***SUMMARY OF LECTURE NOTE 17***

# CODING

**Coding-** 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. The programmers adhere to standard and well defined style of coding which they call their coding standard. This helps to make it more understanding to the average person and promoting good programming practice.

## Characteristics of a Programming Language

Readability, Portability, Generality, Brevity, Error checking, Cost, Familiar notation, Quick translation, Efficiency, Modularity, Widely Available.

## Coding standards and guidelines

Good software development organizations usually develop their own coding standards and guidelines depending on what best suits their organization and the type of products they develop.

The following are some representative coding guidelines recommended:

1. Do not use a coding style that is too clever or too difficult to understand.
2. Avoid obscure side effects.
3. Do not use an identifier for multiple purposes.
4. The code should be well-documented.
5. The length of any function should not exceed 10 source lines.
6. Do not use goto statements.

## 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. These two types code review techniques are:

## Code Walk Throughs

In this informal style, few members in the team are given modules to evaluate and find out logical and algorithmic errors and discuss this in their next meeting. Thus, the main coder should not feel too big or small since more brains are better than one especially in discussing errors.

## 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. Following is a list of some classical programming errors which can be checked during code inspection:

Use of uninitialized variables, Jumps into loops, Nonterminating loops, Incompatible.

## Clean Room Testing

This is based on avoiding software defects by using a rigorous inspection process. The objective of this software is zero-defect software based on: formal specification, Incremental development, structured programing, static verification and statistical testing of the system.

## Software Documentation

Other files such as users’ manual, software requirements specification (SRS) documents, design documents, etc. apart from the executable files are also developed as part of any software engineering process. Types are:

**Internal documentation** is the code comprehension features provided as part of the source code itself.

**External documentation** is provided through various types of supporting documents such as requirements specification document, design document, test documents.

***SUMMARY OF LECTURE NOTE 18***

# TESTING

Testing a program consists of providing the program with a set of test inputs and observing if the program behaves as expected. Some commonly used terms associated with testing are:

* **Failure:** This is a manifestation of an error (or defect or bug). But, the mere presence of an error may not necessarily lead to a failure.
* **Test case:** This is the triplet [I,S,O], where I is the data input, S is the state at which the data is input, and O is the expected output of the system.
* **Test suite:** This is the set of all test cases with which a given software product is to be tested.

## Verification Vs Validation

Verification is concerned with phase containment of errors, the aim of validation is that the final product be error free.

## Design of Test Cases

Test cases should be designed based on optimally detecting errors in a systematic manner.

## Functional Testing Vs. Structural Testing

Black-box testing approach (or functional), tests functional specification of the software, i.e. without any knowledge of the internal structure of the software while white-box testing approach (or structural), designing tests about the internal structure of software.

# BLACK-BOX TESTING

## Testing in the large vs. testing in the small

Firstly, at the individual component (or unit) level called testing in the small, then components are slowly integrated and tested at each level of integration (integration testing) then fully integrated system is tested (called system testing). The two last testing are testing in the large.

## Unit Testing

Unit testing is undertaken after a module has been coded and successfully reviewed.

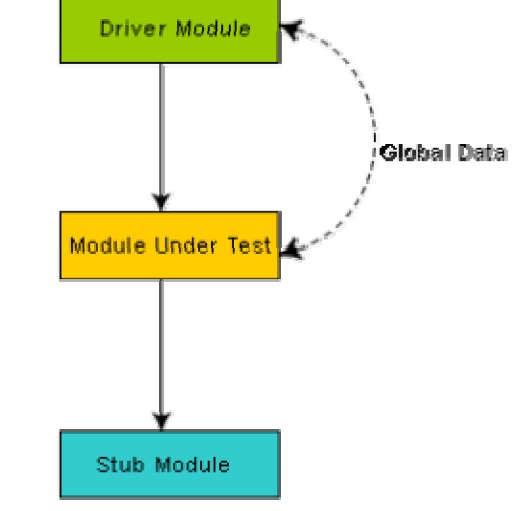


Fig. 19.1: Unit testing with the help of driver and stub modules

## Boundary Value Analysis

An error may occur due to a programmer making use < instead of <=, or conversely <= for < leading to selection of test cases at the boundaries of the different equivalence classes. **Example,** For a function that computes the square root of integer values in the range of 0 and 5000, the test cases must include the following values: {0, -1, 5000, 5001}.

