Software Testing

Sibylle Schupp[1]

[1] Institute for Software Systems/Institut für Softwaresysteme
Hamburg University of Technology (TUHH)

Spring 2022

Lecture 5

Outline

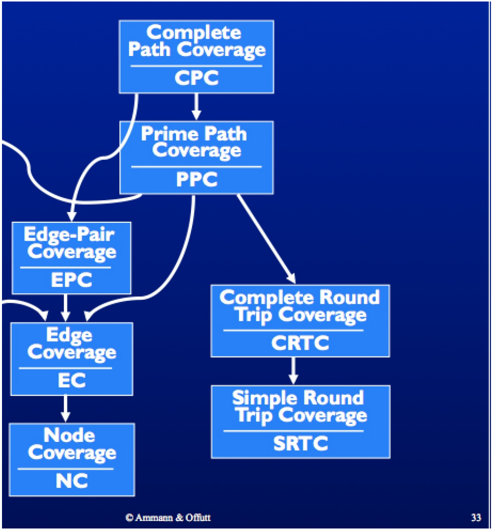
1. Graph coverage II

Recap

Graph-based testing:

- Test sets are represented by *test* paths. A path is a test path if it starts at an initial node and ends at a final node.
  - Test sets assume a concrete executable (modeled by the graph).
- Test requirements are derived from a graph-based coverage criterion.
  - TRs assume a concrete graph.
  - TRs are formulated in terms of graph entities.
  - Major classes: structural criteria, data-flow criteria
- Loop testing requires special criteria, based on prime paths.
  - Prime paths contain no inner loops (“simple path”) and are no proper subpath of another simple path.

Recap (structural coverage)



In-class exercise

Outline

1. Graph coverage II
   - Data-flow criteria
   - Structural coverage for source code
   - Data-flow coverage for source code

Data-flow criteria

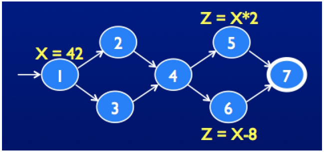
- Data-flow testing tests whether values are computed and used correctly.

Definition

A program point (node) is a def node if the value of a variable is stored at it. It is a use node if the value of a variable is accessed at it.

The set of variables that are defined by node *n* (edge *e*) is denoted as def(n) (or (def(e)). The set of variables that are used by node *n* (edge *e*) is denoted as use(n) (or (use(e)).

- Example:



def(1) $=$ {X}, def(5) $=$ {Z} $=$ def(6)
use(5)$=$ {X} $=$ use (6)

DU pairs and def-clear paths

Definition

- A pair of program points (n_i, n_j) is called a DU pair if there exists a variable that is defined at n_i and used at n_j.
- A path from n_i to n_j is called a def-clear path with respect to variable v if $v \notin \text{def}(n_k)$ for any node n_k, $k \neq i$, on that path.
  - If $v \in \text{def}(n_i)$ and there is a def-clear path from n_i to n_j with respect to v, the def of v at n_i reaches the use of v at n_j.
  - The definition can also be expressed in terms of edges. A path from n_i to n_j is called a def-clear path with respect to variable v if $v \notin \text{def}(e_k)$ for any edge $e_k = (n_k, n_l)$, $k, l \neq i, j$ on that path.

DU paths

Definition

- A simple subpath that is def-clear with respect to v from a def of v to a use of v is called a DU path.
- Properties
  - A DU path is always qualified by a particular variable.
  - A DU path contains one def, it may contain more than one use.
- The following sets of DU paths are relevant:
  - The def-path set $\text{du}(n_i, v)$ is the set of DU paths for v that start at n_i.
  - The def-pair set $\text{du}(n_i, n_j, v)$ is the set of DU paths for v that start at n_i and end at n_j.

Touring DU paths

Definition

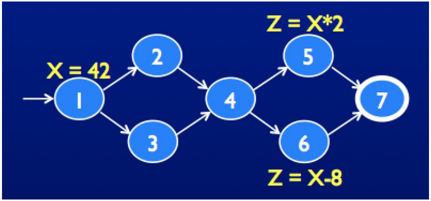
A test path p du-tours a subpath d with respect to v if p tours d and the subpath taken is def-clear with respect to v.

