# CITS5501 Software Testing and Quality Assurance
# CITS5501
# Semester 1, 2018
# Workshop 3 – Whitebox testing

## 1. Control flow

Consider the following Java method for collapsing sequences of blanks, taken from the `StringUtils` class of Apache Velocity (http://velocity.apache.org/), version 1.3.1.

```
1  /**
2  * Remove/collapse multiple spaces.
3  *
4  * @param String string to remove multiple spaces from.
5  * @return String
6  */
7  public static String collapseSpaces(String argStr) {
8    char last = argStr.charAt(0);
9    StringBuffer argBuf = new StringBuffer();
10
11   for (int cIdx = 0 ; cIdx < argStr.length(); cIdx++) {
12     char ch = argStr.charAt(cIdx);
13     if (ch != ' ' || last != ' ') {
14       argBuf.append(ch);
15       last = ch;
16     }
17   }
18   return argBuf.toString();
19 }
```

1. Construct a control flow graph of the method. How many nodes are required? How many edges?
2. Can you construct test cases that execute the loop ...
   ... zero times?
   ... once?
   ... more than once?

3. What sort of coverage do your test cases have? Do they have . . .
   . . . node coverage?
   . . . statement coverage?
   . . . edge coverage?

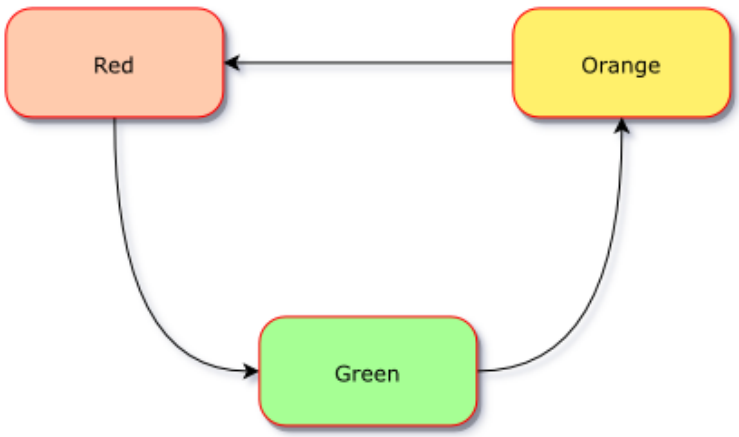
4. What are the prime paths in the graph? (Reminder: a simple path never re-visits the same node, except that the first and last nodes may be the same. A prime path is a simple path that is not a proper subpath of any other simple path – i.e., it's a sort of "maximal simple path".)

   Do your tests have prime path coverage?

## 2. State diagrams

State diagrams are used to give an abstract description of the behaviour of a stateful system. *States* are represented by nodes in a directed graph, and permissible *transitions* between states are represented by edges.

For example, the following state diagram could be used to represent a (very simple) traffic light system[1]:

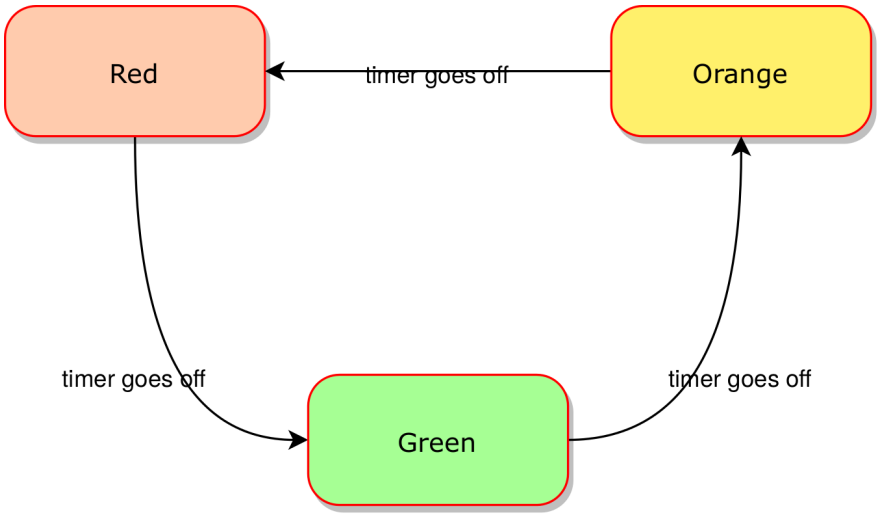



This diagram doesn't represent any information about when the traffic light changes state, or under what circumstances – just what the valid *state transitions* are.

In most systems, we want to represent what *events* will *trigger* transitions. (These events might arise from within the system – e.g., a timer going off – or from outside – e.g. a button being pressed by a user.) We can indicate events using labels on the node edges. (This is a simplified version of the syntax used for UML state charts.)

[1]Bonus question: how hard do you think it is to hack into a traffic light system? (See https://www.csoonline.com/article/2466551/microsoft-subnet/hacking-traffic-lights-with-a-laptop-is-easy.html.) Is this a scenario that could have been identified by any test case design techniques we have looked at?

Suppose for our traffic light system that an internal timer goes off every 10 seconds, making the system change state. (This is a *very* fast-changing traffic light. Perhaps the designers hate all pedestrians and drivers.) We could represent that as follows:




1. Draw a state diagram for a lift (or elevator, in American). The lift services a building with two floors. On each floor is a single button which call calls the lift to that floor. On the inside of the lift are two buttons, one which directs the lift to go to the ground floor (the "first floor", in American), and another which directs the lift to go the first floor (the "second floor", in American).

   What event types will be required? What states will be required?

2. We can also add *guards* on each edge, which limit the circumstances in which a transition will occur. These are added in square brackets after the event name.

   Are any guards needed for our lift system?

3. If we define test cases for this system, what will be the inputs? And what will be the outputs?

   Can you define a set of test cases which give *edge coverage* of the lift system?