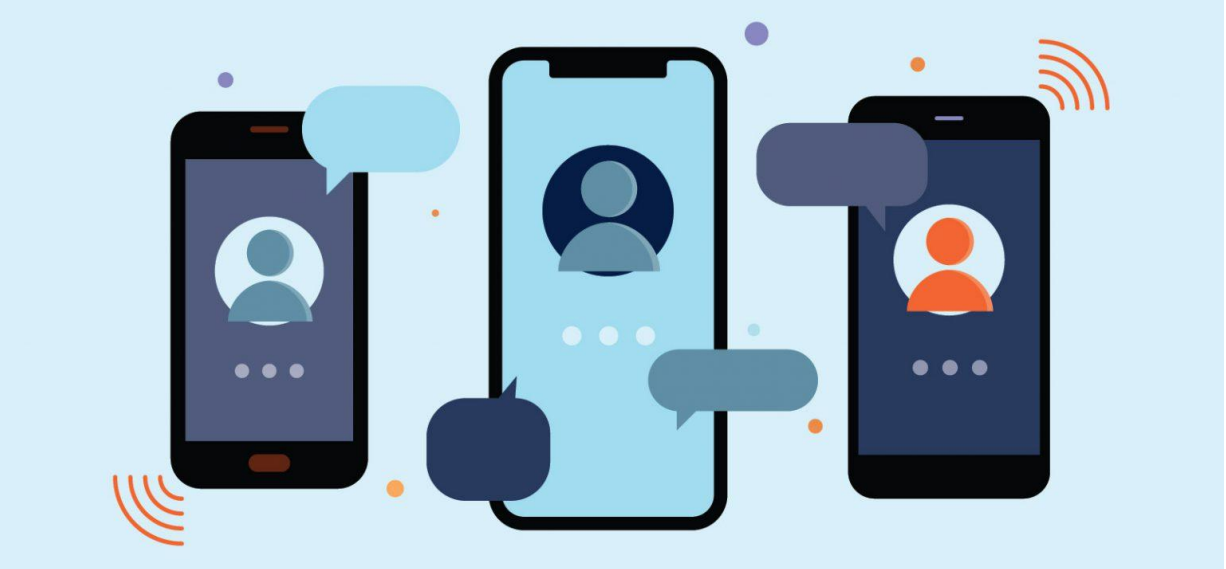
TÉCNICO LISBOA

Software Testing and Validation 2022/2023

Instituto Superior Técnico

Project



Violaine BEC	107810
Miguel Alves	95650
Jean-Simon HOULE	107676

Project System Description

The system to test it is a manager of a network of communication terminals. Generally, the program manages the clients, terminals, and communications of the network. Specifically, the system provides several services to its users, including: registering clients; registering terminals; registering data about communications made; searching for communications made, and accounting for the balance associated with terminals. The main entities of this system are: TerminalNetwork, Terminal, and Client.

Class scope test cases for the TerminalNetwork class:

About TerminalNetwork class

The network of terminals is represented by the TerminalNetwork class. The network maintains the list of clients and terminals registered on the network. A terminal network has a maximum number of clients, which is indicated at the time of creation and can be changed later. The number of clients cannot exceed 50000, and the name of each client is a unique identifier within the context of the terminal network. A terminal network has a name, and the number of characters in the name must be greater than or equal to 3 and less than 10. Each terminal in a terminal network is associated with a client registered on the same network. If the invocation of one of the methods of the TerminalNetwork class invalidates any of these conditions, the method in question should have no effect and should throw the InvalidInvocationException exception.

