Module: III

Software Implementation and Testing

Coding

- The goal of the coding or programming activity is to implement the design in the best possible manner.
- The coding activity affects both testing and maintenance.
- The time spent in coding is a small percentage of the total software cost, while testing and maintenance consume the major portion..

Coding

- During coding the programs should not be constructed so that they are easy to write, but in a manner that they are easy to read and understand

Programming principles and guidelines

- Writing solid code is a skill that can only be acquired by practice.
- However, based on experience, some general rules and guidelines can be given for the programmer.

Structured Programming

- Structured programming is often regarded as “goto-less” programming.
- A program has a static structure as well as a dynamic structure.
- The static structure is the structure of the text of the program, which is just a linear organization of statements of the program.

Structured Programming

- The dynamic structure of the program is the sequence of statements executed during the execution of the program.
- The static structure of a program is fixed, while the dynamic structure can change from execution to execution.

Structured Programming

- The goal of structured programming is to ensure that the static structure and the dynamic structures are the same.
- That is, the objective of structured programming is to write programs so that the sequence of statements executed during the execution of a program is the same as the sequence of statements in the text of that program

Structured Programming

- In structured programming, a statement is not a simple assignment statement, it is a structured statement.
- The key property of a structured statement is that it has a **single-entry** and a **single-exit.**
- That is, during execution, the execution of the (structured) statement starts from one defined

Structured Programming

- The most commonly used single-entry and single-exit statements are
- ***Selection:*** if B then S1 else S2

  if B then S1
- ***Iteration:*** While B do S

  repeat S until B
- ***Sequencing:*** S1; S2; S3;………..

Information hiding

- When a software is developed to solve a problem, the software uses some data structures to capture the information in the problem domain.
- When the information is represented as data structures, only some defined operations should be performed on the data structures.
- This is the principle of **information hiding**.

Information hiding

- The information captured in the data structures should be hidden from the rest of the system, and only the access functions on the data structures that represent the operations performed on the information should be visible.

Information hiding

- Information hiding can reduce the coupling between modules and make the system more maintainable. Most modern OO languages provide linguistic mechanisms to implement information hiding.

Some programming practices

- There are certain rules that have been found to make code easier to read as well as avoid some of the errors.
- **1.Control Constructs:** It is desirable that as much as possible single-entry, single-exit constructs be used.
- **2.Gotos:** Gotos should be used sparingly and in a disciplined manner. If a goto must be used, forward transfers (or a jump to a later statement) are more acceptable than a

Some programming practices

- **3.Information Hiding:** Only the access functions for the data structures should be made visible while hiding the data structure behind these functions.
- **4.User-Defined Types:** When user defined types like enumerated type are available, they should be exploited where applicable.
- **5.Nesting:** Where possible, deep

Some programming practices

- **6.Module Size:** Large modules often will not be functionally cohesive.
- **7.Module Interface:** Any module whose interface has more than five parameters should be carefully examined and broken into multiple modules with a simpler interface if possible.

Some programming practices

- **8.Side Effects:** Side effects caused by invocation of modules should be avoided where possible, and if a module has side effects, they should be properly documented.
- **9.Robustness:** A program is robust if it does something planned even for exceptional conditions. A program might encounter exceptional conditions in such forms as incorrect input, the incorrect value of some

Some programming practices

- If such situations do arise, the program should not just “crash” or “core dump”; it should produce some meaningful message and exit gracefully.
- **10.Switch Case with Default:** If there is no default case in a “switch” statement, the behavior can be unpredictable if that case arises at some point of time which was not predictable at development stage

Some programming practices

- **11.Empty Catch Block:** An exception is caught, but if there is no action, it may represent a scenario where some of the operations to be done are not performed. Whenever exceptions are caught, it is a good practice to take some default action, even if it is just printing an error message.
- **12.Empty if, while Statement:** A condition is checked but nothing is done based on the check. This often occurs

Some programming practices

- **13.Read Return to Be Checked:** Sometimes the result from a read can be different from what is expected, and this can cause failures later. For example, if read from scanf() is more than expected, then it may cause a buffer overflow.
- **14.Return from Finally Block:** One should not return from finally block, as it can create false beliefs.

