Chapter Seven

Software Quality Assurance

An overview of testing

- **Error, Fault, and Failure**: the IEEE definitions for these terms
  - The term error is used in two different ways.
    - It refers to the discrepancy between a computed, observed, or measured value and the true, specified, or theoretically correct value.
    - Error is also used to refer to human action that results in software containing a defect or fault. This definition is quite general and encompasses all the phases.
  - Fault is a condition that causes a system to fail in performing its required function.
    - A fault is the basic reason for software malfunction and is synonymous with the commonly used term *bug.*
  - Failure is the inability of a system or component to perform a required function according to its specifications.
    - A software failure occurs if the behavior of the software is different from the specified behavior.
    - A failure is produced only when there is a fault in the system.

- Presence of a fault does not guarantee a failure.
- In other words, faults have the potential to cause failures and their presence is a necessary but not a sufficient condition for failure to occur.
- Note that the definition does not imply that a failure must be observed.
- It is possible that a failure may occur but not be detected.

- What is called a "failure" is dependent on the project, and its exact definition is often left to the tester or project manager.
- For example,
- is a misplaced line in the output a failure or not? Clearly, it depends on the project; some will consider it a failure and others will not.
- Take another example.
  - If the output is not produced within a given time period, is it a failure or not?
  - For a real-time system this may be viewed as a failure, but for an operating system it may not be viewed as a failure.

- Presence of an error (in the state) implies that a failure must have occurred, and the observance of a failure implies that a fault must be present in the system.
- However, the presence of a fault does not imply that a failure must occur.
- The presence of a fault in a system only implies that the fault has a potential to cause a failure to occur

- Whether a fault actually manifests itself in a certain time duration depends on many factors.
- This means that if we observe the behavior of a system for some time duration and we do not observe any errors, we cannot say anything about the presence or absence of faults in the system.
- If, on the other hand, we observe some failure in this duration, we can say that there are some faults in the system.

Testing Concepts

- Debugging is the process of finding out where something went wrong and correcting the code to eliminate the errors or bugs that cause unexpected results.
- Errors can be :
  - Language (syntax): errors result from incorrectly constructed code, such as an incorrectly typed keyword or some necessary punctuation omitted. These are the easiest types of errors to detect. For the most part needs debugging tools to detect them.
  - Runtime errors: occurs and detected as the program running, when a statement t attempts an operation that is impossible to carry out.
  - Logic errors occur when code does not perform the way you intended. The code might be syntactically valid and run with out performing any invalid operations and yet produce incorrect results.

- The elimination of the syntactical bug is the process of debugging , whereas the detection and elimination of the logical bug is the process of testing.
- Quality assurance testing can be:
  - **Error- based testing:** techniques search a given class method for particular clues of interests then describe how these clues should be tested.
    - Say we want to test the payrollComputation method of an Employee class: anEmployee.computePayroll(hours). To test this method, we must try different values for hours (say 40,0,100 and 10) to see if the program can handle them.
  - **Scenario based testing (usage based testing):** concentrates on what the user does, not what the product does. This means capturing use cases and the tasks users perform, then performing them and their variants as tests.

Test Cases and Test Criteria

- Having test cases that are good at revealing the presence of faults is central to successful testing.
- The reason for this is that if there is a fault in a program, the program can still provide the expected behavior for many inputs.
  - Hence, it is fair to say that testing is as good as its test cases.

- Ideally, we would like to determine a set of test cases such that successful execution of all of them implies that there are no errors in the program.
- This ideal goal cannot usually be achieved due to practical and theoretical constraints.
- Each test case costs money, as effort is needed to generate the test case, machine time is needed to execute the program for that test case, and more effort is needed to evaluate the results.

- Therefore, we would also like to minimize the number of test cases needed to detect errors.
- These are the two fundamental goals of a practical testing activity—maximize the number of errors detected and minimize the number of test cases (i.e., minimize the cost).

Testing strategies
Black-Box Testing

- Also called functional or behavioral testing.
- Test cases are decided solely on the basis of the requirements or specifications of the program or module
    - the internals of the module or the program are not considered for selection of test cases.
- the basis for deciding test cases in functional testing is the requirements or specifications of the system or module

- There are no formal rules for designing test cases for functional testing.
- But there are a number of techniques or heuristics that can be used to select test cases that have been found to be very successful in detecting errors.

White-Box Testing

- Is also called structural testing
- It is concerned with testing the implementation of the program.
- The intent of this testing is not to exercise all the different input or output conditions but to exercise the different programming structures and data structures used in the program.

- Other testing strategies
  - Bottom up testing
  - Top down testing

Test Case Template

- IEEE Standard 610 (1990) defines test case as follows:
- (1) A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement.
- (2) (IEEE Std 829-1983) Documentation specifying inputs, predicted results, and a set of execution conditions for a test item.

Test Case #:	**Test Case Name:**	**Page: 1 of ..**
System:	**Subsystem:**	
Designed by:	**Design Date:**	
Executed by:	**Execution Date:**	
Short Description:		

Pre-conditions

Step	Action	Expected System Response	Pass/ Fail	Comment
1				
2				

Post-conditions

The End