Test designer QL1-QL4-user guide

HBT3.0 application suite

Version 1.1

STAG Software Private Limited

6/2/2014

Test designer is one of the applications from HBT3.0 application suite that helps to design test scenarios/cases for each quality level. This document details the step by step procedure for the test design at quality level QL1-QL4.

Contents

OVERVIEW

Hypothesis Based Testing (HBT) is a personal test methodology that is unique in its approach to ensuring cleanliness of software. It is not a process system; rather it is a methodology for an individual to deliver clean software. It is based on sound engineering principles geared to deliver the promise of clean software.
The central theme is of constructing a hypothesis of potential defects that may be probable, and then scientifically proving that they do not indeed exist.

NINE quality levels in HBT is looking for 29 standard cleanliness criteria, 155 potential defect type by conducting 17 types of test. Cleanliness criteria are based on the product quality metrics defined by ISO 25010.

This document outlines the test design approach for each Quality Levels defined by HBT3.0.
- For each Quality Level – What are the cleanliness criteria to be satisfied?
- To satisfy each cleanliness criteria, what type of potential defect types to be looking for?
- What type of test to be run to find out those potential defect types?

The Potential Defect Types listed here is common potential defects and not any application specific. We may need to select the PDTs that are applicable for the Context of the application under test, add new PDTs based on the application context, customize existing PDTs based on the context of the application. Generally PDTs mentioned in the early stages (QL1-QL3) can be used directly and check the existence of applicable PDTs in the application. PDTs mentioned in QL4-QL5 (those marked as Generic PDT), may not able to use directly. Functionality and interaction behavior of applications is very specific to each application and cannot be generalized. We have to customize those PDTs based on the context of the application to adapt accordingly.

QL1: INPUT CLEANLINESS - DETAILS.

QL1 – Input cleanliness: CC-PDT-TT Mapping			
Cleanliness criteria	**Potential defect type**	**Test Type**	**Conditions to be considered**
CC1: Input data integrity	PDT1 - Data type issue	TT1 Input validation test	Data Type Data Format/Syntax Data Boundaries/Limits Data value conditions
	PDT2 - Data format issue		
	PDT3 - Data boundary issue		
	PDT4 - Data dependency issue		
	PDT5 - Data value set issue		

QL1 – Input cleanliness: QL, TT, CC, PDT definitions	
QL1: Input cleanliness	To validate whether the validation of inputs are handled well. That accepts the valid values and rejects the invalid input values gracefully.
TT1:Input validation test	This test is conducting to ensure that the "input cleanliness criteria" met. We should have test scenarios to validate the presence of each applicable PDT (PDT that affects the "input cleanliness criteria") as part of this test.
CC1: Input data integrity	Degree to which a system, product or component prevents unauthorized access to, or modification of, computer programs or data. That is, in this context, are we validating the input data before processing? Accept only valid values and reject the invalid values gracefully.
PDT1: Data type issue.	That the type of input data may not be validated properly.
PDT2: Data format issue.	That the input data format may not be validated properly.
PDT3: Data boundary issue.	That the boundary (upper and lower limits) of input data may not be validated properly.
PDT4: Data dependency issue.	That the dependency among input fields may not be validated properly.
PDT5: Data value set issue.	That the possible set of values for a specific input field may not be correct. White list values many not be clear for each field where ever it is applicable.

QL1 – Test design.

Let us take an example of a function which creates an employee with certain details. The employee creation API looks as follows.

CreatEmployee (String empname, String gender, Date dateofbirth). This API can be represented as a GUI also.

To design QL1 test cases for this API, we need to follows the steps mentioned above. Let us represent it in a table for better understanding.

1. Identify the input fields of the feature
2. Understand the input specifications for each input fields.
3. Identify/select the PDTs applicable for each input field.
4. Design scenarios for each applicable PDTs for each input field.
5. Design test cases for each test scenario.

