Chapter 23

Software Testing

Ian Sommerville
Lutz Prechelt

Objectives

- To discuss the distinctions between validation testing and defect testing
- To describe the principles of system and component testing
- To describe strategies for generating system test cases
- To understand the essential characteristics of tool used for test automation

Topics covered

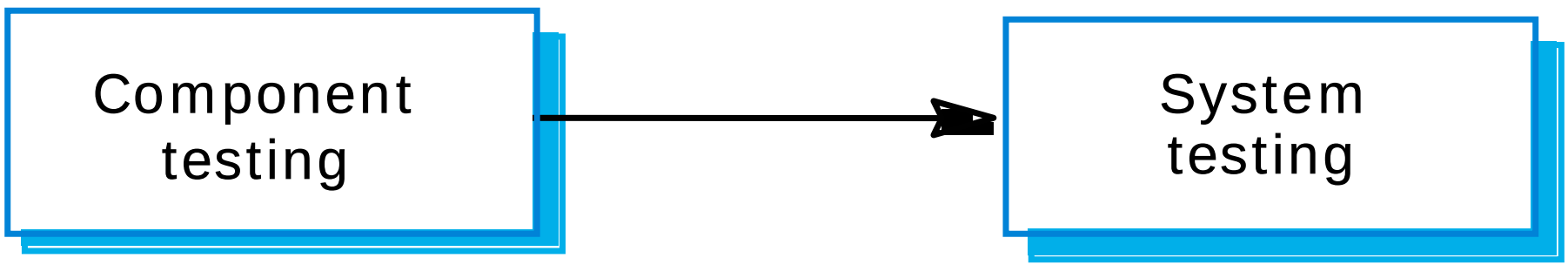
- System testing
- Component testing
- Test case design
- Test automation

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.

- System 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



Software developer

Independent testing team

Defect testing

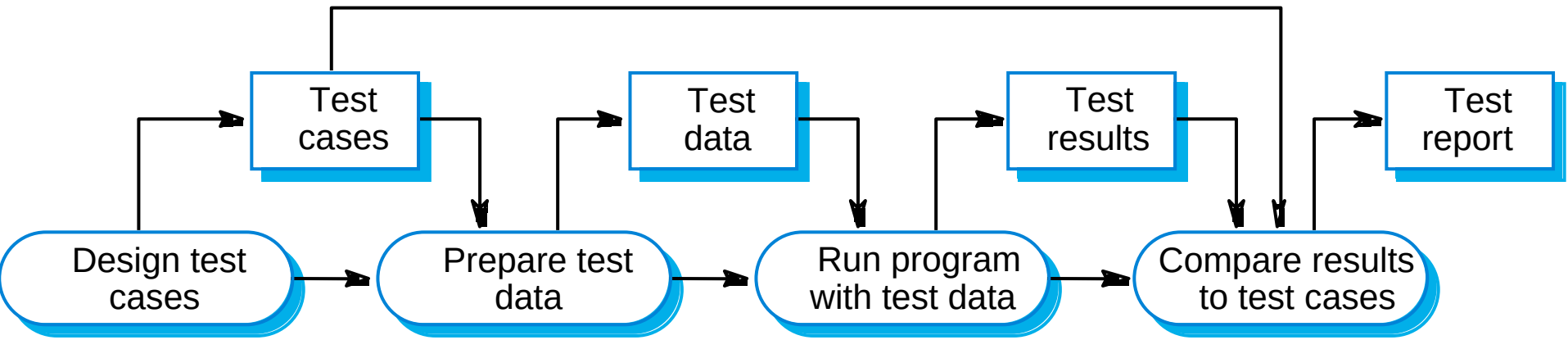
- The goal of defect testing is to discover defects in programs.
- A *successful* defect test is a test which causes a program to behave in an anomalous way.
  - This is the opposite of the view taken by validation testing!
- Tests show the presence not the absence of defects.

Testing process goals

- Defect testing
  - To discover faults or defects in the software where its behaviour is incorrect or not in conformance with its specification;
  - A successful test is a test that makes the system perform incorrectly and so exposes a defect in the system.
- Validation testing
  - To demonstrate to the developer and the system customer that the software meets its requirements;
  - A successful test shows that the system operates as intended.

- We start with defect testing and perform validation testing later in the process.

The software testing process



- Note: For many (if not most) people, the term "test case" includes the test data as well.

Testing policies

- Only exhaustive testing could show a program is free from defects.
  - However, exhaustive testing is impossible.

- Testing policies define the approach to be used in selecting system tests. Examples:
  - All functions accessed through menus should be tested;
  - Combinations of functions accessed through the same menu should be tested;
  - Where user input is required, all functions must be tested both with correct input and with incorrect input.

