Software Testing and Validation

**Project Report - 2019-2020**

Group 10 - João Freitas (87671), Diogo Faustino (97081)

**Test cases for computeCreditBill method**

To test this method, we applied the Combinational Functional Test Pattern because of the complex logic behind de process of choosing the discount value.

We elaborated a decision tree:

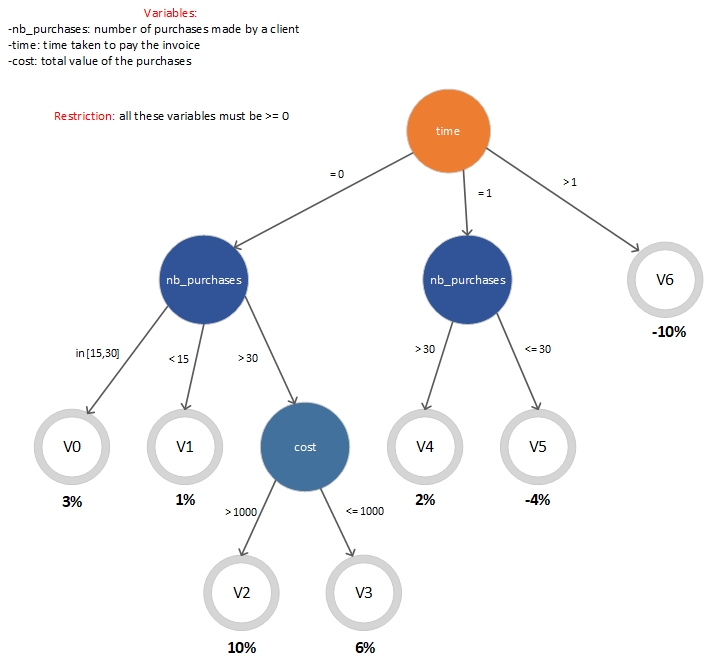


Fig. 1 – Decision tree regarding discount value

Boundary conditions for each variant:

* V0 -> time = 0 & 15 <= nb\_purchases <= 30
* V1 -> time = 0 & nb\_purchases < 15
* V2 -> time = 0 & nb\_purchases > 30 & cost > 1000
* V3 -> time = 1 & nb\_purchases > 30 & cost <= 1000
* V4 -> time = 1 & nb\_purchases > 30
* V5 -> time = 1 & nb\_purchases <= 30
* V6 -> time > 1

**Domain matrixes for variants**















**Description of the test cases**

* In total we have 21 test cases
* We made a domain matrix for each variant in order to exercise all the branches in the graph. In the matrix, each row represents a set of input values and each column a valid or invalid combination of instance variables
* For all conditions we have one On point and one OFF point, except for the cases the condition is an equality. For those cases we have one ON point and two OFF points
* The expected results marked with an X are test cases that contain an invalid value for time (-1) which isn’t supposed to happen because the time variable must be >= 0. As it was said in the project description, the expected result for these test cases is that they throw an InvalidOperationException exception.
* The expected results marked with a variant number are test cases that belong to another variant, so we don’t need to repeat them

**Test cases for PostOffice class**

We identified the type of this class as non-modal because the constraints are not related to the history nor the message sequences. As a result, we applied the Non-modal Test Pattern.

We started by identifying the class invariant by analyzing the restrictions:

* It is impossible to have two products with the same name registered in the same post office **(for any p1,p2 in PostOffice.products, p1.name = p2.name => p1 = p2)**
* The total amount of products presented at a post office cannot exceed a given threshold(…) This maximum number of products can vary between 2 and 20 and it is specified when you create a post office **(for each PostOffice as po, po.products.size() <= po.maxNumberOfProducts & 2 <= po.maxNumberOfProducts <= 20)**
* The unit price and the number of units of a product cannot be a negative number **(for each p in PostOffice.products, p.price >=0 & p.quantity >= 0)**

**PostOffice class invariant:** for any p1,p2 in PostOffice.products, p1.name = p2.name => p1 = p2 & for each PostOffice as po, po.products.size() <= po.maxNumberOfProducts & 2 <= po.maxNumberOfProducts <= 20 & for each p in PostOffice.products, p.price >=0 & p.quantity >= 0

**Domain matrix for PostOffice class**



**Description of the test cases**

* In total we have 12 test cases
* In the matrix, each row represents a set of input values and each column a valid or invalid combination of instance variables (A - accepted, R - rejected)
* We implemented the following test cases: TC1(T,2,4,1,1), TC2(F,3,5,2,2), TC3(T,11,13,15,3), TC4(T,12,14,25,4), TC5(T,2,2,40,5) and TC6(T,5,1,45,6)