Input name	**Input specification**	**PDT1** **Data type issue**	**PDT2** **Data format issue**	**PDT3** **Data boundary issue**	**PDT4** **Data dependency**	**PDT5** **Data value set issue**
empname	Length 3-10, only alphabets accepts, space is allowed as a special character.	X		X		
gender	Fixed set of values(Male/Female)					X
dateofbirth	Date format(dd-mm-yyyy), for males age range is(21-50), for females age range is (18-40)	X	X	X	X	

Now we are clear about what kind of issues need to be validated for each input field. Let us design scenarios for each input based on the above understanding. For each identified PDT for each field we need to have a scenario to ensure the acceptance of valid input values and to ensure the graceful rejection of invalid values. If there are many inputs having similar specification, then we can have a common scenario and need to execute that for all the applicable inputs.

TS ID	Scenario Description	Type	PDT
TS1	Ensure that employee name accepts any alphabets within 3-10 range.	+ve	PDT1, PDT3
TS2	Ensure that employee name rejects any alphabets outside 3-10 range and all numerics and special characters except space.	-ve	PDT1,PDT3
TS3	Ensure that gender accepts the values male/female	+ve	PDT5
TS4	Ensure that the gender rejects any value other than Male/Female	-ve	PDT5
TS5	Ensure that the dateofbirth accepts dates with the format dd-mm-yyyy	+ve	PDT1,PDT2
TS6	Ensure that the dateofbirth will not accept any value other the dates and in the format dd-mm-yyyy	-ve	PDT1,PDT2
TS7	Ensure that the dateofbirth value should be within the age 21-50 if the gender is Male	+ve	PDT3,PDT4
---	----------		

As above, we need to design the test scenarios based on the above understanding. Some scenarios can be mapped to more than one PDT at this level, if we combine the logic to be validated properly. Eg:: TS1, if we use alphabet value within the specified boundary, both the data type and boundary issues can be covered in one scenario.

Let us design test cases for each test scenario.

Note: Only basic attributes are mentioned here for test case template. There are other attributes like priority, execution stage, execution model etc which can be included as per the requirement/organizational teat case template.

There should be at least one test case for each test scenario.

TCID	Test case objective	Test data	Expected result
TS1.TC1	Verify that the employee name accepts alphabets with length 3	Employeename – Ram, (valid values for other fields, if all fields are required to execute)	Employename with length 3 alphabet accepts
TS1.TC2	Verify that the employee name accepts alphabets with length 10	Employeename – Ramanujanm, (valid values for other fields, if all fields are required to execute)	Employename with length 10 alphabet accepts
--	---		
TS2.TC1			
TS2.TC2			
--			

QL2: INPUT INTERFACE CLEANLINESS – DETAILS

QL2 – Input interface cleanliness: CC-PDT-TT Mapping			
Cleanliness criteria	**Potential defect type**	**Test Type**	**Conditions to be considered**
CC2: User error protection	PDT6: Wrong/ambiguous error message	TT2: GUI validation test	API- API name Accessibility scope Order of inputs. Mandatory inputs. Dependency of inputs. Type of inputs. GUI- No of GUI elements Grouping of elements. Presentation of elements. Mandatory elements. Color/Font of elements.
	PDT7: Wrong/ambiguous help information.		
	PDT8: Lack of warnings/error messages.		
	PDT9: Improper scope of interface access.	TT3: API validation test	
	PDT10: Misleading API name.		
	PDT11: Insufficient information to use the API.		
CC3: User interface aesthetics	PDT12: Missing/extra components.	TT2: GUI validation test	
	PDT13: Component placement issue.		
	PDT14: Color/font/alignments issue.		
	PDT15: Spelling/grammar mistakes.		
	PDT16: Lack of display clarity.		
	PDT17: Incorrect no of arguments.	TT3: API validation test	
	PDT18: Wrong order of arguments.		
	PDT19: Incorrect data type/size of arguments.		
	PDT20: Invalid return type.		
CC4: Operability	PDT21: Incorrect UI element response.	TT2: GUI validation test	
	PDT22: Wrong/No progress message.		
	PDT23: Incorrect tab sequence.		
	PDT24: Cursor displayed in the wrong place.		
	PDT25: Lack of real-time inline validation.		

