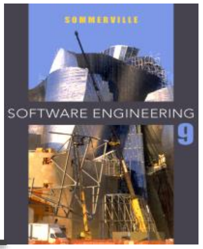


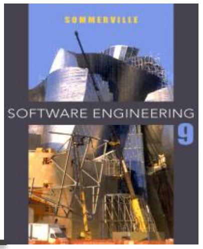
Software Enginnering-2

Software Engineering Program

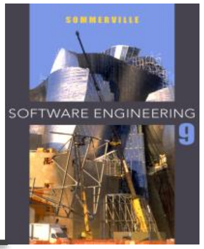
Course Content in Software Enginnering-1

- **Chapter 1** - Introduction
- **Chapter 2** - Software Processes
- **Chapter 3** - Agile Software Development
- **Chapter 4** - Requirements Engineering
- **Chapter 5** - System Modeling
- **Chapter 6** - Architectural Design
- **Chapter 7** - Design and Implementation

Course Content in Software Enginnering-2



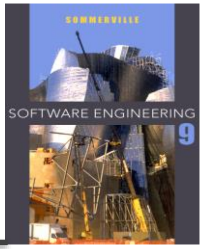
- ✧ **Chapter 8** - Software Testing
- ✧ **Chapter 9** - Software Evolution
- ✧ **Chapter 10** - Sociotechnical Systems
- ✧ **Chapter 11** - Dependability and Security
- ✧ **Chapter 12** - Dependability and Security Specification
- ✧ **Chapter 13** - Dependability Engineering
- ✧ **Chapter 14** - Security Engineering
- ✧ **Chapter 15** - Dependability and Security Assurance



Chapter 8 – Software Testing

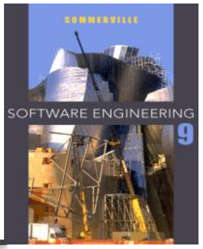
Lecture 1

Topics covered



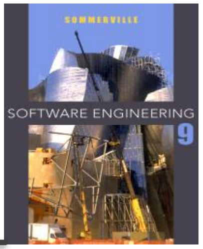
- ✧ Development testing
- ✧ Test-driven development
- ✧ Release testing
- ✧ User testing

Program testing



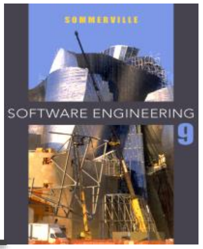
- Testing is intended to show that a program does what it is intended to do and to discover program defects before it is put into use.
- When you test software, you execute a program using artificial data.
- You check the results of the test run for errors, anomalies or information about the program's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

Program testing goals



- ✧ To demonstrate to the developer and the customer that the software meets its requirements.
  - For custom software, this means that there should be at least one test for every requirement in the requirements document. For generic software products, it means that there should be tests for all of the system features, plus combinations of these features, that will be incorporated in the product release.
- ✧ To discover situations in which the behavior of the software is incorrect, undesirable or does not conform to its specification.
  - Defect testing is concerned with rooting out undesirable system behavior such as system crashes, unwanted interactions with other systems, incorrect computations and data corruption.

Validation and defect testing

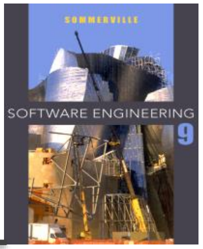


- ✧ The first goal leads to validation testing
  - You expect the system to perform correctly using a given set of test cases that reflect the system's expected use.
- ✧ The second goal leads to defect testing
  - The test cases are designed to expose defects. The test cases in defect testing can be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

