SOFTWARE ENGINEERING

CO3001

CHAPTER 8 – SOFTWARE TESTING

WEEK 11

Adapted from https://iansommerville.com/software-engineering-book/slides/

TOPICS COVERED

- ✓ Development testing
- ✓ Test-driven development
- ✓ Release testing
- ✓ User testing

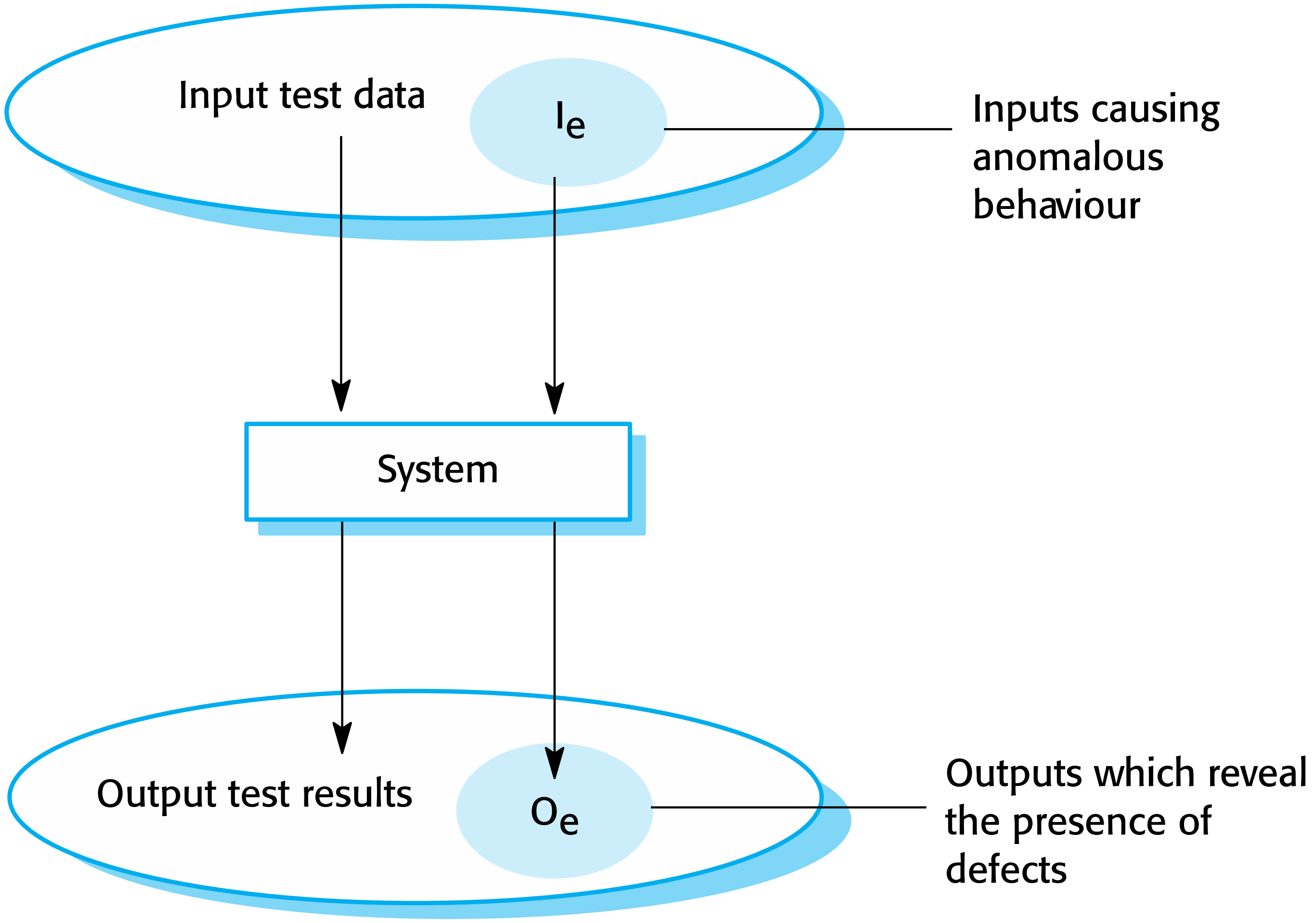
PROGRAM TESTING

- ✓ 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.
- ✓ **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

- ✓ To demonstrate to the developer and the customer that the software meets its requirements.
  - validation testing
- ✓ To discover situations in which the behavior of the software is incorrect, undesirable or does not conform to its specification.
  - defect testing

