Lecture 6 – Implementation and Testing (Sommerville Ch. 20)

Karl R. Wilcox
K.R.Wilcox@reading.ac.uk

Objectives

- To understand some of the issues of project implementation
- To understand at a high level various testing techniques and approaches

- Today’s seminar
  - Q&A on Assignment

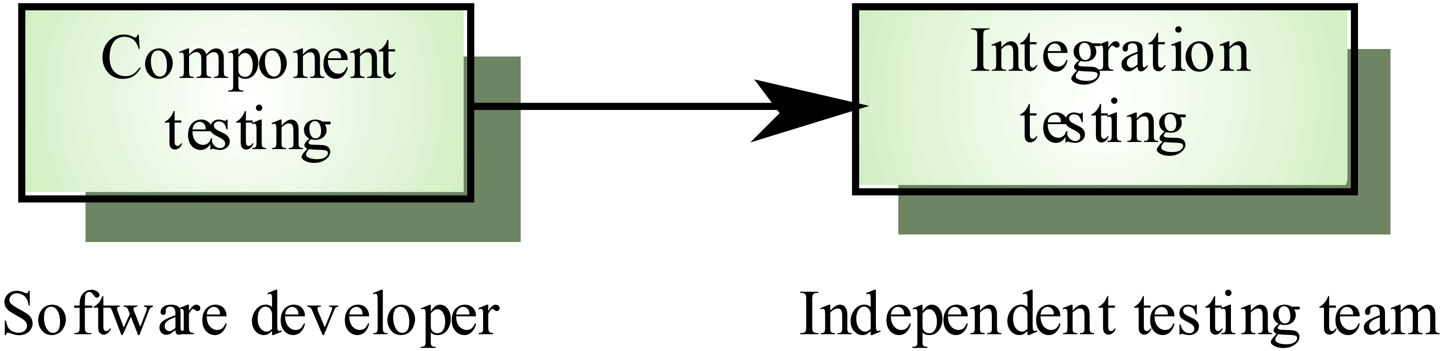
Implementation

- Making the damn thing work…
- May involve:
  - End user and system training
  - Data migration
  - Equipment and software roll out
  - Parallel running
  - Fallback arrangements
  - Workarounds
  - Late nights, high stress levels, broken marriages…

