



Chapter 2

Principles of Programming & Software Engineering

Problem Solving and Software Engineering

- Coding without a solution design increases debugging time
- A team of programmers is needed for a large software development project
- Teamwork requires:
 - An overall plan
 - Organization
 - Communication
- Software engineering
 - Provides techniques to facilitate the development of computer programs

What is Problem Solving?

- Problem solving
 - The process of taking the statement of a problem and developing a computer program that solves that problem
- A solution consists of:
 - Algorithms
 - Algorithm: a step-by-step specification of a method to solve a problem within a finite amount of time
 - Ways to store data

The Life Cycle of Software

- The life cycle of a software
 - A lengthy and continuing process
 - Required for the development of good software
 - Programmer can move from any phase of the cycle to any other phase

The Life Cycle of Software

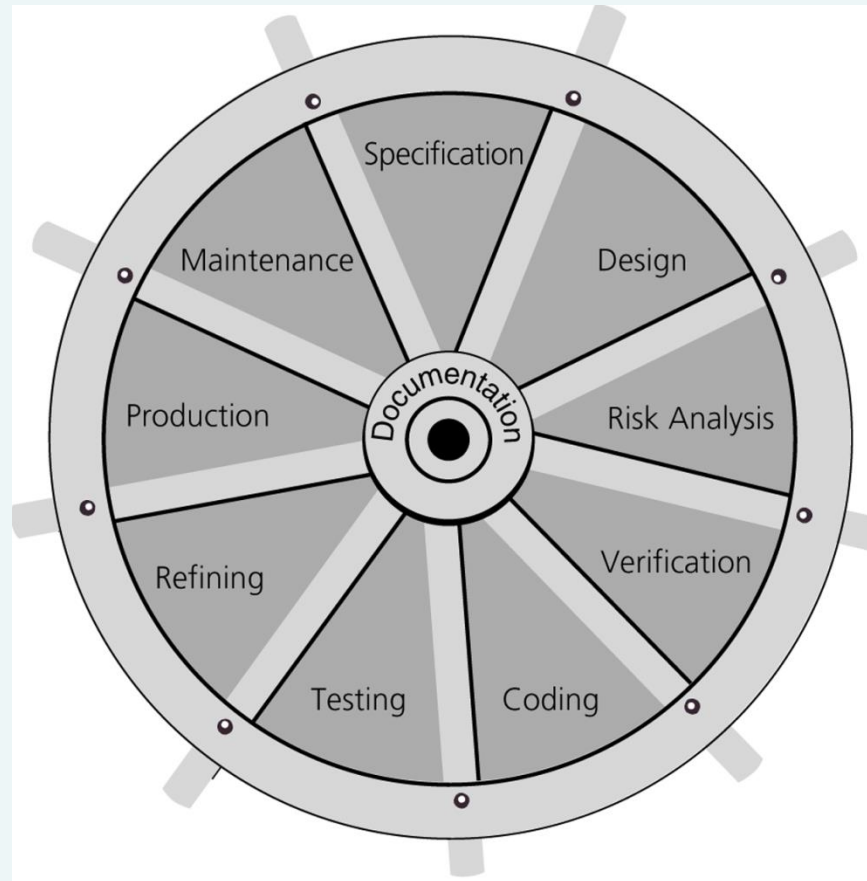


Figure 2-1

The life cycle of software as a water wheel that can rotate from one phase to any other phase

The Life Cycle of Software

- Phase 1: Specification
 - Aspects of the problem which must be specified:
 - What is the input data?
 - What data is valid and what data is invalid?
 - Who will use the software, and what user interface should be used?
 - What error detection and error messages are desirable?
 - What assumptions are possible?
 - Are there special cases?
 - What is the form of the output?
 - What documentation is necessary?
 - What enhancements to the program are likely in the future?

The Life Cycle of Software

- Phase 1: Specification (Continued)
 - Prototype program
 - A program that simulates the behavior of portions of the desired software product
- Phase 2: Design
 - Includes:
 - Dividing the program into modules
 - Specifying the purpose of each module
 - Specifying the data flow among modules

The Life Cycle of Software

- Phase 2: Design (Continued)
 - Modules
 - Self-contained units of code
 - Should be designed to be:
 - Loosely coupled
 - Highly cohesive
 - Interfaces
 - Communication mechanisms among modules

The Life Cycle of Software

- Phase 2: Design (Continued)
 - Specifications of a method
 - A contract between the method and the module that calls it
 - Should not commit the method to a particular way of performing its task
 - Include the method's:
 - Precondition
 - » A statement of the conditions that must exist at the beginning of the method
 - Postcondition
 - » A statement of the conditions at the end of the method