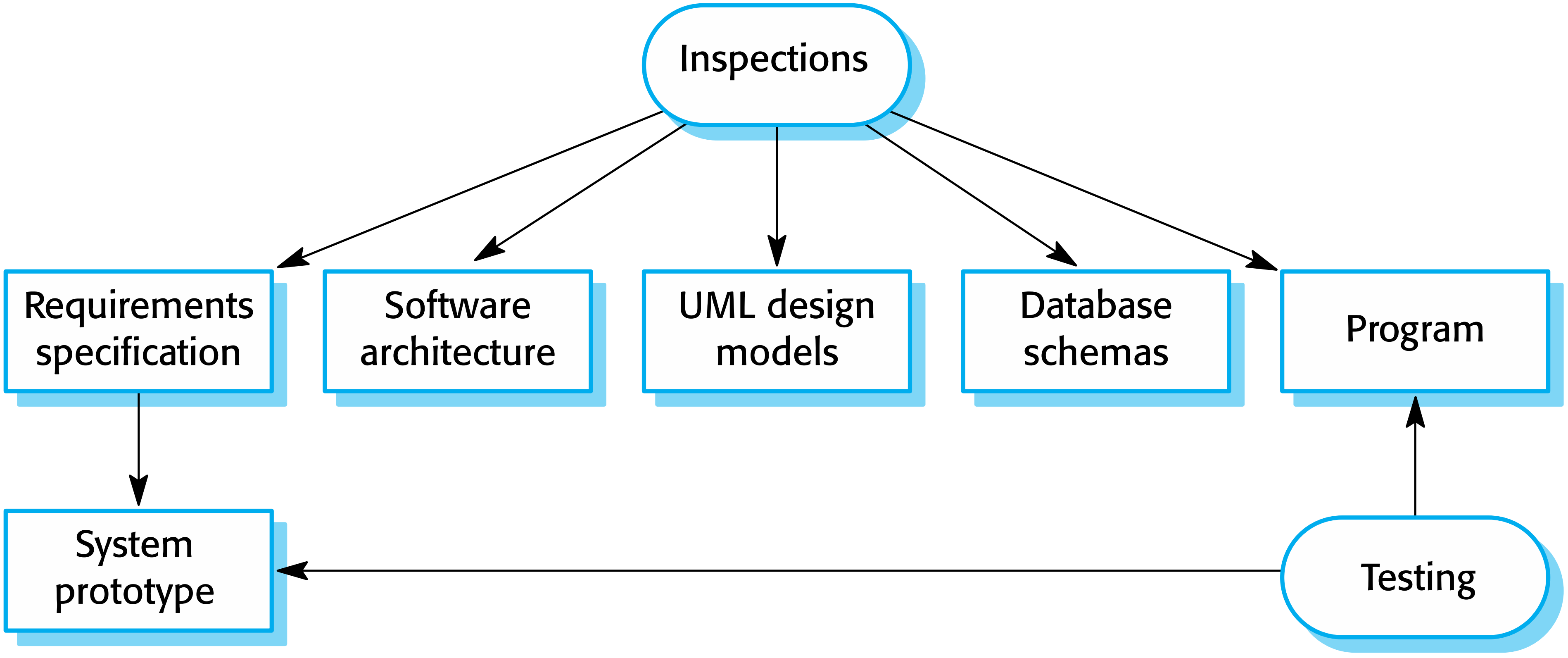
AN INPUT-OUTPUT MODEL OF PROGRAM TESTING


Input test data
I_e
Inputs causing anomalous behaviour
System
Output test results
O_e
Outputs which reveal the presence of defects

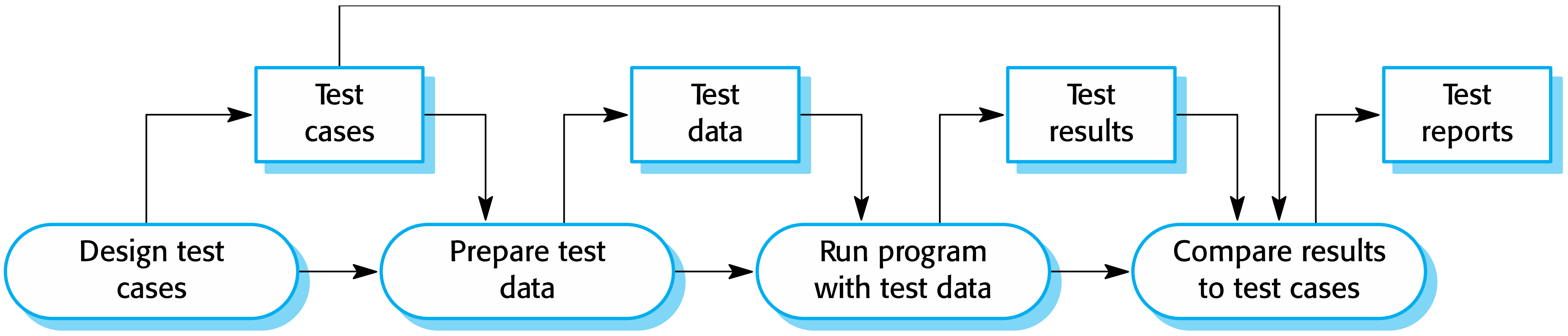
INSPECTIONS AND TESTING

- ✓ 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



A MODEL OF THE SOFTWARE TESTING PROCESS



STAGES OF TESTING

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

DEVELOPMENT TESTING

DEVELOPMENT TESTING

carried out by the team developing the system.

