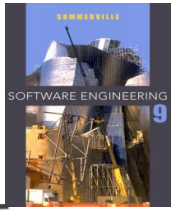


# Chapter 8 – Software Testing

## Lecture 1

# Topics covered

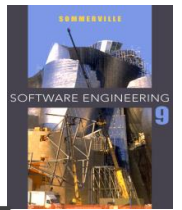
---



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

# Program testing

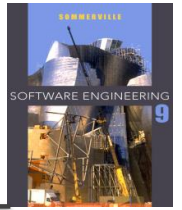
---



- ✧ 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, which also includes static validation techniques.

# Testing process goals

---



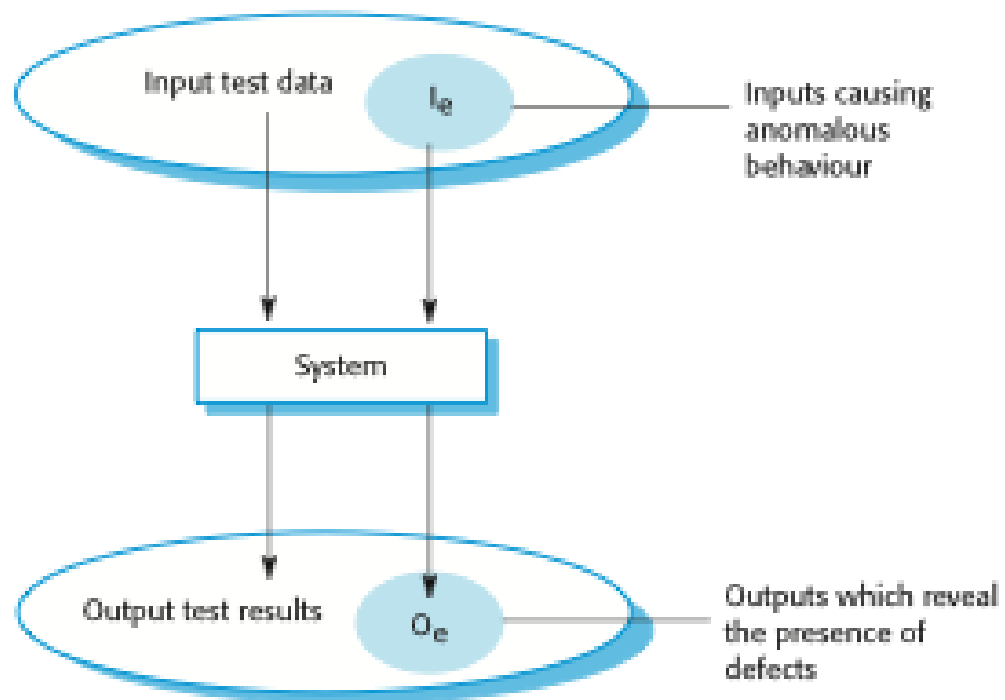
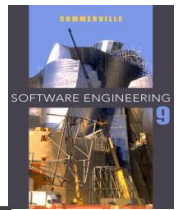
## ✧ Validation testing

- 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

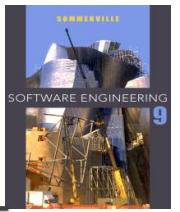
- To discover faults or defects in the software where its behaviour is incorrect or not in conformance with its specification
- A successful test is a test that makes the system perform incorrectly and so exposes a defect in the system.

