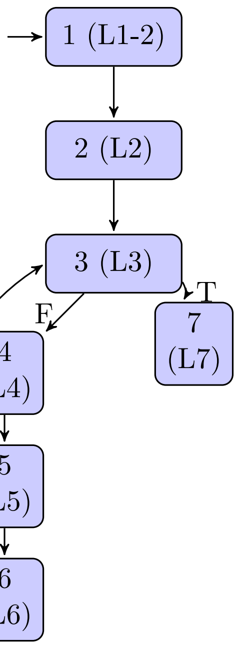
**Software Testing, Quality Assurance and Maintenance** Winter 2019

# Lecture 7 (worksheet) — January 21, 2019

*Patrick Lam* *version 1*

## Active Learning Exercise

Where are the basic blocks?



**if statements:** What's the control flow graph fragment?

```
if (z < 17)
  print (x);
else
  print (y);
```

```
if (z < 17)
  print(x);
```

Short-circuit `if` evaluation is more complicated and is why Jacoco can sometimes give mysterious branch coverage results.

```
if (z < 17 || q > 8)
  print (x);
else
  print (y);
```

**while statements:**

```
x = 0; y = 20;
while (x < y) {
  x ++; y --;
}
```

**for statements:**

```
for (int i = 0; i < 57; i++) {
  if (i % 3 == 0) {
    print (i);
  }
}
```

**Enhanced for loops:**

```
for (Widget w : widgetList) {
  decorate(w);
}
```

**case / switch statements:**

```
switch (n) {
  case `I': ...; break;
  case `J': ...; // fall thru
  case `K': ...; break;
}
```

**Larger examples:**

```
  /** Binary search for target in sorted subarray a[low..high] */
  int binary_search(int[] a, int low, int high, int target) {
    while (low <= high) {
      int middle = low + (high-low)/2;
      if (target < a[middle)
        high = middle - 1;
      else if (target > a[middle])
        low = middle + 1;
      else
        return middle;
    }
    return -1; /* not found in a[low..high] */
  }
```

```
/* effects: if x==null, throw NullPointerException
            otherwise, return number of elements in x that are odd, positive or both. */
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;
}

// example test case: input: x=[-3, -2, 0, 1, 4]; output: 3
```

Finally, we have a really poorly-designed API (I'd give it a D at most, maybe an F) because it's impossible to succinctly describe what it does. **Do not design functions with interfaces like this.** But we can still draw a CFG, no matter how bad the code is.

```
/** Returns the mean of the first maxSize numbers in the array,
    if they are between min and max. Otherwise, skip the numbers. */
double computeMean(int[] value, int maxSize, int min, int max) {
  int i, ti, tv, sum;

  i = 0; ti = 0; tv = 0; sum = 0;
  while (ti < maxSize) {
    ti++;
    if (value[i] >= min && value[i] <= max) {
      tv++;
      sum += value[i];
    }
    i++;
  }
  if (tv > 0)
    return (double)sum/tv;
  else
    throw new IllegalArgumentException();
}
```