Software Testing

Logistics

• Project
  – Part 3 (block) due Sunday, Oct 30
  – Feedback by Monday

Logistics

• Project
  – Part 4 (clock variant) due Sunday, Nov 13th
  • Individual submission
  • Recommended: Submit by Nov 6th
• Scoring
  • Functionality (10)
  • Use of framework (5)
  • Code Style (5)
  • Memory Management (5)
• Questions?

Logistics

• Final exam
  – Good news…bad news
  – Good news
    • Last day of finals, November 18th
  – Bad news
    • 8am-10am
    • Room 01-3338

Plan for this week

• Yesterday: Exam 2 return
• Today: Software Testing
• Thursday: Pointers to functions

Before we begin

• Questions?
Software Testing

• From the software testing FAQ
  – TESTING means "quality control"
  – QUALITY CONTROL measures the quality of a product
  – QUALITY ASSURANCE measures the quality of processes used to create a quality product.

• No such thing as bug-free code!

Software testing

• Some definitions:
  – Error – Improper action of a programmer
  – Fault – The result of an error (improper logic).
  – Failure – Improper action of an executing program due to a fault.

Software Development Cycle

• Gather Requirements
  – Find out what the user needs
• System Analysis
  – Express these needs formally in system terms
• Design
  – Design a high level solution
• Implementation
  – Turn solution into code
• Testing
  – Verify that the solution works
• Maintenance
  – Iterate the cycle

Testing Strategies

1. All tests should be traceable to customer requirements.
2. Tests should be planned long before testing begins.
3. 80% of errors are traceable to 20% of the modules (Pareto Principle)
4. Testing should begin in the small and progress to larger components.
5. Exhaustive testing is NOT possible.
6. Testing is more effective when conducted by an independent party.

Software Testing

• When to test
  – Incrementally during implementation phase
    • Assure each unit or class meets design and functional specs
  – Limited testing of overall system during implementation
  – Formal system test during testing phase (after implementation is complete)
    • Alpha / Beta Testing
    • Tests program requirements
Software Testing

- Levels of testing
  - Unit testing
    - individual classes
  - Integration Testing
    - Assembly of one or more classes
  - System Test
    - System as a whole

- Types of Testing
  - Formal Verification
    - Reduce program to logical assertions and “prove” mathematically that the program is correct
  - Empirical Testing
    - Generate Test cases to see where errors exist.
    - Most testing that you will do will be empirical

Software Testing

- Empirical Testing
  - White Box testing
    - Assumes access to the code
    - Test all program flows
    - Covers all statements and conditions
  - Black Box Testing
    - Assumes no access to code or knowledge of implementation
    - Test cases generated based on requirements
    - Test valid and invalid input
    - Follow the contract

Programming by Contract

SomeClass::someFunction ( AnotherClass *fillMeWithData )
{
// check any preconditions here
preCondition ( fillMeWithData );

// non-NULL check
// do your stuff to add the functionality here ...

// check post conditions
postCondition ( fillMeWithData->hasData() ); // did we do what we said
postCondition ( checkInvariant() ); // class invariant check required
}

Programming by Contract

- Introduced by Bertrand Meyer, the creator of Eiffel.
- Creates a contract between the software developer and software user
  - Every feature, or method, starts with a precondition that must be satisfied by the consumer of the routine.
  - each feature ends with postconditions which the supplier guarantees to be true (if and only if the preconditions were met).
  - each class has an invariant which must be satisfied after any changes to the object represented by the class.

Assertions

- Debugging mechanism to test condition at any point in the code
  - If condition is false, the program aborts and dumps core.
  - Useful for testing preconditions, postconditions and invariant checks.
# Assertions

```cpp
#include <cassert>

void foo (int *p)
{
    // At this point p should not be null
    assert (p != 0);
    ...
}
```

## Questions?

### Unit Testing

- A unit test tests at a “unit” (in C++, class) level.
- Why test classes individually?
  - Limit the scope of testing
  - Easier to generate test cases
  - Bugs found earlier (before integration) are easier to fix.

## Unit Testing

- Black box approach
  - Must rely on functional specs and contracts
  - Supply inputs, check outputs.
    - Call methods with well chosen parameters
    - Call methods in various orders
    - Check object state via access methods.

- White Box Approach
  - You have the code, look directly at execution,
    - Use debugger to set data member values
    - Use debugger to get data member values
    - Use debugger to check flow of execution
    - Must test all execution paths
Unit Testing – White Box

- Testing Loops
  - Simple loops (for \( I=0; I < n; I++ \))
    - 1. Skip the loop entirely.
    - 2. Only one pass through the loop.
    - 3. Two passes through the loop.
    - 4. \( m \) passes through the loop where \( m < n \).
    - 5. \( n-1, n, n+1 \) passes through the loop

Unit Testing – White Box

- Loop Testing
  - Nested Loops
    - 1. Start with the innermost loop. Set all other loops to minimum values.
    - 2. Conduct simple loop tests for the innermost loop while holding the outer loops at their minimum iteration values.
    - 3. Work outward, conducting tests for the next loop, but keeping all other outer loops at this minimum iteration count.
    - 4. Continue until all loop have been tested.

Unit Testing

- Consider all paths

Unit Testing

- About writing test cases
  - Testing for success
    - Method gives expected results on good input
  - Testing for failure
    - Methods should fail on bad input
  - Test the contract!
    - values within ranges
      - Test boundaries and within range

Unit Testing

- About writing test cases (black box)
  - A test case should be able to...
    - ...run completely by itself, without any human input. Unit testing is about automation.
    - ...determine by itself whether the function it is testing has passed or failed, without a human interpreting the results.
    - ...run in isolation, separate from any other test cases (even if they test the same functions). Each test case is an island.
    - This is what \texttt{try} does for lab submissions!

Equivalence Classes

- Testing all possible stimuli sequences is impossible
- Choose “groups” of stimuli that you guess will cause the same reaction
  - Based on your knowledge of the code
- These groups are equivalence classes.
- Choose only one test from each class.
- Still a large number of tests to be run
**Equivalence classes**

- Two inputs are in the same Equivalence Class if they are handled similarly by system
  - e.g. data field valid value in 1-50
  - So, 20, 38, 1, 47 belong to the same Equivalence Class
  - no need to test multiple values from same Equivalent Class
  - Bounds testing
    - e.g. test 38, then end points 1 and 50
    - test valid and invalid equivalence classes
    - reduces the number of test cases required

**Example:** 3 inputs
- I1 has 10 equivalence classes
- I2 has 10 equivalence classes
- I3 has 10 equivalence classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Total test cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>I1</td>
<td>10 x 10 x 10 = 1000 test cases</td>
</tr>
<tr>
<td>I2</td>
<td>10 x 10 x 10 = 1000 test cases</td>
</tr>
</tbody>
</table>

**Unit Testing**

- Questions?

---

**Summary**

- Software Testing
- Error / Fault / Failure
- Level of Testing
- White Box vs. Black Box
- Unit Testing

- Questions