QL2 – Input interface cleanliness: QL, TT, CC, PDT definitions	
QL2:Input interface cleanliness	To ensure that interface through which the inputs are accepted is clean. Here the interface can be a GUI or an API invoked from a command line or from any other application/system.
TT2: GUI validation test	This test is conducting to ensure that the interface is clean if the interface is a GUI. Application will be interacting/accepting necessary inputs from the user via this interface.
CC2: User error protection	Degree to which a system protects users against making errors. That is, how the interface helps the user to avoid making mistakes. At QL2, the focus is on preventing mistakes in individual element/action for an entity under test, If the interface is a GUI.
PDT6: Wrong/ambiguous error message	That the application interface may not give proper error response and that may lead to processing the wrong correction mechanism.
PDT7: Wrong/ambiguous help information.	That the application interface may not give correct help message and that may lead to use the application in a wrong way.
PDT8: Lack of warnings/error messages.	That the application interface may not give warning messages for the destructive operations performed by the user.

CC3: User interface aesthetics	Degree to which a user interface enables pleasing and satisfying interaction for the user. That is, how is the look and feel of the interface? Is the combination of colors used is appealing to the user? Are the components are grouped logically so that it looks compacted and easy for the user to use? At QL2, the focus is on the look and feel for the entity under test, If the interface is a GUI.
PDT12: Missing/extra components.	That the GUI may contains extra components (elements) than the required/specified.
PDT13: Component placement issue.	That the GUI elements may not be placed as per the specified layout.
PDT14: Color/font/alignments issue.	That the usage of font/color of various GUI elements may not consistent and not as per the specification.
PDT15: Spelling/grammar mistakes.	That the messages/help information/other details on the GUI may not be free from spelling/error mistakes.
PDT16: Lack of display clarity.	That the information displayed on the GUI may not be clear/completed. Like truncated/blurred information.
CC4: Operability	Degree to which a product or system has attributes that make it easy to operate and control. That is, how easy the user can use the system to achieve their specific tasks? At QL2, the focus is on ease of performing a task for the entity under test, If the interface is a GUI.
PDT21: Incorrect UI element response.	That the expected response from GUI elements may not correct - Selection may not be highlighted, selected value may not be populated in a list/combo box, selected buttons may not be enabled etc.
PDT22: Wrong/No progress message.	That the progress of time consuming operations may not be displayed to the user.
PDT23: Incorrect tab sequence.	That the order of tab sequence may not be as per the usage sequence.
PDT24: Cursor displayed in the wrong place.	That the cursor placement/focus may not be in the active GUI element.
PDT25: Lack of real-time inline validation.	That the inline validation of certain important elements may not be present - instead of validating all the elements at the end, inline validation of each field can improve the usability.
TT3: API validation test	This test is conducting to ensure that the interface is clean if the interface is a non-GUI (Command line interface). If the interface invoked from command line, or directly from some other applications.
CC2: User error protection	Degree to which a system protects users against making errors. That is, how the interface helps the user to avoid making mistakes. At QL2, the focus is on preventing mistakes in individual element/action for an entity under test., If the interface is a Non-GUI.
PDT9: Improper scope of interface access.	That the scope interface may not be define correctly - APIs with wrong accessibility scope - public, private.
PDT10: Misleading API name.	That the name of the API may not be indicating the functionality - operation- of what it does. This may lead to wrong usage of the API.
PDT11: Insufficient information to use the API.	That the information of how to use the API may not be available.
CC3: User interface aesthetics	Degree to which a user interface enables pleasing and satisfying interaction for the user. That is, how is the look and feel of the interface? Is the combination of colors used is appealing to the user? Are the components are grouped logically so that it looks compacted and easy for the user to use? At QL2, the focus is on the look and feel for the entity under test, If the interface is a Non-GUI.
PDT17: Incorrect no of arguments.	That the number of arguments specified for an API may not be as per the specification.
PDT18: Wrong order of arguments.	That the order of arguments used in the API may not be as per the specification.
PDT19: Incorrect data type/size of arguments.	That the type/size of each argument in the API may not as per the specification.
PDT20: Invalid return type.	That the return type of the API may not be as per the specification.

QL2 – Test design.

As mentioned above, scenarios at QL2 will be a kind of check lists which can be verified for the interface under test.

