Com S 417
Software Testing

Fall 2017 – Week 4, Lecture 8

Announcements

- Lab 2 is posted.
  - Due before class Sept. 19
- Exam 1
  - In class (1 hour long) Thursday, Sept. 21
  - Material covered in weeks 1-4.

Designing with Path Coverage

Some useful concepts

- Complete path
  - First node is Start and last node is the exit (end).
- Independent path
  - A complete path which *introduces* (to some set of paths being evaluated) at least one new set of processing statements (node) or a new condition (connection to a new link).
- Cyclomatic Complexity
  - A measure of the number of linearly independent paths through a graph. See Wikipedia.

Basis Paths

- The minimum set of independent paths required to *guarantee complete branch coverage* (same as edge coverage).
  - The number of independent paths corresponds to the minimum number of test cases required to achieve branch coverage!
  - The number of Basis Paths is the same as the graph cyclomatic complexity.

Cyclomatic Complexity

Cyclomatic complexity is a software metric

- Cyclomatic complexity, V(G), for a flow graph G is defined as

$$V(G) = (E - N) + 2$$

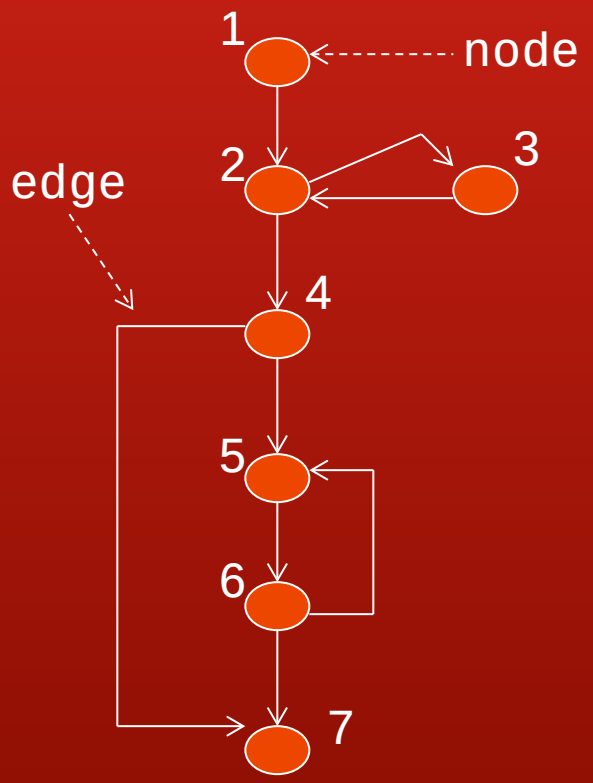
  where E is the number of flow graph edges and N is the number of flow graph nodes.

- Cyclomatic complexity, alternatively

$$V(G) = P + 1$$

  where P is the number of predicate nodes contained in the flow graph G.

