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

- **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

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

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

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 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

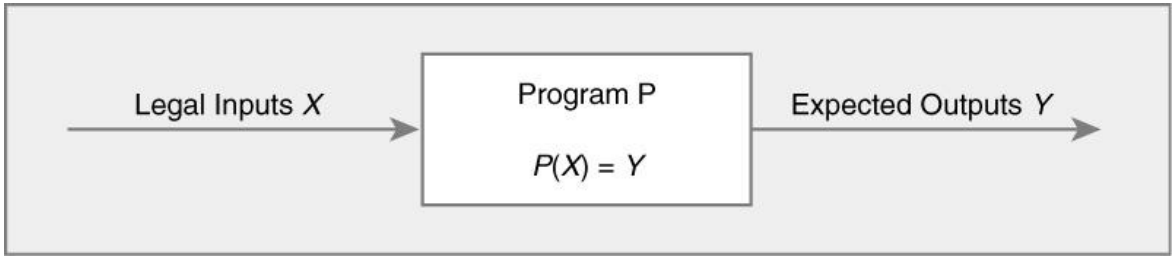
1. Problem specification
2. Program design
3. Algorithms and data structures
4. Coding and debugging
5. Testing and verification
6. Deployment
7. Documentation and support
8. Maintenance and Upgrades

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

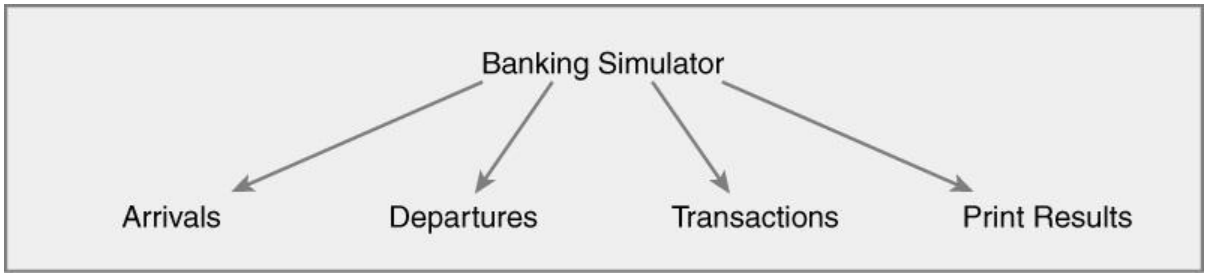
- **Goal**
  - Create complete, accurate, and unambiguous statement of problem to be solved (not as simple as it looks)
- **Example**
  - Specification of input & output of program



