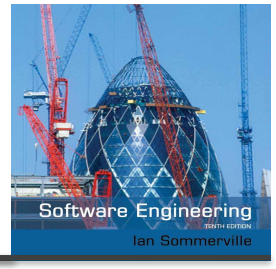
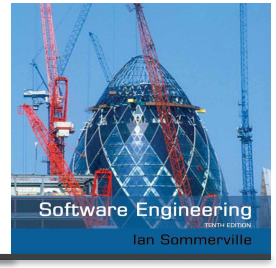
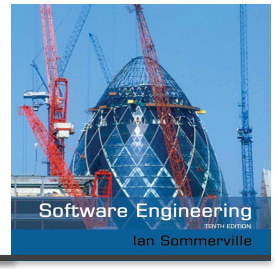


Chapter 8 – Software Testing



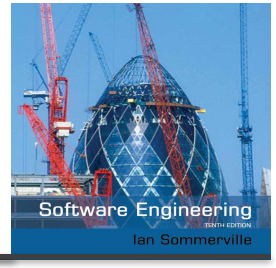
Development testing

Development testing



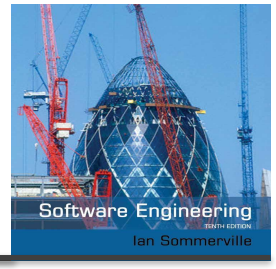
- **Development testing:**
  - Includes all testing activities that are carried out by the team developing the system.
  - The approaches for development testing are:
    - The **development team might be testing the software**
    - **developers/testers working in pairs** where a tester is testing the code simultaneously.
    - In safety-critical systems, a **separate testing team** works with developers and keeps a record of every test case run on the system.

Stages of Development testing



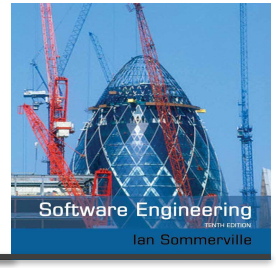
- **Unit testing,**
  - where individual program units/object classes are tested.
  - It focuses on **testing the functionality of objects or methods.**
  - Done in the development phase of SDLC
- **Component testing,**
  - Multiple individual units integrate to create composite components.
  - It focuses on testing components' compatibility and co-existence with other components. (proper data transfer, if they are dependent components)
  - Done in the testing phase of SDLC
- **System testing,**
  - where some or all the components in a system are integrated and the overall system is tested.
  - It focuses on **testing component interactions**.
  - Done in the testing phase of SDLC

Unit testing



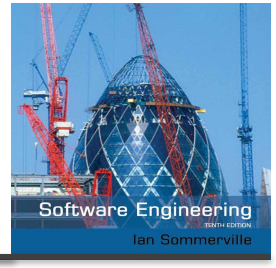
- Unit testing is the process of **testing individual components in isolation.**
- It is a **defect testing process.**
- Done in the development phase of SDLC
- **Check for all events** that can change an object state
- **Units** may be:
  - **Individual functions** in an object
  - **classes** with several attributes and methods
  - Composite components with defined interfaces used to access their functionality.

Object class testing



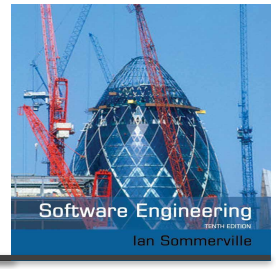
- Complete test coverage of a class involves
  - Testing all operations (methods) associated with an object
  - Checking for all object attributes
  - Simulate for all events causing a state change in system.
- **Inheritance makes it more difficult to design object class tests, as the information to be tested is not localized.**
  - Check for that inherited operatoion in all subclasses as the implementation might have been revised for subclasses because of differing properties.

The weather station object interface



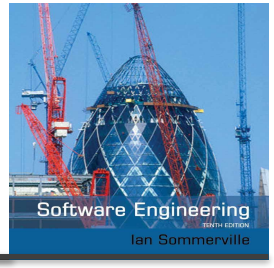
WeatherStation
identifier
reportWeather ( ) reportStatus ( ) powerSave (instruments) remoteControl (commands) reconfigure (commands) restart (instruments) shutdown (instruments)

Weather station testing



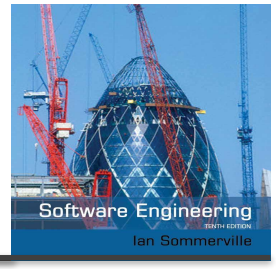
- If I want to check that the waeatherstation has been installed properly, I will make test cases for reportWeather and reportStatus.
- Ideally, methods are tested in isolation but sometimes **methods testcases sequence matters.**
  - To test shutdown method you need to execute the restart method to verify results.

