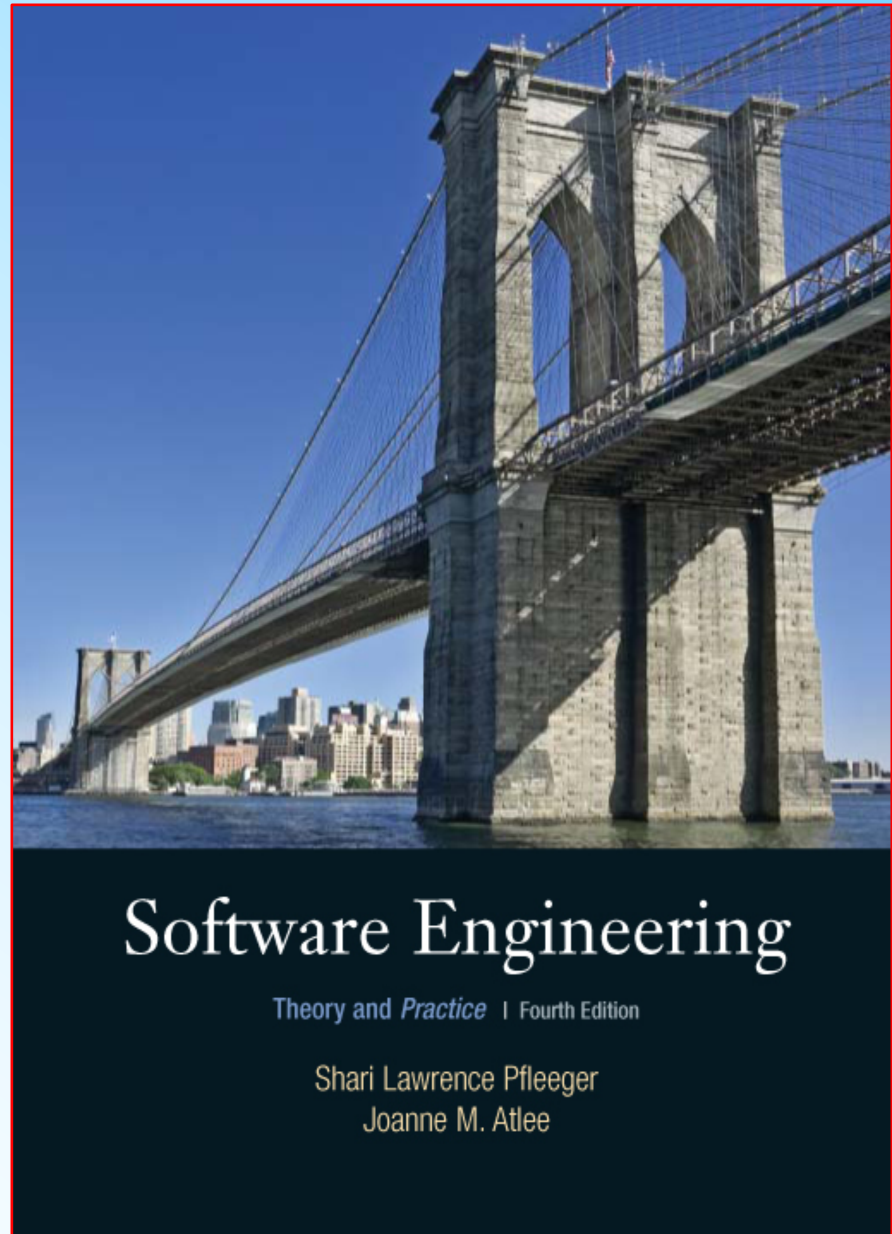
Chapter 7

Writing the Programs

Shari L. Pfleeger
Joann M. Atlee

4th Edition



Contents

7.1 Programming Standards and Procedures
7.2 Programming Guidelines
7.3 Documentation
7.4 The Programming Process
7.5 Information System Example
7.6 Real Time Example
7.7 What this Chapter Means for You

Chapter 7 Objectives

- Standards for programming
- Guidelines for reuse
- Using design to frame the code
- Internal and external documentation

7.1 Programming Standards and Procedures

- Standards for you
  - methods of code documentation
- Standards for others
  - Integrators, maintainers, testers
  - Prologue documentation
  - Automated tools to identify dependencies
- Matching design with implementation
  - Low coupling, high cohesion, well-defined interfaces

7.1 Programming Standards and Procedures
Sidebar 7.1 Programming Standards at Microsoft

- Allow flexibility to be creative and evolve product's details in stages
- Flexibility does not preclude standards

