Tutorial 10
System & Object-oriented Testing

Week 10, 2023

Please do try the questions before coming to the tutorial. Your active participation is the most important!

1. Assume you are given the task of develop a project *ProList*, with the goal to create a simple Web application for proposal management, dealing with `User`s, `Proposal`s and `AuditLog`s.

   Assume the handling of accessing proposal details is implemented with a number of additional objects. With some simplification, the loading of the `Proposal` page is shown below in Figure 1. The component **Router** is responsible for routing HTTP requests to the appropriate *resource*s that handles them. As the name suggests, `ProposalResource` is responsible for creating pages to present proposal details. Once authentication and authorisation are determined to be required, the resource consults the `LoginManager` to grant/refuse access based on user input. The `LoginManager` in turn invokes `UserDAO` to load user details given the user name. If access is granted, the resource then invokes `ProposalDAO` to obtain proposal details. If access control is not required, or that the user is denied access, `ProposalResource` constructs a page correspondingly. Finally, `ProposalResource` returns the page back to the `Router`.

   Note that the graphs within each component represent greatly simplified program graphs, with only some important control structures preserved. Nodes in these graphs don't represent individual statements, but rather blocks of statements that can be grouped together logically. The *italic label* beside each node explains the main functionality of that node. Edges still represent control flow. Thick edges represent transfer of control (message passing and return) between components.

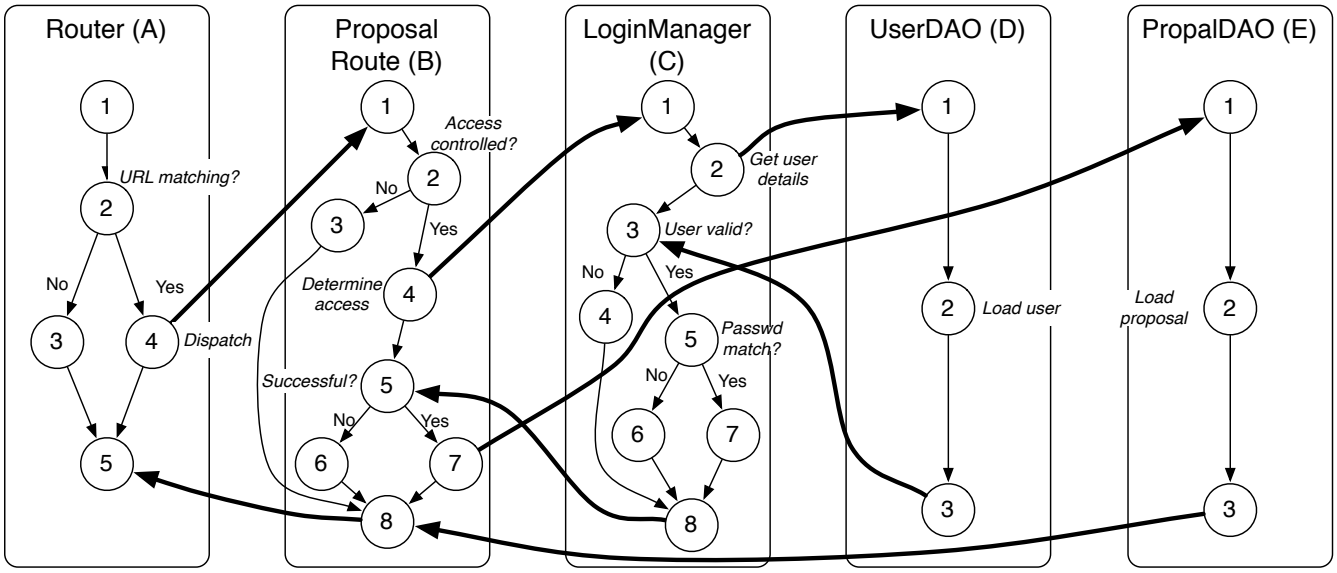

Figure 1: The interactions of some components of the *ProList* system that handle proposal presentation.