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

Testing phases



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

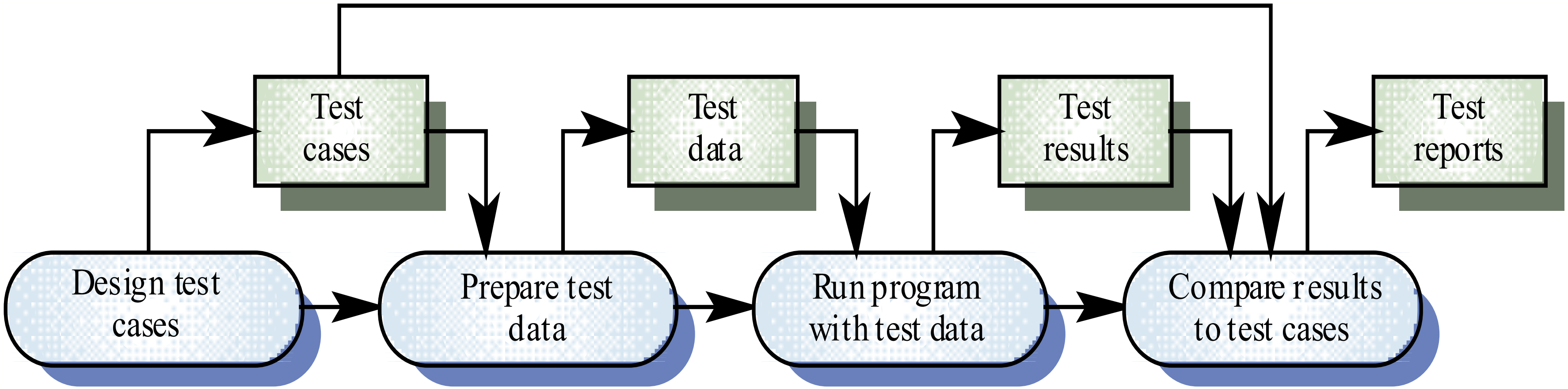
Testing priorities

- 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

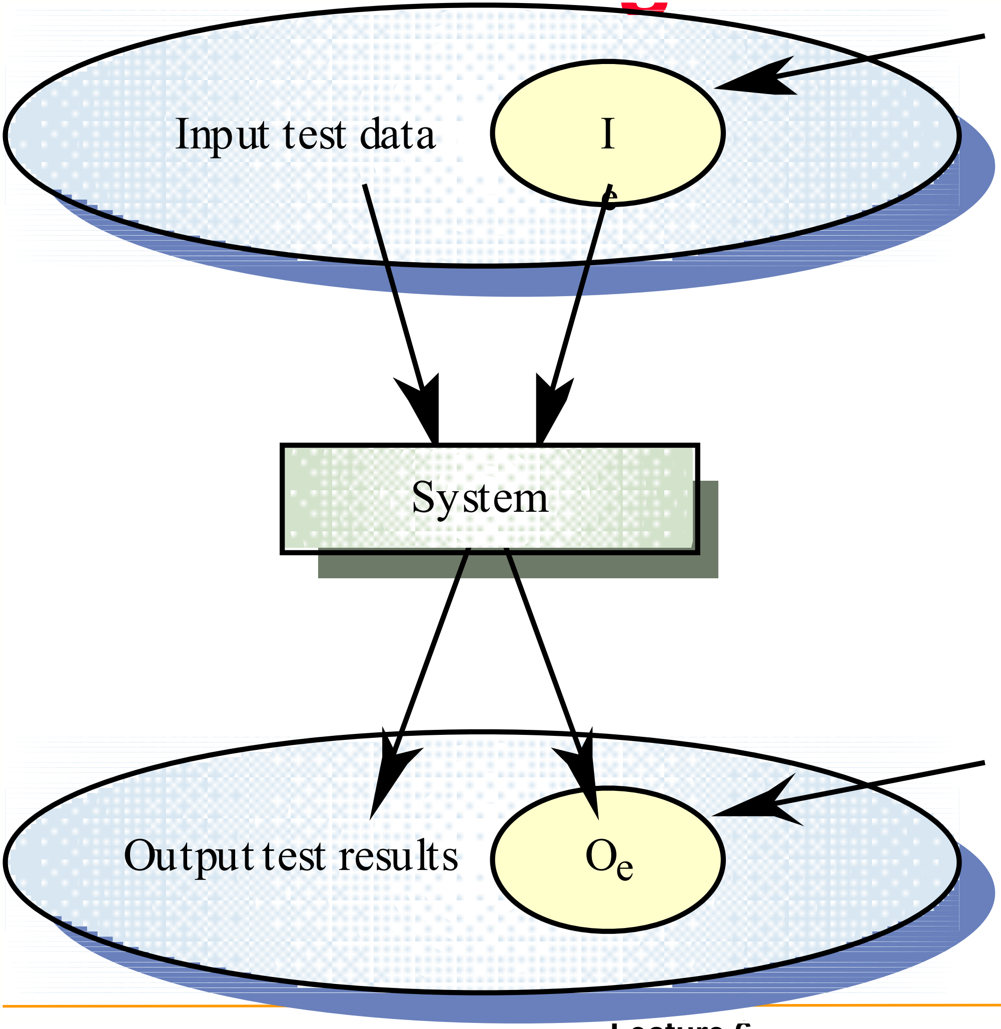
The defect testing process



Black-box testing

- An approach to testing where the program is considered as a ‘black-box’
- The program test cases are based on the system specification
- Test planning can begin early in the software process

