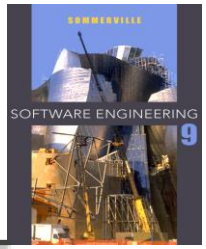
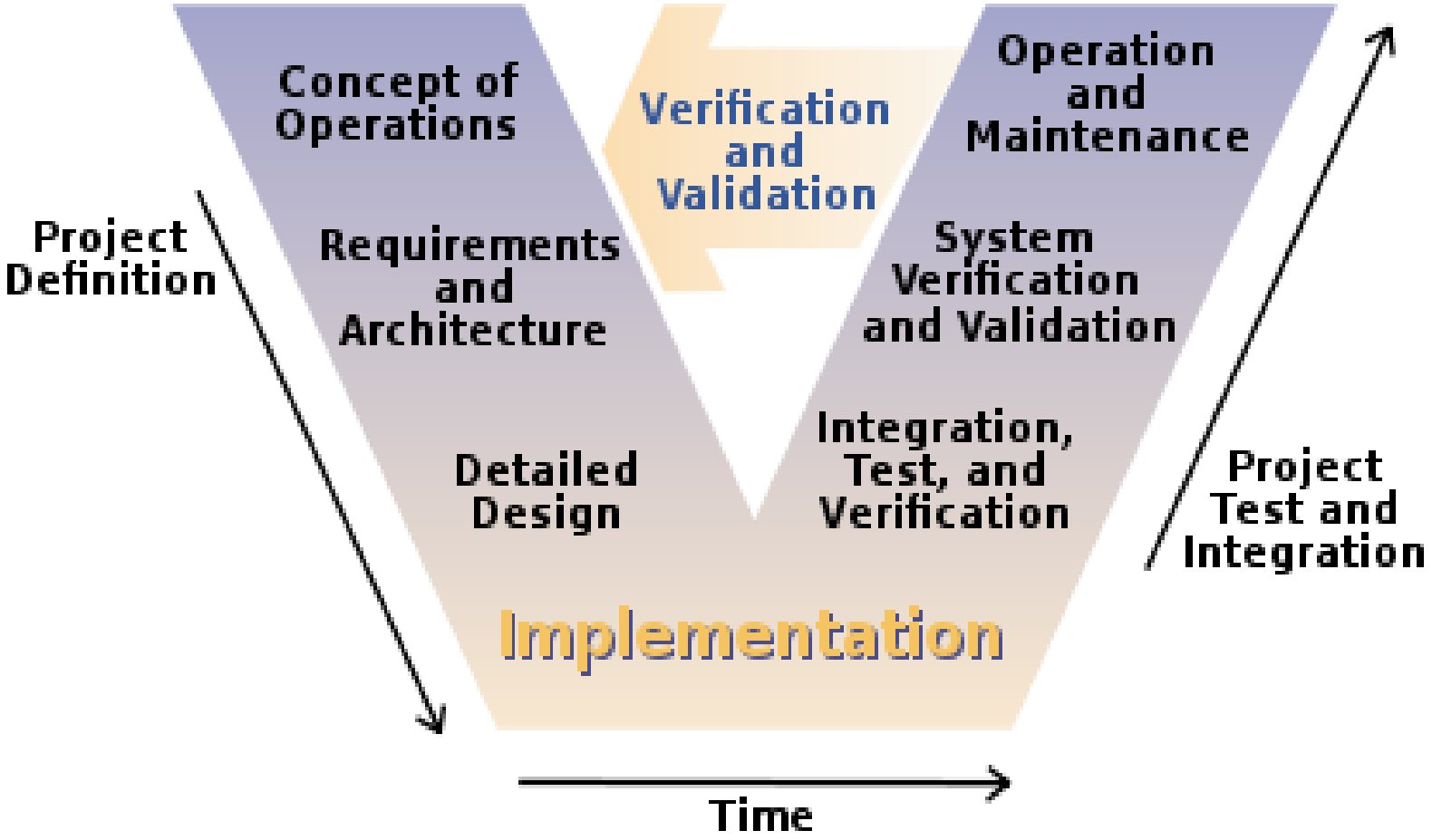
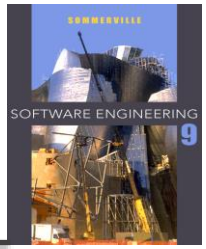


