CSI3105: Software Testing

Lecture 6: Data-flow Testing Recap

Incorporated content from A survey on data-flow testing. In *ACM Computing Surveys* (Vol. 50, Issue 1). https://doi.org/10.1145/3020266

Condensed version

- This is a condensed version of the main lecture with examples taken from the paper "A survey on data-flow testing" by Su et al. (2017)

- Su, T., Wu, K., Miao, W., Pu, G., He, J., Chen, Y., & Su, Z. (2017). A survey on data-flow testing. In *ACM Computing Surveys* (Vol. 50, Issue 1). https://doi.org/10.1145/3020266

Aim

- Data-flow testing (DFT) is a family of testing strategies designed to verify the interactions between each program variable's definition and its uses (Su et al., 2017)
- Su, T., Wu, K., Miao, W., Pu, G., He, J., Chen, Y., & Su, Z. (2017). A survey on data-flow testing. In *ACM Computing Surveys* (Vol. 50, Issue 1). https://doi.org/10.1145/3020266

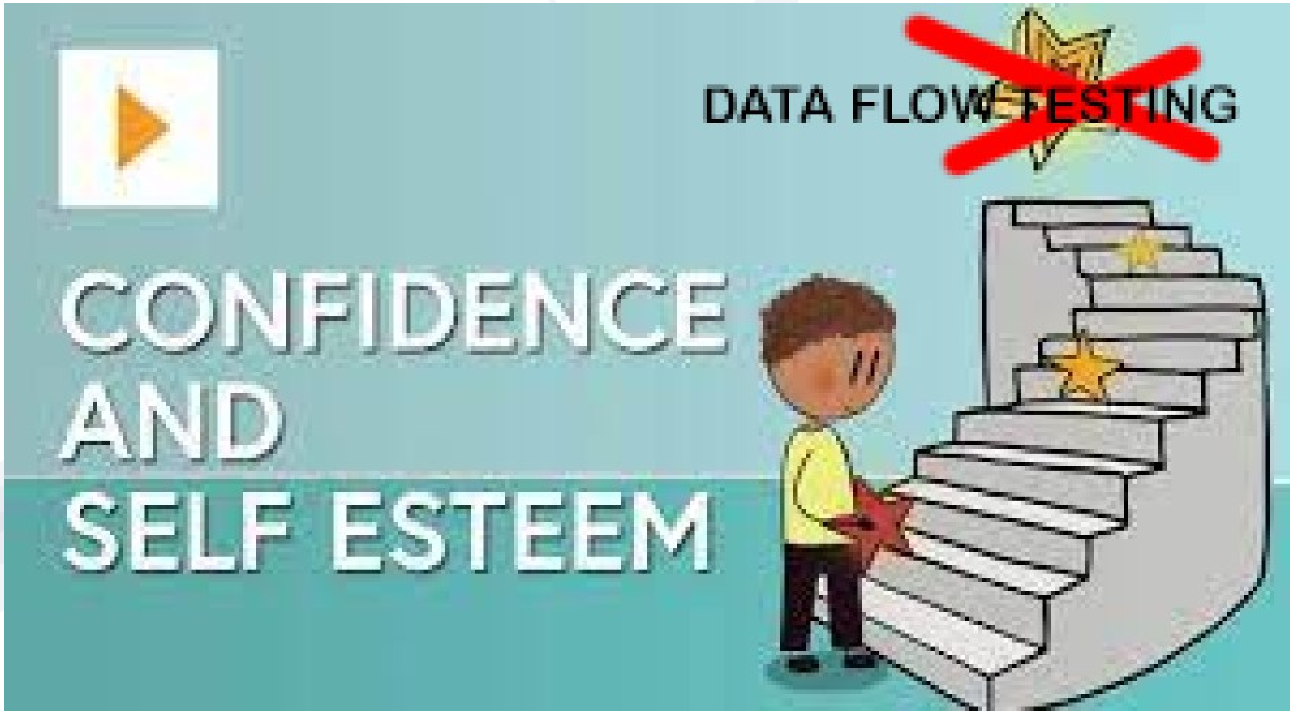
There are two motivations for data flow testing.

- The first is to ensure that the memory location for a variable is accessed in a desirable way. For example, we want to ensure that we have stored a value into that int x location before using it.