عمدا

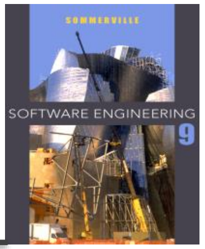
غامض

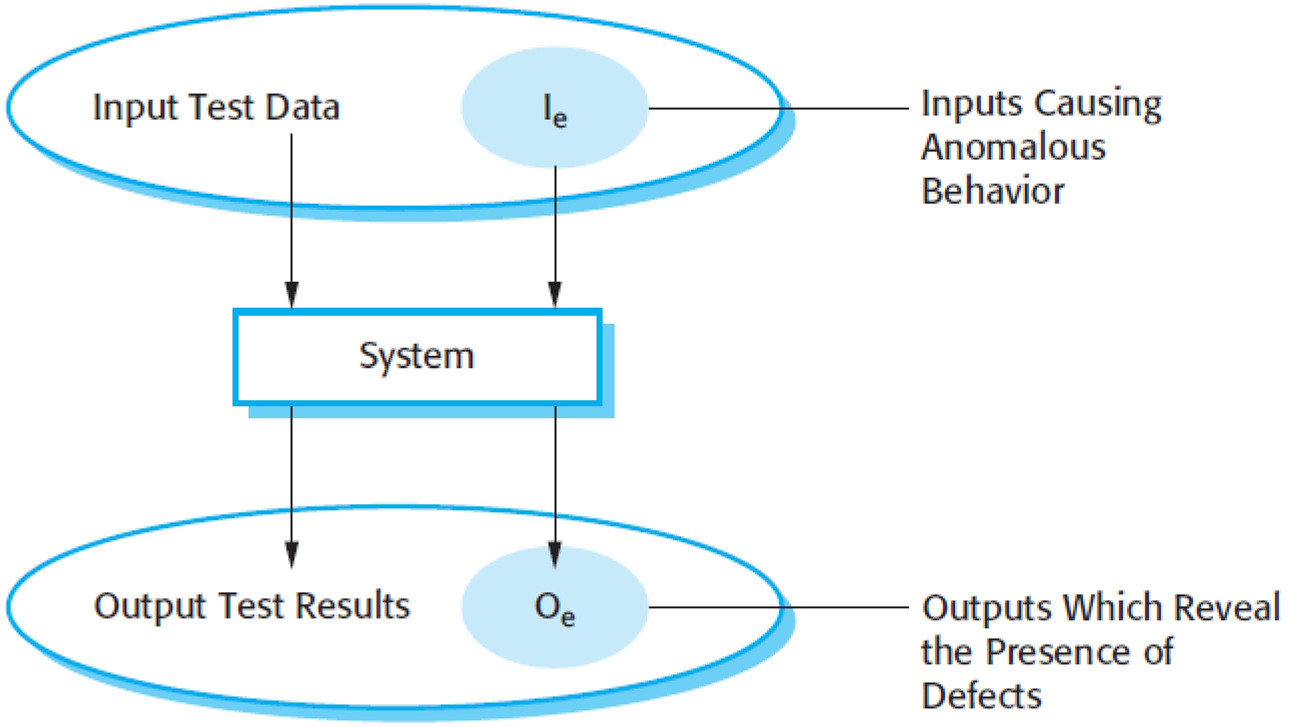
Testing process goals



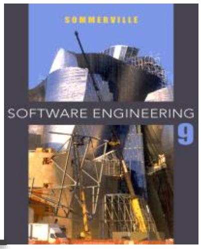
- ✧ Validation testing
  - To demonstrate to the developer and the system customer that the software meets its requirements
  - A successful test shows that the system operates as intended.
- ✧ Defect testing
  - To discover faults or defects in the software where its behaviour is incorrect or not in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

An input-output model of program testing




Input Test Data
I_e
Inputs Causing Anomalous Behavior
System
Output Test Results
O_e
Outputs Which Reveal the Presence of Defects

Verification vs validation



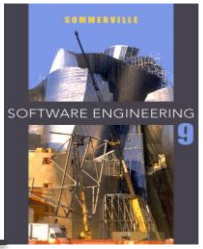
- Verification:
  "Are we building the product right”.

  The software should conform to its specification.

- Validation:
  "Are we building the right product”.

  The software should do what the user really requires.

V & V confidence

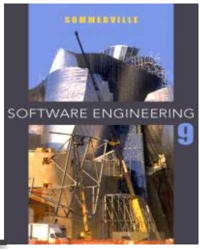


✧ Aim of V & V is to establish confidence that the system is 'fit for purpose'.

✧ Depends on system's purpose, user expectations and marketing environment

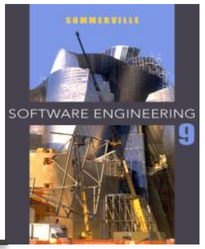
- Software purpose
  - The level of confidence depends on how critical the software is to an organisation.
- User expectations
  - Users may have low expectations of certain kinds of software.
- Marketing environment
  - Getting a product to market early may be more important than finding defects in the program.