Software Testing

V model


Concept of Operations
Requirements and Architecture
Detailed Design
Implementation
Integration, Test, and Verification
System Verification and Validation
Operation and Maintenance
Verification and Validation
Project Definition
Project Test and Integration
Time

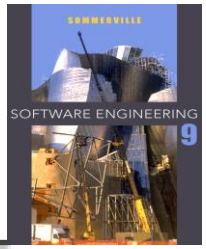
Program testing goals



- To demonstrate to the developer and the customer that the software **meets its requirements**.
  => leads to validation testing

- To discover situations in which the behavior of the software is incorrect, undesirable or does **not conform to its specification**.
  => leads to defect testing

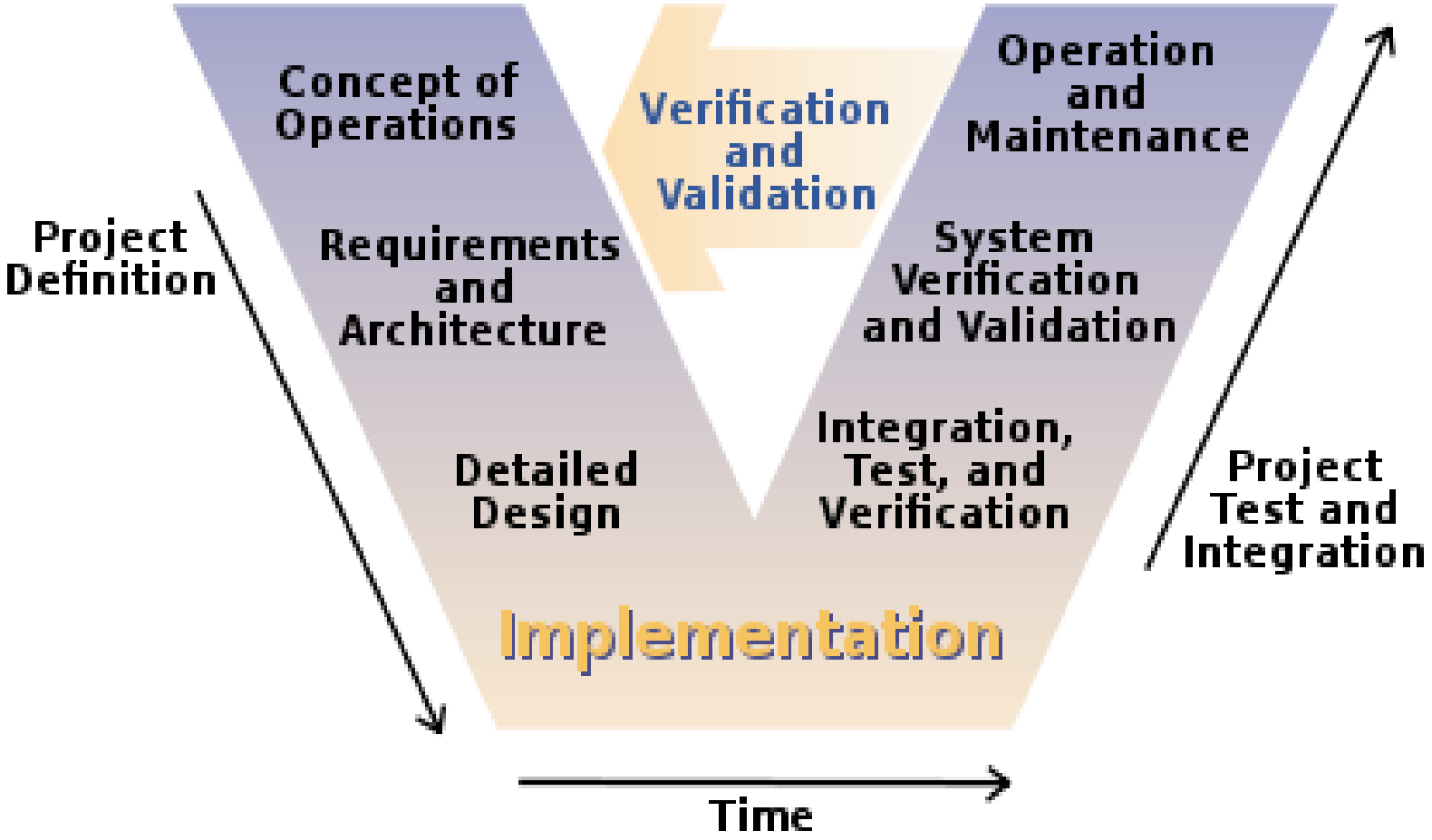
Verification vs validation



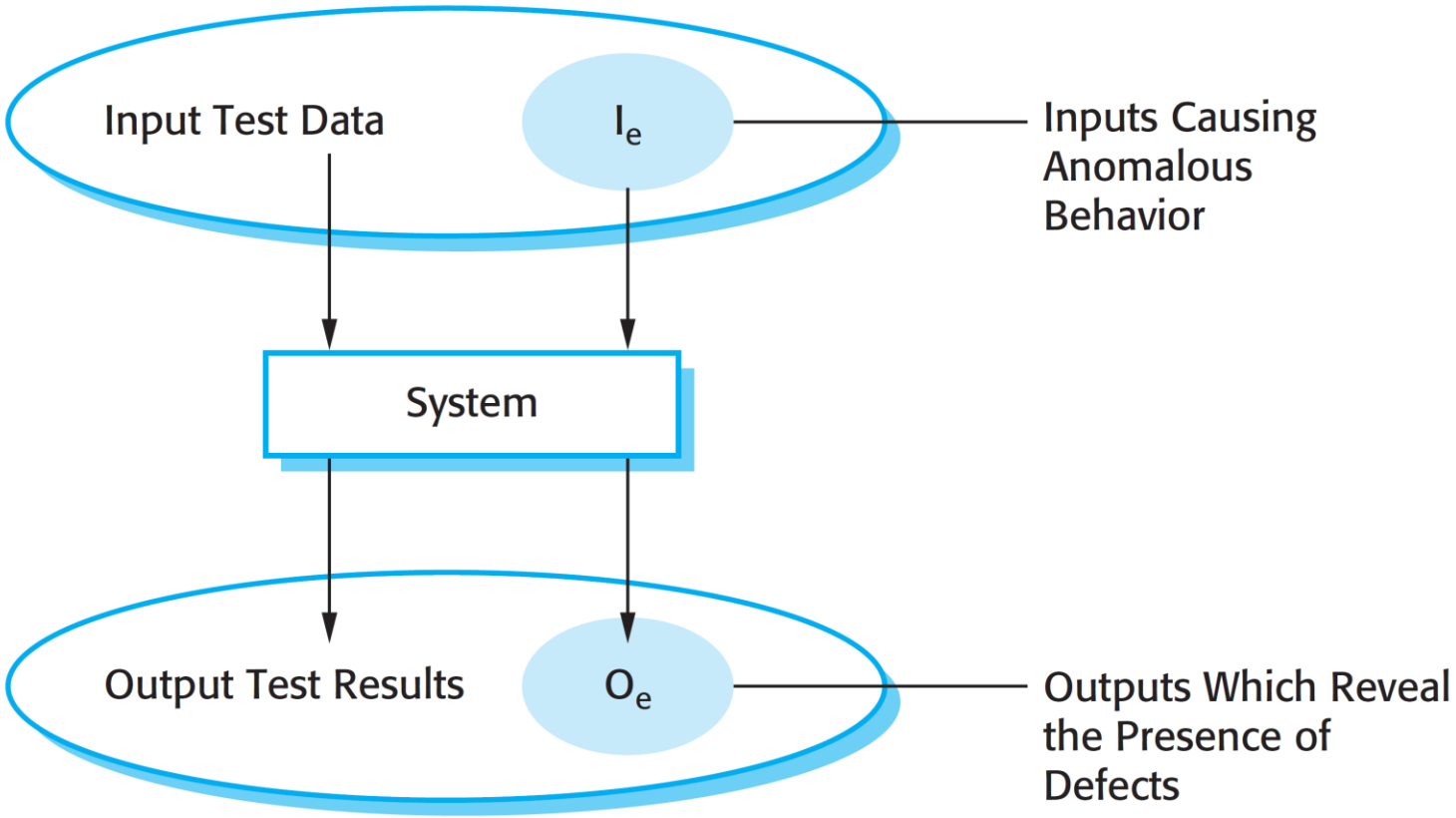
- **Verification**: (testing) – defect testing
  "Are we building the product right".
  - The software should conform to its specification.