- The second motivation is to Verify the correctness of data values "defined. We need to observe that all the "uses" of the value produce the desired results

Aim

– One should not feel confident that a variable has been **assigned the correct value**, if no test causes the execution of a **path** from the point of assignment to a point where the value is **used**.



Process

- Data flow testing is outlined as follows:
  - Draw a data flow graph from a program.
  - Select one or more data flow testing criteria.
    - Define and measure test case adequacy
    - Less redundant tests
  - Identify paths in the data flow graph satisfying the selection criteria.
    - Produces a set of test paths
  - Derive path predicate expressions from the selected paths (Last weeks lecture.)
  - Solve the path predicate expressions to derive test inputs (Last weeks lecture)
    - Outputs a set of test paths and their associated inputs to force the path and the expected output

Def? P-use?

Definition: A variable gets a new value.

- int i = x; YASSSS QWEEEEN

- int I;… no ☹, reserving memory, but not actually defining a value

– Computation use (c-use)
  – Example: x = 2*y;
    » /* y has been used to compute a value of x. */
- - Print(x);
  – C-use, we are USING x to print to console

– Predicate use (p-use)
  – Example: if (y > 100) { …} /* y has been used in a condition. */
  – Used within an if, while, ..any decision point

Def-Use Pairs

```
double power(int x,int y){
      int exp;
      double res;
      if (y>0)
         exp = y;
      else
         exp = -y;
      res=1;
      while (exp!=0){
         res *= x;
         exp -= 1;
      }
      if (y<=0)
         if(x==0)
            abort;
         else
            return 1.0/res;
      return res;
}
```

This code represents a function that will return the power of two ints.

The function takes in two integers x and y and returns the output of x^y.

Def-Use Pairs

```
1  double power(int x,int y){
2       int exp;
3       double res;
4       if (y>0)
5          exp = y;
6       else
7          exp = -y;
8       res=1;
9       while (exp!=0){
10         res *= x;
11         exp -= 1;
12      }
13      if (y<=0)
14         if(x==0)
15            abort;
16         else
17            return 1.0/res;
18      return res;
19 }
```

- What are Def-use pairs?
- The definition of a variable and its use in the functions
  - Computational use
  - Predicate use

- Du pair = (def, use, var)
- A partial example for the variable **res**
- Du1 = (line 8,line 10, res)
- Du2 = (line 8,line 17, res)
- Du3 = (line 8,line 18, res)
- Du4 = (line 10,line 10, res)
- Du5 = (line 10,line 17, res)
- Du6 = (line 10,line 18, res)

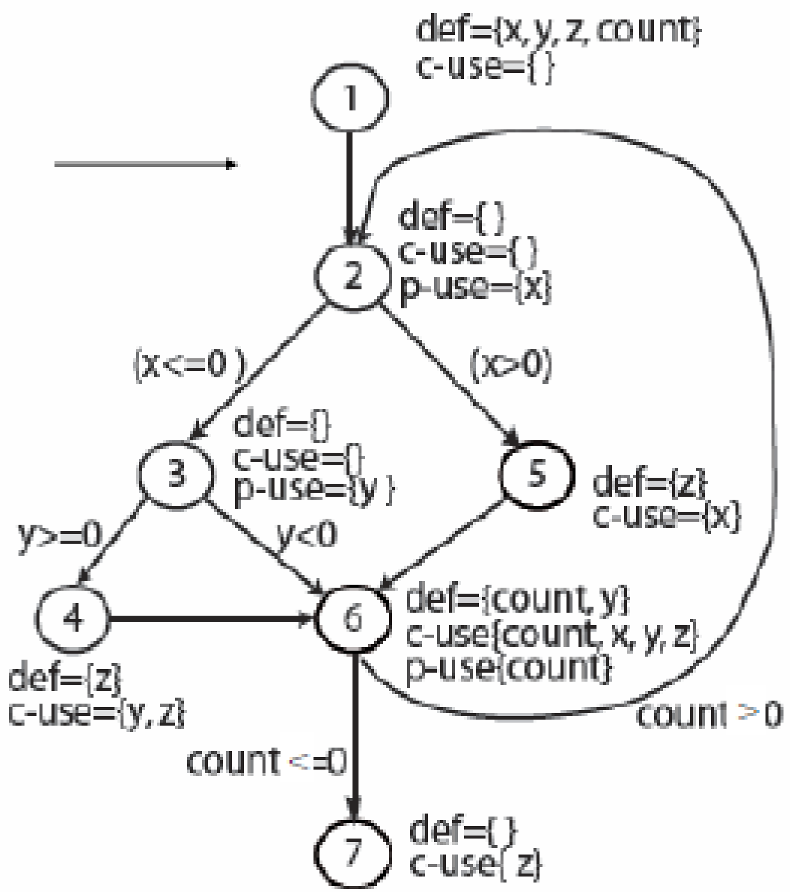
Def-Use Pairs (Make a table)

```
1  double power(int x,int y){
2       int exp;
3       double res;
4       if (y>0)
5          exp = y;
6       else
7          exp = -y;
8       res=1;
9       while (exp!=0){
10         res *= x;
11         exp -= 1;
12      }
13      if (y<=0)
14         if(x==0)
15           abort;
16         else
17           return 1.0/res;
18      return res;
19 }
```

