

MODULE -4

- Software Testing Principles
- Program Inspections
- Program Walkthroughs
- Program Reviews

INTRODUCTION

- **Software testing** is a process, or a series of processes, designed to make sure computer code does what it was designed to do and, conversely, that it does not do anything unintended.
- Software should be predictable and consistent, presenting no surprises to users.

- Even a seemingly simple program can have hundreds or thousands of possible input and output combinations.
- Creating test cases for all of these possibilities is impractical.
- Complete testing of a complex application would take too long and require too many human resources to be economically feasible.

- When you test a program, you want to add some value to it.
- Adding value through testing means raising the quality or reliability of the program.
- Raising the reliability of the program means finding and removing errors.
- Therefore, don't test a program to show that it works; rather, start with the assumption that the program contains errors (a valid assumption for almost any program) and then test the program to find as many of the errors as possible.
- Thus, a more appropriate definition is this:

Testing is the process of executing a program with the intent of finding errors.

- Two of the most prevalent strategies include black-box testing and white-box testing

Black-Box Testing

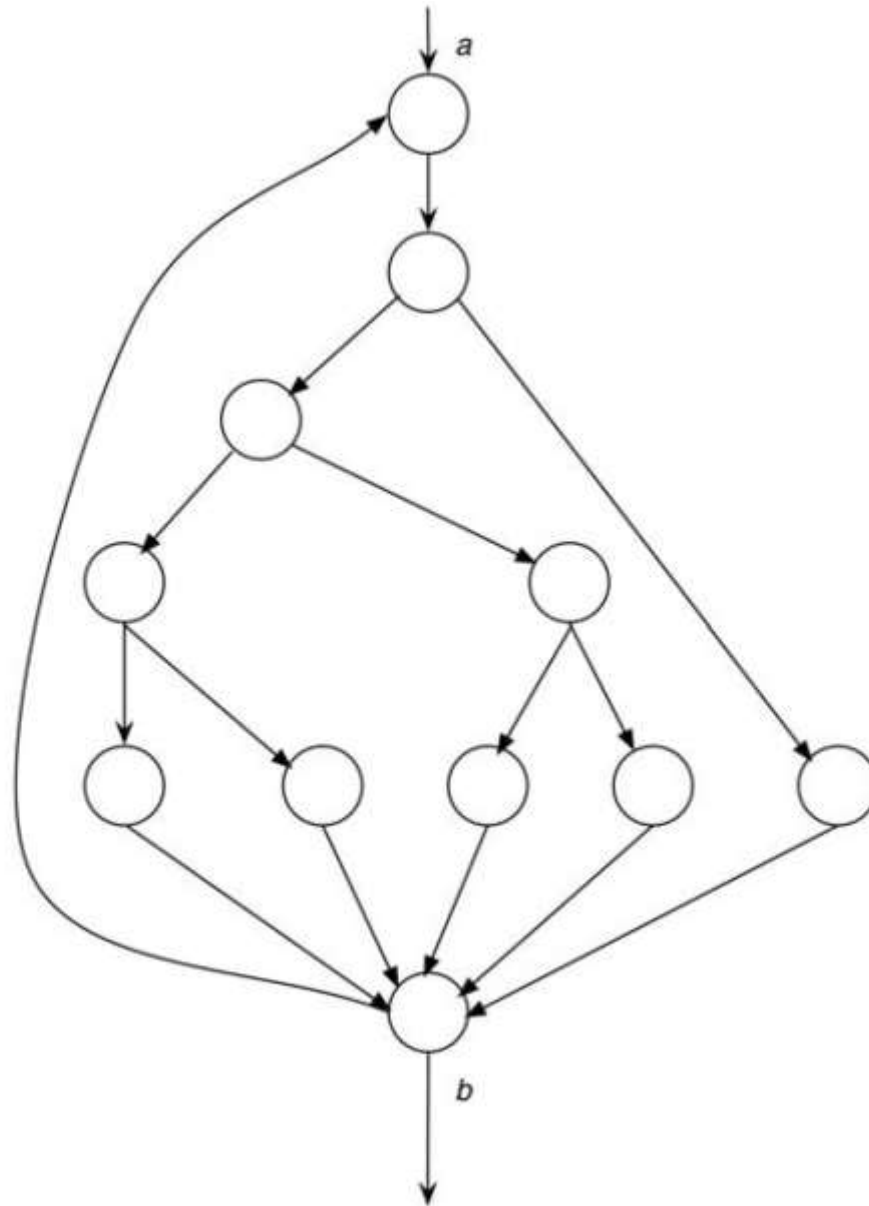
- One important testing strategy is black-box testing (also known as datadriven or input/output-driven testing).
- Your goal is to be completely unconcerned about the internal behavior and structure of the program. Instead, concentrate on finding circumstances in which the program does not behave according to its specifications.
- In this approach, test data are derived solely from the specifications (i.e., without taking advantage of knowledge of the internal structure of the program).

