IGHODARO OSATO ALEXANDRA

15/ENG05/010

MCT 506 ASSIGMENT TWO

CODING

The coding phase transforms the design of a system into code in a high-level language and then to unit test this code. The coding standard is what programmers use as standard and well-defined style of coding. Below are important reasons for a standard coding style

i. A coding standard gives uniform appearances to the code written by different engineers
ii. It facilitates code of understanding and promotes good programming practices.

Characteristics of a Programming Language: Readability, portability, generality, brevity, error checking, efficiency, modularity, widely available, cost, familiar notation, quick translation etc.

Coding standards and guidelines

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

2. Contents of the headers preceding codes for different modules: The headers of different modules should be standard for an organization. The exact format in which the header information is organize in the header can also be specify.

3. 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.

4. Error return conventions and exception handling mechanisms: For example, different functions while encountering an error condition should return either a 0 or 1 consistently.

Below are some vital representative recommended coding guidelines;

1. Do not use a coding style that is too clever or too difficult to understand: Code should be easy to understand. Clever coding can obscure meaning of the code and hamper understanding.

2. Avoid obscure side effects: An obscure side effect is one that is not obvious from a casual examination of the code.

3. Do not use an identifier for multiple purposes: Programmers often use the same identifier to denote several temporary entities.

4. Well documentation of code: As a rule of thumb, there must be at least one comment line on the average for every three-source line.

5. The length of any function should not exceed 10 source lines: Lengthy functions are likely to have disproportionately larger number of bugs.

6. Do not use go to statements: Use of go to statements makes a program unstructured.

Code Review: Code review for a model is carried out after the module is successfully compiled and the all the syntax errors have been eliminated.

Code Walk Through: Code walk through is an informal code analysis technique. After a module has been coded, successfully compiled and all syntax errors eliminated. A few members of the development team are given the code few days before the walk through meeting to read and understand code. Each member selects some test cases and simulates execution of the code by hand. The main objectives of the walk through are to discover the algorithmic and logical errors in the code.

Code Inspection: The aim of code inspection is to discover some common types of errors caused due to oversight and improper programming.

Clean Room Testing: IBM pioneered Clean room testing. This type of testing relies heavily on walk through inspection. The programmers are not to test any of their code by executing the code other than doing some syntax testing using a compiler.

Internal documentation: Internal documentation is the code comprehension features provided as part of the source code. It is provided through module headers and comments embedded in the source code.

External documentation: External documentation are gotten through types of supporting documents such as requirements specification document, design document, test documents.

Program Testing: Testing a program consists of providing the program with a set of test inputs and observing if the program behaves as expected.

Aim of Testing: The aim of the testing process is to identify all defects existing in a software product.

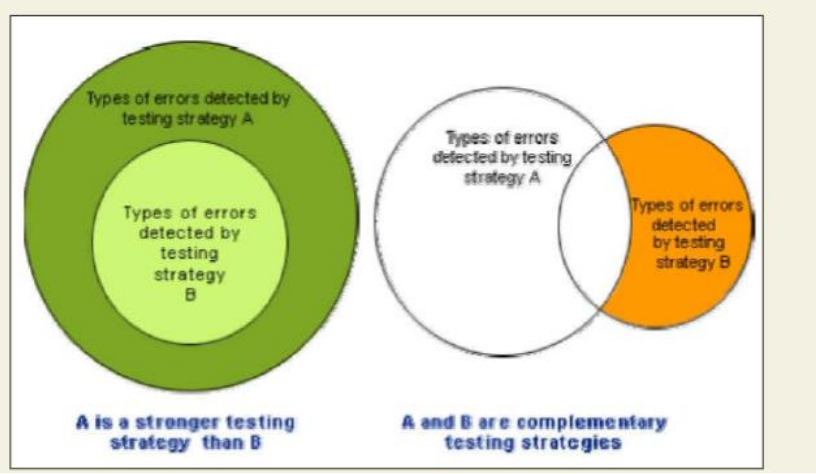
Verification Vs Validation: Verification is the process of determining whether the output of one phase of software development conforms to that of its previous phase, whereas validation is the process of determining whether a fully developed system conforms to its requirements.

BLACK-BOX TESTING

Testing in the large vs. testing in the small: Software products are tested first at the unit level. This is testing in the small. After testing all the components individually, the components are slowly integrated and tested at each level of integration (integration testing). The fully integrated system is tested (system testing). Integration and system testing are known as testing in the large.