Var (v)	Defined in line (n)	C-use in line	P-use in line
X	1	10	14
Y	1	5,7	4,13
Exp	5	11	9
Exp	7	11	9
Exp	11	Etc..	
res	8	10,17,18	
res	10	10,17,18	

Draw DFG

```
1   begin
2     float x, y, z=0.0;
3     int count;
4     input (x, y, count);
5     do {
6       if (x≤0) {
7         if (y≥0) {
8           z=y*z+1;
9         }
10      }
11      else{
12        z=1/x;
13      }
14      y=x*y+z
15      count=count-1
16    while (count>0)
17    output (z);
18  end
```



Node	Lines
1	1, 2, 3, 4
2	5, 6
3	7
4	8, 9, 10
5	11, 12, 13
6	14, 15, 16
7	17, 18

Choose a criteria

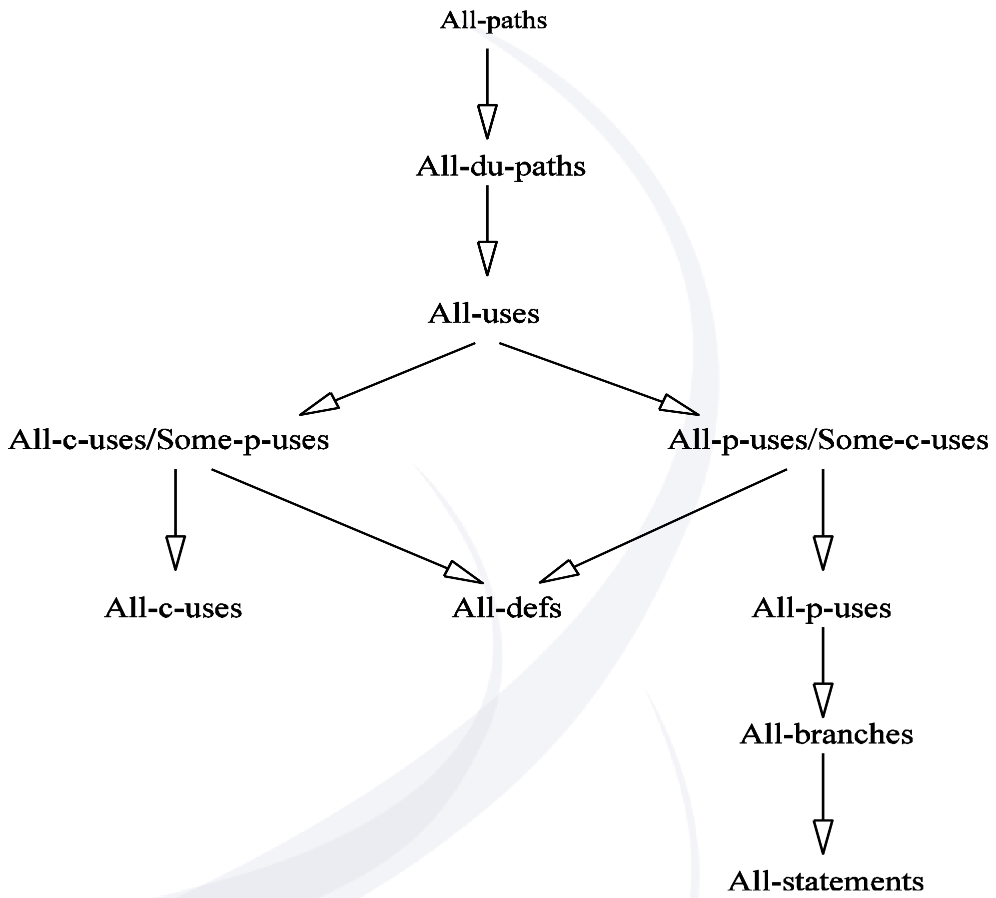
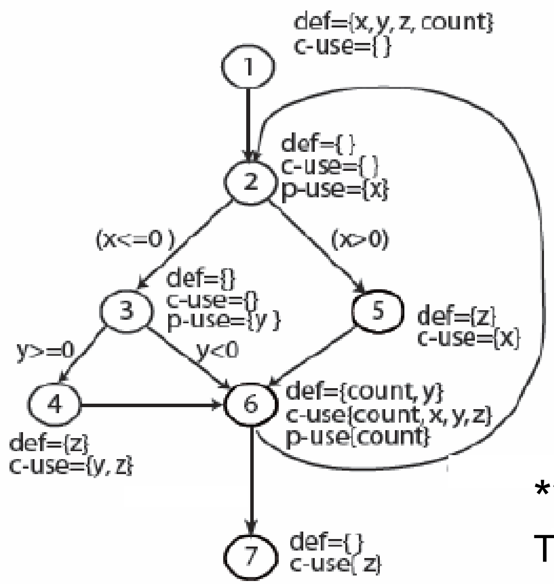