- Sidetrips possible (as before)

Data-flow coverage criteria

- The most general criterion is ADUPC, which includes every DU path.
  All-du-path coverage (ADUPC): For each def-pair set S,
  $S = du(n_i, n_j, v)$*, TR contains every path* $d \in S$.
- All-uses coverage includes for every du-pair at least one path:
  All-uses coverage (AUC): For each def-pair set S,
  $S = du(n_i, n_j, v)$*, TR contains at least one path* $d \in S$.
- All-defs coverage includes one DU path per definition:
  All-defs coverage (ADC): For each def-path set S,
  $S = du(n, v)$*, TR contains at least one path* $d \in S$.

Example (data-flow testing)



- All-defs for X: [1,2,4,5]
- All-uses for X: [1,2,4,5], [1,2,4,6]
- All-du-paths for X: [1,2,4,5], [1,2,4,6], [1,3,4,5], [1,3,4,6]

Subsumption

- ADUPC subsumes AUC, and AUC subsumes ADC.
- Further, Prime Path Coverage subsumes ADUPC.

Next: From graphs at the abstract level . . . to graphs at source-code level, specification level, design level.

Outline

1. Graph coverage II
   - Data-flow criteria
   - Structural coverage for source code
   - Data-flow coverage for source code

Graph coverage for source code

- Structural coverage
  - Graph = control-flow graph (CFG)
  - Node coverage: execute every statement
  - Edge coverage: execute every branch
- Data-flow coverage
  - Graph = augmented CFG
  - defs: assignments
  - uses: readings

CFGs

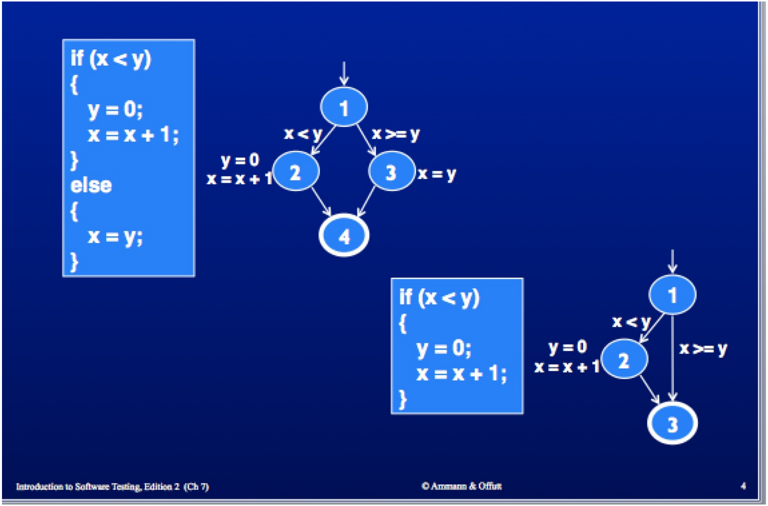
- A CFG models all executions of a method.

Definition

A control-flow graph of a function is a graph where each statement (or each basic block) of the function forms a node, and edges represent the transfer of control.
A basic block is a sequence of statements such that if the first statement is executed, all statements will be.

- The CFG can be annotated with extra information (predicates, defs, uses).
- The CFG can be obtained automatically from the source code.

If-statements



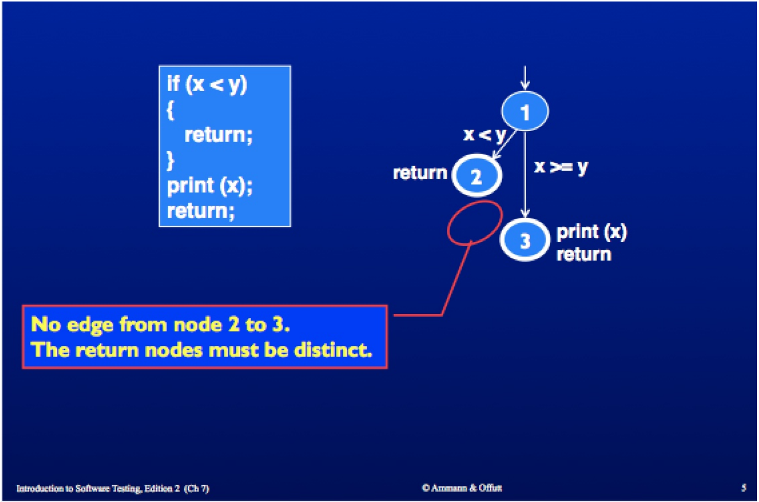
```
if (x < y)
{
  y = 0;
  x = x + 1;
}
else
{
  x = y;
}
```

```
if (x < y)
{
  y = 0;
  x = x + 1;
}
```

Introduction to Software Testing, Edition 2 (Ch 7) © Ammann & Offutt 4

- The CFG fragment contains a fork and a join node.

