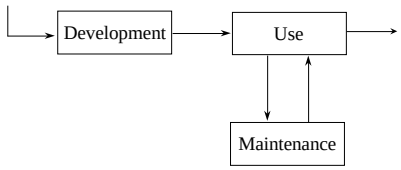
Chapter 10: Software Engineering

- The quality of the software we create is a direct result of the process we follow to develop it
- Chapter 10 focuses on:
  - software life cycle
  - development models
  - prototypes

The Program Life Cycle

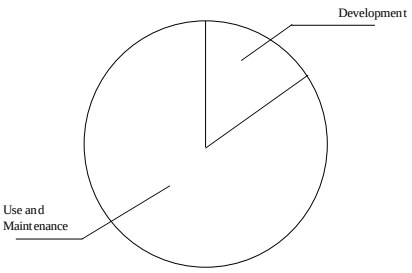
- The overall life of a program includes use and maintenance:



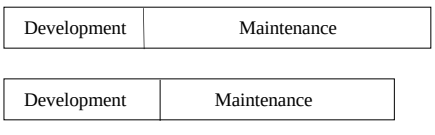
Maintenance

- Maintenance tasks include any modifications to an existing program
- It includes defect removal and enhancements
- The characteristics of a program that make it easy to develop also make it easy to maintain
- Maintenance efforts tend to far outweigh the development effort in today's software
- Small increases in effort at the development stage can greatly reduce maintenance tasks

Development vs. Maintenance



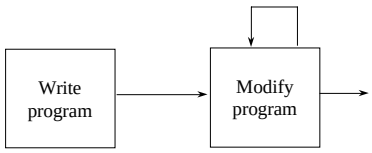
Development and Maintenance Effort



Development Process Models

- Too many programmers follow a *build-and-fix* approach
- They write a program and modify it until it is functional, without regard to system design
- Errors are haphazardly addressed as they are discovered
- It is not really a development model at all

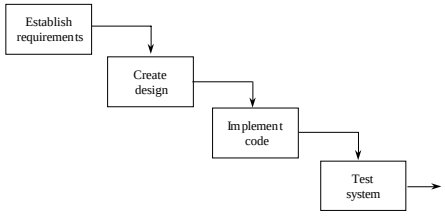
The Build-and-Fix Approach



The Waterfall Model

- Developed in the mid 1970s
- Activities that must be specifically addressed during development include:
  - Establishing clear and unambiguous requirements
  - Creating a clean design from the requirements
  - Implementing the design
  - Testing the implementation
- Originally it was proposed as a linear model, with little or no backtracking
- It is a nice goal, but unrealistic

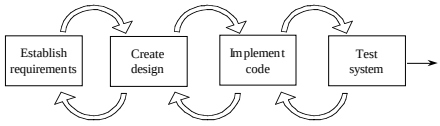
The Waterfall Model



An Iterative Process

- Allows the developer to cycle through the different development stages
- Essentially the waterfall model with backtracking
- However backtracking should not be used as a crutch
- It should be used as a technique available to the developer in order to deal with unexpected problems that may arise in later stages of development

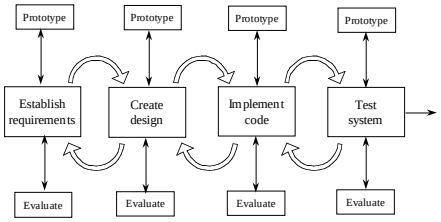
An Iterative Development Process



Prototype

- A program created to explore a particular concept
- More useful, time-effective, and cost-effective than merely acting on an assumption that may later backfire
- Usually created to communicate to the client:
  - a particular task
  - the feasibility of a requirement
  - a user interface
- A way of validating requirements

An Enhanced Iterative Model



Evaluation

- The results of each stage should be evaluated carefully prior to going on to the next stage
- Before moving on to the design, for example, the requirements should be evaluated to ensure completeness, consistency, and clarity
- A design evaluation should ensure that each requirement was adequately addressed
- Prior to testing, the implementation should be give a thorough *code walkthrough*

Testing Techniques

- Goal: to find errors
- Called *defect testing*
- A good test will uncover problems in a program
- A *test case* includes
  - a set of inputs
  - user actions or other initial conditions
  - expected output
- It is not feasible to exhaust every possible case

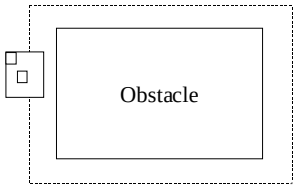
Black-Box Testing

- Mapping a set of specific inputs to a set of expected outputs
- An *equivalence category* is a collection of input sets
- Two input sets belong to the same equivalence category if there is no reason to believe that if one works, the other will not
- Therefore testing one input set essentially tests the entire category

White-Box Testing

- Also referred to as glass-box testing
- Focuses on the internal logic such as the implementation of a method
- Statement coverage guarantees that all statements in a method are executed
- *Condition coverage* guarantees that all paths through a method are executed

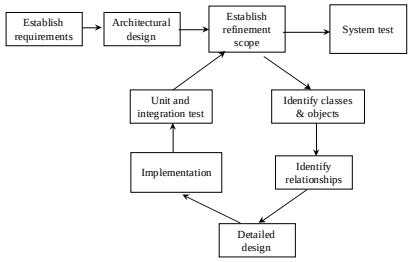
Collision Detection



An Evolutionary Development Model

- Divide the process of design into
  - *architectural design* - primary classes and interaction
  - *detailed design* - specific classes, methods, and algorithms
- Create a *refinement cycle*
- Each refinement focuses on one aspect of the system
- As each refinement is addressed, the system evolves

An Evolutionary Development Model



Refinement Cycle

- Establish refinement scope
- Define the specific nature of the next refinement
- Such as:
  - user interface
  - a particular algorithm
  - a particular requirement
- Choosing the most appropriate next refinement is important and requires experience

Refinement Cycle

- Identifying classes and objects
- The ones that relate to the current refinement
- Could overlap with other refinements
- Can often define by focusing on the roles they play in the system
- Consider reusing existing classes

Refinement Cycle

- Identifying relationships
- Inheritance (is-a) relationships
- The *uses relationship* establishes another kind of bond between classes
- Class A uses class B in some way
- Can express cardinality
- Example: A Car has (uses) four wheels

Refinement Cycle

- Detailed design, implementation and test
- Design of specific methods and their translation into code
- A *unit test* focuses on one particular component, such as a method or class
- An *integration test* focuses on the interaction between components