Figure 5.5: The relationship among DF (data flow) testing criteria [6] (©[1988] IEEE).

Build Def-Use Pairs table



A du-path with respect to variable v is a simple path, that is Def-clear from a definition of v to a use of v

Variable (v)	Defined in node (n)	dcu (v, n)	dpu (v, n)
x	1	{5, 6}	{(2, 3), (2, 5)}
y	1	{4, 6}	{(3, 4), (3, 6)}
y	6	{4, 6}	{(3, 4), (3, 6)}
z	1	{4, 6, 7}	{ }
z	4	{4, 6, 7 }	{ }
z	5	{4, 6, 7}	{ }
count	1	{6}	{(6, 2), (6, 7) }
count	6	{6}	{(6, 2), (6, 7) }

****NOTE**

The du path is underlined and bold. The du-path is sitting inside the whole path

All-defs (Each def to at least one use)

X – du (1,5,x) – path = **<u>1,2,5</u>**,6,2,3,6,7

All-c-uses (Each def reaches all c-use)

X – du (1,5,x) – path = **<u>1,2,5</u>**,6,2,3,6,7

X – du (1,6,x) – path = **<u>1,2,3,6</u>**,7

All-uses (Each def reaches all p and c use)

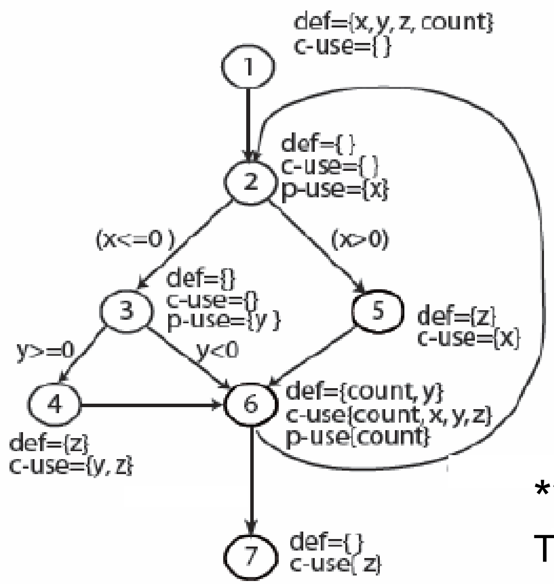
X – du (1,5,x) – path = **<u>1,2,5</u>**,6,2,3,6,7

