Software Testing

- Defect Testing
    - Testing programs to establish the presence of system defects
    - Objectives
        - To understand testing techniques that are geared to discover program faults
        - To introduce guidelines for interface testing
        - To understand specific approaches to object-oriented testing
        - To understand the principles of CASE tool support for testing

    - Topics Covered
        - Defect testing
        - Integration Testing
        - Object-Oriented testing
        - Testing workbenches

    - The Testing Process
        - Component Testing
            - Testing of individual program components
            - Usually the responsibility of the component developer (except sometimes for critical systems)
            - Tests are derived from the the developer's experience

        - Integration Testing
            - Testing of groups of components integrated to create a system or sub-system
            - The responsibility of an independent testing team
            - Tests are based on a system specification

    - The goal of defect testing is to discover defects in programs
    - Defect Removal Efficiency
        - DRE = E / (E + D)
            - E is the number of errors found before delivery
            - D is the number of defects found after delivery

    - Testing Priorities
        - Only **exhaustive** testing can show a program free from defects
            - However, exhaustive testing is impossible

        - Tests should exercise a system's **capabilities** rather than its components
        - Testing old capabilities is more important than testing new capabilities

    - Testing **typical situations** is more important than boundary value cases

  - The more errors found, more likely there are more errors
    - Redesign when this starts happening

  - Exhaustive Testing
    - With code with any level of complexity, the possible code paths grows to be too large to test

  - Test Data and Test Cases
    - Test Data
      - Inputs which have been devised to test the system

    - Test Cases
      - Inputs to test the system and the predicted outputs from these inputs if the system operates according to its specification

    - Black Box Testing
      - Based completely on system specification
      - No knowledge of the implementation
      - Equivalence Partitioning
        - Input and results often fall into different classes where all members of a class are related
        - Test cases should be chosen from each partition
        - Examples
          - < 0, 0, 1, >10,000, etc

    - Structural Testing
      - Sometimes called **white-box** testing/**clear-box** testing
      - Derivation of test cases according to program structure
      - Objective is to exercise **all program statements** (not all path combinations)
      - Knowledge of the program is used to identify test cases

    - Path Testing
      - Each path through the program is executed at least once
      - Different paths can be visualized as nodes in a **flow graph**
        - Need to cover each edge in flow graph

      - Cyclomatic Complexity
        - (number of edges) - (number of nodes) + 2
        - Number of test cases required

      - All paths are executed, but not all combinations are tested

- Integration Testing
    - Tests complete systems or subsystems composed of integrated components
    - Integration testing should be black box testing with tests derived from the specification
    - Main difficulty is localising erros
    - Incremental integration testing reduces the problem
    - Approaches
        - Top-Down
        - Bottom-up
            - Test smallest, lowest level pieces
        - In practice, most integration involves a combination of these strategies
- Other Testing Issues
    - Interface Testing
        - Takes place once other components are fully tested
        - Objectives are to detect faults due to interface errors or invalid assumptions about interfaces
    - Stress Testing
        - Exercise system beyond normal load
        - Important for distributed systems
        - Attempt to fail cleanly and not catastrophically
- Object Oriented Systems
    - Components to be tested are object classes that are instantiated as objects
    - No obvious top to the system (nor bottom)
    - Complete test coverage of a class involves
        - Testing all operations associated with an object
        - Setting and interrogating all object attributes
        - Exercising the object in all possible states
    - Inheritance makes it difficult
    - Object Integration Testing
        - Cluster testing is concerned with integrated and testing clusters of cooperating objects
        - identify clusters using knowledge of the operation of objects and the system features that are implemented by these clusters
        - Approaches
            - Use case or scenario testing
            - Thread Testing
            - Object interaction testing

- Key Points

- Key Points
  - Test parts which are commonly used
  - Equivalence partitions are sets of test cases where the program should behave in an equivalent way
  - Black-box testing is based on the system specification
  - Test coverage measures ensure that all statements have ben executed at least once
  - Interface defects arise because of specification misreading