COMP201 – Software Engineering I
Lecture 27 – Software Testing

Lecturer: Dr. T. Carroll
Email: Thomas.Carroll2@Liverpool.ac.uk
Office: G.14
See Vital for all notes

Recap

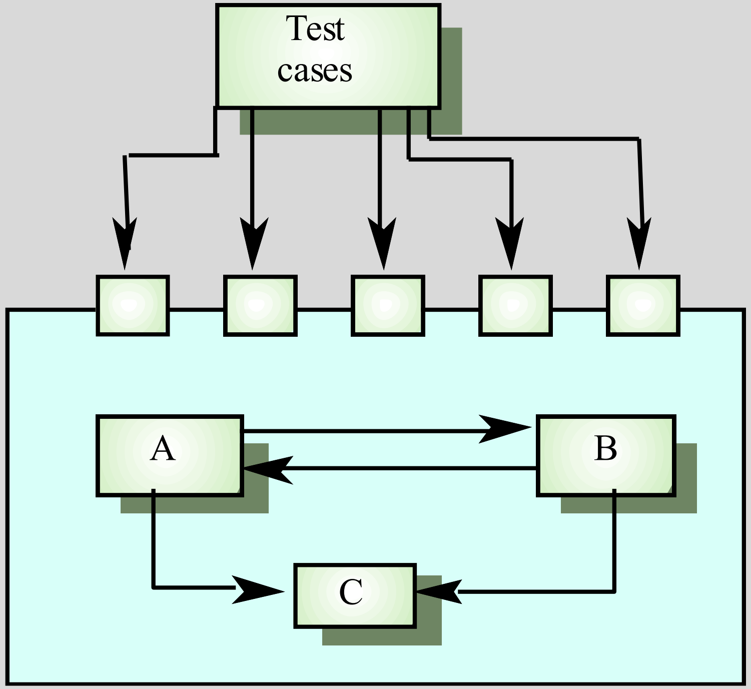
Lecture 26 Recap

- Path testing is ensuring that each possible path of the programs control flow has been tested
- Control Flow Graphs can help us to visualise this
- **Cyclomatic complexity** allows us to measure the amount of tests required – but not the adequacy of them
- Integration testing comes **after** component testing
  - Top-Down good for architecture testing, system usability testing/demos, etc…
  - Bottom-up good for OO, real time systems, systems with performance constraints

Interface Testing

Interface Testing

- Takes place when modules or sub-systems are integrated to create larger systems
- Objectives are to detect faults due to **interface errors** or invalid assumptions about interfaces
- Particularly important for OO development: **objects are defined by their interfaces**



Interfaces Types

- Parameter interfaces
  - Data passed from one procedure to another
- Shared memory interfaces
  - Block of memory is shared between procedures
- 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 Testing uncovers 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  - The called and the calling component operate at different speeds and out-of-date information is accessed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 (think of **partition testing…**)
- 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

Stress Testing

- Exercises the system **beyond its maximum design load**.
- Stressing the system often causes defects to come to light
- Stressing the system tests **failure behaviour**.
  - Systems should not fail catastrophically.
- Stress testing checks for unacceptable loss of service or data
- Particularly relevant to **distributed systems** which can exhibit severe degradation as a network becomes overloaded

OO Testing

Object-Oriented Testing

- The components to be tested are object classes that are instantiated as objects
- Larger grain than individual functions
  - We must **extend** glass-box testing
- No obvious ‘top’ to the system for top-down integration and testing

Testing Levels

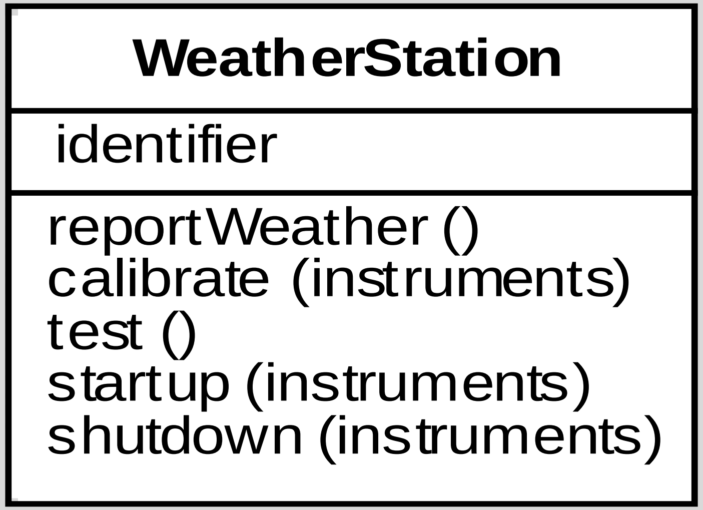
- Testing object **classes** (including all **operations** associated with objects)
- Testing **clusters** of cooperating objects
- Testing the **complete OO system**

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

Object Class Testing Example :
Weather Station Object Interface

- Test cases are needed for **all operations**
- Use a **state model** to identify state transitions for testing
- Examples of testing sequences
  - Shutdown → Waiting → Shutdown
  - Waiting → Calibrating → Testing → Transmitting → Waiting
  - Waiting → Collecting → Waiting → Summarising → Transmitting → Waiting