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

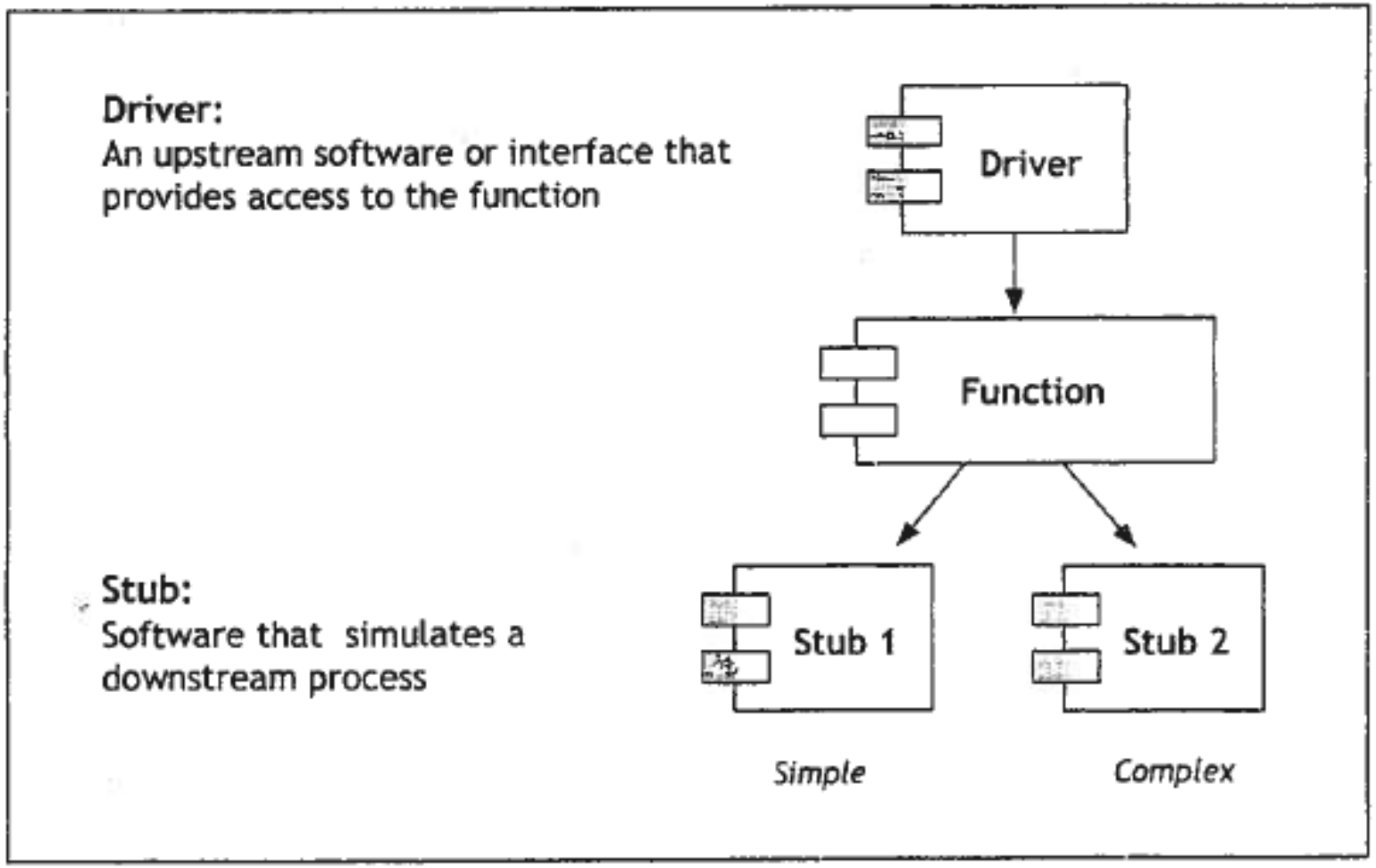
V model


Concept of Operations
Requirements and Architecture
Detailed Design
Implementation
Integration, Test, and Verification
System Verification and Validation
Operation and Maintenance
Verification and Validation
Project Definition
Project Test and Integration
Time