(a) Draw a sequence diagram depicting the interactions between the above objects. Note that for simplicity reasons, do not include model objects such as `User` and `Proposal`.

2. Finite state machines can be very useful in testing by capturing system functionality in terms of states, events, transitions, actions and guards.

   Suppose we want to implement a Web service that allows users to create proposals programmatically. A user will specify the user name, password and the details of the proposals to be created. The system creates the proposal upon successfully validating the provided information. A *timeout* function that puts a limit on how long the service can take to create an proposal will also be implemented. That is, the system only waits for a certain amount of time to receive user inputs. When the time is up, the system will go back to the state where it is ready to accept another request.

   Draw a system-level finite state machine depicting this scenario. If you are thinking of input validation, *do not* break it down into validation of different inputs now. Since the system is a long-running system, we will assume that there is no final state. That is, after an proposal is successfully created, the system goes back to the state where it is ready to create another proposal.

Solution:

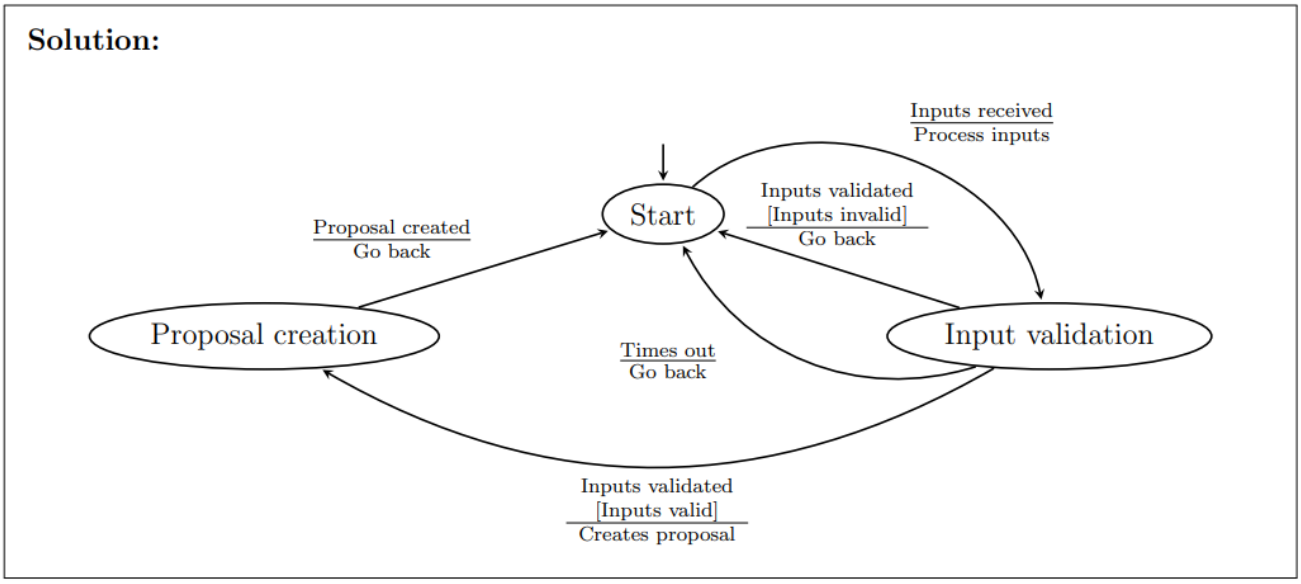

Figure 2

Object-Oriented Testing/ Metric The Java class below contains a number of methods. The method `fizzBuzz` takes a `String` representation of an integer as parameter and returns itself (as a `String`) or some other `String` values. The method `fizzBuzzRange` takes two integers and performs `FizzBuzz` on all the integers within the range.

Listing 1 below shows, on two pages, the class `Foo` in Java. Answer question 3 based on the `Foo` class.