For GUI interface,

- Are all the required elements are present in the GUI?
- Is the alignment of each elements are correct?
- Are the messages/element name / information provided in the interface are free from spelling/grammar mistakes?
- Is the elements are placed in the interface as per the required layout etc…

To come up with these check lists, we need to understand the details of the interface like,

- What are the required elements in the GUI
- What are the interface specifications
  - For each element in the GUI is there anything to be specified.
  - What are the dependencies among the elements in the GUI
    Etc.

For API interface,

- Understand about the expected signature (no of arguments, type of each argument, order of arguments, scope of the interface, name of the interface etc) of the API.
  Ensure that the API defined as per the specification. This equivalent of testing the GUI structure.
- Check for the correctness of the error responses. These are not the response from the logic errors. Basically at this level we need to validate whether the argument validation responses are correct. This is like, if we provide wrong type of argument, what will be the response?
  In QL1, we were validating the rejection of the invalid inputs, here in QL2, we will ensure that it rejected correctly. Some cases these kind of validations can be combined with QL1 and covered as per of QL1 test cases. The intention here is to ensure that we have not missed these kind of validations also. Since QL1 and QL2 are part of Unit test, does not matter whether we executed and validated these issues part of QL1/QL2. But it has to be considered.

QL3: STRUCTURAL INTEGRITY – DETAILS

QL3 – Structural integrity: CC-PDT-TT Mapping			
Cleanliness criteria	**Potential defect type**	**Test Type**	**Conditions to be considered**
CC8: Fault tolerance	PDT26: Improper initialization.	TT4 Structural test	Resource use policy. Concurrency related. Timing related – timeout. Composition of internal elements. Linkage of components. Non terminating loop conditions. Data usage conditions. Error handling conditions.
	PDT27: Cached/stale data usage.		
	PDT28: Improper loop condition use.		
	PDT29: Improper exception handling.		
	PDT30: Incorrect data conversion.		
CC12: Resource usage	PDT31: Timely non-release of used resources.		
	PDT32: Unsynchronized concurrent resource usage.		
	PDT33: Wrong resource usage configuration.		
	PDT34: Not checking availability of resource before use.		
	PDT35: Improper resource/task allocation in a distributed environment.		
CC29: Modularity	PDT36: Wrong usage of building blocks.		
	PDT37: Improper runtime task allocation.		
CC27: Confidentiality	PDT38: No/incorrect permission check.		

QL3 – Structural integrity: QL, TT, CC, PDT definitions	
QL3:Structural integrity	To ensure that the internal structure of the code is designed in such a way that it has taken care of common flaws that generally happens with the design and implementation. This is a preventive measure to reduce the risk of introducing some code level defects.
TT4: Structural test	This test is conducting to ensure that the internal design and implementation has taken care the common errors/exceptions at design level. Usually this is done at the code level not by the typical design and execution of the test cases. These are certain points that are to be taken care during the design and implementation of the application. These points can be checked as part of code review also. This is possible only when we have access to the code and so is done by developers at the implementation time (recommended)/later.
CC8: Fault tolerance	Degree to which a system, product or component operates as intended despite the presence of hardware or software faults. At QL3, the focus is on ensuring that the possible hardware/software faults are handled gracefully in design/code.
PDT26: Improper initialization.	That the variables may not be initialized to the correct defaults/values before usage.
PDT27: Cached/stale data usage.	That the variable may be using some staled/cached irrelevant value.
PDT28: Improper loop condition use.	That the loop conditions may not be closed/exited with proper conditions checks.
PDT29: Improper exception handling.	That the possible error/exceptions may not handle properly in the code.
PDT30: Incorrect data conversion.	That the data loss due to conversion from one type to another may not be considered properly.
CC12: Resource usage	Degree to which the amounts and types of resources used by a product or system when performing its functions meet requirements. At QL3, the focus is on ensuring that the various available resources are consumed / released appropriately in design/code.