If-return statements

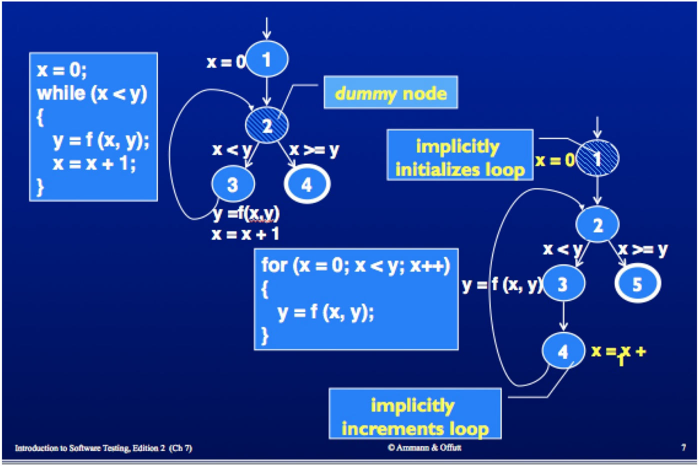


```
if (x < y)
{
   return;
}
print (x);
return;
```

No edge from node 2 to 3.
The return nodes must be distinct.

Introduction to Software Testing, Edition 2 (Ch 7) © Ammann & Offutt 5

While- and for-statements



```
x = 0;
while (x < y)
{
   y = f (x, y);
   x = x + 1;
}
```

dummy node

```
for (x = 0; x < y; x++)
{
   y = f (x, y);
}
```

implicitly initializes loop

implicitly increments loop

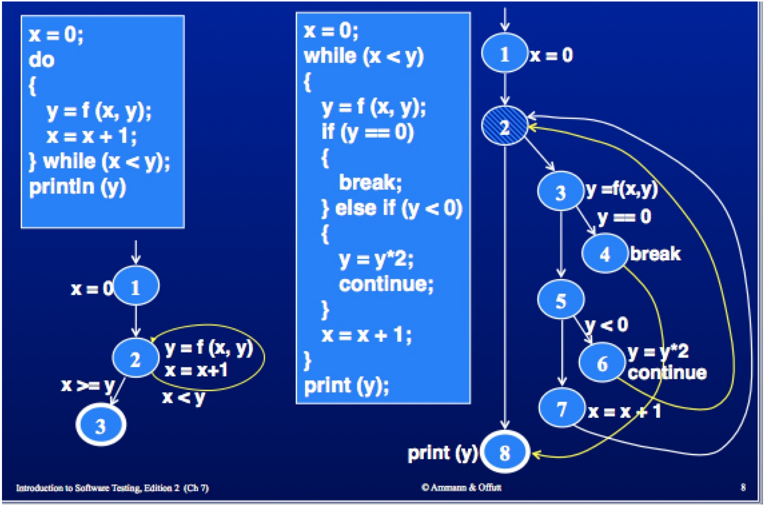
Introduction to Software Testing, Edition 2 (Ch 7)

© Ammann & Offutt

7

- While- and for-loops require separate nodes for the loop header.

Do loop, break and continue

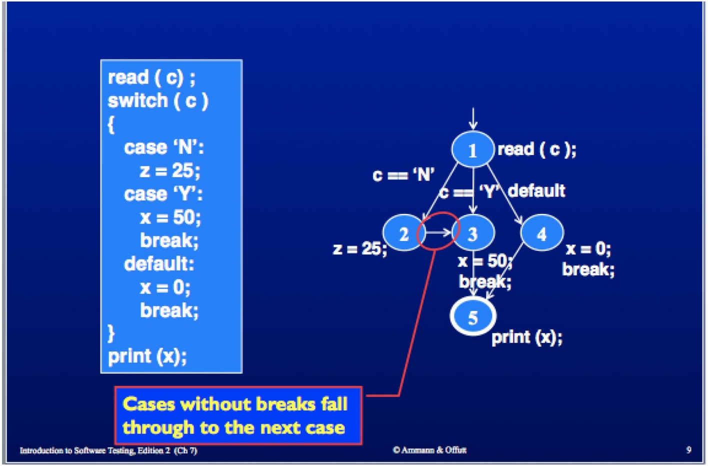


```
x = 0;
do
{
  y = f (x, y);
  x = x + 1;
} while (x < y);
println (y)
```

```
x = 0;
while (x < y)
{
  y = f (x, y);
  if (y == 0)
  {
    break;
  } else if (y < 0)
  {
    y = y*2;
    continue;
  }
  x = x + 1;
}
print (y);
```

