BEST PRACTICE
Specific Testing Techniques

There are numerous specific testing techniques. Some can be performed using test tools. A discussion of the more commonly used specific techniques follow:

- White-Box Testing
- Black-Box Testing
- Incremental Testing
- Thread Testing
- Requirements Tracing
- Desk Checking and Peer Review
- Walkthroughs, Inspections, and Reviews
- Proof of Correctness Techniques
- Simulation
- Boundary Value Analysis
- Error Guessing and Special Value Analysis
- Cause-Effect Graphing
- Design-Based Functional Testing
- Coverage-Based Testing
- Complexity-Based Testing
- Statistical Analyses and Error Seeding
- Mutation Analysis
- Flow Analysis
- Symbolic Execution

White-Box Testing
Assumes that the path of logic in a unit or program is known. The following are white-box testing techniques:

- Statement Coverage
- Decision Coverage
- Condition Coverage
- Decision/Condition Coverage
- Multiple Condition Coverage

Black-Box Testing
Focuses on testing the function of the program or application against its specification. Three successful techniques for managing the amount of input data required include:

- Equivalence Partitioning
- Boundary Analysis

- Error Guessing

Incremental Testing
Disciplined method of testing the interfaces between unit-tested programs as well as between system components. There are two types of incremental testing:

- Top-down
- Bottom-up

Thread Testing
Often used during early integration testing, demonstrates key functional capabilities by testing a string of units that accomplish a specific function in the application.

Requirements Tracing
The primary goal of software testing is to prove that the user or stakeholder requirements are actually delivered in the final product developed. This can be accomplished by tracing these requirements, both functional and non-functional,

Desk Checking and Peer Review
Desk checking is the most traditional means for analyzing a program. It is the foundation for the more disciplined techniques of walkthroughs, inspections, and reviews.

Walkthroughs, Inspections, and Reviews
Inspection involves a step-by-step reading of the product, with each step checked against a predetermined list of criteria.

Walkthroughs differ from inspections in that the programmer does not narrate a reading of the product by the team, but provides test data and leads the team through a manual simulation of the system. Walkthroughs and inspections should again be performed at the preliminary and detailed design stages.

Design reviews and audits are commonly performed as stages in software development as follows:

- System Requirements Review
- System Design Review
- Preliminary Design Review
- Final Design Review
- Final Review

Proof of Correctness Techniques

These techniques usually consist of validating the consistency of an output "assertion" (specification) with respect to a program (or requirements or design specification) and an input assertion (specification).

Simulation
Simulation is used in real-time systems development where the "real-world" interface is critical and integration with the system hardware is central to the total design. In many non real-time applications, simulation is a cost effective verification and test-data generation technique. Simulation also plays a useful role in determining the performance of algorithms.

Boundary Value Analysis
Partition the program domain in some meaningful way so that input data sets, which span the partition, can be determined.

Error Guessing and Special Value Analysis
Zero input values and input values that cause zero outputs are examples of where a tester may guess an error could occur.

Cause-Effect Graphing
Technique for developing test cases for programs from the high-level specifications.

Design-Based Functional Testing
Functional analysis can be extended to functions used in the design process. A distinction can be made between requirements functions and design functions. Requirements functions describe the overall functional capabilities of a program. In order to implement a requirements function it is usually necessary to invent other "smaller functions." These other functions are used to design the program.

Coverage-Based Testing
Most coverage metrics are based on the number of statements, branches, or paths in the program, which are exercised by the test data. Such metrics can be used both to evaluate the test data and to aid in the generation of the test data.

Complexity-Based Testing
These are cyclomatic complexity and software complexity measures. These and many other metrics are designed to analyze the complexity of software systems,

Statistical Analyses and Error Seeding
The most common type of test data analysis is statistical. An estimate of the number of errors in a program can be obtained from an analysis of the errors uncovered by the test data.

Mutation Analysis

This method rests on the competent programmer hypothesis that states that a program written by a competent programmer will be, after debugging and testing, "almost correct." The basic idea of the method is to seed the program to be tested with errors, creating several mutants of the original program.

Flow Analysis

The control flow graph is used to analyze the program behavior, to locate instrumentation breakpoints, to identify paths, and in other static analysis activities.

Symbolic Execution

Method of symbolically defining data that forces program paths to be executed.

References

Guide – CSTE Common Body Of Knowledge, V6.1