The Life Cycle of Software

- Phase 3: Risk Analysis
 - Building software entails risks
 - Techniques exist to identify, assess, and manage the risks of creating a software product
- Phase 4: Verification
 - Formal methods can be used to prove that an algorithm is correct
 - Assertion
 - A statement about a particular condition at a certain point in an algorithm
 - Java's assert statement: `assert booleanExpression;`

The Life Cycle of Software

- Phase 4: Verification (Continued)
 - Invariant
 - A condition that is always true at a particular point in an algorithm
 - Loop invariant
 - A condition that is true before and after each execution of an algorithm's loop
 - Can be used to detect errors before coding is started

The Life Cycle of Software

- Phase 4: Verification (Continued)
 - The invariant for a correct loop is true:
 - Initially, after any initialization steps, but before the loop begins execution
 - Before every iteration of the loop
 - After every iteration of the loop
 - After the loop terminates
- Phase 5: Coding
 - Involves:
 - Translating the design into a particular programming language
 - Removing syntax errors

The Life Cycle of Software

- Phase 6: Testing
 - Involves:
 - Removing the logical errors
 - Test data should include:
 - Valid data that leads to a known result
 - Invalid data
 - Random data
 - Actual data

The Life Cycle of Software

- Phase 7: Refining the Solution
 - During phases 1 through 6
 - A working program is developed under simplifying assumptions
 - During phase 7
 - Refining sophistication is added, such as:
 - More sophisticated input and output routines
 - Additional features
 - More error checks

The Life Cycle of Software

- Phase 8: Production
 - Involves:
 - Distribution to the intended users
 - Use by the users
- Phase 9: Maintenance
 - Involves
 - Correcting user-detected errors
 - Adding more features
 - Modifying existing portions to suit the users better

What is a Good Solution?

- A solution is good if:
 - The total cost it incurs over all phases of its life cycle is minimal
- The cost of a solution includes:
 - Computer resources that the program consumes
 - Difficulties encountered by those who use the program
 - Consequences of a program that does not behave correctly
- Programs must be well structured and documented
- Efficiency is only one aspect of a solution's cost

Achieving an Object-Oriented Design: Abstraction and Information Hiding

- A modular solution to a problem should specify what to do, not how to do it
- Abstraction
 - Separates the purpose of a module from its implementation
- Procedural abstraction
 - Separates the purpose of a method from its implementation

