Software Engineering

Week # 7 (b)

LECTURER: ABDULRAHMAN QAIM

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.

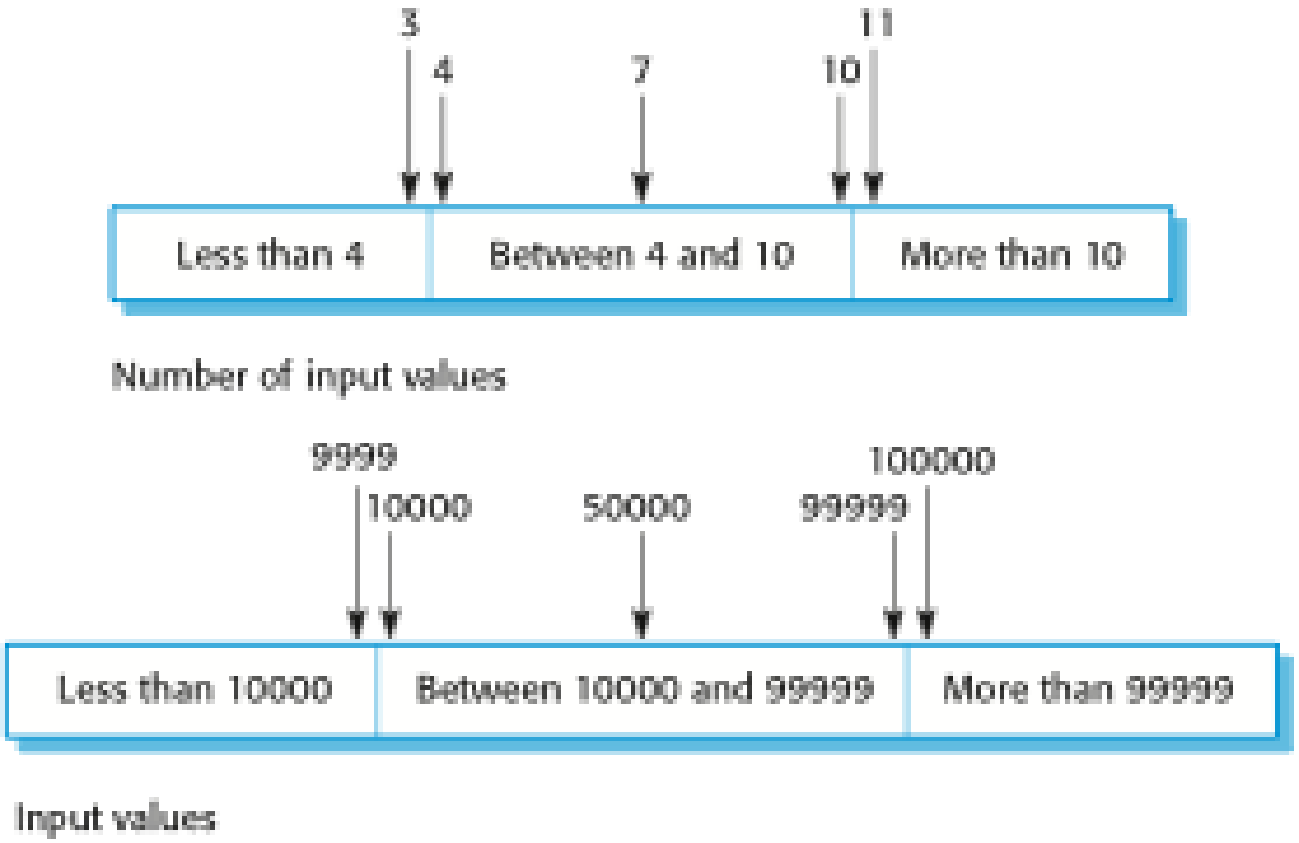
Partition testing

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 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.

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

Interface errors

Interface misuse

- A calling component calls another component and makes an error in its use of its interface e.g. parameters in the wrong order.

Interface misunderstanding

- A calling component embeds assumptions about the behaviour of the called component which are incorrect.

Timing errors

- The called and the calling component operate at different speeds and out-of-date information is accessed.

