# Verification and Validation

**Verification** checks that the program conforms to its specification.

- Are we building the product right?

**Validation** checks that the program as implemented meets the expectations of the user.

- Are we building the right product?

## Verification and validation

- Interoperability Issues.
- Availability Issues.
- Integrity Issues.
- Maintenance Issues.
- Reliability Issues.
- Usability Issues.

## Verification and Validation

- Correctness Issues.
- Efficiency Issues.
- Flexibility Issues.
- Portability Issues.
- Reusability Issues.
- Testability Issues.

## Static Verification

- Program inspection
- Formal method

## Testing

- **Failure**: The departure of program operation from user requirements.

- **Fault**: A defect in a program that may cause a failure.

- **Error**: Human action that results in software containing a fault.

## Types of Faults

- algorithmic faults
- computation and precision faults
- documentation faults
- stress or overloaded faults
- capacity or boundary faults
- timing or coordination faults
- throughput or performance faults

## IBM Orthogonal Defect Classification

**Function:** fault that affects capability, end-user interfaces, product interface with hardware architecture, or global data structure.

**Interface:** fault in interfacing with other components or drives via calls, macros, control blocks, or parameter lists.

**Checking:** fault in program login that fails to validate data and values properly before they are used.

**Assignment:** fault in data structure or code block initialization.

**Timing/serialization:** fault that involves timing of shared and real-time resources.

**Build/package/merge:** fault that occurs because of problems in repositories, management changes or version control.

**Documentation:** fault that affects publications and maintenance notes.

**Algorithm:** fault involving efficiency or correctness of algorithm or data structure but not design.

# The Testing Process

1. Unit testing
2. Component testing
3. Integration testing
4. System testing
5. Acceptance testing

## Testing Strategies

1. Top-down testing
2. Bottom-up testing
3. Thread testing
4. Stress testing
5. Back-to-back testing

## Defect Testing

- Black-box testing
- Interface testing
- Structural testing

## Black-Box Testing

- Graph-based testing methods
- Equivalence Partitioning
- Boundary value analysis

## Graph-Based Testing

- Transaction flow modeling
- Finite state modeling
- Data flow modeling
- Timing modeling

## Partition Testing

1. If an input condition specifies a *range*, one valid and two invalid equivalence classes are defined.

2. If an input condition requires a specific *value*, one valid and two invalid equivalence classes are defined.

3. If an input condition specifies a member of a *set*, one valid and one invalid class are defined.

4. If an input condition is *boolean*, one valid and one invalid class are defined.

## Boundary Value Analysis

1. If an input condition specifies a *range* bounded by values *a* and *b*, test cases should be designed with values *a* and *b*, just above and just below *a* and *b*, respectively.

2. If an input condition specifies a number of values, test cases should be developed that exercise the minimum and maximum numbers. Values just above and below minimum and maximum are also tested.

3. Apply guidelines 1 and 2 to output conditions.

4. If internal program data structures have prescribed boundaries, be certain to design a test case to exercise the data structure at its boundary.

## Interface Testing

- Parameter interfaces
- Shared memory interfaces
- Procedural interfaces
- Message passing interfaces

## Integration testing

- top-down integration
- bottom-up integration
- incremental testing
- regression testing

## White-box testing

1. Statement Coverage Criterion
2. branch coverage coverage criterion
3. Data flow coverage criterion
4. Path coverage criterion

## Mutation testing

- A program under test is seeded with a single error to produce a "mutant" program.

- A test covers a mutant if the output of the mutant and the program under test differ for that test input.

- The mutation coverage measure for a test set is the ratio of mutants covered to total mutants.

## Debugging

**Static slicing:** decomposes a program by statically analyzing data-flow and control flow of the program.

- A static program slice for a given variable at a given statement contains all the executable statements that could influence the value of that variable at the given statement.

- The exact execution path for a given input is a subset of the static program slice with respect to the output variables at the given checkpoint.

- Focus is an automatic debugging tool based on static program slicing to locate bugs.

## Dynamic Program Slicing

**dynamic data slice:** A dynamic data slice with respect to a given expression, location, and test case is a set of all assignments whose computations have propagated into the current value of the given expression at the given location.

**dynamic control slice:** A dynamic control slice with respect to a given location and test case is a set of all predicates that enclose the given location.

**SPYDER:** is an automatic debugging tool based on dynamic program slicing to locate bugs.