Choose of a test pattern

For this class scope, we will use the **Invariant Boundaries.**

Step 1: list of class invariants

- A terminal network has a maximum number of clients:
  nbOfClients ≤ maxClients ≤ 50000
- For each client c1 in terminal network its identifier is unique:
  ∀ c1, c2 in a terminal network, if c1.id = c2.id => c1 = c2: **cond 2**
- Each terminal network has a name, the length of this name must be greater than or equal to 3 and less than 10: name.length ≥ 3 and name.length < 10
- Each terminal in a terminal network is associated with a client registered on the same network: **cond 4**

Step 2: Domain matrix

Test case			1	2	3	4	5	6	7	8	9	10	11	12
nbOfClients	≤ maxClients	On	10000											
		Off		10501										
	Typical	In			500	750	1000	1100	1200	1300	1400	1500	1600	1700
maxClients	≤ 50000	On			50000									
		Off				50001								
	Typical	In	10000	10500			10000	11000	12000	13000	14000	15000	16000	17000
client.id	cond 2	On					T							
		Off						F						
	Typical	In	T	T	T	T			T	T	T	T	T	T
length of name	≥ 3	On							3					
		Off								2				
	< 10	On									10			
		Off										9		
	Typical	In	4	5	5	6	6	7					8	4
terminal.client	cond 4	On											T	
		Off												F
	Typical	In	T	T	T	T	T	T	T	T	T	T		
Expected result			V	X	V	X	V	X	V	X	X	V	V	X

Conclusion

Test case	nbOfClients	maxClients	client.id	TerminalNetwork name	terminal.client	Expected result
1	10000	10000	cond2 = T	efgh	cond 4 = T	
2	10501	10500	cond2 = T	abcde	cond 4 = T	InvalidInvocationException
3	500	50000	cond2 = T	abcde	cond 4 = T	
4	750	50001	cond2 = T	abcde	cond 4 = T	InvalidInvocationException
5	1000	10000	cond2 = T	abcdef	cond 4 = T	
6	1100	11000	cond2 = F	abcdefg	cond 4 = T	InvalidInvocationException
7	1200	12000	cond2 = T	abc	cond 4 = T	
8	1300	13000	cond2 = T	ab	cond 4 = T	InvalidInvocationException
9	1400	14000	cond2 = T	abcdefghij	cond 4 = T	InvalidInvocationException
10	1500	15000	cond2 = T	abcdefghi	cond 4 = T	
11	1600	16000	cond2 = T	abcdefgh	cond 4 = T	
12	1700	17000	cond2 = T	abcd	cond 4 = F	InvalidInvocationException

Implemented Test Cases

NB: Actually, we don't really need test cases 3, 5 and 11 because condition true is already tested in the other test cases.

You can see the implementation of these test cases in the java file.

Class scope test cases for the Terminal class:

