

This document explains the approach of test design in L1-L4 with an example.

The sample application considered here is a POS application from Unicenta (<http://unicenta.com>).

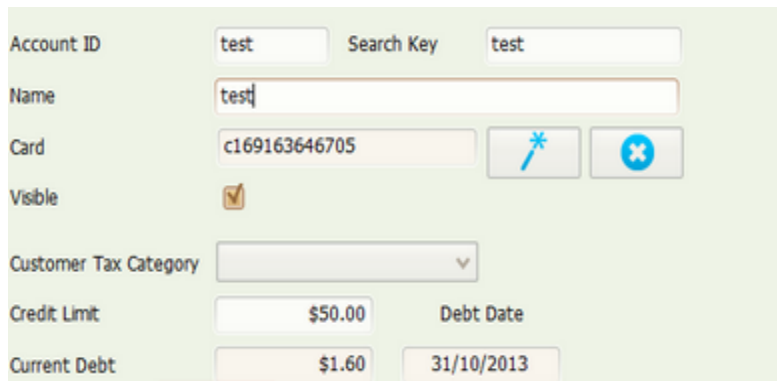
UnicentaPOS is an open source point of sale application. This application caters into Retail, Hospitality and Mixed verticals of the market. The application has lots of interesting features that support the POS.

The EUT considered here is a feature that allows adding customer accountID with a specified credit limit.

## Design at L1 - Input cleanliness

EUT - System should allow adding customer accountID with a specified credit limit (FE1).

The snapshot of the GUI screen for the feature is as shown below.



**Step 1 - Identify the inputs for the feature and understand the specification of each input.**

Input	Input specification
AccountID	Accepts string, number; Max length allowed is 8 and Min length required is 4; Special characters not allowed.
Search Key	Any string, number in the range of 4-8 character length. No space allowed.
Name	Same specification as AccountID
Credit Limit	Should be numeric values, always show in decimal with precision 2 and \$symbol
Visible	Yes/No. Implemented as check box
Other inputs Card, Current Debt, Debt date are non-editable fields and values are populated automatically. So not considered as part of input validation here.	

**Step 2 - Identify the PDTs that are applicable to each input based on the input specification.**

Note: As per HBT, at L1 we are looking for FIVE kinds of issues (PDTs) for each input. The PDTs are 1) PDT1 - Data type issue, PDT2 - Data format issue, PDT3 - Data boundary issue, PDT4 - Data dependency issue and PDT5 - Data value set issue. It is important to identify what these issues meant in your context of EUT and then appropriately customize it.

Input	Input specification	PDT1	PDT2	PDT3	PDT4	PDT5
AccountID	Accepts string, number; Max length allowed is 8 and Min length required is 4; Special characters not allowed.	x		x		
Search Key	Any string, number in the range of 4-8 character length. No space allowed.	x		x		
Name	Same specification as AccountID	x		x		
Credit Limit	Should be numeric values, always show in decimal with precision 2 and \$symbol	x				
Visible	Yes/No. Implemented as check box					x

Now we are clear about what type of issues need to be look for each input.

**Step 3 - Design the scenarios based on the identification of applicable PDTs for each input.**

TSID	Scenario description	Type	PDT Map
FE1.TS1	Ensure that the accountID does not accept invalid values	-ve	PDT1,PDT3
FE1.TS2	Ensure that searchKey does not accept invalid values	-ve	PDT1,PDT3
FE1.TS3	Ensure that Name field does not accept invalid values	-ve	PDT1,PDT3
---	-----	---	---
---	-----	--	---

**Step 4 - Generate the test cases for each scenario.**

To generate test cases, we need to identify the test data values for each input based on the specification.

Input	Input specification	Test data value set (-ve value set)
AccountID	Accepts string, number; Max length allowed is 8 and Min length required is 4; Special characters not allowed.	String/Number with length 9, with length 3 Value with length 4 that has a special character
Search Key	Any string, number in the range of 4-8 character length. No space allowed.	Same as above, here special character is space
Name	Same specification as AccountID	Same as accountID

Credit Limit	Should be positive numeric values, always show in decimal with precision 2 and \$symbol	Credit value less than ZERO, without decimal point, without the \$ symbol, Non-numeric value
Visible	Yes/No. Implemented as check box	No specific value can given as -ve here as it is implemented as a check box.

Here apply techniques like BVA, Equivalence partitioning, Error guessing, special value on each input and identify the negative set of values based on the specification of each input. Considering only negative values as in L1 the objective is to ensure that the invalid values are rejected gracefully by each input.

TCID	Test case description	Test data	Expected Result
FE1.TS1.T C1	Verify that the AccountID does not accept string with length 3	Acc	Display message that AccountID cannot be less than 4 character length
FE1.TS1.T C2	Verify that the AccountID does not accept string with special characters	Account !	Display message that AccountID cannot include special character
---	-----	---	---
---	-----	--	---

## Design at L2 - Input interface cleanliness

Here the feature (FE1) is used by the users by a GUI.

**Step 1 - Identify the elements of the FE1 interface and the specification of each element in the interface (GUI).**

Here the elements are AccountID, search key, Name, Card, Visible, Credit limit etc.

Interface specification are like Card, CurrentDate, Deb date should be non-editable, Visible should be a check box, there should be two icons next to card, one for searching the card and second for removing the selected card details etc.

**Step 2 - Identify the applicable PDTs with respect to the FE1 and understand what it means in this context**

There are some common PDTs that needs to be validated at Level 2 is given already. We need to select the applicable PDTs in the current context (GUI of FE1) and understand what it means here. Please add more PDTs specific to the context if necessary. This is a base list of common PDTs at level L2.