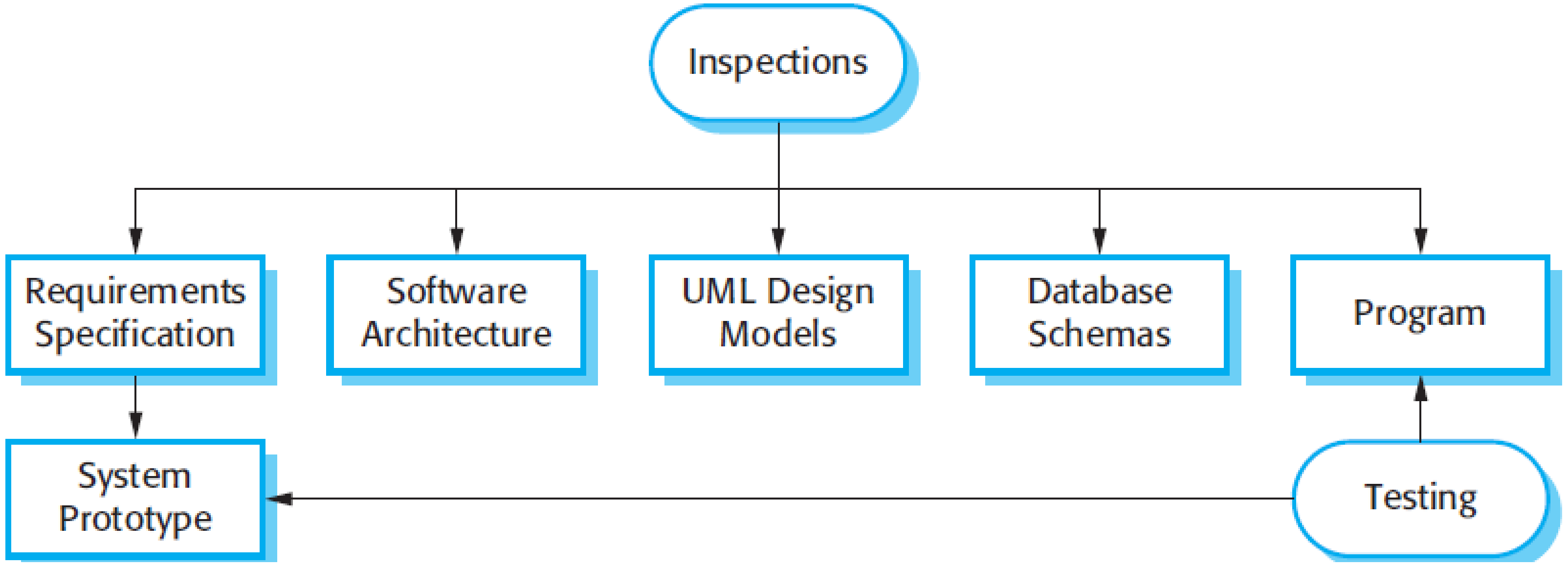
Inspections and testing



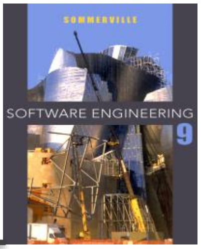
- Software inspections Concerned with analysis of the static system representation to discover problems (static verification)
  - May be supplement by tool-based document and code analysis.
- Software testing Concerned with exercising and observing product behaviour (dynamic verification)
  - The system is executed with test data and its operational behaviour is observed.

