Software Testing I

Mojtaba Shahin

Week #9: Lecture - Part #2

Topics

- **Part 2**
  - Development Testing

Notes and Acknowledgements

- Slides/images come from the following main sources:
  - Ian Sommerville, Software Engineering, 10th Edition, 2015.
    - https://iansommerville.com/software-engineering-book/slides/
  - (Partial) Introduction to Software Engineering Practices and Methods by Dr. Laurie Williams NCSU CSC326 Course Pack 2010-2011 (Seventh) Edition
    - https://sdc.csc.ncsu.edu/files/resources/williams-software-engineering-2011.pdf

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

- 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.

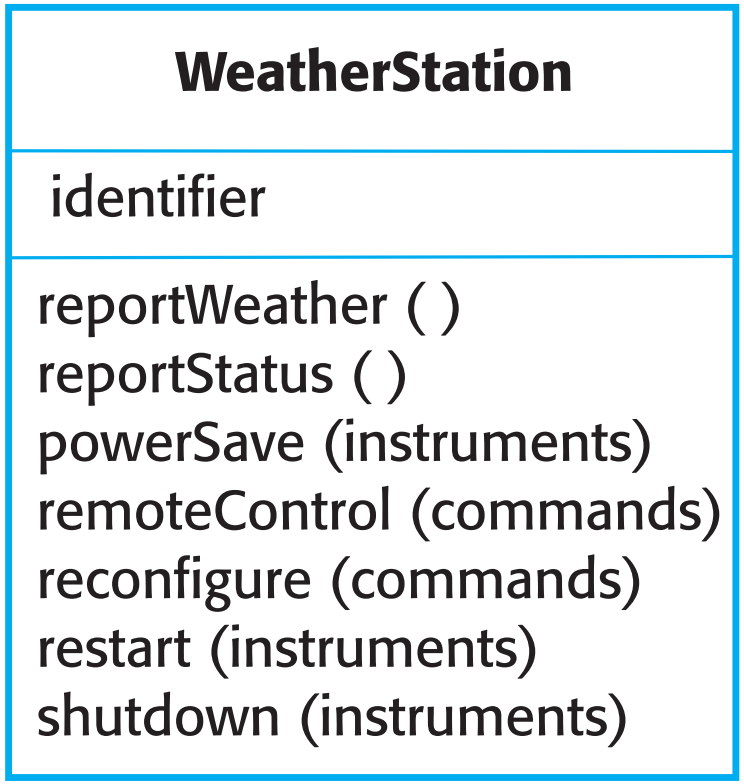
Unit testing

- Unit testing is the process of testing individual units 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 are used to access their functionality.

Object class testing

- Complete test coverage of a class involves
  - Testing all operations associated with an object
  - Testing all object attributes
  - Exercising the object in all possible states.

The weather station object interface



Weather station testing

- The class has a single attribute, which is its **identifier**. This is a constant that is set when the weather station is installed. You therefore only need a test that checks if it has been properly set up.
- You need to define test cases for all of the methods associated with the object such as **reportWeather** and **reportStatus**.

Choosing unit test cases

- There should be 2 types of unit test cases:
  - The first of these should **reflect the normal operation of a program** and should show that the component/unit works as expected (***success path***).
  - The other kind of test case should be based on the 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 (***failure path***).

Tests from Customer Requirements

- Consider the following Mentcare system requirements that are concerned with checking for drug allergies:

Requirement

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.

Requirement

If a prescriber chooses to ignore an allergy warning, they shall provide a reason why this has been ignored.

Several related tests related the two requirements

1. 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.
2. 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.
3. 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.
4. **Prescribe two drugs that the patient is allergic to. Check that two warnings are correctly issued.**
5. 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.

Tests from Customer Requirements

Requirement

When a user lands on the "Go to Jail" cell, the player goes directly to jail, does not pass go, and does not collect $200. On the next turn, the player must pay $50 to get out of jail and does not roll the dice or advance. If the player does not have enough money, he or she is out of the game.

- **There are many things to test in this short requirement above, including:**
  1. Does the player get sent to jail after landing on "Go to Jail"?
  2. Does the player receive $200 if "Go" is between the current space and jail?
  3. Is $50 correctly deducted if the player has more than $50?
  4. Is the player out of the game if he or she has less than $50?

