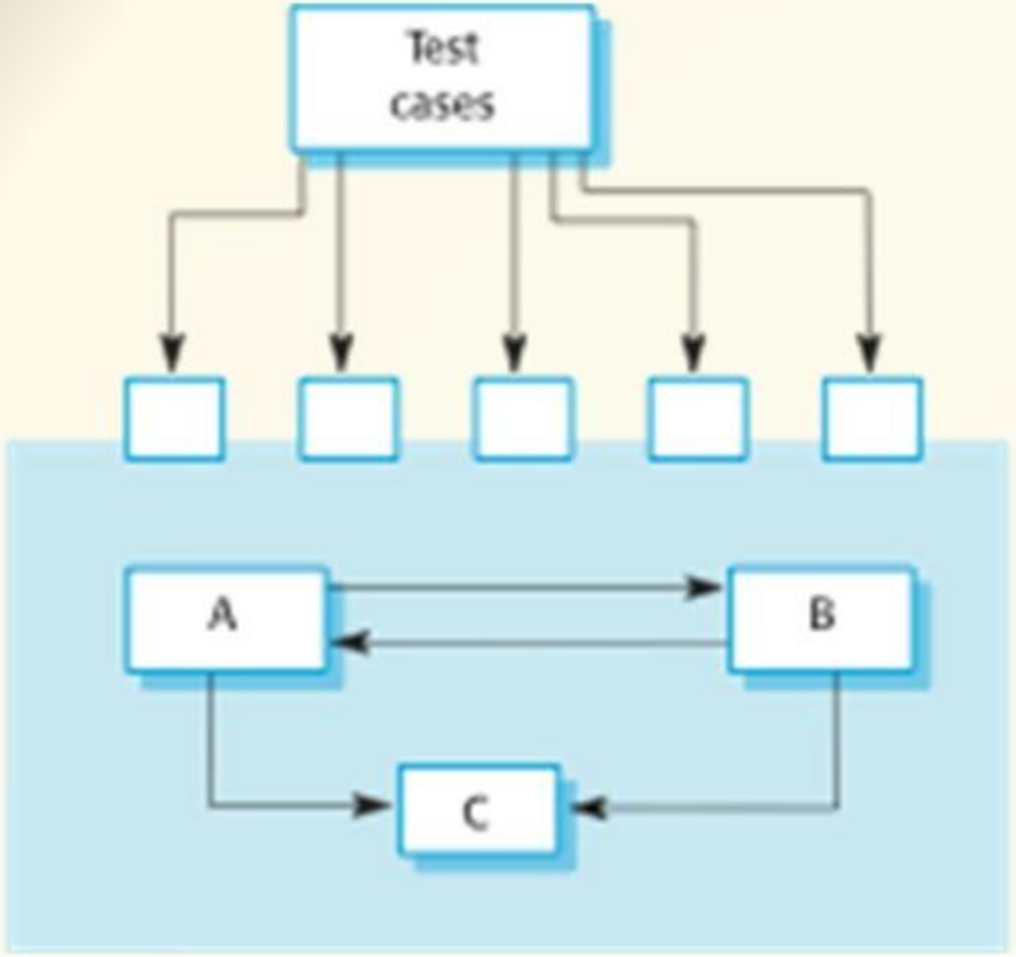
Chapter 6 – Software Testing

Lecture 2

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



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- The called and the calling component operate at different speeds and out-of-date information is accessed.

Interface testing guidelines

- Design tests so that parameters to a called procedure are at the extreme ends of their ranges.
- Always test pointer parameters with null pointers.
- Design tests which cause the component to fail.
- Use stress testing in message passing systems.
- In shared memory systems, vary the order in which components are activated.

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.