About Terminal class

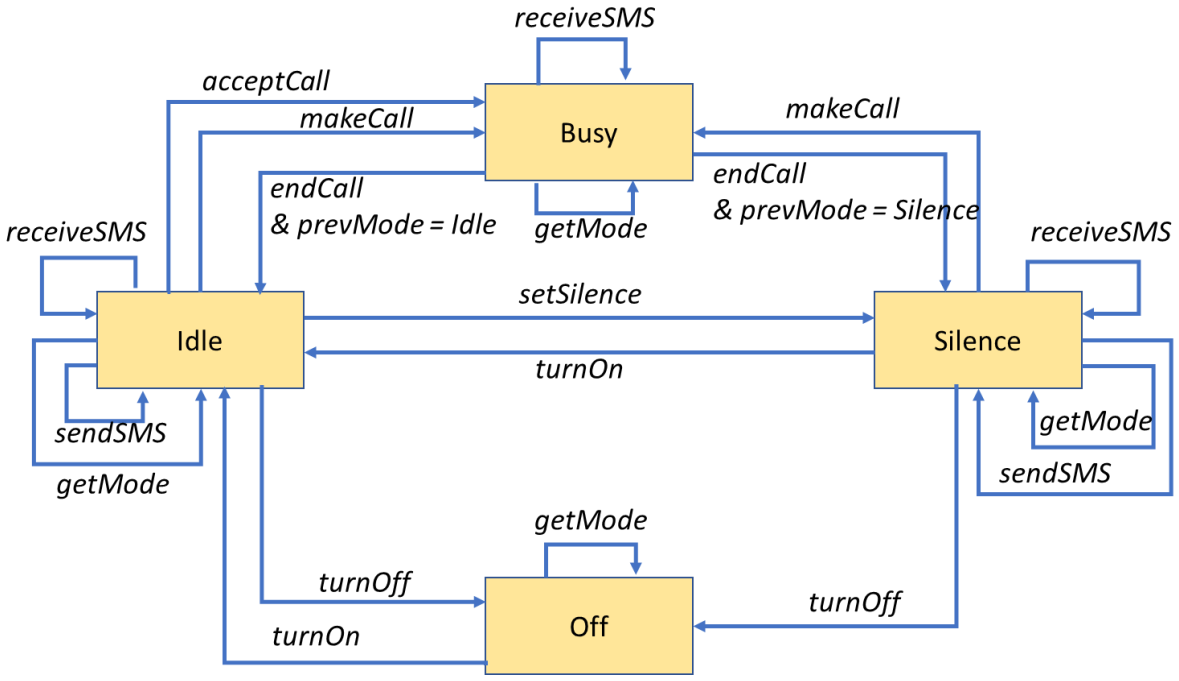
A terminal can make a call with another terminal or send and receive SMS messages. Each terminal stores the SMS messages sent and received successfully. A terminal also keeps information regarding voice communications (calls) initiated by the terminal and successfully established. A terminal can be in various modes: off or on. When on, it can be idle (idle), in silence (silence), or busy (busy). A terminal that is on but it is not busy can send SMS messages and make calls. The big difference between an idle terminal and a silent terminal is that the latter does not accept calls. A terminal that is on can receive SMS messages. Consider that any invocation of a method of this class in an undescribed situation corresponds to an invalid invocation and should result in the InvalidInvocationException exception being thrown.

When a terminal is created, it is turned off. A terminal can be turned on by invoking the turnOn() method, which puts it into the idle mode. A terminal in the idle or silence mode can be turned off by invoking the method turnOff(). A non-busy turned-on terminal can send an SMS to another turned-on terminal through the sendSMS method. The return value of this method indicates whether the SMS was successfully delivered or not. The delivery of an SMS to the destination terminal is handled by the method receiveSMS, which is valid only if the terminal in question is turned on. A non-busy turned on terminal can also make a call to another terminal through the boolean makeCall method, provided that the destination terminal is in the idle mode. In this case, communication is established between both terminals, and the calling terminal goes into the busy mode. An idle terminal can accept an incoming call through the acceptCall method, which puts it into the busy mode. The call ends when the endCall method is invoked on both terminals and each terminal returns to the previous mode. Finally, it is possible to put an idle terminal into the silence mode by invoking the method setSilence, and put a terminal in the silence mode back into the idle mode through the turnOn method. At any time, it is possible to know the mode of the terminal through the getMode method.

Choose of a test pattern

For this class scope we will use the **FSM based testing.** In fact, the terminal class has a certain number of finite states.

