TÉCNICO LISBOA

Software Testing and Validation

Master Degree in Computer Science and Engineering

Authors:

Allan Fernandes (97281) allancravid@tecnico.ulisboa.pt
Miguel Prazeres (95649) miguel.neves.reis.prazeres@tecnico.ulisboa.pt
João Salgueiro (100740) joao.cunha.salgueiro@tecnico.ulisboa.pt

Group 15

2022/2023 – 2nd Semester, P4

1 Introduction

In this report we will present the test cases that resulted in the application of the most appropriate test patterns. First, we will provide a brief explanation about the choice of test patterns we chose for each method/class under test. Then, the actual test cases. The battery of tests utilizing the TestNG framework is written in a separate Java file.

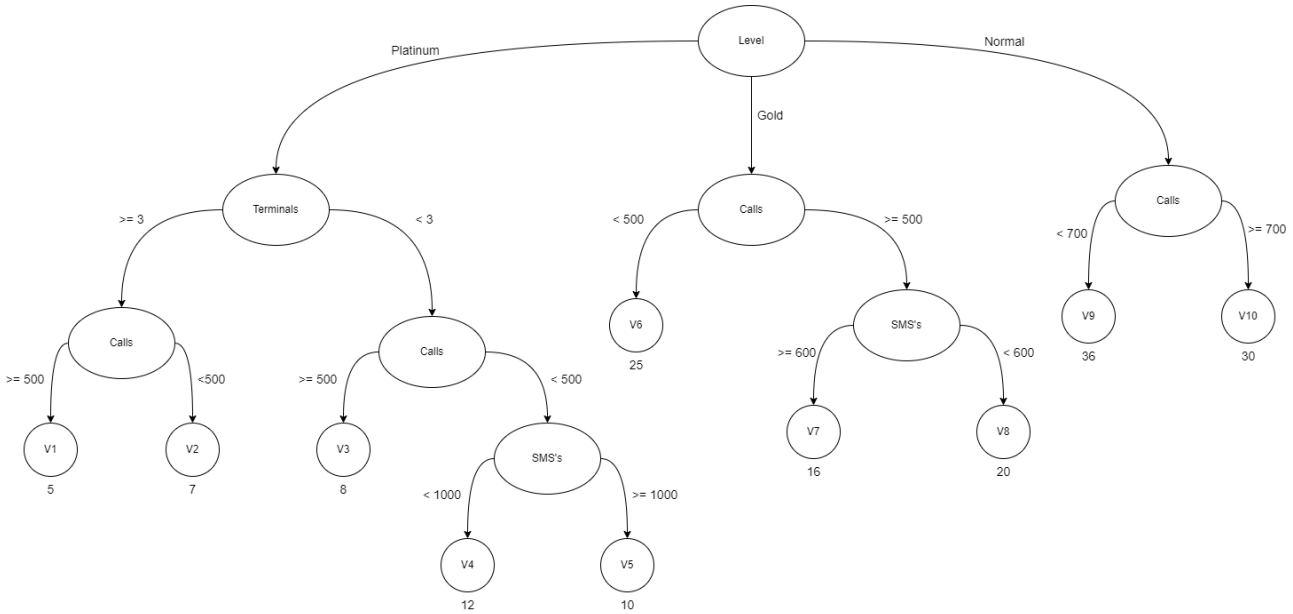
2 Method Scope test cases

2.1 computeCallUnitCost() of Class Client

Since the call cost relies on several variables and has a complex logic behind it, we found the Combinational Functional Test Pattern the most adequate.

Variables:

- **level** - level of the client
- **terminals** - number of terminals the client has
- **calls** - number of calls made by the client
- **SMS's** - number of SMS's sent by the client



To simplify the test cases we separated them into the levels of the client: Platinum, Gold and Normal.