Abstraction and Information Hiding

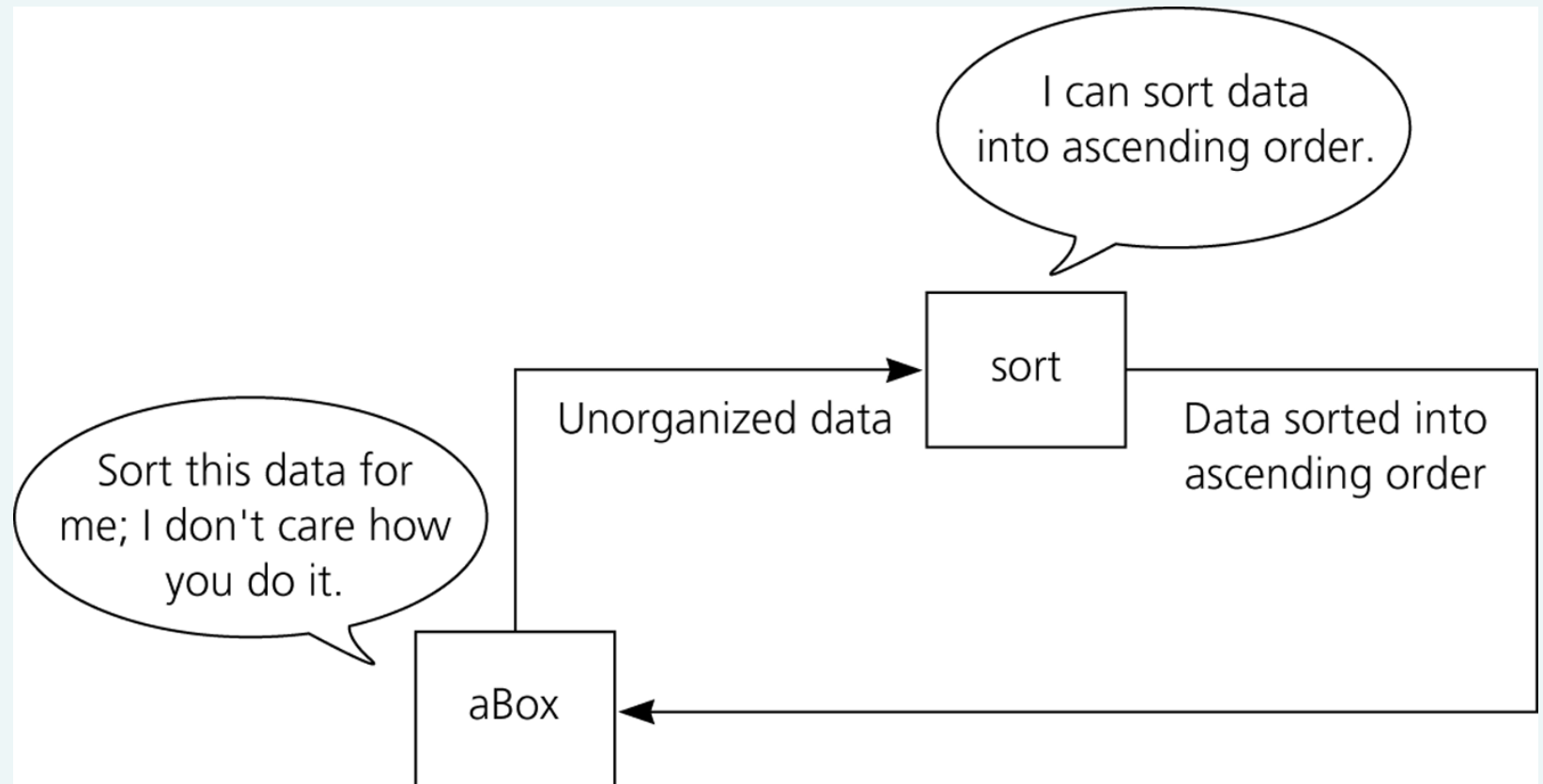


Figure 2-2

The details of the sorting algorithm are hidden from other parts of the solution.

Abstraction and Information Hiding

- Data abstraction
 - Focuses of the operations of data, not on the implementation of the operations
 - Abstract data type (ADT)
 - A collection of data and a set of operations on the data
 - An ADT's operations can be used without knowing how the operations are implemented, if:
 - the operations' specifications are known
 - Data structure
 - A construct that can be defined within a programming language to store a collection of data

Abstraction and Information Hiding

- Public view of a module
 - Described by its specifications
- Private view of a module
 - Consists of details which should not be described by the specifications
- Principle of information hiding
 - Hide details within a module
 - Ensure that no other module can tamper with these hidden details

Object-Oriented Design

- Object-oriented approach to modularity
 - Produces a collection of objects that have behaviors
- Object
 - An instance of a class
 - Combines data and operations on that data
- Encapsulation
 - A technique that hides inner details
 - Methods encapsulate actions
 - Objects encapsulate data as well as actions

Object-Oriented Design

- Principles of object-oriented programming (OOP)
 - Encapsulation
 - Objects combine data and operations
 - Inheritance
 - Classes can inherit properties from other classes
 - Polymorphism
 - Objects can determine appropriate operations at execution time

Functional Decomposition

- Object-oriented design (OOD)
 - Produces modular solutions for problems that primarily involve data
 - Identifies objects by focusing on the nouns in the problem statement
- Functional Decomposition (FD)
 - Produces modular solutions for problems in which the emphasis is on the algorithms
 - Identifies actions by focusing on the verbs in the problem statement
 - A task is addressed at successively lower levels of detail

