Software Testing

Sample Questions

Question 1:

```
/**
 * Count odd or postive elements
 *
 * @param x array to search
 * @return count of odd/positive values in x
 * @throws NullPointerException if x is null
 */
public static int oddOrPos(int[] x)
{
   int count = 0;
   for (int i = 0; i < x.length; i++)
   {
      if (x[i]%2 == 1 || x[i] > 0)
      {
         count++;
      }
   }
   return count;
}
// test:  x = [-3, -2, 0, 1, 4]; Expected = 3
// Book website: OddOrPos.java
// Book website: OddOrPosTest.java
```

(a) Explain what is wrong with the given code. Describe the fault precisely by proposing a modification to the code.

(b) If possible, give a test case that does **not** execute the fault. If not, briefly explain why not.

(c) If possible, give a test case that executes the fault, but does **not** result in an error state. If not, briefly explain why not.

(d) If possible give a test case that results in an error state, but **not** a failure. Hint: Don't forget about the program counter. If not, briefly explain why not.

(e) For the given test case, describe the first error state. Be sure to describe the complete state.

Question 2:

Derive input space partitioning test inputs for the **BoundedQueue** class with the following method signatures:

public BoundedQueue (int capacity); // The maximum number of elements

public void enQueue (Object X);

public Object deQueue ();

public boolean isEmpty ();

public boolean isFull ();

Assume the usual semantics for a queue with a fixed, maximal capacity. Try to keep your partitioning simple—choose a small number of partitions and blocks.

(a) List all of the input variables, including the state variables.

(b) Define characteristics of the input variables. Make sure you cover all input variables.

(c) Partition the characteristics into blocks.

(d) Define values for each block.

(e) Define a test set that satisfies pair-wise coverage.