

Testing

- Testing programs to establish the presence of system defects

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 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

Defect testing

- The goal of defect testing is to discover defects in programs
- A *successful* defect test is a test which causes a program to behave in an anomalous way
- Tests show the presence not the absence of defects

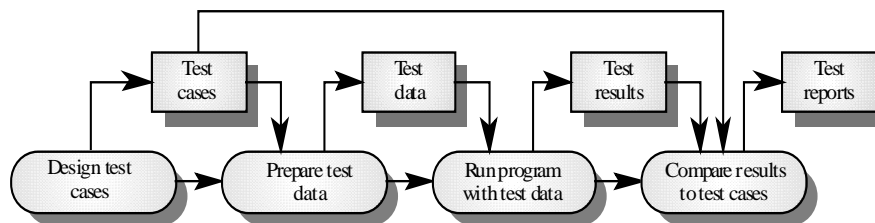
Testing priorities

- Only exhaustive testing can show a program is 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

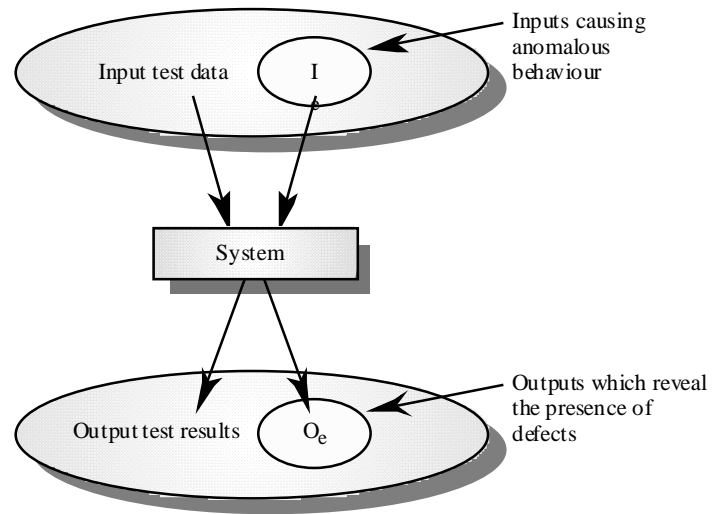
Test data and test cases

- *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

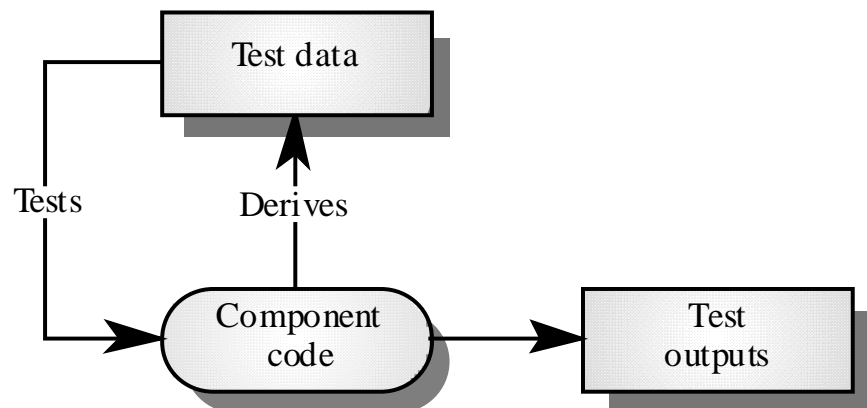
The defect testing process



Black-box testing



White-box testing

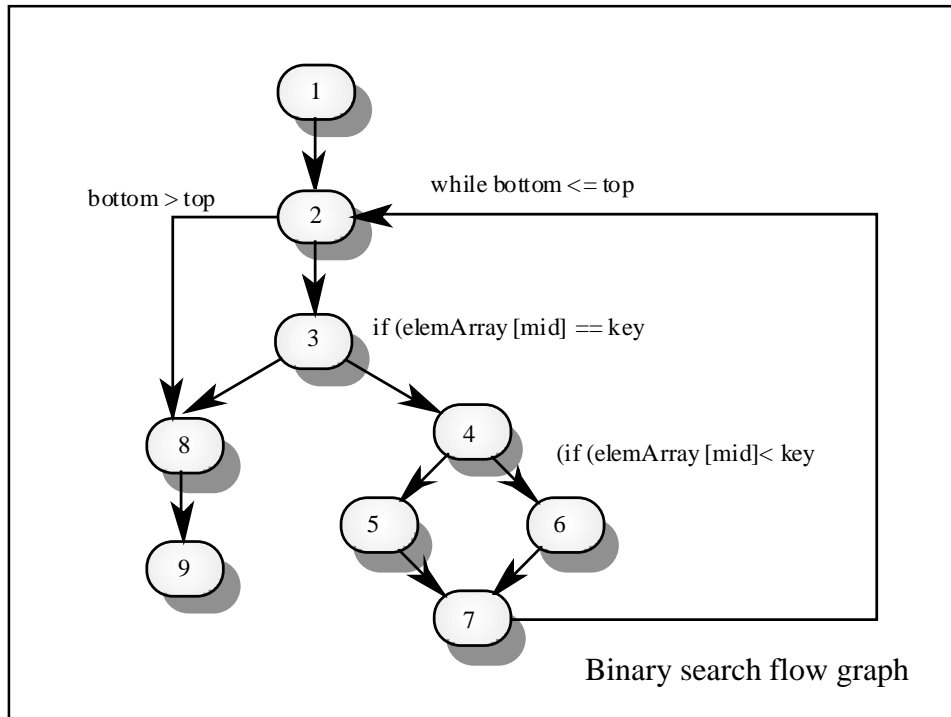


Path testing

- The objective of path testing is to ensure that the set of test cases is such that each path through the program is executed at least once
- The starting point for path testing is a program flow graph that shows nodes representing program decisions and arcs representing the flow of control
- Statements with conditions are therefore nodes in the flow graph