Inspections and testing



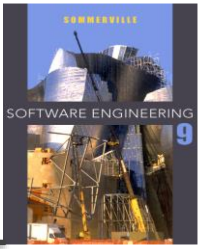


Software inspections



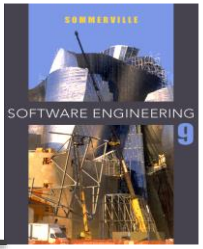
- ✧ These involve people examining the source representation with the aim of discovering anomalies and defects.
- ✧ Inspections not require execution of a system so may be used before implementation.
- ✧ They may be applied to any representation of the system (requirements, design,configuration data,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

Advantages of inspections



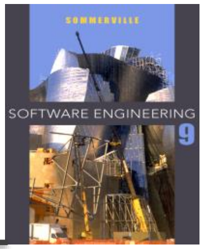
- ✧ During testing, errors can mask (hide) other errors. Because inspection is a static process, you don’t have to be concerned with interactions between errors.
- ✧ Incomplete versions of a system can be inspected without additional costs. If a program is incomplete, then you need to develop specialized 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.

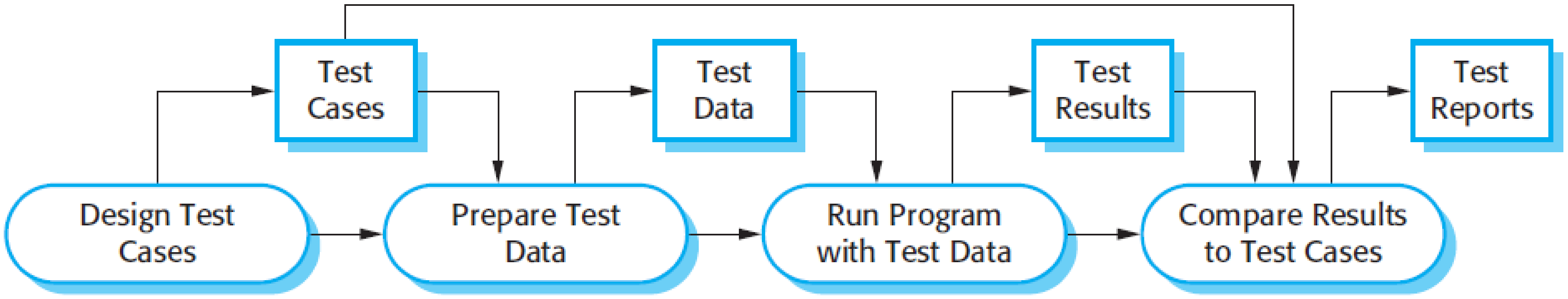
Inspections and testing



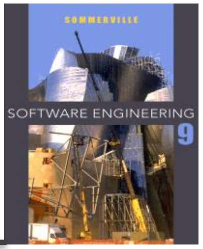
- ✧ Inspections and testing are complementary and not opposing verification techniques.
- ✧ Both should be used during the V & V process.
- ✧ Inspections can check conformance with a specification but not conformance with the customer's real requirements.
- ✧ Inspections cannot check non-functional characteristics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

A model of the software testing process



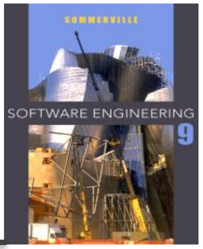


Stages of testing



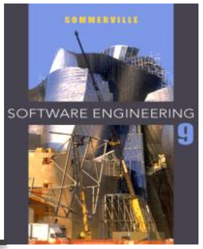
- ✧ Development testing, where the system is tested during development to discover bugs and defects.
- ✧ Release testing, where a separate testing team test a complete version of the system before it is released to users.
- ✧ User testing, where users or potential users of a system test the system in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

Development testing



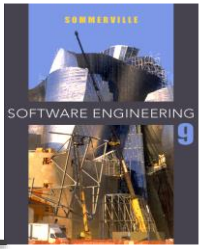
- Development testing includes all testing activities that are carried out by the team developing the system.
  - Unit testing, where individual program units or object classes are tested. Unit testing should focus on testing the functionality of objects or methods.
  - Component testing, where several 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