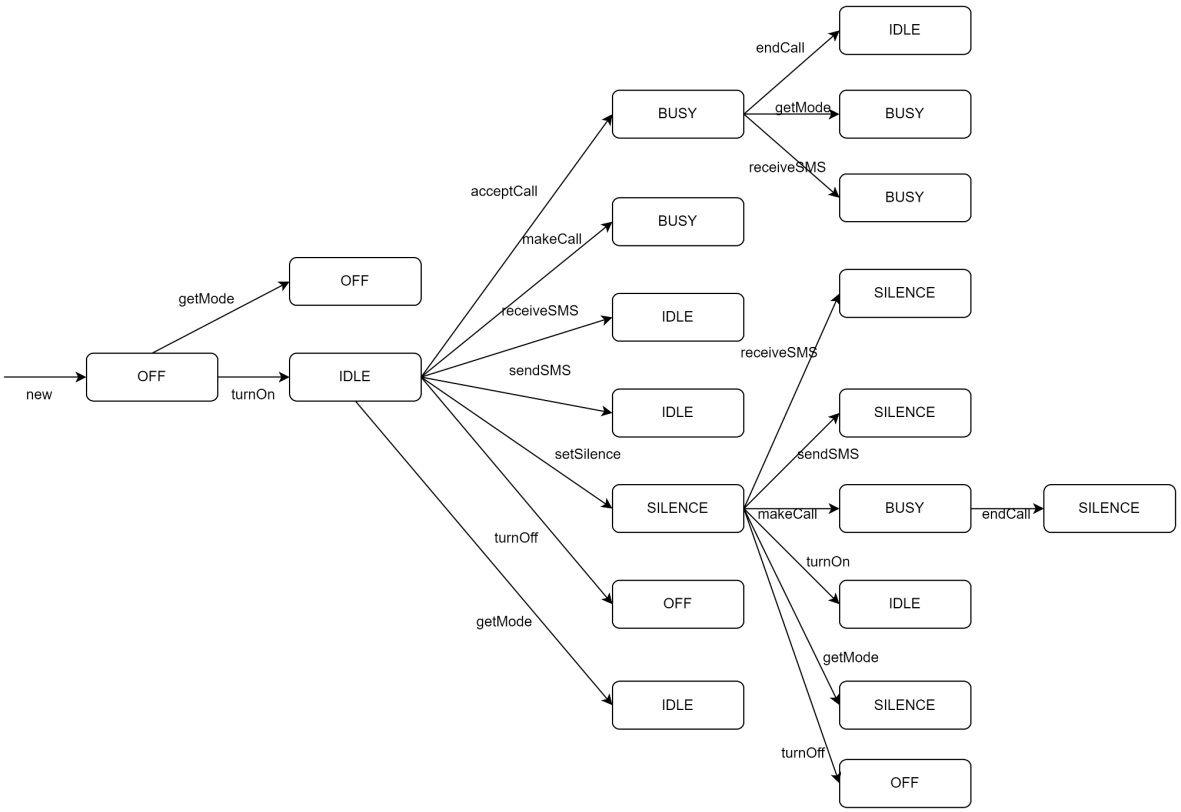
Step 1: Generating state model for CUT



Step 2: Expand the guards of transitions with conditions (pre or post)

State	Message	Condition	Next State
Busy	endCall	Pre: Mode = Idle	Idle
Busy	endCall	Pre: Mode = Silence	Silence

Step 3: Generate transition tree



Step 4: Generate conformance test suite

Run	Test Run / Event Path					Expected Terminal State	Exception
	Level 1	Level 2	Level 3	Level 4	Level 5		
1	new					Off	--
2	new	getMode				Off	--
3	new	turnOn				Idle	--
4	new	turnOn	getMode			Idle	--
5	new	turnOn	acceptCall			Busy	--
6	new	turnOn	acceptCall	getMode		Busy	--
7	new	turnOn	acceptCall	endCall		Idle	--
8	new	turnOn	acceptCall	receiveSMS		Busy	--
9	new	turnOn	makeCall			Busy	--
10	new	turnOn	receiveSMS			Idle	--
11	new	turnOn	sendSMS			Idle	--
12	new	turnOn	setSilence			Silence	--
13	new	turnOn	setSilence	getMode		Silence	
14	new	turnOn	setSilence	receiveSMS		Silence	--
15	new	turnOn	setSilence	SendSMS		Silence	--
16	new	turnOn	setSilence	makeCall		Busy	--
17	new	turnOn	setSilence	makeCall	endCall	Silence	--
18	new	turnOn	setSilence	turnOff		Off	--
19	new	turnOn	setSilence	turnOn		Idle	--
20	new	turnOn	turnOff			Off	--

Step 5: Develop a sneak path test suite (transition table)

