Software Testing
CS II: Data Structures & Abstraction

Jonathan I. Maletic

Kent State University

Spring 2013

What is Software Testing?

- The objective of software testing is to find errors.
- An error is incorrect output for a given input.
- Errors are caused by faults in your program.
- A fault (bug) is an incorrect part of your code.
  - Missing condition
  - Incorrect stopping condition

How to Find the Faults?

- Test cases are developed to exercise your program with the goal of uncovering faults
  - This will lead you to the error in the code
- The best test case is one that has a high probability to uncover a fault

- Start by testing each method (unit tests)
- Then each class in full (module tests)
- Then the whole program (system tests)

Information Needed

- A method/function is defined by an input/output specification (I/O spec).
  - The pre and post conditions describe the I/O spec

- A method/function is also defined by its implementation details
  - For-loop vs while loop vs recursive

Black Box vs Glass Box

- Black box testing uses only the I/O spec to develop test cases

- Glass box uses only the implementation details to develop test cases

- Both types of information are necessary to develop a good set of test cases for a method/ function

How Many Tests Cases are There?

- Most functions have a very large (i.e., infinite) number of possible inputs and outputs

- Do you need to test all of these to be satisfied your function behaves correctly? NO!

- Again, the best test case is one that has a high probability in uncovering a fault.

Pairing Down Test Cases

- Can take advantage of symmetries, equivalencies, and independencies in the data to reduce the number of necessary test cases.
  - Equivalence Testing
  - Boundary Value Analysis
- Determine the ranges of the working system
- Develop equivalence classes of test cases (I/O)
- Examine the boundaries of these classes carefully

Equivalence Partitioning

- Input data and output results often fall into different classes where all members of a class are related
- Each of these classes is an equivalence partition where the program behaves in an equivalent way for each class member
- Test cases should be chosen from each of the different partition

Boundary Value Analysis

- Partition system inputs and outputs into "equivalence sets"
  - If input is a 5-digit integer between 10,000 and 99,999, equivalence partitions are < 10,000,
    10,000 - 99, 999 and > 10, 000
- Choose test cases at the boundary of these sets
  - 00000, 09999, 10000, 99999, 10001

Example

- Search an array A of size N for a key K, return the location of first occurrence
- N: 0, 1, [2, max-1], max
- K: negative, zero, positive
- A: contains K in position 0, 1, [2, max-1], max
- A: does not contain K
- A: contains multiple K

Test Driven Development

- Testing is an integral part of development
- To write a method/function:
  1. Determine the I/O spec
  2. Develop test cases
  3. Implement the method
  4. Run the method against the test cases
  5. Fix any faults (debugging and implementation)
  6. Go to 4 (or if serious problems 1)

Unit Test Driver

- Build a program (main) for unit testing
- One test driver (main) for each method
- Test (in this order):
  - Constructors
  - Accessors
  - Copy, assignment, destructor
  - I/O
  - Relational operators
  - Complex operators

Regression Testing

- Each time you add a new method to your class or fix a fault run your test cases (all of them)
- Adding something new or fixing a problem may have side effects
- Re-running your test cases will uncover these problems

Example Test Case

```
#include <cassert>
int main() {
    Set a;
    assert(a.card() == 0);

    Set b(1, 4);
    assert(b.card() == 2);
    assert(b == set(1, 4));
    assert(b != a);

    std::cout << “{1, 4} == ” << b << endl;
    std::cout << “All Tests Completed” << endl;
    return 0;
}
```

Mantra

- Develop test cases before you code!
- Test as you go!
- Test always and often!