- **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**



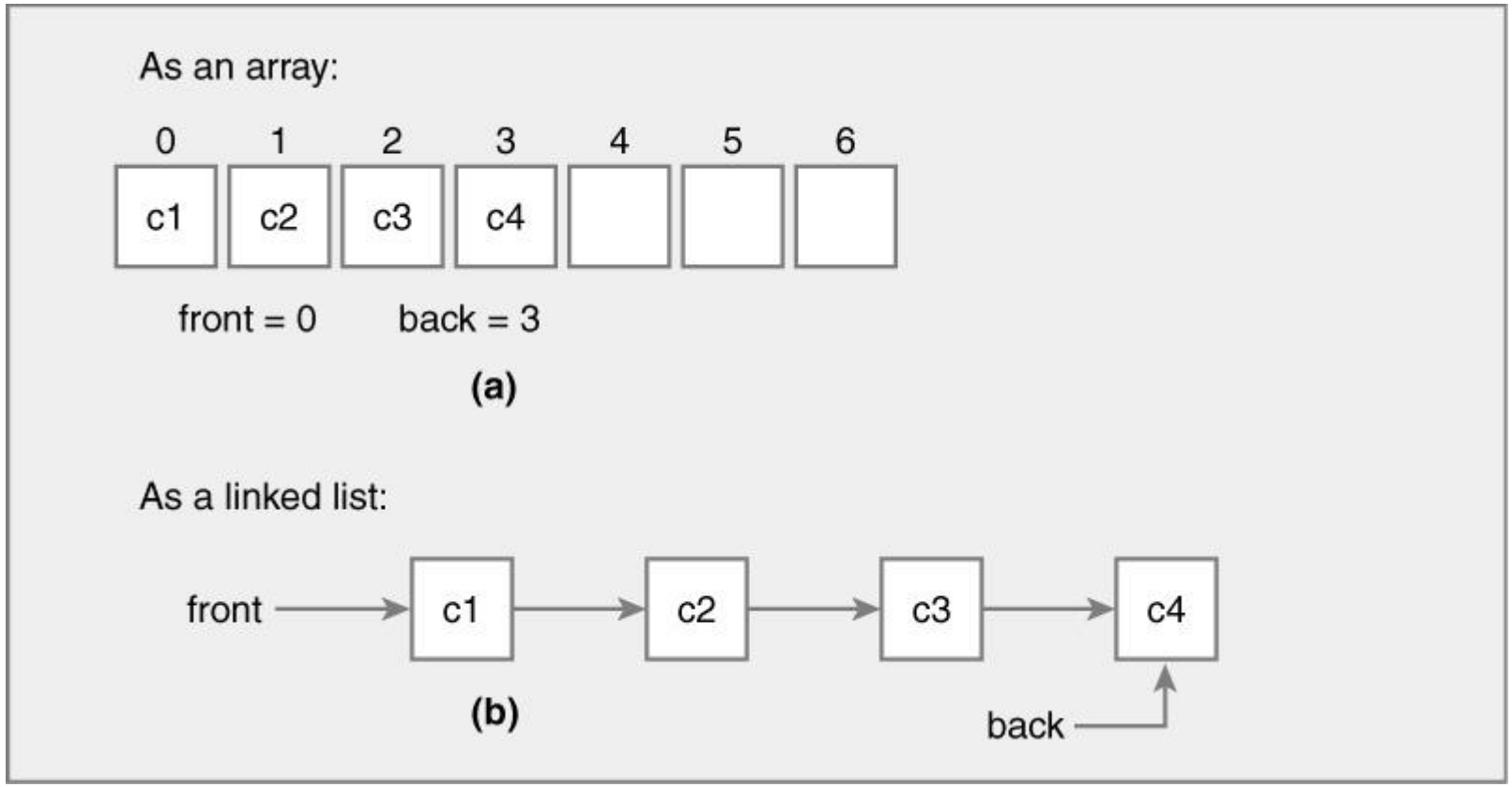
- **Problems**
  - Methods for decomposing problem
  - How components work together

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

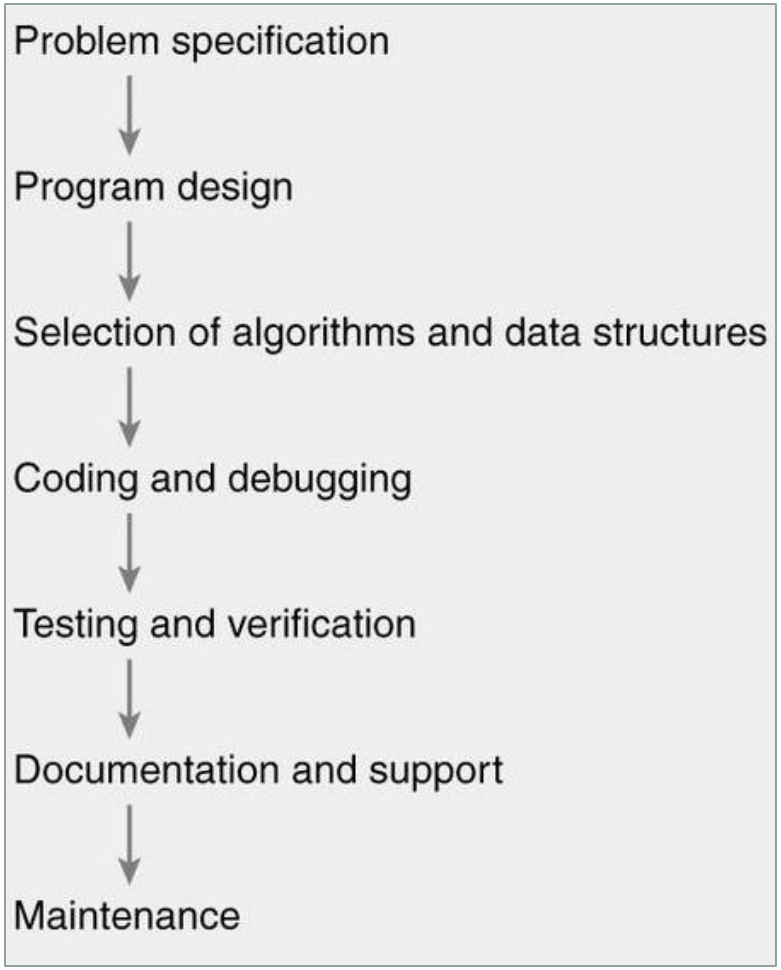
- **Goal**
  - Keep software working over time
- **Problems**
  - Fix errors
  - Improve features
  - Meet changing specification
  - Add new functionality

