CISC327 - Software Quality Assurance

Lecture 11

Black Box Testing

Black Box Testing

- Outline
  - Last time we started with black box testing and looked at functionality coverage testing
  - Today we look at the second kind of black box method, input coverage testing
  - We'll look at:
    - Exhaustive testing
    - Input partitioning
    - Shotgun testing
    - Robustness testing

Input Coverage Testing

- Input Coverage
  - The second kind of black box testing
  - Idea: Analyze all the possible inputs allowed by the functional specifications (requirements), create test sets based on the analysis
  - Input coverage methods: exhaustive, input partitioning, shotgun, (robustness) boundary
  - Objective: Show software correctly handles all allowed inputs
  - Question: What does "all" mean?

Exhaustive Testing

- What does "all" mean?
  - Ideally, input coverage testing should try every possible input to the program
    - This is called exhaustive testing
    - Involves testing the program with every possible input
    - Yields a strong result: virtually certain that the program is correct
  - Easy system for test cases, obvious when done
  - But usually impractical, even for very small programs

Exhaustive Testing

- What does "all" mean?
  - Example: Our program specification from last class takes any two integers as input, so we would have to test the program with an infinite number of pairs of integers
  - Even if we limit the input integers to 32 bits each, there are still more than 16,000,000,000,000,000,000 pairs to test

Exhaustive Testing

- But sometimes...
  - However, sometimes exhaustive testing is practical (and when it is, we should do it!)
  - Example: Y2K conversion (Legasys Corp.)
    - The Year 2000 fix automatically applied to 2-digit year comparisons used a conversion like this:

    if YY1 > YY2 then ... (fails when YY1 becomes 00)

    becomes..

    if FourDigit(YY1) > FourDigit(YY2) then ....

    - Regardless of how FourDigit is implemented, the comparison change can be exhaustively tested since there are only 100 YY1 values times 100 YY2 values, for a total of 10,000 different pairs, and every case can be automatically checked

Input Partition Testing

- However, cases where exhaustive testing is practical are extremely rare
- So we must choose another way to decide when we are done testing inputs
- The most common way is to partition all the possible inputs into equivalence classes which characterize sets of inputs with something in common

Input Partition Testing

- Recall our example functional specification from last class
  - "Given as input two integers x and y, output all the numbers smaller than or equal to x that are evenly divisible by y. If either x or y is zero, then output zero."
- The functional specification identifies three special cases for input: the case where x=0, the case where y=0, and the case where neither is zero

Input Partition Testing

- Since the input set is to be integers, we can further partition into negative and positive cases for x and y, giving us the set of input partitions

Partition	x input	y input
P1	0	nonzero
P2	nonzero	0
P3	0	0
P4	less than zero	less than zero
P5	less than zero	greater than zero
P6	greater than zero	less than zero
P7	greater than zero	greater than zero

Input Partition Testing

- Covering Partitions
  - The partitions give us our test cases - all we must do now is design test cases to cover each partition
  - For the reasons we saw last time, for each case, we choose the simplest input values and vary them as little as possible

Input Partition Testing

- Covering Partitions
  - Notice that we do not take into account the intention or actions of the program, only that it handles all its input classes

Partition	x partition	y partition	x input	y input
P1	0	nonzero	**0**	**1**
P2	nonzero	0	**1**	**0**
P3	0	0	**0**	**0**
P4	less than zero	less than zero	**-1**	**-1**
P5	less than zero	greater than zero	**-1**	**1**
P6	greater than zero	less than zero	**1**	**-1**
P7	greater than zero	greater than zero	**1**	**1**

Catching Errors in Requirements

- Blindly following our input partitioning system led us to create tests for negative input, which probably was not even intended
  - We can't tell that from the requirements, which is really the point
- Systematic creation of tests from different points of view is intended to expose problems not only in the software, but also in the specification
  - As a matter of fact, most potential failures caught by systematic testing don't fail when tested, since they instead are fixed by fixing the requirements or the tests before the tests are actually run!

Advantages of Input Partition Testing

- Input partitioning is what many of us think of intuitively for testing
  - That is, testing the response to each kind of input
- It is usually straightforward to identify a set of partitions given the functional specification (although it may require insight)
- We know when we are done: when we have run at least one representative test for each partition
- It gives confidence that the program can at least handle one example of each different kind of input correctly

Black Box Shotgun Testing

- Black box shotgun testing consists of choosing random values for inputs (with or without worrying about legality) over a large number of test runs
- We then verify that the outputs are correct for the legal inputs, and that the program didn't crash for illegal ones
- More practically, we usually choose inputs from the legal set and inputs from the illegal set as separate sets of shotgun tests

Test	x input	y input
T1	682	27631
T2	-89	5244
T3	7368279	-82763

and so on..

Shotgun Testing: Systematic?

- Black box shotgun testing is still a black box method, but it is not really systematic
  - We don't need the code to invent inputs at random
  - Although there is a system for choosing test cases (randomly), there is no completion criterion
- So to gain any confidence, we must run a very large number of test cases, and rely on a statistical argument that the probability of remaining errors is low because of the randomness
- Not really useful, unless there is some way to automate verification of correct output for inputs
  - We don't want to verify output from thousands of runs by hand!

Input Partition Shotgun Testing

- A Hybrid Method
  - Shotgun testing is nevertheless interesting because it tries lots of different inputs
  - We can use it to strengthen input partition testing by applying the shotgun method to choose random input values within each partition
    - That way we have the confidence of input partition testing and the additional confidence that our simple input values are not the only ones that work
  - However, we still need automated output verification to be practical, and we must be careful about experimental design
    - We should run the ordinary simple input partition tests first, then load the shotgun

Automated Output Verification: Differential Testing

"Two men say they're Jesus
One of them must be wrong"

—Dire Straits

- Given a "defined" C program, different C compilers should produce equivalent code
- If they don't, at least one of them is broken

"…we have found and reported more than 325 bugs in mainstream C compilers including GCC, LLVM, and commercial tools."
http://www.cs.utah.edu/~regehr/papers/pldi11-preprint.pdf

Input Robustness Testing

- Robustness is the property that a program doesn't crash or halt unexpectedly, no matter what the input
- Robustness testing tests for this property
- Two kinds of robustness testing:
  1. Shotgun robustness testing (random garbage input)
  2. Boundary value robustness testing

Input Boundary Testing

- Boundary Values
  - Even when programs behave well with input values well outside their expected range, it is typical that failures come at the boundaries of the legal or expected range of values
  - Example: If a sort program expects a list of numbers to sort, it often fails with lists of length one or zero, and with lists exactly as large as the largest allowed (the end of array problem)

Input Boundary Testing

- Boundary Values
  - For this reason, black box testers often create boundary value tests to check that the program is robust with inputs on the edge
  - Unlike shotgun testing, boundary value testing is a systematic test method
    - It has both an easy way to choose test cases, and an easy way to know when we are done (when all boundary values have been tested)

Summary

- Black Box Testing
  - Input coverage methods analyze the set of possible inputs specified and create tests to cover them
  - Exhaustive testing is usually impractical, but we can approximate it using input partitioning
  - Shotgun testing can be added to input partitioning to give additional confidence
  - Robustness testing checks for crashes on unexpected or unusual input, such as the boundaries of the input range

Next Time

- More black box testing:
  output coverage methods