- ✓ Unit testing:
  - for individual program units or object classes
  - focus on testing the functionality of objects or methods.
- ✓ Component testing:
  - several individual units are integrated to create composite components
  - focus on testing component interfaces.
- ✓ System testing:
  - some or all of the components in a system are integrated and the system is tested as a whole
  - 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.

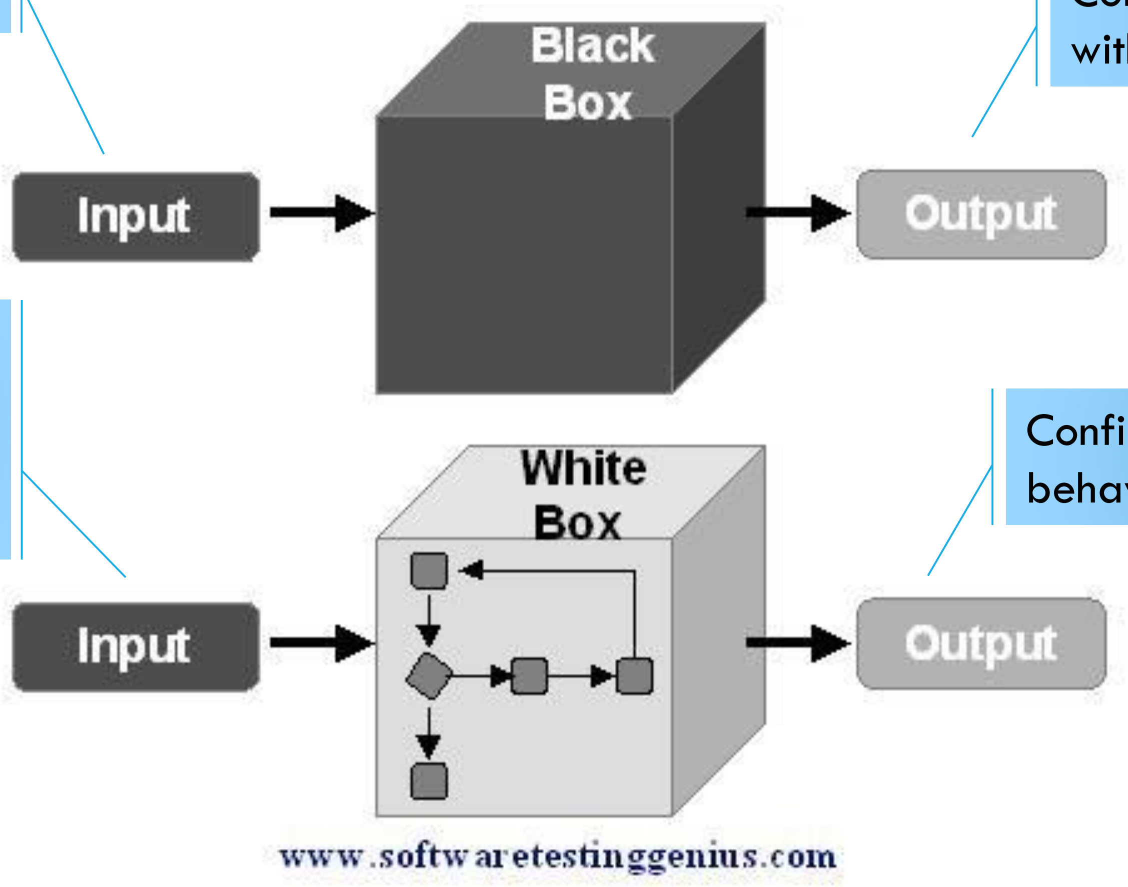
UNIT TESTING: BLACK-/WHITE-BOX TEST

from requirements

Compare actual output with require output

from requirements & key design elements

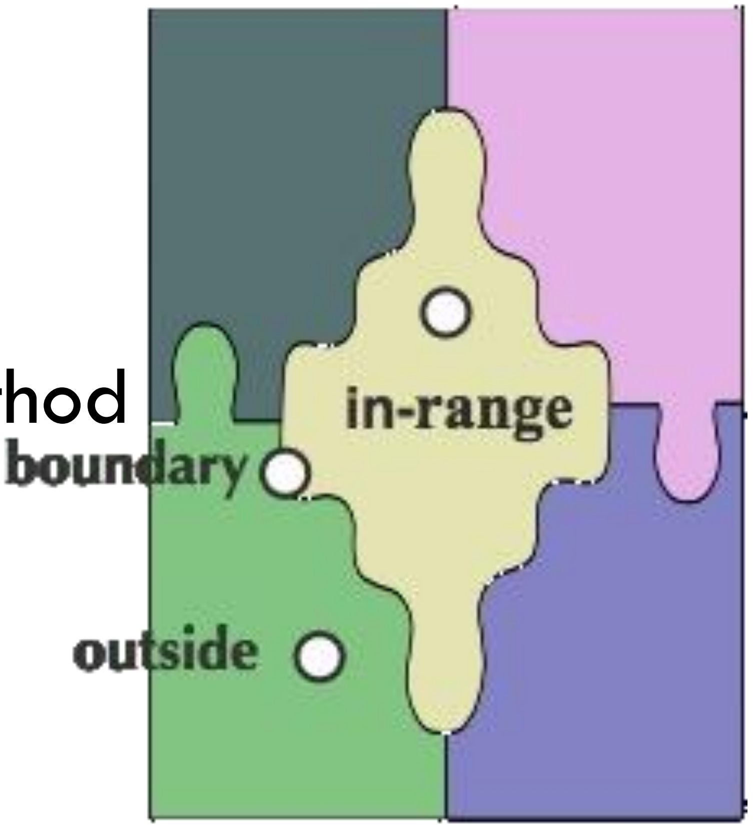