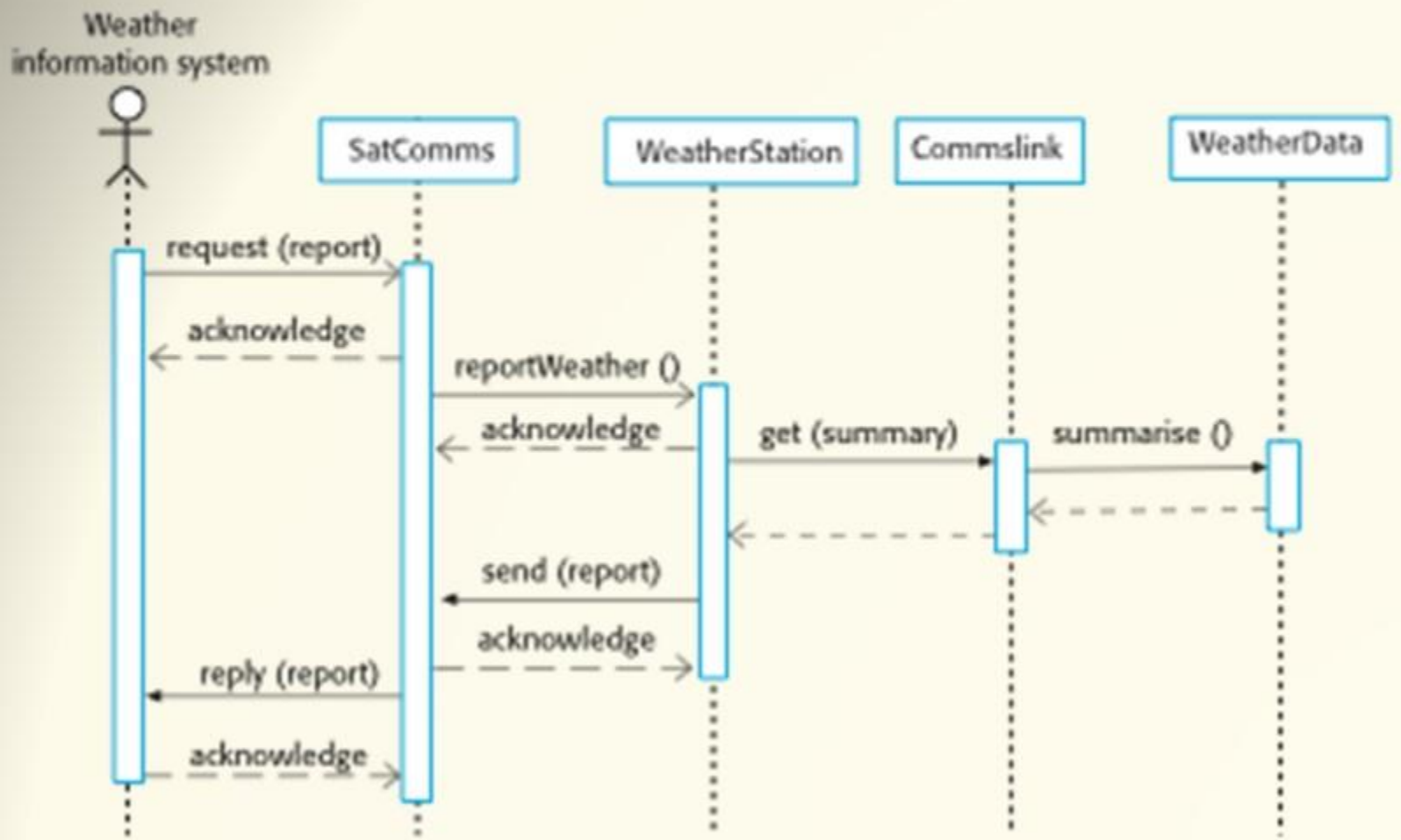
System and component testing

- During system testing, reusable components that have been separately developed and off-the-shelf systems may be integrated with newly developed components. The complete system is then tested.
- Components developed by different team members or sub-teams may be integrated at this stage. System testing is a collective rather than an individual process.
  - In some companies, system testing may involve a separate testing team with no involvement from designers and programmers.

Use-case testing

- The use-cases developed to identify system interactions can be used as a basis for system testing.
- Each use case usually involves several system components so testing the use case forces these interactions to occur.
- The sequence diagrams associated with the use case documents the components and interactions that are being tested.

