UNIVERSITY of GUELPH

ENGINEERING

Software Testing

Fattane Zarrinkalam

Program testing

- Testing is intended to show that a program does what it is intended to do and to discover program defects before it is put into use.
- When you test software, you execute a program using artificial data.
- You check the results of the test run for errors, anomalies or information about the program's non-functional attributes.
- Can reveal the presence of errors NOT their absence.
- Testing is part of a more general verification and validation process, which also includes static validation techniques.

Program Testing Goals

- Validation testing
  - You expect the system to perform correctly using a given set of test cases that reflect the system's expected use.
- Defect testing
  - The test cases are designed to expose defects. The test cases in defect testing can be deliberately obscure and need not reflect how the system is normally used.

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 behavior 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.

Verification vs validation

- Verification: "Are we building the product right".
  - The software should conform to its specification.

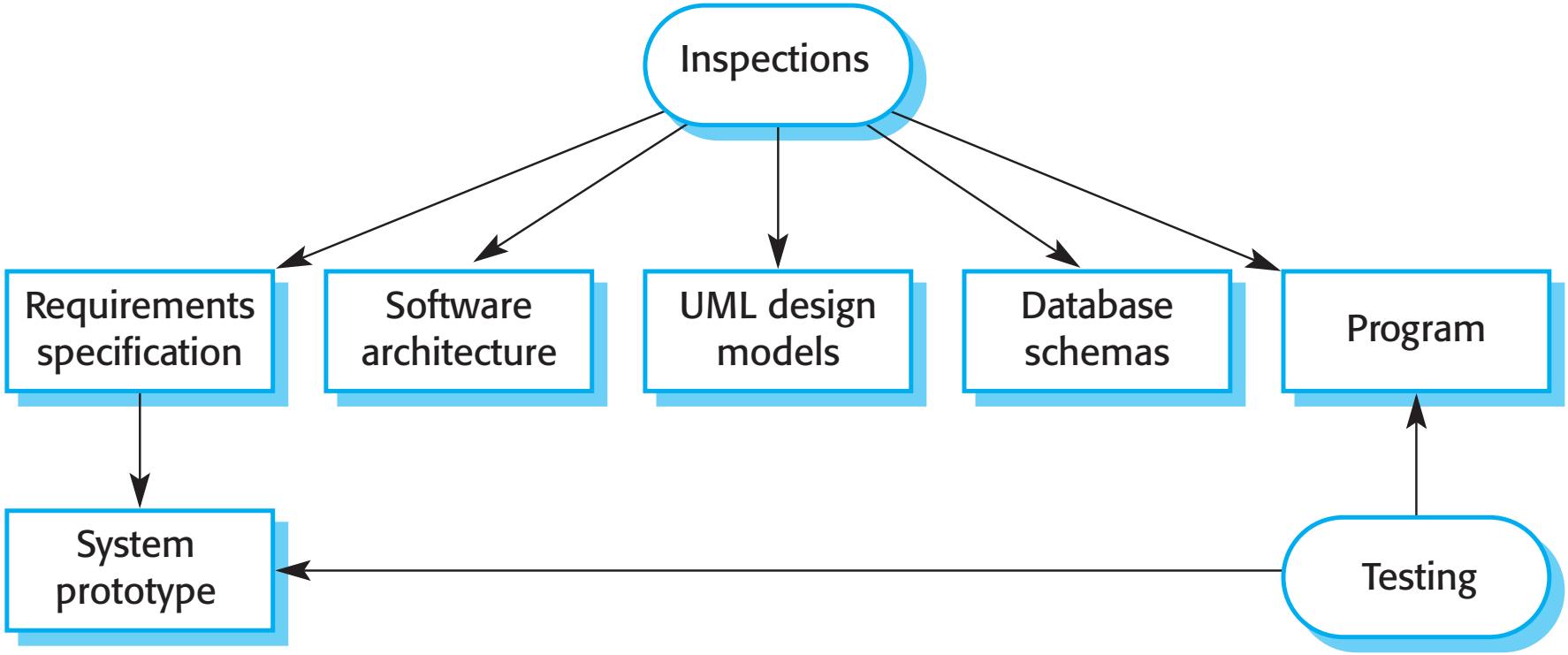
- Validation: "Are we building the right product".
  - The software should do what the user really requires.

Inspections and testing

- Software inspections Concerned with analysis of the static system representation to discover problems *(*static verification)
  - May be supplement by tool-based document and code analysis.

