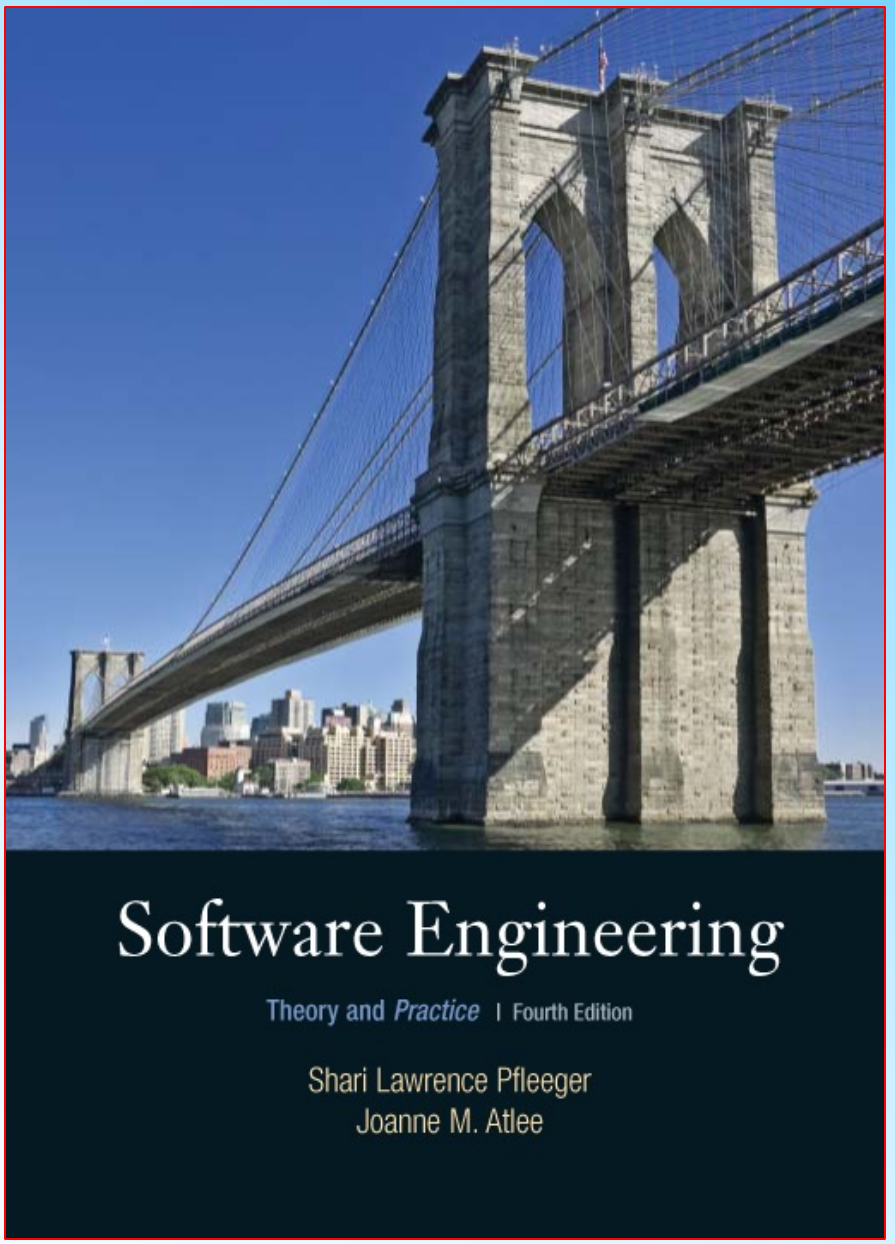
Writing the Programs

Shari L. Pfleeger
Joann M. Atlee

4th Edition



Chapter 7 Objectives

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

Programming as a Challenging Task

- All detailed issues have not been addressed
- Structures and Relationships that are easy to draw may not be straightforward to program
- Code understandable for others
- Benefit from characteristics of design as well as keeping the code reusable

7.1 Programming Standards and Procedures

- Standards for you
  - methods of code documentation
- Standards for others
  - Integrators, maintainers, testers
  - Prologue documentation
- Matching design with implementation
  - Changes in design can be easily implemented in code
    - Carry forward the design's modularity

7.1 Programming Standards and Procedures Prologue Documentation

```
*************************************************************
*
* COMPONENT TO FIND INTERSECTION OF TWO LINES
*
* COMPONENT NAME: FINDPT
* PROGRAMMER: E. ELLIS
* VERSION: 1.0 (2 FEBRUARY 2001)
*
* PROCEDURE INVOCATION:
*   CALL FINDPT (A1, B1, C1, A2, B2, C2, XS, YS, FLAG)
* INPUT PARAMETERS:
*   INPUT LINES ARE OF THE FORM
*        A1*X + B1*Y + C1 = 0 AND
*        A2*X + B2*Y + C2 = 0
*   SO INPUT IS COEFFICIENTS A1, B1, C1 AND A2, B2, C2
* OUTPUT PARAMETERS:
*   IF LINES ARE PARALLEL, FLAG SET TO 1.
*   ELSE  FLAG = 0 AND POINT OF INTERSECTION IS (XS, YS)
*
*************************************************************
```