Some programming practices

- **15.Trusted Data Sources:** Counter checks should be made before accessing the input data, particularly if the input data is being provided by the user or is being obtained over the network.
- **16.Give Importance to Exceptions:** To make a software system more reliable, a programmer should consider all possibilities and write suitable exception handlers to prevent

Coding standards

- Coding standards provide rules and guidelines for some aspects of programming in order to make code easier to read.
- They provide guidelines for programmers regarding naming, file organization, statements and declarations, and layout and comments.
- Some of the coding standards for

Naming Conventions

- 1.Package names should be in lowercase
- 2.Type names should be nouns and should start with uppercase (e.g., Day, DateOfBirth, EventHandler).
- 3.Variable names should be nouns starting with lowercase (e.g., name, amount).
- 4.Constant names should be all uppercase (e.g., PI,

Naming Conventions

- 5.Method names should be verbs starting with lowercase (e.g., getValue()).
- 6.Private class variables should have the _ suffix (e.g., “private int value_”).
- 7.Variables with a large scope should have long names; variables with a small scope can have short names; loop iterators should be named i, j, k,

Naming Conventions

- 8.The prefix ***is*** should be used for Boolean variables and methods to avoid confusion (e.g., isStatus should be used instead of status); negative Boolean variable names (e.g., isNotCorrect) should be avoided.
- 9.The term compute can be used for methods where something is being computed; the term find can be used where something is being looked up (e.g., computeMean(), findMin()).

Naming Conventions

- 10.Exception classes should be suffixed with *Exception* (e.g., OutOfBoundException).

Files

- Conventions for naming Java files:
- 1.Java source files should have the extension .java—this is enforced by most compilers and tools.
- 2.Each file should contain one outer class and the class name should be the same as the file name.

Files

- 3.Line length should be limited to less than 80 columns and special characters should be avoided. If the line is longer, it should be continued and the continuation should be made very clear.

Statements

- These guidelines are for the declaration and executable statements in the source code. Some examples are given below.
- 1.Variables should be initialized where declared, and they should be declared in the smallest possible scope.
- 2.Declare related variables together in a common statement. Unrelated variables should not be declared in

Statements

- 3.Class variables should never be declared public.
- 4.Use only loop control statements in a for loop.
- 5.Loop variables should be initialized immediately before the loop.

Statements

- 6.Avoid the use of **break** and **continue** in a loop.
- 7.Avoid the use of **do … while** construct.
- 8.Avoid complex conditional expressions—introduce temporary boolean variables instead.
- 9.Avoid executable statements in conditionals.

Commenting and Layout

- Comments are textual statements that are meant for the program reader to aid the understanding of code
- Comments should generally be provided for blocks of code, and in many cases, only comments for the modules need to be provided.

Commenting and Layout

- Comments for a module are often called **prologue** for the module, which describes the functionality and the purpose of the module, its public interface and how the module is to be used, parameters of the interface, assumptions it makes about the parameters, and any side effects it has. Other features may also be included

Commenting and Layout

- Java provides documentation comments that are delimited by “/** ... */”, and which could be extracted to HTML files.
- In addition to prologue for modules, coding standards may specify how and where comments should be located. Some such guidelines are:

Commenting and Layout

- 1.Single line comments for a block of code should be aligned with the code they are meant for.
- 2.There should be comments for all major variables explaining what they represent.
- 3.A block of comments should be preceded by a blank comment line with just “/*” and ended with a line containing just “*/”.
- Trailing comments after statements

Layout guidelines

- **Layout guidelines** focus on how a program should be indented, how it should use blank lines, white spaces, etc., to make it more easily readable.
- Indentation guidelines are sometimes provided for each type of programming construct.
- However, most programmers learn these by seeing the code of others and the code fragments in books and documents, and many of these have

Incrementally Developing Code

- The coding activity starts when some form of design has been done and the specifications of the modules to be developed are available.
- With the design, modules are assigned to developers for coding. Developers use different processes for developing code.

