

**Programming Documentation Standards
(Object-Oriented version)
Revised Fall 2006
by
Jim Etheredge & Frank Ducrest**

A. Introduction

A.1 Purpose of Documentation Standards

Documentation standards describe a standard documentation format for programming work done in computer science courses. The intent is for students to become familiar with standard documentation techniques, and to improve their system development skills through increased attention to detail at all phases of the software development process.

A.2 Overview of Documentation

The materials comprising the documentation cover the four phases of problem solution - analysis, design, coding and testing. In the analysis phase, the problem is defined and the requirements documentation is produced. In the design phase, a software solution to the problem is planned, organized and detailed to produce the design documentation. During the coding phase the design is converted to program code, which, together with appropriate comments, becomes the implementation documentation. In the testing phase, the program is run to test whether it accurately produces the results specified in the problem definition. Difficulties encountered in the test runs are removed through debugging and making needed corrections. Test data and final test results are recorded in the verification and validation documentation.

NOTE: All documentation must be typed, formatted correctly, and carefully checked for both spelling and grammar.

B. Content Outline of the Documentation Components

I. Requirements Documentation

1. Description of the Problem
2. Input Information
3. Output Information
4. User Interface Information

II. Design Documentation

1. System Architecture Description
2. Object Information
3. System Driver Information
4. Diagrams

III. Implementation Documentation

1. Program code

IV. Verification & Validation Documentation

1. Test data
2. Test results

C. Content Details of the Documentation Components

The following sections explain the content of each of the four major components of the documentation standards.

I. Requirements Documentation

The purpose of this section of the documentation is to define the problem with sufficient detail so that the solution can be planned.

I.1 Description of the Problem

Name: Give a short title.

Problem Statement: Tell what needs to be done. This should be brief (1 or 2 sentences) and provide a high level description of the problem to be solved.

Problem Description: Give a complete and detailed specification of the problem. State any assumptions you have made regarding the problem. This specification is intended to provide a real world description of the problem, its input, its output, and its processing. *No implementation-specific details should be included.*

I.2 Input Information

Input Streams: (Repeat the following for each input stream.)

Name: Give the name of the input stream.

Description: How is it used? What is its purpose?

I.3 Output Information

Output Streams: (Repeat the following for each output stream.)

Name: Give the name of the output stream.

Description: How is it used? What is its purpose?

I.4 User Interface Information

I.4.1 Description:

The user interface, which determines how the user will interact with the program, should be described in enough detail to make its functionality clear.

I.4.2 Sample:

Include illustrations of screens and/or dialogues when appropriate.

II. Design Documentation

The purpose of this section of the documentation is to describe a plan for the solution of the problem using an object-oriented design method.

The solution to a problem will usually consist of a software system that is comprised of a system driver/coordinator and a set of interacting objects. The system driver consists of the algorithm that orchestrates the usage of the objects in the system.

Adhere to recommended good programming practices, such as:

- Make certain that components are loosely coupled; i.e, avoid global objects and variables.
- Implement operations using the most appropriate type of subroutines; i.e.;procedures or functions.
- Pass parameters by value or by reference as appropriate.
- Make certain that components are highly cohesive.

II.1 System Architecture Description

This section includes the system driver and the objects that comprise the system. For each system component briefly describe its role in the overall system.

II.2 Information about the Objects

For each class specification, include the following text and information:

Class Information

Name: Name the class.

Description: Briefly describe its purpose.

Base Class: Identify the base class, if appropriate

Class Attributes (data members) (Repeat for each attribute.)

Name: Name the attribute.

Description: Briefly describe its function or purpose.

Type: Indicate its data type or class.

Range of acceptable values: Identify the acceptable range of values.

(Note: Usually the attributes of an object are placed in the private view of the class specification.)

Class Operations (member functions)(Repeat for each operation.)

Name: Name the operation.

Description: Briefly describe the task it performs.