- If you want to use this approach to find all errors in the program, the criterion is exhaustive input testing, making use of every possible input condition as a test case.
- Exhaustive input testing is impossible.

White-Box Testing

- Another testing strategy, white-box (or logic-driven) testing, permits you to examine the internal structure of the program.
- This strategy derives test data from an examination of the program's logic .
- To find all errors in the program, the criterion is exhaustive input testing, making use of every possible input condition as a test case.
- The analog is usually considered to be exhaustive path testing.
- Exhaustive path testing-. That is, if you execute, via test cases, all possible paths of control flow through the program, then possibly the program has been completely tested.

Control-Flow Graph of a Small Program.



- The diagram depicts a CFG of a 10- to 20-statement program consisting of a DO loop that iterates up to 20 times. Within the body of the DO loop is a set of nested IF statements. Determining the number of unique logic paths is the same as determining the total number of unique ways of moving from point a to point b (assuming that all decisions in the program are independent from one another).
- This number is approximately 10^{14} , It is computed from $5^{20} + 5^{19} + \dots + 5^1$, where 5 is the number of paths through the loop body.
- Exhaustive path testing, like exhaustive input testing, appears to be impractical, if not impossible.

Software Testing Principles

TABLE 2.1 Vital Program Testing Guidelines

Principle Number	Principle
1	A necessary part of a test case is a definition of the expected output or result.
2	A programmer should avoid attempting to test his or her own program.
3	A programming organization should not test its own programs.
4	Any testing process should include a thorough inspection of the results of each test.
5	Test cases must be written for input conditions that are invalid and unexpected, as well as for those that are valid and expected.
6	Examining a program to see if it <i>does not do what it is supposed to do</i> is only half the battle; the other half is seeing whether the program <i>does what it is not supposed to do</i> .
7	Avoid throwaway test cases unless the program is truly a throwaway program.
8	Do not plan a testing effort under the tacit assumption that no errors will be found.
9	The probability of the existence of more errors in a section of a program is proportional to the number of errors already found in that section.
10	Testing is an extremely creative and intellectually challenging task.

Principle 1: A necessary part of a test case is a definition of the expected output or result.

- In spite of the proper destructive definition of testing, there is still a subconscious desire to see the correct result.
- One way of combating this is to encourage a detailed examination of all output by precisely spelling out, in advance, the expected output of the program.
- Therefore, a test case must consist of two components:
 1. A description of the input data to the program.
 2. A precise description of the correct output of the program for that set of input data.

Principle 2: A programmer should avoid attempting to test his or her own program.

- it's a bad idea to attempt to edit or proofread his or her own work.
- they really don't want to find errors in their own work
- After a programmer has constructively designed and coded a program, it is extremely difficult to suddenly change perspective to look at the program with a destructive eye.
- The program may contain errors due to the programmer's misunderstanding of the problem statement or specification.