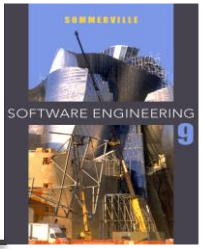
Object class testing

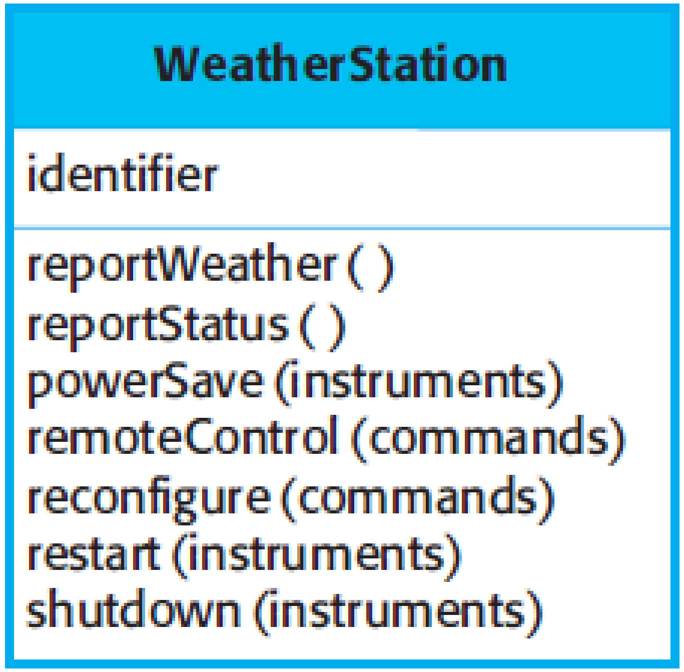


✧ Complete test coverage of a class involves
  - Testing all operations associated with an object
  - Setting and interrogating all object attributes
  - Exercising the object in all possible states.

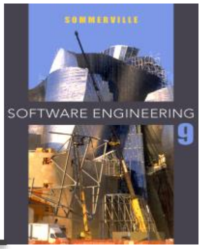
✧ Inheritance makes it more difficult to design object class tests as the information to be tested is not localised.

The weather station object interface





Weather station testing



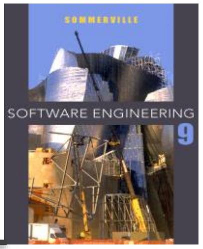
✧ Need to define test cases for reportWeather, calibrate, test, startup and shutdown.

✧ Using a state model, identify sequences of state transitions to be tested and the event sequences to cause these transitions

✧ For example:

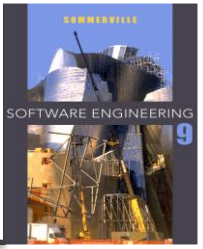
- Shutdown -> Running-> Shutdown
- Configuring-> Running-> Testing -> Transmitting -> Running
- Running-> Collecting-> Running-> Summarizing -> Transmitting -> Running

Automated testing



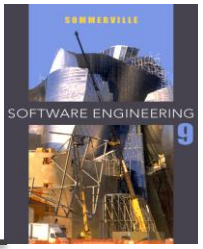
- ✧ Whenever possible, unit testing should be automated so that tests are run and checked without manual intervention.
- ✧ In automated unit testing, you make use of a test automation framework (such as JUnit) to write and run your program tests.
- ✧ Unit testing frameworks provide generic test classes 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.

Unit test effectiveness

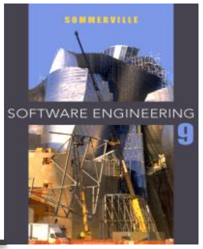


- ✧ The test cases should show that, when used as expected, the component that you are testing does what it is supposed to do.
- ✧ If there are defects in the component, these should be revealed by test cases.
- ✧ This leads to 2 types of unit test case:
  - The first of these should reflect normal operation of a program and should show that the component works as expected.
  - The other kind of test case should be based on testing experience of where common problems arise. It should use abnormal inputs to check that these are properly processed and do not crash the component.

Testing strategies

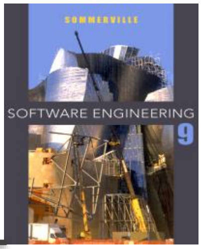
- ✧ Partition testing, where you identify groups of inputs that have common characteristics and should be processed in the same way.
  - You should choose tests from within each of these groups.
- ✧ Guideline-based testing, where you use testing guidelines to choose test cases.
  - These guidelines reflect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