Testing - principles

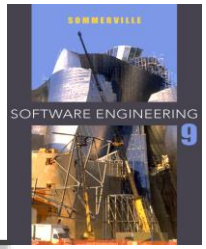


Set up Test


Driver:
An upstream software or interface that provides access to the function
Driver
Function
Stub:
Software that simulates a downstream process
Stub 1
Stub 2
Simple
Complex

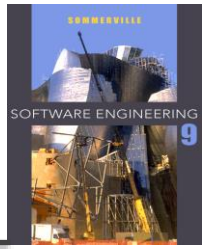
Different levels of testing

related to the V-model



- Validation of the concepts and requirements
  e.g. Are the domain model right? The use stories? (the **users /PO**)
- Validation of the design
  e.g. design class diagrams and design sequence diagrams
  (Reviews or Technical inspection by the **project team**)
- Component Verification
  e.g. **unit test** and test cases (**implementer / programmer**)
- System and integration verification
  e.g. system/integration test (**project team**)
- Operation Verification
  e.g. acceptance test

Black Box & White Box test



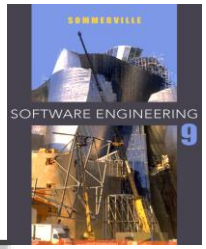
✧ Black box

- Look at methods (system part) as a closed box
- Know only interface

✧ White box

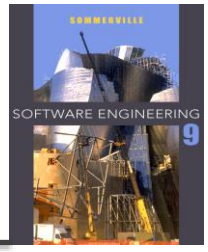
- Look inside the methods (system part)
- Look at all possible path through the methods