7.2 Programming Guidelines
Control Structures

- Make the code easy to read
- Build the program from modular blocks
- Make the code not too specific, and not too general
- Use parameter names and comments to exhibit coupling among components
- Make the dependency among components visible

7.2 Programming Guidelines
Example of Control Structures

- Control skips around among the program's statements

```
        benefit = minimum;
        if (age < 75) goto A;
        benefit = maximum;
        goto C;
        if (AGE < 65) goto B;
        if (AGE < 55) goto C;
A:      if (AGE < 65) goto B;
        benefit = benefit * 1.5 + bonus;
        goto C;
B:      if (age < 55) goto C;
        benefit = benefit * 1.5;
C:      next statement
```

- Rearrange the code

```
        if (age < 55) benefit = minimum;
        elseif (AGE < 65) benefit = minimum + bonus;
        elseif (AGE < 75) benefit = minimum * 1.5 + bonus;
        else benefit = maximum;
```

7.2 Programming Guidelines
Algorithms

- Common objective and concern: performance (speed)
- Efficiency may have hidden costs
  - cost to write the code faster
  - cost to test the code
  - cost to understand the code
  - cost to modify the code

7.2 Programming Guidelines
Data Structures

- Several techniques that used the structure of data to organize the program
  - keeping the program simple
  - using a data structure to determine a program structure

7.2 Programming Guidelines
Keep the Program Simple
Example: Determining Federal Income Tax

1. For the first $10,000 of income, the tax is 10%
2. For the next $10,000 of income above $10,000, the tax is 12 percent
3. For the next $10,000 of income above $20,000, the tax is 15 percent
4. For the next $10,000 of income above $30,000, the tax is 18 percent
5. For any income above $40,000, the tax is 20 percent

```
tax = 0.
if (taxable_income == 0) goto EXIT;
if (taxable_income > 10000) tax = tax + 1000;
else{
        tax = tax + .10*taxable_income;
        goto EXIT;
}
if (taxable_income > 20000) tax = tax + 1200;
else{
        tax = tax + .12*(taxable_income-10000):
        goto EXIT;
}
if (taxable_income > 30000) tax = tax + 1500;
else{
        tax = tax + .15*(taxable_income-20000);
        goto EXIT;
}
if (taxable_income < 40000){
        tax = tax + .18*(taxable_income-30000);
        goto EXIT;
}
else
        tax = tax + 1800. + .20*(taxable_income-40000);
EXIT;
```

7.2 Programming Guidelines
Keep the Program Simple Example (continued)

- Define a tax table for each “bracket” of tax liability

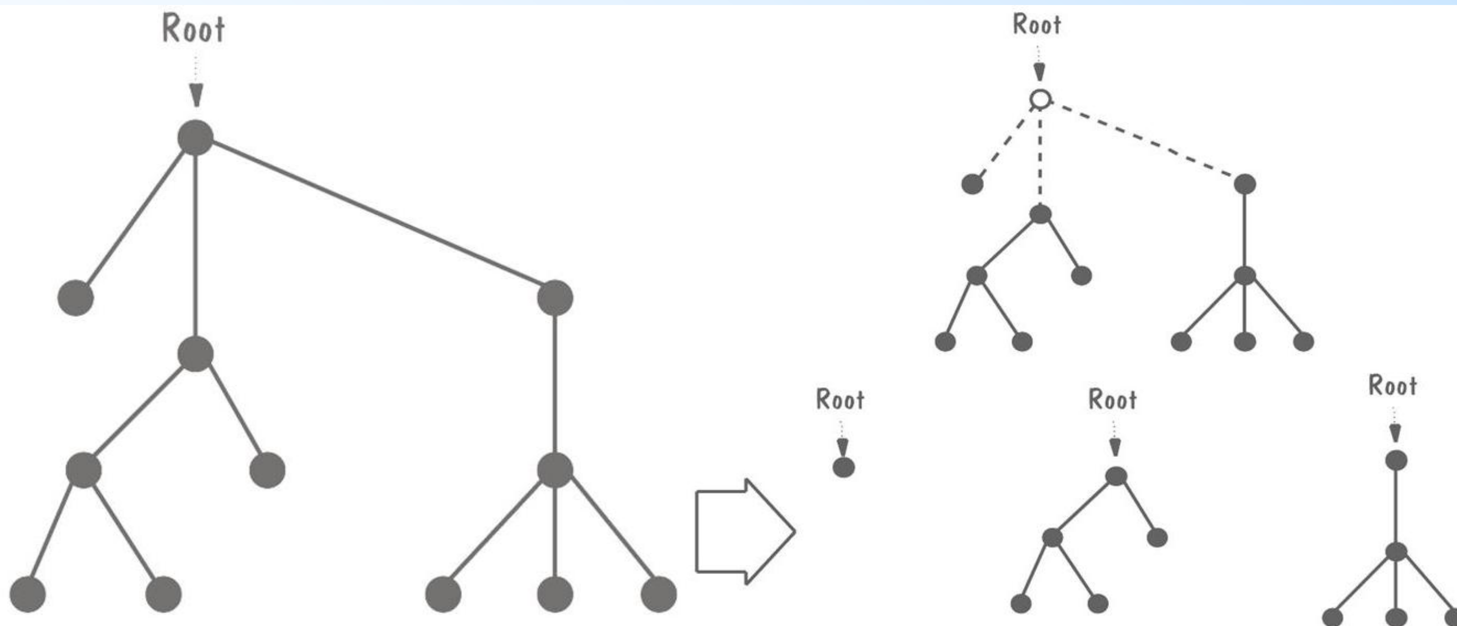
Bracket	Base	Percent
0	0	10
10,000	1000	12
20,000	2200	15
30,000	3700	18
40,000	55000	20