Events	States			
	Off	Idle	Silence	Busy
turnOff	PSP	✔	✔	PSP
turnOn	✔	PSP	✔	PSP
setSilence	PSP	✔	PSP	PSP
sendSMS	PSP	✔	✔	PSP
receiveSMS	PSP	✔	✔	✔
makeCall	PSP	✔	✔	PSP
acceptCall	PSP	✔	PSP	PSP
endCall	PSP	PSP	PSP	✔
getMode	✔	✔	✔	✔

✔: Valid transition PSP = Possible sneak path

Step 6: Develop sneak path test suite

Run	Test Run / Event Path				Expected State	Exception
	Level 1	**Level 2**	**Level 3**	**Level 4**		
21	new	turnOff			Off	*InvalidInvocationException*
22	new	turnOn	makeCall	turnOff	Busy	*InvalidInvocationException*
23	new	turnOn	turnOn		Idle	*InvalidInvocationException*
24	new	turnOn	makeCall	turnOn	Busy	*InvalidInvocationException*
25	new	setSilence			Off	*InvalidInvocationException*
26	new	turnOn	setSilence	setSilence	Silence	*InvalidInvocationException*
27	new	turnOn	makeCall	setSilence	Busy	*InvalidInvocationException*
28	new	sendSMS			Off	*InvalidInvocationException*
29	new	turnOn	makeCall	sendSMS	Busy	*InvalidInvocationException*
30	new	receiveSMS			Off	*InvalidInvocationException*
31	new	makeCall			Off	*InvalidInvocationException*
32	new	turnOn	makeCall	makeCall	Busy	*InvalidInvocationException*
33	new	acceptCall			Off	*InvalidInvocationException*
34	new	turnOn	setSilence	acceptCall	Silence	*InvalidInvocationException*
35	new	turnOn	makeCall	acceptCall	Busy	*InvalidInvocationException*
36	new	endCall			Off	*InvalidInvocationException*
37	new	turnOn	endCall		Idle	*InvalidInvocationException*
38	new	turnOn	setSilence	endCall	Silence	*InvalidInvocationException*

Method scope test cases for the sendSMS method:

About sendSMS method

This method is implemented in the Terminal class. A non-busy turned-on terminal can send an SMS to another turned-on terminal through the sendSMS(String msg, Terminal t) method. The return value of this method indicates whether the SMS was successfully delivered or not. If the length of the SMS is valid, and the destination terminal is turned on, the method returns true. If the length of the SMS is valid but the destination terminal is invalid, the method returns false. Finally, if the length of the SMS is invalid, then this method throws the IllegalArgumentException. An SMS can only be sent if its length is between 10 and 200 characters.

Choose of a test pattern

This method is not recursive and not polymorphic. The logic of the method is not too complex. In this way we choose the **Category-Partition** test pattern.

Step 1: Functions of the method

The different functions of the method can be resumed by:

1. If length of the SMS is valid and destination is turned on then return true
2. If length of the SMS is valid but destination terminal invalid then return false
3. If the length of the SMS is invalid then throws an exception IllegalArgumentException

Step 2: Inputs and Outputs of each functions

Function	Inputs	Outputs
1	length of SMS, destination terminal	returned value (true)
2	length of SMS, destination terminal	returned value (false)
3	length of SMS	exception

Step 3: Partition the domain of each inputs

	Categories	Choices
length of SMS	valid (in [10, 200])	10, 200, 105
	invalid [error]	9, 201
destination terminal	valid (could receive SMS)	turned on (idle), turned on (busy), turned on (silence)
	invalid (couldn't receive SMS)	turned off

Step 4: Restrict the test cases

At this point, we should have 20 test cases. However some categories are incompatible between them. If the length of the SMS in invalid then it is not necessary to test it with all the different choices of the destination terminal, we just have to test it once. It leads us to 14 test cases.