Weather station testing



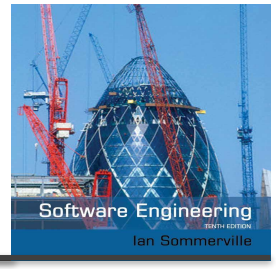
- Using a state model, **identify sequences of state transitions** to be tested **and the event sequences to cause these transitions**
- **Checking model for every possible state change which is expensive of course.**
- For example:
  - Shutdown -> Running-> Shutdown
  - Configuring-> Running-> Testing -> Transmitting -> Running
  - Running-> Collecting-> Running-> Summarizing -> Transmitting -> Running

Automated testing



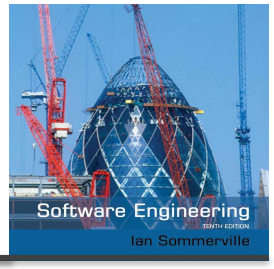
- unit testing should be automated so that tests are run and checked **without manual intervention**.
- Different frameworks could be used for automated testing.
- **These frameworks provide different test classes with which you can create specific test cases**. Entire test suite can run within seconds and show results.
- These frameworks can run all of the tests that you have implemented and report, success/failure status via GUI

Automated test components



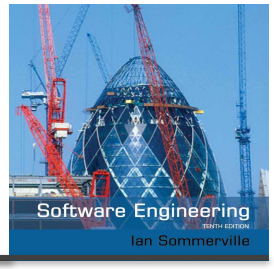
- **A setup part**, where you initialize the system with the test case, namely the inputs and expected outputs.
- **A call part**, where you call the object or method to be tested.
- **An assertion part** where you compare the result of the call with the expected result. If the assertion evaluates to true, the test has been successful, if false, then it has failed.
- It is **possible that the developed object has some dependencies on an object which is under development**. That will cause a delay. Meanwhile, use dummy objects for testing.

Automated test components



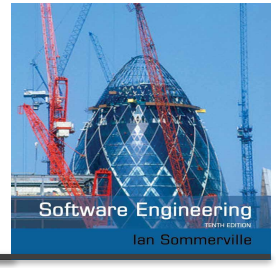

```
import org.jUnit.*;
@Test
class CalculatorTest{
    public void checkadd(){
        Calculator c = new Calculator();
        assertequals(12,c.add(10,2));
    }
}
```

Choosing unit test cases



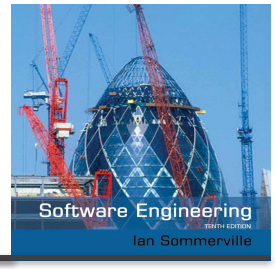
- The testing is a time taking procedure so its necessary to **define effective test cases.**
- **By effective we mean,**
  - *A test case should do what it is supposed to do* when given correct input
  - *A test case should identify defects* in component.

Choosing unit test cases



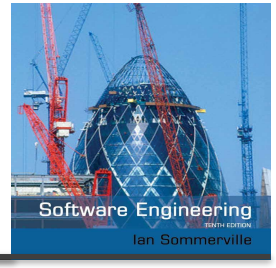
- Consider a scenario that takes the input of patient's record and initialize the inputs to the fields in databse
- This leads to two types of test cases:
  - The first of these should **reflect normal operation of a program and should show that the component works as expected.**
    - Inputs initialized and saved in db
  - 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.**
    - Abnormal inputs should not crash the unit. Rather they must generate errors gracefully and then terminate.