X – du (1,6,x) – path = **<u>1,2,3,6</u>**,7

X – du (1,(2,3),x) – path = **<u>1,2,3</u>**, 6,7

X – du (1,(2,5),x) – path = **<u>1,2,5</u>**, 6, 7

Def-Use Pairs



A du-path with respect to variable v is a simple path, that is Def-clear from a definition of v to a use of v

Variable (v)	Defined in node (n)	dcu (v, n)	dpu (v, n)
x	1	{5, 6}	{(2, 3), (2, 5)}
y	1	{4, 6}	{(3, 4), (3, 6)}
y	6	{4, 6}	{(3, 4), (3, 6)}
z	1	{4, 6, 7}	{ }
z	4	{4, 6, 7 }	{ }
z	5	{4, 6, 7}	{ }
count	1	{6}	{(6, 2), (6, 7) }
count	6	{6}	{(6, 2), (6, 7) }

****NOTE**

The du path is underlined and bold. The du-path is sitting inside the whole path

All-defs (Each def to at least one use)

y – du (1,4,y) – path = **1,2,3,4**,6,7

y – du (6,4,y) – path = 1,2,3,**6,2,3,4**,6,7

All-uses (Each def reaches all p and c use)

y – du (1,4,y) – path = **1,2,3,4**,6,7

y – du (1,6,y) – path = **1,2,3,6**,7

y – du (1,(3,4),y) – path = **1,2,3,4**,6,7 (dup)

y – du (1,(3,6),y) – path = **1,2,3,6**,7 (dup)

y – du (6,4,y) – path = 1,2,3,**6,2,3,4**,6,7

y – du (6,6,y) – path = 1,2,5,**6,2,3,6**,7

y – du (6,(3,4),y) – path = **a path…**

y – du (6,(3,6),y) – path = **a path…**

All-du-paths (Each def to all possible du-paths)

y – du (1,4,y) – path = **1,2,3,4**,6,7

y – du (1,6,y) – path = **1,2,3,6**,7

y – du (1,6,y) – path = **1,2,5,6**,7 (extra path compared to all-uses)

y – du (1,(3,4),y) – path = **1,2,3,4**,6,7 (dup)

y – du (1,(3,6),y) – path = **1,2,3,6**,7 (dup)

y – du (6,4,y) – path = 1,2,3,**6,5,2,3,4**,6,7

y – du (6,6,y) – path = 1,2,5,**6,5,2,3,6**,7

y – du (6,6,y) – path = 1,2,5,**6,5,2,3,6**,7

y – du (6,(3,4),y) – path = **a path…**

y – du (6,(3,6),y) – path = **a path…**

Data Flow Terms

- We are interested in finding *paths* that include pairs of **definition and use of variables**
- **Global c-use**: A c-use of a variable x in node i is said to be a global c-use if x has been defined before in a node other than node i.
  - Example: The c-use of variable tv in node 9 (Figure 5.4) is a global c-use.

- **Definition clear path**: A path (i – n_1 – … n_m – j), m ≥ 0, is called a definition clear path (def-clear path) with respect to variable x

  from node i to node j, and

  from node i to edge (n_m, j),

  if x has been neither defined nor undefined in nodes n_1 – … n_m.
  - Example: (2 – 3 – 4 – 6 – 3 – 4 – 6 – 3 – 4 – 5) is a def-clear path w.r.t. tv in Fig. 5.4.
  - Example: (2 – 3 – 4 – 5) and (2 – 3 – 4 – 6) are def-clear paths w.r.t. variable tv from node 2 to 5 and from node 2 to 6, respectively, in Fig. 5.4.

Data Flow Terms

- **Global definition**: A node i has a global definition of variable x if node i has a definition of x and there is a def-clear path w.r.t. x from node i to some

  node containing a global c-use, or

  edge containing a p-use of variable x

  . Tv – global def in 2, global c use in 9 (2, 3, 7, 9)
- **Simple path**: A simple path is a path in which all nodes, except possibly the first and the last, are distinct.
  - Example: Paths (2 – 3 – 4 – 5) and (3 – 4 – 6 – 3) are simple paths.
- **Loop-free paths**: A loop-free path is a path in which all nodes are distinct.
- **Complete path**: A complete path is a path from the entry node to the exit node.