Introduction to Software Testing, Edition 2 (Ch 7) © Ammann & Offutt 8

Case statements



```
read ( c) ;
switch ( c )
{
   case 'N':
      z = 25;
   case 'Y':
      x = 50;
      break;
   default:
      x = 0;
      break;
}
print (x);
```

Cases without breaks fall through to the next case

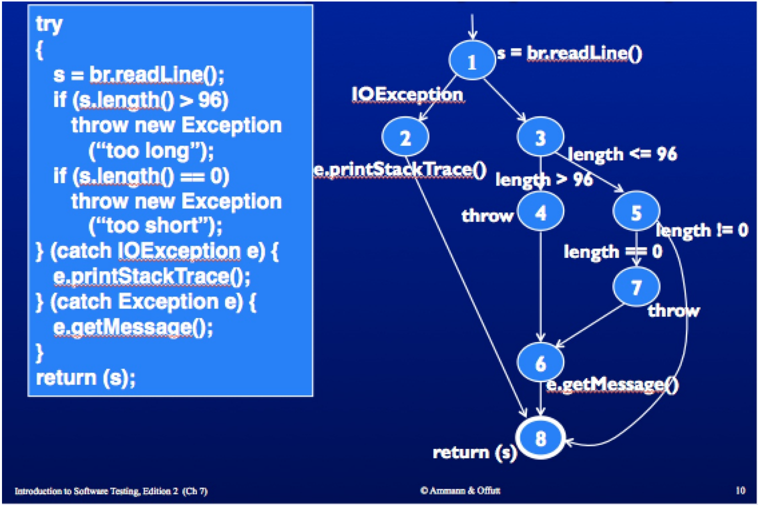
Introduction to Software Testing, Edition 2 (Ch 7)

© Ammann & Offutt

9

- Note: for languages with a different semantics (no fall through), the CFG fragment looks different.

Try-catch statements



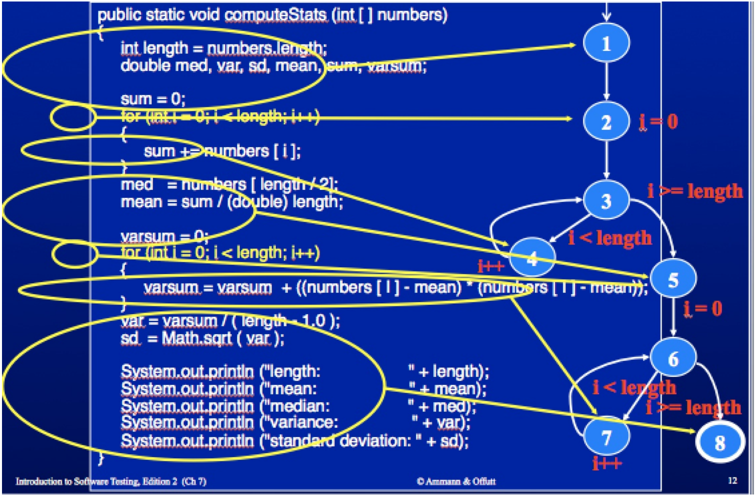
```
try
{
   s = br.readLine();
   if (s.length() > 96)
      throw new Exception
         (“too long”);
   if (s.length() == 0)
      throw new Exception
         (“too short”);
} (catch IOException e) {
   e.printStackTrace();
} (catch Exception e) {
   e.getMessage();
}
return (s);
```

Introduction to Software Testing, Edition 2 (Ch 7) © Ammann & Offutt 10

Example (CFG)

```
public static void computeStats (int [ ] numbers)
{
    int length = numbers.length;
    double med, var, sd, mean, sum, varsum;
    sum = 0;
    for (int i = 0; i < length; i++)
    {
        sum += numbers[i];
    }
    med   = numbers[length/2];
    mean = sum / (double) length;
    varsum = 0;
    for (int i = 0; i < length; i++)
    {
        varsum = varsum + ((numbers[i] - mean) * (numbers[i] - mean));
    }
    var = varsum / ( length - 1.0 );
    sd  = Math.sqrt( var );
    System.out.println ("length:                     " + length);
    System.out.println ("mean:                      " + mean);
    System.out.println ("median:                   " + med);
}
```