- If this is the case, it is likely that the programmer will carry the same misunderstanding into tests of his or her own program.
- This does not mean that it is impossible for a programmer to test his or her own program. Rather, it implies that testing is more effective and successful if someone else does it.
- this argument does not apply to debugging (correcting known errors); debugging is more efficiently performed by the original programmer.

Principle 3: A programming organization should not test its own programs.

- The argument here is similar to that made in the previous principle.
- One reason for this is that it is easy to measure time and cost objectives, whereas it is extremely difficult to quantify the reliability of a program.
- Therefore, it is difficult for a programming organization to be objective in testing its own programs, because the testing process, if approached with the proper definition, may be viewed as decreasing the probability of meeting the schedule and the cost objectives.
- this does not say that it is impossible for a programming organization to find some of its errors, because organizations do accomplish this with some degree of success.
- Rather, it implies that it is more economical for testing to be performed by an objective, independent party.

Principle 4: Any testing process should include a thorough inspection of the results of each test.

- This is probably the most obvious principle, but again it is something that is often unnoticed.
- There are numerous experiments that show many subjects failed to detect certain errors, even when symptoms of those errors were clearly observable on the output listings.
- Put another way, errors that are found in later tests were often missed in the results from earlier tests.

Principle 5: Test cases must be written for input conditions that are invalid and unexpected, as well as for those that are valid and expected.

- There is a natural tendency when testing a program to concentrate on the valid and expected input conditions, to the neglect of the invalid and unexpected conditions.
- Test cases representing unexpected and invalid input conditions seem to have a higher error detection yield than do test cases for valid input conditions.

Principle 6: Examining a program to see if it does not do what it is supposed to do is only half the battle; the other half is seeing whether the program does what it is not supposed to do.

- This is a consequence to the previous principle.
- Programs must be examined for unwanted side effects.
- For instance, a payroll program that produces the correct pay checks is still an erroneous program if it also produces extra checks for non-existent employees

Principle 7: Avoid throwaway test cases unless the program is truly a throwaway program.

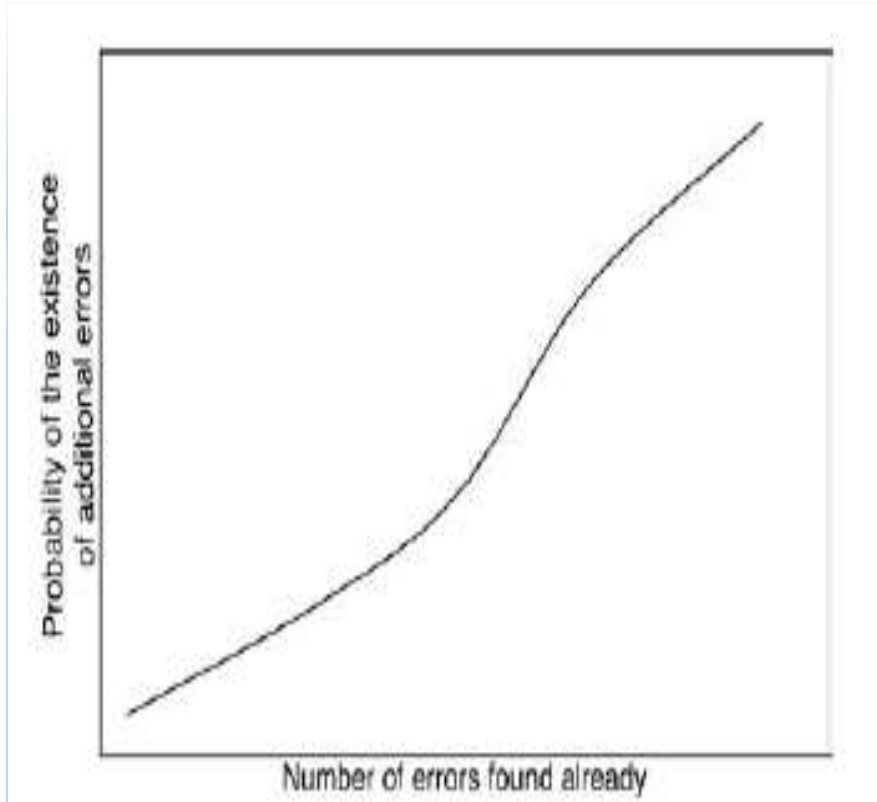
- Whenever the program has to be tested again (e.g., after correcting an error or making an improvement), the test cases must be reinvented.
- since this reinvention requires a considerable amount of work, people tend to avoid it.
- The retest of the program is rarely as rigorous as the original test, meaning that if the modification causes a previously functional part of the program to fail, this error often goes undetected.
- Saving test cases and running them again after changes to other components of the program is known as regression testing.

