Faculty of ECMS / School of Computer Science

Software Engineering & Project
Software Testing

Software Testing

Lecture 16

I don’t test because……

- “Coding tests takes too much time”
- “I am too busy fixing bugs to write tests”
- “My code is virtually flawless”
- “Testing is boring – it stifles my creativity”
- “Testing is better done by the testing department”
- “I will go back and write tests after I get the code working”

Why Testing is Important?

- Identify defects early in the development cycle
- Many small bugs ultimately leads to chaotic system behavior
- Testing forces you to read your code – spend more time thinking than writing
- Testing affects the design of your code

Successful tests breed confidence and peace of mind

Topics Covered

- Software Testing: An Overview
- Component Testing
- System Testing
- Test-Driven Development

Testing Goals

- To demonstrate to the developers and the system customers that the software meets its requirements

  Validation testing

  A successful test shows that the system operates as intended.

- To discover faults or defects in the software where its behaviour is incorrect or not in conformance with its specification

  Defect testing

  A successful test is a test that makes the system to perform incorrectly and so exposes a defect in the system.

“Testing can only show the presence of errors, not their absence.”
– Edsger Dijkstra, 1972

To convince developers and customers that the software is *good enough* for operational use.

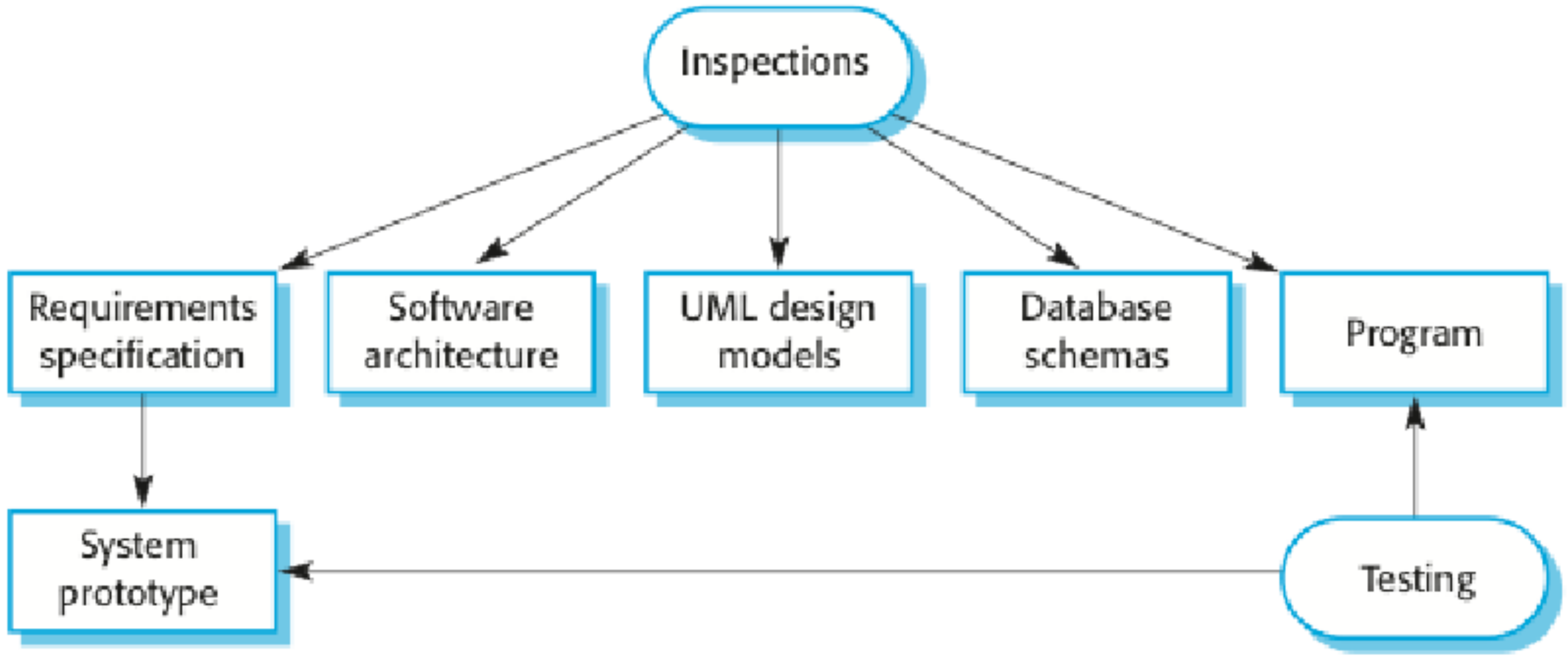
Verification vs Validation

- 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 supplemented 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



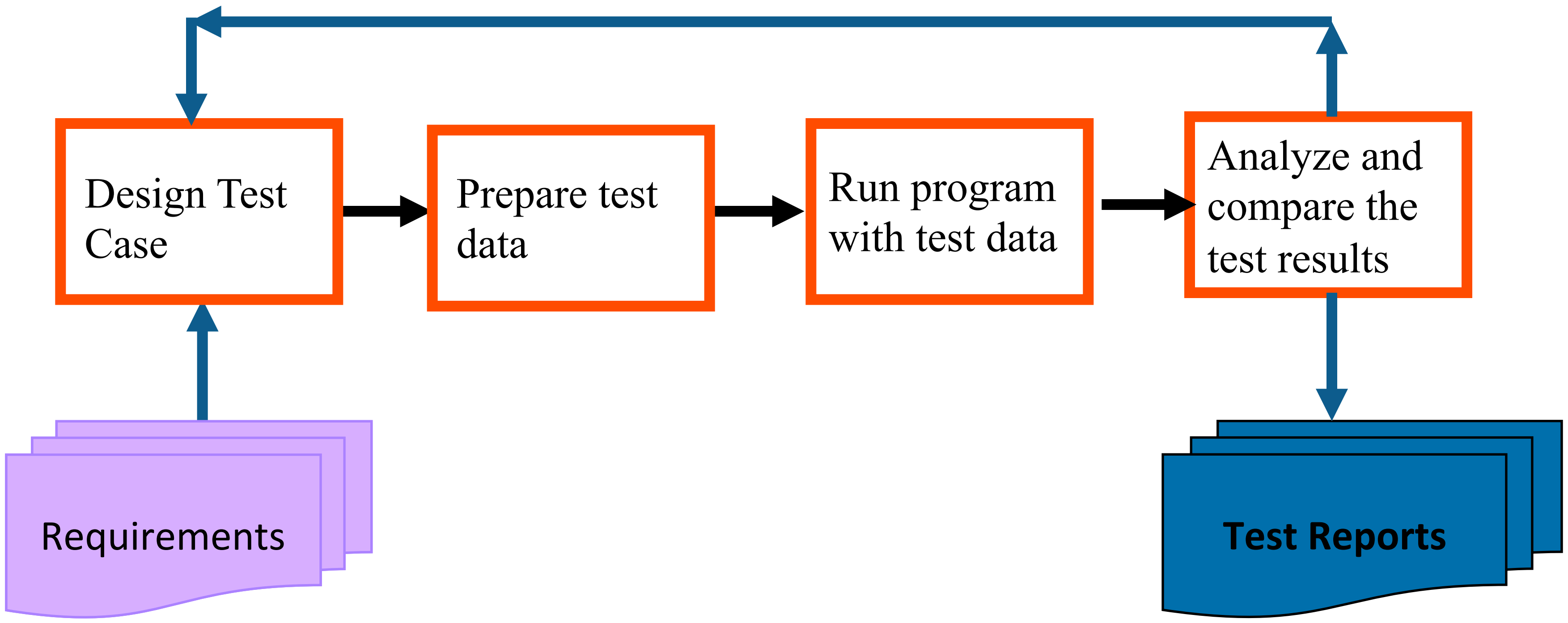
Software Inspections

- These involve people examining the source representation with the aim of discovering anomalies and defects.
- Do not require execution of a system.
  - so may be used before implementation.
- May be applied to any representation of the system (requirements, design, configuration data, test data, etc.).
- Have been shown to be an effective technique for discovering program errors.
  - (between 60% - 90%)

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.