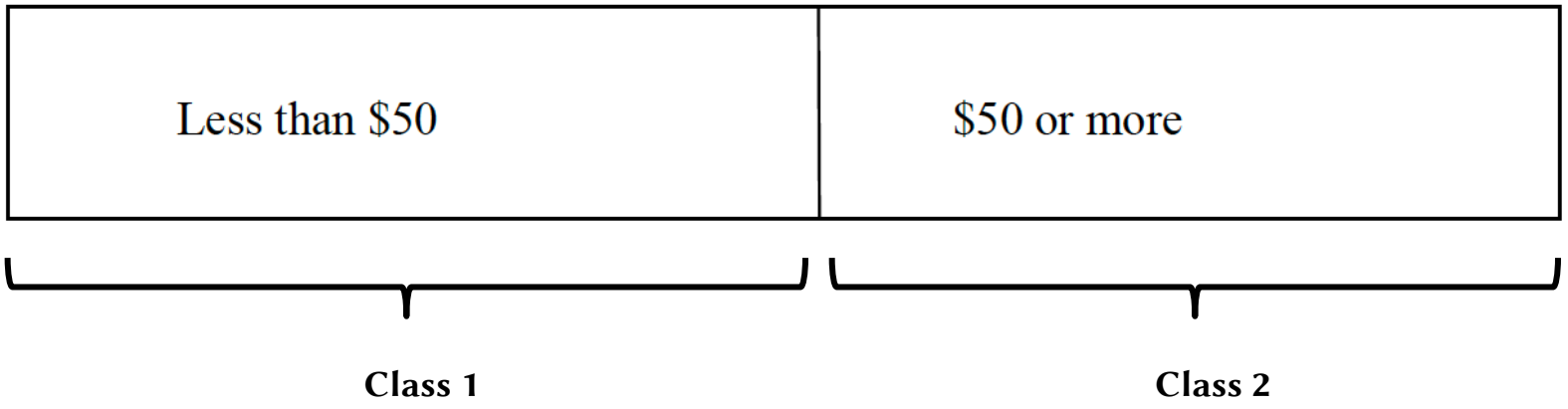
(Partial) Introduction to Software Engineering Practices and Methods by Dr. Laurie Williams NCSU CSC326 Course Pack 2010-2011 (Seventh) Edition https://sdc.csc.ncsu.edu/files/resources/williams-software-engineering-2011.pdf

Equivalence Partitioning

- To keep down our testing costs, we don't want to write several test cases that test the same aspect of our program.
- **Equivalence partitioning** is a strategy that can be used to reduce the number of test cases that need to be developed.
- **Equivalence partitioning** divides the input domain of a program into classes.
  - For each of these equivalence classes, the set of data should be treated the same by the module under test and should produce the same answer.
- **Once you have identified these partitions, create test cases for each partition.**

Equivalence Partitioning - Example

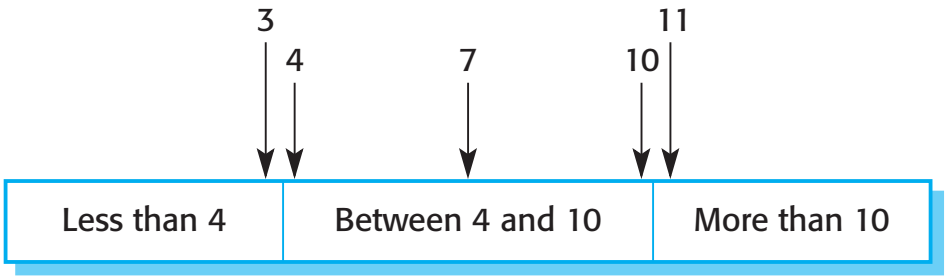
- For example, for tests of "**Go to Jail**" the most important thing is whether the player has enough money to pay the $50 fine. Therefore, the two equivalence classes can be partitioned