Example (CFG), cont'd



```
public static void computeStats (int [ ] numbers)
{
     int length = numbers.length;
     double med, var, sd, mean, sum, varsum;

     sum = 0;
     for (int i = 0; i < length; i++)
     {
          sum += numbers [ i ];
     }
     med   = numbers [ length / 2];
     mean = sum / (double) length;

     varsum = 0;
     for (int i = 0; i < length; i++)
     {
          varsum = varsum  + ((numbers [ I ] - mean) * (numbers [ I ] - mean));
     }
     var = varsum / ( length - 1.0 );
     sd  = Math.sqrt ( var );

     System.out.println ("length:              " + length);
     System.out.println ("mean:                " + mean);
     System.out.println ("median:              " + med);
     System.out.println ("variance:             " + var);
     System.out.println ("standard deviation: " + sd);
}
```

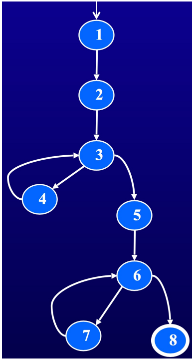
Introduction to Software Testing, Edition 2 (Ch 7)

© Ammann & Offutt

12

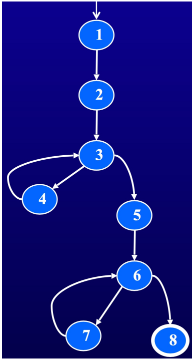
TRs and test paths (EC)

TR	Test path
A. [1,2]	[1,2,3,4,3,5,6,7,6,8]
B. [2,3]	
C. [3,4]	
D. [3,5]	
E. [4,3]	
F. [5,6]	
G. [6,7]	
H. [6,8]	
I. [7,6]	



TRs and test paths (EPC)

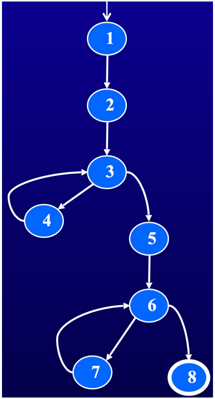
TR	Test paths	Tours
A. [1,2,3]	i. [1,2,3,4,3,5,6,7,6,8]	A,B,D,E,F,G,I,J
B. [2,3,4]	ii. [1,2,3,5,6,8]	A,C,E,H
C. [2,3,5]	iii. [1,2,3,4,3,4,3,5,6,7,6,7,6,8]	A,B,D,E,F,G,I,J, K, L
D. [3,4,3]		
E. [3,5,6]		
F. [4,3,5]		
G. [5,6,7]		
H. [5,6,8]		
I. [6,7,6]		
J. [7,6,8]		Sidetrips
K. [4,3,4]		Test i: C, H
L. [7,6,7]		Test iii: C, H
∪ E ∪ N		



Could reduce the test set to Test (iii), at the price of 2 side trips.

TRs and test paths (PPC)

TR	Test paths	Tours
A. [3,4,3]	i. [1,2,3,4,3,5,6,7,6,8]	A,D,E,F,G
B. [4,3,4]	ii. [1,2,3,4,3,4,3,5,6,7,6,7,6,8]	A,B,C,D,E,F,G
C. [7,6,7]	iii. [1,2,3,4,3,5,6,8]	A,F,H
D. [7,6,8]	iv. [1,2,3,5,6,7,6,8]	D,E,F,I
E. [6,7,6]	v. [1,2,3,5,6,8]	J
F. [1,2,3,4]		Sidetrips
G. [4,3,5,6,7]		Test i: H,I,J
H. [4,3,5,6,8]		Test ii: H,I,J
I. [1,2,3,5,6,7]		Test iii: J
J. [1,2,3,5,6,8]		Test iv: J



Outline

1. Graph coverage II
   - Data-flow criteria
   - Structural coverage for source code
   - Data-flow coverage for source code