Black-box testing



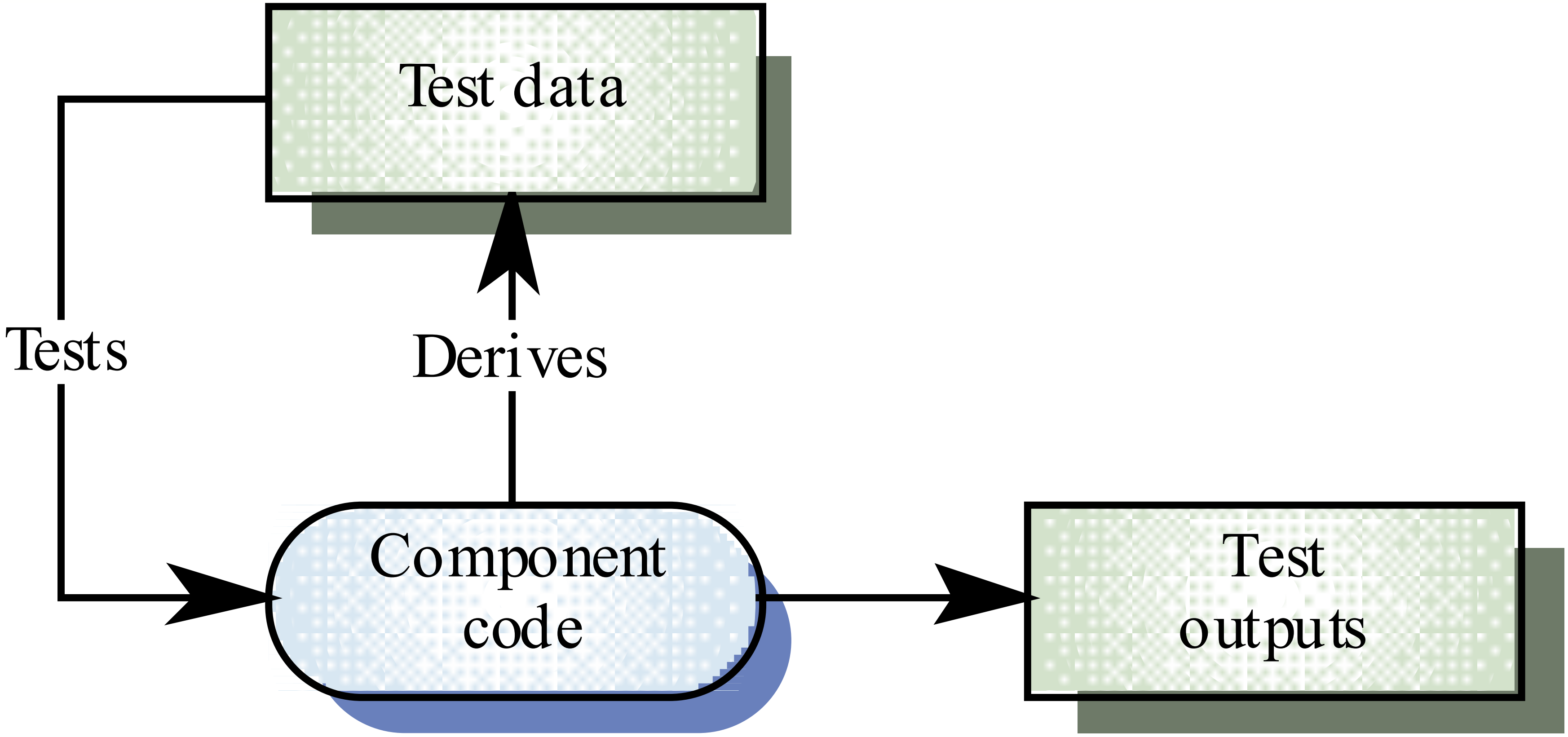
Inputs causing anomalous behaviour

Outputs which reveal the presence of defects

Structural testing

- Sometime called white-box testing
- Derivation of test cases according to program structure. Knowledge of the program is used to identify additional test cases
- Objective is to exercise all program statements (not all path combinations)