Data Flow Terms

- **Du-path**: A path ($n_1 - n_2 - \ldots - n_j - n_k$) is a du-path path w.r.t. variable x if node n_1 has a global definition of x and <u>either</u>
  - node n_k has a global c-use of x and ($n_1 - n_2 - \ldots - n_j - n_k$) is a def-clear simple path w.r.t. x, <u>or</u>
  - Edge (n_j, n_k) has a p-use of x and ($n_1 - n_2 - \ldots - n_j - n_k$) is a def-clear, loop-free path w.r.t. x.

  – Example: Considering the global definition and global c-use of variable tv in nodes 2 and 5, respectively, (2 – 3 – 4 – 5) is a du-path.

  – Example: Considering the global definition and p-use of variable tv in nodes 2 and on edge (7, 9), respectively, (2 – 3 – 7 – 9) is a du-path.

Comparison of Data Flow Testing Criteria

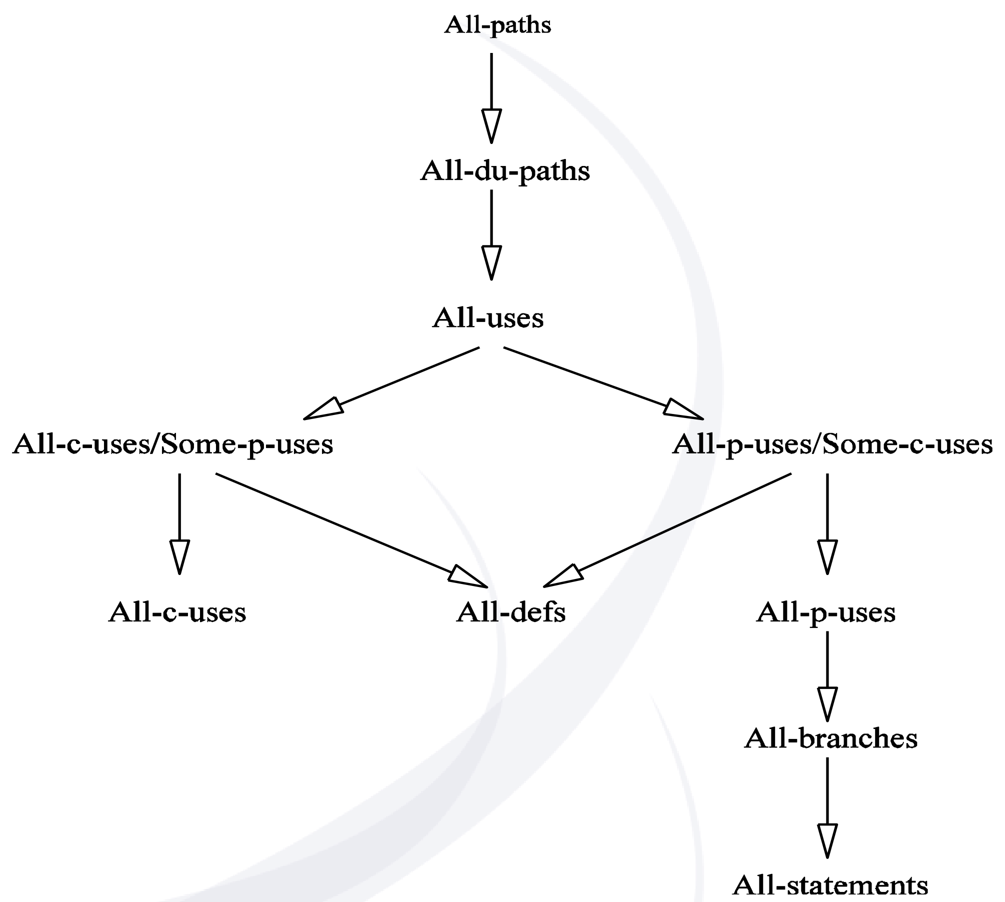


Figure 5.5: The relationship among DF (data flow) testing criteria [6] (©[1988] IEEE).

Once you have your paths

- Path predicate
- Symbolic execution
- Check path feasibility
- Generate test case input and expected output to force the path
- Win (grades)!

References

- Su, T., Wu, K., Miao, W., Pu, G., He, J., Chen, Y., & Su, Z. (2017). A survey on data-flow testing. In *ACM Computing Surveys* (Vol. 50, Issue 1). https://doi.org/10.1145/3020266