# Software testing



## 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

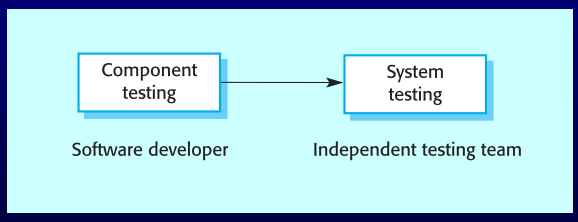


## The testing process

- 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






## Defect testing

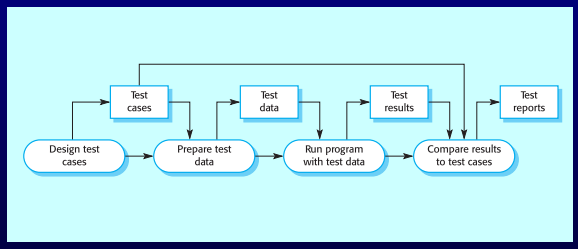
- The goal of defect testing is to discover defects in programs
- A *successful* defect test is a test which causes a program to behave in an anomalous way
- Tests show the presence not the absence of defects

# Testing process goals

- 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.
- 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.



# The software testing process






# Testing policies

- Only exhaustive testing can show a program is free from defects. However, exhaustive testing is impossible,
- Testing policies define the approach to be used in selecting system tests:
  - 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 with correct and 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 to be delivered as a black-box.

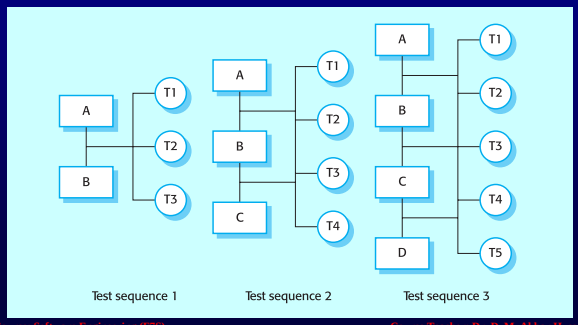


# Integration testing

- Involves building a system from its components and testing it for problems that arise from component interactions.
- Top-down integration
  - Develop the skeleton of the system and populate it with components.
- Bottom-up integration
  - Integrate infrastructure components then add functional components.
- To simplify error localisation, systems should be incrementally integrated.



# Incremental integration testing



# 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.

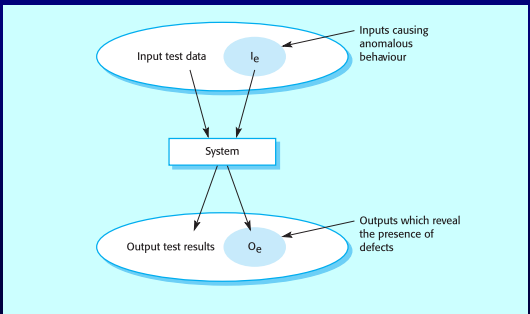


# Release testing

- 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;
  - 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 student in Scotland is studying American History and has been asked to write a paper on Õfrontier mentality in the American West from 1840 to 1880Ó.To do this, she needs to find sources from a range of libraries. She logs on to the LIBSYS system and uses the search facility to discover if she can acce ss original documents from that time. She discovers sources in various US university libraries and down loads copies of some of these. However, for one document, she needs to have confirmation from her university 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 libraryÕs server and printed for her. She receives a message from LIBSYS telling her that she will receive an e-mail message when th e printed document is available for collection.



# System tests

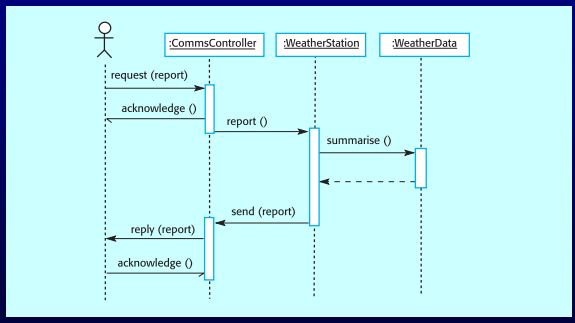
1. Test the login mechanism using correct and incorrect logins to check that valid users are accepted and invalid users are rejected.
2. Test the search facility using different queries against known sources to check that the search mechanism is actually finding documents.
3. Test the system presentation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

## 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 come to light.
- Stressing the system test failure behaviour.. Systems should not fail 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

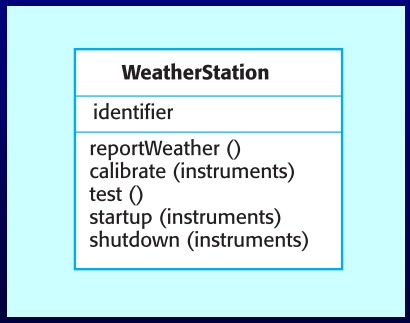
- Component or unit testing is the process of testing individual components in isolation.
- It is a defect testing process.
- Components may be:
  - Individual functions or methods within an object;
  - Object classes with several attributes and methods;
  - Composite components with defined interfaces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



## Object class testing

- Complete test coverage of a class involves
  - 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.

## 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 -> Transmitting -> Waiting

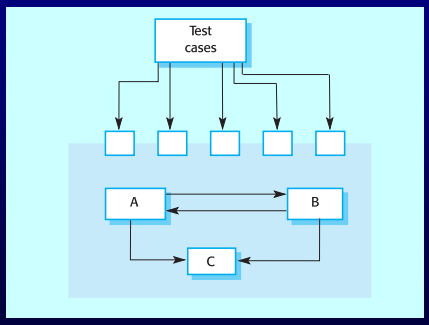


## Interface testing

- 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.