Confirm expected behaviour



Gray-box: mix of black- and white-box testing

BLACK-BOX TESTING

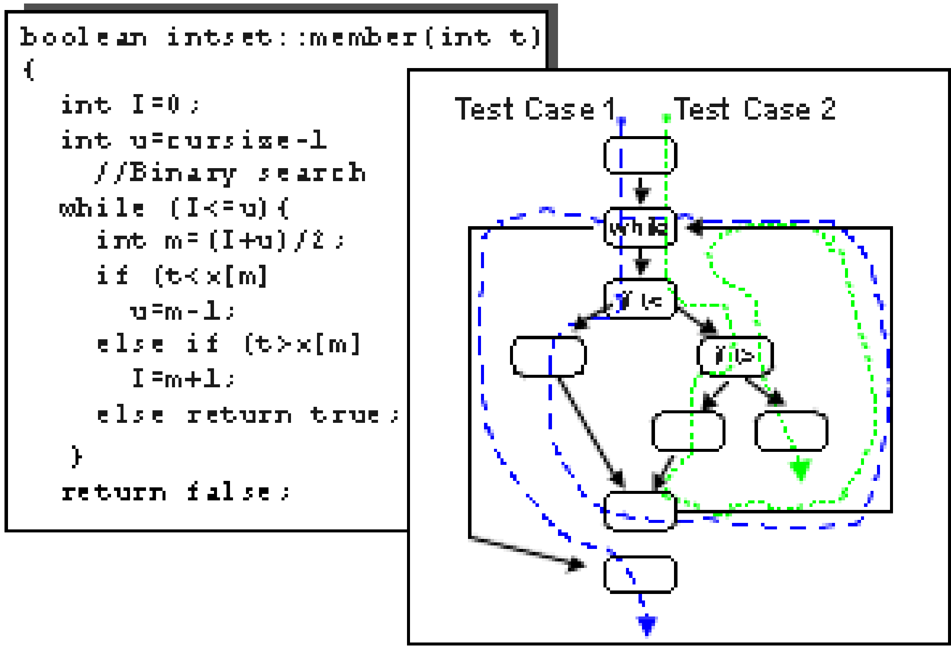
- ✓ Input
  - Partitioning approach
- ✓ Execution/Simulation method
  - ?
- ✓ Expected output
  - ?



WHITE-BOX TESTING

- ✓ Statement coverage
  - Good
  - Not sufficient

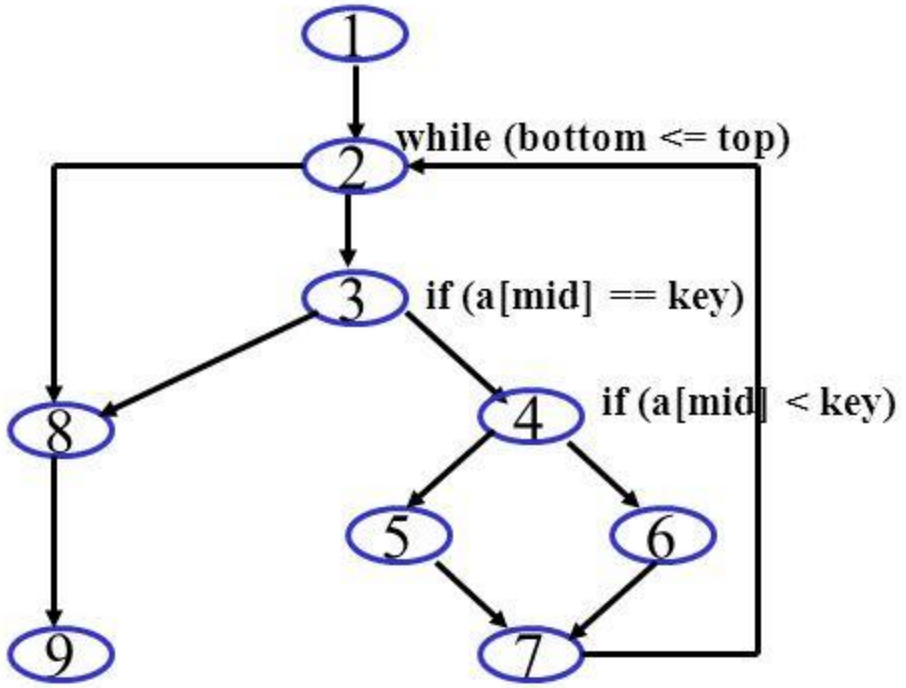
- ✓ Decision/branching/
  - Loop?



```
boolean intset::member(int t)
{
   int I=0;
   int u=cursize-1
     //Binary search
   while (I<=u){
     int m=(I+u)/2;
     if (t<x[m])
       u=m-1;
     else if (t>x[m])
       I=m+1;
     else return true;
   }
   return false;
```