- Software testing Concerned with exercising and observing product behaviour (dynamic verification)
  - The system is executed with test data and its operational behaviour is observed.

Inspections and testing



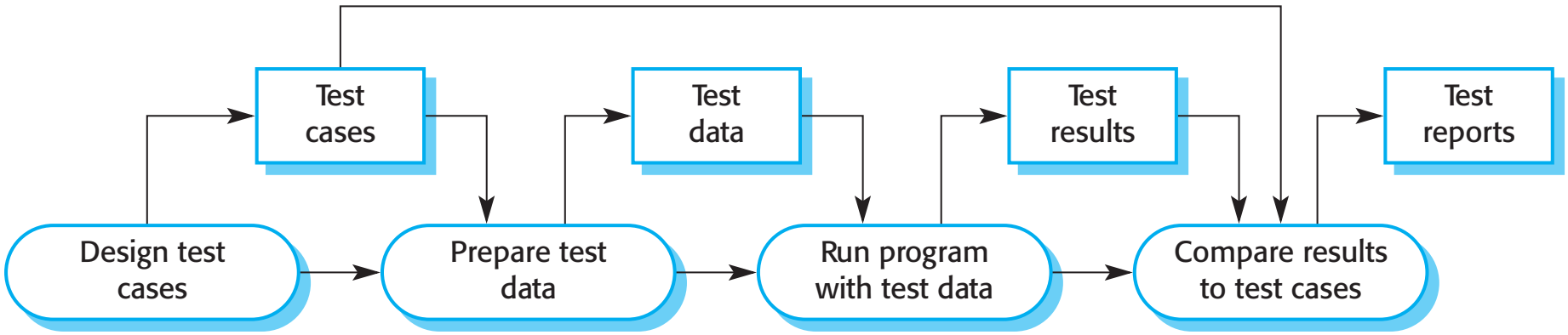
Advantages of inspections

- During testing, errors can mask (hide) other errors. Because inspection is a static process, you don't have to be concerned with interactions between errors.
- Incomplete versions of a system can be inspected without additional costs. If a program is incomplete, then you need to develop specialized test harnesses to test the parts that are available.
- As well as searching for program defects, an inspection can also consider broader quality attributes of a program, such as compliance with standards, portability and maintainability.

Inspections and testing

- Inspections and testing are complementary and not opposing verification techniques.
- Both should be used during the V & V process.
- Inspections can check conformance with a specification but not conformance with the customer's real requirements.
- Inspections cannot check non-functional characteristics such as performance, usability, etc.

A model of the software testing process



Stages of testing

- Development testing, where the system is tested during development to discover bugs and defects.
- Release testing, where a separate testing team test a complete version of the system before it is released to users.
- User testing, where users or potential users of a system test the system in their own environment.

Development testing

Development testing

- Development testing includes all testing activities that are carried out by the team developing the system.
  - Unit testing, where individual program units or object classes are tested. Unit testing should focus on testing the functionality of objects or methods.
  - Component testing, where several individual units are integrated to create composite components. Component testing should focus on testing component interfaces.
  - System testing, where some or all of the components in a system are integrated and the system is tested as a whole. System testing should focus on testing component interactions.

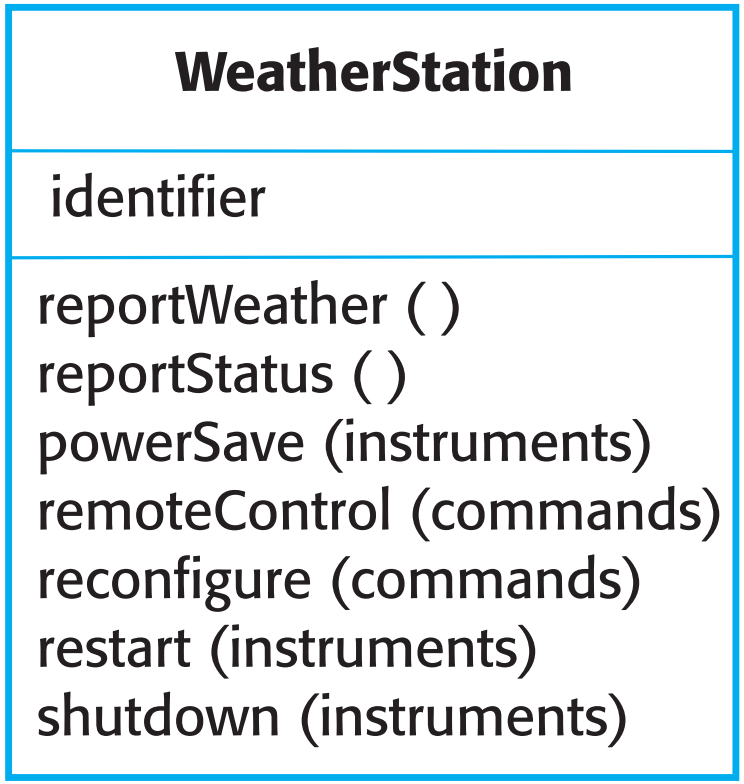
Unit testing

- Unit testing is the process of testing individual components in isolation.
- It is a defect testing process.
- Units 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.