The Software Testing Process


Design Test Case
Prepare test data
Run program with test data
Analyze and compare the test results
Requirements
Test Reports

Test Cases Design

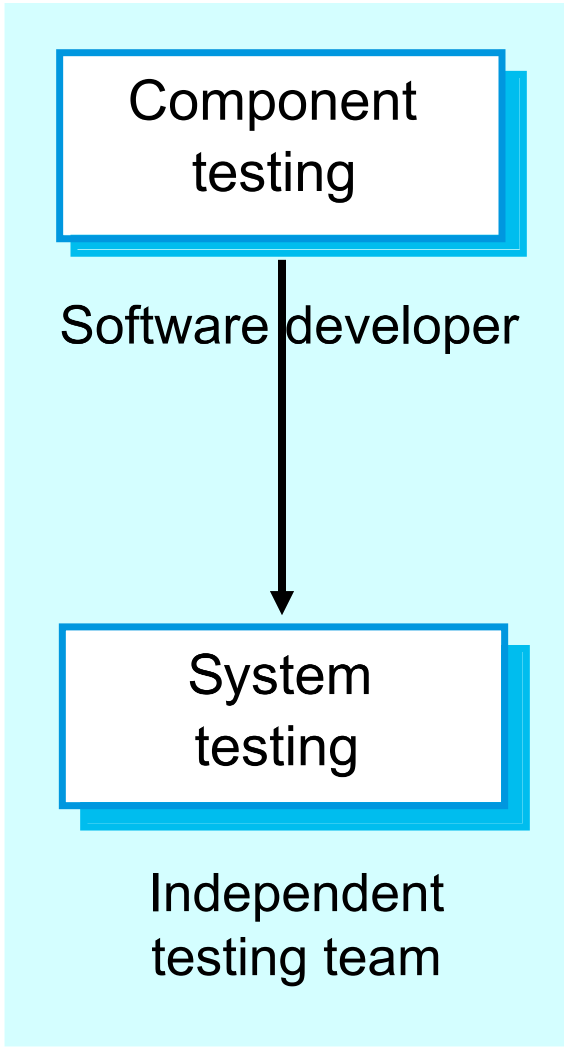
- Involves designing the test cases (inputs and outputs) used to test the system
- The goal of test case design is to create a set of tests that are effective in validation and defect testing
- We will cover several techniques on test case design throughout this lecture

Testing Policies

- Only exhaustive testing can show a program is free from defects. However,
- Testing policies or guidelines define the approach to be used in selecting tests, for instance:
  - 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.

Types of Testing

- **Component (unit) testing**
  - Testing of individual components in isolation
  - It is usually the responsibility of the component developer (except sometimes for critical systems)
  - Tests are derived from the developer' s experience and/or system specification
  - It is a defect testing process
- **System testing**
  - Testing of groups of components integrated to create a system or sub-system
  - Usually done by an independent testing team
  - Tests are based on a system specification (e.g., functional and non-functional requirements)



Topics Covered

- Software Testing: An Overview
- Component Testing
- System Testing
- Test-Driven Development

Component Testing

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

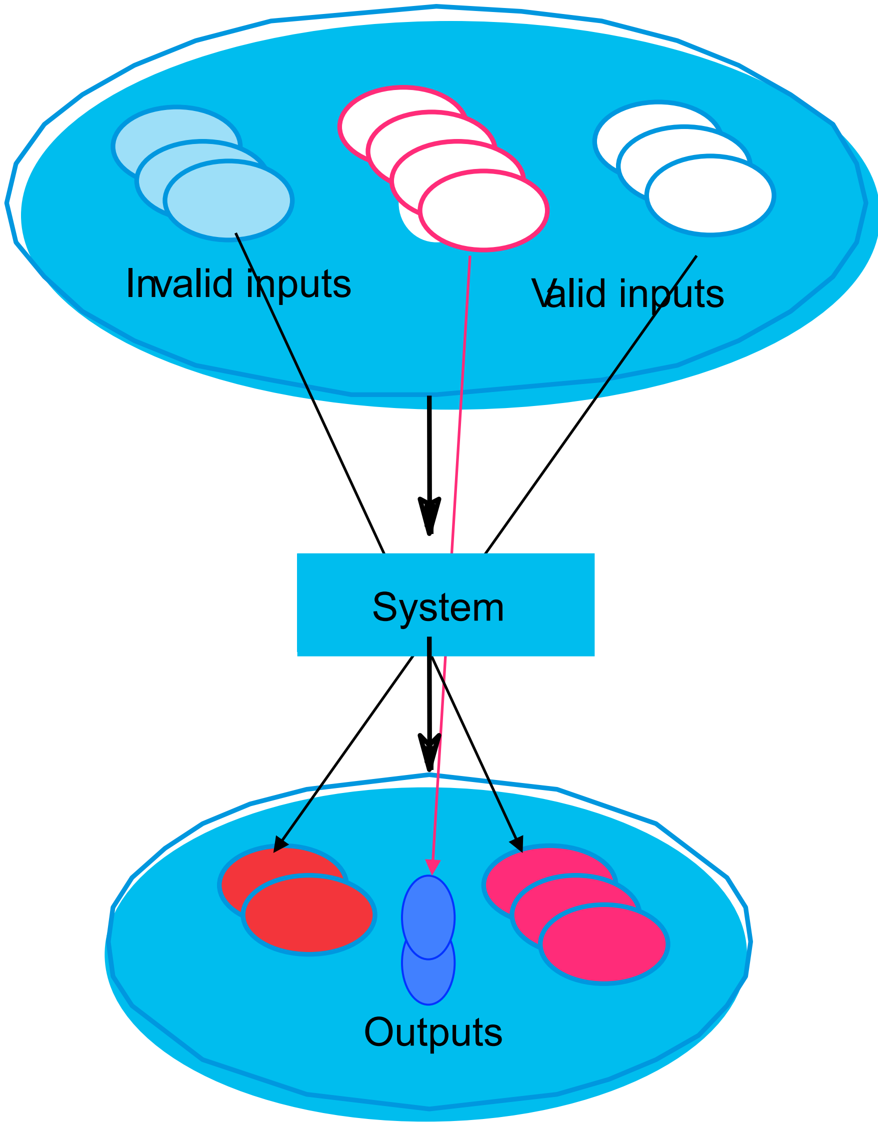
The method accepts 4 to 10 inputs that are 5-digit integers greater than 10,000

