L20 - Software testing overview and black box testing

CS3028 - Principles of Software Engineering

Ernesto Compatangelo

Department of Computing Science

UNIVERSITY OF ABERDEEN

20.1 Reminding past issues and mapping them to current topics

Where are we now?

⇒ · · · · · ·

⇒ Elaboration

⇒ Construction (third UP phase)

⇒ Software Quality Control

⇒ Software Testing

⇒ · · · · · ·

Software testability - what are its properties?

- **Operability** - can be executed for testing
- **Observability** - test results are *visible*
- **Controllability** - exhaustive execution is possible through some combination of input (may not actually carry it out)
- **Decomposability** - is composed of independently testable modules
- **Simplicity** - testing suite built from simpler architecture /code
- **Stability** - subject to infrequent changes that do not affect testing
- **Understandability** - high quality software documentation is available

Test plan components

- Devised **before** any code is actually written
- Test items (classes, packages, components, subsystems, interfaces. . .)
- Features to be tested
- Features not to be tested
- Specific techniques and tools to be used
- Criteria for item pass/fail
- Deliverables (test cases specification, logs)
- Managerial organisation (people, tasks, schedules, contingency plans)

Test cases

- Software tests are made up of test cases
- Each test case defines:
  - sample input set
  - execution conditions
  - expected result for the given set
- A good test case has high probability of finding an as-yet undiscovered error

Testing techniques

- **Black-box** testing
  - Focuses on relationship between inputs and outputs
  - Ignores the internals of the system
  - Uses a small (but relevant) subset of all possible inputs
- **White-box** (glass-box) testing
  - Focuses on the internal structure of the system
  - Aims to test every statement at least once
  - Uses program graphs to derive linearly independent test cases

The staged approach to testing

- **Unit testing**
  - Tests individual units (packages, classes)
  - Largely uses white-box techniques (actual code matters)
- **Integration testing**
  - Tests integrated units/components or the complete system
  - Uses a combination of white-box and black-box techniques

Testing taxonomy

- **Regressive** testing focuses on retesting after changes are made
- **Progressive** testing focuses on new features after regressive testing of previous features
- **System** testing
- **Acceptance** testing (black box, performed on the system as a whole)
- **Performance** testing
- **Portability** testing
- **Safety** testing (focuses on whether safety-critical software under test fulfills corresponding safety specifications)
- **Alpha and Beta** testing

20.2 Black-box testing: a perspective

Black-box testing

- B-B tests focuses on **designing test cases without referring to code**
- B-B testing focuses on the **information domain**
- **Derived test cases are designed from requirements**; for this reason they are often called functional tests or behavioural tests
- Objective is to derive representative rather than exhaustive test cases
- Derived test cases should uncover **classes of errors** rather than individual errors
- Should be used in conjunction with other complementary testing techniques such as white-box techniques
- Should fit into the overall testing strategy

Classes of errors

Black-box testing uncovers the following classes of errors:

- Incorrect or missing functions
- Interface errors
- Errors in data structures or external data base access
- Performance errors
- Initialisation and termination errors

20.3 Black-box testing techniques

Black-box testing techniques

- Equivalent partitioning
- Boundary value analysis
- Graph-based methods, *i.e.*
  - Control flow testing
  - Data flow testing
  - Transaction flow testing
  - Finite-state machine testing
- To understand the above techniques, we need to review some notions about algebraic relations

Black-box testing technique: equivalence partitioning

- Partitions/divides input domain into equivalence classes
- Equivalence partitioning generates representative test cases for each equivalence class
- **Example**: consider a function that finds the square root of a real number — two input equivalence classes exists: (i) The class of numbers ≥ 0, and (ii) the class of numbers < 0

Equivalence partitioning example

- Consider a program that asks for student marks as a percentage (%)
- There are three equivalence classes, namely:
  - Less than 0
  - 0 to 100
  - Greater than 100
- For each of these classes we generate a test case:

Input	**Expected result**
-50	should print an error message and prompt for the correct input
50	should accept the input and proceed with the computation
150	should print an error message and prompt for the correct input

Black-box testing technique: boundary-value analysis

- Errors are known to occur at boundaries of an input domain
- This technique is complementary to equivalence partitioning
- Test cases are selected from boundaries of equivalence classes

Boundary-value analysis example

- Consider a program that asks for student marks as a percentage (%)
- Boundary values are 0 and 100
- Test cases for the lower bound, 0 (similar results hold for the higher bound, 100)

Input	**Expected result**
-1	should print an error message and prompt for the correct input
0	should accept the input and proceed with the computation
1	should accept the input and proceed with thecom-putation

Further hints

- Use random inputs
- Use impossible values
- Use project/programmer-specific test cases
- Imagine writing code that maximises errors rather than minimising them:
  - Identify those scenarios where it is easy to introduce errors and use several methods on them
  - Identify those scenarios where it is difficult to introduce errors and use common sense...

20.4 Preparing for the topic ahead

Next week. . .

More on testing

More specifically, we will focus on:

- White-box testing
- Overall testing strategies (arch-important!)
- Peculiarities of testing Object-Oriented systems and GUIs