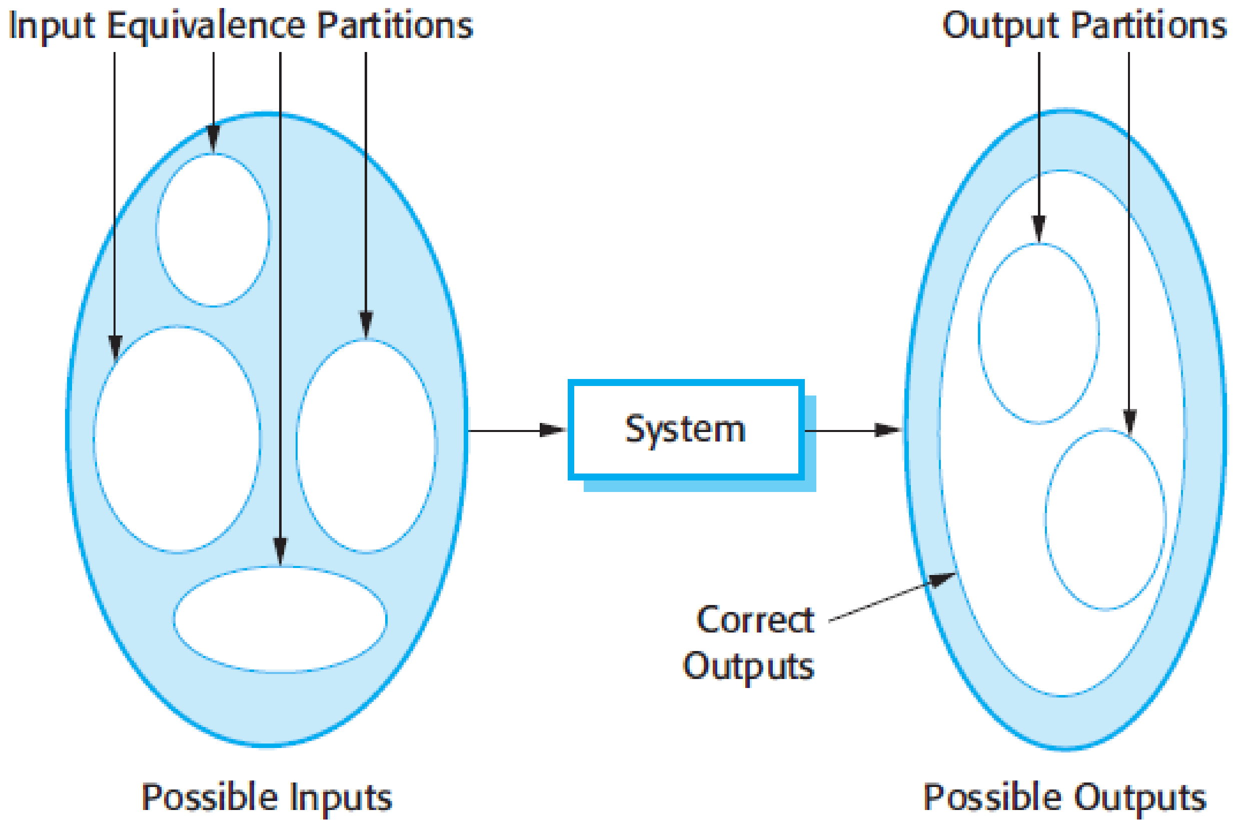
Partition testing



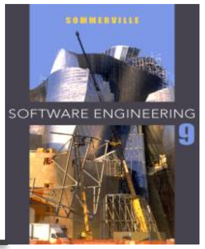
- ✧ Input data and output results often fall into different classes where all members of a class are related.
- ✧ Each of these classes is an equivalence partition or domain where the program behaves in an equivalent way for each class member.
- ✧ Test cases should be chosen from each partition.