An Example of Cyc. Complexity



- No. of edges = 9
- No. of nodes = 7
- No. of predicate nodes = 3

V(G) = P + 1

V(G) = 3 + 1 = 4

V(G) = (E – N) + 2

V(G) = (9 - 7) + 2 = 4

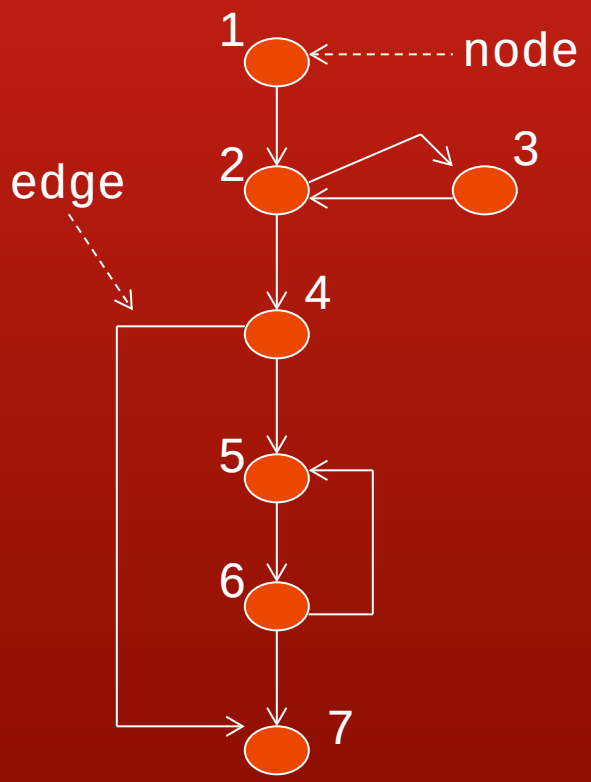
Adapted from materials by Dr. Mitra

Designing Basis Path Test Sets

1. Using the design or code as a foundation, draw a corresponding flow graph.
2. Determine the cyclomatic complexity of the resultant flow graph.
3. Determine a minimum basis set of linearly independent paths.

   For example,

   - path 1: 1-2-4-5-6-7
   - path 2: 1-2-4-7
   - path 3: 1-2-3-2-4-5-6-7
   - path 4: 1-2-4-5-6-5-6-7
4. Prepare test cases that will force execution of each path in the basis set.
5. Run the test cases and check their results



Adapted from materials by Dr. Mitra

Another Example

V(G) = P + 1

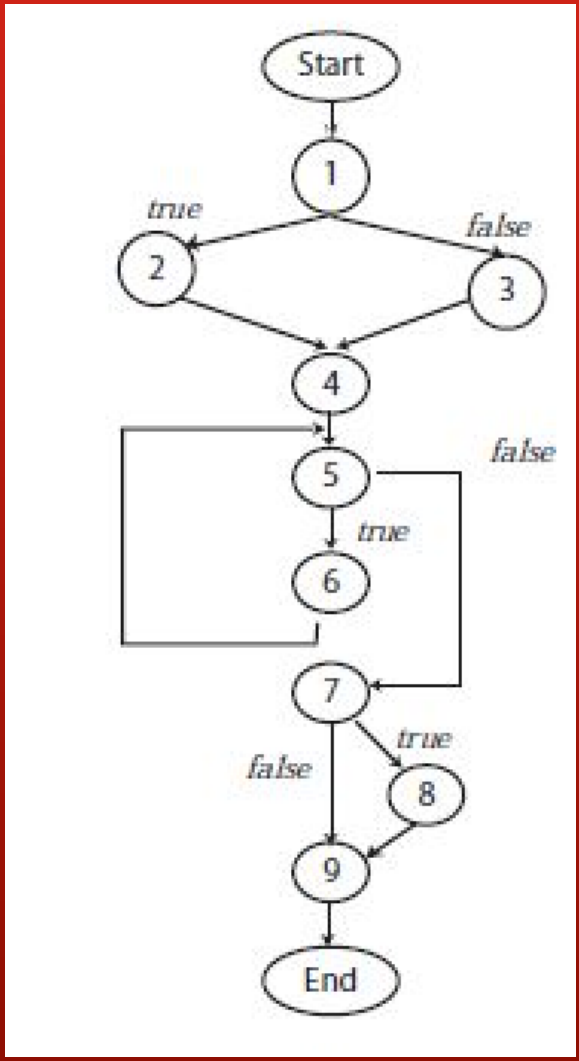
= 3 + 1

= 4

What are the paths?

Remember:

There is no guarantee all independent paths are feasible.



Limited Applicability

Loops are still a challenge

Every decision doubles the number of paths and

Every loop multiplies the paths by the number of iterations through the loop.

- For example:

```
for (i=1; i<=1000; i++)
  for (j=1; j<=1000; j++)
    for (k=1; k<=1000; k++)
      doSomethingWith(i,j,k);
```

executes doSomethingWith() one billion times (1000×1000×1000).

Each unique path deserves to be tested.

Limited Warranty

Guaranteed Branch Coverage Doesn’t guarantee Finding Faults

Required behavior may be missing
in otherwise correct code.

Example:

```
if (a>0) doIsGreater();
if (a==0) doIsEqual();
// missing statement - if (a<0) doIsLess();
```

Limited Warranty

Guaranteed Branch Coverage Doesn't guarantee Infection

Infection may also depend upon data values:

The module may execute correctly for almost all data values but fail for a few.

```
int blech (int a, int b) {
   return a/b;
 }
```

fails if **b** has the value 0 but executes correctly if **b** is not 0.

