# Testing

Ran Liao

May 23, 2019

---

## Terminology

- **Fault**

  A *fault* is injected into a software product when a human makes a mistake.

- **Failure**

  A *failure* is the observed incorrect behavior of the software product as a consequence of a fault.

- **Error**

  The *error* is the amount by which a result is incorrect.

- **Defect**

  The word *defect* is a generic term for a fault, failure, or error.

- **Quality**

  The *quality* of software is the extent to which the product satisfies its specifications.

## Categorized by Timing

- **Unit testing**

  Each component is tested as soon as it has been implemented.

- **Integration testing**

  At the end of each iteration, the completed components are combined and tested.

- **Product testing**

  When the product appears to be complete, it is tested as a whole.

- **Acceptance testing**

  Once the completed product has been installed on the client's computer, the client tests it.

## Categorized by Methdology

- **Non-execution-based testing**

  Code Walkthroughs and Inspections.

- **Execution-based testing**

  - **Test to Specifications / Black-box / Data-driven**
    Ignore the code and use the specifications to select test cases.
    The combinatorial explosion makes testing to specifications impossible.
  - **Test to Code / Glass-box / Logic-driven**
    Ignore the specifications and use the code to select test cases.
    The combinatorial explosion makes testing to specifications impossible. And it is not reliable, we can exercise every path without detecting every fault.

## Equivalence Testing

Any one member of an equivalence class is as good a test case as any other member of the equivalence class. And in addition to input specifications, we also need to perform equivalence testing of the output specifications.

## Boundary Value Analysis

Select test cases on or just to one side of the boundary of equivalence classes. This can greatly increases the probability of detecting a fault.

## Combined Approach

Equivalence classes together with boundary value analysis to test both input specifications and output specifications. This approach generates a small set of test data with the potential of uncovering a large number of faults.

## Glass-Box Unit-Testing

- **Statement Coverage**

  Running a series of test cases during which every statement is executed at least once.

- **Branch Coverage**

  Running a series of tests to ensure that all branches are tested at least once.

- **Condition Coverage**

  Condition coverage is also known as **Predicate Coverage** in which each one of the Boolean expression have been evaluated to both TRUE and FALSE.

- **Branch-condition Coverage**

  Every point of entry and exit in the program has been invoked at least once, every condition in a decision in the program has taken all possible outcomes at least once, and every decision in the program has taken all possible outcomes at least once.

- **Path Coverage**

  Testing all paths.

- **Linear Code Sequences**

  Identify the set of points L from which control flow may jump, plus entry and exit points. Restrict test cases to paths that begin and end with elements of L. This uncovers many faults without testing every path.

- **All-Definition-Use-Path Coverage**

  Identify all paths from the definition of a variable to the use of that definition. And a test case is set up for each such path.

- **Infeasible Code**

  We may have an infeasible path ("dead code") in the artifact. Frequently this is evidence of a fault.