Example: CFG of a binary search routine ([Som00], ch. 20)

```
class BinSearch {
  ...
  public static void search (int key, int[] a,Rez r)
  {
    int mid; int bottom = 0;
    int top = a.length – 1;
    r.found = false;  r.index = -1;
    while (bottom <= top) {
        mid = (top + bottom) / 2;
        if (a[mid] == key)  {
            r.index = mid;
            r.found = true;
            return;
        } else  {
            if (a[mid] < key)
              bottom = mid + 1;
            else
              top = mid - 1;
        }
    }
  }
}
```


$$\begin{tikzcd}
 & & 1 \arrow[d] & & \\
 & & 2 \arrow[d] \arrow[dll] & & \\
 & & 3 \arrow[dll] \arrow[dr] & & \\
8 \arrow[dd] & & & 4 \arrow[dl] \arrow[dr] & \\
 & & 5 \arrow[dr] & & 6 \arrow[dl] \\
9 & & & 7 \arrow[uuuul, bend right] &
\end{tikzcd}$$

cc(CFG)
= noEdges – noNodes + 2
= noBinaryDecisionPredicates + 1
= 4

A set of independent paths:
- **1,2,8,9**
- **1,2,3,8,9**
- **1,2,3,4,5,7,2,8,9**
- **1,2,3,4,6,7,2,8,9**

PERFORM METHOD TESTING 1/2

One way to ...

- ✓ 1. Verify operation at normal parameter values
  - (a black box test based on the unit's requirements)
- ✓ 2. Verify operation at limit parameter values
  - (black box)
- ✓ 3. Verify operation outside parameter values
  - (black box)
- ✓ 4. Ensure that all instructions execute
  - (statement coverage)
- ✓ 5. Check all paths, including both sides of all branches
  - (decision coverage)
- ✓ 6. Check the use of all called objects
- ✓ 7. Verify the handling of all data structures
- ✓ 8. Verify the handling of all files

PERFORM METHOD TESTING 2/2

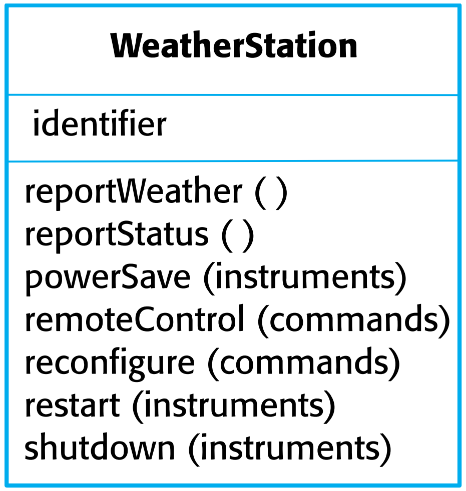
One way to ...

- ✓ 9. Check normal termination of all loops
  - (part of a correctness proof)
- ✓ 10. Check abnormal termination of all loops
- ✓ 11. Check normal termination of all recursions
- ✓ 12. Check abnormal termination of all recursions
- ✓ 13. Verify the handling of all error conditions
- ✓ 14. Check timing and synchronization
- ✓ 15. Verify all hardware dependencies

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.

EXAMPLE: WEATHER STATION TESTING



- ✓ Define test cases for reportWeather, calibrate, test, startup and shutdown.
- ✓ 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
- ✓ Use of a test automation framework (such as JUnit)