## 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-system.s



## 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 guidelines

- 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.
- 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 accountâ permanent storage area.



## LIBSYS tests

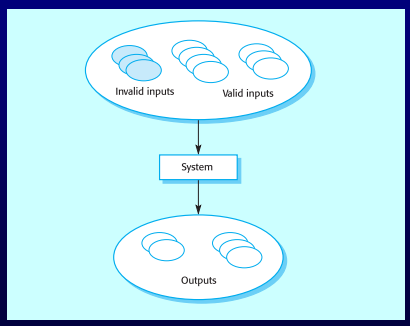
- Initiate user search for searches for items that are known to be present and known not to be present, where the set of databases includes 1 database.
- Initiate user searches for items that are known to be present and known not to be present, where the set of databases includes 2 databases
- Initiate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



## Partition testing

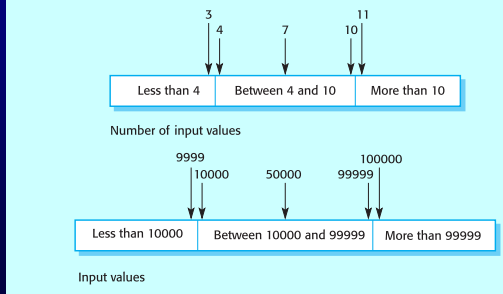
- Input data and output results often fall into different classes where all members of a class are related.
- Each of these classes is an equivalence partition or domain where the program behaves in an equivalent way for each class member.
- Test cases should be chosen from each partition.

## Equivalence partitioning






## Equivalence partitions






## 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.
- 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 |

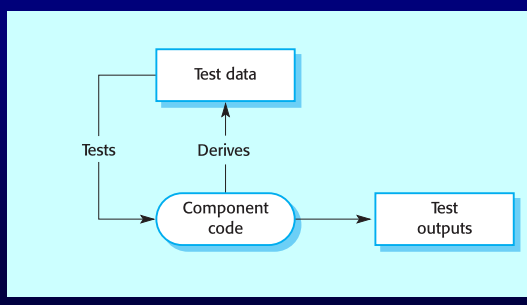
| **Input sequence** (T) | **Key** (Key) | **Output** (Found, L) |
|---|---|---|
| 17 | 17 | true, 1 |
| 17 | 0 | false, ?? |
| 17, 29, 21, 23 | 17 | true, 1 |
| 41, 18, 9, 31, 30, 16, 45 | 45 | true, 7 |
| 17, 18, 21, 23, 29, 41, 38 | 23 | true, 4 |
| 21, 23, 29, 33, 38 | 25 | false, ?? |

## Structural testing

- 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).



## Structural testing



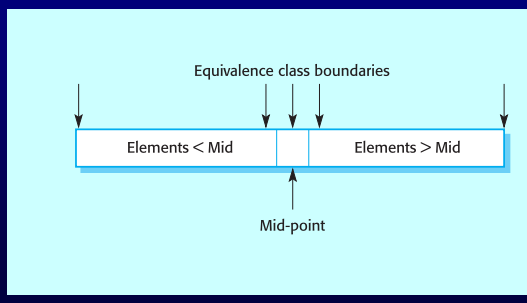



## Binary search - equiv. partitions

- 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

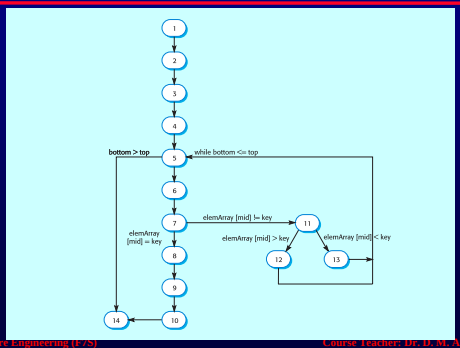
| **Input array** (T) | **Key** (Key) | **Output** (Found, L) |
|---|---|---|
| 17 | 17 | true, 1 |
| 17 | 0 | false, ?? |
| 17, 21, 23, 29 | 17 | true, 1 |
| 9, 16, 18, 30, 31, 41, 45 | 45 | true, 7 |
| 17, 18, 21, 23, 29, 38, 41 | 23 | true, 4 |
| 17, 18, 21, 23, 29, 33, 38 | 21 | true, 3 |
| 12, 18,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



## 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.

# 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

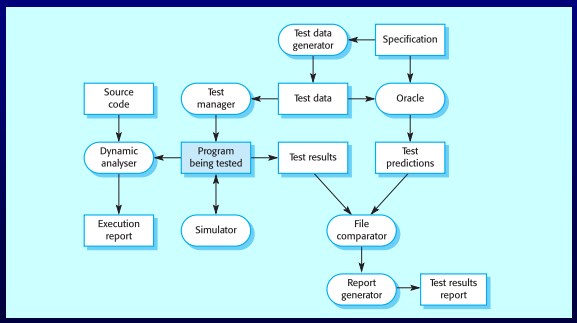


# Test automation

- 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.
- 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.
- Special-purpose file comparators may be developed.



# Key points

- Testing can show the presence of faults in a system; it cannot prove there are no remaining faults.
- Component developers are responsible for component testing; system testing is the responsibility of a separate team.
- Integration testing is testing increments of the system; release testing involves testing a system to be released to a customer.
- Use experience and guidelines to design test cases in defect testing.

# Key points

- Interface testing is designed to discover defects in the interfaces of composite components.
- Equivalence partitioning is a way of discovering test cases - all cases in a partition should behave in the same way.
- Structural analysis 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.