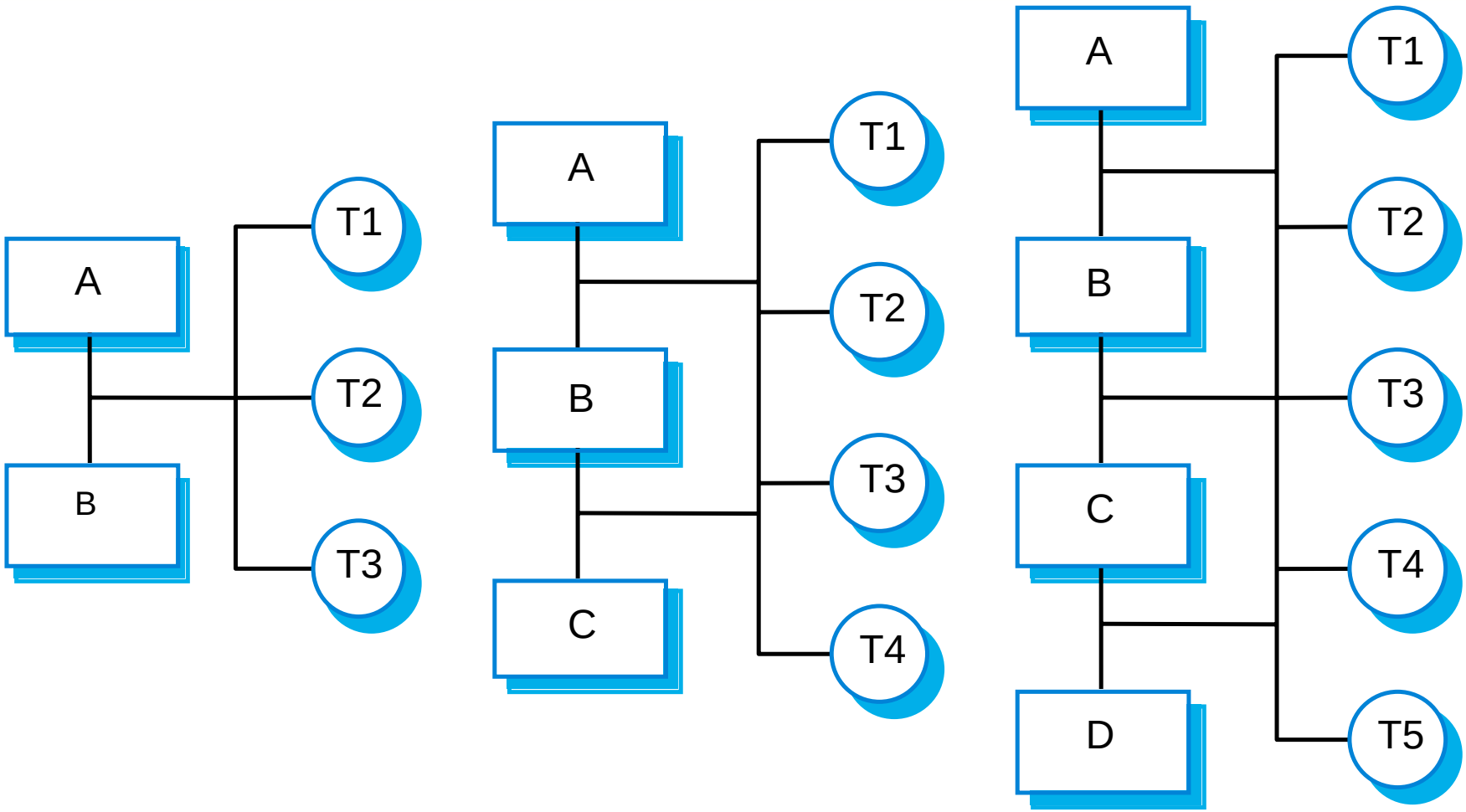
System testing

- Involves integrating components to create a system or sub-system.
- May involve testing an increment to be delivered to the customer.
- Two phases:
  - Integration testing – the test team have access to the system source code. The system is tested as components are integrated.
  - Release testing – the test team test the complete system as a black-box.

Integration testing

- Involves building a system from its components and testing it for problems that arise from component interactions.
  - Done incrementally to simplify error localisation.
- Top-down integration
  - Rarely used (except for prototypes). Develop the skeleton of the system and populate it with components.
  - Requires writing 'stubs' as placeholders for yet-missing sub-systems or components.
- Bottom-up integration
  - Commonly used. Integrate low-level infrastructure components first, then add functional components.
  - Requires writing 'drivers' for calling the components and producing output.

Incremental integration testing



Test sequence 1 Test sequence 2 Test sequence 3

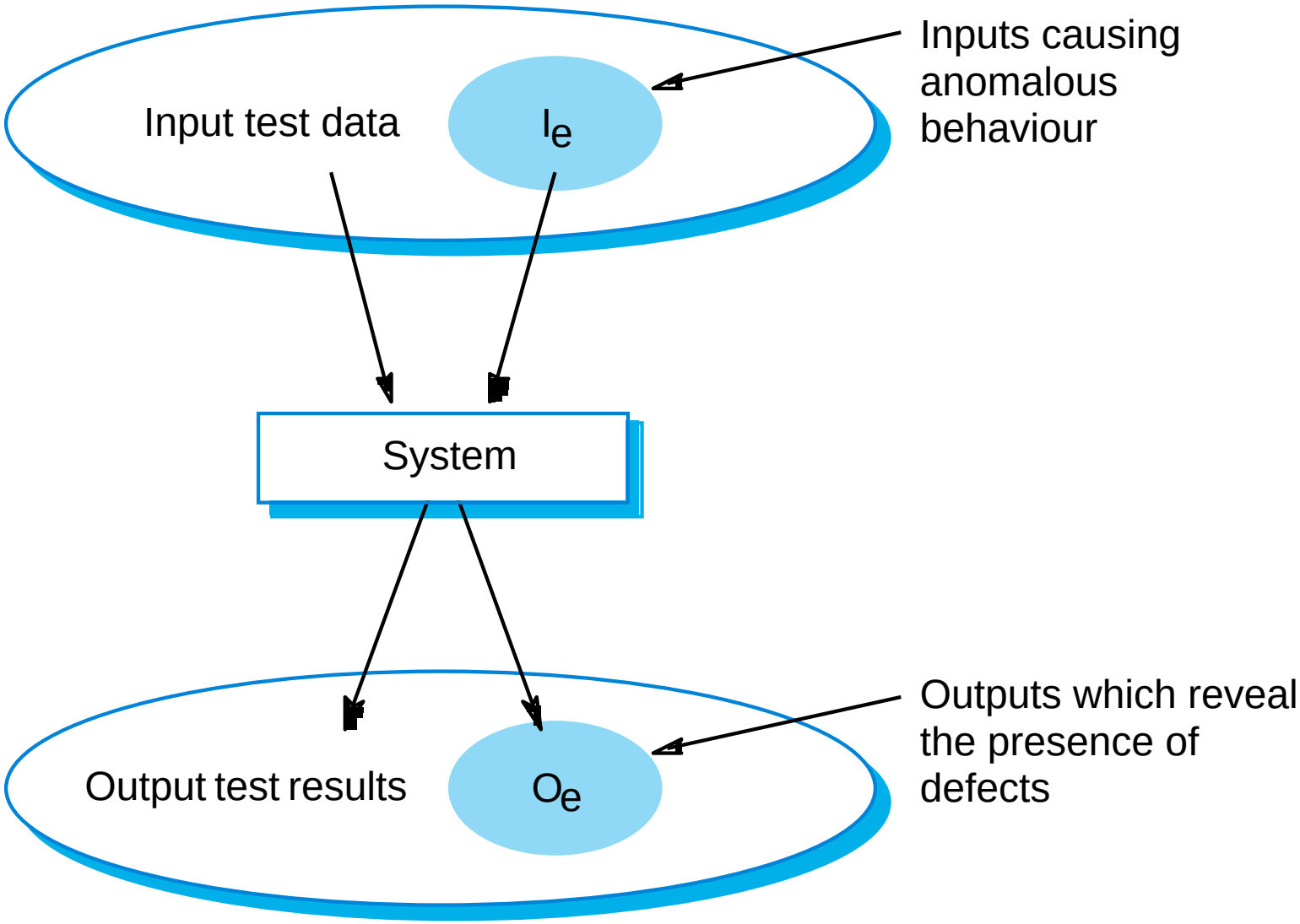
Testing approaches

- 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.

Release testing (acceptance testing)

- The process of testing a release of a system that will be distributed to customers.
- Primary goal is to increase the supplier's confidence that the system meets its requirements.
- Release testing is usually black-box or functional testing.
  - Based on the system specification only;
  - Testers do not have knowledge of the system implementation.

Black-box testing



Testing guidelines

- Testing guidelines are hints for the testing team to help them choose tests that will reveal defects in the system
  - Choose inputs that force the system to generate all error messages;
  - Choose inputs that are legal, but unlikely;
  - Design inputs that cause buffers to overflow;
  - Repeat the same input or input series several times;
  - Force invalid outputs to be generated;
  - Force computation results to be too large or too small.

Testing scenario

A scotish student of American History has to write a paper on 'Frontier mentality in the American West from 1840 to 1880'.

She logs on to the LIBSYS system and uses the search facility to discover if she can access original documents from that time. She discovers sources in various US university libraries and downloads copies of some of these.