***SUMMARY OF LECTURE NOTE 20***

# WHITE-BOX TESTING

It is said to be the stronger than strategy if it detects more errors than the other strategy but if they have common errors with and non-common errors then it becomes complimentary strategy.

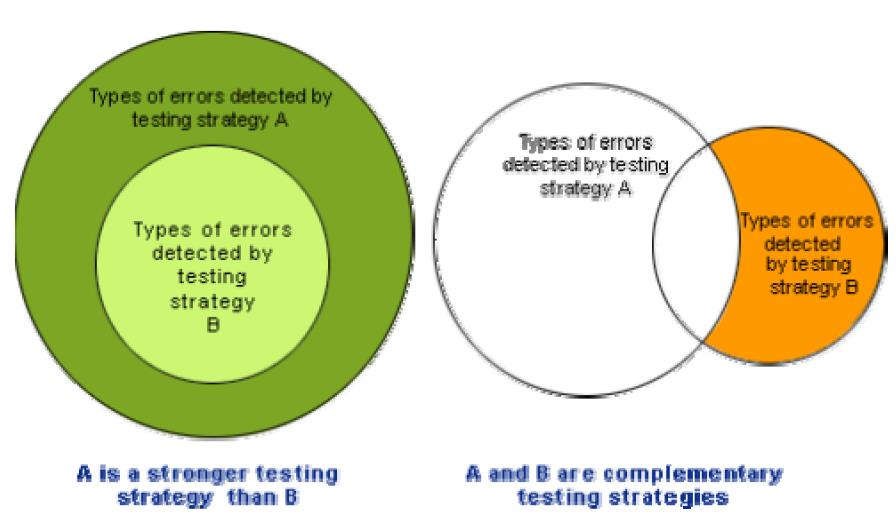


Fig. 20.1: Stronger and complementary testing strategies

## Statement Coverage

Aims to design test cases so that every statement in a program is executed at least once.

## 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.

## 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)

It describes the sequence in which the different instructions of a program get executed.

***SUMMARY OF LECTURE NOTE 22***

# DEBUGGING, INTEGRATION AND SYSTEM TESTING

## Need for Debugging

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 those up are known as debugging. Debugging approaches include: Brute Force Method.

## Debugging Guidelines

Full understanding of the program design.

One must be beware of the possibility that an error correction may introduce new errors. Therefore after every round of error-fixing, regression testing must be carried out.

## Program Analysis Tools

It is an automated tool that produces reports on characteristics like its size, complexity, adherence to programming standards, etc. using the source code or the executable code of a program as input.

Classified under: Static Analysis tools, Dynamic Analysis tools and Static program analysis tools.

***Static Analysis Tool -*** assesses and computes various characteristics of a software product without executing it e.g. static program analysis tool.

***Dynamic program analysis tools -*** Dynamic program analysis techniques require the program to be executed and its actual behavior recorded.

***SUMMARY OF LECTURE NOTE 23***

# INTEGRATION TESTING

It involves modules coming together for testing. Four testing approaches are

## Big-Bang Integration Testing

Where all the modules making up a system are integrated in a single step. Hard to find the error.

## Bottom-Up Integration Testing

In bottom-up testing, each subsystem is tested separately and then the full system is tested.Top-Down Integration Testing

## Top-down integration Testing

This approach is that in the absence of lower-level routines.

## Mixed Integration Testing

A mixed integration testing follows a combination of top-down and bottom-up testing.