Principle 8: Do not plan a testing effort under the tacit assumption that no errors will be found.

- This is a mistake project managers often make and is a sign of the use of the incorrect definition of testing—that is, the assumption that testing is the process of showing that the program functions correctly.
- Once again, the definition of testing is the process of executing a program with the intent of finding errors.
- It is impossible to develop a program that is completely error free.
- Even after extensive testing and error correction, it is safe to assume that errors still exist; they simply have not yet been found.

Principle 9: The probability of the existence of more errors in a section of a program is proportional to the number of errors already found in that section.

- Errors tend to come in clusters and that some sections seem to be much more prone to errors than other sections.
- nobody has supplied a good explanation of why this occurs.



The Surprising Relationship between Errors Remaining and Errors Found.

Principle 10: Testing is an extremely creative and intellectually challenging task.

- The creativity required in testing a large program exceeds the creativity required in designing that program.

Inspections and Walkthroughs

- The three primary human testing methods are code inspections, walkthroughs and user (or usability) testing.
- Code inspections and walkthroughs are code-oriented methods.
- These methods can be used at virtually any stage of software development, after an application is deemed to be complete or as each module or unit is complete.
- The two code inspection methods have a lot in common (similarities).

Similarities are given below...

- Inspections and walkthroughs involve a team of people reading or visually inspecting a program.
- With either method, participants must conduct some preparatory work.
- The objective of the meeting is to find errors but not to find solutions to the errors—that is, to test, not debug.

- In a walkthrough, a group of developers—with three or four being an optimal number—performs the review.
- Only one of the participants is the author of the program.
- Therefore, the majority of program testing is conducted by people other than the author, which follows testing principle 2, which states that an individual is usually ineffective in testing his or her own program.

Code Inspections (Program Inspections)

- A **code inspection** is a set of procedures and error-detection techniques for group code reading.
- An **inspection team** usually consists of four people.
- The first of the four plays the role of **moderator** (quality-control engineer).
- The moderator is expected to be a competent programmer, but he or she is not the author of the program and need not be familiar with the details of the program.

- Moderator duties include:
 - Distributing materials for, and scheduling, the inspection session.
 - Leading the session.
 - Recording all errors found.
 - Ensuring that the errors are subsequently corrected.
- The second team member is the programmer.
- The remaining team members usually are the program's designer and a test specialist.

Inspection Agenda:-

- Several days in advance of the inspection session, the moderator distributes the program's listing and design specification to the other participants.
- The participants are expected to familiarize themselves with the material prior to the session.
- During the session, two activities occur:
 1. The programmer narrates, statement by statement, the logic of the program. During the discourse, other participants should raise questions, which should be pursued to determine whether errors exist. In other words, the simple act of reading aloud a program to an audience seems to be a remarkably effective error-detection technique.

2. The program is analysed with respect to checklists of historically common programming errors.

- The moderator is responsible for ensuring that the discussions proceed along productive lines and that the participants focus their attention on finding errors, not correcting them.
- The programmer corrects errors after the inspection session.
- Upon the conclusion of the inspection session, the programmer is given a list of the errors uncovered.
- List of errors is also analysed, categorized, and used to refine the error checklist to improve the effectiveness of future inspections.