White-box testing



Path testing

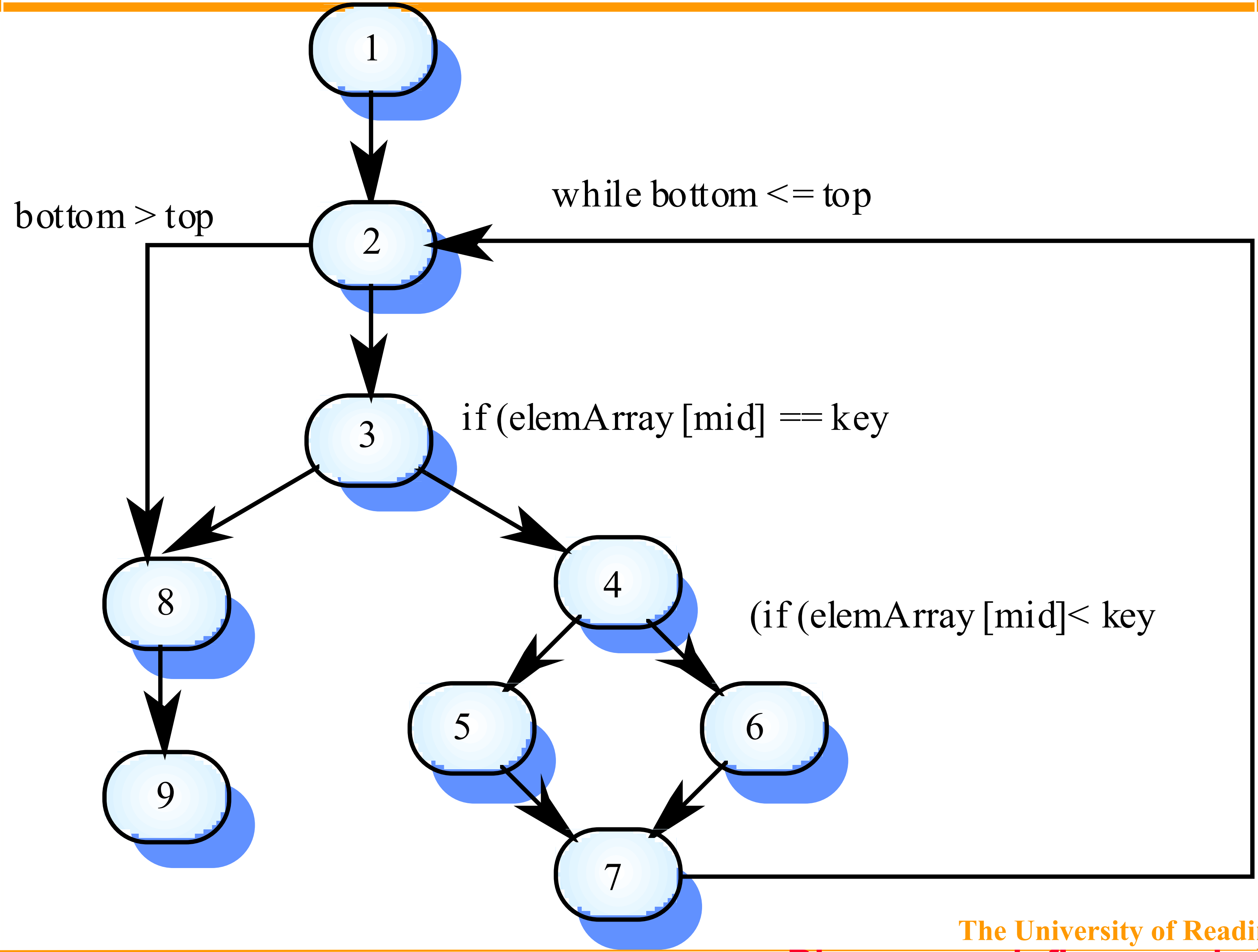
- The objective of path testing is to ensure that the set of test cases is such that each path through the program is executed at least once
- The starting point for path testing is a program flow graph that shows nodes representing program decisions and arcs representing the flow of control
- Statements 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 = Number of edges - Number of nodes +2

Cyclomatic complexity

- The number of tests to test all control statements equals the cyclomatic complexity
- Cyclomatic complexity equals number of conditions in a program
- Useful if used with care. Does not imply adequacy of testing.
- Although all paths are executed, all combinations of paths are not executed