- Simplified algorithm

```
for (int i=2; level=1; i <= 5; i++)
        if (taxable_icome > bracket[i])
               level = level + 1;
tax= base[level]+percent[level] * (taxable_income - bracket[level]);
```

7.2 Programming Guidelines
Data Structures Example: Rooted Tree

- Recursive data structure
- Graph composed of nodes and lines
  - Exactly one node as root
  - If the lines emanating from the root are erased, the resulting graph is a rooted tree



7.2 Programming Guidelines
General Guidelines to Preserve Quality

- Localize input and output
- Employ pseudocode
- Revise and rewrite, rather than patch
- Reuse
  - Producer reuse: create components designed to be reused in future applications
  - Consumer reuse: reuse components initially developed for other projects

7.2 Programming Guidelines
Example of Pseudocode

- The design for a component of a text processing system states

```
COMPONENT PARSE_LINE
    Read nest eighty characters.
        IF this is a continuation of the previous line,
            Call CONTINUE
        ELSE determine command type
        ENDIF
    CASE of COMMAND_TYPE
        COMMAND_TYPE is paragraph: Call PARAGRAPH
        COMMAND_TYPE is indent : Call INDENT
        COMMAND_TYPE is skip line: Call SKIP_LINE
        COMMAND_TYPE is margin : Call MARGIN
        COMMAND_TYPE is new page : Call PAGE
        COMMAND_TYPE is double space : Call DOUBLE_SPACE
        COMMAND_TYPE is single space : Call SINGLE_SPACE
        COMMAND_TYPE is break : Call BREAK
        COMMAND_TYPE is anything else: Call ERROR
    ENDCASE
```

7.2 Programming Guidelines
Example of Pseudocode (continued)

- Intermediate pseudocode

PARAGRAPH:

Break line, flush line buffer. Advance one line between paragraph. If fewer than 2 line left on page, eject. Set line pointer to paragraph indent.

INDENT:

Break line, flush line buffer. Get indent parameter. Set line pointer to indent parameter, set left margin to indent.

SKIP_LINE:

Break line, flush line buffer. Get line parameter. Advance (parameter) lines or eject if not enough space left on current page.

MARGIN:

Break line, flush line buffer. Get margin parameter. Set line pointer to left margin. Set right margin to margin.

PAGE:

Break line, flush line buffer. Eject page. Set line pointer to left margin

SOUBLE_SPACE:

Set interline space to 2.

SINGLE_SPACE:

Set interline space to 1

BREAK:

Break line, flush line buffer. Set pointer to left margin

7.2 Programming Guidelines
Example of Pseudocode (continued)

- Regroyped

```
FIRST:
    PARAGRAPH, INDENT, SKIP_LINE, MARGIN, BREAK, PAGE:
        Break line, flush line buffer.
    SOUBLE_SPACE, SINGLE_SPACE :
        No break line, no flush line buffer.
SECOND:
    INDENT, SKIP_LINE, MARGIN:
        Get parameter.
    PARAGRAPH, BREAK, PAGE, DOUBLE_SPACE, SINGLE_SPACE:
        No parameter needed.
THIRD:
    PARAGRAPH, INDENT, SKIP_LINE, MARGIN, BREAK, PAGE:
        Set new line pointer.
    DOUBLE_SPACE, SINGLE_SPACE:
        New line pointer unchanged.
FOURTH:
        Individual action taken
```

7.2 Programming Guidelines
Example of Pseudocode (continued)

- Final pseudocode

INITIAL:

Get parameter for indent, skip_line, margin.