Functional Decomposition

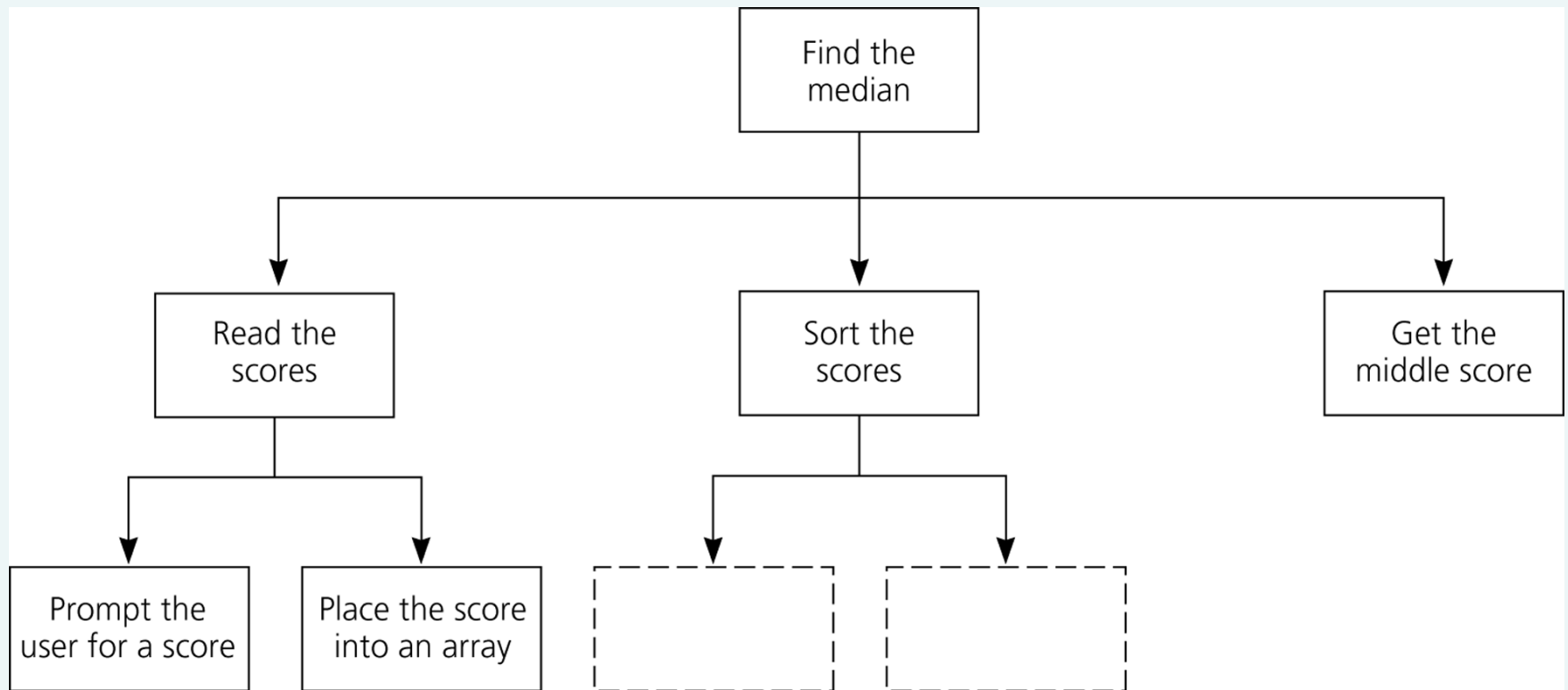


Figure 2-4

A structure chart showing the hierarchy of modules

General Design Guidelines

- Use OOD and FD together
- Use OOD for problems that primarily involve data
- Use FD to design algorithms for an object's operations
- Consider FD to design solutions to problems that emphasize algorithms over data
- Focus on what, not how, when designing both ADTs and algorithms
- Consider incorporating previously written software components into your design

Modeling Object-Oriented Designs Using IML

- Unified Modeling Language (UML): language to express OO designs
- Class diagrams include name, data, operations
- Text-based notation: more complete specifications

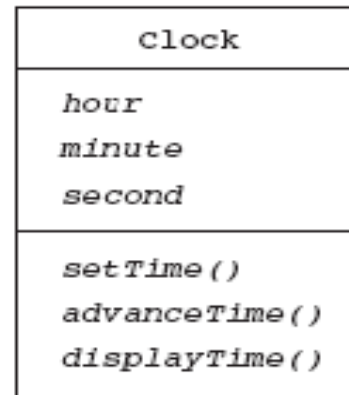


FIGURE 2-5

UML diagram for the class `Clock`

A Summary of Key Issues in Programming

- Modularity
 - Favorable impact on program development
- Modifiability
 - Use of methods and named constants
- Ease of Use
 - Considerations for the user interface
 - Program should prompt the user for input
 - A program should always echo its input
 - The output should be well labeled and easy to read

A Summary of Key Issues in Programming

- Fail-Safe Programming
 - Fail-safe program
 - A program that will perform reasonably no matter how anyone uses it
 - Types of errors:
 - Errors in input data
 - Errors in the program logic

A Summary of Key Issues in Programming

- Style
 - Five issues of style
 - Extensive use of methods
 - Use of private data fields
 - Error handling
 - Readability
 - Documentation

A Summary of Key Issues in Programming

- Debugging
 - Programmer must systematically check a program's logic to determine where an error occurs
 - Tools to use while debugging:
 - Watches
 - Breakpoints
 - `System.out.println` statements
 - Dump methods

Summary

- Software engineering studies ways to facilitate the development of computer programs
- Software life cycle consists of:
 - Specifying the problem
 - Designing the algorithm
 - Analyzing the risks
 - Verifying the algorithm
 - Coding the programs
 - Testing the programs
 - Refining the solution
 - Using the solution
 - Maintaining the software

Summary

- A loop invariant is a property of an algorithm that is true before and after each iteration of a loop
- An evaluation of the quality of a solution must take into consideration
 - The solution's correctness
 - The solution's efficiency
 - The time that went into the development of the solution
 - The solution's ease of use
 - The cost of modifying and expanding the solution

Summary

- A combination of object-oriented and functional decomposition techniques will lead to a modular solution
- The final solution should be as easy to modify as possible
- A method should be as independent as possible and perform one well-defined task
- A method should always include an initial comment that states its purpose, its precondition, and its postcondition

Summary

- A program should be as fail-safe as possible
- Effective use of available diagnostic aids is one of the keys to debugging
- To make it easier to examine the contents of complex data structures while debugging, dump methods that display the contents of the data structures should be used