Partition Testing

Input data and output results fall into different classes where all members of a class have common characteristics

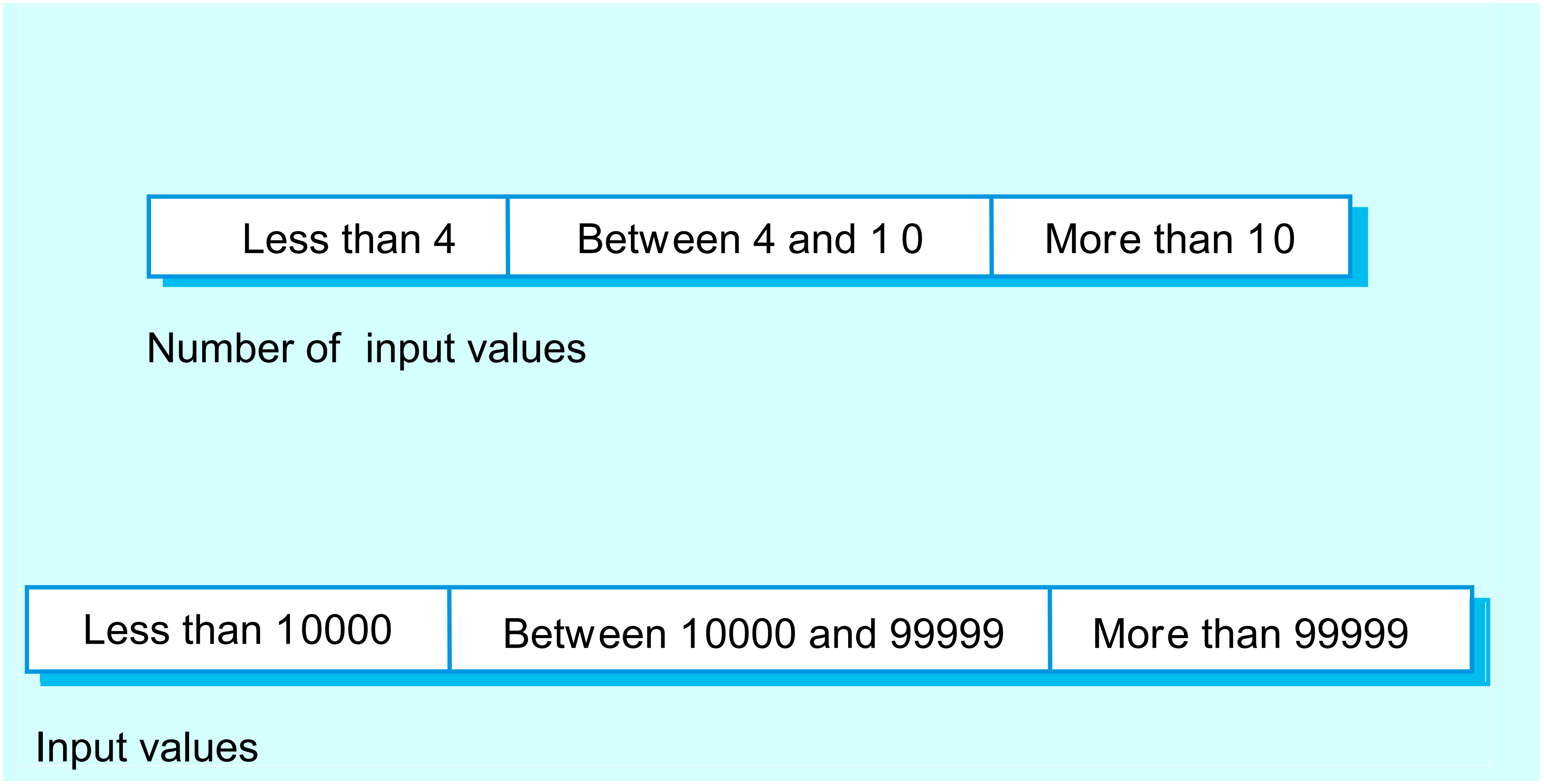
Each of these classes is an (equivalence) partition of the domain where the program behaves in an equivalent way for each class member

Test cases should be chosen from each partition



Equivalence Partitions: An Example

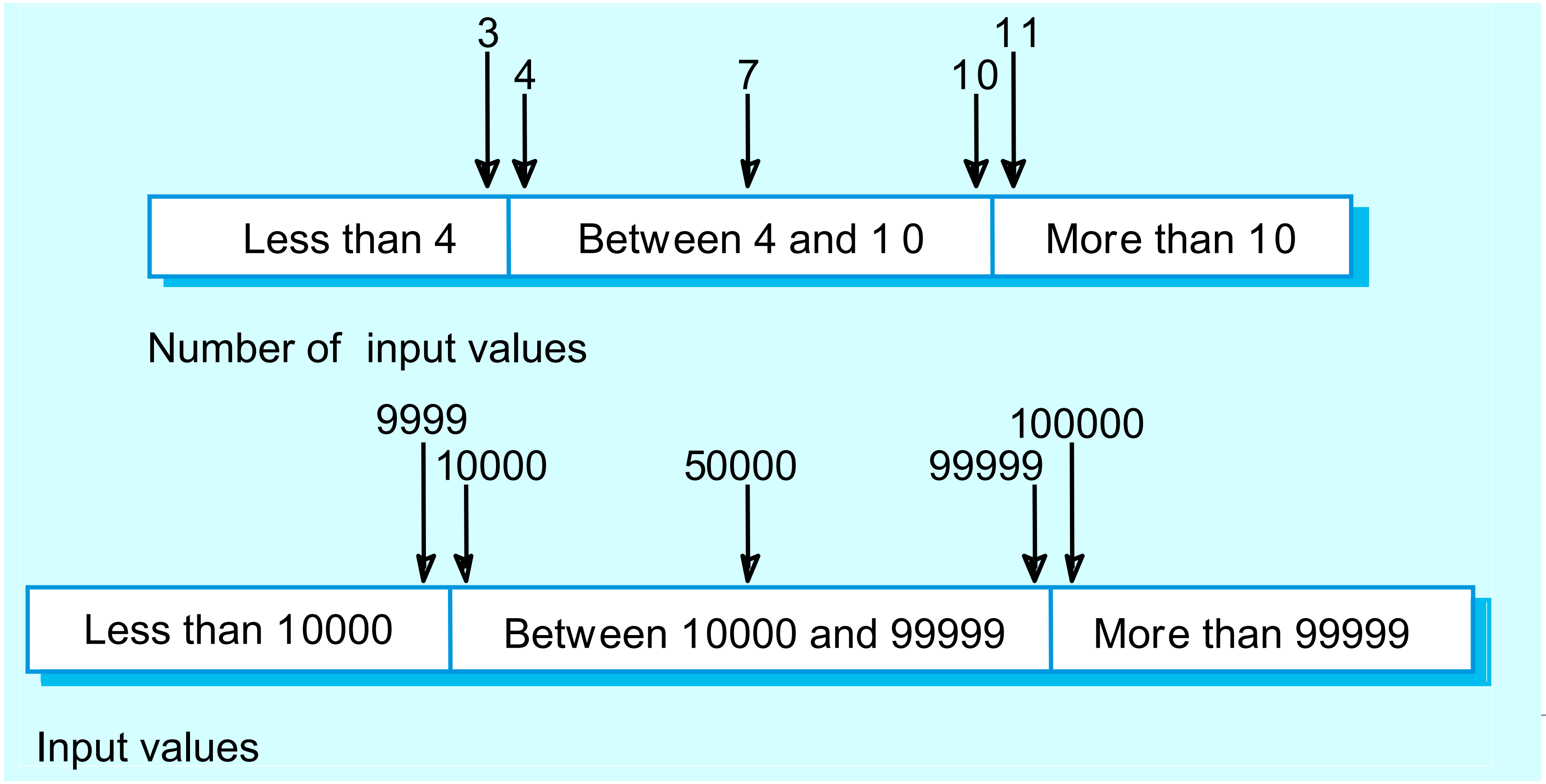
The program accepts 4 to 10 inputs that are 5-digit integers greater than 10,000



