Testing

Testing vs Debugging

- Testing is the process of detecting defects
- Debugging is the process of diagnosing and correcting defects that have already been identified
- The two are not the same!

Test Classification

- 1. Input validation
  - Provides robustness in the face of unexpected input
- 2. Boundary cases
  - Ensures we have the boundaries handled correctly
- 3. Control flow
  - Look for requirement operation (good and bad outcomes)
- 4. Data flow
  - Look for the ordering of steps (good and bad orderings)

Structured Approach to Test Cases

- Look to validate your inputs
  - Existence and format, unless all calls are coming from within your own code
  - Too much data, too little data, wrong size of data, uninitialized data
  - Nominal (normal or middle-of-the-road) data

- Look to boundary cases
  - Edge cases of the input
    - Include minimum and maximum normal configurations
  - Edge cases of the output format or overall outcome
  - Boundary cases in your conditional and loop statements (i.e., control flow)
    - Also look for compound boundary cases that involve several variables

Structured Approach to Test Cases

- Control-flow approaches (white box testing)
  - Structured basis testing
    - Exercise each statement at least once
  - Path testing
    - Test each path through the code at least once

Input validation – parameter to method

- Boolean testMe(String value)

  ok test: value could be object is null or not filled (just initialized):
  testMe(null);
  testMe(“”);

  bad test: send an integer as value:
  testMe(10);

- Boolean testMe2 (int value2)

Input validation – user interface

- User input could be a string , integer, ..

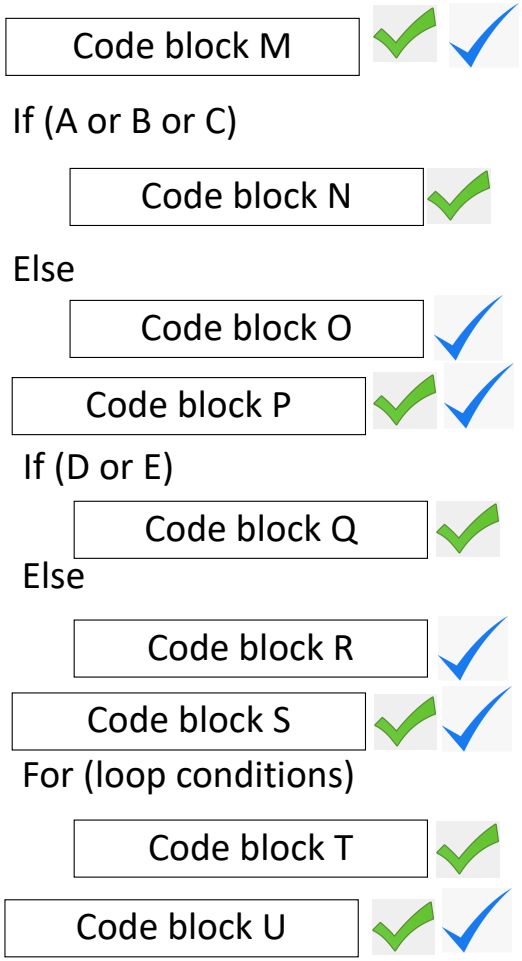
Boundary cases

- Look to any special case that is separated from other cases by some continuous parameter boundaries or parameters within a range
- Test each side of those boundaries
- Example:
  - Continuous range: integer temperature parameter and we are checking for the transition state for water
    - 0 degrees for freezing
      - Check temperatures 0 (freeze) and 1 (barely not freezing)
    - 100 degrees for boiling
      - Check temperatures 99 (not steam yet) and 100 (now steam)
  - Parameter range: Sport registration for kids aged 12-15
    - Check kids of ages 11, 12, 15, and 16

Control flow

- Look to different functionality in the requirements
  - Good outcome
  - Error handling
- Look to different paths through your code

Structured Basis Testing



First test case code:

code block M
code block N
code block P
code block Q
code block S
code block T
code block U

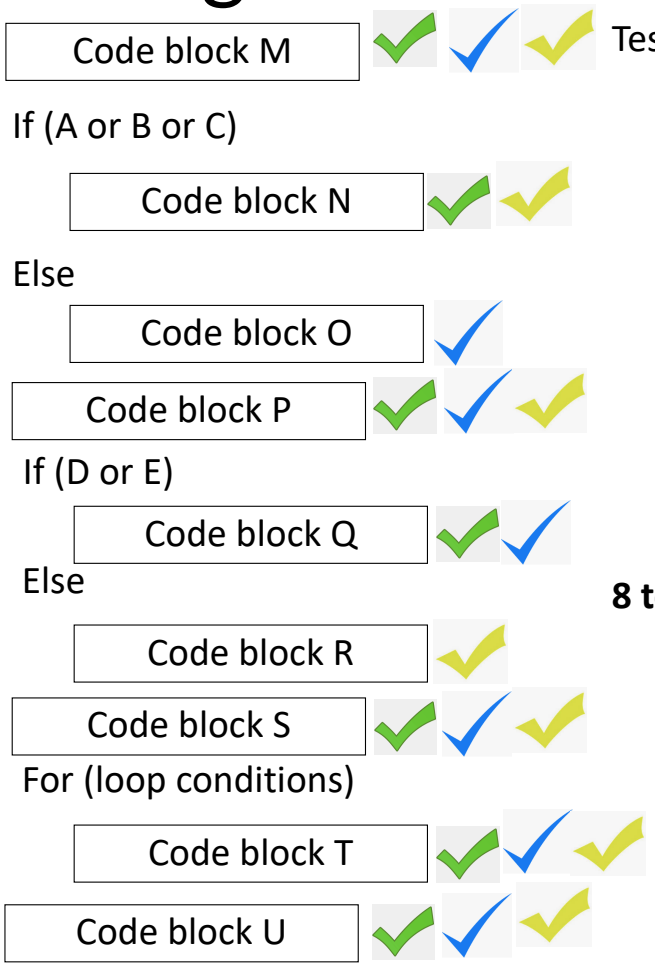
Second test case code:

code block M
code block O
code block P
code block R
code block S
code block U

2 test cases exercise every line of code

Code blocks have no execution branches in them.