Program flow graphs

- Describes the program control flow. Each branch is shown as a separate path and loops are shown by arrows looping back to the loop condition node
- Used as a basis for computing the cyclomatic complexity
- Cyclomatic complexity
 - The number of tests to test all control statements equals the cyclomatic complexity
 - Cyclomatic complexity equals number of conditions in a program



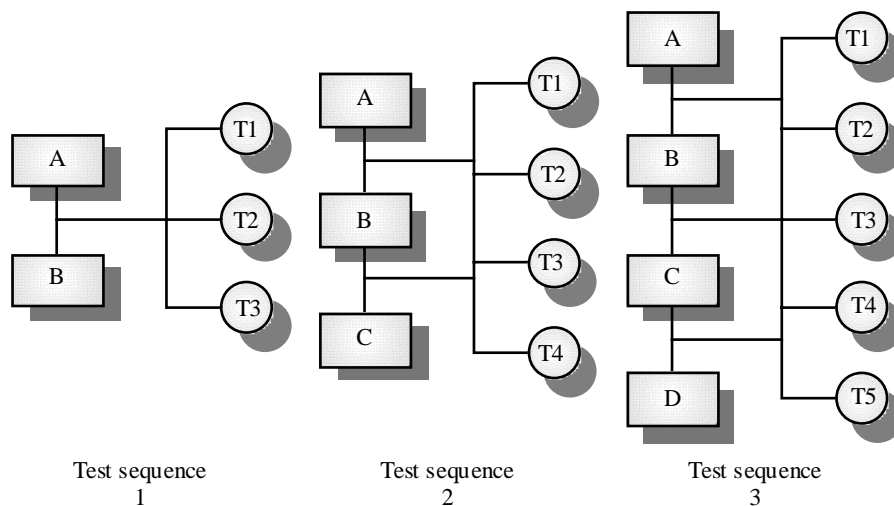
Independent paths

- 1, 2, 3, 8, 9
- 1, 2, 3, 4, 6, 7, 2
- 1, 2, 3, 4, 5, 7, 2
- 1, 2, 3, 4, 6, 7, 2, 8, 9
- Test cases should be derived so that all of these paths are executed
- A dynamic program analyser may be used to check that paths have been executed

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 errors
- Incremental integration testing reduces this problem

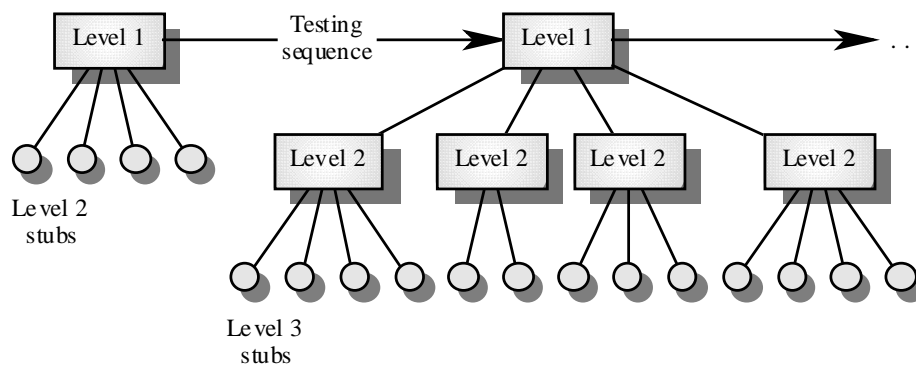
Incremental integration testing



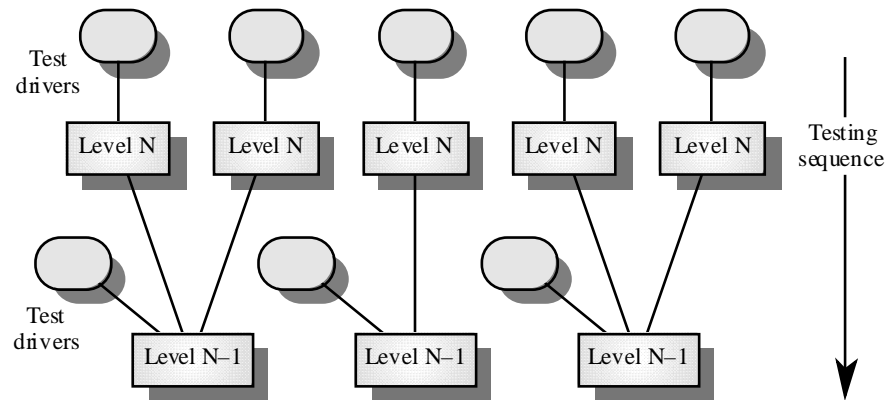
Approaches to integration testing

- Top-down testing
 - Start with high-level system and integrate from the top-down replacing individual components by stubs where appropriate
- Bottom-up testing
 - Integrate individual components in levels until the complete system is created
- In practice, most integration involves a combination of these strategies

Top-down testing



Bottom-up testing



Stress testing

- Exercises the system beyond its maximum design load. Stressing the system often causes defects to come to light
- Stressing the system test failure behaviour.. Systems should not fail catastrophically. Stress testing checks for unacceptable loss of service or data
- Particularly relevant to distributed systems which can exhibit severe degradation as a network becomes overloaded

Object-oriented testing

- The components to be tested are object classes that are instantiated as objects
- Testing levels
 - Testing operations associated with objects
 - Testing object classes
 - Testing clusters of cooperating objects
 - Testing the complete OO system

A testing workbench