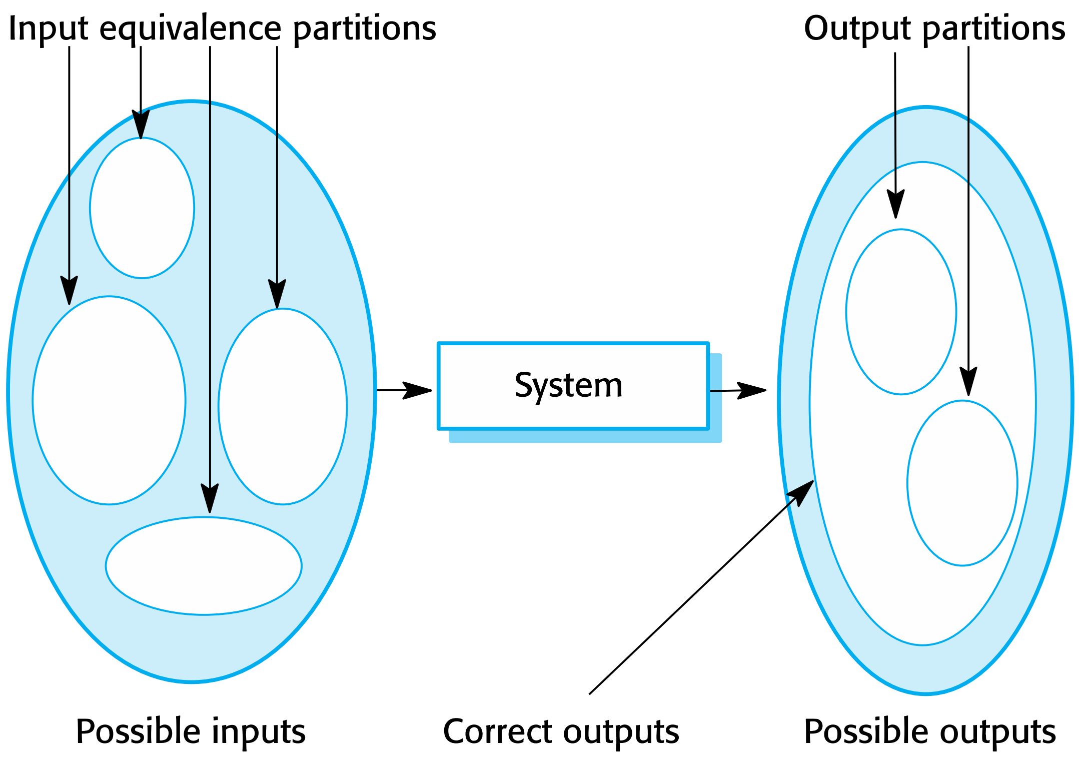
UNIT TEST EFFECTIVENESS

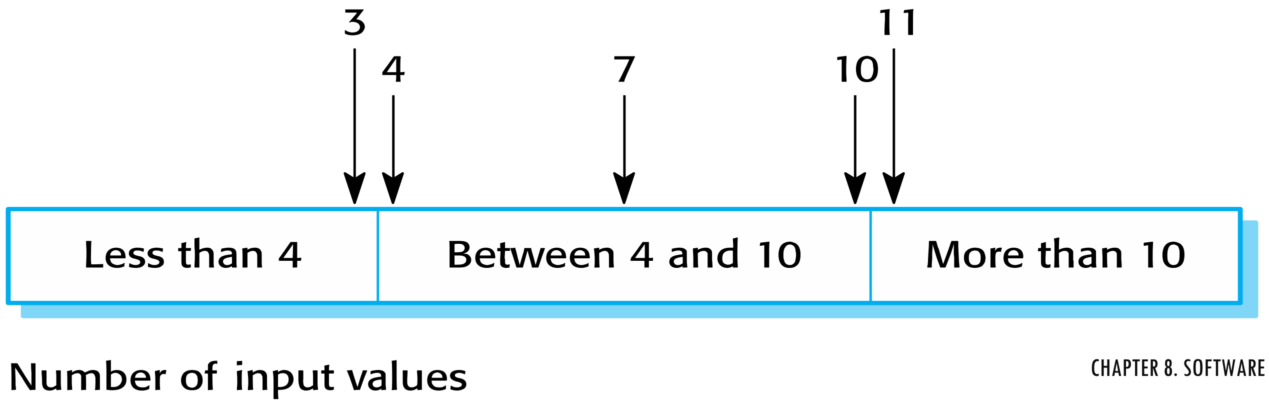
- ✓ Show that, when used as expected, the component does what it is supposed to do.
- ✓ If there are defects in the component, these should be revealed by test cases.

PARTITION TESTING

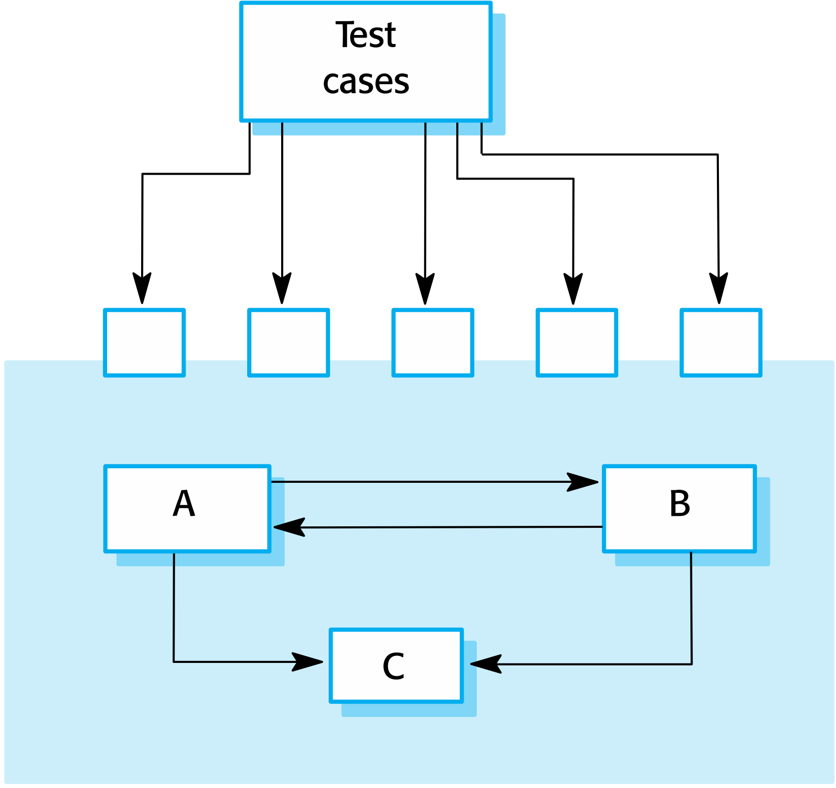
- ✓ Input data and output results often fall into different classes where all members of a class are related.
- ✓ Each of these classes is an equivalence partition or domain where the program behaves in an equivalent way for each class member.
- ✓ Test cases should be chosen from each partition.