Path Testing



Test cases:

M, N, P, Q, S, T, U

M, O, P, Q, S, T, U

M, N, P, R, S, T, U

M, O, P, R, S, T, U

M, N, P, Q, S, U

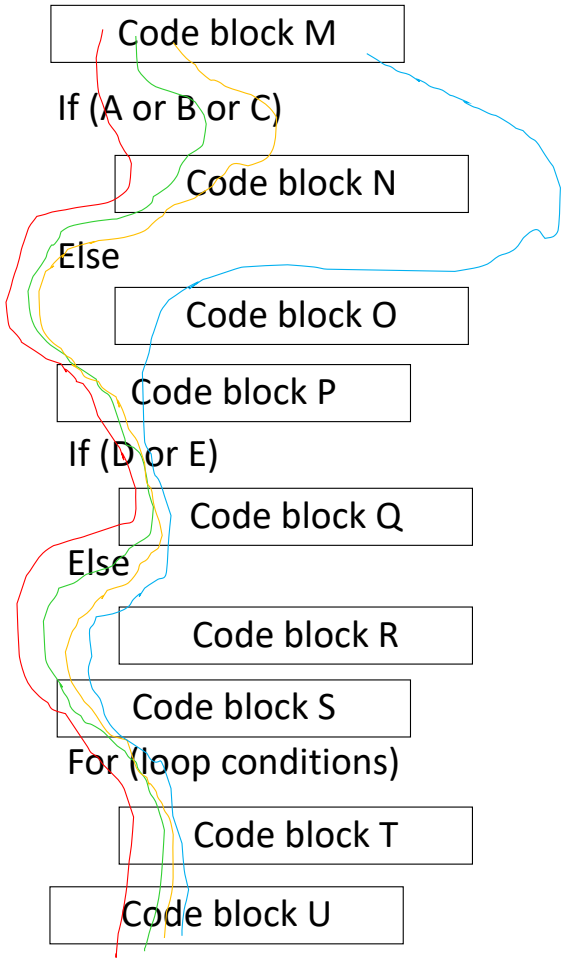
M, O, P, Q, S, U

M, N, P, R, S, U

M, O, P, R, S, U

8 test cases to cover all block paths

Path and Expression Testing



Test cases:

M, A, N, P, D, Q, S, T, U
M, B, N, P, D, Q, S, T, U
M, C, N, P, D, Q, S, T, U
M, A, N, P, E, Q, S, T, U
M, B, N, P, E, Q, S, T, U
M, C, N, P, E, Q, S, T, U
M, O, P, D, Q, S, T, U
M, O, P, E, Q, S, T, U
M, A, N, P, R, S, T, U
M, B, N, P, R, S, T, U
M, B, N, P, R, S, T, U
M, O, P, R, S, T, U

Repeat again to not do the for loop body

24 test cases to cover all block and expression paths

Apply logic to eliminate non-meaningful paths

Structured Approach to Test Cases

- Data-flow approach (transparent or white box testing)
  - Test for paths in the data that force unexpected uses of data
    - Define a variable then never use it in the method
    - Define a variable then free it before the method ends
    - Undefine a variable twice
    - Using a variable before having it be initialized

Data Flow Testing Example

- You have a spreadsheet class where you normally expect to:
  - Create the object
  - Read data in from a file
  - Add cells
  - Calculate values
  - Write the object to a file
  - Destroy the object

Data Flow Testing Example

- Look at variants to that flow:

Create the object
Add cells

Create the object
Destroy the object
Destroy the object

Create the object
Write the object to a file

Create the object
Add cells
Calculate values
Read data from a file

Create the object
Destroy the object

Create the object
Calculate values
Add cells

Directed Approach to Test Cases

- Concentrate on picking test cases that tell you something different about the outcome
  - Aim for non-overlapping tests if you can

- Error-guessing
  - Create test cases about where your program might have the most errors based on your past experience
  - Use your knowledge of which code is the least familiar, lowest confidence, most complex, or most commonly done poorly in the organization

Managing Your Tests

- Divide your test cases into equivalence classes
  - Have one test per class
  - Eg. searching an array of 10 items for the 4th, 5th, and 6th items is likely to look the same, so just search for one
- Use test cases that make hand-checks convenient
- Some people prefer large, all-encompassing tests while other people prefer many small and independent tests
  - **All-encompassing:** lots tested at one time, but if one item fails then there is less information on the defect and you don't get to try the remaining test cases
  - **Small:** many tests to run, but a failure gives you more precise information and lets other tests continue. A fundamental error can lead to many failures from one bug.

JUnit

- A framework that lets you have tests without writing your own main() method to run them

- Detects and reports which test pass and which tests fail, when you run the set of tests

JUnit

- Tests are put in a separate testing folder
- Tests begin with the designation @Test

- Write methods as usual:
  - Check conditions with methods that take the expected result, the computed result and an error message:
    - assertTrue
    - assertFalse
    - assertEquals