Step 5: Test cases / conclusion

	Inputs		**Outputs**	
Test case	length of SMS	destination terminal	returned value	exception
1	10	turned on (idle)	true	-
2	10	turned on (busy)	true	-
3	10	turned on (silence)	true	-
4	10	turned off	false	-
5	200	turned on (idle)	true	-
6	200	turned on (busy)	true	-
7	200	turned on (silence)	true	-
8	200	turned off	false	-
9	105	turned on (idle)	true	-
10	105	turned on (busy)	true	-
11	105	turned on (silence)	true	-
12	105	turned off	false	-
13	9	turned on (idle)	-	IllegalArgumentException
14	201	turned off	-	IllegalArgumentException

Method scope test cases for the computeCallUnitCost:

About computeCallUnitCost method

The computeCallUnitCost() method is responsible for computing the unit cost of voice communications (cost in cents per minute). The computation of this cost depends on the client's level, the calls and SMS's made by the client, and the number of terminals the client has. This cost is determined as follows:

• Client with the Platinum level

If the number of terminals the client has is greater than or equal to 3, then the cost is 5 if the number of calls is greater than or equal to 500, and 7 otherwise. If the number of terminals is less than 3, then the cost is 8 if the number of calls is greater than or equal to 500. If the number of calls is less than 500, then the cost is 12 if the number of SMS's is less than 1000, or 10 if the number of SMS's is greater than or equal to 1000.

• Client with the Gold level

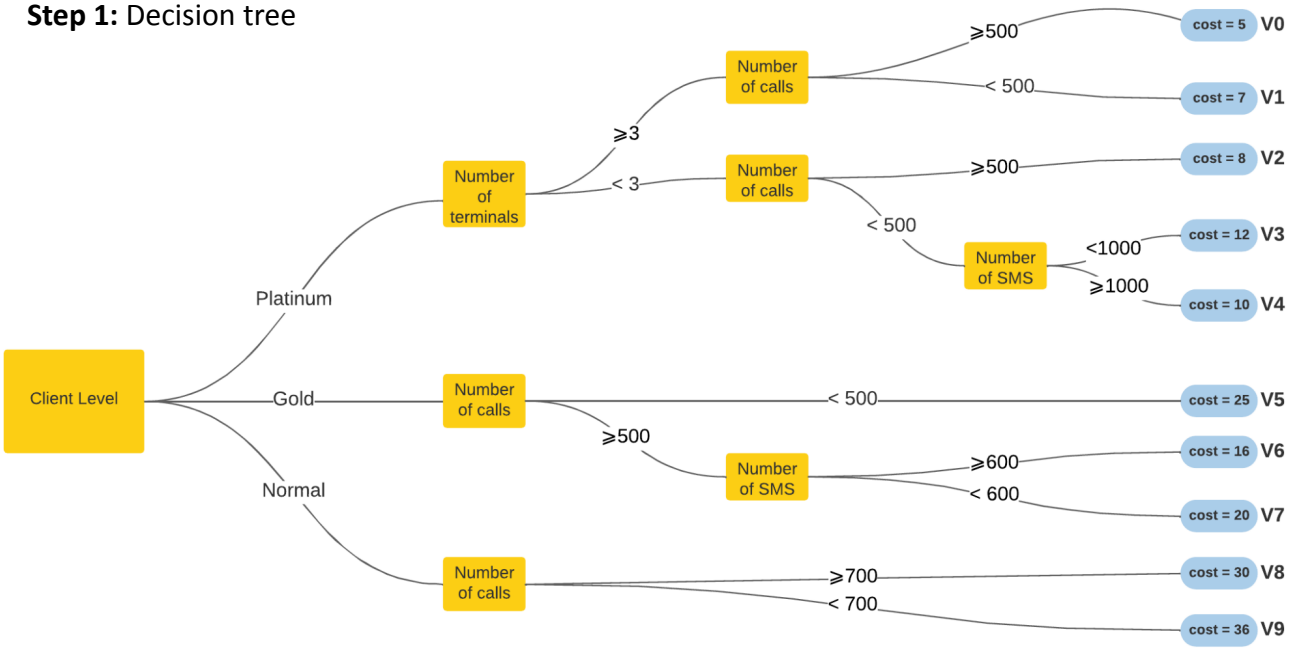
If the number of calls is less than 500, then the cost is 25. If the number of calls is greater than or equal to 500, then the cost depends also on the number of SMS's. If the number of SMS's is greater than or equal to 600, the cost is 16, otherwise, the cost is 20.