Equivalence Partitions: An Example

Choose test cases on the boundaries of the partitions PLUS cases close to the mid-point of the partition

Boundary values are often overlooked!



Example 2 - ATM

- You are required to implement and **test a method** for checking whether an amount of money, expressed as an integer, can be paid as a valid combination of $20’s and $50’s.
- The method should return true if the input amount can be paid using only $20’s and $50’s
- Otherwise false is returned
- An input of 0 shall return false

ATM – partitioning input

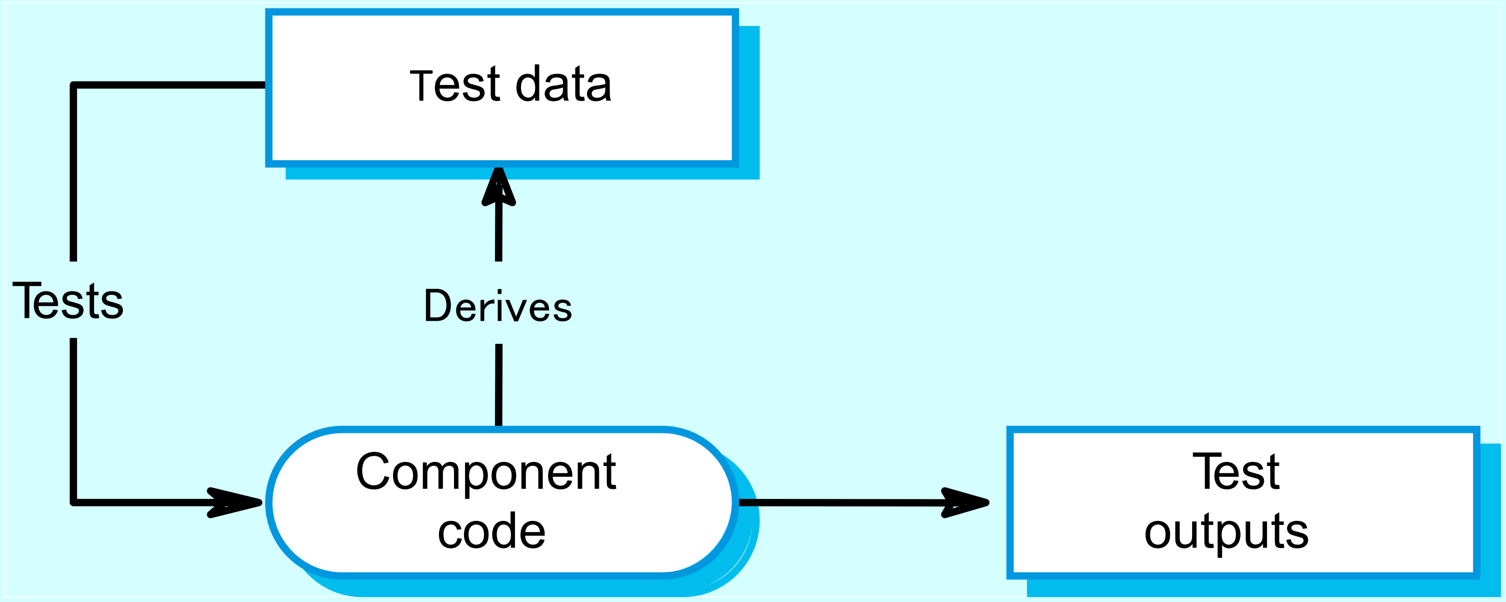
- Consider both valid (returns true) and invalid (returns false) inputs

- Valid amounts
  - Single note ($20 or $50)
  - Multiple notes of same denomination
  - Single one denomination, multiple other
  - Multiple both

- Invalid amounts
  - Negative amount
  - Zero amount
  - Less than $20
  - Between $20 and $50 (except $40)
  - ...

Structural Testing

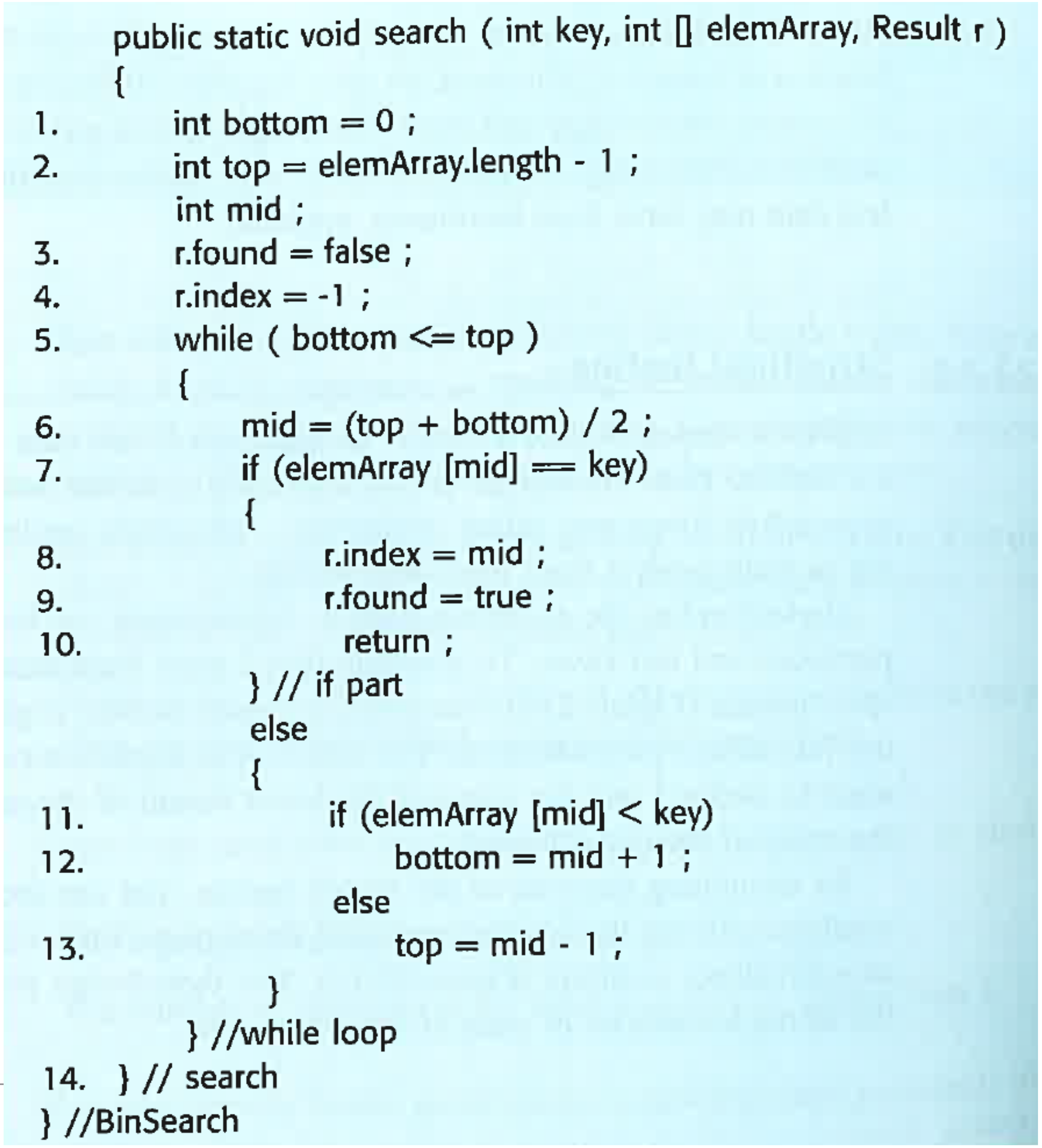
- Sometimes called ***white-box*** (or glass box or clear box) testing
- Derivation of test cases according to *program structure and implementation*. Knowledge of the program is used to identify additional test cases
- Objective is to exercise all program statements



