Testing and Test cases

Murali P

Today's Agenda

- Correctness Issues
  - Motivation ✓
  - Correctness
    - Modules ✓
    - Contracts ✓
    - Invariants ✓
  - **Tests and Test Cases**

Why Testing?

- Verify fidelity of implementation.
  - Does the implementation meet design specifications?
- Verify robustness of design.
  - Not everything is specified in design.
  - Not everything about design is provably correct.
- Identify and understand exceptions.

Testing and Design

- Identification of test cases
  - Should be done during or before Design Phase.
- High level test cases
  - Based on problem specification.
- Low level test cases
  - Based on solution (design)

Test Case Identification [1]

- E.g. Binary Search

  Problem Specification:

  Find the position of an element x

  in an ordered list A

  (of finite length, say N).

Binary Search Algorithm

```
low = 1;  high = N;
while (low < =high) do
   mid = (low + high) /2;
   if (A[mid] = = x)  return x;
   else if (A[mid] < x) low = mid +1;
   else high = mid – 1;
endwhile;
   return Not_Found;
```

Test Case Identification [2]

- What is a test case?
  - Any input: e.g., x = 3, A = {2, 3, 5, 7, 18}, N=5
- How many test cases are there ? (for a fixed N)
  - If each location of A can take k possible values, A has k^N possible values.
  - So, there are at least k^{N+1} possible values

Test Case Identification [3]

- How many cases are enough?
  - Ideally all cases must be covered.
  - Similar cases may be grouped into classes.

  e.g. x and A such that x < A{1}
- Which cases are to be tested?
  - One sample case per class must be tested.

Test Case Identification [4]

- Will the algorithm handle one element array?
  - e.g. x = 0 and A = {1}
  - e.g. x = 1 and A = {1}
  - e.g. x = 2 and A = {1}
- Will it handle the case where all array values are equal?
  - e.g. x = 5 and A = {5,5,5,5,5,5,5}
- What if the value is the first element of the array?
  - e.g. x = 5 and A = {5, 7, 9, 12}

Test Case Identification [5]

- What if the value is the last element of the array?
  - e.g. x = 12 and A = {5, 7, 9, 11, 12}
- What if the value is not found in the array?
  - the value is within the range:

    e.g. x = 8 and A = {5, 7, 9, 12}
  - the value is below the range:

    e.g. x = 4 and A = {5, 7, 9, 12}
  - the value is above the range:

    e.g. x = 13 and A = {5, 7, 9, 12}

Test Case Identification [6]

- All of the above questions are answerable at design time.
  - Testing ensures “fidelity” of implementation.
- Other questions?
  - Is the design complete?
  - Is the design proven?

Test Case Identification [7]

- Is the solution dependent on even or odd locations?
  - Needed because: (low + high)/2;
- Is the solution affected by large values of low OR high?
  - Suppose $N=2^{19}$ in LC-3. What happens when low > 2^8

Test Case Identification [8]

- E.g. quad

  Problem Specification:

  Given a, b, and c, solve quadratic equation:

  $a*x^2 + b*x + c = 0$

Test Case Identification [9]

- Solution:

```
// Pre-condition:   b*b > 4*a*c
#include <assert.h>
function quad(float a, float b, float c, boolean sign)  returns float
 {
 assert(a!=0 || b!=0);
 if(a==0)  return – c/b;
 int disc = b*b – 4*a*c;
 If(disc == 0) return –b/2*a;
 assert(disc > 0);
 if (sign) return (-b + sqrt(disc)) / (2 *a);
 else return (-b – sqrt(disc)) / (2*a);
 }
```

Test Case Identification [10]

- Does the algorithm handle positive values for a,b, and c?
- Does the algorithm handle negative values for a,b, and c?
- What if a or b or c is zero?
- What if all of them are zeroes?
- Are calls to sqrt correct?
- The key point here is: Are we checking all rare cases.
- Do some of these rare cases require a re-visit of our code?

Test Case Identification [11]

- Open issues in design:
  - Will a high value of b cause an overflow?
  - Will high values of a and c cause an overflow?
  - Will there be underflow?
    - b* b – 4 * a *c
    - (-b + sqrt(disc)) / (2*a)

sqrt function

```
/*
Pre condition:
    m > 0
Post condition:
    R = sqrt(m) (with a relative
    error less than 1%)
    Err < 0.01
*/
```

```
float sqrt (float m)
{
    float R=m/2;
    float Err= (R*R – m)/m;
    while (Err >= 0.01)
    {
        R= (R + m/R)/2;
        Err=(R*R – m)/m;
    }
    return R;
}
```

Test Case Identification [12]

- function sqrt(float x) returns float
- Will sqrt be able to handle all positive values?
- Will there be any underflows?

Test Case Identification [13]

- Exceptions
  - What happens when contracts are violated?

    e.g. sqrt(-1)

    e.g. quad(2, 1, 2, true)

  - Is graceful termination guaranteed?

Test Case Identification [14]

- Exercise:
  - Identify test cases for function pow2(x,y) that computes x^y when y is a power of 2.
  - Identify test cases for function pow(x,y) that computes x^y.
  - Identify exception (crash) scenarios!

Common Hour Problem 1

- A year is called leap year if it is (divisible by 400) or (divisible by 4 but not divisible by 100). Consider a function *isLeapYear* which takes year as parameter and returns a Boolean value true or false depending upon whether the year passed to it is a leap year or not.
- You are required to write the sufficient test cases to test this function.

CH2.1 contd

- Precondition: Year must be >0
- Postcondition: Returned value is TRUE or FALSE depending upon whether year is a leap year or not.

- Conditions for test cases classes:
- divisible by 400 e.g. 2000 (TRUE)
- not divisible by 400
  - divisible by 100 e.g. 2100 (FALSE)
  - not divisible by 100
    - divisible by 4 e.g. 1996 (TRUE)
    - not divisible by 4 e.g. 1997/8/9 (FALSE)

CH2.1 Test Cases

1. Checking for precondition violation
Input: 0 or -100
Output: Assertion `year>0' failed.
Aborted
2. Checking a sample input for divisibility by 400 (Test case 1)
Input: 2000
Output: TRUE
3. Checking a sample input which is not divisible by 400 but divisible by 100
Input: 2100 (Test case 2a)
Output: FALSE
4. Checking another sample input for divisibility by 4 but not divisible by 100
Input: 2004 (Test case 2b(i))
Output: TRUE
5. Checking a sample input which is neither divisible by 4 nor by 100 (Test case 2b(ii))
Input: 1997
Output: FALSE