# An input-output model of program testing



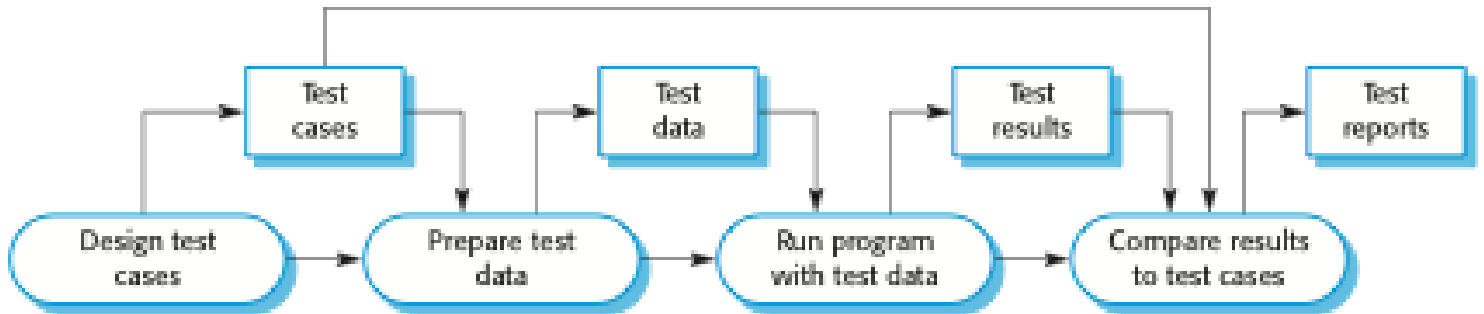
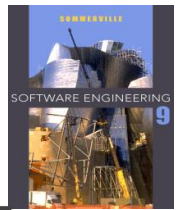
# Verification vs validation

---



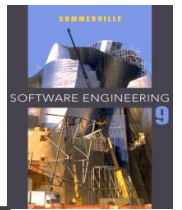
- ✧ Verification:
  - "Are we building the product right".
- ✧ The software should conform to its specification.
- ✧ Validation:
  - "Are we building the right product".
- ✧ The software should do what the user really requires.

# A model of the software testing process



# Stages of testing

---

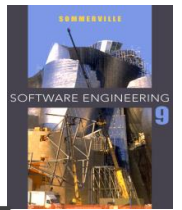


- ✧ Development testing, where the system is tested during development to discover bugs and defects.
- ✧ Release testing, where a separate testing team test a complete version of the system before it is released to users.
- ✧ User testing, where users or potential users of a system test the system in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



# Development testing

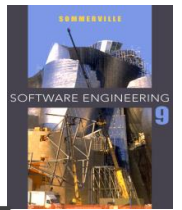
---



- ✧ 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

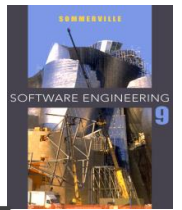
---



- ✧ 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
  - Composite components with defined interfaces used to access their functionality.

# Object class testing

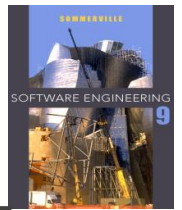
---



- ✧ 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

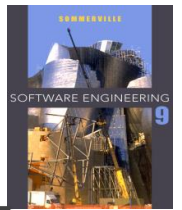
---



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

# Weather station testing

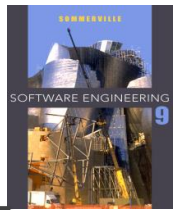
---



- ✧ 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

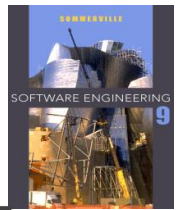
---



- ✧ 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 or otherwise of the tests.

# Automated test components

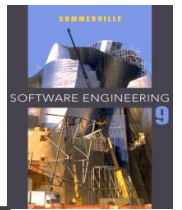
---



- ✧ 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.

# Testing strategies

---

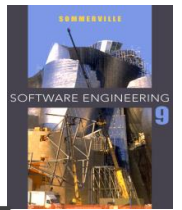


- ✧ Partition testing, where you identify groups of inputs that have common characteristics and should be processed in the same way.
  - You should choose tests from within each of these groups.
- ✧ Guideline-based testing, where you use testing guidelines to choose test cases.
  - These guidelines reflect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