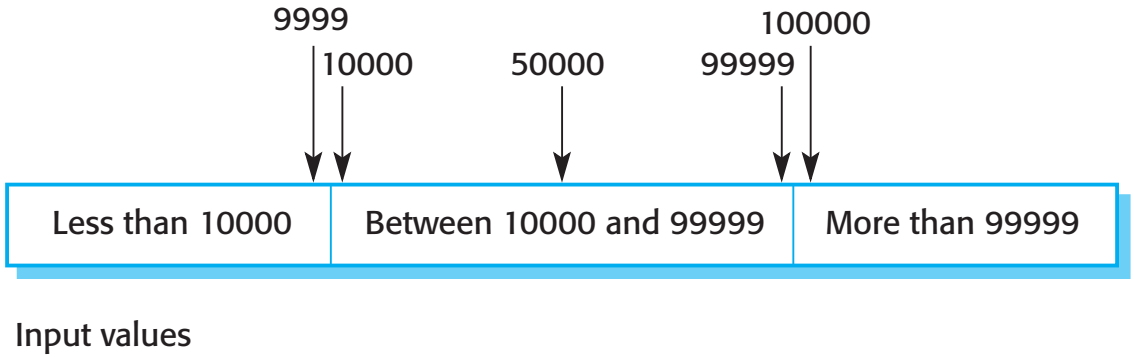


(Partial) Introduction to Software Engineering Practices and Methods by Dr. Laurie Williams NCSU CSC326 Course Pack 2010-2011 (Seventh) Edition
https://sdc.csc.ncsu.edu/files/resources/williams-software-engineering-2011.pdf

Equivalence partitions

A good rule of thumb for test-case selection is to choose test cases on the boundaries of the partitions, plus cases close to the midpoint of the partition.





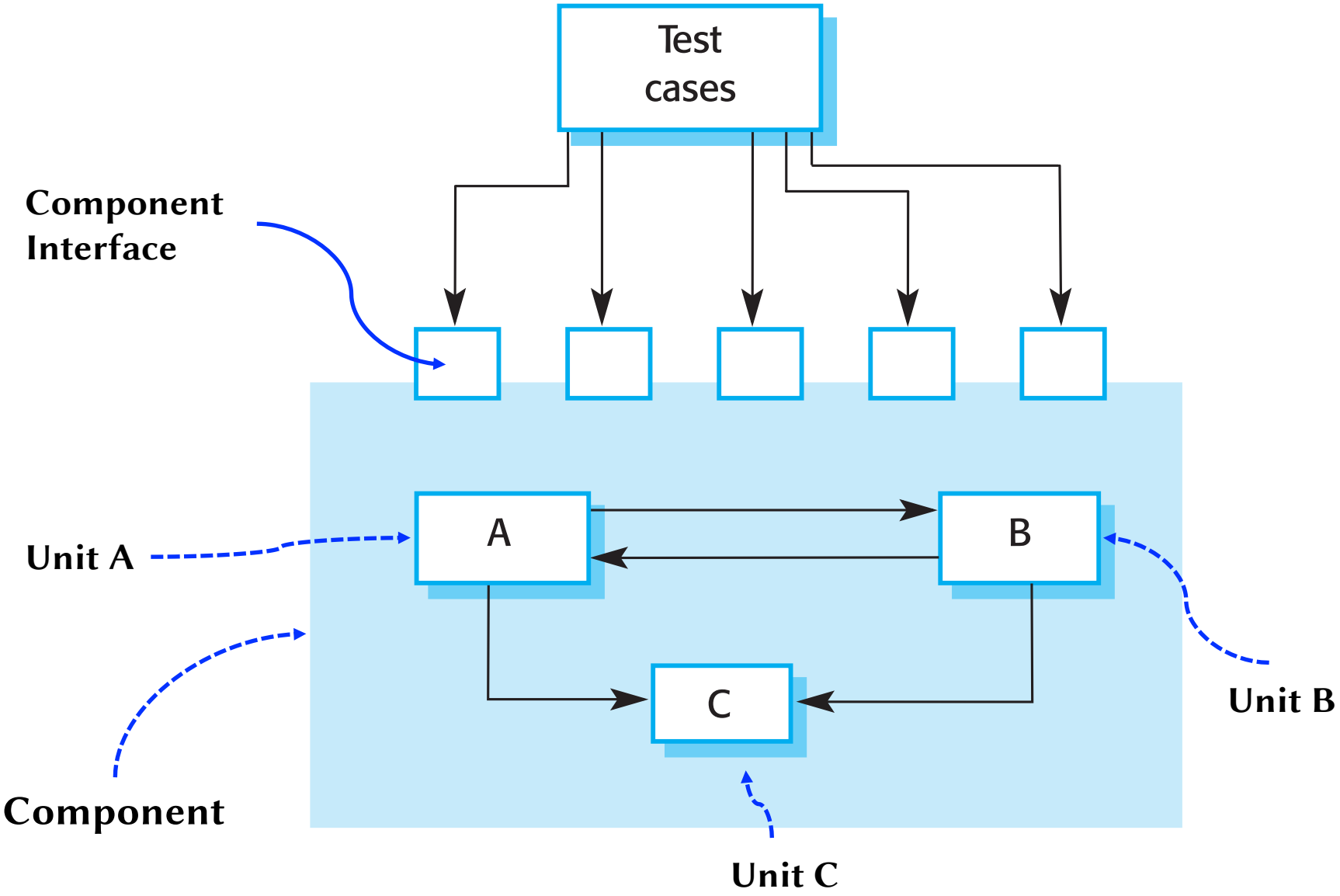
General testing guidelines

- 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.
- 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



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 testing vs component testing

- System testing obviously overlaps with component testing, but there are two important differences:
  - 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.