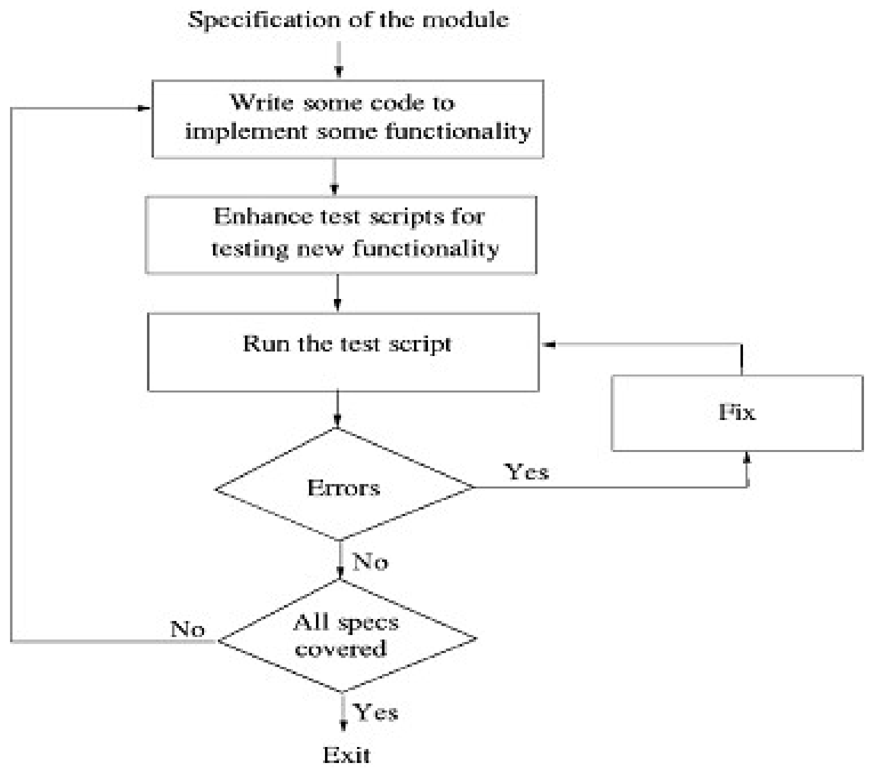
An Incremental Coding Process

- This method is usually followed by experienced developers in which code is developed incrementally.
- That is, write code for implementing only part of the functionality of the module.
- This code is compiled and tested with some quick tests to check the code that has been written so far.

An Incremental Coding Process

- When the code passes these tests, the developer proceeds to add further functionality to the code, which is then tested again. Here code is developed incrementally, testing it as it is built.

An Incremental Coding Process


Specification of the module
Write some code to implement some functionality
Enhance test scripts for testing new functionality
Run the test script
Fix
Errors
Yes
No
All specs covered
No
Yes
Exit