- *Platinum:*

		V1	1	-	2	-
# Terminals	>= 3	ON	3			
		OFF		2		
	typical	IN			5	10
# Calls	>= 500	ON			500	
		OFF				499
	typical	IN	600	700		
exp. Output			5	V3	5	V2

		V2	3	-	-	4
# Terminals	>= 3	ON	3			
		OFF		2		
	typical	IN			10	20
# Calls	< 500	ON			500	
		OFF				499
	typical	IN	400	300		
exp. Output			7	V4/V5	V1	7

		V3	-	5	6	-
# Terminals	< 3	ON	3			
		OFF		2		
	typical	IN			1	0
# Calls	>= 500	ON			500	
		OFF				499
	typical	IN	550	1000		
exp. Output			V1	8	8	V4/V5

		V4	-	7	-	8	-	9
# Terminals	< 3	ON	3					
		OFF		2				
	typical	IN			0	1	0	1
# Calls	< 500	ON			500			
		OFF				499		
	typical	IN	400	300			200	100
# SMS's	< 1000	ON					1000	
		OFF						999
	typical	IN	500	400	300	200		
exp. Output			V2	12	V3	12	V5	12

		V5	-	10	-	11	12	-
# Terminals	< 3	ON	3					
		OFF		2				
	typical	IN			1	0	1	0
# Calls	< 500	ON			500			
		OFF				499		
	typical	IN	100	200			300	400
# SMS's	>= 1000	ON					1000	
		OFF						999
	typical	IN	1100	1200	1300	1400		
exp. Output			V2	10	V3	10	10	V4

- *Gold:*

		V6	-	13
# Calls	< 500	ON	500	
		OFF		499
	typical	IN		
exp. Output			V7/V8	25

		V7	14	-	15	-
# Calls	>= 500	ON	500			
		OFF		499		
	typical	IN			600	700
# SMS's	>= 600	ON			600	
		OFF				599
	typical	IN	700	1000		
exp. Output			16	V6	16	V8

		V8	16	-	-	17
# Calls	>= 500	ON	500			
		OFF		499		
	typical	IN			800	1000
# SMS's	< 600	ON			600	
		OFF				599
	typical	IN	500	400		
exp. Output			20	V6	V7	20

- *Normal:*

		V9	-	18
# Calls	< 700	ON	700	
		OFF		699
	typical	IN		
exp. Output			V10	36

		V10	19	-
# Calls	>= 700	ON	700	
		OFF		699
	typical	IN		
exp. Output			30	V9

- Description of the test cases:

  We made a domain matrix for every leaf on the tree, being careful not to repeat the same tests.

  For every condition we only have an On and Off point since there are no equalities. Finally we got 19 test cases.

2.2 sendSMS(String msg, Terminal to) of Class Terminal

We have selected the Category-Partition Test Pattern to test this method, since its logic is relatively straightforward, and it encompasses only a few functions.

- 1st step: Identify all functions of the MUT.
  - **Primary Function:**
    - send an SMS message to a Terminal, if possible.
  - **Secondary Functions:**
    - Update the sent messages, if sent.
    - If SMS invalid throw IllegalArgumentException.
    - If receiver terminal null, throw InvalidInvocationException.
    - If receiver terminal is in Off mode, throw InvalidInvocationException.
    - If sender terminal is in Busy/Off mode, throw InvalidInvocationException.

- 2nd step: Identify all input and output parameters of each function.
  - **Input:**
    - String msg
    - Terminal to
    - Terminal from
  - **Output:**
    - Boolean value - True or False
    - Exception - IllegalArgumentException, InvalidInvocationException

- 3rd step: Identify categories for each input parameter.

Parameter	Category
String msg	Valid msg Invalid msg
Terminal to	Valid (Idle/Silence/Busy) Invalid (Off)
Terminal from	Valid (Idle/Silence) Invalid (Off/Busy)

- 4^{th} step: Partition each category into choices

Parameter	Category	Choices
String msg	Valid msg	msg.length() = 10 characters msg.length() = 200 characters msg.length() > 10 characters, msg.length() < 200 characters
	Invalid msg	msg.length() = 5 characters msg.length() = 210 characters
Terminal to	On	to.getMode() == Idle to.getMode() == Silence to.getMode() == Busy
	Off	to.getMode() == Off
Terminal from	Valid	from.getMode() == Idle from.getMode() == Silence
	Invalid	from.getMode() == Busy from.getMode() == Off

- 5^{th} step: Identify constraints on choices

  Nothing to add.

