Western

UNIVERSITY · CANADA

Topic 12

Functional Testing

Computer Science 2212b
Introduction to Software Engineering
Winter 2014

Jeff Shantz
jeff@csd.uwo

Functional Testing: Motivation

With structural testing, we can guarantee all code has been executed with tests

- No guarantee that we've tested all of our **requirements**
  - We might not have implemented some requirements in code
  - No code = nothing to cover!
- Hence, we can achieve 100% coverage *without* actually satisfying the requirements of the program!

Functional Testing: Motivation

We need to test based on our code (structural testing), but we also need to test based on our *requirements*

- This is **functional testing**
- We will need to also consider *unstated requirements* like error handling
- Ideally, we should go *systematically* through all stated and unstated requirements and develop tests for them
  - Standard methods of doing so exist

Functional Testing Example: Triangle Analyzer

Requirements

- Program prompts user for input
- User enters three real numbers, separated by commas
  - e.g. `2.5, 6, 6.5`
- Program responds with:
  - **Equilateral**: if there is an equilateral triangle with those sides
  - **Isosceles**: likewise
  - **Scalene**: likewise
  - **Not a triangle**: if there is no valid triangle with those side lengths (e.g. `3, 4, 1000` since 3 + 4 < 1000 [triangle inequality)

Functional Testing Example: Triangle Analyzer

What tests cases will we use?

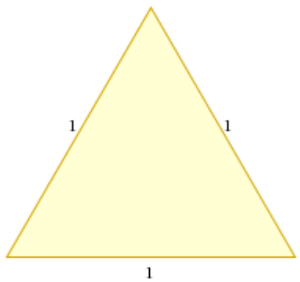
- These will be ***functional*** (black-box) test cases, since we are working only from the requirements

Think of some test cases you might use to test the analyzer

Functional Testing Example: Triangle Analyzer

Test Case 1: Equilateral triangle

`1,1,1`

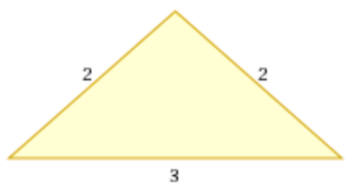


Test Case 3: Scalene triangle

`5,2,4`

Test Case 2: Isosceles triangle

`3,2,2`



Test Case 4: Not a triangle

`7,2,4`

Probably want to test with all possible combinations of each test as well
e.g. `2,4,5`; `2,5,4`; `4,2,5`; `4,5,2`; `5,2,4`; `5,4,2`

Functional Testing Example: Triangle Analyzer

Need to consider unstated requirements as well:

- Invalid input characters: `1,a,2`
- Invalid number of inputs `1,2`
- Invalid input format: `1  2  3`
- etc.

Functional Testing: Basic Methods

How to choose good functional test cases? We will look at each of the following methods:

- Test all possible outputs
- Test both valid and invalid inputs
- Test around boundaries
- Test extreme values
- Test input syntax
- Guess at possible errors

Functional Testing: Basic Methods

These techniques will generate *lots* of test cases. In general,

- A test case may be one input, or a *sequence* of inputs (depending on the program)
- We will probably have more test cases for *erroneous* input than for valid input

Test All Possible Outputs

For each possible output we know the program could produce:

- Write a test case that will produce that kind of output
- Our textbook calls these *equivalence classes* (chapter 10)

Examples:

- Triangle analyzer
  - One case for each of Equilateral, Isosceles, Scalene, and Not a Triangle
- Parking garage simulator
  - A case for parking a car; one for retrieving a car
  - Should also have cases for *garage full* and *no such car*

Test Valid and Invalid Inputs

Often, an individual input `x` to a program has:

- A valid range `x >= 0` `1 <= x <= 12`
- A valid set of values
  - `x` is a string of alphanumeric characters
  - `x ∈ {red, green, blue}`

Inputs outside these ranges/sets are invalid. For each input to the program:

- Test at least one *valid* value
- Test at least one *invalid* value
- Test invalid values off either end of the value range (-1 and 13)

Test Valid and Invalid Inputs

Individual inputs to a program can include:

- Things types in to a console or a GUI control
- Command-line options
- Values in configuration files
- etc.

Examples of invalid inputs:

- Triangle analyzer: Test `-1` or `Z` as the length of a side
- Day planner program: Enter `Jqx` as the name of a month

Test Around Boundaries

Failures often occur close to *boundaries*

- Boundaries between different kinds of output
- Boundaries between valid and invalid inputs

Such failures are often due to faults such as

- Errors in arithmetic
- Using <= instead of <
- Not initializing a loop properly

Therefore, we should test at or near boundaries

Test Around Boundaries

Triangle analyzer

- Test case: `2, 2, 4.00001` (almost, but not a triangle)
- Test case: `2, 2, 4` (right on boundary)

Pop machine dispenser software:

- If the user has exactly enough money to buy a can, it should be dispensed
- However, if the program contains a test like:

```
if (balance > cost)
```

  it will not be allowed (since the test should be >=)
- Thus, need to test this situation

Test Extreme Values

Often, software does not handle *very large* or *very small* values correctly due to things like

- Buffer or arithmetic overflows
- Mistaken assumptions that a string will be non-empty

Therefore, these sorts of values may be a way to break a program

Test Extreme Values

Examples:

- With just about any program that accepts user input:
  - Empty strings
  - Very long strings
- Triangle analyzer:
- `4321432134, 543234344, 6566765888` (very large)
- `0.00000003, 0.00000008, 0.00000005` (very small)

Test Input Syntax

What happens if the user enters something incorrectly?

- Something omitted: 5, 12   13   (comma missing)
- Too few/many values: 5, 12 or 5, 12, 13, 20
- Invalid tokens: 5, 12, qwe!

In these situations

- Program shouldn't just crash
- Shouldn't give an uninformative error message
- Shouldn't just accept and process as if correct
  - This could be even worse than crashing altogether
- Should give an informative message, recover from the error

Guess At Faults

Finally, use intuition to think of how a program *might* be wrong:

- Might be better to get person **A** to think of possible faults in person **B**'s code

Example with the triangle analyzer:

- To see whether a triangle is isosceles, the code must test all three distinct pairs amongst the three numbers for equality:
  - What if not all three pairs have been tested by the code?
  - Thus, test all of `2,2,3`; `2,3,2`; and `3,2,2`

This method is not based on systemic study

- More like educated guesses

Exercise

Assume you are writing a series of JUnit tests to test the method with the following signature:

```
public int maxOfThreeNumbers(int n1, int n2, int n3)
```

Think of some test cases you would write to thoroughly test it.

Description	N1	N2	N3	Expected Output

Functional vs. Structural Testing

Functional (black-box) testing

- **Advantages**
  - Ensures program meets requirements
  - Test boundaries, etc., explicitly
- **Disadvantages**
  - Cannot test undocumented features
  - May not test hidden implementation details thoroughly

Functional vs. Structural Testing

Structural (white-box) testing

- **Advantages**
  - Tests all code and implementation details
  - Coverage metrics can give us an idea of the extent to which our code is tested
- **Disadvantages**
  - Cannot test whether all desired features are implemented
  - Metrics are not a panacea
    - We saw how 100% statement coverage essentially means nothing in relation to code quality / correctness

Functional vs. Structural Testing

In practice, we'll use a combination of both to ensure that both our code and requirements are tested.

1878

Western

UNIVERSITY · CANADA