# SOFTWARE ENGINEERING

## CODING STANDARD & GUIDELINES

# Coding Standard & Guidelines

Good software development organizations normally require their programmers to adhere to some well-defined and standard style of coding called coding standards. Most software development organizations formulate their own coding standards that suit them most, and require their engineers to follow these standards rigorously. The purpose of requiring all engineers of an organization to adhere to a standard style of coding is the following:

- A coding standard gives a uniform appearance to the codes written by different engineers.
- It enhances code understanding.
- It encourages good programming practices.

The following are some representative coding standards.

**Rules for limiting the use of global:** These rules list what types of data can be declared global and what cannot.

**Contents of the headers preceding codes for different modules:** The information contained in the headers of different modules should be standard for an organization. The exact format in which the header information is organized in the header can also be specified. The following are some standard header data:

- Name of the module.
- Date on which the module was created.
- Author's name.
- Modification history.
- Synopsis of the module.
- Different functions supported, along with their input/output parameters.
- Global variables accessed/modified by the module.

**Naming conventions for global variables, local variables, and constant identifiers:** A possible naming convention can be that global variable names always start with a capital letter, local variable names are made of small letters, and constant names are always capital letters.

**Error return conventions and exception handling mechanisms:** The way error conditions are reported by different functions in a program are handled should be standard within an organization. For example, different functions while encountering an error condition should either return a 0 or 1 consistently.

The following are some representative coding guidelines recommended by many software development organizations.

**Do not use a coding style that is too clever or too difficult to understand:** Code should be easy to understand. Many inexperienced engineers actually take pride in writing cryptic and incomprehensible code. Clever coding can obscure meaning of the code and hamper understanding. It also makes maintenance difficult.

**Avoid obscure side effects:** The side effects of a function call include modification of parameters passed by reference, modification of global variables, and I/O operations. An obscure side effect is one that is not obvious from a casual examination of the code. Obscure side effects make it difficult to understand a piece of code. For example, if a global variable is changed obscurely in a called module or some file I/O is performed which is difficult to infer from the function's name and header information, it becomes difficult for anybody trying to understand the code.

**Do not use an identifier for multiple purposes:** Programmers often use the same identifier to denote several temporary entities. For example, some programmers use a temporary loop variable for computing and a storing the final result. The rationale that is usually given by these programmers for such multiple uses of variables is memory efficiency, e.g. three variables use up three memory locations, whereas the same variable used in three different ways uses just one memory location. However, there are several things wrong with this approach and hence should be avoided. Some of the problems caused by use of variables for multiple purposes as follows:

- Each variable should be given a descriptive name indicating its purpose. This is not possible if an identifier is used for multiple purposes. Use of a variable for multiple purposes can lead to confusion and make it difficult for somebody trying to read and understand the code.

- Use of variables for multiple purposes usually makes future enhancements more difficult.

**The code should be well-documented:** As a rule of thumb, there must be at least one comment line on the average for every three-source line.

**The length of any function should not exceed 10 source lines:** A function that is very lengthy is usually very difficult to understand as it probably carries out many different functions. For the same reason, lengthy functions are likely to have disproportionately larger number of bugs.

**Do not use goto statements:** Use of goto statements makes a program unstructured and makes it very difficult to understand.