PDT31: Timely non-release of used resources.	That the resources used by the application may not be released on time after the use.
PDT32: Unsynchronized concurrent resource usage.	That the concurrent usage of shared data may not be synchronized properly.
PDT33: Wrong resource usage configuration.	That the configuration of resource usage information may not be correct. Mis configuration may lead to misuse of resources.
PDT34: Not checking availability of resource before use.	That the availability of resources may not be checking before the actual usage.
PDT35: Improper resource/task allocation in a distributed environment.	That the task/resources allocated in a distributed environment may not correct.
CC29: Modularity	Degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components.
PDT36: Wrong usage of building blocks.	That the components/building blocks used may not be in the proper way - wrong usage of building blocks within the application may result in to poor maintainability - change impact will be more, if not designed/used the blocks properly.
PDT37: Improper runtime task allocation.	That the dynamic task/resource allocation may not proper in a multi-threaded environment.
CC27: Confidentiality	Degree to which a product or system ensures that data are accessible only to those authorized to have access. At QL3, the focus is on ensuring that proper authorization is implemented in design/code.
PDT38: No/incorrect permission check.	That the authorization to use a specific resource/task may not be validated properly. Access of some sensitive data by some code.

QL3 – Test design.

As mentioned above, QL3 PDTs are typically validating against the design/code. From developer point of view, the possibility of occurrence of these types of issues can be prevented in the coding stage itself. These are kind of Preventive measures than to be detected in later testing. From the test design point of view at QL3, these are kind of check lists that can be verified in the code once before releasing to QA. We need to go through each PDT mentioned and ensure that possibility of the occurrence of applicable PDT is handled at the code.

QL4: BEHAVIOR CORRECTNESS – DETAILS.

QL4 – Behavior correctness: CC-PDT-TT Mapping			
Cleanliness criteria	Potential defect type	Test Type	Conditions to be considered
CC15: Functional completeness	PDT39: Missing conditions.	TT5 Functionality test	Those conditions that govern the functional behavior of the EUT. Identification of expected behaviors from the EUT.
	PDT40: Unnecessary/redundant condition.		
	PDT41: Wrong condition identification.		
CC16: Functional correctness	PDT42: Wrong values for condition.		
	PDT43: Wrong output identification.		
	PDT44: Wrong sequence of conditions.		
	PDT45: Wrong/ambiguous output.		
CC12: Resource usage	PDT46: Unusual resource consumption by the entity.		
CC13: Time behavior	PDT47: High response time.		
CC27: Confidentiality	PDT48: Missing/Incorrect authorization.	TT6: Authorization test.	
CC26: Accountability	PDT49: No/insufficient logging.		
	PDT50: Wrong transaction logging.		

QL4 – Behavior correctness: QL, TT, CC, PDT definitions	
QL4: Behavior correctness	To validate whether the functionality of the EUT has implemented completely and correctly as per the specification.
TT5: Functionality test	This test is conducting to ensure that the Functional behavior of the entity has implemented completely and correctly.
CC15: Functional completeness	Degree to which the set of functions covers all the specified tasks and user objectives. At QL4, the focus is on validating all the expected functionality is met by the entity under test.
PDT39: Missing conditions	That the identified conditions which govern the functional behavior of the entity may not be complete.
PDT40: Unnecessary / redundant condition	That the conditions identified may be extra.
PDT41: Wrong condition identification	That the conditions identified may be wrong.
Above mentioned PDTs are generic in nature. As per HBT, functional behavior of any entity is controlled by a set of conditions. If all the conditions met, then the functional behavior of the entity will be as expected and violation of any condition will result into an unexpected behavior of the entity. Any of the above situations may lead to the presence of some of the potential defects in the EUT.	