However, for one document, her university must confirm that she is a genuine student and that use is for non-commercial purposes. The student then uses the facility in LIBSYS that can request such permission and registers her request.

If granted, the document will be downloaded to the registered library's server and printed for her.

She receives a message from LIBSYS telling her that she will receive an e-mail message when the printed document is available for collection.

System tests

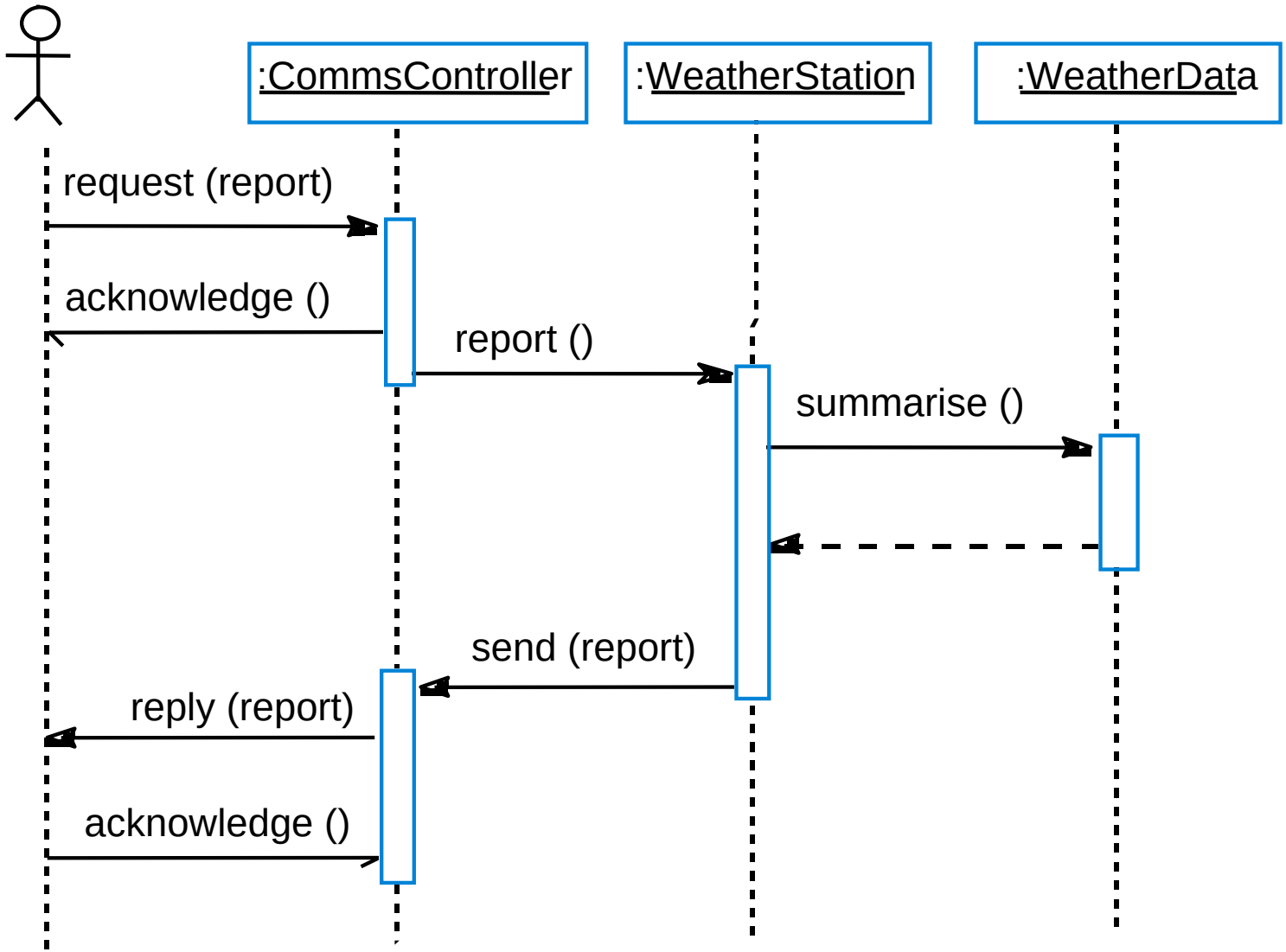
- Test the login mechanism using correct and incorrect logins
  - to check that valid users are accepted and invalid users are rejected.
- Test the search facility using different queries against known sources
  - to check that the search mechanism is actually finding documents.
- Test the system presentation facility
  - to check that information about documents is displayed properly.
- Test the mechanism to request permission for downloading.
- Test the e-mail response indicating that the downloaded document is available.

UML: Use cases

- Use cases can be a basis for deriving the tests for a system.
  - They help identify operations to be tested and help design the required test cases.

- From an associated sequence diagram, the inputs and outputs to be created for the tests can be identified.

Collect weather data sequence chart



Performance testing

- Part of release testing may involve testing the emergent properties of a system
  - such as performance and reliability.

- Performance tests usually involve planning a series of tests where the load is steadily increased until the system performance becomes unacceptable.

