Slide 18: Coding

Made by Md. Mehedi Hasan Rafy

Testing

Testing means giving a program different test inputs and checking if it behaves as expected.

- ***Failure*** – when the program shows wrong behavior. But an error in the code doesn't always cause a failure.
- ***Test case*** – a triplet [I, S, O] is called a test case where —
  - I → the data input of the system.
  - S → the state of the system at which the data is input.
  - O → the expected output of the system.
- ***Test suite*** – the full collection of test cases used to test the software.

Aim of Testing

The goal is to find as many defects as possible in the software. Complete testing is impossible, because input possibilities are too large. Even with this limitation, testing is valuable since it catches many defects. Testing helps reduce defects and increase user confidence in the software.

Verification vs Validation

Verification – is the process of determining whether the output of one phase of software conforms to that of its previous phase.

- Checks if each development phase matches the previous phase.
- Goal: catch errors early (within phases).

Validation – is the process of determining a fully developed system conforms to its requirements specification.

- Checks if the final system meets user requirements.
- Goal: ensure the end product is error-free from the user's perspective.

Design of Test Cases

- Exhaustive testing is impossible because domain of input value is too large.
- A good test suite should be small but effective — it should detect different errors, not just repeat the same ones.
- Random test cases don't guarantee more error detection; careful design is better.
- Hence, the number of random test cases in a test suite is not an indication of the effectiveness of the testing.
- Example: A wrong code for finding greater of two integer x and y:

```
if (x > y)
      max  = x;
else
      max = x;
```

  max(x, y) → a small, well-chosen test suite like {{x = 3, y = 2}, {x = 2, y = 3}} can catch the error, while a larger random one {{x = 3, y = 2}, {x = 5, y = 0}, {x = -2, y = -3}} might miss it.

- ***Optimal test suite****:* every test case should aim to catch a different possible error.

Functional vs Structural Testing

- **Black-box (Functional)**: design tests from input-output behavior, without looking at code.
- **White-box (Structural)**: design tests by looking inside the code and using its internal structure.

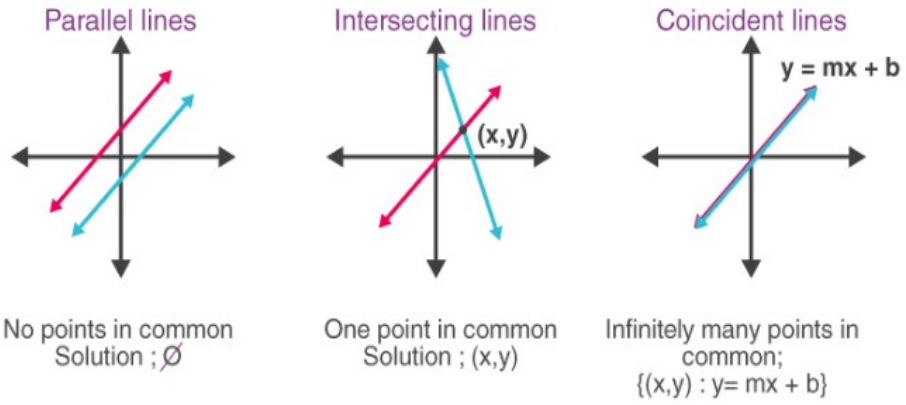
Black-Box Testing

- Focuses only on inputs and outputs, ignores internal code.
- Two main design methods:
  1. Equivalence Class Partitioning
  2. Boundary Value Analysis.

Equivalence Class Partitioning (ECP)

- Input domain is divided into a set of equivalence classes where program behavior is similar.
- So, testing one value from each class is enough.
- ***Guidelines:***
  - For ranges → 1 valid + 2 invalid equivalence classes should be defined.
  - For discrete values → 1 valid + 1 invalid equivalence classes should be defined.
- ***Example 01:*** Square root (0 – 5000)

  Classes = {negative, 0 – 5000, > 5000} → Test set = {-5, 500, 6000}.



- ***Example 02:*** Intersection point of two lines of the form y = mx + c. Input two integer pairs (m1, c1) and (m2, c2) are read.

  Classes = {parallel (m1 = m2, c1 != c2), intersecting (m1, != m2), coincident(m1 = m2, c1 = c2} → Test suite = {{(2, 2), (2, 5)}, {(5, 5), (7, 7)}, {10, 10}, {20, 20}}.

Boundary Value Analysis (BVA)

- Errors often happen at boundaries of different equivalence classes of inputs (e.g., < vs <=).
- Test cases chosen at edges of ranges.
- ***Example 01:*** Square root (0–5000) → Test {0, -1, 5000, 5001}.

AGE Enter Age *Accepts value 18 to 56

BOUNDARY VALUE ANALYSIS		
Invalid (min -1)	**Valid (min, +min, -max, max)**	**Invalid (max +1)**
17	18, 19, 55, 56	57

- ***Example 02:*** Age 18–56 → Valid: {18, 19, 55, 56}, Invalid: {17, 57}.