## Phased Vs. Incremental Testing

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

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

**System testing** are designed to validate a fully developed system. Types are:

**Alpha Testing.** It is carried out by the test team within the developing organization.

**Beta testing.** Beta testing is performed by a select group of friendly customers.

**Acceptance Testing.** Performed by the customer to accept or reject the delivery of the system.

## Performance Testing

Performance testing is carried out to check whether the system needs the non-functional requirements identified in the SRS document. All performance tests can be considered as black-box tests and its types are Stress, Volume, Configuration, Compatibility, Regression, Recovery, Maintenance, Documentation and Usability testing.

## Error Seeding

Sometimes the customer might specify the maximum number of allowable errors that may be present in the delivered system

Let N be the total number of defects in the system and let n of these defects be found by testing. Let S be the total number of seeded defects, and let s of these defects be found during testing.

## n/N = s/S or N = S × n/s

***SUMMARY OF LECTURE NOTE 24***

# SOFTWARE MAINTENANCE

Every software product continues to evolve after its development through maintenance efforts. It can be stated that software maintenance is needed to correct errors and enhance features.

## Types of software maintenance are Corrective, Adaptive and Perfective.

## Software Reverse Engineering is the process of recovering the design and the requirements specification of a product from an analysis of its code

The first stage of reverse engineering usually focuses on carrying out cosmetic changes to the code to improve its readability, structure, and understandability, without changing of its functionalities.

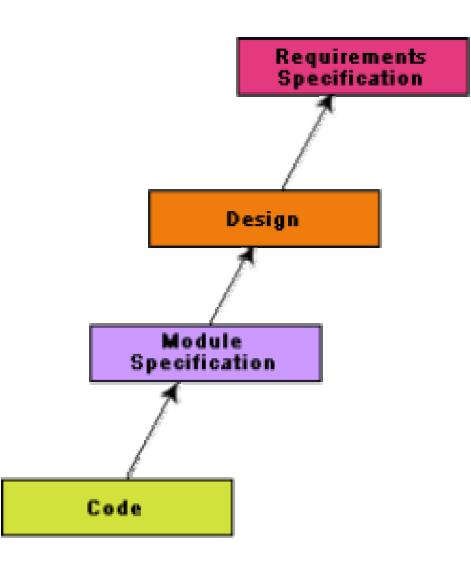


Fig. 24.1: A process model for reverse engineering

After the cosmetic changes have been carried out on a legacy software the process of extracting the code, design, and the requirements specification can begin

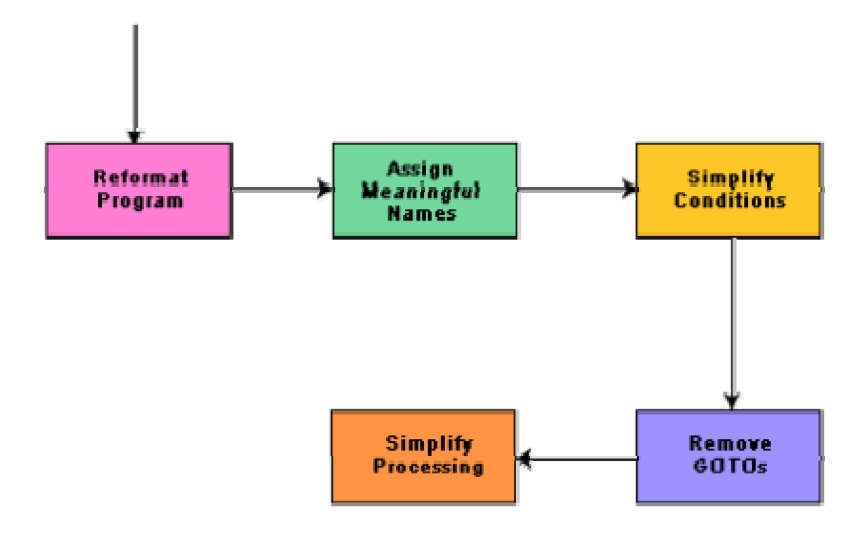


Fig. 24.2: Cosmetic changes carried out before reverse engineering.

**Legacy Software Product** prudently, is any software system that is hard to maintain.