White Box Testing III

Outline

- Today we continue our look at white box testing methods, with mutation testing
- We will look at :
  - definition and role of mutation testing
    - what is a mutation?
    - how is mutation testing used?
  - mutation testing methods
    - value mutations
    - decision mutations
    - other mutations
    - examples

Mutation Testing

What is it for?

- Mutation testing is a white box method for checking the adequacy of test suites
- As you have already discovered, creating a test suite can be an expensive and time consuming effort
- No matter what test method is used, discovering if test suites are adequate to uncover faults is itself an even more difficult task
- Mutation testing offers an almost completely automated way to check the adequacy of a set of tests in uncovering faults in the software

Mutation Testing

How does it work?

- In order to test the adequacy of a test suite, we first run the software on the suite and fix any problems until we are satisfied that the software passes the tests
- We then save the results of the tests in a file or set of files to serve as the correct output to compare to

Mutation Testing

How does it work?

- We then use mutation of the source code to create a set of mutants, each of which is a program slightly different from the original
- For each mutant, we run the test suite on the mutant and compare the results to the saved results from the original
  - If the results differ, then the mutant has been “killed” (detected) by the test suite
  - If the results do not differ, then the test suite is inadequate to detect the mutant, and a new test must be added to the suite to “kill” that mutant

Systematic Mutation

Systematic Approach

- In order for mutation testing to be systematic, there must be a system and a completion criterion for creating mutants
- The system for mutation normally specifies simple syntactic changes to the program source representing errors in the code
- Each mutant has exactly **one** change in it
- We are done when every possible single change mutant of the system has been generated and tested

Systematic Mutation

Systematic Approach

- Example systematic mutations are:
  - value mutations (changing constants, subscripts or parameters by adding or subtracting one, etc.)
  - decision mutations (inverting or otherwise modifying the sense of each decision condition in the program)
  - statement mutations (deleting or exchanging individual statements in the program)

Example #1: Value Mutation

Value Mutation Example

- System: Mutate the value of each constant in the program to be off by one (or more generally, each integer expression)
- Completion criterion: One mutant for each constant in the program

- Note that there are many other possible value mutations:
  - constants modified in some other way, e.g. off by -1
  - all integer expressions modified (not just constants),
    e.g., x changed to x+1, etc.

Example #1: Value Mutation

```
// calculate numbers less than x
//   which are divisible by y

int x, y;
x = c.readInt();
y = c.readInt();


if (y == 0)
  c.println ("y is 0");
else if (x == 0)
  c.println ("x is 0");
else
{
  for (int i = 1; i <= x; i++)
  {
    if (i % y == 0)
      c.println (i);
  }
}
```

Example test suite
(statement coverage):

Test	x	y	output
T1	0	0	“y is 0”
T2	0	1	“x is 0”
T3	1	1	1

Example #1: Value Mutation

```
// calculate numbers less than x
//   which are divisible by y

int x, y;
x = c.readInt();
y = c.readInt();


if (y == 1)
  c.println ("y is 0");
else if (x == 0)
  c.println ("x is 0");
else
{
  for (int i = 1; i <= x; i++)
  {
    if (i % y == 0)
      c.println (i);
  }
}
```

Test	x	y	original output	mutant output
T1	0	0	“y is 0”	“x is 0”
T2	0	1	“x is 0”	“y is 0”
T3	1	1	1	“y is 0”

Example #1: Value Mutation

```
// calculate numbers less than x
//   which are divisible by y

int x, y;
x = c.readInt();
y = c.readInt();


if (y == 0)
  c.println ("y is 0");
else if (x == 1)
  c.println ("x is 0");
else
{
  for (int i = 1; i <= x; i++)
  {
    if (i % y == 0)
      c.println (i);
  }
}
```

Test	x	y	original output	mutant output
T1	0	0	“y is 0”	“y is 0”
T2	0	1	“x is 0”	
T3	1	1	1	“x is 0”

Example #1: Value Mutation

```
// calculate numbers less than x
//   which are divisible by y

int x, y;
x = c.readInt();
y = c.readInt();


if (y == 0)
  c.println ("y is 0");
else if (x == 0)
  c.println ("x is 0");
else
{
  for (int i = 2; i <= x; i++)
  {
    if (i % y == 0)
      c.println (i);
  }
}
```

