Software Life Cycle

Fawzi Emad
Chau-Wen Tseng

Department of Computer Science
University of Maryland, College Park

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
Program Design

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

Problems
- Methods for decomposing problem
  - How to divide work
  - What work to divide
  - How components work together

Design – How To Divide Work

Decomposing problem
- Break large problem into many smaller problems
  - Cannot solve large problems directly
- Divide and conquer
  1. Break problem up into simpler sub-problems
  2. Repeat for each sub-problem
  3. Stop when sub-problem can be solved easily
Design – How To Divide Work

- Functional approach
  - Treat problem as a collection of functions

- Techniques
  - Top-down design
    - Successively split problem into smaller problems
  - Bottom-up design
    - Start from small tasks and combine

Design – Decomposition Example

- Top-down design of banking simulator
Design – How To Divide Work

- **Object-oriented approach**
  - Treat problem as a collection of data **objects**
  - **Objects**
    - Entities that exist in problem
    - Contain data
    - Perform actions associated with data

**Design – Comparison Example**

- **Bank simulation**
  - **Functional programming**
    - Arrivals, departures, transactions
  - **Object-oriented programming**
    - Customers, lines, tellers, transactions
Design – Comparing Approaches

- Functional approach
  - Treat problem as a collection of functions
  - Functions perform actions
  - Think of functions as verbs

- Object-oriented approach
  - Treat problem as a collection of data objects
  - Objects are entities that exist in problem
  - Think of objects as nouns

Design – Comparing Approaches

- Advantages to object-oriented approach
  - Helps to abstract problem
    - Simpler high-level view
  - Helps to encapsulate data
    - Hides details of internals of objects
    - Centralizes and protects all accesses to data
  - Seems to scale better for larger projects

- In practice
  - Tend to use a combination of all approaches
Design – Components

Components must work together easily
Each component requires
  - Interface
    - How component is accessed
  - Pre-conditions
    - What conditions are true before invocation
  - Post-conditions
    - What conditions are true after invocation

Design – Interface & Conditions

Function positivePower()
  - Calculate $x^n$ for positive values of $x$ & $n$

Interface
  - public static float positivePower(float $x$, int $n$)

Pre-conditions
  - $x$ has positive floating point value $> 0.0$
  - $n$ has positive integer value $\geq 0$

Post-conditions
  - Returns $x^n$ if preconditions are met
  - Returns $-1.0$ otherwise
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

Example
- Implement list as array or linked list

As an array:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>c1</td>
<td>c2</td>
<td>c3</td>
<td>c4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

front = 0  back = 3

(a)

As a linked list:

front → c1 → c2 → c3 → c4

(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