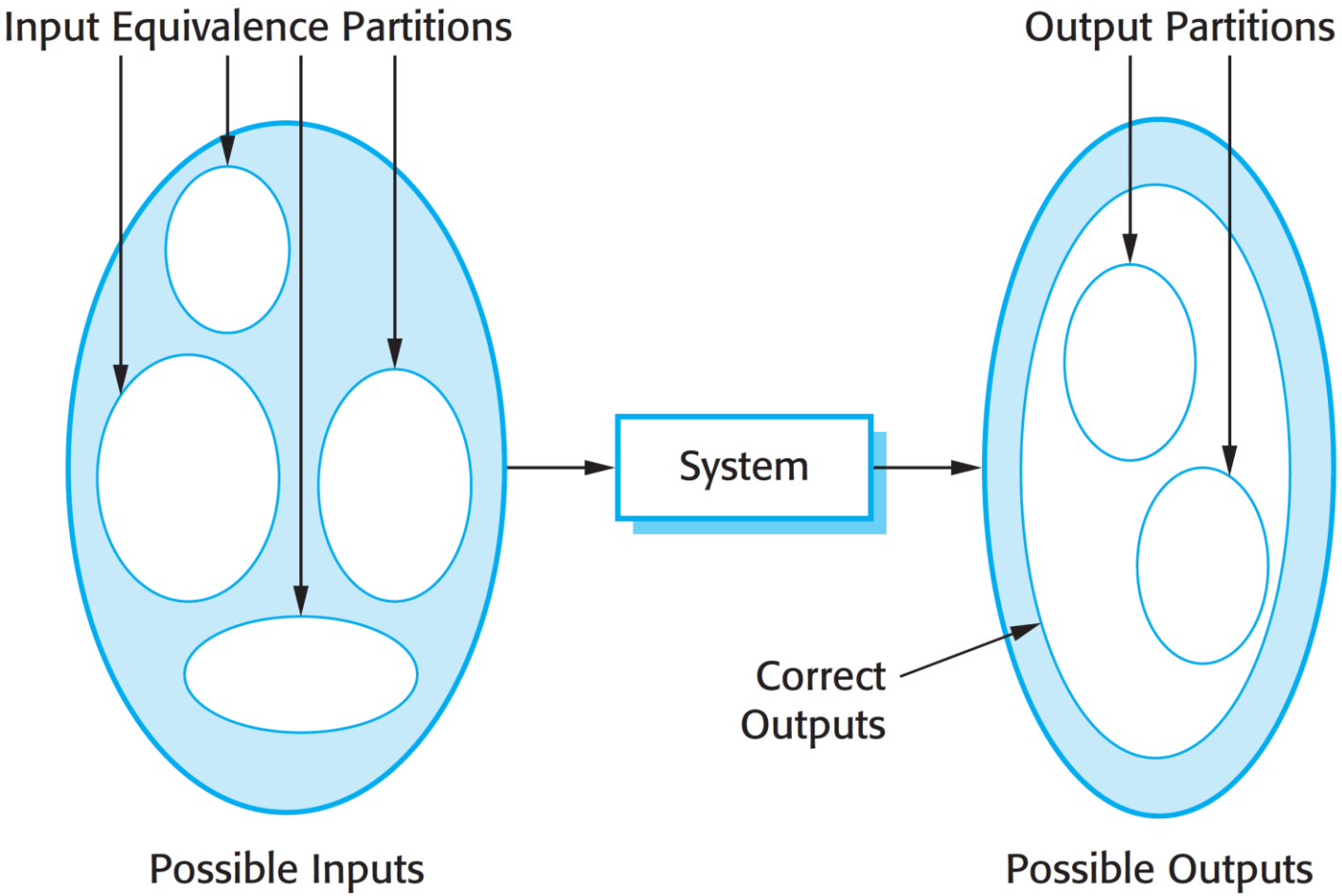
Black box testing



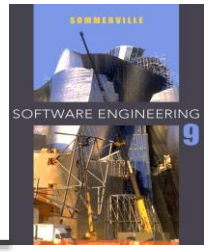
- ✧ The system code is ‘unknown’ -> a black box
- ✧ Look only at the methods signatures
- ✧ **Testing all kind of possible input and output**
- ✧ In C# create a Unit Test