WHITE-BOX TESTING

One white-box testing strategy is said to be stronger than another strategy, if all types of errors detected by the first testing strategy is also detected by the second testing strategy, and the second testing strategy additionally detects some more types of errors. When two testing strategies detect errors that are different at least with respect to some types of errors, then they are complementary.



Figure; stronger and complementary testing strategies

DEBUGGING, INTEGRATION AND SYSTEM TESTING

Once errors are identified in a program code, it is necessary to first identify the precise program statements responsible for the errors and then to fix them. Identifying errors in a program code and then fix them up are known as debugging.

The following are some of the approaches for debugging.

i. Brute Force Method: This is the most common method of debugging but is the least efficient method
ii. Backtracking: This is also a fairly common approach. In this approach, beginning from the statement at which an error symptom has been observed, the source code is traced backwards until the error is discovered.
iii. Cause Elimination Method: In this approach, a list of causes which could possibly have contributed to the error symptom is developed and tests are conducted to eliminate each.
iv. Program Slicing: This technique is similar to back tracking. Here there is reduction of search page by defining slices. A slice of a program for a particular variable at a particular statement is the set of source lines preceding this statement that can influence the value of that variable.

The following are some general guidelines for effective debugging:

Many times debugging requires a thorough understanding of the program design. Trying to debug based on a partial understanding of the system design and implementation may require an inordinate amount of effort to be put into debugging even simple problems.

Debugging may sometimes even require full redesign of the system. In such cases, a common mistake that novice programmers often make is attempting not to fix the error but its symptoms After every round of error-fixing, regression testing must be carried out.

Program Analysis Tools: A program analysis tool means an automated tool that takes the source code or the executable code of a program as input and produces reports regarding several important characteristics of the program, such as its size, complexity, adequacy of commenting, adherence to programming standards, etc. We can classify these into two broad categories of program analysis tools: Static Analysis tools and Dynamic Analysis tools.

INTEGRATION TESTING

The primary objective of integration testing is to test the module interfaces, i.e. there are no errors in the parameter passing, when one module invokes another module. During integration testing, different modules of a system are integrated in a planned manner using an integration plan. The integration plan specifies the steps and the order in which modules are combined to realize the full system. After each integration step, the partially integrated system is tested.

There are four types of integration testing approaches. Any one (or a mixture) of the following approaches can be used to develop the integration test plan. Those approaches are the following:

i. Big bang approach
ii. Bottom- up approach
iii. Top-down approach
iv. Mixed-approach

Phased Vs. Incremental Testing

The different integration testing strategies are either phased or incremental. A comparison of these two strategies is as follows:

In incremental integration testing, only one new module is added to the partial system each time.

In phased integration, a group of related modules are added to the partial system each time

SOFTWARE MAINTENANCE

Software maintenance is becoming an important activity of a large number of software organizations. This is no surprise, given the rate of hardware obsolescence, the immortality of a software product per se, and the demand of the user community to see the existing software products run on newer platforms, run in newer environments, and/or with enhanced features. When the hardware platform is changed, and a software product performs some low-level functions, maintenance is necessary. Also, whenever the support environment of a software product changes, the software product requires rework to cope up with the newer interface. For instance, a software product may need to be maintained when the operating system changes.

Thus, every software product continues to evolve after its development through maintenance efforts. Therefore it can be stated that software maintenance is needed to correct errors, enhance features, port the software to new platforms,

Types of software maintenance

There are three types of software maintenance.

i. Corrective: Corrective maintenance of a software product is necessary to rectify the bugs observed while the system is in use.
ii. Adaptive: A software product might need maintenance when the customers need the product to run on new platforms, on new operating systems, or when they need the product to interface with new hardware or software.
iii. Perfective: A software product needs maintenance to support the new features that users want it to support, to change different functionalities of the system according to customer demands, or to enhance the performance of the system.

Problems associated with software maintenance

The software maintenance is one of the most neglected areas of software engineering. Even though software maintenance is fast becoming an important area of work for many companies as the software products of before. Software maintenance has a very poor image in industry. Therefore, an organization often cannot employ bright engineers to carry out maintenance work. Even though maintenance suffers from a poor image, the work involved is often more challenging than development work.

Software Reverse Engineering: Software reverse engineering is the process of recovering the design and the requirements specification of a product from an analysis of its code. The purpose of reverse engineering is to facilitate maintenance work by improving the understandability of a system and to produce the necessary documents for a legacy system. Reverse engineering is becoming important, since legacy software products lack proper documentation, and are highly unstructured. Even well designed products become legacy software as their structure degrades through a series of maintenance efforts.