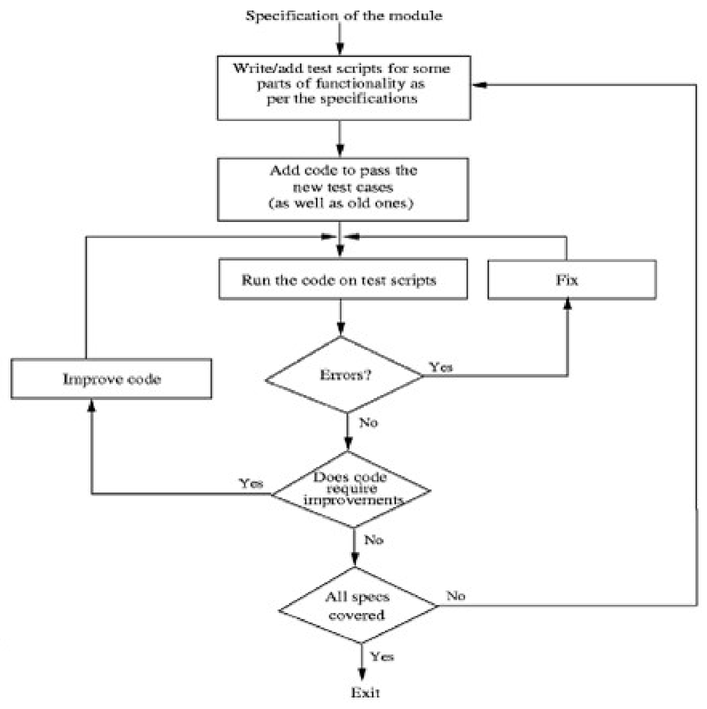
An Incremental Coding Proce

- The basic advantage of developing code incrementally is that once some error is found in some testing, it can be identified as somewhere in the latest code added after the last successful testing.
- For this purpose, it is essential that testing be done through test scripts. New test cases can be added easily to these test scripts whenever new functionality is added to the code

Test-Driven Development

- In Test-Driven development, instead of writing code and then developing test cases to check the code, a programmer first writes test scripts, and then writes the code to pass the tests, The whole process is done incrementally, with tests being written based on the specifications and code being written to pass the tests.
- This is a relatively new approach,

Test-Driven Development


Specification of the module
Write/add test scripts for some parts of functionality as per the specifications
Add code to pass the new test cases (as well as old ones)
Run the code on test scripts
Fix
Improve code
Errors?
Yes
No
Does code require improvements
Yes
No
All specs covered
No
Yes
Exit

Test-Driven Development

- In TDD, the test cases for higher-priority features are developed first and those for lower-priority features are developed later.
- Consequently, code for high-priority features will be developed first and lower-priority items will be developed later.
- This has the benefit that higher-priority items get done first, but has the drawback that some of the lower-

Pair Programming

- Pair programming is also a coding process that has been proposed as a key technique in extreme programming (XP) methodology.
- In pair programming, code is not written by individual programmers but by a pair of programmers.
- That is, the coding work is assigned not to an individual but to a pair of individuals. This pair together writes

Pair Programming

- In this process, one person will type the program while the other will actively participate and constantly review what is being typed.
- When errors are noticed, they are pointed out and corrected.
- The pair discusses the algorithms, data structures, or strategies to be used in the code to be written.
- The roles are rotated frequently making

Pair Programming

- In this method, instead of writing code and then getting it reviewed by another programmer, we have a programmer who is constantly reviewing the code being written.
- Besides, having two programmers result in better decisions being taken about the data structures, algorithms, interfaces, logic etc.
- Errors are also likely to be dealt with

Pair Programming

- The main drawback of pair programming is that it may result in loss of productivity by assigning two people for a programming task.

Code Inspection

- Code inspection can be viewed as “static testing” in which defects are detected in the code not by executing the code but through a manual process.
- Code inspection is a review of code by a group of peers following a clearly defined process. The basic goal of inspections is to improve the quality of code by finding defects.
- [illegible]

Code Inspection

- Code inspection is conducted by programmers and for programmers
- It is a structured process with defined roles for the participants.
- The focus is on identifying defects, not fixing them.
- Inspection data is recorded and used for monitoring the effectiveness of the inspection process.

Code Inspection

- Inspections are performed by a team of reviewers (or inspectors) including the author, with one of them being the moderator.
- The moderator has the overall responsibility to ensure that the review is done in a proper manner and all steps in the review process are followed.

Code Inspection

- The different stages in this process are:
- 1.planning
- 2. self-review
- 3.group review meeting
- 4.rework and follow-up.
- These stages are generally executed in a linear order.

Planning

- The objective of the planning phase is to prepare for inspection.
- An inspection team is formed, which should include the programmer whose code is to be reviewed.
- The team should consist of at least three people, though sometimes four-or-five member teams are also formed.
- A moderator is appointed.

Planning

- The author of the code ensures that the code is ready for inspection and the entry criteria are satisfied.
- Commonly used entry criteria is that the code compiles correctly.
- The moderator checks that the entry criteria are satisfied by the code.

Planning

- A package is prepared and distributed to the inspection team.
- The package typically consists of the code to be inspected, the specifications for which the code was developed, and the checklist that should be used for inspection.