Path Testing

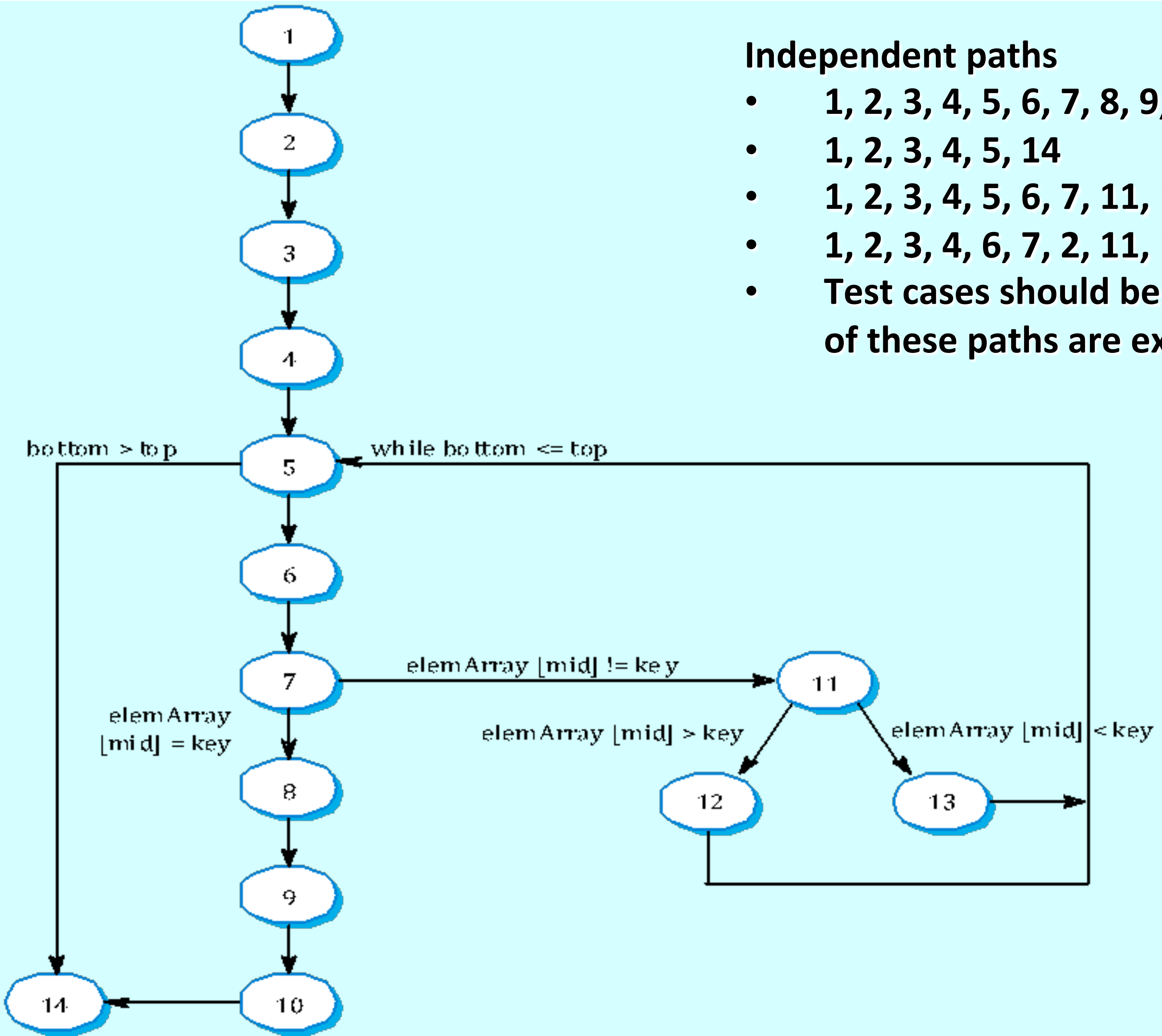
- The objective of path testing is to ensure that the set of test cases is such that each path through the program is executed **at least once**. *It is structural testing!*
- The starting point for path testing is a ***program flow graph*** that shows nodes representing program decisions and arcs representing the flow of control
- Statements with conditions are therefore nodes in the flow graph

Java implementation of a binary search routine



```
        public static void search ( int key, int [] elemArray, Result r )
        {
1.          int bottom = 0 ;
2.          int top = elemArray.length - 1 ;
            int mid ;
3.          r.found = false ;
4.          r.index = -1 ;
5.          while ( bottom <= top )
            {
6.              mid = (top + bottom) / 2 ;
7.              if (elemArray [mid] == key)
                {
8.                  r.index = mid ;
9.                  r.found = true ;
10.                  return ;
                } // if part
                else
                {
11.                 if (elemArray [mid] < key)
12.                     bottom = mid + 1 ;
                    else
13.                     top = mid - 1 ;
                }
            } //while loop
14.     } // search
} //BinSearch
```