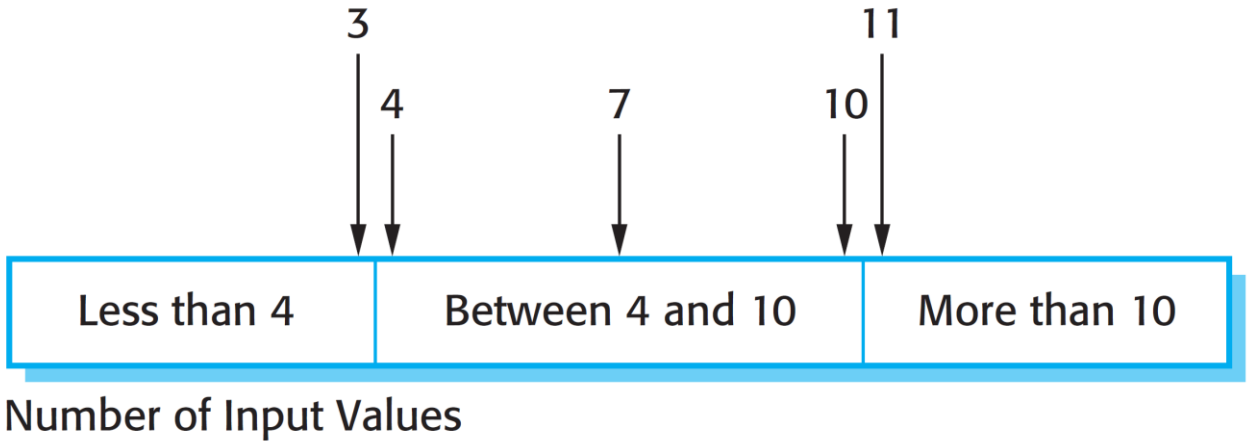
Equivalence partitioning

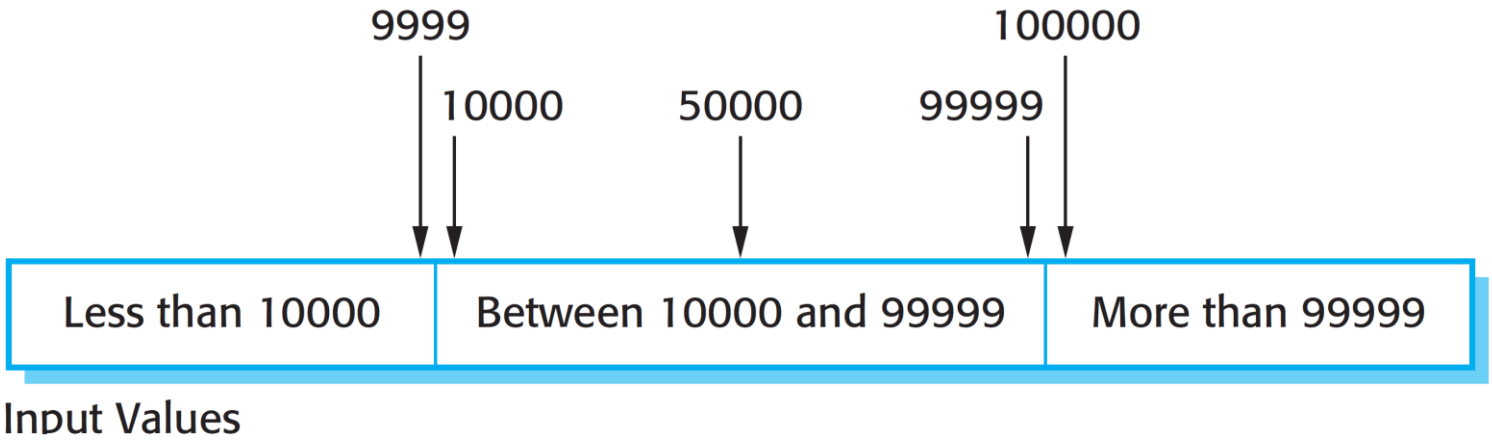




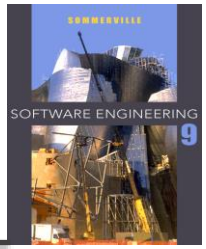
Equivalence partitions







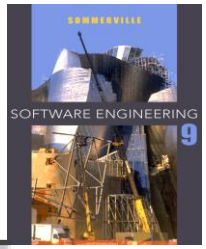
Unit test in c#



✧ Console Programs / (and later) Razor Pages

- Create a test unit project (MSTest Test Project),
- Add reference to the project,
- Remember to have the class - to be tested - **public.**
- Make a test method for each test case

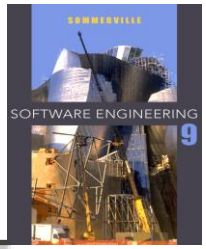
What can we do in in a test unit



- **Annotations**
- [TestClass] : set up the test
- [TestMethod] : This is a test method to be run
- [TestInitialize] : Run this before each test method
- [ClassInitialize] : Run this before the test starts

- **Testing verification**
- Assert.AreEqual(expected, actual)
- Assert.IsTrue(actual)

Test case in UNIT test



✧ Arrange

- Set up the test (part could be in test TestInitialize)
- Give all input the testing data
- Give expected data the expected values

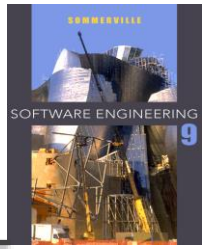
✧ Act

- Run the method

✧ Assert

- Check if the test have succeed

Special for exception



✧ Console programs

- Assert.ThrowsException<xxxException>(() => call method)
- Make try – catch : NB! The catch is ok = green
  - ```
    Try{
            Call method;
            Assert.Fail();
        Catch (){
            //Ok
        }
    ```

- Alternative make an annotation
  [ExpectedException typeof (xxxException)]
    ```