Data Flow Testing

Data Flow Testing

- Data flow testing is a powerful tool to detect improper use of data values due to coding errors.

```
//c code
main() {
        int x;
        if (x==42){ ...}
  }
```

what is value of x in the condition? (This is an error)

Data (Resource) Life Cycle

Variables, including objects that contain data values, have a defined life cycle.

- They are ***defined***,
- they are ***used***, and
- they are ***killed*** (destroyed)

```
{ // begin outer block
   int x;      // x is defined as an
               //integer within this outer block
   x = 0;
   dosomething(x);          // x can be accessed here
   {           // begin inner block
      int y; // y is defined within this inner block
             // both x and y can be accessed here
      y = 2;
      dosomething(x, y);
      // y is automatically destroyed
      // at the end of this block
   }
   // x can still be accessed, but y is gone
   dosomething(x);
}
```


Adapted with permission from materials by Dr. Mitra

Data Flow Testing

- Variables can be used
  - in computation
  - in conditionals
- Possibilities for the first occurrence of a variable through a program path (~ represents variable non-extant)
  - ~d the variable does not exist, then it is defined (d)
  - ~u the variable does not exist, then it is used (u)
  - ~k the variable does not exist, then it is killed or destroyed (k)

Adapted with permission from materials by Dr. Mitra

Data Flow Testing

Time-sequenced pairs can identify mis-uses.

defined (d), used (u), and killed (k)

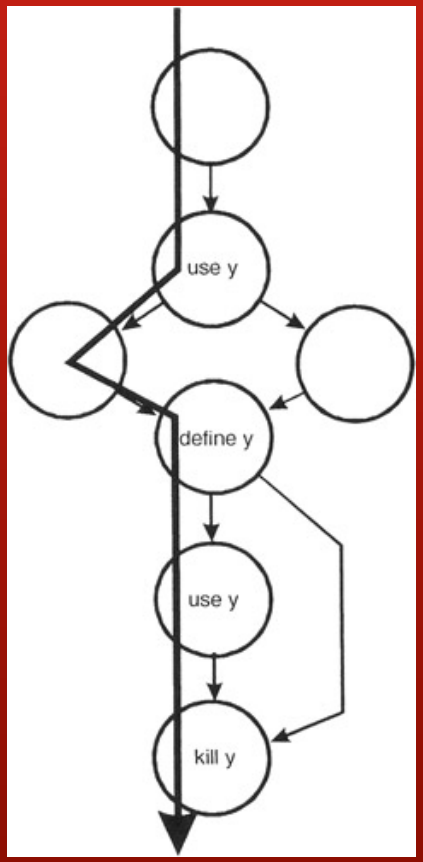
Pair	Determination
dd	not invalid but suspicious. Probably an error.
du	perfectly correct. The normal case.
dk	not invalid but probably an error.
ud	acceptable
uu	acceptable

Pair	Determination
uk	acceptable
kd	acceptable
ku	a serious defect. Using a variable that does not exist or is undefined is always an error.
kk	probably a programming error.