• Client with the Normal level: If the number of calls is less than 700, then the cost is 36, otherwise, the cost is 30.

Choose of a test pattern

This method implements a complex logic algorithm. In this way, we should use the **Combinational Functional Test**

Step 1: Decision tree



Step 2: Conditions

V0: clientLevel == Platinum ∧ nbOfTerminals ≥ 3 ∧ nbOfCalls ≥ 500
V1: clientLevel == Platinum ∧ nbOfTerminals ≥ 3 ∧ nbOfCalls < 500
V2: clientLevel == Platinum ∧ nbOfTerminals < 3 ∧ nbOfCalls ≥ 500
V3: clientLevel == Platinum ∧ nbOfTerminals < 3 ∧ nbOfCalls < 500 ∧ nbOfSMS < 1000
V4: clientLevel == Platinum ∧ nbOfTerminals < 3 ∧ nbOfCalls < 500 ∧ nbOfSMS ≥ 1000
V5: clientLevel == Gold ∧ nbOfCalls < 500
V6: clientLevel == Gold ∧ nbOfCalls ≥ 500 ∧ nbOfSMS ≥ 600
V7: clientLevel == Gold ∧ nbOfCalls ≥ 500 ∧ nbOfSMS < 600
V8: clientLevel == Normal ∧ nbOfCalls ≥ 700
V9: clientLevel == Normal ∧ nbOfCalls < 700

Step 3: Domain Matrix

V0: clientLevel == Platinum ∧ nbOfTerminals ≥ 3 ∧ nbOfCalls ≥ 500

V0			**1**	**-**	**2**	**-**
clientLevel	Platinum	In	Platinum	Platinum	Platinum	Platinum
nbOfTerminals	≥ 3	On	3			
		Off		2		
	Typical	In			4	5
nbOfCalls	≥ 500	On			500	
		Off				499
	Typical	In	550	700		
nbOfSMS	Typical	In	100	200	300	400
Expected result			**5**	V2	**5**	V1

V1: clientLevel == Platinum ∧ nbOfTerminals ≥ 3 ∧ nbOfCalls < 500

V1			**3**	-	-	**4**
clientLevel	Platinum	In	Platinum	Platinum	Platinum	Platinum
nbOfTerminals	≥ 3	On	3			
		Off		2		
	Typical	In			5	6
nbOfCalls	< 500	On			500	
		Off				499
	Typical	In	200	300		
nbOfSMS	Typical	In	300	500	700	1000
Expected result			**7**	V3	V0	**7**

V2: clientLevel == Platinum ∧ nbOfTerminals < 3 ∧ nbOfCalls ≥ 500

V2			-	**5**	**6**	-
clientLevel	Platinum	In	Platinum	Platinum	Platinum	Platinum
nbOfTerminals	< 3	On	3			
		Off		2		
	Typical	In			1	2
nbOfCalls	≥ 500	On			500	
		Off				499
	Typical	In	600	750		
nbOfSMS	Typical	In	100	400	550	800
Expected result			V0	**8**	**8**	V3

V3: clientLevel == Platinum ∧ nbOfTerminals < 3 ∧ nbOfCalls < 500 ∧ nbOfSMS < 1000

