**NLEMADIM CHIAZOR**

**15/ENG05/015**

**CODING**

Coding, in simpler terms, means feeding our commands in the computer in a language the computer understands, so that the computer can carry out the said command, and perform the task.The objective of the coding phase is to transform the design of a system into code in a high level language and then to unit test this code.

**Characteristics of a Programming Language**

Readability, Cost, Error checking, Portability, Generality, Brevity, Familiar notation, Quick translation, Efficiency, Modularity and widely available

 **Coding standards and guidelines**

The following are some representative coding standards.

1. Rules for limiting the use of global

2. Contents of the headers preceding codes for different modules

3. Naming conventions for global variables, local variables, and constant identifiers

4. Error return conventions and exception handling mechanisms

**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 reviews are extremely cost-effective strategies for reduction in coding errors and to produce high quality code. There are two types of code review techniques, they are code walk through and code inspection

**Code Walk Throughs**

Code walk through is an informal code analysis technique. In this 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 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**

In contrast to code walk through, the aim of code inspection is to discover some common types of errors caused due to oversight and improper programming. In other words, during code inspection the code is examined for the presence of certain kinds of errors, in contrast to the hand simulation of code execution done in code walk throughs’. Following is a list of some classical programming errors which can be checked during code inspection: Use of uninitialized variables, Jumps into loops, Non-terminating loops, Incompatible assignments, Array indices out of bounds, improper storage allocation, Mismatches between actual and formal parameter in procedure calls e.t.c.

**Clean Room**

Clean room testing was pioneered by IBM. This type of testing relies heavily on walk throughs, inspection, and formal verification. The programmers are not allowed to test any of their code by executing the code other than doing some syntax testing using a compiler. The clean room approach to software development is based on five characteristics: Formal specification, Incremental development, Structured programming, Static verification, and Statistical testing of the system.

**Software Documentation**

When various kinds of software products are developed then not only the executable files and the source codes are developed but also various kinds of documents such as users’ manual, software requirements specification (SRS) documents, design documents, test documents, installation manual, etc are also developed as part of any software engineering process. Different types of software documents can broadly be classified into the following; Internal documentation and External documentation

**TESTING**

**Program Testing**

Testing a program consists of providing the program with a set of test inputs and observing if the program behaves as expected. If the program fails to behave as expected, then the conditions under which failure occurs are noted for later debugging and correction. 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 specification.

**Functional Testing Vs. Structural Testing**

 In the black-box testing approach, test cases are designed using only the functional specification of the software, i.e. without any knowledge of the internal structure of the software. For this reason, black-box testing is known as functional testing. On the other hand, in the white-box testing approach, designing test cases requires thorough knowledge about the internal structure of software, and therefore the white-box testing is called structural testing.

**BLACK-BOX TESTING**

Software products are normally tested first at the individual component (or unit) level. This is referred to as testing in the small. After testing all the components individually, the components are slowly integrated and tested at each level of integration (integration testing). Finally, the fully integrated system is tested (called system testing). Integration and system testing are known as testing in the large.

**Unit Testing**

Unit testing is undertaken after a module has been coded and successfully reviewed. Unit testing (or module testing) is the testing of different units (or modules) of a system in isolation. 

**Diagram of unit testing**

**Black Box Testing**

In the black-box testing, test cases are designed from an examination of the input/output values only and no knowledge of designer code is required. There are the two main approaches to designing black box test cases and they are; Equivalence Class Partitioning and Boundary Value Analysis

**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 called complementary.

**Statement Coverage**

 The statement coverage strategy aims to design test cases so that every statement in a program is executed at least once. The principal idea governing the statement coverage strategy is that unless a statement is executed, it is very hard to determine if an error exists in that statement.

**Branch Coverage**

In the branch coverage-based testing strategy, test cases are designed to make each branch condition to assume true and false values in turn. Branch testing is also known as edge testing.

**Condition Coverage**

In this structural testing, test cases are designed to make each component of a composite conditional expression to assume both true and false values.

**Path Coverage**

 The path coverage-based testing strategy requires us to design test cases such that all linearly independent paths in the program are executed at least once.

**Control Flow Graph (CFG)**

A control flow graph describes the sequence in which the different instructions of a program get executed. In other words, a control flow graph describes how the control flows through the program. Control Flow Graph is represented differently for all statements and loops. Following images describe If-else, while, do-while and for



**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 is known as debugging. The following are some of the approaches popularly adopted by programmers for debugging: Brute Force Method, Backtracking, Cause Elimination Method and Program Slicing:

**Debugging Guidelines**

Debugging is often carried out by programmers based on their ingenuity. The following are some general guidelines for effective debugging:

Many times debugging requires a thorough understanding of the program design.

Debugging may sometimes even require full redesign of the system.

One must be beware of the possibility that an error correction may introduce new errors.

 **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 Tool and Dynamic program analysis tools

**INTEGRATION TESTING**

The primary objective of integration testing is to test the module interfaces. 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. There are four types of integration testing approaches and they are; Big-Bang Integration Testing, Bottom-Up Integration Testing, Top-Down Integration Testing and Mixed Integration Testing

**Phased Vs. Incremental Testing**

1. 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.
2. Phased integration requires less number of integration steps compared to the incremental integration approach.

There are essentially three main kinds of system testing, they are;Alpha Testing Beta testing and Acceptance Testing

**Performance Testing**

Performance testing is carried out to check whether the system needs the non-functional requirements identified in the SRS document. There are several types of performance testing; Stress Testing, Volume Testing , Configuration Testing , Compatibility Testing, Regression Testing, Recovery Testing, Maintenance Testing, Documentation Testing, Usability Testing-

**SOFTWARE MAINTENANCE**

Software maintenance is becoming an important activity of a large number of software organizations. When the hardware platform is changed, and a software product performs some low-level functions, maintenance is necessary.

**Types of software maintenance**

Corrective

Adaptive

Perfective

**Problems associated with software maintenance**

1 Software maintenance work typically is much more expensive than what it should be and takes more time than required.

2 Software maintenance has a very poor image in industry.

3 Another problem associated with maintenance work is that the majority of software products needing maintenance are legacy products.

**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.

**Steps of Software Reverse Engineering:**

1. **Collection Information**
2. **Examining the information**
3. **Extracting the structure**
4. **Recording the functionality**
5. **Recording data flow**
6. **Recording control flow**
7. **Review extracted design**
8. **Generate documentation**



**Software development process model diagram**

**Legacy software products**

 The typical problems associated with legacy systems are poor documentation, unstructured (spaghetti code with ugly control structure), and lack of personnel knowledgeable in the product. Many of the legacy systems were developed long time back.