- 6^{th} and 7^{th} step: Generate test cases by enumerating all choices and Develop expected values for each test case

	Input				**Expected Output**
Test Case	String msg	Terminal to	Terminal from	Returned	Exception
1	msg.length = 10	Idle	Idle	True	-
2	msg.length = 10	Silence	Idle	True	-
3	msg.length = 10	Busy	Idle	False	-
4	msg.length = 10	Off	Idle	False	-
5	msg.length = 200	Idle	Idle	True	-
6	msg.length = 200	Silence	Idle	True	-
7	msg.length = 200	Busy	Silence	False	-
8	msg.length = 200	Off	Silence	False	-
9	msg.length = rand(10,200)	Idle	Silence	True	-
10	msg.length = rand(10,200)	Silence	Silence	True	-
11	msg.length = rand(10,200)	Busy	Silence	False	-
12	msg.length = rand(10,200)	Off	Silence	False	-
13	msg.length = 5	Idle	Idle	-	IllegalArgumentException
14	msg.length = 5	Silence	Idle	-	IllegalArgumentException
15	msg.length = 5	Busy	Idle	-	IllegalArgumentException
16	msg.length = 5	Off	Idle	-	IllegalArgumentException
17	msg.length = 210	Idle	Silence	-	IllegalArgumentException
18	msg.length = 210	Silence	Silence	-	IllegalArgumentException

19	msg.length = 210	Busy	Silence	-	IllegalArgumentException
20	msg.length = 210	Off	Silence	-	IllegalArgumentException
21	msg.length = 10	Idle	Busy	-	InvalidInvocationException
22	msg.length = 10	Silence	Busy	-	InvalidInvocationException
23	msg.length = 10	Busy	Busy	-	InvalidInvocationException
24	msg.length = 10	Off	Busy	-	InvalidInvocationException
25	msg.length = 200	Idle	Busy	-	InvalidInvocationException
26	msg.length = 200	Silence	Off	-	InvalidInvocationException
27	msg.length = 200	Busy	Off	-	InvalidInvocationException
28	msg.length = 200	Off	Off	-	InvalidInvocationException
29	msg.length = rand(10,200)	Idle	Off	-	InvalidInvocationException
30	msg.length = rand(10,200)	Silence	Off	-	InvalidInvocationException
			...		

- **Description of the test cases**
  - In order to cover all possible combinations of inputs, a total of 96 test cases would be required. These test cases are obtained by multiplying the number of cases for message length (6), destination terminal states (4), and origin terminal states (4).We have only displayed the first 30 combinations above.
  - Each test case's expected output represents the corresponding output of the MUT for that specific combination.

3 Class Scope test cases

3.1 TerminalNetwork Class

Although this class has some constraints on the content of the messages (if the number of clients exceeds the maximum, that client cannot be added; among others), these constraints are not in the messages sequence or history. For those reasons, we applied the Non-Modal Test Pattern.

- **Class invariant:**

  $\forall c1, c2 \in clients : c1.name == c2.name \implies c1 == c2 \wedge \#clients >= 0 \wedge \#clients <= 50000 \wedge \forall t \in terminals : t.name.size() >= 3 \wedge t.name.size() < 10 \wedge \forall t \in terminals : t1.client \implies client \in clients$

- **Domain Matrix:**

Boundary			Input Test Values											
Variable	Condition	Type	1	2	3	4	5	6	7	8	9	10	11	12
#clients	>= 0	On	0											
		Off		-1										
	<= 50000	On			50000									
		Off				50001								
	Typical	In					1	1000	10000	2000	3000	4000	5000	6000
t.name.size	>= 3	On					3							
		Off						2						
	< 10	On							10					
		Off								9				
	Typical	In	4	5	6	7					8	4	5	6
c.name	unique	On									T			
		Off										F		
	Typical	In	T	T	T	T	T	T	T	T			T	T
t.client	valid (ϵ clients)	On											T	
		Off												F
	Typical	In	T	T	T	T	T	T	T	T	T	T		
Expected Output			P	F	P	F	P	F	F	P	P	F	P	F