This needs to be identified with respect the context of each EUT. Refer the test design example given below for more details.	
CC16: Functional correctness	Degree to which a product or system provides the correct results with the needed degree of precision. At QL4, the focus is on validating the expected functional behavior of the feature under test.
PDT42: Wrong values for condition.	That the possible values identified for each condition may be wrong.
PDT43: Wrong output identification.	That the expected behavior identified from the entity may be wrong.
PDT44: Wrong sequence of conditions.	That the sequence of conditions combined may be wrong.
The above mentioned PDTs also generic in nature. Please refer the test design example of QL4 mentioned below for clarification.	
PDT45: Wrong/ambiguous output.	That the expected output identified from an entity may not be correct.
CC12: Resource usage	Degree to which the amounts and types of resources used by a product or system when performing its functions meet requirements. At QL4, the focus is on validating that the resources are consumed optimally by the entity under test.
PDT46: Unusual resource consumption by the entity	That the resource usage by the entity many not as expected/specified. To know this we need to understand the resource usage expected by the entity under test.
CC13: Time behavior	Degree to which the response and processing times and throughput rates of a product or system, when performing its functions, meet requirements. At QL4, the focus is on validating the response time of the entity under test to perform the intended action.
PDT47: High response time	That the repose time expected from the entity may not in the accepted limit.
TT6: Authorization test	This test is conducting to ensure that the functionality is accessed by the authorized person only. We should have test scenarios to validate the specified authorization rules to access the functionality of the EUT.
CC27: Confidentiality	Degree to which a product or system ensures that data are accessible only to those authorized to have access. At QL4, the focus is on validating that the functionality/content applicable in the entity under test is accessed by the authorized users only.
PDT48: Missing/Incorrect authorization.	That the entity may be used by an un-authorized person/component due to lack of authorization.
CC26: Accountability	Degree to which the actions of an entity can be traced uniquely to the entity. At QL4, the focus is on validating the logging/tracking transactions involved by the entity under test.
PDT49: No/insufficient logging.	That the required information may not be logged by the application/entity.
PDT50: Wrong transaction logging.	That the logged information may not be same as the actual transaction.

QL4 – Test design.

Test scenario design approach at QL4 is show below.

Understand the functionality of the entity under test

Read the specification, discuss with the concerned people, explore the application etc.

Identify the conditions that govern the functionality

Understand the Business Logic implemented to achieve the functionality and then identify the conditions that control the business logic.

Identify the possible value set of each condition

Conditions	Possible value set	
	(+ve)	(-ve)
Condition1	v11,v12,v13	v11',v12'
Condition2	v21,v22	v21'

Check with the concerned team/people and ensure that the conditions and possible values of each conditions identified are correct and complete. This helps to avoid any unnecessary conditions (if considered) and design effective test scenarios. Here we are eliminating the presence of PDT39-45 which can affect the cleanliness criteria CC15&16(Functional completeness and Functional correctness)

Combine the conditions by combining the possible values of each condition and identify the expected output from each combination. Consider only that combination that is applicable in the application context.

Conditions	Possible value set		Combinations				
	+ve	-ve	R1	R2	R3	R4	--
Condition1	v11,v12,v13	v11',v12'	V11	V12	V13	V11'	--
Condition2	v21,v22	v21'	V21	V22	V21	V22	--

Expected output							
Output1			x				
Output2				x	x		
Output3						x	

The intention here is to ensure that all the possible combinations (functional behavior flows) and the corresponding outputs have identified. If we know the expected outputs already, then identify the corresponding flow and then the conditions. This is a forward-backward approach that ensures the completeness of flow. We may need to check with concerned people here to ensure the correctness in terms of the order of conditions, mapping to the expected output, validity of combinations etc.

Convert each combination (rules) in to test scenario with an expected output.

TSID	Scenario description	Type	Exp Result
TS1(R1)	Ensure that..	+ve	Outpt1
TS2(R2)	Ensure that..	+ve	Outpt2
TS4(R4)	Ensure that..	-ve	Outpt3

Identify the test cases for each scenario. For each of the possible value set, identify the actual value sets and then design the test cases. For each scenario, there should be a minimum of one test cases. All test cases for positive test scenario are considered as +ve test cases and all for negative test scenario is considered as –ve test cases.

