8.1 CODING

The input to the coding phase is the design document. During the coding phase, different modules identified in the design document are coded according to the module specifications. It may be recollected that at the end of the design phase, we have not only the module structure (e.g. structure chart) of the system but also the module specifications where the data structures and algorithms for each module are specified. Therefore, we can say that the objective of the coding phase is to transform the design of the system, as given by its module specification, into a high-level language code and then to unit test this code.

Normally, good software development organizations 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. The reasons for adhering to a standard coding style are the following:

- it gives a uniform appearance to the codes written by different engineers,
- it enhances code understanding, and
- it encourages good programming practices.

A coding standard sets out standard ways of doing several things such as the way variables are to be named, the code is to be laid out, maximum number of source lines that can be allowed per function,

etc. Besides the coding standards, several coding guidelines are also suggested by software companies. Coding guidelines provide some general suggestions regarding the coding style to be followed but leave the actual implementation of these guidelines to the discretion of the individual engineers.

After a module has been coded, usually *code inspection* and *code walk-throughs* are carried out to ensure that coding standards are followed and to detect as many errors as possible before testing. It is important to detect as many errors as possible during code inspection, code walk-throughs, and unit testing because an error detected at these stages requires much less effort for debugging than the effort that would be needed if the same error was detected during integration or system testing. We first discuss some representative coding standards and guidelines, before discussing code walk-throughs and code inspection. We will then take up different testing techniques.

8.1.1 Coding Standards and Guidelines

Different organizations usually develop their own coding standards and guidelines depending on what best suits them. Therefore, we list here a few representative coding standards and guidelines commonly adopted by many software development organizations.

Representative Coding Standards

Rules for limiting the use of globals. 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 in a standard format. 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 up 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 and the way common exception conditions are handled should be standardized within an organization. For example, different functions when they encounter an error condition should either return a 0 or 1 consistently.

Representative Coding Guidelines

The following are some representative coding guidelines recommended by many software development organizations. Wherever necessary, the rationale underlying these guidelines is also mentioned.

Do not use too clever and difficult to understand coding style. A code should be easy to understand. Many inexperienced engineers actually take pride in writing cryptic and incomprehensible codes. Clever coding can obscure the 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, it becomes difficult for anybody 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 also storing the final result. The rationale that is usually given by these programmers for such multiple use 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 practice and hence it should be avoided. Some of the inconveniences caused by such multiple use of variables are 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. The use of a variable for multiple purposes can lead to confusion and annoyance for somebody trying to read and understand the code.
- The use of variables for multiple purposes usually makes future enhancements extremely 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 lines.

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.

Do not use goto statements, etc. The use of goto statements makes a program unstructured and difficult to understand.