Precondition(s): Give input assertion(s) describing the truths that the operation expects at the moment the caller invokes the function

Postcondition(s): Give output assertion(s) describing the state of the computation at the moment the function terminates.

Prototype: Give the operation/function prototype.

Visibility: public, protected, or private

Objects

List all objects of the given class.

II.3 Information about the System Driver

In an object-oriented design the system driver is the coordinating algorithm of the software product. The driver may consist of several subroutines. Often the driver is only responsible for processing user commands and then delegating tasks to objects.

This section describes, in a readable and modular form, how the system driver controls the software. The description of the system driver must include the detailed logic of the driver, including flow of control. This description must be presented in pseudocode.

II.4 Design Diagrams

The design effort will be summarized in three diagrams:

Object Interaction Diagram

Show a diagram that illustrates the calling interaction of the system. This diagram consists of an ellipse for the system driver, a box for each class, and directed lines from clients to servers. Each box contains the name of the class and its operations.

Inheritance Diagram

Show an inheritance diagram for each class, derived or not. One diagram will be needed for each inheritance scheme.

III. Implementation Documentation

The purpose of this documentation is to present a well-engineered version of the program, along with information needed to clarify how it has been encoded.

III.1 Program Code (Source Code)

The source listing is included here. The following describes the required organization and internal documentation for the program.

Physical Organization of System Components: It is expected that different components of the system architecture will appear in different compilation units, provided that the implementation language/environment supports this type of organization. Information is to be shared among components through the inclusion of header files (when feasible).

Comments: You should import the program design information from your design documentation and build your code around the design. Comments should be used when the encoding or translation of a design is not obvious.

- System documentation (should appear in your system driver component):
Programmer information -- (import from the beginning of written documentation)
Problem statement -- (import from Requirements Doc. part 1)
Problem specification -- (import from Requirements Doc. part 1)
System architecture description -- (import from Design Doc.)
- Component documentation: Each component should include as introductory documentation, the name of the source file it is located in, and the general description of its role in the system from the System Architecture Description (import from Design Doc, section II.1).
- Class and object documentation: For each class import from Design Doc., the relevant information from section II.2.

Style: Programming style refers to those conventions that enhance the readability of programs. Some of those conventions include:

- Prettyprinting: Use indentation and skipped lines so that the visual appearance of a program listing mirrors its logical structure. (Be consistent with your indentation increments!). Declare only one data item per line. For each declared data item include a brief comment documenting its purpose. Write only one program statement per line.
- Meaningful identifier names: Well-chosen identifiers significantly enhance readability, and as such are considered a significant element of internal program documentation. Identifiers should
 - be meaningful; avoid cuteness, single-letter identifiers, meaningless abbreviations, identifiers that too closely resemble one another. (ex. HT is not a meaningful variable name for a hash table.)
 - be accurate (ex. COUNT is not the best name for an integer that indexes an array; object names should indicate entities, operation names should indicate actions)
 - observe standards for abbreviations, prefixes, and suffixes.
- Organizational consistency: Be systematic in grouping and ordering of declarations. For example, declared variables might be grouped by similar role, or listed alphabetically, but should not appear in random order. The same applies for all other declarations, such as subprogram declarations.

IV. Verification and Validation Documentation

The purpose of this documentation is to demonstrate the operation of the program, describe how it is run on the machine, and present evidence of program verification and validation. (This information should be divided into the following three parts:)

IV.1 Test Plan

Include a list of input data sets that thoroughly test the logic of the program and demonstrate that the program satisfies its requirements. Explain what requirement of the problem will be exercised by each set of test data. (The test plan should be prepared at the time the Requirements Documentation is written.)

IV.2 Test Results

Include a script file showing the results of running the program with the test data set. The output listing should be marked, if necessary, so that corresponding input can be identified for each output. That is, if the test data or program logic being exercised by this test is not obvious, mark the output listing with this information.