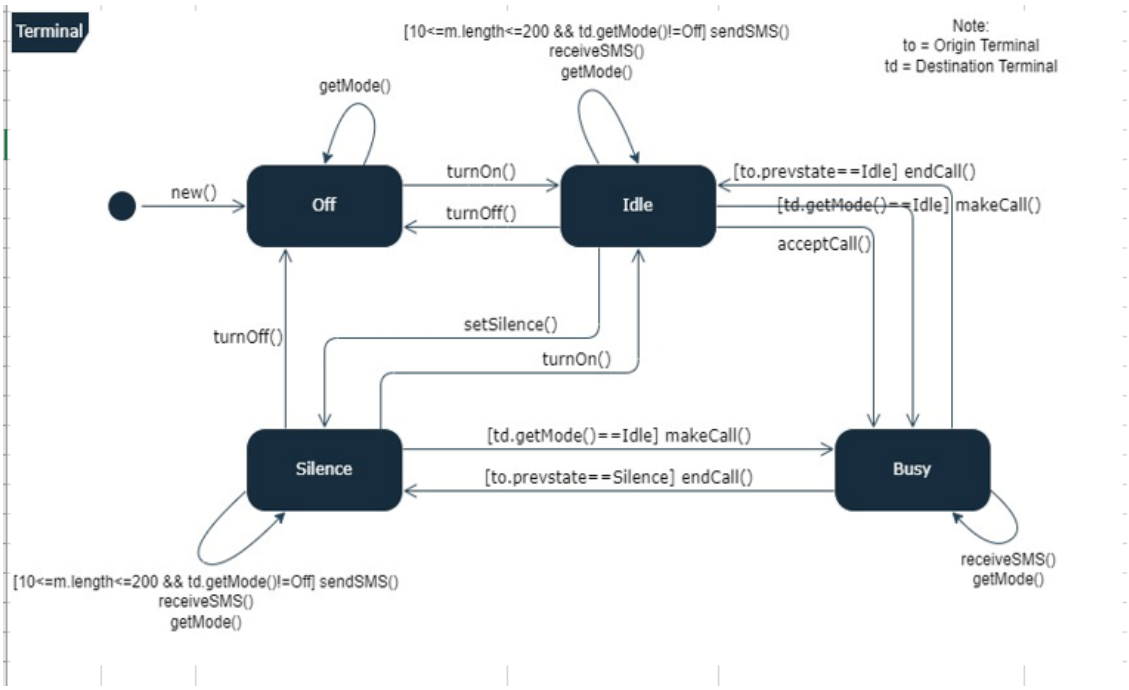
- **Description of the test cases**
  - The class invariant is build through several restrictions that resulted in 6 different conditions.
  - For every condition we have an Off and On points. The conditions that are not numerical (unique and belong) only have Boolean values.
  - Finally we have 12 tests. For the tests realized in the Java file we chose (by order) the Tests:

    3, 4, 5, 7, 9, 10, 11, 12

3.2 Terminal Class

For this class, we chose the Modal Class Test Pattern with the Finite State Machine Test Pattern, since there are different states the object can be, as well as constraints in the sequence and contents of the messages.

