Software Testing, Quality Assurance and Maintenance Winter 2010

Lecture 24 — March 5, 2010

Patrick Lam *version 1*

Mutation Operators

We'll define a number of mutation operators, although precise definitions are specific to a language of interest. Typical mutation operators will encode typical programmer mistakes, e.g. by changing relational operators or variable references; or common testing heuristics, e.g. fail on zero. Some mutation operators are better than others.

The book contains a more exhaustive list of mutation operators. How many (intraprocedural) mutation operators can you invent for the following code?

```
int mutationTest(int a, b) {
  int x = 3 * a, y;
  if (m > n) {
    y = -n;
  }
  else if (!(a > -b)) {
    x = a * b;
  }
  return x;
}
```

Integration Mutation. We can go beyond mutating method bodies by also mutating interfaces between methods, e.g.

- change calling method by changing actual parameter values;
- change calling method by changing callee; or
- change callee by changing inputs and outputs.

```
class M {
  int f, g;

  void c(int x) {
    foo (x, g);
    bar (3, x);
  }

  int foo(int a, int b) {
    return a + b * f;
  }

  int bar(int a, int b) {
    return a * b;
  }
}
```

Mutation for OO Programs. Here are some operators specific to object-oriented programs.

```
class A {
  public int x;
  Object f;
  Square s;

  void m() {
    int x;
    f = new Object();
    this.x = 5;
  }
}

class B extends A {
  int x;
}
```

Exercise. Come up with a test case to kill each of these types of mutants.