Software Process Models

- **Software methodology**
  - Codified set of practices
  - **Repeatable process for producing quality software**
- **Software process model**
  - Methodology for organizing software life cycle
- **Major approaches**
  - **Waterfall model**
  - **Iterative development**
    - Unified model
    - Agile software development
      - Extreme programming (XP) (prominent example)
  - **Formal methods**

Waterfall Model

- **Approach**
  - Perform steps in order
  - Begin new step only when previous step is complete
  - Result of each step flow into next step

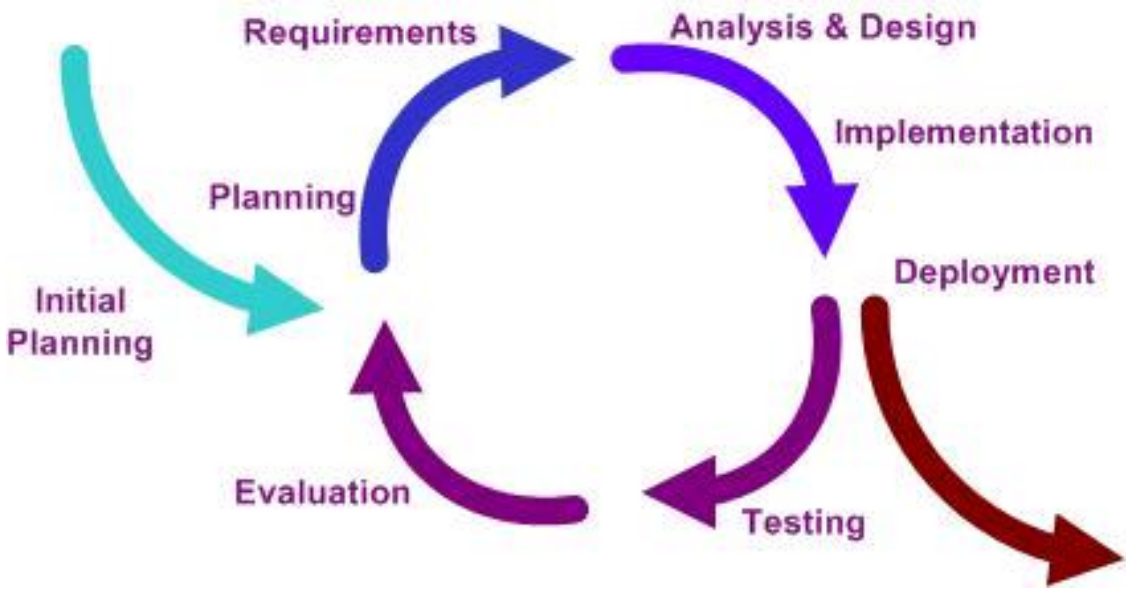


Waterfall Model

- **Advantages**
  - Simple
  - Predictable results (emphasizes predictability)
    - Software follows specifications
  - Reasonable for small projects
- **Problems**
  - **In real life**
    - May need to return to previous step
    - Steps may be more integrated
    - Steps may occur at same time
  - **Unworkable for large projects**

Iterative Software Development

- **Approach**
  - Iteratively add incremental improvements
  - Take advantage of what was learned from earlier versions of the system
  - Use working prototypes to refine specifications