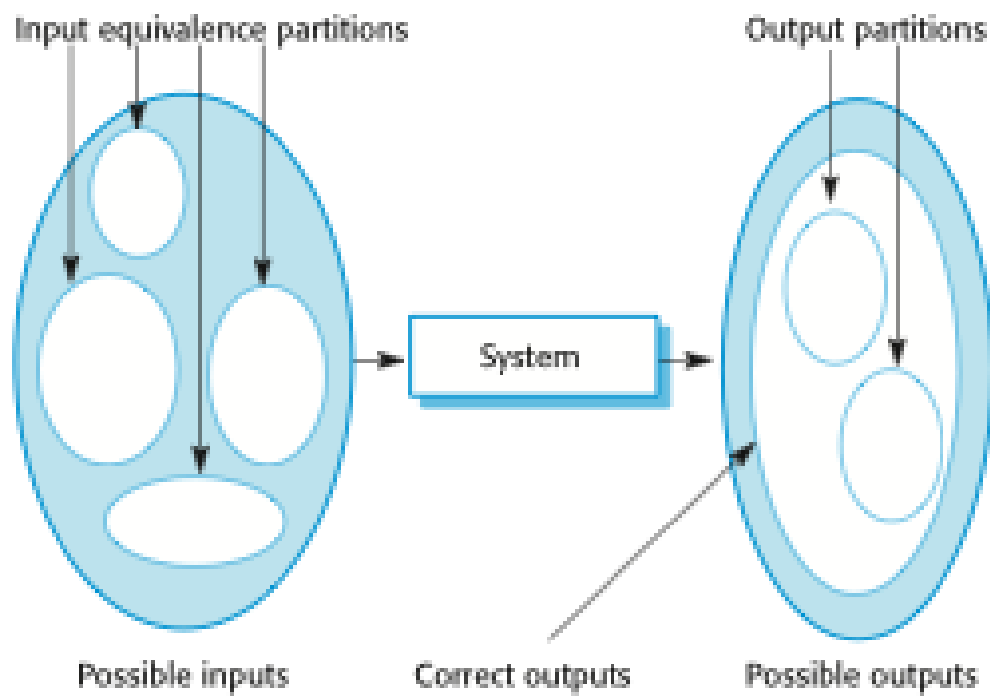
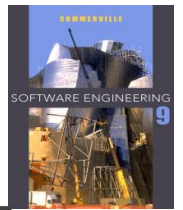
# Partition testing

---

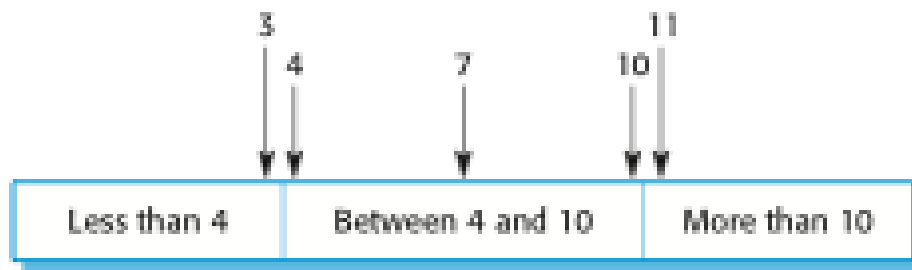
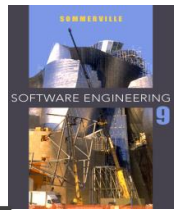


- ✧ Input data and output results often fall into different classes where all members of a class are related.
- ✧ Each of these classes is an **equivalence partition** or domain where the program behaves in an equivalent way for each class member.
- ✧ Test cases should be chosen from each partition.

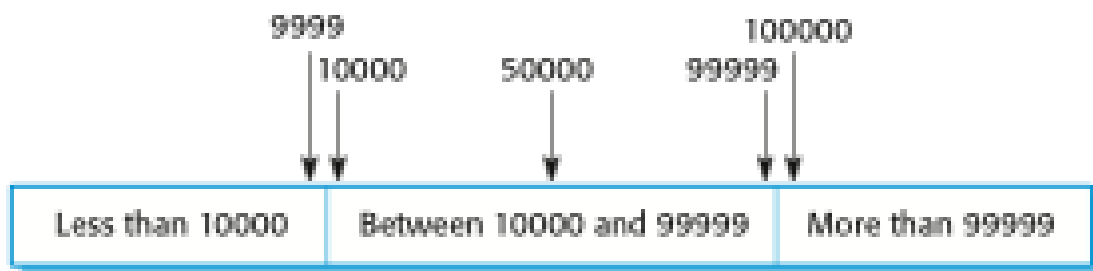
# Equivalence partitioning



# Equivalence partitions



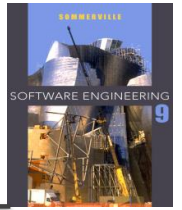
Number of input values



Input values

# Testing guidelines (sequences)

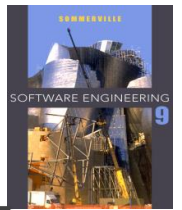
---



- ✧ 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

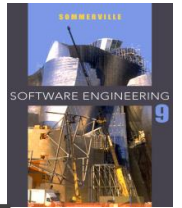
---



- ✧ 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.