Strategies to choose test cases



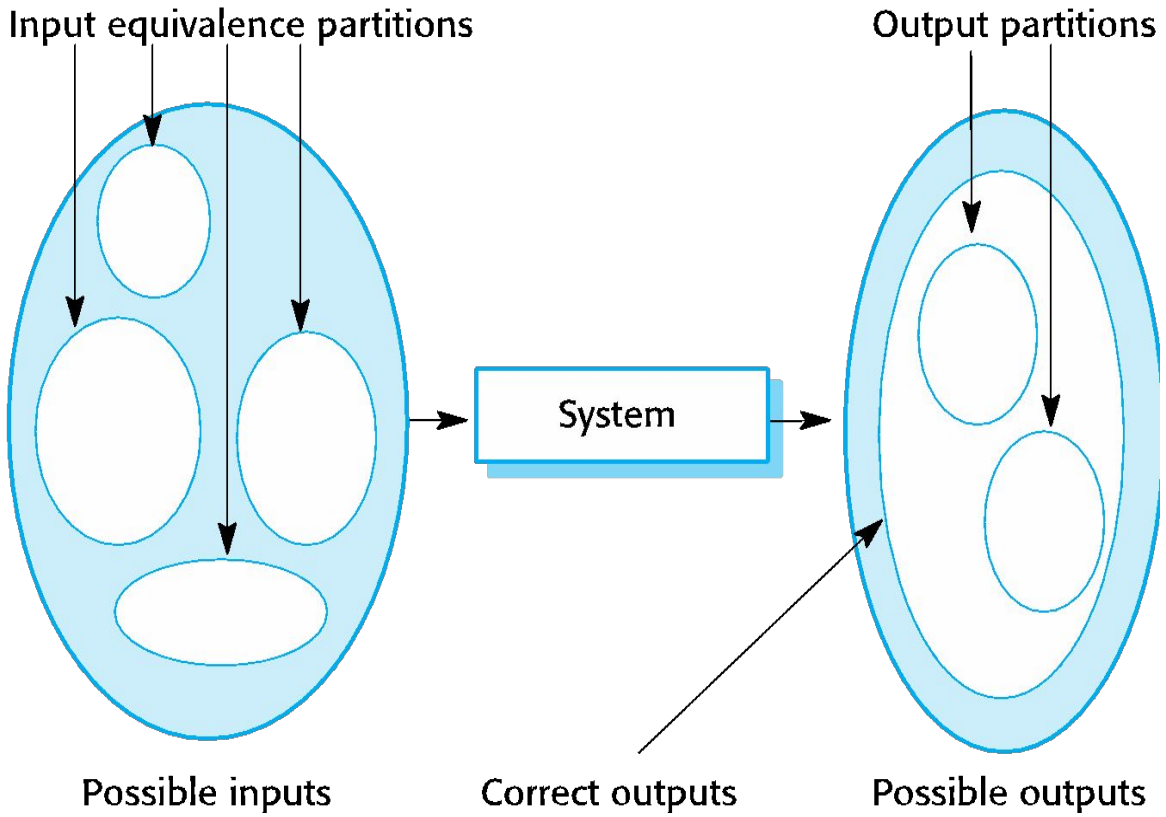
- **Partition testing,**
  - Identify groups of inputs having common characteristics and should be processed in the same way.
  - You should choose tests from within each of these groups.
- **Guideline-based testing,**
  - where you use testing guidelines to choose test cases.
  - These guidelines reflect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

Partition testing

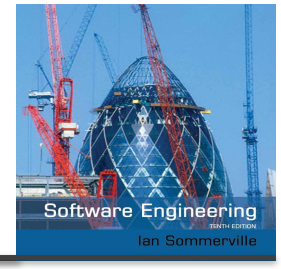


- Input data and output results often fall into different classes where all members of a class are related.
- For example,
  - If a unit sums up the inputs given to it, then create a test case for two positive numbers addition.
  - This test case will be valid for testing sum of all positive numbers.
- That's way for each class, its called as equivalence partitioning.
- Test cases should be chosen from each partition.
- Better to check test cases for boundary values of the partitions as they are generally the atypical values.
  - For example: results on 0 might differ from result for negative values.