PDT	Description	Applicable?	Meaning in the context
1	Wrong/ambiguous error message	Yes	Input validation messages, responses after actions may show wrong messages

2	Wrong/ambiguous help information	No	
3	Lack of warnings/error messages	No	
4	Improper scope of interface access	No	
5	Misleading API name	No	
6	Insufficient information to use the API	No	
7	Missing/extra components	Yes	The screen should contain all the fields mentioned in the screen shot (ideally we need to list down the fields as per specification and check for the existence of those in the GUI).
8	Component placement issue	Yes	The order of fields placement need to be checked as per the specification. Icons for edit and remove should display after card.
9	Color/font/alignments issue	Yes	Font style and alignment of fields in this feature to be checked
10	Spelling/grammar mistakes	Yes	All field labels and messages to be checked for spelling mistakes
11	Lack of display clarity	No	
12	Incorrect no of arguments	No	
13	Wrong order of arguments	No	
14	Incorrect data type/size of arguments	No	
15	Invalid return type	No	
16	Incorrect UI element response	Yes	Card, Customer Tax, Current Debt should be disabled from editing, Current Debt should calculate once we enter Credit Limit, Date format should display in dd/mm/yyyy format in Debt Date.
17	Wrong/No progress message	No	
18	Incorrect tab sequence	No	
19	Cursor displayed in the wrong place	Yes	Cursor has to be displayed in the component where the focus is present
20	Lack of real-time inline validation	Yes	Credit limit should be validated inline.

Once we identify the PDTs applicable and what it means to the context, and then check for all these in the given GUI of the feature. This can be treated as a check list and check for these issues. No need to write separate test cases at this level.

In addition to the above mentioned PDTs, please check if any other interface behaviors need to be checked for the given Feature and cover that also. (E.g. like, the company icon should be in the left corner, mandatory fields should show with red asterisk, some buttons should be disabled based on some other field's value etc)

## Design at L3 - Structural integrity

Validation at this level is more of the internal design and code level issues. We need some code level information like internal design, coding standards to be followed, the implementation logic etc.

The common issue looking at this level is whether the expected exceptions are handles well, is the resources used are released in timely manner, is the internal implementation follows the coding and guidelines it is supposed to follow etc.

This is more like preventing defects in the implementation stage rather than detecting these issues at the later stage.

## Design at L4 - Behaviour correctness

At this level the entity (FE1) will be evaluated for the completeness and correctness of its intended functional behavior.

### Step 1 - Understand the functional behavior of the feature FE1.

The feature will allow adding customer account with credit limit. Duplicate customers are not allowed. Customers will be created only when all the mandatory details are provided with valid set of values. The account can be created with TWO types of pre-defined tax category. The created account will be visible to the public or hide from the public based on the selection of visible tag during account creation.

### Step 2 - Identify the conditions that govern the behavior of the feature FE1.

1. Mandatory detail completed or partial
2. Account created with hidden/visible mode
3. Account already exists will be rejected
4. Account with 2 tax category can be created

### Step 3 - Identify the possible values of each of these conditions identified.

Each conditions value will be controlled by the value of some data input. So we need to identify the data elements that control the value of each of these conditions to understand the possible values of each of these conditions.

Input conditions	Positive value set	Negative value set
Is the account Id valid?	New account id	already existing
Mandatory fields complete and valid?	Complete and valid, Partial information	Invalid information
Tax category	Category1, Category2 (Assumed TWO tax category)	
Visibility flag	ON/OFF	

**Step 4 - Logically combine the conditions based on the possible values identified for each condition.**

Here we are using Decision Table technique to come up with the logical combination of the conditions.

Input conditions	Positive value set	Negative value set	Combination of conditions (Rules in Decision Table)			
			R1	R2	R3	R4
Is the account Id valid?	New account id, already existing	Invalid account ID	New account	New account	Existing account	New account
Mandatory fields complete and valid?	Complete and valid, Partial information	Invalid information	Complete info	Complete info	Complete info	Partial info
Tax category	Category1, Category2 (Assumed TWO tax category)		Category 1	Category 2	Category1	Category2
Visibility flag	ON/OFF		ON	OFF	ON	OFF
<b>Expected Output (result)</b>						
O1 - Account Created with visible flag ON for the selected tax category			x			
O2 - Account created with visible flag OFF for the selected tax category				x		
O3 - Display message - AccountID already exists					x	
O4 - Display message account cannot be created						x

Each rule in the decision table is a scenario with a one-one mapping to the expected result. (O1, O2, O3, O4 are the outputs)

Like this we need to combine all possible combination of conditions and derive the scenarios. Some combinations may not make sense in the given context. Neglect such combinations.

**Step 5 - Convert each rule into the test scenario with a description of each scenario**

TSID	Scenario description	Type	Expected Result
FE1.L4.TS 1	Ensure that the account can be created by providing valid mandatory information that can be visible to everyone	+ve	O1
FE1.L4.TS 2	Ensure that the account can be created by providing valid mandatory information that can be visible to selected members and hidden from public	+ve	O2
FE1.L4.TS 3	Ensure that duplicate accounts cannot be created	-ve	O3
FE1.L4.TS 4	Ensure that account cannot be created without providing the mandatory information	-ve	O3
---	-----	--	---

**Step 6 - Generate the test cases for each scenario**

TCID	Test case description	Test data	Expected Result
FE1.L4.TS1.T C1	Verify that the account can be created with visibility flag ON provided all the mandatory fields are valid.	Provide non-existing accountID with length 4	O1
FE1.L4.TS1.T C2	Verify that the account can be created with visibility flag ON provided all the mandatory fields are valid.	Provide non-existing accountID with length 9	O1
---	-----	---	---
---	-----	--	---

Note: This document is intended to describe the approach of test design. It is not required to create and collect the details in the format specified always. The information can be collected in any format. What is important is the approach to test design at each level.