Iterative Software Development

- **Goals**
  - Emphasize **adaptability** instead of **predictability**
  - Respond to changes in customer requirements
- **Examples**
  - Unified model
  - **Agile software development**
    - Extreme programming (XP)

Formal Methods

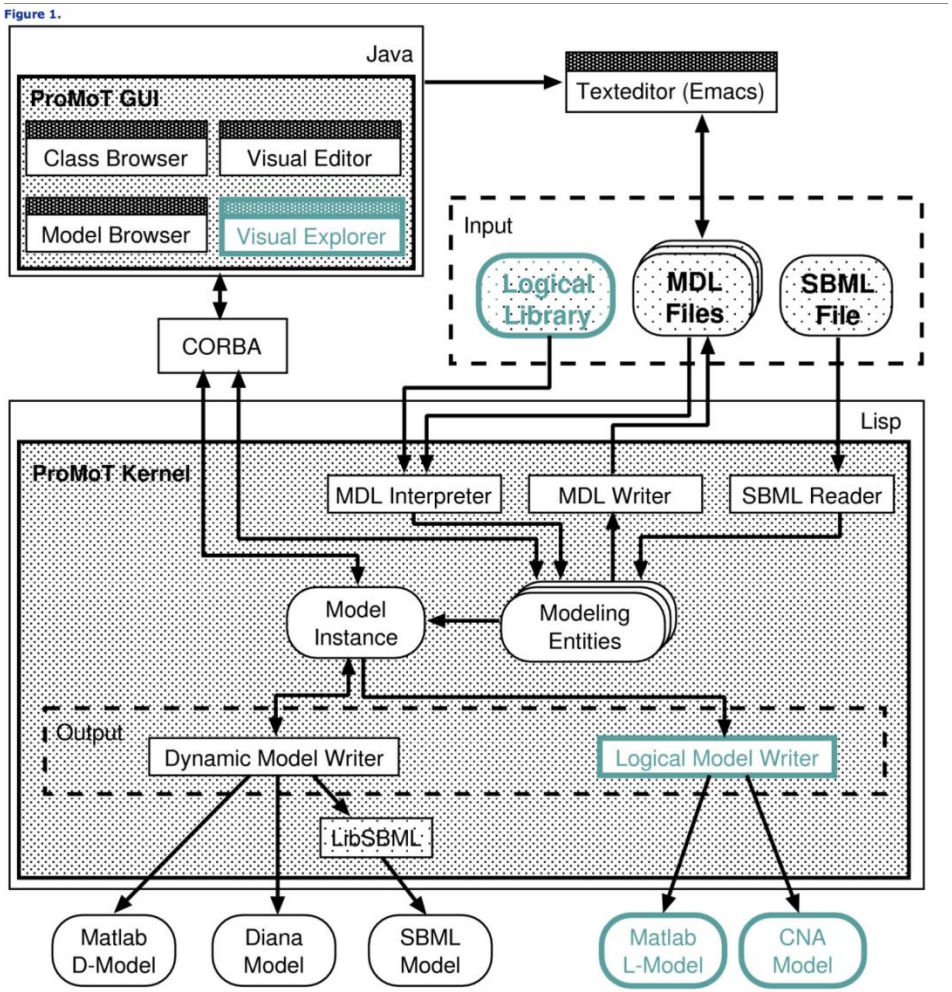
- **Mathematically-based techniques for**
  - Specification, development, and verification
  - Software and hardware systems
- **Intended for high-integrity systems**
  - Safety
  - Security

Software Architecture

- **Software Architecture**
  - Big picture of the software
  - Components generally bigger than objects or classes

Architecture of ProMoT

Just an arbitrary example of a real-world software architecture



Different Architecture Styles

- **The same system can be described using several different architecture styles**
  - **Pipes and filters**
    - What is the data, and what components do they move through
  - **Blackboard**
    - Components communicate through a shared, updatable blackboard

