Categories of Test Techniques
and Their Characteristics

Purpose of test technique:

to identify test conditions, test cases, and test data

Choice of technique is based on factors:

- component / system complexity
- regulatory standards
- customer or contractual requirements
- risk levels and types
- available documentation
- tester knowledge and skills
- available tools
- time and budget
- software development lifecycle model
- the types of defects expected in the component / system

Formality:

the use of test technique can range from very informal to very formal

Different test techniques can be used **in combination** for creation of test cases to get the best result from the test effort.

Black-box test techniques

(behavioral or behavior-based techniques)

Based on an **analysis of the appropriate test basis** (formal requirements documents, specifications, use cases, user stories, or business processing).

Applicable in both **functional and non-functional testing**.

Concentrate on **inputs and outputs of the test object** without reference to its internal structure.

Common characteristics:

- test conditions, test cases, and test data are derived from a test basis that may include software requirements, specifications, use cases, and user stories

- test cases may be used to detect gaps between requirements and the implementation of the requirements, as well as deviation from the requirements

- coverage is measured based on the items tested in the test basis and the technique applied to the test basis

White-box test techniques

(structural or structure-based techniques)

Based on an **analysis of the architechture**, **detailed design**, **internal structure**, or **the test object**.

Concentrate on **the structure and processing within the test object**.

Common characteristics:

- test conditions, test cases, and test data are derived from a test basis that may include code, software architecture, detailed design, or any other source of information regarding the structure of the software

- coverage is measured based on the items tested within a selected structure (the code or interfaces) and the technique applied to the test basis

Experience-based test techniques

Leverage **the experience of developers, testers and users** to design, implement, and execute tests.

Often **combined** with black-box and white-box test techniques.

Common characteristics:

- test conditions, test cases, and test data are derived from a test basis that may include knowledge and experience of testers, developers, users and other stakeholders

The knowledge and experience includes expected use of the software, its environment, likely defects, and the distribution of those defects.

Black-box test techniques

(behavioral or behavior-based techniques)

Equivalence Partitioning

Divides data into partitions (eq.classes, both valid and invalid). All the members of a given partition are expected to be processed in the same way.

- Valid values (valid eq.partition) should be accepted by the component of system.

- Invalid values (invalid eq.partition, should be tested individually, not combined with other invalid eq.partitions) should be rejected.

- Partition can be identified for any data element related to the test object.

- Any partition may be divided into sub partitions.

- Each value must belong to one and only one eq.partition.

Applicable at all test levels.

Boundary Value Analysis (BVA)

It is an extension of eq.partitioning. Can only be used when the partition is prdered, consisting of numeric or sequential data. The min. and max. values of a partition are its boundary values.

Behavior at the boundaries of eq.partitions is more likely to be incorrect than behavior within the partitions.

Can be applied at any test levels. Generally used to test requirements that call for a range of numbers (including dates and times).

Decision Table Testing

A way to record complex rules that a system must implement. The tester identifies conditions (input) and the result of actions (output) of the system.

Helps:

- to identify all the important combinations of conditions, some of which might otherwise be overlooked;

- to find any gaps in the requirements.

May be applied to all situations in which the behavior of the software depends on a combination of conditions, at any test level.

State Transition Testing

A state transition model has 4 basic parts:

- The **states** that the software may occupy.

- The **transitions** from one state to another.

- The **events** that cause a transition –> **Inputs**.

- The **actions** that result from a transition –> **Output**.

State transition tables show all valid transitions and potentially invalid transitions between states, as well as the events, and resulting actions for valid transitions. **State transition diagrams** normally show only the valid transitions and exclude the invalid transitions.

State transition testing is used for menu-based applications and is widely used within the ebmedded software industry. Though it depends more on the way of software realization: state machine, which is an abstract machine that can be in exactly one of a finite number of states at any given time.

It is also suitable for modeling a business scenario having specific states or for testing screen navigation. *Can be applied at all test levels (my guess).*

Use Case Testing

Use cases are specific way of designing interactions with software items. They incorporate requirements for the software functions. They are associated with actors (users, external hardware) and subjects (component or system). Interactions may be represented grafically by work flows, activity diagrams, or business process models.

Tests based on use cases are designed to exercise the defined behaviors: basic, exeptional or alternative, and error handling. *Can be applied at all test levels except component testing (my guess).*

White-box test techniques

(structural or structure-based techniques)

Statement Testing

Exercises the potential executable statements in the code.

Decision Testing

Exercises the decisions in the code and tests the code that is executed based on the decisions outcomes.

Test cases follow the control flows that occur from a decision point:

- for IF statement, one for the true outcome and one for the false outcome;

- for CASE statement, test cases would be required for all the possible outcomes, including the default outcome.

The Value of Statement and Decision Testing

Both of these testing techniques can be used at all test levels, but the are commonly used at the component test level.

Statement testing may provide less coverage than decision testing: in 100% statement coverage all executed statements in the code have been tested at least once, but it does not ensure that all decision logic has been tested. When 100% decision coverage is achieved, it executes all decision outcomes, which includes testing the true outcome and also the false outcome, even when there is no explicit false statement.

Statement coverage helps to find defects in code that was not exercised by other tests. Decision coverage helps to find defects in code where other tests have not taken both true and false outcomes.

Achieving 100% decision coverage guarantees 100% statement coverage, but NOT vice versa.

Experience-based test techniques

With these techniques, the test cases are derived from the tester's skill and intuition, and their experience with similar applications and technologies. They can be helpful in identifying tests that were not easily identified by other more systematic techniques.

Depending on the tester's approach and experience, these techniques may achieve widely varying degrees of coverage and effectiveness.

Coverage can be difficult to assess and may not be measurable with these techniques.

Error Guessing

Used to anticipate the occurance of errors, defects, and failures, based on the tester's knowledge, including:

- how the application has worked in the past

- what kind of errors tend to be made

- failures that have occured in other applications

Methodological approach: to create a list of errors, defects, and failures, and then design tests that will expose those failures and the defects that caused them.

Exploratory Testing

Informal (not pre-defined) tests are designed, executed, logged, and evaluated dynamically during test execution. The test results are used to learn more about the component or system, and to create tests for the areas that may need more testing.

Sometimes conducted using session-based testing to structure the activity: within a defined time-box, and the tester uses a test charter containing test objectives to guide the testing. Using of test session sheets is possible to document the steps followed and the discoveries made.

It is the most useful technique in the absence of adequate specifications or in terms of time pressure. It is usefull to complement other more formal testing techniques.

It is associated with reactive test strategies. It can incorporate the use of other black-box, white-box, and experience-based techniques.

Checklist-based Testing

Testers design, implement, and execute tests to cover test conditions found in checklists. Testers can create a new checklists or expand an existing checklists if necessary.

Checklists are based on experience, knowledge about what is important for the user, or an understanding of why and how software fails.

Checklists can be created for various test types, including functional and non-functional testing.

In the absence of detailed test-cases, this technique can provice guidelines and a degree of consistency.

Checklists are high-level lists, so some variability in the actual testing is likely to occur. However, this may result in greater coverage, but less repeatability.