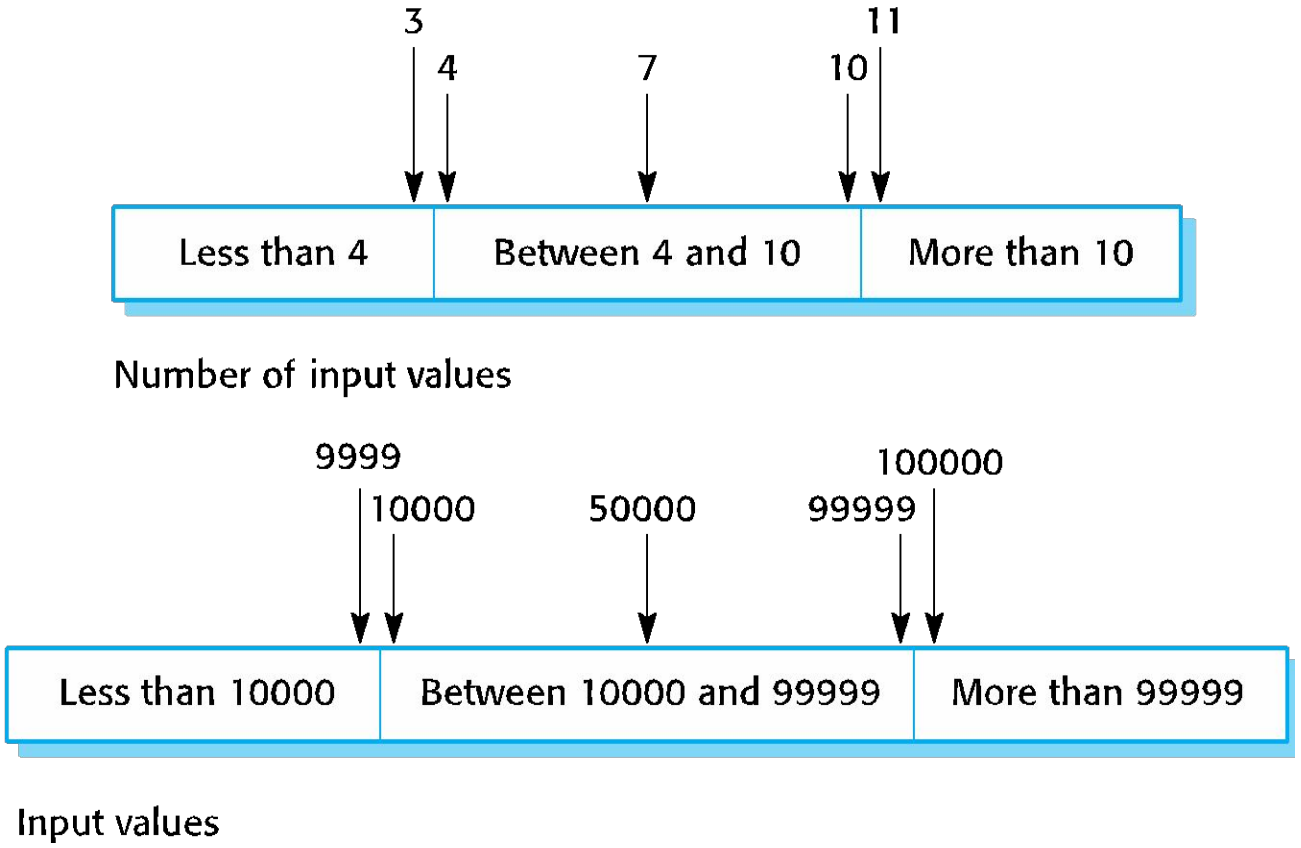
Equivalence partitioning



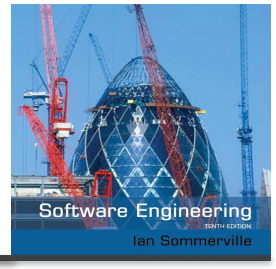
Equivalence partitions



Program accepts 4-10 inputs which are 5 digit int greater than 10,000.

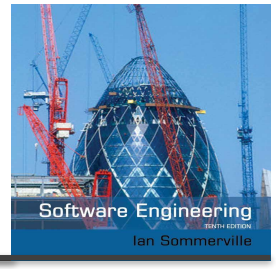


Testing guidelines (sequences)



- Guidelines to identify defects while testing arrays, sequences, lists etc.
- Use arrays of different sizes in different tests.
- Derive tests so that the first, middle, and last elements of the arrays are accessed.
- Test for an array length of -1.

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 to verify system consistency.
- Force invalid outputs to be generated
- Force computation results to be too large or too small.