List of some classical programming errors

1. Use of uninitialised variables.
2. Jumps into loops.
3. Non-terminating loops.
4. Incompatible assignments.
5. Array indices out of bounds.
6. Improper storage allocation and deallocation.
7. Mismatch between actual and formal parameter in procedure calls.
8. Use of incorrect logical operators or incorrect precedence among
9. operators.
10. Improper modification of loop variables.
11. Comparison of equality of floating point values.
12. Dangling reference caused when the referenced memory has not been allocated.

- The optimal amount of time for the inspection session appears to be from 90 to 120 minutes.
- The session is a mentally taxing experience, thus longer sessions tend to be less productive.
- Most inspections proceed at a rate of approximately 150 program statements per hour.
- For that reason, large programs should be examined over multiple inspections.

Walkthroughs

- The code walkthrough, like the inspection, is a set of procedures and error-detection techniques for group code reading.
- The main objective of code walkthrough is to discover the algorithmic and logical errors in the code.
- Like the inspection, the walkthrough is an uninterrupted meeting of one to two hours in duration.
- The walkthrough team consists of three to five people.

- One of these people plays a role similar to that of the moderator in the inspection process; another person plays the role of a secretary (a person who records all errors found); and a third person plays the role of a tester. Of course, the programmer is one of those people. Suggestions as to who the three to five people should be vary.
- Code walkthrough is an informal code analysis technique.
- The main objective of code walkthrough is to discover the algorithmic and logical errors in the code.

- The initial procedure is identical to that of the inspection process.
- Guidelines for the code walk through
 - The team performing code walkthrough should not be either too big or too small.
 - Ideally, it should consist of between three to seven members.
 - Discussions should focus on discovery of errors and avoid deliberations on how to fix the discovered errors.
 - In order to foster co-operation and to avoid the feeling among the engineers that they are being watched and evaluated in the code
 - walkthrough meetings, managers should not attend the walkthrough meetings.

- The designated tester comes to the meeting armed with a small set of paper test cases—representative sets of inputs (and expected outputs) for the program .
- During the meeting, each test case is mentally executed; that is, the test data are “walked through” the logic of the program.
- The state of the program (i.e., the values of the variables) is monitored on paper or a whiteboard.
- The test cases themselves do not play a critical role; rather, they serve as a vehicle for questioning the programmer about the logic and assumptions.

Peer Ratings (Peer Reviews or Program Reviews)

- The last human review process is not associated with program testing (i.e., its objective is not to find errors).
- Peer rating is a technique of evaluating anonymous programs in terms of their overall quality, maintainability, extensibility, usability, and clarity.
- The purpose of the technique is to provide programmer self evaluation.
- A programmer is selected to serve as an administrator of the process.
- The administrator, in turn, selects approximately 6 to 20 participants (6 is the minimum to preserve anonymity).
- The participants are expected to have similar backgrounds.

- Each participant is asked to select two of his or her own programs to be reviewed.
- One program should be representative of what the participant considers to be his or her finest work; the other should be a program that the programmer considers to be poorer in quality.
- Once the programs have been collected, they are randomly distributed to the participants.
- Each participant is given four programs to review. Two of the programs are the “finest” programs and two are “poorer” programs, but the reviewer is not told which is which.
- Each participant spends 30 minutes reviewing each program and then completes an evaluation form.
- After reviewing all four programs, each participant rates the relative quality of the four programs.

- The evaluation form asks the reviewer to answer, on a scale from 1 to 10 (1 meaning definitely yes and 10 meaning definitely no), such questions as:
- Was the program easy to understand?
- Was the high-level design visible and reasonable?
- Was the low-level design visible and reasonable?
- Would it be easy for you to modify this program?
- Would you be proud to have written this program?
- The reviewer also is asked for general comments and suggested improvements.

- After the review, the participants are given the anonymous evaluation forms for their two contributed programs.
- They also are given a statistical summary showing the overall and detailed ranking of their original programs across the entire set of programs, as well as an analysis of how their ratings of other programs compared with those ratings of other reviewers of the same program.
- The purpose of the process is to allow programmers to self-assess their programming skills.