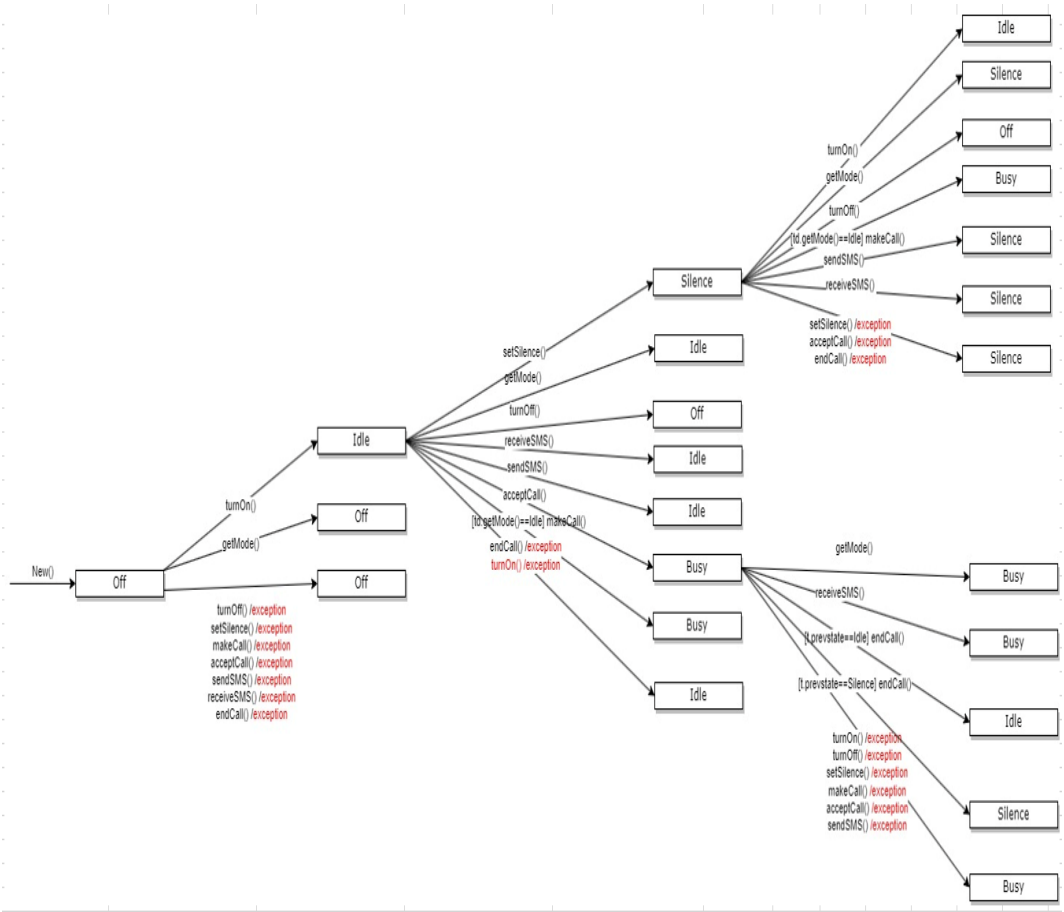
- 1st step: Generating the state model for CUT.



- 2nd step: Full expansion of conditional transition. variants.

State	Event	NextState
Off	turnOn()	Idle
Off	getMode()	Off
Off	turnOff()/exception	Off
Off	setSlience()/exception	Off
Off	makeCall()/exception	Off
Off	acceptCall()/exception	Off
Off	sendSMS()/exception	Off
Off	receiveSMS()/exception	Off
Off	endCall()/exception	Off
Idle	turnOff()	Off
Idle	setSlience()	Silence
Idle	[td.getMode()==Idle] makeCall()	Busy
Idle	acceptCall()	Busy
Idle	sendSMS()	Idle
Idle	receiveSMS()	Idle
Idle	getMode()	Idle
Idle	turnOn()/exception	Idle
Idle	endCall()/exception	Idle
Busy	[to.prevstate==Idle] endCall()	Idle
Busy	[to.prevstate==Silence] endCall()	Silence
Busy	receiveSMS()	Busy
Busy	getMode()	Busy
Busy	turnOn()/exception	Busy
Busy	turnOff()/exception	Busy
Busy	setSlience()/exception	Busy
Busy	makeCall()/exception	Busy
Busy	acceptCall()/exception	Busy
Busy	sendSMS()/exception	Busy
Silence	turnOn()	Idle
Silence	turnOff()	Off
Silence	[td.getMode()==Idle] makeCall()	Busy
Silence	sendSMS()	Silence
Silence	receiveSMS()	Silence
Silence	getMode()	Silence
Silence	setSlience()/exception	Silence
Silence	acceptCall()/exception	Silence
Silence	endCall()/exception	Silence

- 3rd step: Generate transition tree variants.



- 4th step: Generate Conformance Test Suite.

