6. Test-Adequacy II

Assessment Using Control Flow and Data Flow

Andrea Polini

Software Engineering II – Software Testing
MSc in Computer Science
University of Camerino

January 27^{th}, 2015

MC/DC

- Combinations necessary to satisfy the Multiple Condition Coverage is generally too big.
- MC/DC allows a coverage of all decisions and all conditions avoiding the exponential explosion
- To derive the test set the idea is to identify those tuple which can cover the two criteria without requiring a complete conbinations of values.

Let's consider the compound condition $(C_1 \wedge C_2) \vee C_3$

MC/DC

- Combinations necessary to satisfy the Multiple Condition Coverage is generally too big.
- MC/DC allows a coverage of all decisions and all conditions avoiding the exponential explosion
- To derive the test set the idea is to identify those tuple which can cover the two criteria without requiring a complete conbinations of values.

Let's consider the compound condition $(C_1 \wedge C_2) \vee C_3$

MC/DC

- Combinations necessary to satisfy the Multiple Condition Coverage is generally too big.
- MC/DC allows a coverage of all decisions and all conditions avoiding the exponential explosion
- To derive the test set the idea is to identify those tuple which can cover the two criteria without requiring a complete conbinations of values.

Let's consider the compound condition $(C_1 \wedge C_2) \vee C_3$

Definition of MC/DC coverage

The MC/DC criterion requires that:

- Each block in P has been covered
- Each simple condition in P has taken both `true` and `false` value
- Each decision in P has taken all possible outcomes
- Each simple condition within a compound condition C in P has been shown to independently effect the outcome of C.

Example

Consider a program conceived to satisfy the following requirements:

R_1: Given coordinate position `x`, `y`, and `z`, and a direction value *d*, the program must invoke one of the three functions `fire-1`, `fire-2`, and `fire-3` as per conditions below:

- $R_{1,1}$: Invoke `fire-1` when (`x<y` `and` (`z*z>y`) `and` (prev="East") where *prev* and *current* denote, respectively, the previous and current values of *d*.
- $R_{1,2}$: Invoke `fire-2` when (`x<y`) `and` (`z*z≤y`) `or` (current="South")
- $R_{1,3}$: Invoke `fire-3` when none of the two conditions above is `true`

R_2: The invocation described above must continue until an input Boolean variable become `true`

- let's generate test satisfying the conditions and let's analyze the covered decision

Tracing test cases to requirements

Enhancing a test set we should understand *what portions of the requirements are tested when the program under test is executed against the newly added test case?*

- Trace back test to requirements

Data Flow concepts

- Criteria considered so far are based on the control flow
- it is possible to conceive adequacy criteria based on data flow characteristics

Consider the following program:

```
begin
  int x,y; float z;
  input(x,y);
  z=0;
  if (x!=0) z=z+y;
  else z=z-y;
  if (y!=0) z=z/x // Should be (y!=0 and x!=0)
  else z=z*x;
  output(z);
end
```

An MC/DC test set could not reveal the error while a test set based on definition and usage of variables would have been able

Data Flow concepts

- Criteria considered so far are based on the control flow
- it is possible to conceive adequacy criteria based on data flow characteristics

Consider the following program:

```
begin
  int x,y; float z;
  input(x,y);
  z=0;
  if (x!=0) z=z+y;
  else z=z-y;
  if (y!=0) z=z/x // Should be (y!=0 and x!=0)
  else z=z*x;
  output(z);
end
```

An MC/DC test set could not reveal the error while a test set based on definition and usage of variables would have been able

Data flow criteria

- Data flow criteria based on two main concepts:
  - Definition
  - Use (computational usage - c-use - and predicate usage - p-use)

Definition of Data flow graphs:

1. Compute def_i, $c-use_i$ and $p-use_i$ for each block in P
2. Associate each node i in N with def_i, $c-use_i$ and $p-use_i$
3. For each node i that has a non-empty p-use set and ends in condition C, associate edges (i,j) and (i,k) with C and $!C$

```
begin
  int x,y,z;
  input(x,y); z=0;
  if (x<0 and y<0)    {
    z=x*x;
    if (y>=0) z=z+1; }
  else z=x*x*x;
  output(z);
end
```

Data coverage

- c-use coverage
- p-use coverage
- all-uses coverage

Control-flow vs. Data-flow