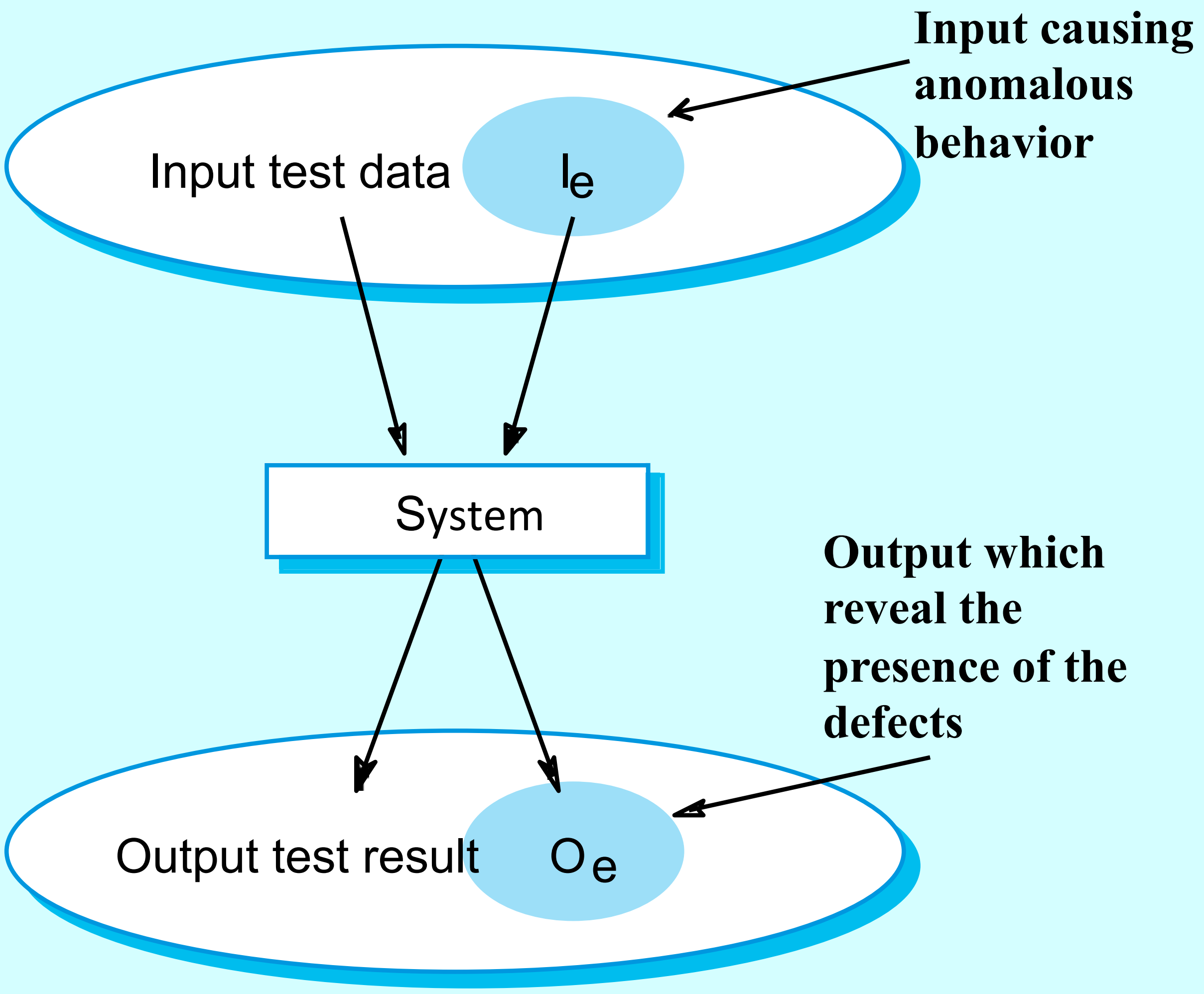
Binary Search Flow Graph

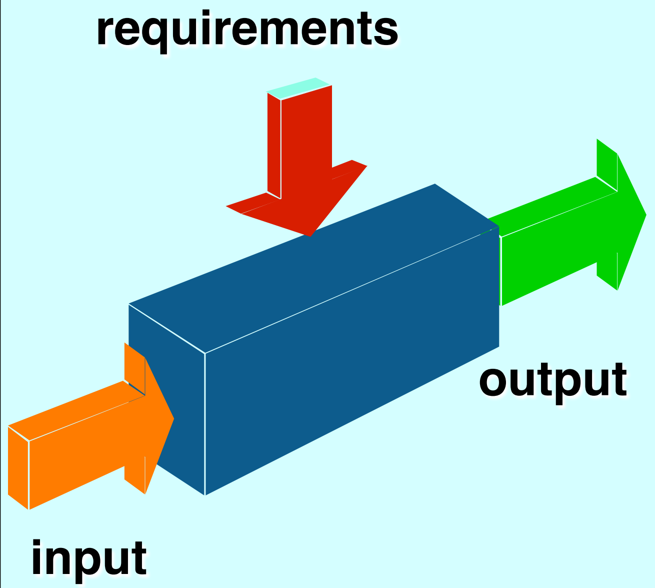


Independent paths

- **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**

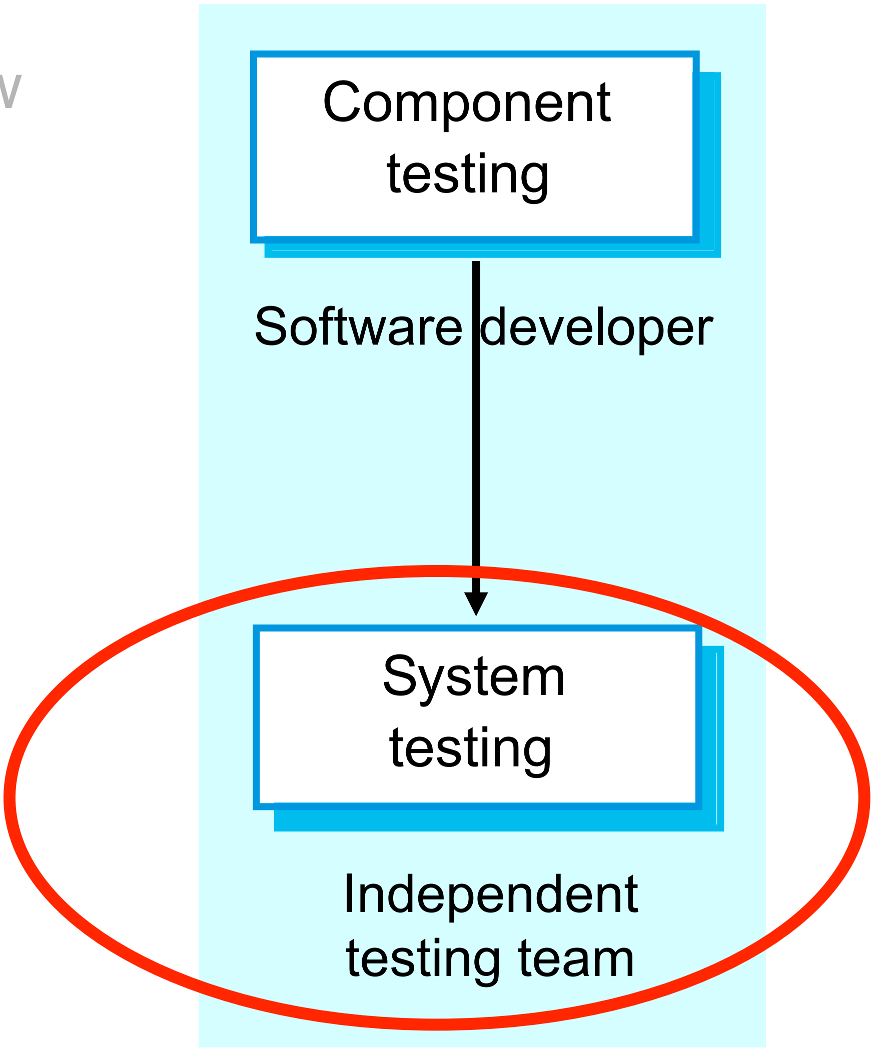
Black-box Testing


Input causing anomalous behavior
Input test data
I_e
System
Output which reveal the presence of the defects
Output test result
O_e