Adapted with permission from materials by Dr. Mitra

Example - variable y

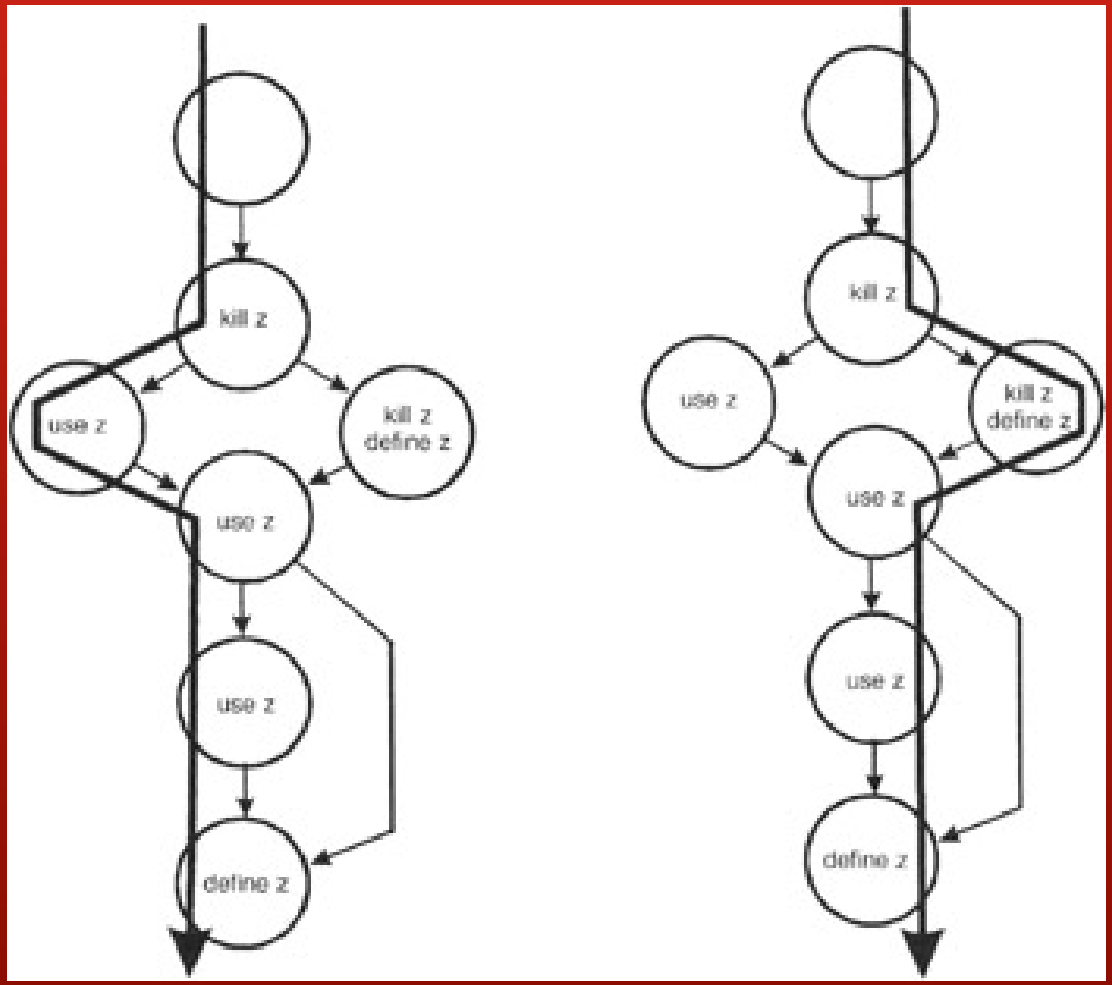
~use	major blunder
use-define	acceptable
define-use	correct, the normal case
use-kill	acceptable



Adapted with permission from materials by Dr. Mitra

EXAMPLE - variable z

Pair	Determination
~k	programming error
ku	major blunder
uu	correct, the normal case
ud	acceptable
kk	probably an error
kd	acceptable
du	correct, the normal case



Adapted with permission from materials by Dr. Mitra

Static Data Flow Testing

In many cases, it is possible to identify improper data usage by compiler techniques.

In some cases, it is not possible to identify. Example: Arrays are collections of data elements that share the same name and type.

For example

```
int test[] = new int[100];    //defines an array
// consisting of 100 integer elements,
// named test[0], test[1], etc.
```

- Arrays are defined and destroyed as a unit but specific elements of the array are used individually.
- Static analysis cannot determine whether the define-use-kill rules have been followed properly unless each element is considered individually.

Dynamic Data Flow Testing

- Data flow testing is based on a module's control flow, it assumes that the control flow is basically correct.
- The data flow testing process is to choose enough test cases so that:
  - Every "define" is traced to each of its "uses"
  - Every "use" is traced from its corresponding "define"

Dynamic DF Testing

The Process

- enumerate the paths through the module.
- Begin at the module's entry point, take the leftmost path through the module to its exit.
- Return to the beginning and vary the first branching condition. Follow that path to the exit.
- Repeat until all the paths are listed.
- For every variable, create at least one test case to cover every define-use pair.
  - How do we choose the values?

Adapted with permission from materials by Dr. Mitra

Questions

Reading, Last Lecture, Lab?

Other Resources

A more elaborate presentation on data flow testing:

https://www.cs.drexel.edu/~spiros/teaching/CS576/slides/4.data-testing.pdf

Reading Assignment

- http://www.mccabe.com/pdf/mccabe-nist235r.pdf
  - Chapter 4 and Chapter 6