On Testing Non-Testable Programs

MILITANT RACCOONS

Chris Colwell    Spencer Ellsworth
Andrew Foong    Brad Barrows    Ben Upp
Author Background

Dr. Elaine J. Weyuker

- Fellow of ACM, IEEE and AT&T Bell Labs for research in software testing
- Ph.D in Computer Science at Rutgers University in 1977
- Worked for numerous companies (Texaco, IBM, etc.)
- Chair of the ACM Women's Council
Introduction

- Impossible to completely test a program
  - Have an infinite number of inputs, cannot extrapolate how program will react to all inputs
- Try to test programs anyway, use Oracle to see if program outputs correctly
- Oracle - A mechanism which validates the correctness of a program.
Oracle Assumption

- Oracle Assumption - The belief that an Oracle is able to determine correctness, and that, without an Oracle, a program cannot be tested.
- Issues with this:
  - An Oracle does not always exist for a program.
  - An Oracle could potentially be found, but the effort required is impractical.
- Programs with these issues are called "Non-Testable" programs
Partial Oracle Solution

- Can use a Partial Oracle when an Oracle is unattainable
- Able to tell a result is incorrect without knowing the answer
  - Example: If a large company's program says their assets are only $300, the program has an incorrect output.
- Assign measure of likelihood to different values to obtain plausible answers
  - Example: Can guess what value sin (x) should have.
Non-Testable Programs

3 Classes of Non-Testable Programs:
1. Programs written to determine the answer to a problem
2. Programs with excessive amounts of output
3. Programs the tester misunderstands (tester becomes the Oracle and is a "bad" Oracle)

Common Fixes to Non-Testable Programs:
1 & 2: Test with simple data we know the answer to.
3: Fix by using many different testers, give them precise specifications and documentation
Testing Without An Oracle

- Pseudo-Oracle (or dual coding): Independently-written program that follows the same specifications.
  - Written in a VERY high level language
    - Example: SETL
  - Arguments for and against time taken
- Use simple data for the inputs and extrapolate
  - Can check and verify outputs for simple input data
- Accept plausible results, realize program is not perfectly tested, but appears to be relatively accurate
Numerical & Scientific Computations

- Most Non-testable programs fall in this "genre".
- Three sources of error:
  - Mathematical model used
  - Program written for computation
  - Features of the programming environment
- What to do about it:
  - Test with known data
  - Compare with related tests
  - Use really complicated equations that convolute the issue
  - Make your software better (Ex. extra word, SigPac)
  - Create a table for checking differences
Consequences of Having An Oracle

- Result correct, but tester/oracle says incorrect
  - Time wasted trying to find non-existent error
  - Delay in the release, useless debugging
  - Try to 'fix' a correct program
- Result incorrect, but tester/oracle says correct
  - Many validated/tested programs still contain errors
  - Potential for error with non-exhaustive testing
Conclusions

Minimal standard part in documentation

1. Criteria for data selection
2. Degree to which the criteria were fulfilled
3. Test data
4. Output produced
5. Determination of correctness
Discussion

- Is the article still applicable today?
- How do you test your programs? CPE101? CSC309?
- Is there a net benefit to using a pseudo-oracle?
- What makes computational programs the hardest to verify?
- Robert Martin (Oredev) says that we should create a program that mimics the correct output of the program that you would like to create - so as not to "block" others that rely on your programs output. This is similar to Weyuker's pseudo oracle suggestion, but for a different purpose.
Bibliography

