NAME: NDIFE KINGSLEY NZUBECHUKWU

MATRIC: 15/ENG05/014

COURSE: COMPUTER SOFTWARE ENGINEERING II

COURSE CODE: MCT 506

ASSIGNMENT 3

DATE: 5 -05 -2020

.

A comprehensive summary on the topics: coding, testing, debugging, integration and system testing, and software reverse Engineering.

**Coding**

According to the department of CSE & IT VSSUT, Burla, 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. This is achieved using a high level language. Programmers adhere to certain standard and guideline while coding for the following reason; it promotes of easy understanding of codes, it promotes of understanding of codes and it facilitates uniform of codes written by different Engineers.

Attributes of a good programming language includes: the ability to allow programs be written in some ways that resemble a quite-English description of the underlying algorithms (readability), the ability to implement the algorithm with less amount of code (brevity), the ability to enforce a great deal of error checking for the programmers (Error checking) among others. Some generally accepted standards in coding are listed below:

1. Rules for limiting global which determines what type of data can be declared global.
2. Contents of the headers preceding codes for different modules which makes emphasis on the way information pertaining to different modules should be specific to an organization.
3. Naming conventions for global variables, local variables, and constant identifiers which emphasis on the format and manner at what constants, identifiers and global variable should be used in a code.
4. Error return conventions and exception handling mechanisms which suggests a way error conditions are reported by different functions in a program and handled.

Some coding guidelines recommended by software developing organizations include:

1. Do not use a coding style that is too complex or too difficult to understand: the coding should be easy to understand as complex codes can obscure meaning of the code and hamper understanding. It also makes maintenance difficult.
2. Programmers should take care to avoid obscure side effect. An obscure effect is one that cannot be observed from casual examination of a code and can occur when a global variable is changed obscurely in a module.
3. Programmers should avoid the use an identifier for multiple purposes.

After a module has been successfully compiled and the all the syntax errors have been eliminated, code reviews is carried out. The two most popular code review widely used are code walkthrough and code inspection. Code walkthrough is an informal code analysis technique whereby some members of the development team are selected and given the code to review before the walk through meeting with the aim of discovering logical and arithmetic errors. This may involves selecting test cases and simulating an execution of the code. On the other hand Code inspection is a formal inspection of the code with the aim of detecting error(s) caused by oversight, improper programming and failure to adhere to programming standards

Clean Testing is testing a technique testing pioneered by IBM that relies heavily on walk through, inspection, and formal verification. This technique reportedly produces documentation and code that is more reliable and maintainable than other development methods relying heavily on code execution-based testing and it has five approach software development which are:

1. A Formal specification which is developed using a state transition model which shows system responses to stimuli is used to express the specification.
2. An Incremental development where in the software is partitioned into increments which are developed and validated separately using the clean room process.
3. Structured programming which determines only a limited number of control and data abstraction constructs are that used
4. Static verification where the developed software is statically verified using rigorous software inspections.
5. Statistical testing of the system where the integrated software increment is tested statistically to determine its reliability. These statistical tests are based on the operational profile which is developed in parallel with the system specification

Software documentation refers to documents that are a vital part of good software development practice and are very useful. They can serve the purpose of enhancing good understandability and maintainability of a software product, effective handling of manpower turnover problem among others. Software document can either be internal document or external document. Internal documents are code comprehension features provided as part of the source code itself this may include comments. The external documents are various supporting document such as user’s manual, design and test document.

**Testing**

The aim of the testing process is to identify all defects existing in a software product. Testing can be in form of verification and validation with the difference being that verification determines whether the output of one phase of software development conforms to that of its previous phase, while validation checks whether a fully developed system conforms to its requirements specification. Testing can be done in large (integration and system testing) and in small (unit level testing).

Testing can also be in the form of functional testing or structural testing. Functional testing also known as black box testing allows test cases to be designed from an examination of the input or output values only without knowledge of design or code. This testing approach employs two techniques:

1. Equivalence class partitioning: In this approach, the domain of input values to a program is partitioned into a set of equivalence classes such that the behavior of the program is similar for every input data belonging to the same equivalence class.
2. Boundary value Analysis: This leads to selection of test cases at the boundaries of the different equivalence classes due to programming error that frequently occurs at the boundaries of different equivalence classes of inputs.