The 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:
  - Shutdown -> Running-> Shutdown
  - Configuring-> Running-> Testing -> Transmitting -> Running
  - Running-> Collecting-> Running-> Summarizing -> Transmitting -> Running

Automated testing

- Whenever possible, unit testing should be automated so that tests are run and checked without manual intervention.
- In automated unit testing, you make use of a test automation framework (such as JUnit) to write and run your program tests.
- Unit testing frameworks provide generic test classes that you extend to create specific test cases. They can then run all of the tests that you have implemented and report, often through some GUI, on the success of otherwise of the tests.

Automated test components

- A setup part, where you initialize the system with the test case, namely the inputs and expected outputs.
- A call part, where you call the object or method to be tested.
- An assertion part where you compare the result of the call with the expected result. If the assertion evaluates to true, the test has been successful if false, then it has failed.

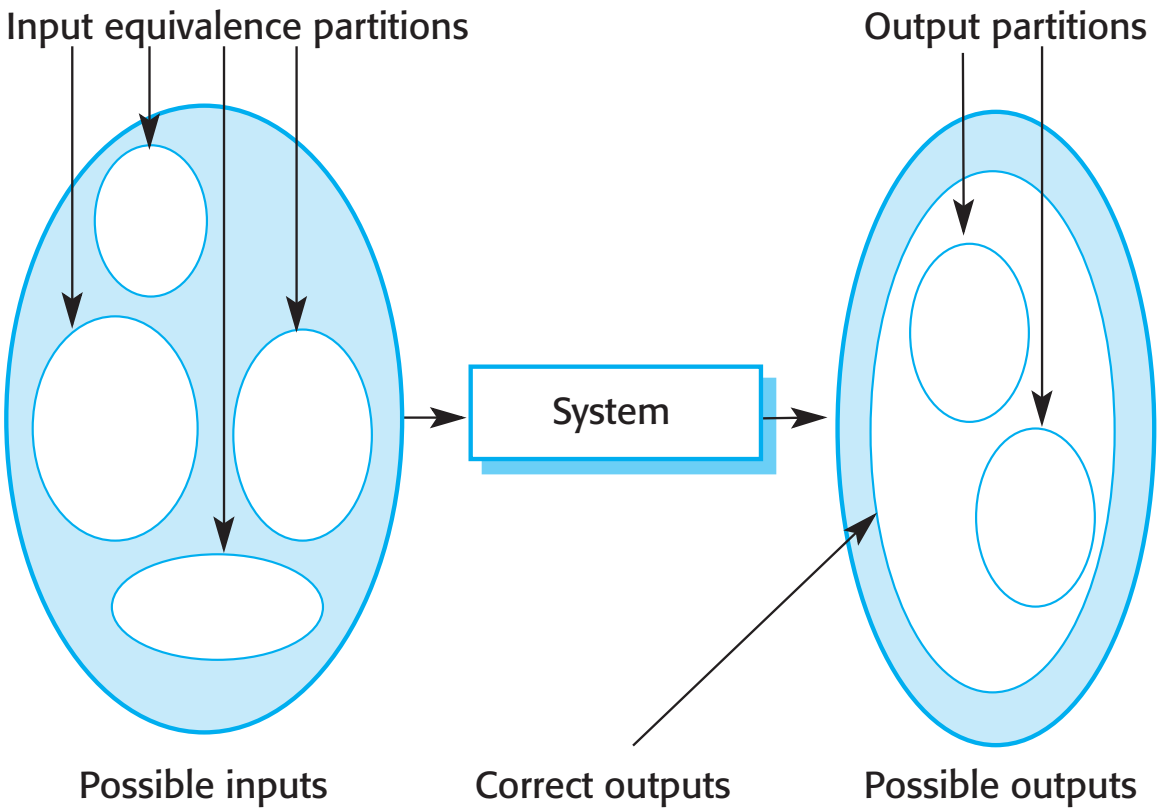
Testing strategies

- Partition testing, where you identify groups of inputs that have common characteristics and should be processed in the same way.
  - You should choose tests from within each of these groups.
- Guideline-based testing, where you use testing guidelines to choose test cases.
  - These guidelines reflect previous experience of the kinds of errors that programmers often make when developing components.

