**Department of Computing**

**Curtin University**

# Software Engineering Testing (SET)

Week 6 Laboratory/Tutorial

The following exercises are intended to be done in a laboratory/tutorial session with a teaching assistant or instructor present. The exercises have been designed to reinforce concepts taught in SET.

1. Use the following program fragment for questions a-e below.

```
w = x;              // node 1
if (m > 0)
{
   w++;             // node 2
}
else
{
   w=2*w;           // node 3
}
// node 4 (no executable statement)
if (y <= 10)
{
   x = 5*y;         // node 5
}
else
{
   x = 3*y+5;       // node 6
}
z = w + x;          // node 7
```

a) Draw a control flow graph for this program fragment. Use the node numbers given above.
b) Which nodes have defs for variable *w*?
c) Which nodes have uses for variable *w*?
d) Are there any du-paths with respect to variable *w* from node 1 to node 7?
   If not, explain why not. If any exist, show one.
e) Enumerate all of the du-paths for variables *w* and *x*.

2. Use the following method **fmtRewrap( )** for questions a-c below.

```java
/** ********************************************************
 *  Rewraps the string (Similar to the Unix fmt).
 *  Given a string S, eliminate existing CRs and add CRs to the
 *  closest spaces before column N.  Two CRs in a row are considered to
 *  be "hard CRs" and are left alone.
********************************************************** */

static final char CR = '\n';
static final int inWord      = 0;
static final int betweenWord = 1;
static final int lineBreak   = 2;
static final int crFound     = 3;
static private String fmtRewrap (String S, int N)
{
   int state = betweenWord;
   int lastSpace = -1;
   int col = 1;
   int i = 0;
   char c;

   char SArr [] = S.toCharArray();
   while (i < S.length())
   {
      c = SArr[i];
      col++;
      if (col >= N)
         state = lineBreak;
      else if (c == CR)
         state = crFound;
      else if (c == ' ')
         state = betweenWord;
      else
         state = inWord;
      switch (state)
      {
      case betweenWord:
         lastSpace = i;
         break;

      case lineBreak:
         SArr [lastSpace] = CR;
         col = i-lastSpace;
         break;

      case crFound:
         if (i+1 < S.length() && SArr[i+1] == CR)
         {
            i++; // Two CRs => hard return
            col = 1;
         }
         else
            SArr[i] = ' ';
         break;

      case inWord:
      default:
         break;
      } // end switch
      i++;
   } // end while
   S = new String (SArr) + CR;
   return (S);
}
```

a) Draw the control flow graph for the **fmtRewrap( )** method.
b) For **fmtRewrap( )**, find a test case such that the corresponding test path visits the edge that connects the beginning of the **while** statement to the **S = new String(SArr) + CR;** statement **without** going through the body of the while loop.
c) Enumerate the test requirements for node coverage, edge coverage and prime path coverage for the graph for **fmtRewrap( )**.

3. **State Transition Task:**

**Scenario:** A website shopping basket starts out as empty. As purchases are selected, they are added to the shopping basket. Items can also be removed from the shopping basket. When the customer decides to check out, a summary of the items in the basket and the total cost are shown, for the customer to say whether this is OK or not. If the contents and price are OK, then you leave the summary display and go to the payment system. Otherwise you go back to shopping (so you can remove items if you want).

(a) Produce a state diagram (or UML state chart) showing the different states and transitions. Define a test, in terms of the sequence of states, to cover all transitions.
(b) Produce a state table. Give an example test for an invalid transition.

Note: The state table lists all the states down one side of the table and all the events that cause transitions along the top (or vice versa). Each cell then represents a state-event pair. The content of each cell indicates which state the system will move to, when the corresponding event occurs while in the associated state. This will include possible erroneous events – events that are not expected to happen in certain states. These are negative test conditions.