SE 3I03 Fall 2004

04 Software Documentation

Instructor: W. M. Farmer

Revised: 18 November 2004

What is a Software Product?

- A software product is represented and distributed as a **package of documentation**
  - Includes many kinds of documentation
  - Often organized as a directory of subdirectories
- Some documentation is expressed in **formal language**
  - Meant to be read by machines or other software
  - Can be very hard or impossible for humans to read
- Other documentation is express in **natural language**
  - Can be very hard for humans to understand
- The use of **logic** and **mathematics** can significantly improve the quality of documentation
  - But can still be very hard for humans to understand

Purpose of Software Documentation

1. Provide a software product in an **executable form**

2. **Describe** the software product

   - The **requirements** the product is intended to satisfy
   - The **design** of the product
   - The **implementation** of the product
   - The **capabilities** and **limitations** of the product
   - The product from **different perspectives**

3. Show that the product is **correct**

4. Make the product easier to:

   - **Use**
   - **Maintain**
   - **Reuse**

Kinds of Software Documentation

1. Requirements specifications
2. Design descriptions
3. Source code
4. Verification and testing results
5. Instructions for the administrator
6. Instructions for the user
7. Standards documentation

What makes Software Documentation Different?

1. Software is not physical or strongly visible
2. A software product is represented and distributed as documentation
3. Software involves formal, machine-readable languages
4. The content of software is largely logic and mathematics
5. Software documentation has many different purposes
6. A software product needs many kinds of documentation

Styles of Software Documentation (1)

1. Tutorial
   - Teaches some aspect of the software
   - Often used for user documentation

2. Reference document
   - Software parts are presented individually
   - Is not intended to be read cover to cover

3. Hyperlinked set of documents
   - Requirements are connected to the design, the design is connected to the code, etc.

4. Layered set of documents
   - Presents software at different layers of abstraction
   - Example: algorithms are presented separately from code

Styles of Software Documentation (2)

5. Derivational document
   - Shows how software is derived from the requirements
   - May be a proof of correctness

6. Single-source document
   - Different documents and different views are generated from a single source
   - Example: **literate programming** where a documentation file and a code file are generated from the same source
   - Example: Javadoc and similar approaches
   - Example: A family of manuals generated from the same source

Styles of Software Documentation (3)

7. On-line, searchable documentation
   - Can include a cut-and-paste facility for assembling new documentation

8. Documents with reader-controlled syntax
   - Reader chooses his or her own presentation syntax, conventions, and format
   - Frees the reader from the standard syntax of a formal language

Code: General Recommendations

- Structure the code in a consistent manner
- As a general rule, choose clarity before efficiency
- Express the structure of the software's design in the software's code
- Follow the conventions of the programming language being used

Keep the Code Simple

- Write procedures that fit on one screen
- Put at most one programming statement on a line
- Keep the following measures low:
  - Loop nesting level
  - Conditional nesting level
  - Number of local variables in a procedure
- Avoid control structures that radically change state
  - Exits, gotos, state jumps, self-modifying code
- Avoid nonstandard language features

Naming Programming Entities

- Naming is an important but difficult task

- One should employ a naming convention
  - Names should be short and descriptive
  - The more global the entity, the more descriptive the name should be
  - The more local, the shorter the name can be

- A name may include:
  - Type of entity or return value
  - Name of module

- Words in a name can be separated by underscores, hyphens, and case changes, but avoid using spaces

Formatting Code

- Use formatting to display the structure of the code
  - Indentation to display subordinate relationships between code
  - Alignment to identify blocks of code
  - Blank lines to separate blocks of code
- Write fully bracketed code to facilitate maintenance
- Write code in tabular form whenever possible
- Avoid “wrap-around” code
- Line up comments to the right of the code

Scope of Variables

- Make the scope of variables as narrow as possible
  - Avoid global variables

- A wide-scoped variable is:
  - Harder to maintain because its instances may appear far apart from each other
  - More easily corrupted because its data can be modified by diverse procedures

- Decrease the scope of a variable by introducing procedures for accessing the variable

Procedures

- Use a convention for naming and ordering parameters
- Make explicit and carefully control any side-effects
  - Keep the use of side-effects to a minimum
- Make the scope of procedures as narrow as possible
- Any code fragment used more than once should be made into a procedure
  - Make procedures powerful
  - Use simple procedures to invoke powerful procedures in special ways

Commenting Code

- Begin every code file with:
  - Copyright statement
  - Authors
  - Description of contents
  - Revision date and log of changes made to the file
- Comment:
  - Each variable declaration
  - Each procedure definition
  - Loops and larger blocks of code
  - Anything that is not obvious
- Avoid excessive comments in procedure bodies
  - **Write code so that what it does is obvious**