EQUIVALENCE PARTITIONING





INTERFACE TESTING



- ✓ Detect faults due to
  - interface errors
  - or invalid assumptions about interfaces.
- ✓ Interface types
  - Parameter interfaces
  - Shared memory interfaces
  - Procedural interfaces
  - Message passing interfaces

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.

SYSTEM TESTING

System testing during development = to create a version of the system and then testing the integrated system.

- ✓ Focus on 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.

- ✓ And tests the emergent behaviour of a system.

TYPES OF SYSTEM TESTS

- ✓ Volume
  - Subject product to large amounts of input.
- ✓ Usability
  - Measure user reaction (e.g., score 1-10).
- ✓ Performance
  - Measure speed under various circumstances.
- ✓ Configuration
  - Configure to various hardware / software
- ✓ Compatibility
  - with other designated applications
- ✓ Reliability / Availability
  - Measure up-time over extended period.
- ✓ Security
  - Subject to compromise attempts.
- ✓ Resource usage
  - Measure usage of RAM and disk space etc.
- ✓ Install-ability
  - Install under various circumstances.
- ✓ Recoverability
  - Force activities that take the application down.
- ✓ Serviceability
  - Service application under various situations.
- ✓ Load / Stress
  - Subject to extreme data & event traffic

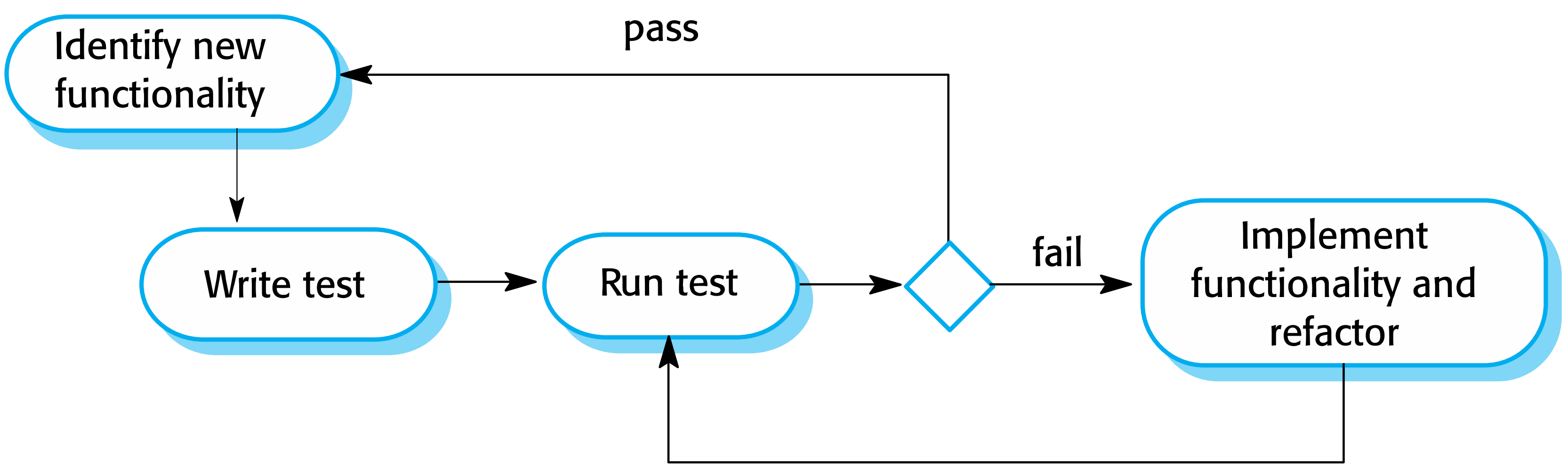
USE-CASE TESTING

The use-cases developed to identify system interactions can be used as a basis for system testing.