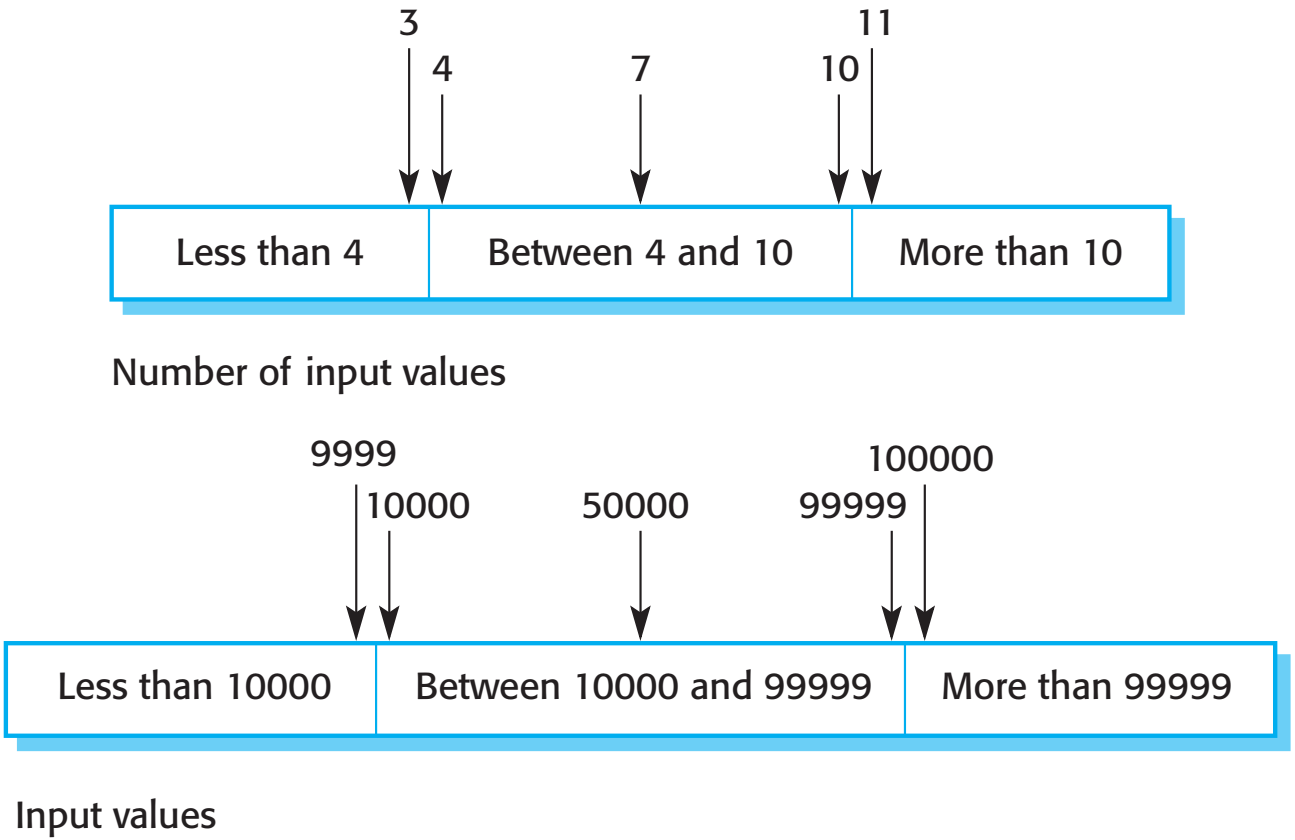
Partition testing

- 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



Testing guidelines (sequences)

- Test software with sequences which have only a single value.
- Use sequences of different sizes in different tests.
- Test with sequences of zero length.

General testing guidelines

- Choose inputs that force the system to generate all error messages
- Design inputs that cause input buffers to overflow
- Force invalid outputs to be generated
- Force computation results to be too large or too small.

