Fail

- Anywhere from 25-50% of custom software fail
- Latest example
 - Jan 13, 2005, LA Times
 - “A new FBI computer program designed to help agents share information to ward off terrorist attacks may have to be scrapped, forcing a further delay in a four-year, half-billion-dollar overhaul of its antiquated computer system... Sources said about \$100 million would be essentially lost if the FBI were to scrap the software...”

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

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

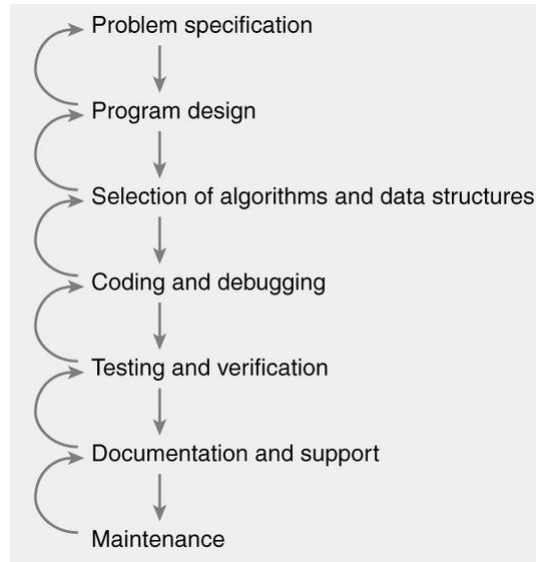
- Coding is only part of software development
- Software engineering requires
 - Preparation before writing code
 - Follow-up work after coding is complete
- Software life cycle
 - Certain essential operations needed for good software development
 - No universal agreement, just general agreement

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

“Waterfall Model” of Life Cycle

- **Simple model**
 - Proceed from one step to next
 - Return to previous step if needed
- **In reality**
 - Steps may be more integrated
 - Steps may overlap and occur at same time



Software Life Cycle

- **Waterfall model**
 - Works better for smaller projects
- **Some alternative approaches**
 - “Extreme programming”
 - Write test cases before writing code
 - “Rapid prototyping”
 - Use working prototype to refine specifications

Problem Specification

■ Goal

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

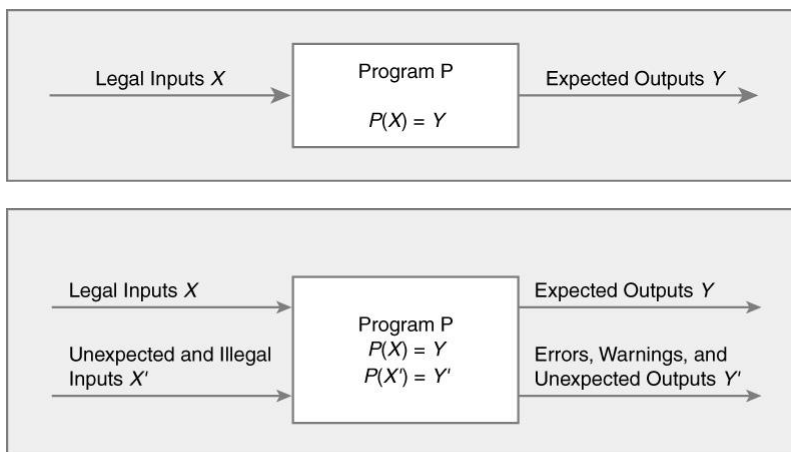
■ Problems

- Description may not be accurate
- Description may change over time
- Difficult to specify behavior for all inputs
- Natural language description is imprecise
- Formal specification languages limited and difficult to understand

Problem Specification

■ Example

- Specification of input & output for program



Problem Specification Problems

- **Description may not be accurate**
 - Problem not understood by customer
- **Description may change over time**
 - Customer changes their mind
- **Difficult to specify behavior for all inputs**
 - Usually only covers common cases
 - Hard to consider all inputs (may be impossible)
 - Example
 - Bart Miller was able to crash most UNIX utilities with randomly generated inputs

Problem Specification Problems

- **Description may be ambiguous**
 - Natural language description is imprecise
 - Why lawyers use legalese for contracts
 - Formal specification languages are limited and may be difficult to understand
 - Examples
 - Find sum of all values in N-element list L between 1 and 100
 - $\sum_{i=0}^{N-1} L_i \ni (L_i \geq 1) \wedge (L_i \leq 100)$
- **Difficult to write specifications that are both readable and precise**