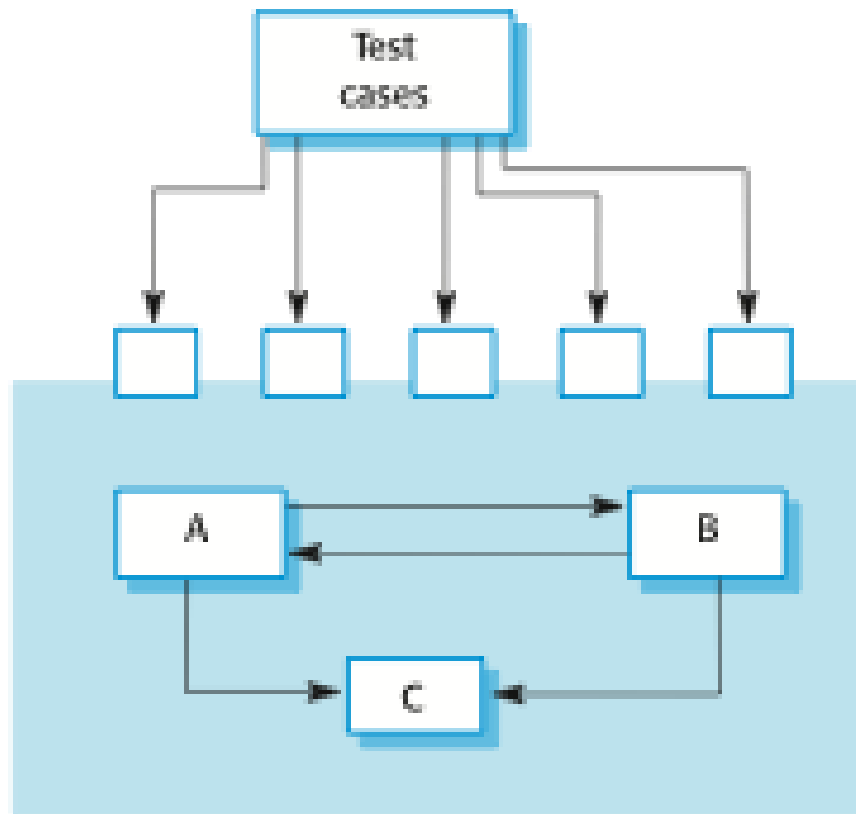
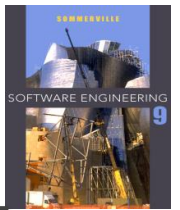
# Component testing

---



- ✧ Software components are often composite components that are made up of several interacting objects.
  - For example, in the weather station system, the reconfiguration component includes objects that deal with each aspect of the reconfiguration.
- ✧ You access the functionality of these objects through the defined component interface.
- ✧ Testing composite components should therefore focus on showing that the component interface behaves according to its specification.
  - You can assume that unit tests on the individual objects within the component have been completed.

# Interface testing

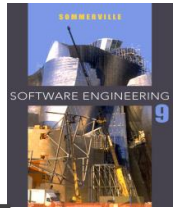


- ✧ Objectives are to detect faults due to interface errors or invalid assumptions about interfaces.
- ✧ Interface types
  - **Parameter interfaces** Data passed from one method or procedure to another.
  - **Shared memory interfaces** Block of memory is shared between procedures or functions.
  - **Procedural interfaces** Sub-system encapsulates a set of procedures to be called by other sub-systems.
  - **Message passing interfaces** Sub-systems request services from other sub-systems



# System testing

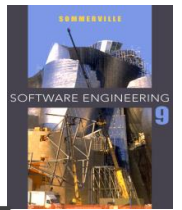
---



- ✧ System testing during development involves integrating components to create a version of the system and then testing the integrated system.
- ✧ The focus in system testing is 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.
- ✧ System testing tests the emergent behaviour of a system.