Listing 1: The Java class `Foo`.

```
1  public class Foo {
2
3      private int min;
4      private int max;
5
6      public String fizzBuzz(String input) {
7          int x = Integer.parseInt(input);
8
9          boolean hasFizz = x % 3 == 0;
10         boolean hasBuzz = x % 5 == 0;
11         if (hasFizz && hasBuzz)
12             return "FizzBuzz";
13         else if (hasFizz)
14             return "Fizz";
15         else if (hasBuzz)
16             return "Buzz";
17         else
18             return input;
19     }
20
21     public String[] fizzBuzzRange(int low, int high) {
22         if (low <= 1)
23             throw new IllegalArgumentException("low should be >= 1");
24         else if (high > 100)
25             throw new IllegalArgumentException("high should be <= 100");
26
27         String[] result = new String[high - low + 1];
28         for (int i = low; i <= high; i++)
29             result[i - low] = fizzBuzz(Integer.toString(i));
30
31         return result;
32     }
```

(Continued overleaf)

```
33      private void findMinMax(int[] array) {
34          min = array[0];
35          max = array[0];
36
37          for (int i : array) {
38              if (min > i)
39                  min = i;
40              else if (max < i)
41                  max = i;
42          }
43      }
44
45      public int[] unique(int[] array) {
46          findMinMax(array);
47
48          boolean[] set = new boolean[max - min + 1];
49          for (int i : array)
50              set[i - min] = true;
51
52          int size = 0;
53          for (boolean i : set) {
54              if (i)
55                  size++;
56          }
57          int[] result = new int[size];
58          int j = 0;
59          for (int i = 0; i < set.length; i++) {
60              if (set[i])
61                  result[j++] = i + min;
62          }
63          return result;
64      }
65
66      public int maxOccurrences(int[] array) {
67          findMinMax(array);
68
69          int[] set = new int[max - min + 1];
70          for (int i : array)
71              set[i - min]++;
72
73          max = set[0];
74          for(int i : set) {
75              max = max < i ? i : max;
76          }
77          return max;
78      }
79  }
```

3. The quality of classes in object-oriented languages such as Java can be measured by different object-oriented metrics. Some metrics measure the *lack of cohesion* of a class based on interactions between its methods and attributes. Answer the following questions on the *cohesion* of the class `Foo`.

   (a) Specifically, the metric LCOM1 is defined as

$$LCOM1 = \begin{cases} P - Q, & \texttt{for } P > Q, \\ 0 & \texttt{otherwise} \end{cases}$$

   where for each pair of different methods (order of methods irrelevant), P is incremented by 1 if they do not access any common attribute, otherwise Q is incremented by 1. The initial values of P and Q are both 0.

   Compute the $LCOM1$ value for the class `Foo`. Include all its methods (`public` and `private`) in your calculation.

Solution:
Basically $LCOM1 = \binom{n}{2} - 2 * e$, where n is the number of methods, hence $\binom{n}{2}$ represents the number of pairs of methods (irrespective of order), and e is the number of pairs of methods that share *some* attribute of the class. In other words, $P = \binom{n}{2} - e$ and $Q = e$.
For class `Foo`, $n = 5$ and $e = 3$ (`findMinMax`, `unique` and `maxOccurrences` all access attributes `min` and `max`). Hence, $LCOM1 = \binom{5}{2} - 2 * 3 = 4$.

Figure 3

   (b) $LCOM2$ is another lack of cohesion metric for classes. Specifically, $LCOM2$ is defined as

$$LCOM2 = 1 - \frac{\sum_{\text{for each class attribute} A} \#m_A}{m * a}$$

   where m is the number of methods, a is the number of attributes, and $\#m_A$ is the number of methods that access a particular attribute A.

   Compute the $LCOM2$ value for the class `Foo`. Include all its methods (`public` and `private`) in your calculation.

Solution:
`Foo` has 2 attributes ($a = 2$): `min` and `max`, each of which is accessed by 3 methods (`findMinMax`, `unique` and `maxOccurrences`: $\#m_{min} = \#m_{max} = 3$. `Foo` has 5 methods in total ($m = 5$). Hence, $LCOM2 = 1 - \frac{3+3}{2*5} = 0.4$.

Figure 4