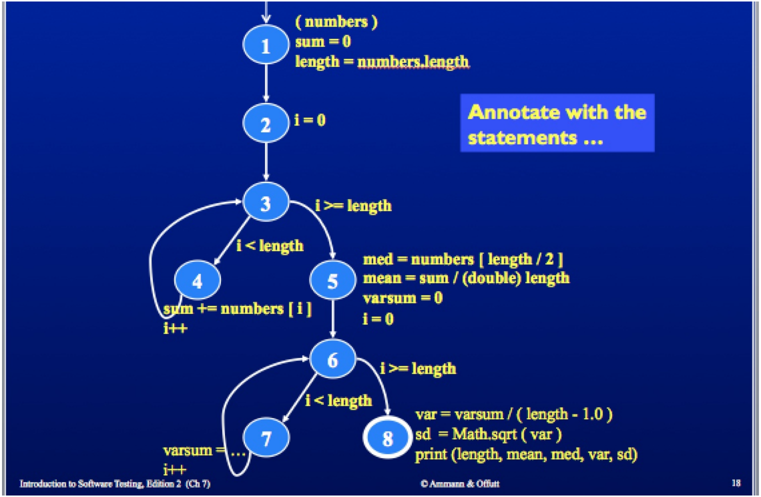
Data-flow coverage

- defs
  - initialization, l-values
  - actual parameters of a method that are changed
  - formal parameters of a method that are (implicitly) initialized
  - input parameters
- uses
  - r-values (in statements, expressions, tests)
  - actual parameters of a method
  - return parameters of a method
  - output parameters
- DU pairs
  - If $v \in \text{def}(n)$ and $v \in \text{use}(n)$, the pair (n, n) forms a DU pair only if the def executes after the use and n is within a loop.

Example (data-flow testing)

```java
public static void computeStats (int [ ] numbers)
{
  int length = numbers.length;
  double med, var, sd, mean, sum, varsum;
  sum = 0;
  for (int i = 0; i < length; i++)
  {
     sum += numbers[i];
  }
  med    = numbers[length/2];
  mean = sum / (double) length;
  varsum = 0;
  for (int i = 0; i < length; i++)
  {
     varsum = varsum + ((numbers[i] - mean) * (numbers[i] - mean));
  }
  var = varsum / ( length - 1.0 );
  sd  = Math.sqrt( var );
  System.out.println ("length:                    " + length);
  System.out.println ("mean:                     " + mean);
  System.out.println ("median:                  " + med);
}
```

Example (cont'd)



Annotate with the statements …

(numbers)
sum = 0
length = numbers.length

i = 0

sum += numbers [i]
i++

med = numbers [length / 2]
mean = sum / (double) length
varsum = 0
i = 0

varsum = …
i++

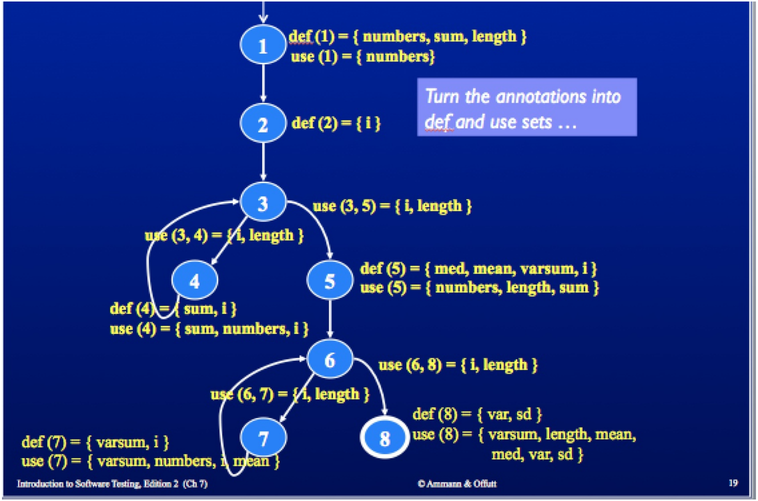
var = varsum / (length - 1.0)
sd = Math.sqrt (var)
print (length, mean, med, var, sd)

Introduction to Software Testing, Edition 2 (Ch 7)