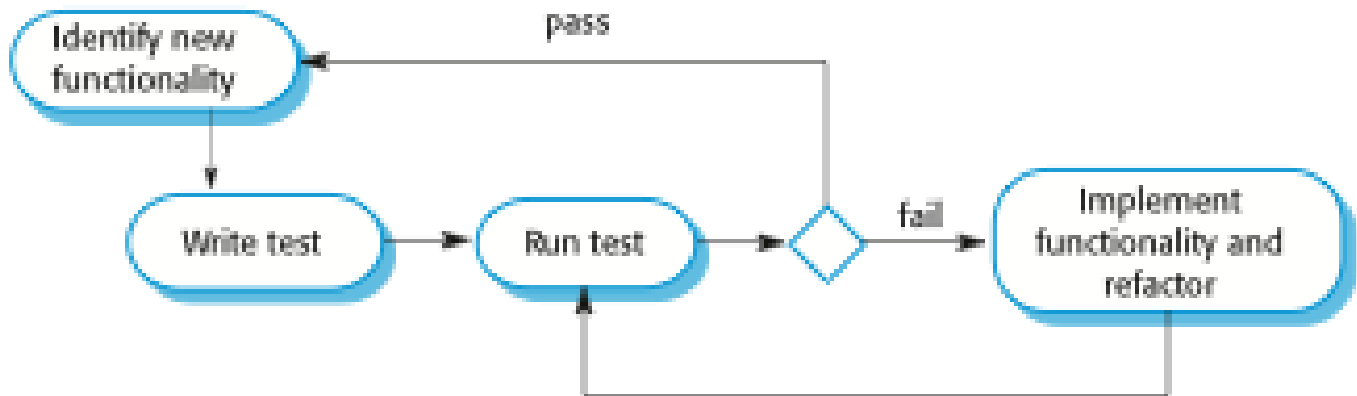
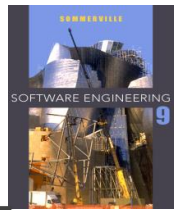
# Test-driven development

---



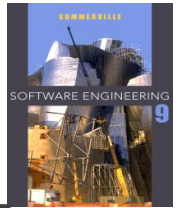
- ✧ Test-driven development (TDD) is an approach to program development in which you inter-leave testing and code development.
- ✧ Tests are written before code and ‘passing’ the tests is the critical driver of development.
- ✧ You develop code incrementally, along with a test for that increment. You 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



# TDD process activities

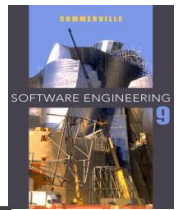
---



- ✧ 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.

# Release testing

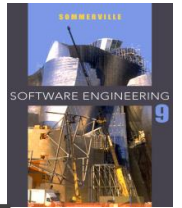
---



- ✧ Release testing is the process of testing a particular release of a system that is intended for use outside of the development team.
- ✧ The primary goal of the release testing process is to convince the supplier of the system that it is good enough for use.
  - Release testing, therefore, has to show that the system delivers its specified functionality, performance and dependability, and that it does not fail during normal use.
- ✧ Release testing is usually a black-box testing process where tests are only derived from the system specification.

# Release testing and system testing

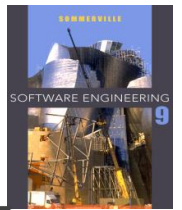
---



- ✧ Release testing is a form of system testing.
- ✧ Important differences:
  - A separate team that has not been involved in the system development, should be responsible for release testing.
  - System testing by the development team should focus on discovering bugs in the system (defect testing). The objective of release testing is to check that the system meets its requirements and is good enough for external use (validation testing).

# Requirements based testing

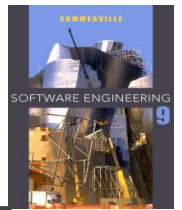
---



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

---

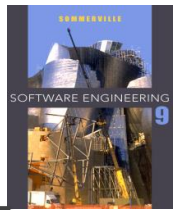


- ✧ Set up a patient record with no known allergies. Prescribe medication for allergies that are known to exist. Check that a warning message is not issued by the system.
- ✧ Set up a patient record with a known allergy. Prescribe the medication to that the patient is allergic to, and check that the 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.