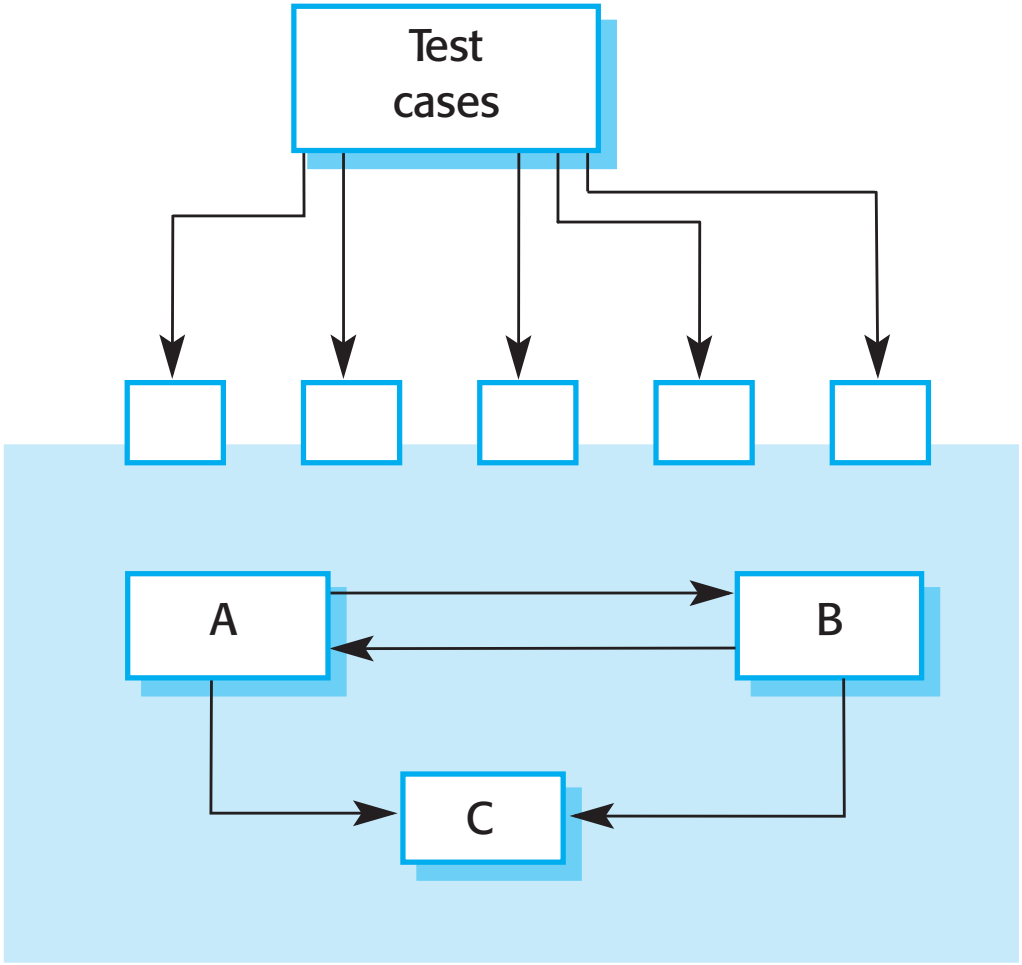
Component testing

- Software components are often composite components that are made up of several interacting objects.
  - For example, in the weather station system, the reconfiguration component includes objects that deal with each aspect of the reconfiguration.
- You access the functionality of these objects through the defined component interface.
- Testing composite components should therefore focus on showing that the component interface behaves according to its specification.
  - You can assume that unit tests on the individual objects within the component have been completed.

Interface testing

- Objectives are to detect faults due to interface errors or invalid assumptions about interfaces.
- Interface types
  - Parameter interfaces Data passed from one method or 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 testing



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.

System testing

- System testing during development involves integrating components to create a version of the system and then testing the integrated system.
- The focus in system testing is testing the interactions between components.
- System testing checks that components are compatible, interact correctly and transfer the right data at the right time across their interfaces.
- System testing tests the emergent behavior of a system.

System and component testing

- During system testing, reusable components that have been separately developed and off-the-shelf systems may be integrated with newly developed components. The complete system is then tested.
- Components developed by different team members or sub-teams may be integrated at this stage. System testing is a collective rather than an individual process.
  - In some companies, system testing may involve a separate testing team with no involvement from designers and programmers.

Summary

Summary

- Testing can only show the presence of errors in a program. It cannot demonstrate that there are no remaining faults.
- Development testing is the responsibility of the software development team. A separate team should be responsible for testing a system before it is released to customers.
- Development testing includes unit testing, in which you test individual objects and methods component testing in which you test related groups of objects and system testing, in which you test partial or complete systems.