Test Case	Level 1	Level 2	Level 3	Level 4	Level 5	Expected State	Exception
1	new()					Off	-
2	new()	turnOn()				Idle	-
3	new()	getMode()				Off	-
11	new()	turnOn()	setSilence()			Silence	-
12	new()	turnOn()	getMode()			Idle	-
13	new()	turnOn()	turnOff()			Off	-
14	new()	turnOn()	[td.getMode()==Idle] makeCall()			Busy	-
16	new()	turnOn()	sendSMS()			Idle	-
17	new()	turnOn()	receiveSMS()			Idle	-
18	new()	turnOn()	acceptCall()			Busy	-
20	new()	turnOn()	setSilence()	turnOn()		Idle	-
21	new()	turnOn()	setSilence()	getMode()		Silence	-
22	new()	turnOn()	setSilence()	turnOff()		Off	-
24	new()	turnOn()	setSilence()	[td.getMode()==Idle] makeCall()		Busy	-
26	new()	turnOn()	setSilence()	sendSMS()		Silence	-
27	new()	turnOn()	setSilence()	receiveSMS()		Silence	-
30	new()	turnOn()	makeCall()	getMode()		Busy	-
36	new()	turnOn()	acceptCall()	receiveSMS()		Busy	-
37	new()	turnOn()	acceptCall()	[to.prevstate==Idle] endCall()		Idle	-
38	new()	turnOn()	setSilence()	makeCall()	[to.prevstate==Silence] endCall()	Silence	-

- 5th step: Develop test data for each path using Invariant Boundaries.

	makeCall in Idle and Silence	
	On	Off
td.getMode()==Idle	td.getMode()==Idle	td.getMode()==Busy
	sendSMS in Idle and Silence	
	On	Off
msg.length()>=10	10	9
msg.length()<=200	200	201
td.getMode() ∈ [Silence,Idle,Busy]	td.getMode() == Silence	td.getMode() == Off

- 6th step: – Develop Sneak Path Test Suite (Transition Table).

	turnOn()	getMode()	turnOff()	setSlience()	makeCall()	acceptCall()	sendSMS()	receiveSMS()	endCall()
Off	√	√	PSP	PSP	PSP	PSP	PSP	PSP	PSP
Idle	PSP	√	√	√	?	√	?	√	PSP
Silence	√	√	√	PSP	?	PSP	?	√	PSP
Busy	PSP	√	PSP	PSP	PSP	PSP	PSP	√	?
√ - Valid Transition									
PSP - Possible Sneak Path									
? - Conditional Transition									

- 7th step: – Develop Sneak Path Test Suite.

Test Case	Level 1	Level 2	Level 3	Level 4	Expected S	Exception
39	new()	turnOff()			Off	√ - InvalidInvocationException
40	new()	setSilence()			Off	√ - InvalidInvocationException
41	new()	makeCall()			Off	√ - InvalidInvocationException
42	new()	acceptCall()			Off	√ - InvalidInvocationException
43	new()	sendSMS()			Off	√ - InvalidInvocationException
44	new()	receiveSMS()			Off	√ - InvalidInvocationException
45	new()	endCall()			Off	√ - InvalidInvocationException
46	new()	turnOn()	turnOn()		Idle	√ - InvalidInvocationException
47	new()	turnOn()	endCall()		Idle	√ - InvalidInvocationException
48	new()	turnOn()	setSilence()	setSilence()	Silence	√ - InvalidInvocationException
49	new()	turnOn()	setSilence()	acceptCall()	Silence	√ - InvalidInvocationException
50	new()	turnOn()	setSilence()	endCall()	Silence	√ - InvalidInvocationException
51	new()	turnOn()	makeCall()	turnOn()	Busy	√ - InvalidInvocationException
52	new()	turnOn()	makeCall()	turnOff()	Busy	√ - InvalidInvocationException
53	new()	turnOn()	makeCall()	setSilence()	Busy	√ - InvalidInvocationException
54	new()	turnOn()	acceptCall()	makeCall()	Busy	√ - InvalidInvocationException
55	new()	turnOn()	acceptCall()	acceptCall()	Busy	√ - InvalidInvocationException
56	new()	turnOn()	acceptCall()	sendSMS()	Busy	√ - InvalidInvocationException

- **Description of the test cases**
  - The conformance test suite is composed of 38 test cases, testing all the "correct" transitions, and the expected behaviour of the class.
  - The Sneak Path Test Suite is composed of more 18 test cases, testing all paths that aren't supposed to exist.