- ✓ Each use case usually involves several system components so testing the use case forces these interactions to occur.
  - The sequence diagrams associated with the use case documents the components and interactions that are being tested.

TEST-DRIVEN DEVELOPMENT

inter-leave testing and code development


Identify new functionality
pass
Write test
Run test
fail
Implement functionality and refactor

Benefits of test-driven development

- Code coverage
- Regression testing
- Simplified debugging
- System documentation

REGRESSION TESTING

Test the system to check that changes have not 'broken' previously working code.

- ✓ Better with automated testing
- ✓ All tests are re-run every time a change is made to the program.

- ✓ Tests must run 'successfully' before the change is committed.

RELEASE TESTING

RELEASE TESTING

Test a particular release of a system that is intended for use outside of the development team.

- ✓ Primary goal: to convince that it is good enough for use.
  - Show that the system delivers its specified functionality, performance and dependability, and that it does not fail during normal use.
- ✓ Is usually a black-box testing
  - tests are only derived from the system specification.
- ✓ Is a form of system testing.

REQUIREMENTS BASED TESTING

Involves examining each requirement and developing a test or tests for it.

- Example: Mentcare system requirements:
  - If a patient is known to be allergic to any particular medication, then prescription of that medication shall result in a warning message being issued to the system user.
  - Set up a patient record with no known allergies. Prescribe medication for allergies that are known to exist. Check that a warning message is not issued by the system.
  - Set up a patient record with a known allergy. Prescribe the medication to that the patient is allergic to, and check that the warning is issued by the system.
  - Set up a patient record in which allergies to two or more drugs are recorded. Prescribe both of these drugs separately and check that the correct warning for each drug is issued.
  - Prescribe two drugs that the patient is allergic to. Check that two warnings are correctly issued.
  - Prescribe a drug that issues a warning and overrule that warning. Check that the system requires the user to provide information explaining why the warning was overruled.

PERFORMANCE TESTING

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

- ✓ Tests should reflect the profile of use of the system.
- ✓ Is usually a series of tests
  - the load is steadily increased until the system performance becomes unacceptable.
- ✓ Stress testing
  - is a form of performance testing where the system is deliberately overloaded to test its failure behaviour.

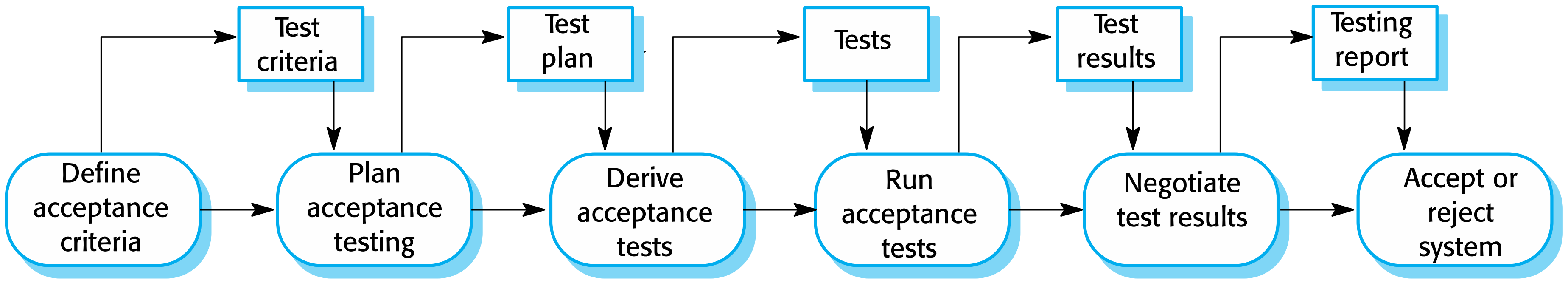
USER TESTING

USER TESTING

A stage in which users or customers provide input and advice on system testing.

- ✓ User testing is essential, even when comprehensive system and release testing have been carried out.
- ✓ Types of user-testing
  - Alpha testing
  - Beta testing
  - Acceptance testing

STAGES IN THE ACCEPTANCE TESTING PROCESS



- ✓ Define acceptance criteria
- ✓ Plan acceptance testing
- ✓ Derive acceptance tests
- ✓ Run acceptance tests
- ✓ Negotiate test results
- ✓ Reject/accept system

STOPPING CRITERIA

- ✓ Completing a particular test methodology
- ✓ Estimated percent coverage for each category
- ✓ Error detection rate
- ✓ Total number of errors found
- ✓ ?

SUMMARY

- ✓ Testing can only show the presence of errors in a program. It cannot demonstrate that there are no remaining faults.
- ✓ Development testing: development team
- ✓ Development testing includes unit testing, component testing, and system testing
- ✓ When testing software: try to 'break' the software by using experience and guidelines
- ✓ Wherever possible, you should write automated tests
- ✓ Test-first development: tests are written before the code
- ✓ Scenario testing involves inventing a typical usage scenario and using this to derive test cases.
- ✓ Acceptance testing: user testing process => if the software is good enough to be deployed and used in its operational environment.