White box testing: this approach requires the developer to have knowledge of the design and code before developing a test case. 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. Some white box testing strategy includes:

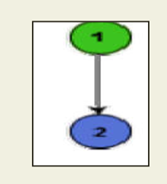
1. Statement coverage strategy which aims to design test cases so that every statement in a program is executed at least once.
2. Branch coverage: Here test cases are designed to make each branch condition to assume true and false values in turn. This is also known as edge testing since the edge of each program control flow graph is transversed once.

Control flow graph: this refers to the order in which the instructions in a program set are carried out. The statements of the program are first numbered before execution, this is referred as node and an edge from one node to another occurs when the execution of the statement in node passes the control to the other node. A control flow graph for a simple program is given below:

Sequence:

a=5;

b= a\*2-1;



**Debugging**

Debugging is the act of identifying errors in a program and fixing them up. Some popular debugging approaches used includes:

1. Brute force method: In this approach, the program is loaded with print statements to print the intermediate values with the hope that some of the printed values will help to identify the statement in error
2. Backtracking: 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.

Some debugging guidelines recommended by the institute includes having a thorough understanding of the code before debugging. Being aware that during debugging, correction could introduce new errors, and debugging could imply redesigning the system.

Program Analysis tool: These are automated tools automated tools that use the program's source code as input and provide reports on the key features of the software, such as its scale, complexity, and adequacy of comments. Program Analysis tools are broadly classified into: Static Analysis tool which assesses and computes various characteristics of a software product without executing it and Dynamic Analysis tool which require the program to be executed and its actual behavior recorded. Dynamic Analysis usually instruments the code by adding additional statements in the source code to collect program execution traces.

**Integration Testing**

The primary objective of integration testing is to test the module interfaces that were integrated in a planned manner using an integration plan. The four major integration testing approaches are

1. Big Bang integration Testing: Being the simplest integration testing approach, all the modules making up a system are integrated in a single step and tested.
2. Bottom- up integration Testing: In bottom-up testing, each subsystem is tested separately and then the full system is tested. The primary purpose of testing each subsystem is to test the interfaces among various modules making up the subsystem.
3. Top-down integration testing: Top-down integration testing starts with the main routine and one or two subordinate routines in the system.
4. Mixed integration testing: In the mixed testing approaches, testing can start as and when modules become available therefore overcoming the shortcoming of the top down or bottom up integration testing.

There are essentially three main kinds of system testing

1. Alpha test: this test is carried out by some members of the development team.
2. Beta testing: here, the system testing performed by a select group of friendly customers.
3. Acceptance Testing: This is a system testing performed by the customer to determine whether he should accept the delivery of the system.

System tests can also be categorized as functionality or performance test. The functionality test tests the functionality of the software to check whether it satisfies the functional requirements as documented in the SRS document. The performance test tests the conformance of the system with the nonfunctional requirements of the system. Some performance test includes stress test, Configuration testing, Compatibility testing among others.

Error seeding: This a technique used to estimate the number of residual errors in a system by introducing known errors into the code. The number of these seeded errors detected in the course of the standard testing procedure is determined and used in conjunction with the number of unseeded errors detected to predict the number of remaining errors in the product and effectiveness of the strategy.

Software maintenance: Software maintenance is necessary to correct errors, enhance features, and port the software to new platforms. Software maintenance can either be

1. Corrective maintenance: this is necessary to rectify the bugs observed while the system is in use.
2. Adaptive maintenance: this is done when the customers need the product to run on new platforms, on new operating systems, or interface with new hardware or software.
3. Perfective maintenance: This is done in order to allow the product support the new features that users want it to support, or to change different functionalities of the system according to customer demands.

Most software maintenance problem includes that it is expensive to carry out and it takes more time than required.

**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 aim of reverse engineering is to encourage maintenance work by improving the system's comprehensibility and generating the documentation required for the legacy system. The first stage of reverse engineering is carrying out cosmetic changes to the code which is done using any printer programs. This is followed by extracting the code, extracting of design done using automatic tools to derive the data flow and control flow diagram from the code. The process ends with writing a requirement specification. The activities involved in a software maintenance largely depends on the extent of modification required, availability of resources and condition of exiting product.