Topics Covered

- Software Testing: An Overview
- Component Testing
- System Testing
- Test-Driven Development



System Testing

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

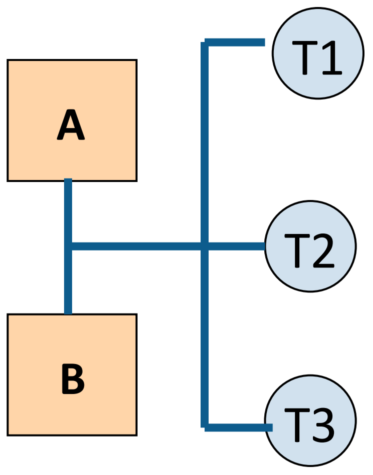
Integration Testing

- 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
- Sandwich integration
  - Combination of top-down and bottom-up

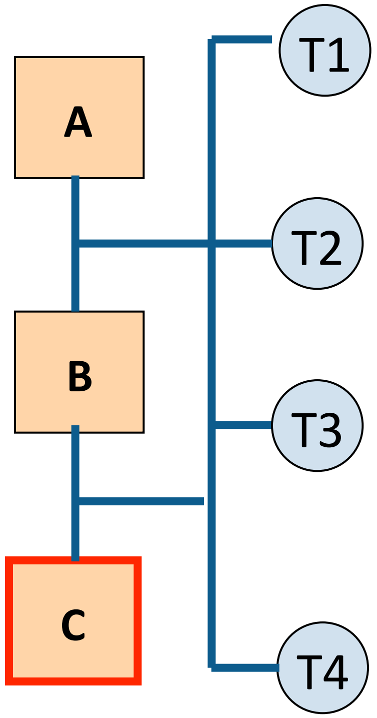
Integration Testing

- One major problem in integration testing is to localise errors
- To simplify error localisation, systems should be **incrementally integrated**

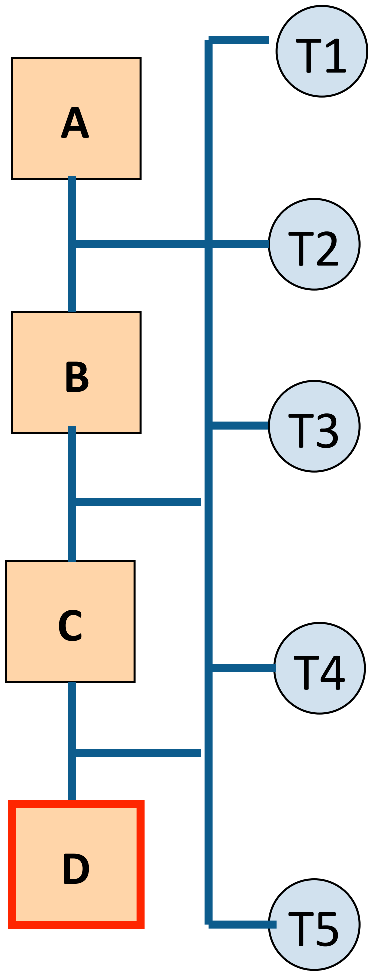
Incremental Integration Testing



Test sequence 1



Test sequence 2



Test sequence 3

Order of Integration

- Customer-driven: consider customer’s priorities
- Usage-frequency: integrate the components that implement the most frequently used functionality first
  - Ensure they will receive the most testing

Regression Testing

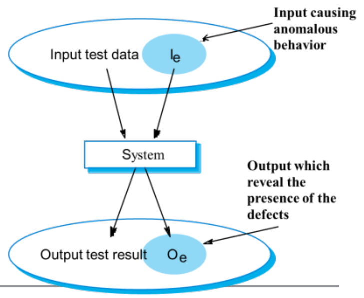
- Introducing new increments **may cause errors** in already tested components
- Regression testing is to test the system and check that new part has not ‘broken’ previously working code, typically by **rerunning the existing set of tests**
- In a manual testing process, regression testing is expensive but, with automated testing, it is simple and straightforward.
- Tests (both old tests and the new ones) must run ‘successfully’ before the change is committed.

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 (good enough to use)
- Release testing is usually black-box testing
  - Based on the system specification only
  - Conducted by a separate team not involved in system development
  - Testers do not have knowledge (even do not concern!) of the system implementation

Guidelines in Release Testing

- The testing team should choose the test cases that will reveal defects in the system
- Some examples that increase the probability for defect reveal:
  - 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


Input causing anomalous behavior
Input test data
Ie
System
Output which reveal the presence of the defects
Output test result
Oe

Preparing Test Cases from Requirements

- A general principle of requirements engineering is that requirements should be testable
- Requirements-based testing is a testing technique where you consider each requirement and derive a set of tests for that requirement
- You are trying to demonstrate that the system has properly implemented its requirements

Requirements-based testing

- Requirements-based testing involves examining each requirement and developing a test or tests for it.

- MHC-PMS (Mental Health Care – Patient Management 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.
  - If a prescriber chooses to ignore an allergy warning, they shall provide a reason why this has been ignored.

Requirements tests

- 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.

Other Types of System Testing

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

Other Types of System Testing

Stress Testing

- Exercises the system beyond its maximum design load
  - Stressing the system often causes defects to come to light
  - Stressing the system tests failure behaviour
    - Systems should not fail catastrophically under the worst situations such as unacceptable loss of service or data
- Stress testing is particularly relevant to distributed systems that can exhibit severe degradation as a network becomes overloaded

User Testing

- User or customer testing is a stage in the testing process 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.
  - Reason: influences from the user's working environment have a major effect on the reliability, performance, usability and robustness of a system. These cannot be replicated in a testing environment.

Types of User Testing

- Alpha testing
  - Users of the software work with the development team to test the software at the developer's site.
- Beta testing
  - A release of the software is made available to users to allow them to experiment and to raise problems that they discover with the system developers.
- Acceptance testing
  - Customers test a system to decide whether or not it is ready to be accepted from the system developers and deployed in the customer environment.