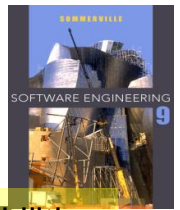
## Features tested by scenario

---



- ✧ Authentication by logging on to the system.
- ✧ Downloading and uploading of specified patient records to a laptop.
- ✧ Home visit scheduling.
- ✧ Encryption and decryption of patient records on a mobile device.
- ✧ Record retrieval and modification.
- ✧ Links with the drugs database that maintains side-effect information.
- ✧ The system for call prompting.

# A usage scenario for the MHC-PMS



Kate is a nurse who specializes in mental health care. One of her responsibilities is to visit patients at home to check that their treatment is effective and that they are not suffering from medication side -effects.

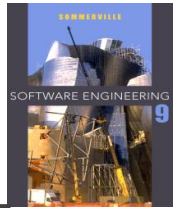
On a day for home visits, Kate logs into the MHC-PMS and uses it to print her schedule of home visits for that day, along with summary information about the patients to be visited. She requests that the records for these patients be downloaded to her laptop. She is prompted for her key phrase to encrypt the records on the laptop.

One of the patients that she visits is Jim, who is being treated with medication for depression. Jim feels that the medication is helping him but believes that it has the side -effect of keeping him awake at night. Kate looks up Jim's record and is prompted for her key phrase to decrypt the record. She checks the drug prescribed and queries its side effects. Sleeplessness is a known side effect so she notes the problem in Jim's record and suggests that he visits the clinic to have his medication changed. He agrees so Kate enters a prompt to call him when she gets back to the clinic to make an appointment with a physician. She ends the consultation and the system re-encrypts Jim's record.

After, finishing her consultations, Kate returns to the clinic and uploads the records of patients visited to the database. The system generates a call list for Kate of those patients who she has to contact for follow-up information and make clinic appointments.

# Performance testing

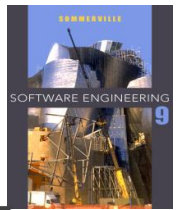
---



- ✧ 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.
- ✧ Performance tests usually involve planning a series of tests where 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

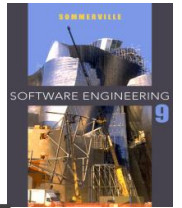
---



- ✧ User 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.
  - The reason for this is that 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.

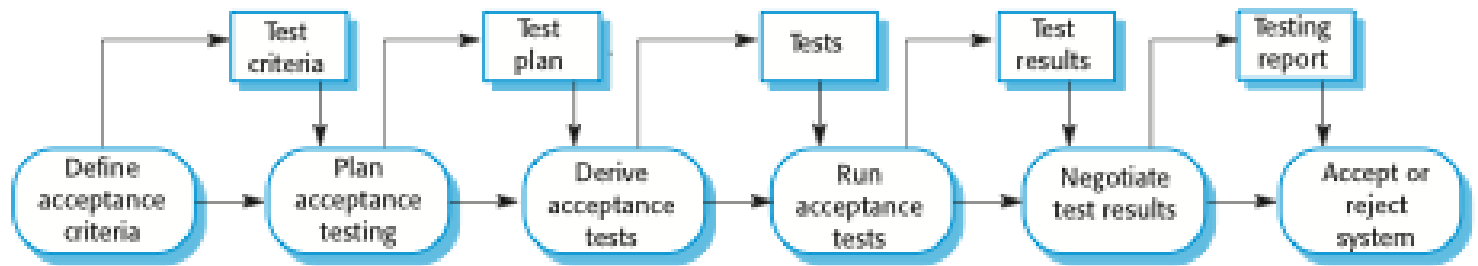
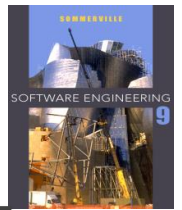
## ✧ Beta testing

- 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

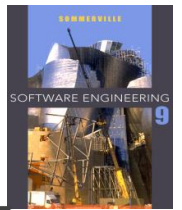
- Customers test a system to decide whether or not it is ready to be accepted from the system developers and deployed in the customer environment. Primarily for custom systems.

# The acceptance testing process



# Stages in the acceptance testing process

---



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