V3			**-**	**7**	**-**	**8**	**-**	**9**
clientLevel	Plat.	In	Platinum	Platinum	Platinum	Platinum	Platinum	Platinum
nbOfTerminals	< 3	On	3					
		Off		2				
	Typical	In			1	1	1	2
nbOfCalls	<500	On			500			
		Off				499		
	Typical	In	200	300			400	490
nbOfSMS	<1000	On					1000	
		Off						999
	Typical	In	100	200	300	500		
Expected result			V1	**12**	V2	**12**	V4	**12**

V4: clientLevel == Platinum ∧ nbOfTerminals < 3 ∧ nbOfCalls < 500 ∧ nbOfSMS ≥ 1000

V4			**-**	**10**	**-**	**11**	**12**	**-**
clientLevel	Plat.	In	Platinum	Platinum	Platinum	Platinum	Platinum	Platinum
nbOfTerminals	<3	On	3					
		Off		2				
	Typical	In			1	2	1	2
nbOfCalls	<500	On			500			
		Off				499		
	Typical	In	100	200			300	400
nbOfSMS	≥ 1000	On					1000	
		Off						999
	Typical	In	1200	1400	1500	2000		
Expected result			V1	**10**	V2	**10**	**10**	V3

V5: clientLevel == Gold ∧ nbOfCalls < 500

V5			**-**	***13***
clientLevel	Gold	In	Gold	Gold
nbOfTerminals	Typical	In	2	4
nbOfCalls	<500	On	500	
		Off		499
	Typical	In		
nbOfSMS	Typical	In	500	1500
Expected result			V7	**25**

V6: clientLevel == Gold ∧ nbOfCalls ≥ 500 ∧ nbOfSMS ≥ 600

V6			**14**	**-**	**15**	**-**
clientLevel	Gold	In	Gold	Gold	Gold	Gold
nbOfTerminals	Typical	In	1	2	3	4
nbOfCalls	≥ 500	On	500			
		Off		499		
	Typical	In			600	700
nbOfSMS	≥ 600	On			600	
		Off				599
	Typical	In	700	800		
Expected result			**16**	V5	**16**	V7

V7: clientLevel == Gold ∧ nbOfCalls ≥ 500 ∧ nbOfSMS < 600

V7			**16**	-	-	**17**
clientLevel	Gold	In	Gold	Gold	Gold	Gold
nbOfTerminals	Typical	In	1	2	3	4
nbOfCalls	≥ 500	On	500			
		Off		499		
	Typical	In			600	700
nbOfSMS	< 600	On			600	
		Off				599
	Typical	In	100	400		
Expected result			**20**	V5	V6	**20**

V8: clientLevel == Normal ∧ nbOfCalls ≥ 700

V8			**18**	-
clientLevel	Normal	In	Normal	Normal
nbOfTerminals	Typical	In	2	3
nbOfCalls	≥ 700	On	700	
		Off		699
nbOfSMS	Typical	In	1000	400
Expected result			**30**	V9

V9: clientLevel == Normal ∧ nbOfCalls < 700

V9			-	**19**
clientLevel	Normal	In	Normal	Normal
nbOfTerminals	Typical	In	4	5
nbOfCalls	< 700	On	700	
		Off		699
nbOfSMS	Typical	In	200	1400
Expected result			V8	**36**

Conclusion

We will then complete 19 test cases for this method (see the array below)

Test case	clientLevel	nbOfTerminals	nbOfCalls	nbOfSMS	Expected result
1	Platinum	3	550	100	5
2	Platinum	4	500	300	5
3	Platinum	3	200	300	7
4	Platinum	6	499	1000	7
5	Platinum	2	750	400	8
6	Platinum	1	500	550	8
7	Platinum	2	300	200	12
8	Platinum	1	499	500	12
9	Platinum	2	490	999	12
10	Platinum	2	200	1400	10
11	Platinum	2	499	2000	10
12	Platinum	1	300	1000	10
13	Gold	4	499	1500	25
14	Gold	1	500	700	16
15	Gold	3	600	600	16
16	Gold	1	500	100	20
17	Gold	4	700	599	20
18	Normal	2	700	1000	30
19	Normal	5	699	1400	36