Collect weather data sequence chart



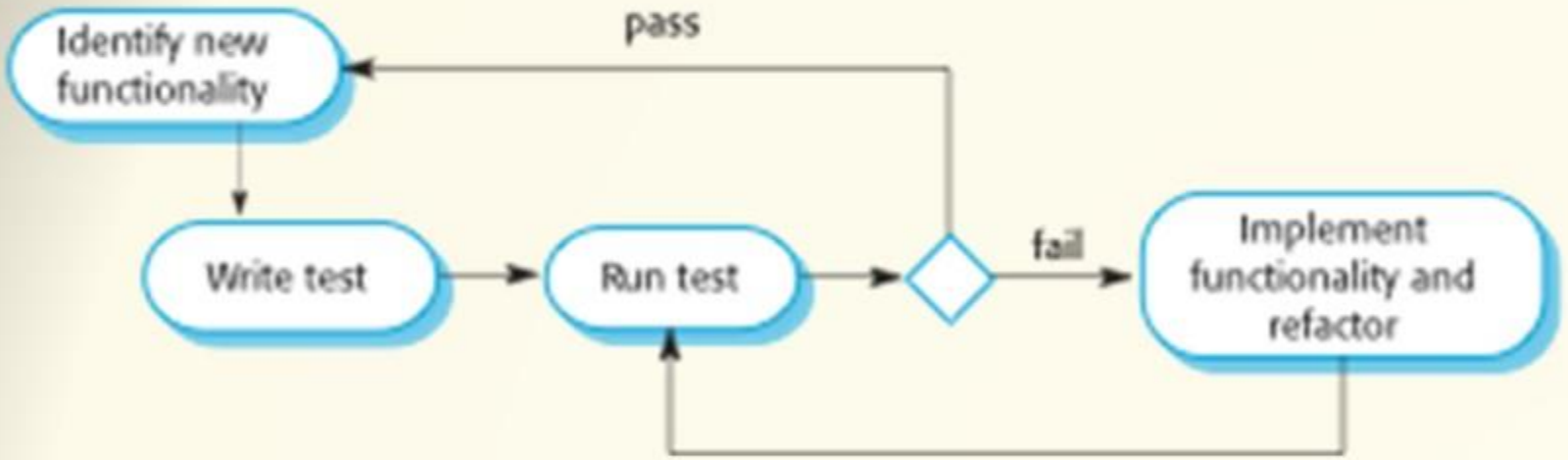
Testing policies

- 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.

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.

Benefits of test-driven development

- Code coverage
  - Every code segment that you write has at least one associated test so all code written has at least one test.
- Regression testing
  - A regression test suite is developed incrementally as a program is developed.
- Simplified debugging
  - When a test fails, it should be obvious where the problem lies. The newly written code needs to be checked and modified.
- System documentation
  - The tests themselves are a form of documentation that describe what the code should be doing.

Regression testing

- 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.

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.

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.

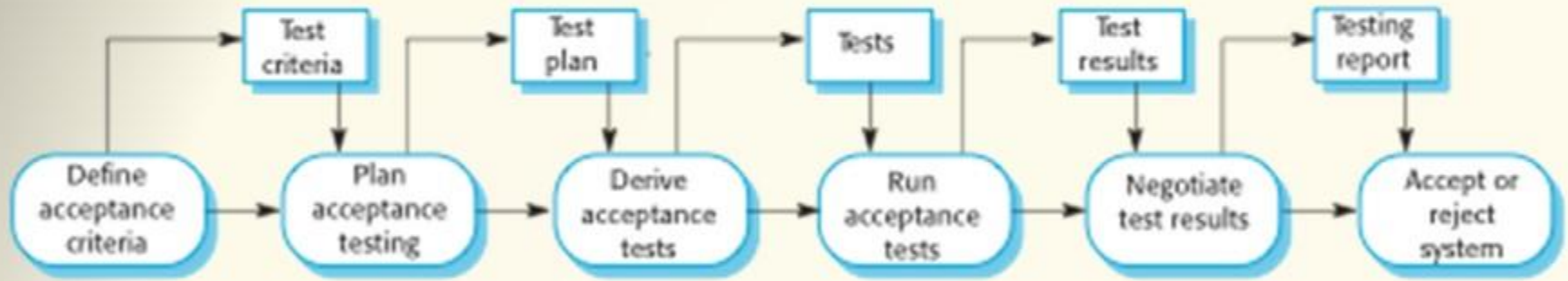
✧ Beta testing

- 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

Agile methods and acceptance testing

- In **agile methods**, the user/customer is part of the development team and is responsible for making decisions on the acceptability of the system.
- Tests are defined by the user/customer and are integrated with other tests in that they are run automatically when changes are made.
- There is no separate acceptance testing process.
- Main problem here is whether or not the embedded user is 'typical' and can represent the interests of all system stakeholders.

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.