Stress testing

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

Component testing

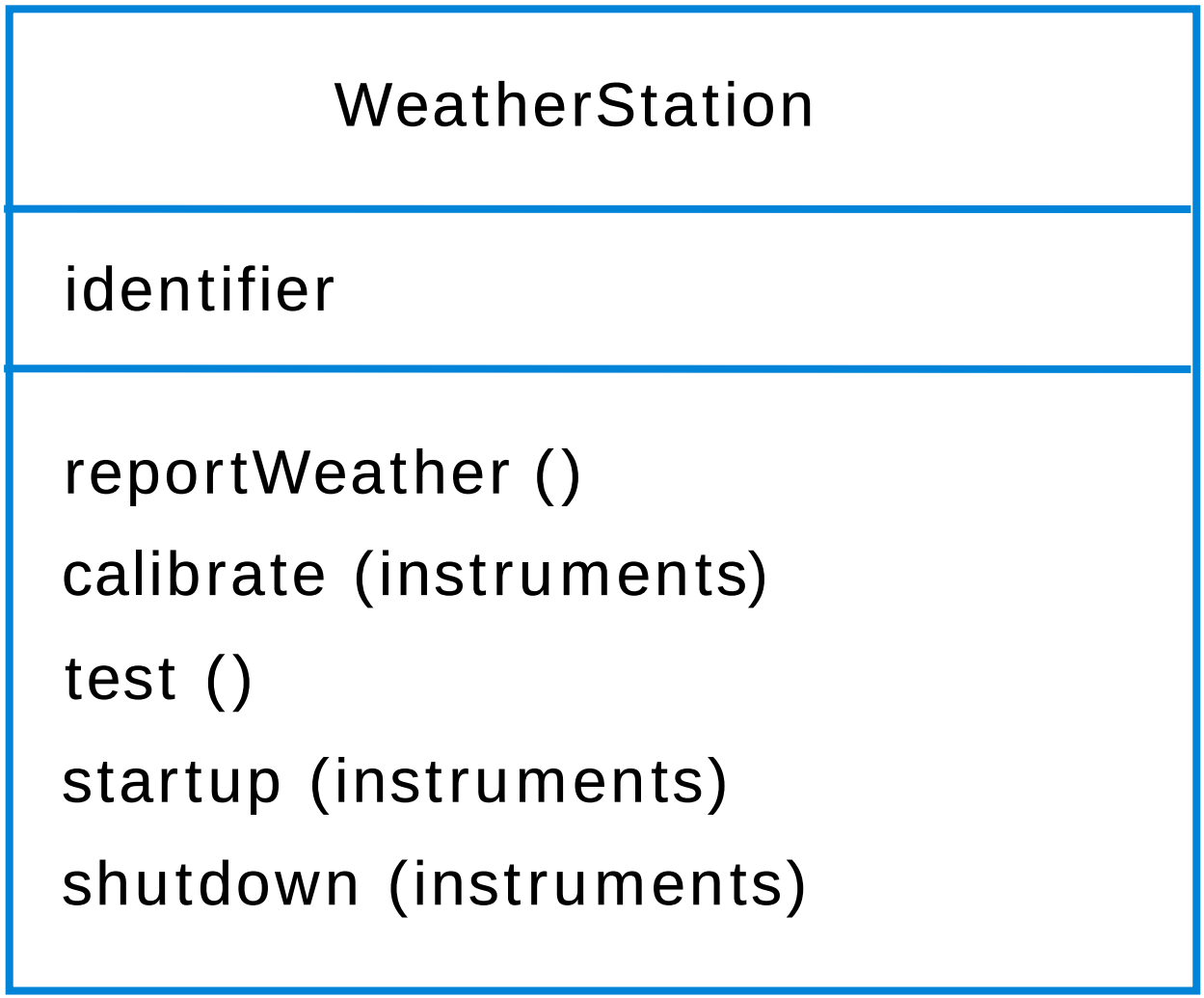
- Component or unit testing is the process of testing individual components in isolation.
- It is a defect testing process.
- Components may be:
  - Individual functions or methods within an object;
  - Object classes with several attributes and methods;
  - Composite components with defined interfaces used to access their functionality.

Object class testing

- Complete test coverage of a class involves
  - Setting and interrogating all object attributes;
  - Testing all operations associated with an object;
  - 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.
  - This is one of the reasons why it is often advisable to inherit only interfaces, not implementations.

Weather station object interface



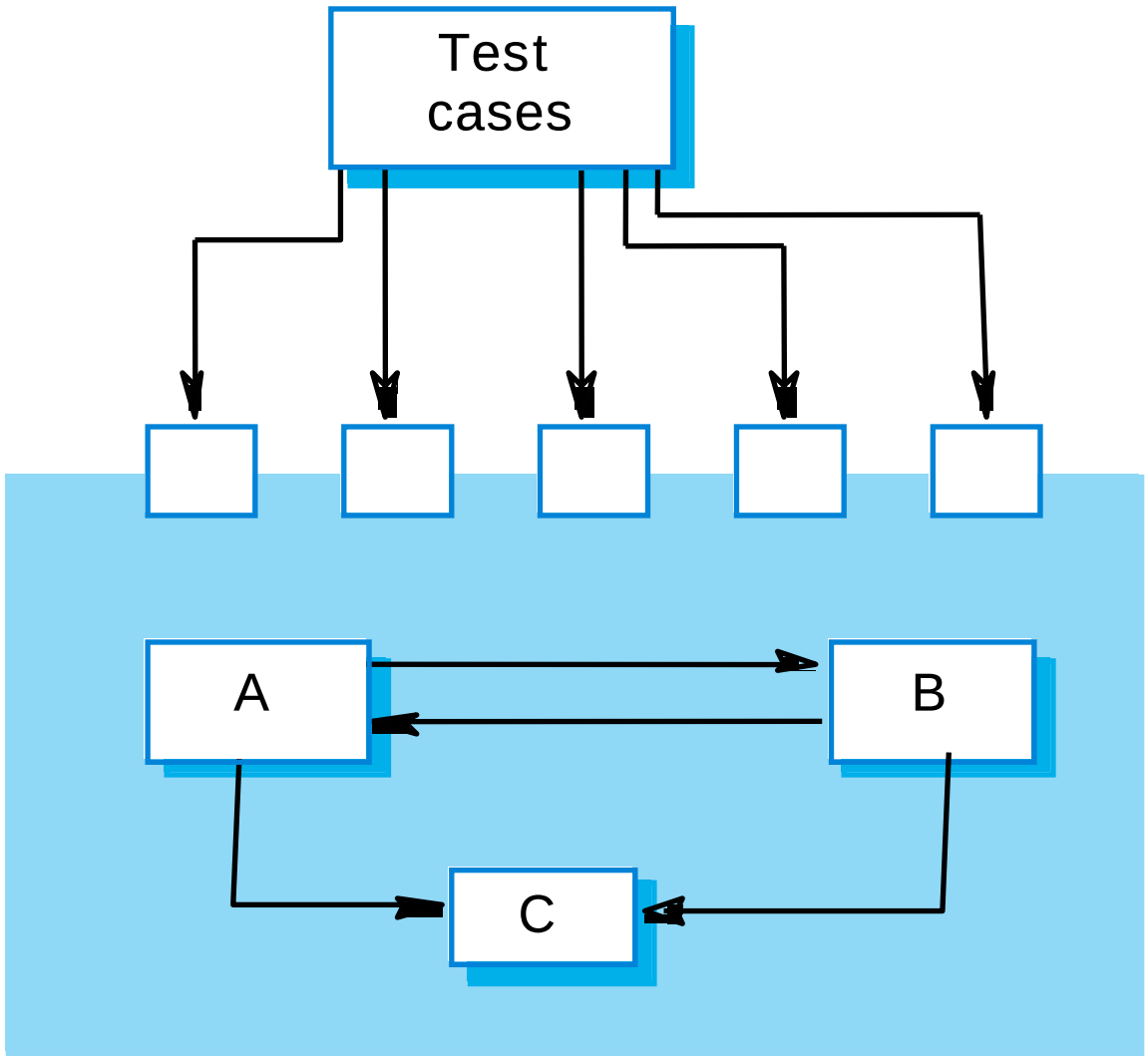
Weather station testing

- Need to define test cases for reportWeather, calibrate, test, startup and shutdown.
- Using a state model, identify sequences of state transitions to be tested
  - and the event sequences to cause these transitions.
- For example:
  - Waiting → Calibrating → Testing → Transmittng → Waiting

Interface testing

- Objectives are to detect faults due to interface errors or invalid assumptions about interfaces.
  - Warning: Never mistake a set of signatures (or some such) for an interface!
- Particularly important for object-oriented development as objects are defined by their interfaces.

Interface testing



Interface types

- Parameter interfaces
  - Data passed from one procedure to another.
- Shared memory interfaces
  - Block of memory is shared between procedures or functions.
- 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.