Set left margin to parameter for indent.

Set temporary line pointer to left margin for all but paragraph; for paragraph, set to paragraph indent.

LINE_BREAKS:

If not (DOUBLE_SPACE or SINGLE_SPACE), break line, flush line buffer and set line pointer to temporary line pointer

If 0 lines left on page, eject page and print page header.

INDIVIDUAL CASES:

INDENT, BREAK: do nothing.

SKIP_LINE: skip parameter lines or eject

PARAGRAPH: advance 1 line; if < 2 lines or page, eject.

MARGIN: right_margin = parameter.

DOUBLE_SPACE: interline_space = 2.

SINGLE_SPACE: interline_space = 1;

PAGE: eject page, print page header

7.2 Programming Guidelines
Consumer Reuse

- Four key characteristics to check about components to reuse
  - does the component perform the function or provide the data needed?
  - is it less modification than building the component from scratch?
  - is the component well-documented?
  - is there a complete record of the component's test and revision history?

7.2 Programming Guidelines
Producer Reuse

- Several issues to keep in mind
  - make the components general
  - separate dependencies (to isolate sections likely to change)
  - keep the component interface general and well-defined
  - include information about any faults found and fixed
  - use clear naming conventions
  - document data structures and algorithms
  - keep the communication and error-handling sections separate and easy to modify

7.2 Programming Guidelines
Sidebar 7.2 Selecting Components for Reuse at Lucent

- Reuse Council
  - Created inventory of components
  - Formed matrix with the features of all past and planned projects
  - Met every week to make component selections, inspect design documentation, and monitor levels of reuse

7.3 Documentation

- Internal documentation
  - header comment block
  - meaningful variable names and statement labels
  - other program comments
  - format to enhance understanding
  - document data (data dictionary)
- External documentation
  - describe the problem
  - describe the algorithm
  - describe the data

7.3 Documentation
Information Included in Header Comment Block

- What is the component called
- Who wrote the component
- Where the component fits in the general system design
- When the component was written and revised
- Why the component exists
- How the component uses its data structures, algorithms, and control

7.4 The Programming Process
Programming as Problem-Solving

- Polya's (1957) four distinct stages of finding a good solution
  - understanding the problem
  - devising plan
  - carrying out the plan
  - looking back

7.4 The Programming Process
Extreme Programming

- Two types of participants
  - *customers*: who define the features using stories, describe detailed tests and assign priorities
  - *programmers*: who implement the stories

7.4 The Programming Process
Pair Programming

- The driver or pilot: controlling the computer and writing the code
- The navigator: reviewing the driver's code and providing feedback

7.4 The Programming Process
Whither Programming?

- Documentation is still essential in agile-methods
  - Assist the developers in planning, as a roadmap
  - Helps describe key abstractions and defines system boundaries
  - Assists in communicating among team members

7.5 Information System Example
Piccadilly System

- Design Description

  Input: *Opposition schedule*
  For each *Television company name*, create *Opposition company*.
  For each *Opposition schedule*,
  Locate the *Episode* where *Episode schedule date* =
  Opposition transmission date AND
  Episode start time = *Opposition transmission time*
  Create instance of *Opposition* program
  Create the relationships *Planning* and *Competing*
  Output: List of *Opposition programs*

- Data dictionary description

  Opposition schedule = * Data flow *
  Television company name
  + {Opposition transmission date
  + Opposition transmission time +
  Opposition program name
  + (Opposition predicted rating)}

7.5 Information System Example
Piccadilly System's Implementation

- Passing by value

```
void Match:: calv(Episode episode_start_time)
{
first_advert = episode_start_time + increment;
// The system makes a copy of Episode
// and your program can use the values directly.
}
```

- Passing by pointer

```
void Match:: calp(Episode* episode)
{
episode->setStart (episode->getStart());
// This example passes a pointer to an instance of Episode.
// Then the routine can invoke the services (such as setStart
// and getStart) of Episode using the -> operator.
}
```

- Passing by reference

```
void Match:: calr(Episode& episode)
{
episode.setStart (episode.getStart());
// This example passes the address of Episode.
// Then the routine can invoke the services (such as setStart
// and getStart) of Episode using the . operator.
}
```

7.6 Real-Time Example
Ariane-5

- Should have included an exception handler

```
try {
}
catch (…..) {
    //attempt to patch up state
    //either satisfy postcondition or raise
      exception again
}
```

7.7 What This Chapter Means for You

- Things to consider when writing a code
  - organizational standards and guidelines
  - reusing code from other projects
  - writing code to make it reusable on future projects
  - using the low-level design as an initial framework, and moving in several iterations from design to code