Compiler Architecture

- **Pipes and Filters (Passing a tree)**

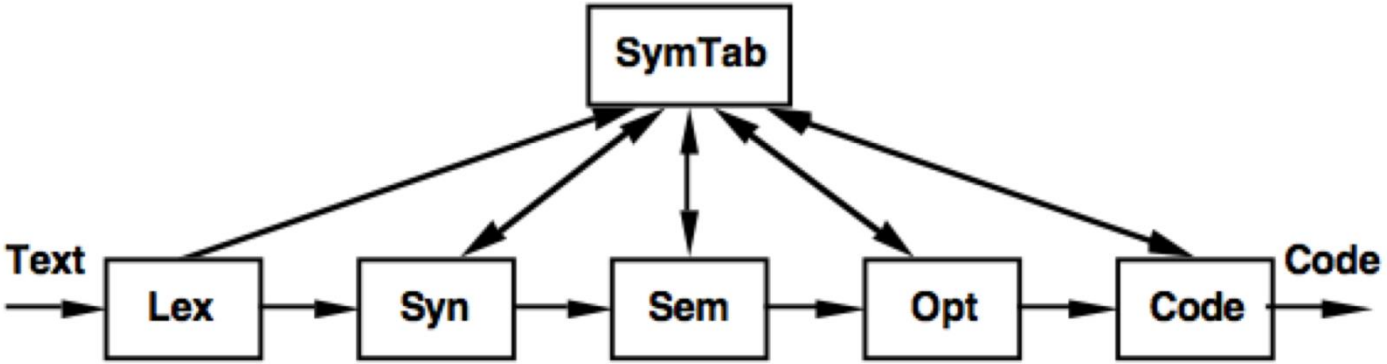


Figure 16: Traditional Compiler Model with Shared Symbol Table

Compiler Architecture, Revisited

- **Blackboard**

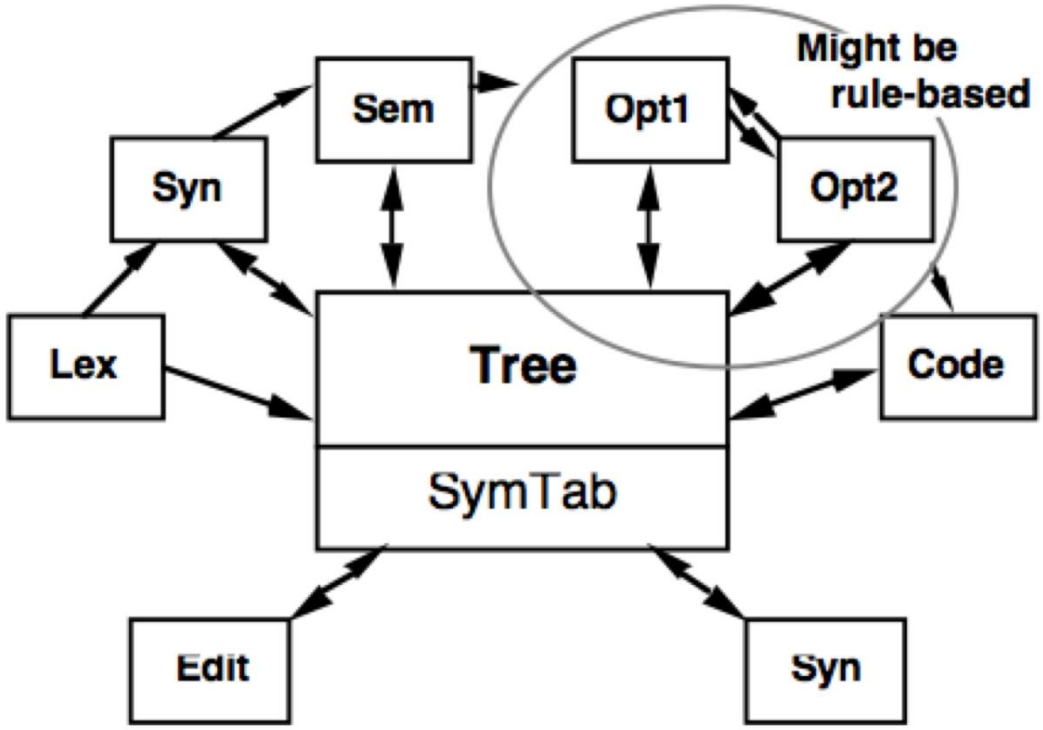


Figure 18: Canonical Compiler, Revisited