Interface testing hints

- Design tests so that parameters to a called procedure are at the extreme ends of their ranges.
- Always test pointer parameters with null pointers.
- Design tests which cause the component to fail.
- Use stress testing in message passing systems.
- In shared memory systems, vary the order in which components are activated.

Test case design

- Involves designing the test cases (inputs and outputs) used to test the system.
- The goal of test case design is to create a set of tests that are effective in validation and defect testing.
- Design approaches:
  - Requirements-based testing;
  - Partition testing;
  - Structural testing.

Requirements based testing

- A general principle of requirements engineering is that requirements should be testable.
  - Some even say "must be": If it cannot be tested, it is not a requirement.

- Requirements-based testing is a validation testing technique where you consider each requirement and derive a set of tests for that requirement.

LIBSYS requirements

- The user shall be able to search either all of the initial set of databases or select a subset from it.
- The system shall provide appropriate viewers for the user to read documents in the document store.
- Every order shall be allocated a unique identifier (ORDER_ID) that the user shall be able to copy to the account's permanent storage area.

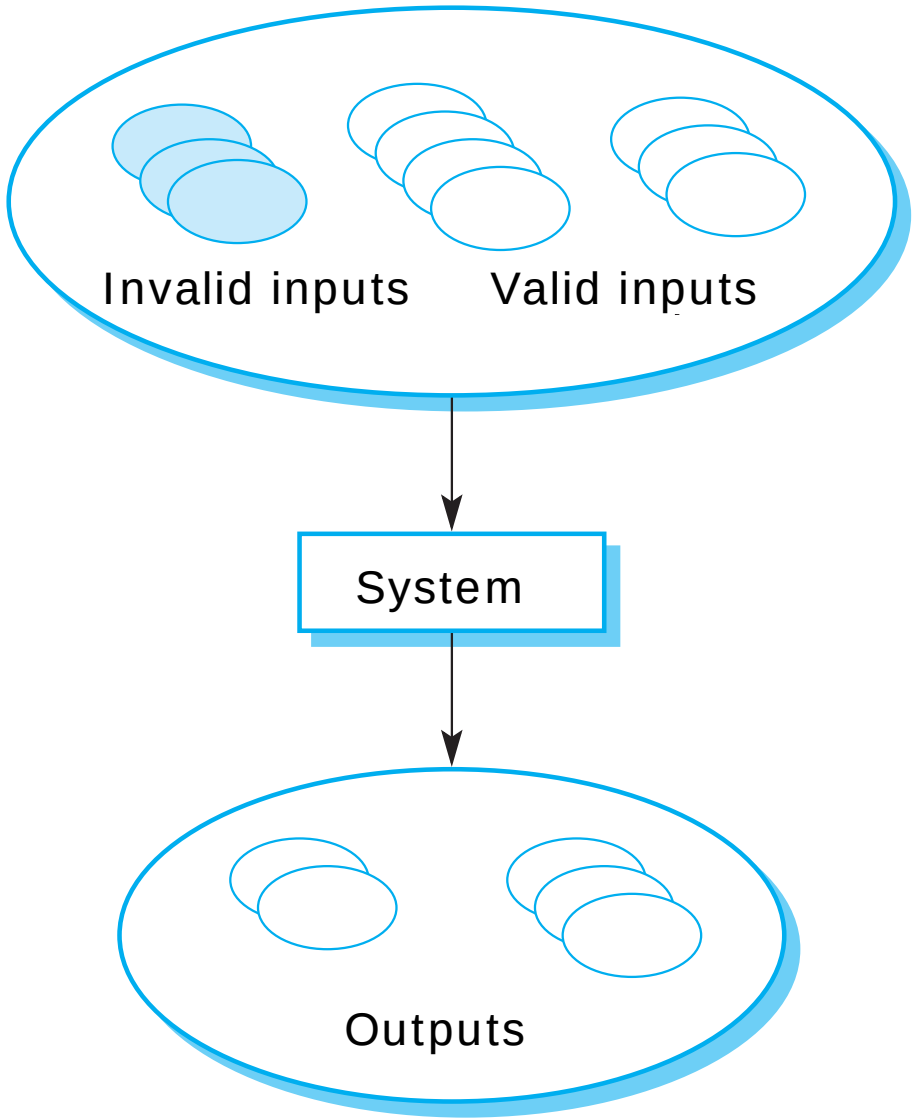
LIBSYS tests

- Initiate user search for searches for items that are known to be present and known not to be present, where the set of databases includes 1 database.
- As before, but where the set of databases includes 2 databases.
- As before, but where the set of databases includes more than 2 databases.
- Select one database from the set of databases and initiate user searches for items that are known to be present and known not to be present.
- As before, but select more than one database.

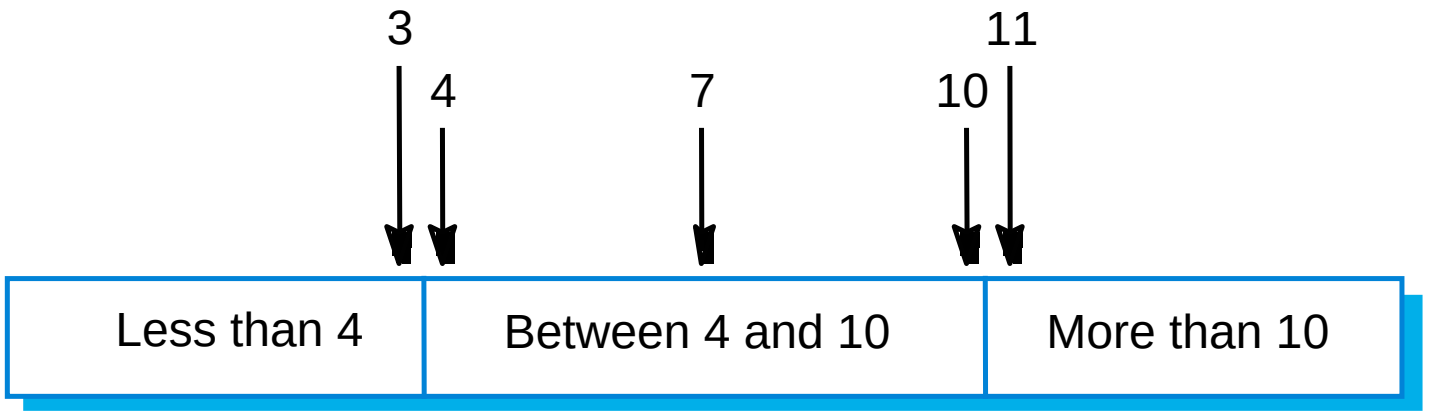
Partition testing

- Input data and output results often fall into different classes where all members of a class are related.
  - The program behaves in an equivalent way for each class member.
- Each of these classes is an equivalence partition or domain.
- Test cases should be chosen from each partition.