System testing

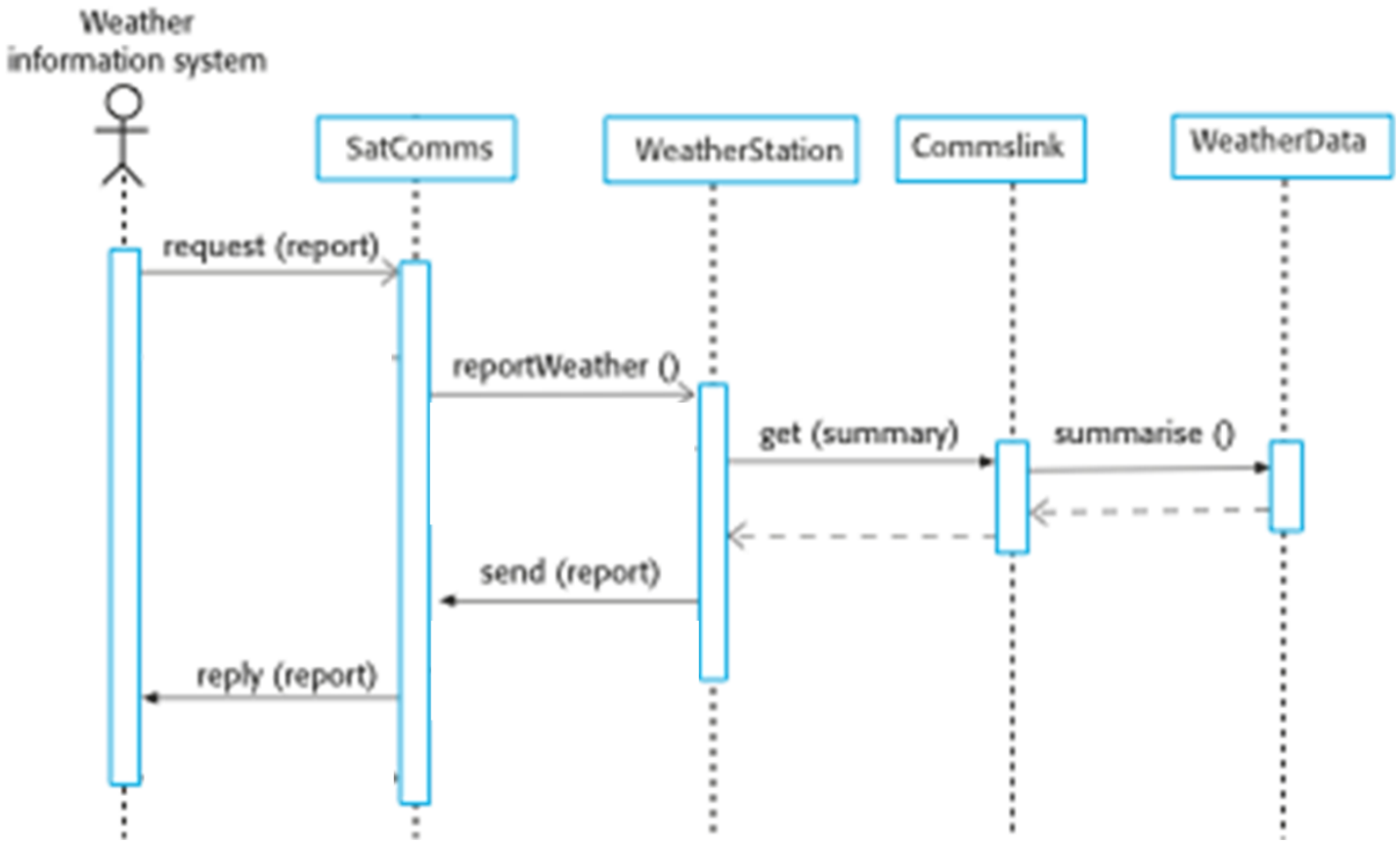
System testing during development involves integrating components to create a version of the system and then testing the integrated system.

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.

Collect weather data sequence chart



Testing policies

Exhaustive system testing is impossible so testing policies which define the required system test coverage may be developed.

Examples of testing policies:

- All system functions that are accessed through menus should be tested.
- Combinations of functions (e.g. text formatting) that are accessed through the same menu must be tested.
- Where user input is provided, all functions must be tested with both correct and incorrect input.

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.

Regression testing (Important)

Regression testing is testing the system to check that changes have not ‘broken’ previously working code.

In a manual testing process, regression testing is expensive but, with automated testing, it is simple and straightforward. All tests are rerun every time a change is made to the program.

Tests must run ‘successfully’ before the change is committed.

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.

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.

Performance testing (Important)

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 (Important)

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.

Retesting (Important)

Retesting is when a test is performed again on a specific feature not functional during the previous test to check for its functionality.

Retesting is typically performed by the same testers who identified the defect in the first place.

Retesting has tests explicitly designed to check whether known bugs have been fixed.	Regression testing isn't targeted testing for known defects.
Since retesting checks for a specific defect, it can't be automated.	Automation is prevalent for regression testing. Manual testing every time a change or update is made to an application would be very irrational.

Security Testing

Security testing is a type of software testing that assesses the security of a software application. It helps to identify vulnerabilities and weaknesses in the system and ensure that sensitive data is protected.

1. Penetration testing: This involves attempting to exploit potential vulnerabilities in the software system by simulating an attack from a hacker or other malicious actor.
2. Fuzz testing: This involves sending many unexpected or malformed input data to the software system to identify potential vulnerabilities related to input validation and handling.
3. Access control testing: This involves testing the software system’s access control mechanisms in order to make sure that access to sensitive data is granted only to authorized users.

Black Box vs White Box Testing

Black box testing is a software testing methodology in which the tester analyzes the functionality of an application without a thorough knowledge of its internal design. Conversely, in white box testing, the tester is knowledgeable of the internal design of the application and analyzes it during testing.

The software application’s internal coding, design, and structure are examined in white box testing to verify data flow from input to output. White box testing is leveraged to improve design, usability, and application security.

Key points

When testing software, you should try to ‘break’ the software by using experience and guidelines to choose types of test case that have been effective in discovering defects in other systems.

Wherever possible, you should write automated tests. The tests are embedded in a program that can be run every time a change is made to a system.

Test-first development is an approach to development where tests are written before the code to be tested.

Scenario testing involves inventing a typical usage scenario and using this to derive test cases.

Acceptance testing is a user testing process where the aim is to decide if the software is good enough to be deployed and used in its operational environment.