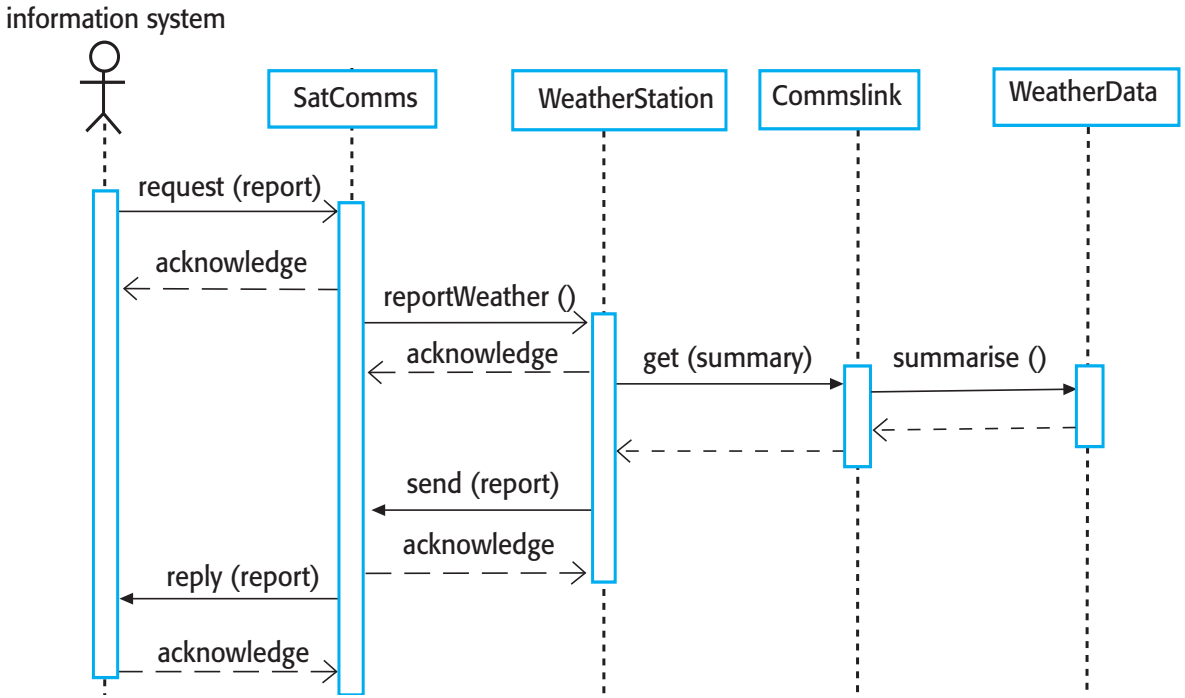
Use-case testing

- 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

- You can use this diagram to identify operations that will be tested and to help design the test cases to execute the tests.
  - Issuing a request for a report will result in the execution of the following thread of methods:

SatComms:request → WeatherStation:reportWeather → Commslink:Get(summary)→ WeatherData:summarize



References

- Ian Sommerville, Software Engineering, 10th Edition, 2015.
  - https://iansommerville.com/software-engineering-book/slides/
- (Partial) Introduction to Software Engineering Practices and Methods by Dr. Laurie Williams NCSU CSC326 Course Pack 2010-2011 (Seventh) Edition
  - https://sdc.csc.ncsu.edu/files/resources/williams-software-engineering-2011.pdf
- B. Beizer, Software Testing Techniques. London: International Thompson Computer Press, 1990.
- Roger, S. Pressman, and R. Maxin Bruce. Software engineering: a practitioner's approach. McGraw-Hill Education, 2015.
- B. Beizer, Black Box Testing. New York: John Wiley & Sons, Inc., 1995.

Thanks!

Mojtaba Shahin

mojtaba.shahin@rmit.edu.au