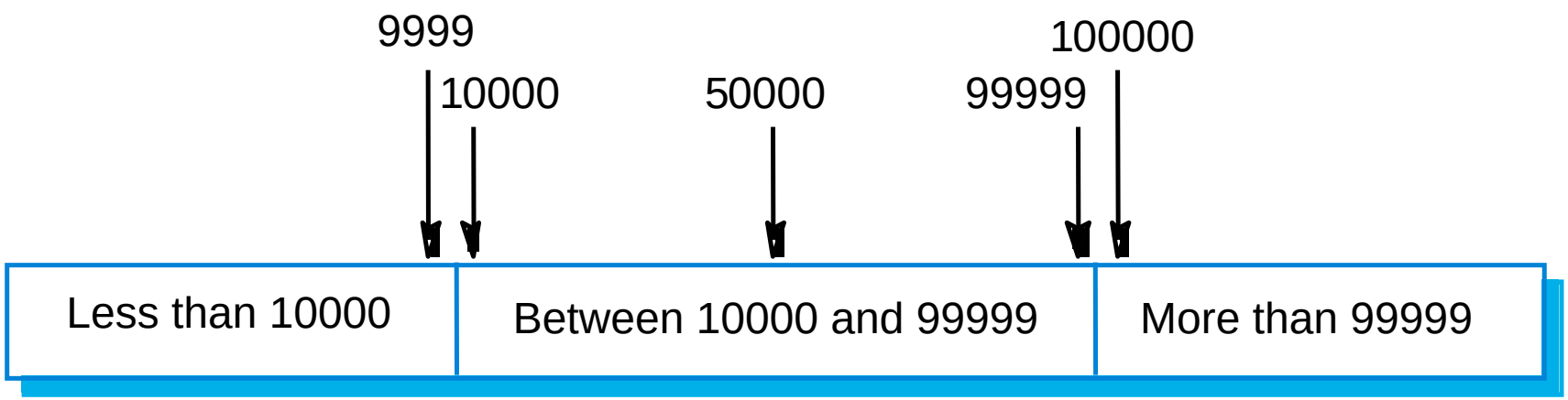
Equivalence partitioning



Equivalence partitions



Number of input values



Input values

Search routine specification

procedure Search (Key : ELEM ; T: SEQ of ELEM;
Found : **in out** BOOLEAN; L: **in out** ELEM_INDEX) ;

Pre-condition
-- the sequence has at least one element
T'FIRST <= T'LAST
Post-condition
-- the element is found and is referenced by L
(Found and T (L) = Key)
or
-- the element is not in the array
(**not** Found **and**
not (**exists** i, T'FIRST >= i <= T'LAST, T (i) = Key))

Search routine – input partitions

- Inputs which conform to the pre-conditions.
- Inputs where a pre-condition does not hold.
  - The specification does not say how the routine must behave, but non-functional requirements may tell us.
- Inputs where the key element is a member of the array.
- Inputs where the key element is not a member of the array.

Testing guidelines (sequences)

- Test software with sequences which have only a single value.
- Use sequences of different sizes in different tests.
- Derive tests so that the first, middle and last elements of the sequence are accessed.
- Test with sequences of zero length.

Search routine – input partitions

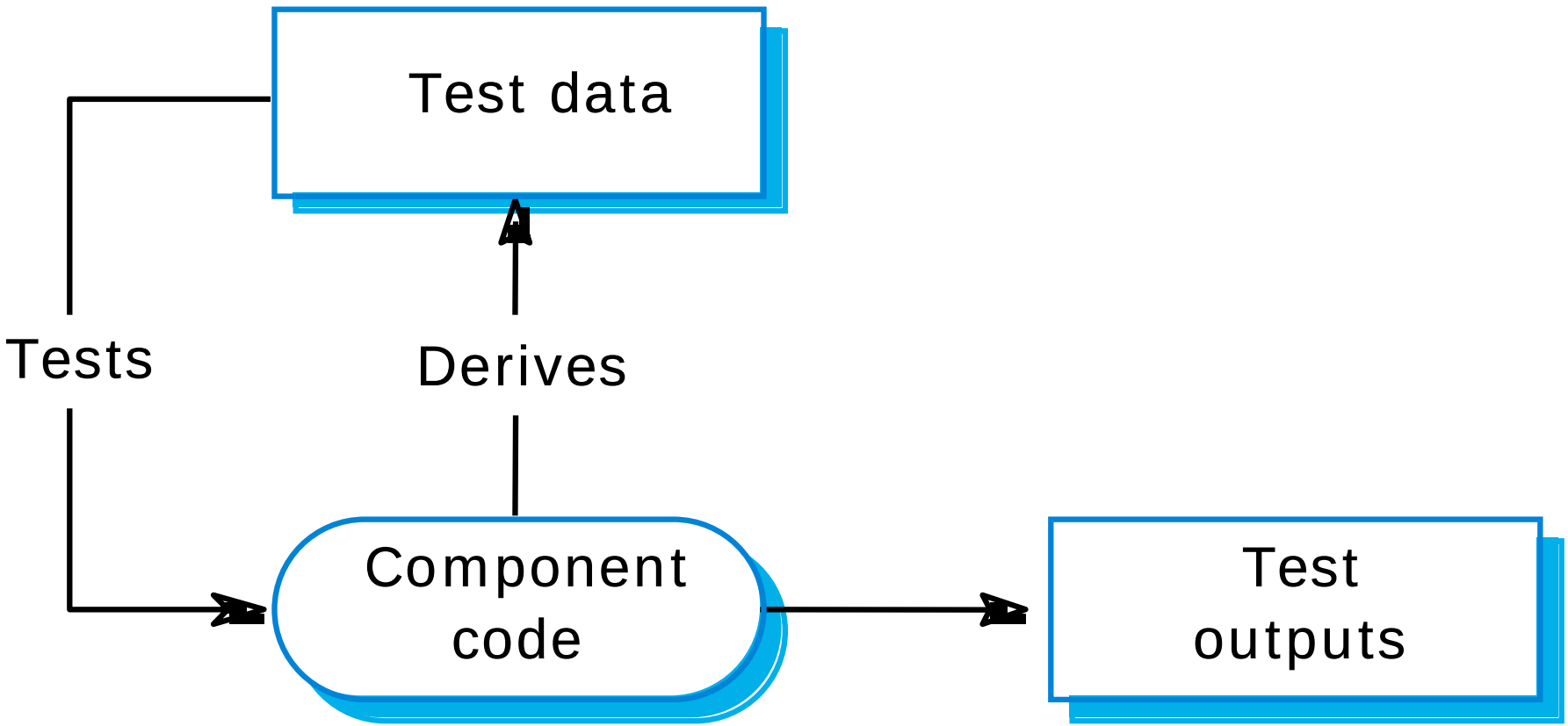
Sequence	**Element**
Single value	In sequence
Single value	Not in sequence
More than 1 value	First element in sequence
More than 1 value	Last element in sequence
More than 1 value	Middle element in sequence
More than 1 value	Not in sequence

Input sequence (T)	**Key** (Key)	**Output** (Found, L)
17	17	true, 1
17	0	false, ??
17, 29, 21, 23	17	true, 1
41, 18, 9, 31, 30, 16, 45	45	true, 7
17, 18, 21, 23, 29, 41, 38	23	true, 4
21, 23, 29, 33, 38	25	false, ??