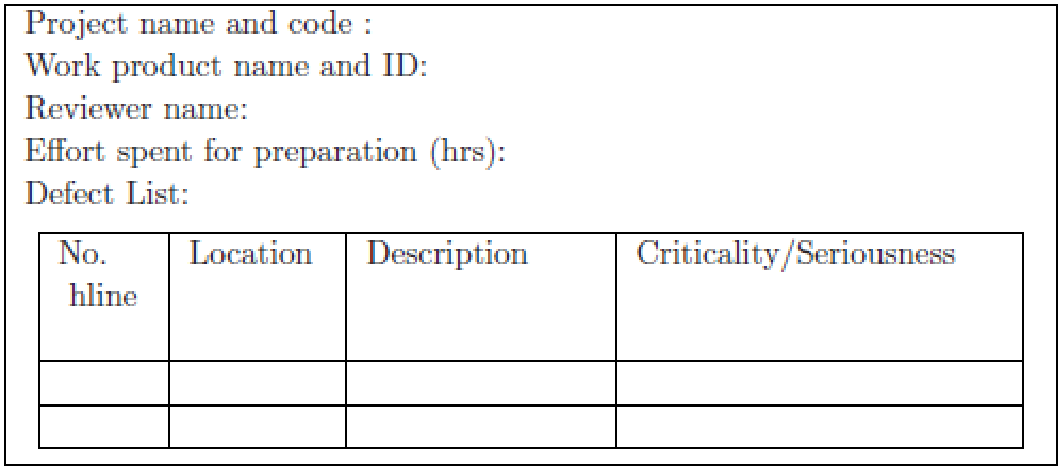
Planning

- The package for review is given to the reviewers.
- The moderator may arrange an opening meeting, if needed, in which the author may provide a brief overview of the product and any special areas that need to be looked at carefully.
- The type of defects the code inspection should focus on is

Self-Review

- In this phase, each reviewer does a self-review of the code.
- During the self-review, a reviewer goes through the entire code and logs all the potential defects he or she finds in the self-preparation log.
- An example form for self-preparation log is as shown.

Self-Review



Project name and code :
Work product name and ID:
Reviewer name:
Effort spent for preparation (hrs):
Defect List:

No. hline	Location	Description	Criticality/Seriousness

Group Review Meeting

- The basic purpose of the group review meeting is to come up with the final defect list, based on the initial list of defects reported by the reviewers and the new ones found during the discussion in the meeting.
- The main outputs of this phase are the defect log and the defect summary report.

Group Review Meeting

- The moderator first checks to see if all the reviewers are fully prepared.
- If everyone is ready, the group review meeting is held.
- A team member (called the reader) goes over the code line by line, and interprets each line to the team.
- At any line, if any reviewer finds any issue, he raises it.

Group Review Meeting

- There could be a discussion on the issue raised.
- The author accepts the issue as a defect or clarifies why it is not a defect.
- After discussion an agreement is reached and one member of the review team (called the scribe) records the identified defects in the defect log.

Group Review Meeting

- At the end of the meeting, the scribe reads out the defects recorded in the defect log for a final review by the team members.

Group Review Meeting

- The final defect log is the official record of the defects identified in the inspection and may also be used to track the defects to closure.
- For analysing the effectiveness of an inspection, however, only summary-level information is needed, for which a summary report is prepared.

Testing Concepts

- There are two types of approaches for identifying defects in the software—static and dynamic.
- In static analysis, the code is not executed but is evaluated through some process or some tools for locating defects (for e.g. Code inspections).
- In dynamic analysis, code is executed, and the execution is used for determining defects. Testing is the

Error

- It refers to the discrepancy between a computed, observed, or measured value and the true, specified, or theoretically correct value.
- That is, error refers to the difference between the actual output of a software and the correct output.

Fault

- Fault is a condition that causes a system to fail in performing its required function.
- Also called bug or defect.

Failure

- Failure is the inability of a system or component to perform a required function according to its specifications.
- Presence of error implies that a failure must have occurred, and the observance of a failure implies that a fault must be present in the system.
- Testing reveals the presence of faults. Debugging identifies the