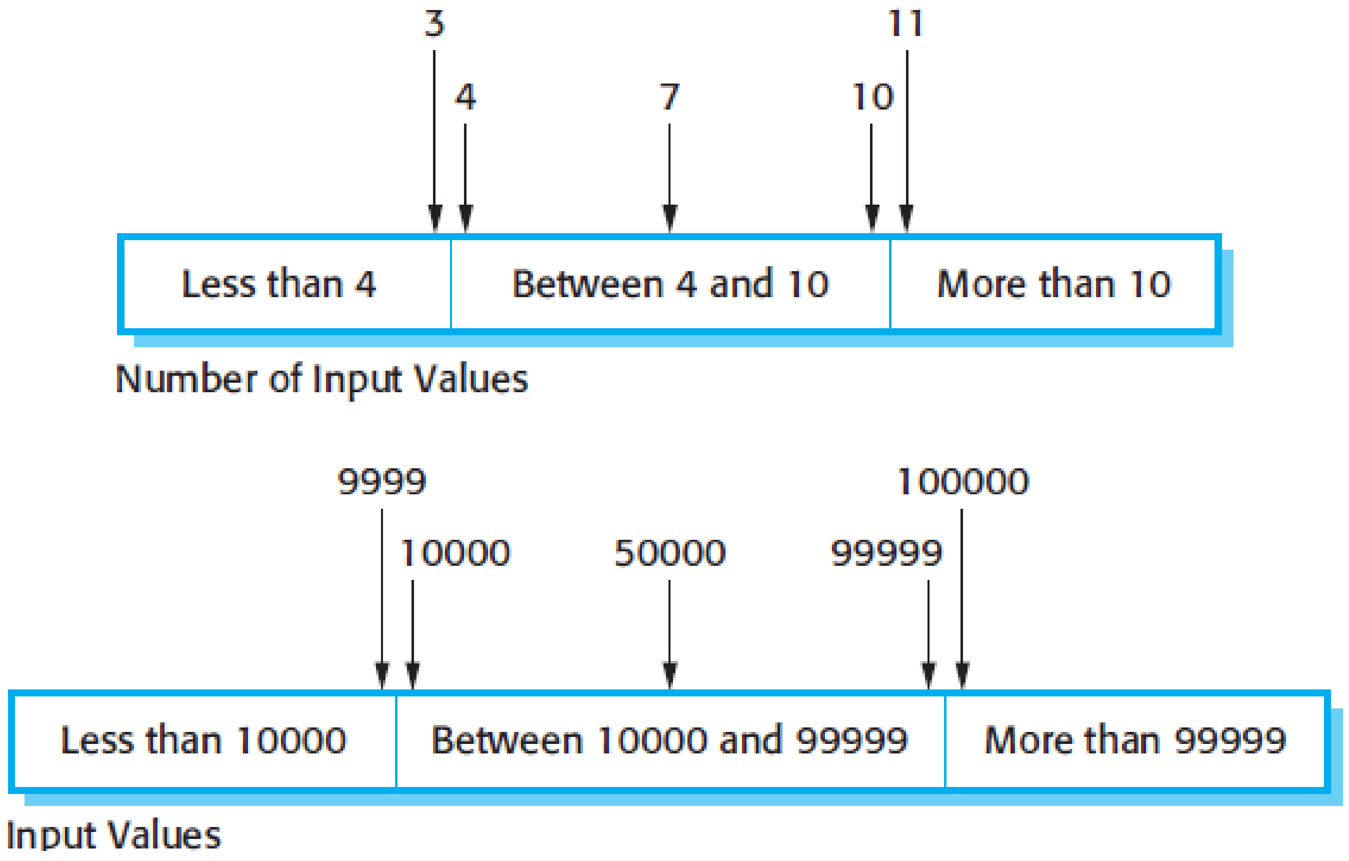
Equivalence partitioning



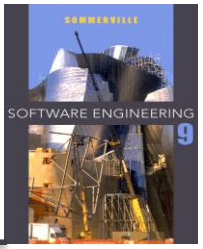


Equivalence partitions



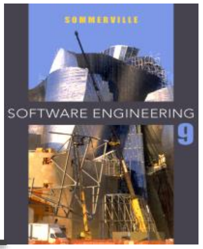


Testing guidelines (sequences)



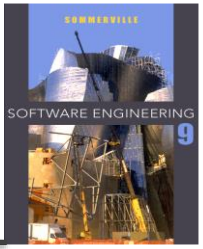
- ✧ Test software with sequences which have only a single value.
- ✧ Use sequences of different sizes in different tests.
- ✧ Derive tests so that the first, middle and last elements of the sequence are accessed.
- ✧ Test with sequences of zero length.

General testing guidelines



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

Key points



- ✧ 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.