Structural testing

- Sometimes called white-box testing.
- Derivation of test cases according to program structure.
  - Knowledge of the program is used to identify additional test cases.
- Possible objectives:
  - Exercise all program statements (statement coverage, C0).
  - Make each decision have each possible result (branch coverage, C1).
    - Note that exception handling (try/catch) introduces implicit additional decision points.
  - Exercise all possible data flows of certain kinds within the program (data flow testing).

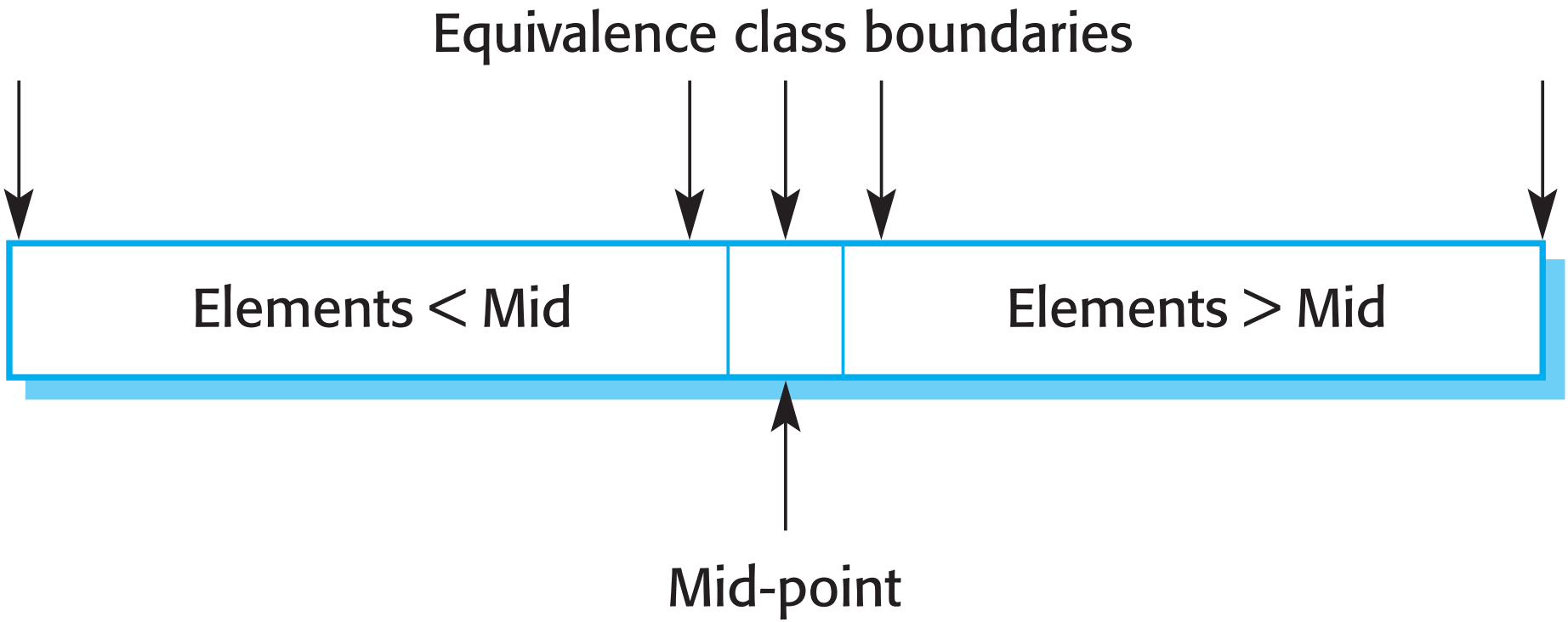
White-box testing



Binary search – equiv. partitions

- Pre-conditions satisfied, key element in array.
- Pre-conditions satisfied, key element not in array.
- Pre-conditions unsatisfied, key element in array.
- Pre-conditions unsatisfied, key element not in array.
- Input array has a single value.
- Input array has an even number of values.
- Input array has an odd number of values.