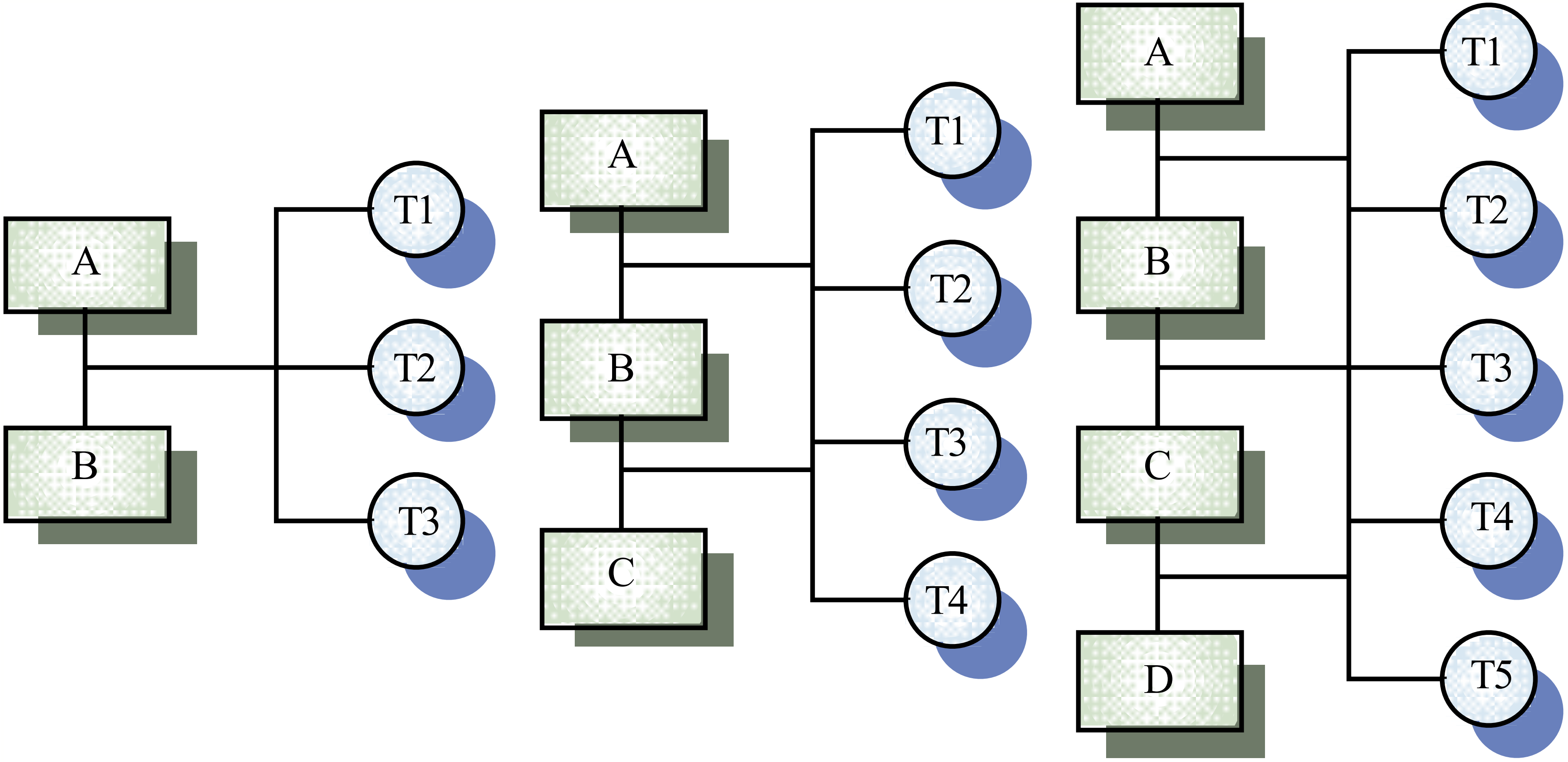


Binary search flow graph

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



Test sequence 1

Test sequence 2

Test sequence 3

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

Tetsing approaches

- Architectural validation
  - Top-down integration testing is better at discovering errors in the system architecture
- System demonstration
  - Top-down integration testing allows a limited demonstration at an early stage in the development
- Test implementation
  - Often easier with bottom-up integration testing
- Test observation
  - Problems with both approaches. Extra code may be required to observe tests

Interface testing

- Takes place when modules or sub-systems are integrated to create larger systems
- Objectives are to detect faults due to interface errors or invalid assumptions about interfaces
- Particularly important for object-oriented development as objects are defined by their interfaces

Interfaces types

- Parameter interfaces
  - Data passed from one procedure to another
- Shared memory interfaces
  - Block of memory is shared between procedures
- Procedural interfaces
  - Sub-system encapsulates a set of procedures to be called by other sub-systems
- Message passing interfaces
  - Sub-systems request services from other sub-systems

Interface errors

- Interface misuse
  - A calling component calls another component and makes an error in its use of its interface e.g. parameters in the wrong order
- Interface misunderstanding
  - A calling component embeds assumptions about the behaviour of the called component which are incorrect
- Timing errors
  - The called and the calling component operate at different speeds and out-of-date information is accessed

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
- Larger grain than individual functions so approaches to white-box testing have to be extended
- No obvious ‘top’ to the system for top-down integration and testing

Object class testing

- 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 more difficult to design object class tests as the information to be tested is not localised

Object integration

- Levels of integration are less distinct in object-oriented systems
- Cluster testing is concerned with integrating 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

Scenario-based testing

- Identify scenarios from use-cases and supplement these with interaction diagrams that show the objects involved in the scenario
- Consider the scenario in the weather station system where a report is generated

Key points

- Test parts of a system which are commonly used rather than those which are rarely executed
- Black-box testing is based on the system specification
- Structural testing identifies test cases which cause all paths through the program to be executed

Key points

- Test coverage measures ensure that all statements have been executed at least once.
- Interface defects arise because of specification misreading, misunderstanding, errors or invalid timing assumptions
- To test object classes, test all operations, attributes and states
- Integrate object-oriented systems around clusters of objects