Test	x	y	original output	mutant output
T1	0	0	“y is 0”	“y is 0”
T2	0	1	“x is 0”	“x is 0”
T3	1	1	1	

Example #1: Value Mutation

```
// calculate numbers less than x
//   which are divisible by y

int x, y;
x = c.readInt();
y = c.readInt();


if (y == 0)
  c.println ("y is 0");
else if (x == 0)
  c.println ("x is 0");
else
{
  for (int i = 1; i <= x; i++)
  {
    if (i % y == 1)
      c.println (i);
  }
}
```

			original	mutant
Test	x	y	output	output
T1	0	0	"y is 0"	"y is 0"
T2	0	1	"x is 0"	"x is 0"
T3	1	1	1	

Example #2: Decision Mutation

Decision Mutation Example

- System: Invert the sense of each decision condition in the program (e.g., change > to <, == to !=, and so on)
- Completion criterion: One mutant for each decision condition in the program

Example #2: Decision Mutation

```
// calculate numbers less than x
//   which are divisible by y

int x, y;
x = c.readInt();
y = c.readInt();


if (y != 0)
  c.println ("y is 0");
else if (x == 0)
  c.println ("x is 0");
else
{
  for (int i = 1; i <= x; i++)
  {
    if (i % y == 0)
      c.println (i);
  }
}
```

			original	mutant
Test	x	y	output	output
T1	0	0	“y is 0”	“x is 0”
T2	0	1	“x is 0”	“y is 0”
T3	1	1	1	“y is 0”

Example #2: Decision Mutation

```
// calculate numbers less than x
//   which are divisible by y

int x, y;
x = c.readInt();
y = c.readInt();


if (y == 0)
  c.println ("y is 0");
else if (x != 0)
  c.println ("x is 0");
else
{
  for (int i = 1; i <= x; i++)
  {
    if (i % y == 0)
      c.println (i);
  }
}
```

Test	x	y	original output	mutant output
T1	0	0	“y is 0”	“y is 0”
T2	0	1	“x is 0”	
T3	1	1	1	“x is 0”

Example #2: Decision Mutation

```
// calculate numbers less than x
//   which are divisible by y

int x, y;
x = c.readInt();
y = c.readInt();


if (y == 0)
  c.println ("y is 0");
else if (x == 0)
  c.println ("x is 0");
else
{
  for (int i = 1; i <= x; i++)
  {
    if (i % y != 0)
      c.println (i);
  }
}
```

Test	x	y	original output	mutant output
T1	0	0	“y is 0”	“y is 0”
T2	0	1	“x is 0”	“x is 0”
T3	1	1	1	🟨

Example #3: Statement Mutation

Statement Mutation Example

- System: Delete each statement in the program
- Completion criterion: One mutant for each statement

- Note that there are many other possible statement mutations:
  - interchanging adjacent statements
  - shuffling sequences of statements
  - doubling statements
  - many more

Example #3: Statement Mutation

```
// calculate numbers less than x
//   which are divisible by y

int x, y;
x = c.readInt();
y = c.readInt();


if (y == 0)
  ;
else if (x == 0)
  c.println ("x is 0");
else
{
  for (int i = 1; i <= x; i++)
  {
    if (i % y == 0)
      c.println (i);
  }
}
```

Test	x	y	original output	mutant output
T1	0	0	“y is 0”	
T2	0	1	“x is 0”	“x is 0”
T3	1	1	1	1

This time, we show only one example - you can make the rest!
Again, all mutants turn out to be “killed”

Mutation Testing in Practice

Some Final Observations

- In practice, simple statement coverage tests are often sufficient to "kill" most kinds of mutants Not really a surprise, when you think about it - and the main reason coverage tests are worth doing (more on this next time)
- If we do mutation testing on acceptance, functionality coverage, input/output coverage or other black box test suites, on the other hand, we are likely to find many mutants not "killed" by the tests
- Since most projects use primarily black box techniques, automated mutation testing can be a very valuable help in making test suites more effective

White Box Testing III

Summary

- Mutation Testing is a white box method for automatically checking test suites for completeness
- Mutations are simple, syntactic variants of programs that can be generated automatically
- Typical mutations are value mutations, decision mutations, statement mutations