Binary search equiv. partitions



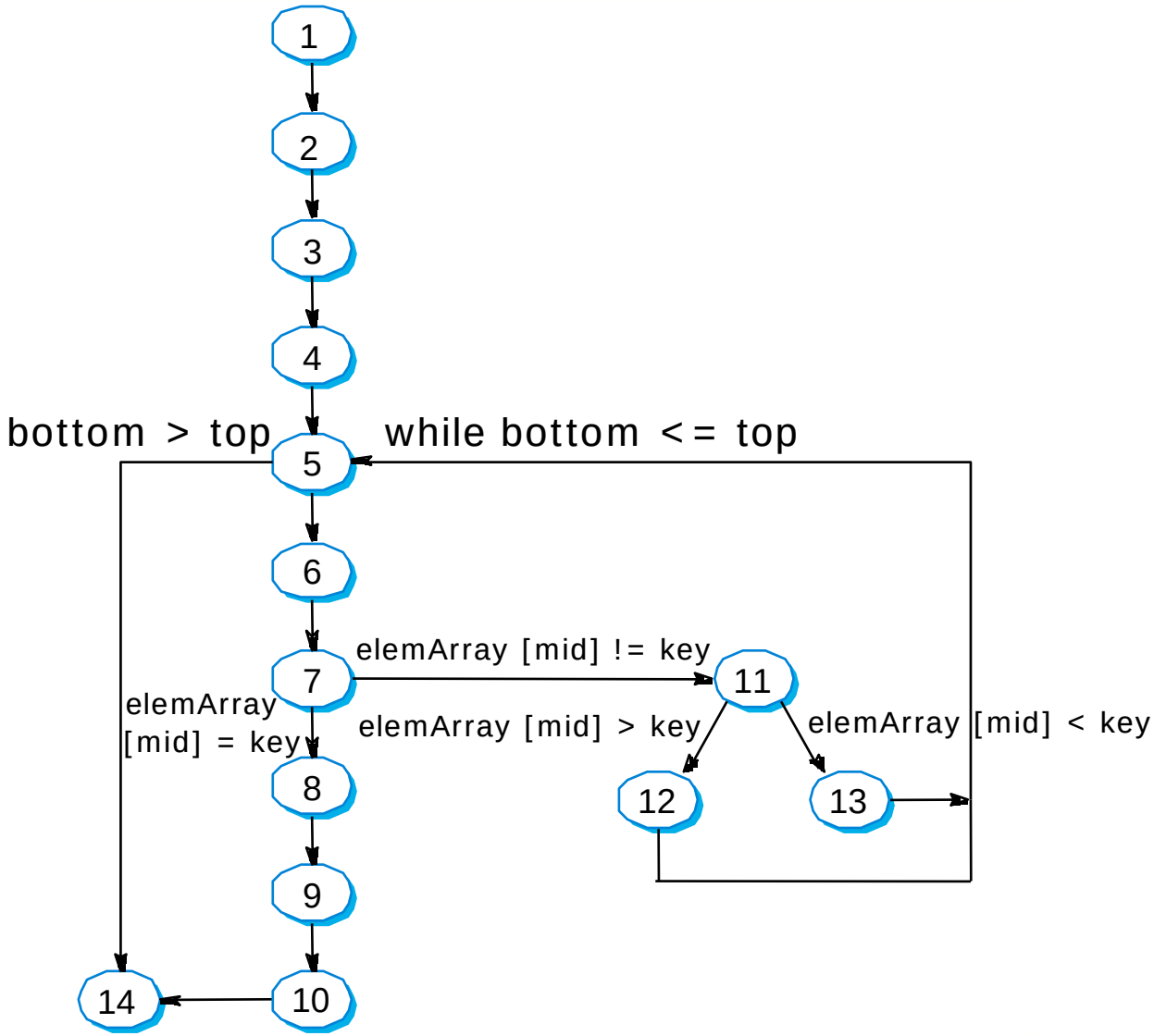
Binary search – test cases

Input array (T)	**Key** (Key)	**Output** (Found, L)
17	17	true, 1
17	0	false, ??
17, 21, 23, 29	17	true, 1
9, 16, 18, 30, 31, 41, 45	45	true, 7
17, 18, 21, 23, 29, 38, 41	23	true, 4
17, 18, 21, 23, 29, 33, 38	21	true, 3
12, 18, 21, 23, 32	23	true, 4
21, 23, 29, 33, 38	25	false, ??

Branch coverage: Path testing

- The objective of path testing is to ensure that the set of test cases is such that each turn is taken at each decision 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.

Binary search flow graph



Independent paths

- 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 14
- 1, 2, 3, 4, 5, 14
- 1, 2, 3, 4, 5, 6, 7, 11, 12, 5, ...
- 1, 2, 3, 4, 6, 7, 2, 11, 13, 5, ...

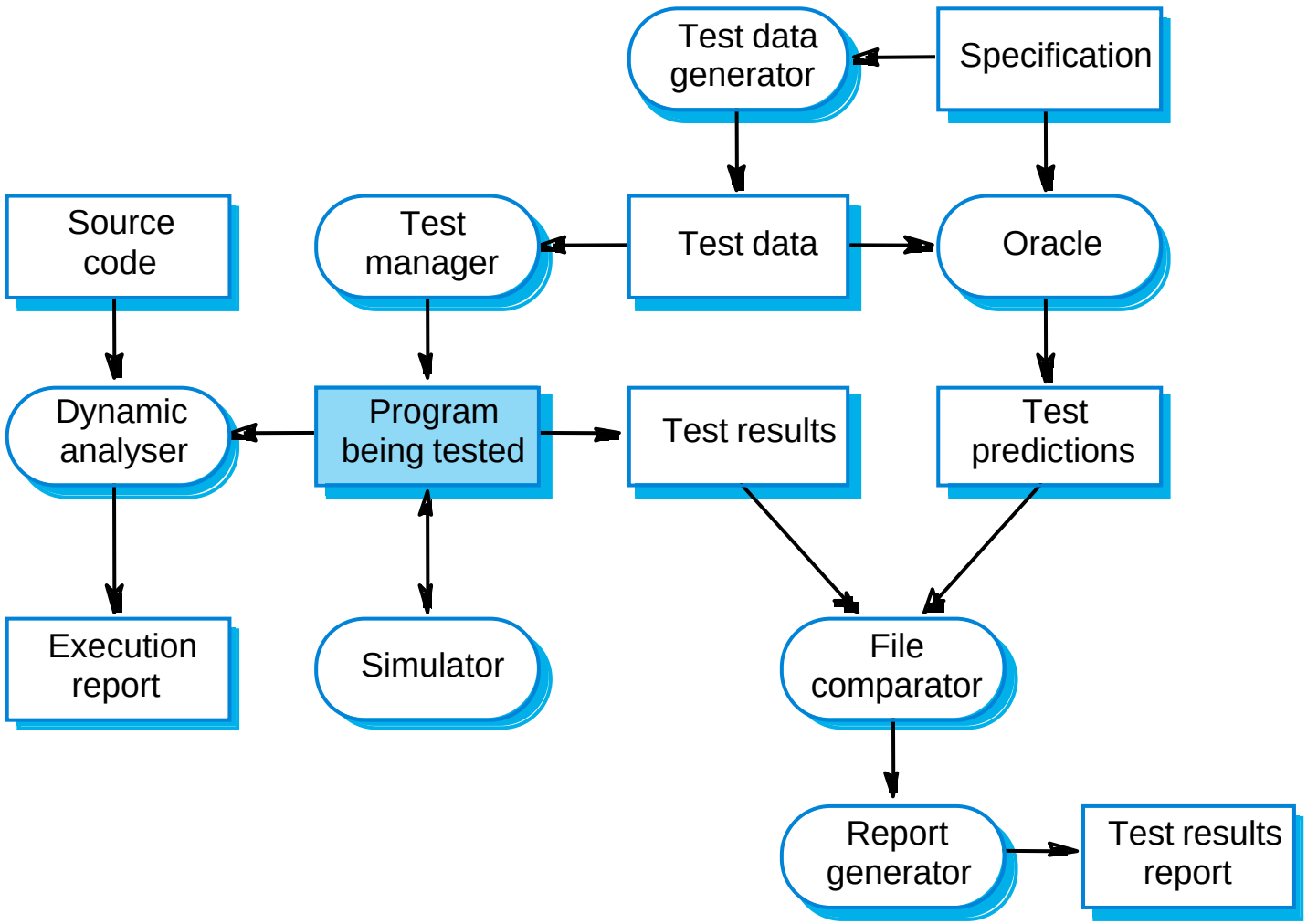
- Test cases should be derived so that all of these paths are executed.

- A dynamic program analyser may be used to check that paths have been executed.

Test automation

- Testing is an expensive process phase.
- Testing workbenches provide a range of tools to reduce the time required and total testing costs.
  - Systems such as Junit support the automatic execution of tests.
  - Other systems also support the creation of automated tests without much programming (typically for GUI tests).
- Most testing workbenches are open systems because testing needs are organisation-specific.
- They are sometimes difficult to integrate with closed design and analysis workbenches.

A testing workbench



Testing workbench adaptation

- Scripts may be developed for user interface simulators and patterns for test data generators.
- Test outputs may have to be prepared manually for comparison.
  - Sometimes they can be derived by using an existing SW system.
  - Or oracles can check the correctness (at least partially).
- Special-purpose file comparators may be developed.

Key points

- Testing can show the presence of defects in a system.
  - It cannot prove there are no remaining defects.
- Component developers are responsible for component testing.
- System testing is done by a separate team.
  - The last is called release testing or acceptance testing.
- Integration testing checks increments of the system.
- Use experience and guidelines to design effective test cases for defect testing.

Key points (2)

- Interface testing aims at discovering defects in the interfaces of composite components.
  - Used during component testing and integration testing.
- Equivalence partitioning is a technique for designing test cases.
  - All cases in a partition should behave in the same way.
- Structural testing relies on analysing a program and deriving tests from this analysis.
- Test automation reduces testing costs by supporting the test process with a range of software tools.