© Ammann & Offutt

18

Example (cont'd): from annotations to defs and uses



def (1) = { numbers, sum, length }
use (1) = { numbers}

def (2) = { i }

Turn the annotations into def and use sets …

use (3, 5) = { i, length }

use (3, 4) = { i, length }

def (5) = { med, mean, varsum, i }
use (5) = { numbers, length, sum }

def (4) = { sum, i }
use (4) = { sum, numbers, i }

use (6, 8) = { i, length }

use (6, 7) = { i, length }

def (8) = { var, sd }
use (8) = { varsum, length, mean, med, var, sd }

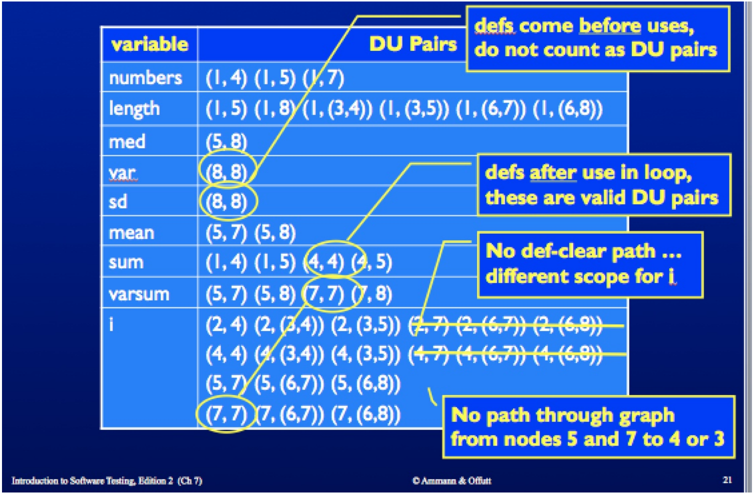
def (7) = { varsum, i }
use (7) = { varsum, numbers, i, mean }

Introduction to Software Testing, Edition 2 (Ch 7)

© Ammann & Offutt

19

Example: DU pairs



variable	DU Pairs
numbers	(1, 4) (1, 5) (1, 7)
length	(1, 5) (1, 8) (1, (3,4)) (1, (3,5)) (1, (6,7)) (1, (6,8))
med	(5, 8)
var	(8, 8)
sd	(8, 8)
mean	(5, 7) (5, 8)
sum	(1, 4) (1, 5) (4, 4) (4, 5)
varsum	(5, 7) (5, 8) (7, 7) (7, 8)
i	(2, 4) (2, (3,4)) (2, (3,5)) ~~(2, 7) (2, (6,7)) (2, (6,8))~~ (4, 4) (4, (3,4)) (4, (3,5)) ~~(4, 7) (4, (6,7)) (4, (6,8))~~ (5, 7) (5, (6,7)) (5, (6,8)) (7, 7) (7, (6,7)) (7, (6,8))

defs come before uses, do not count as DU pairs

defs after use in loop, these are valid DU pairs

No def-clear path … different scope for i

No path through graph from nodes 5 and 7 to 4 or 3

Introduction to Software Testing, Edition 2 (Ch 7) © Ammann & Offutt 21

Example: from DU pairs to DU paths

var	DU pair	DU path
number	(1,4)	[1,2,3,4]
	(1,5)	[1 2,3,5]
	(1,7)	[1,2,3,5,6,7]
length	(1,5)	[1,2,3,5]
	(1,8)	[1,2,3,5,6,8]
	(1,(3,4))	[1,2,3,4]
	(1,(3,5))	[1,2,3,5]
	(1,(6,7))	[1,2,3,5,6,7]
	(1,(6,8))	[1,2,3,5,6,8]
med		. . .
. . .		

Example: unique DU paths

In the example, there is a total of 12 (unique) DU paths.

- 4 paths skip the loop

  [1,2,3,5], [2,3,5], [1,2,3,5,6,8], [5,6,8]

- 6 paths require at least one iteration

  [1,2,3,4], [1,2,3,5,6,7], [2,3,4], [4,3,5], [5,6,7], [7,6,8]

- 2 paths require two iterations

  [4,3,4], [7,6,7]

Test cases and test paths

- Test case: numbers = [44], length = 1
  - Test path: [1,2,3,4,3,5,6,7,6,8]
  - DU paths covered without sidetrip: 5 of the 6 paths that require at least one iteration ([1,2,3,4], [2,3,4], [4,3,5], [5,6,7], [7,6,8])
- Test case: numbers = [2,10,15], length = 3
  - Test path: [1,2,3,4,3,4,3,4,3, 5,6,7,6,7,6,7,6,7,6,8]
  - DU paths covered without sidetrip: both paths that require two iterations [4,3,4], [7,6,7] (plus some that require one iteration)
- Test case: numbers [], length = 0
  - Test path: [1,2,3,5!
  - Failure (`med = numbers[length/2]`). Good!

References

- AO, Ch. 7.2.3, 7.3