Test Automation

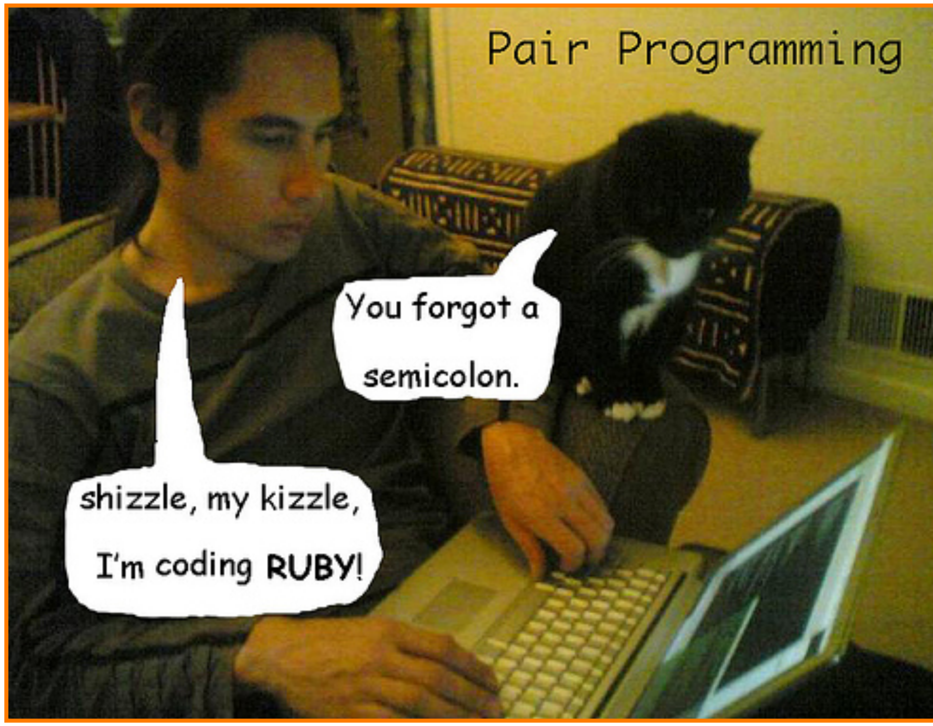
- Testing is an expensive and laborious process. Testing workbenches provide a range of tools to reduce the time required and total testing costs.
- Systems such as JUnit support the automatic execution of tests.
- Junit integrated nicely with IDEs (e.g., JBuilder, Eclipse, BlueJ) and Ant

Topics Covered

- Software Testing: An Overview
- Component Testing
- System Testing
- **Test-Driven Development**

Agile Approaches to Quality

- Pair programming (Develop software artefacts as a team)
  - Driver writes the software
  - Other person eyeballs the software looking for defects
  - Both contribute to the thought processes and problem solving
- Studies show a marked increase in quality
  - Especially in terms of defect rate
  - Strengthening the case for pair-programming (IEEE Software July/ August 2000)
- Cost is no where near double that of single programmer



Agile Approaches to Quality

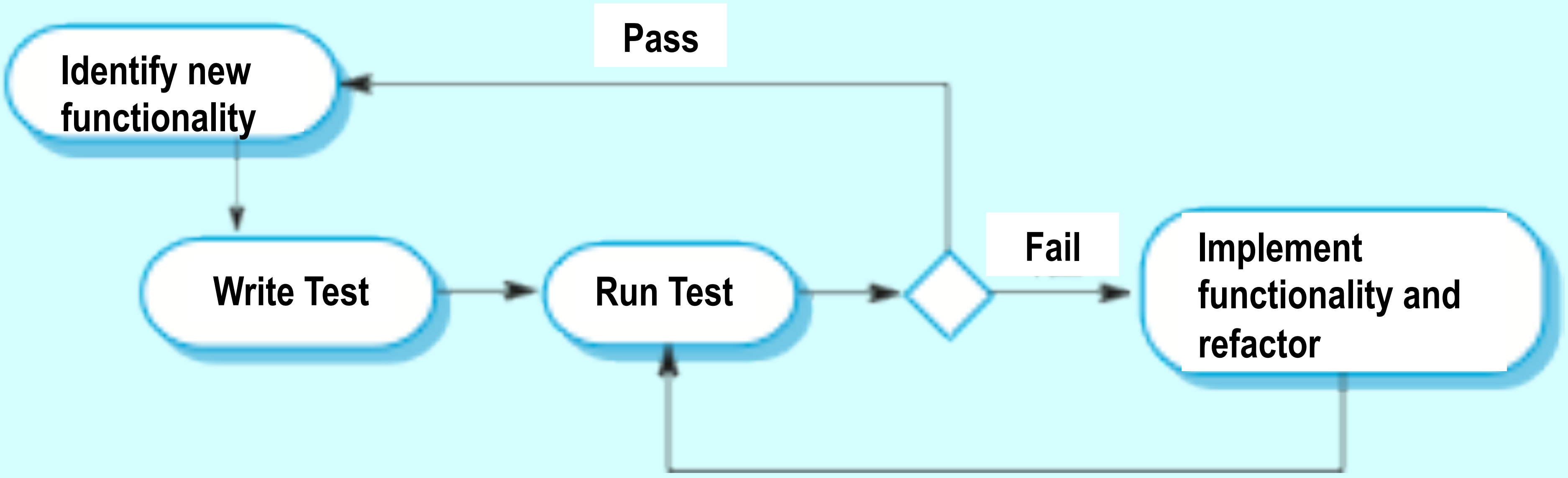
- Test-driven development
  - Test-first development
  - Incremental test development from scenarios (or use cases)
  - User involvement in the test development and validation
  - The use of automated test harnesses

The Three Laws of TDD (Featuring Kotlin) by Robert Martin - https://youtu.be/qkblc5WRn-U

Test-Driven Development

- Test-driven development (TDD) is an approach to program development in which you inter-leave testing and code development.
- Tests are written before code and ‘passing’ the tests is the critical driver of development.
- You develop code incrementally, along with a test for that increment. You don’t move on to the next increment until the code that you have developed passes its test.
- TDD was introduced as part of agile methods such as Extreme Programming. However, it can also be used in plan-driven development processes.

Test-Driven Development


Identify new functionality
Pass
Write Test
Run Test
Fail
Implement functionality and refactor

Otherwise known as "Red, Green, Refactor"

TDD Process Activities

- Start by identifying the increment of functionality that is required. This should normally be small and implementable in a few lines of code.
- Write a test for this functionality and implement this as an automated test.
- Run the test, along with all other tests that have been implemented. Initially, you have not implemented the functionality so the new test will fail.
- Implement the functionality and re-run the test. Once all tests run successfully, you move on to implementing the next chunk of functionality.

Benefits of TDD

- Code coverage
  - Every code segment that you write has at least one associated test so all code written has at least one test.
- Regression testing
  - A regression test suite is developed incrementally as a program is developed.
- Simplified debugging
  - When a test fails, it should be obvious where the problem lies. The newly written code needs to be checked and modified.
- System documentation
  - The tests themselves are a form of documentation that describe what the code should be doing.

Key Points Revisit

- 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 customers.
- Use experience and guidelines to design test cases in defect testing.

Key Points Revisit (cont.)

- 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.
- **JUnit** is a Java framework for performing unit tests on code
- Test automation reduces testing costs by supporting the test process with a range of software tools.
- Test-driven development is an agile approach to software development