WeatherStation

identifier

reportWeather ()
calibrate (instruments)
test ()
startup (instruments)
shutdown (instruments)

Object Integration

- Levels of integration are less distinct in object-oriented systems
- **Cluster testing** is concerned with integrating and testing clusters of cooperating objects
- Identify clusters using:
  - knowledge of the operation of objects
  - system features that are implemented by these clusters

Approaches to Cluster Testing

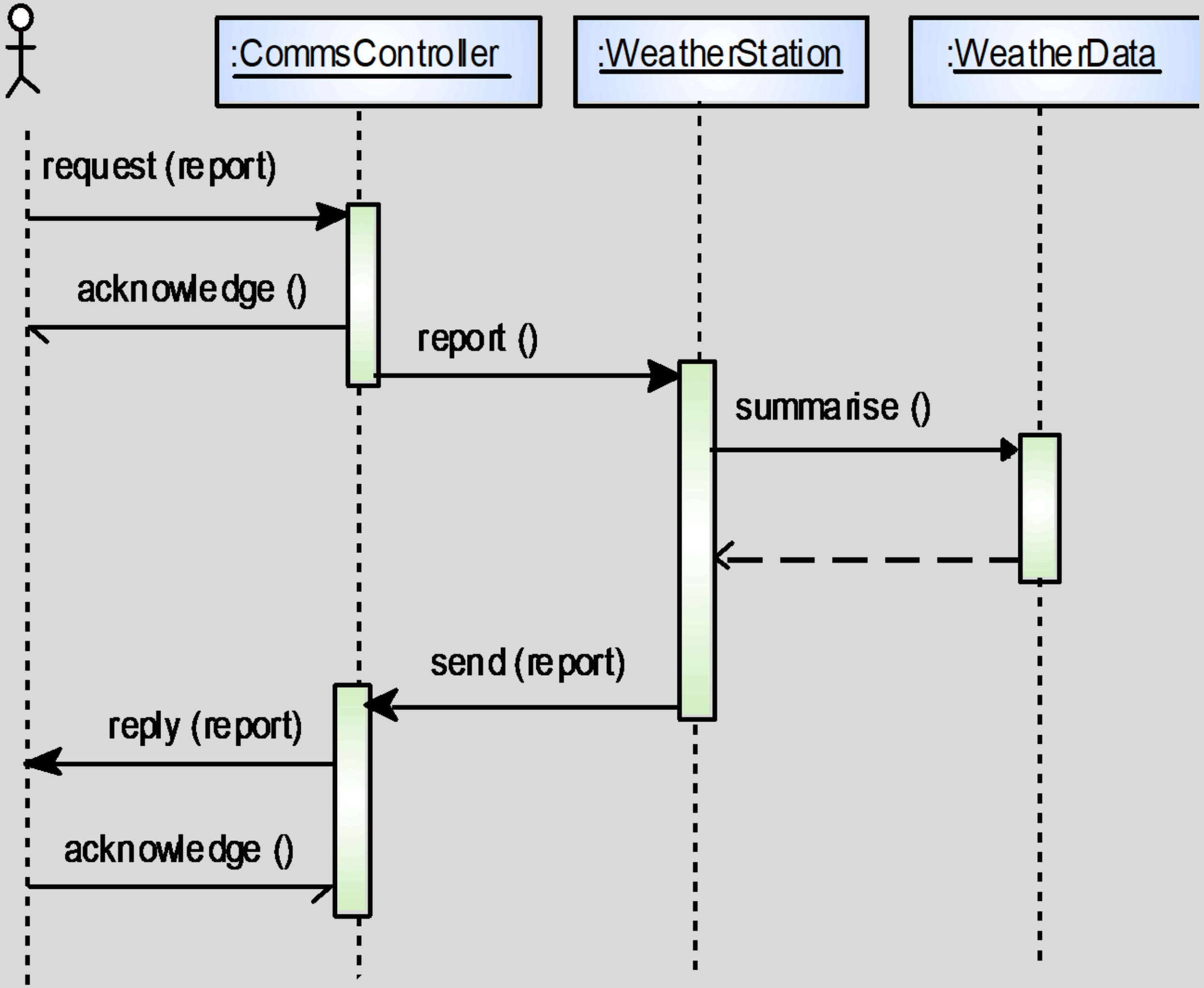
- Use-case or scenario testing
  - Based on user interactions with the system
  - Tests system features as experienced by users
- Thread testing
  - Tests the systems response to events as processing threads through the system
- Object interaction testing
  - Tests sequences of object interactions that stop when an object operation does not call on services from another object

Scenario-Based Testing

- Identify scenarios from use-cases
- Supplement these with **interaction diagrams** that show the objects involved in the scenario

Scenario Testing Example: Weather Station Testing

- Thread of methods executed
  - CommsController:request→ WeatherStation:report→ WeatherData:summarise
- Inputs and outputs
  - Input of report request with associated acknowledge and a final output of a report
  - Can be tested by creating raw data and ensuring that it is summarised properly
  - Use the same raw data to test the WeatherData object



Lecture Recap

Lecture Key Points

- Interface defects arise because of specification misreading, misunderstanding, errors or invalid timing assumptions
- To test object classes, we must:
  - Test all operations
  - Test all attributes
  - Test all states
- Integrate object-oriented systems around **clusters of objects**