7.2 Programming Guidelines
Control Structures

- Make the code easy to read
  - Reader should not worry about understanding the control flow
  - Reflect the design's control structure

7.2 Programming Guidelines
Example of Control Structures

- Control skips around among the program's statements

```
        benefit = minimum;
        if (age < 75) goto A;//may be a function call
        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
```

Restructuring

- 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
Control Structures

- Make the code not too specific, and not too general
  - Specific enough to be understood, generic enough to be re-used
- Use parameter names and comments to exhibit coupling among components

```
Reestimate TAX
```

it is better to write

```
Reestimate TAX based on values of GROSS_INC and DEDUCTS
```

- Make the dependency among components visible

7.2 Programming Guidelines
Algorithms

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

Learn compiler's way of optimization

Do not sacrifice clarity and correctness for speed

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	5500	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
General Guidelines to Preserve Quality

- Employ pseudocode
- Revise and rewrite,
- Reuse

7.2 Programming Guidelines

Consumer Reuse(Developed for other projects)

- 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(subsequent apps)

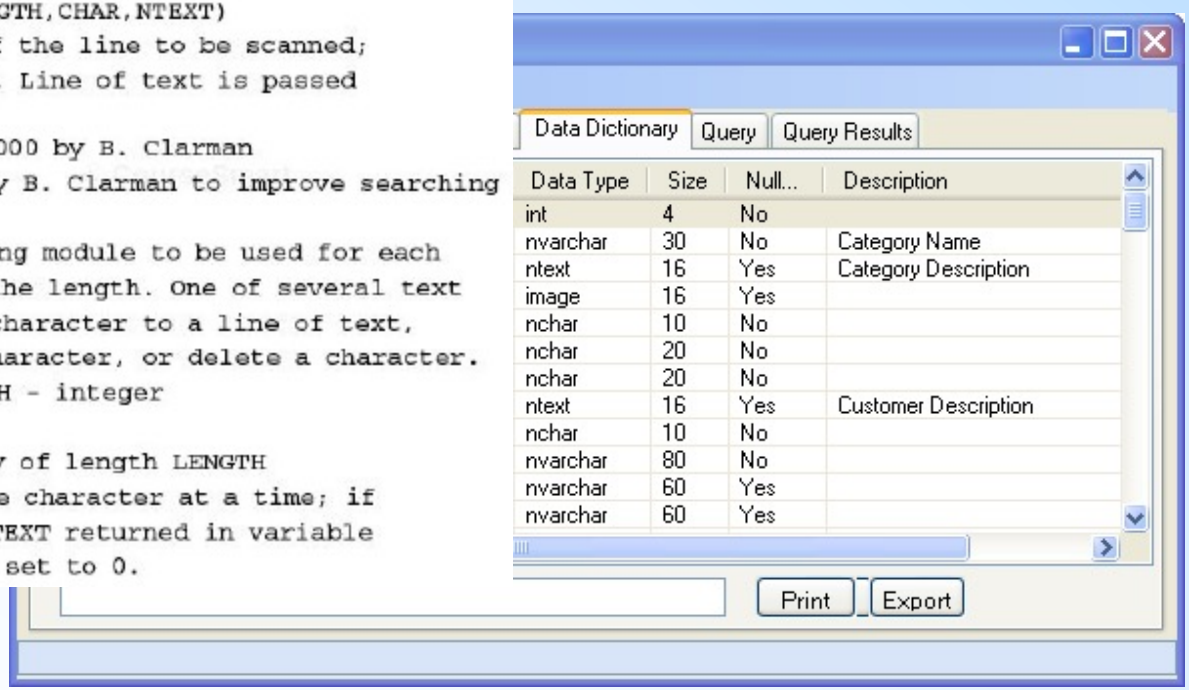
- Several issues to keep in mind
  - make the components general
  - separate dependencies (to isolate sections likely to change)
  - keep the component interface general and well-defined
  - include information about any faults found and fixed
  - use clear naming conventions
  - document data structures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

7.3 Documentation

- Internal documentation
  - header comment block

```
PROGRAM SCAN: Program to scan a line of text for a given
character
PROGRAMMER: Beatrice Clarman (718) 345-6789/bc@power.com
CALLING SEQUENCE: CALL SCAN(LENGTH,CHAR,NTEXT)
   where LENGTH is the length of the line to be scanned;
   CHAR is the character sought. Line of text is passed
   as array NTEXT.
VERSION 1: written 3 November 2000 by B. Clarman
REVISION 1.1: 5 December 2001 by B. Clarman to improve searching
   algorithm.
PURPOSE: General-purpose scanning module to be used for each
   new line of text, no matter the length. One of several text
   utilities designed to add a character to a line of text,
   read a character, change a character, or delete a character.
DATA STRUCTURES: Variable LENGTH - integer
   Variable CHAR - character
   Array NTEXT - character array of length LENGTH
ALGORITHM: Reads array NTEXT one character at a time; if
   CHAR is found, position in NTEXT returned in variable
   LENGTH; else variable LENGTH set to 0.
```

d statement labels



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

7.3 Documentation

- External documentation
  - describe the problem
  - describe the algorithm
  - describe the data

7.4 The Programming Process

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

Disclaimer

- Minor modifications have been made in content and flow of UCF slides