Let us take an example of simple ticket reservation system.
Expected behaviors,

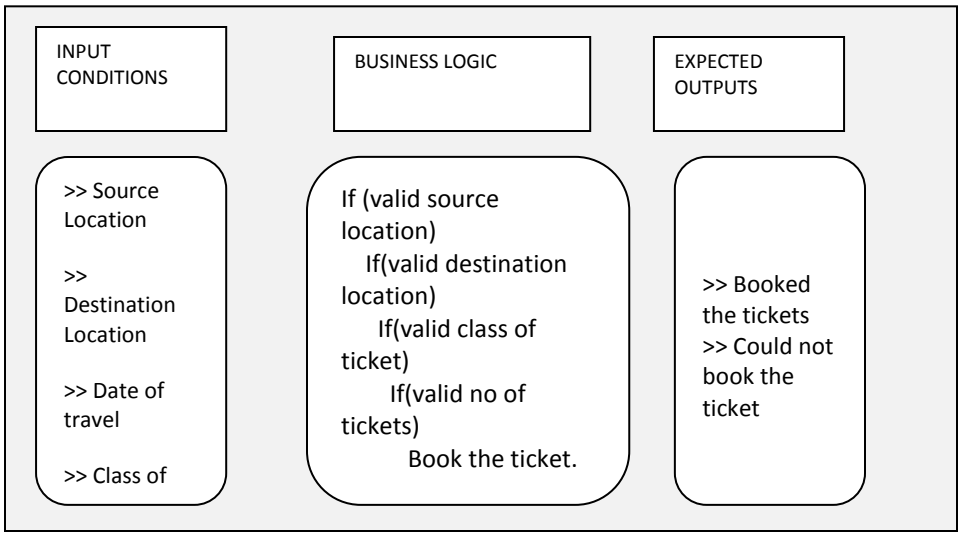
- User should be able to book ticket from source to destination on the specified date if tickets are available on the specified class of travel.

Let us identify the conditions that govern this behavior.
What the specifications says is, use should be able to book a ticket from source to destination if

- Source is valid
- Destination is valid
- Date of travel is valid (cannot be less than the current date)
- Valid class of travel (may be available list of travel class)
- Valid no of tickets (greater than 0 and max of available/allowed booking per person)

If all the above conditions satisfy, then the user should be able to book the ticket.

STEP 1 - Let us use the behavior modeling (Box Model technique) to list out the conditions and expected



Please note that it is not mandatory to use Box Model to identify the business conditions and its possible values. This is one way to represent the connection between input conditions/business logic and the expected outputs. Flow chart will be another way of representation where each condition and its possible values can be identified and tracked to the expected end result.

What are the possible values for each of these conditions?

- Source location – any location from the given list.
- Destination location – any location from the given list (other than the selected source)
- Date of travel - current day to the max allowed reservation date.
- Class of travel – any class from the available list. (SL, AC, CC, 2S)
- No of tickets – 1 to max allowed per person

Combine all the conditions with its possible values result into each possible flows of booking.
Should we need to consider all the possible values of each condition while making combination of conditions? Not necessary. This has to be decided based on the context of EUT.
Best way to identify the class of values to be considered is apply Equivalence partitioning.
Eg: Source/Destination location – we can categorize locations into different regions like within state/outside state/ within district etc. then take one sample value from each categorization.

STEP 2 - Let us create a Decision Table to come up with the test scenarios.

Input conditions	Possible values		Rules				
	+ve set	-ve set	R1	R2	R3	R4	R5
Source location	SR1(SL1-5), SR2(SL6-8), ...SRN		SR1	SR2	SR2	SR1	SRN
Destination location	DR1(DL1-3), DR2(DL4-9)…DRN		DR1	DR2	DR1	DR2	DRN
Date of travel	Valid range	Invalid range	Valid	Valid	Valid	Valid	Invalid
Class of travel	SL,AC,CC,2S		SL	CC	2S	SL	AC
No of tickets	Valid range(#<=available)	0, Invalid range (#> available)	Valid	Valid	0	Invalid range	Valid
Expected Output							
Ticket has booked			X	X			
Ticket Not booked (Invalid no of ticket range)					X	X	
Ticket Not booked (Invalid date of travel)							X

Note: Here range SR1 (SL1-5) represents locations 1-5 where it belongs to a certain region.

At this context, the customized PDT for the EUT (E1) can be as follows,

- That the ticket may not be booked with all the necessary conditions provided. (E1PDT1)
- That the ticket may be booked even when one or more conditions are missed. (E1PDT2)

STEP 3 - Each rule in the DT represents the test scenario. We need write a descriptive meaning for each scenario.

TS ID	Scenario Description	Type	PDT
TS1	Ensure that ticket can be reserved from SR1 to DR1 with valid details.	+ve	E1PDT1
MTS2	Ensure that ticket can be reserved from SR2 to DR2 with valid details	+ve	E1PDT1
TS3	Ensure that the tickets cannot be booked if the no of tickets is 0	-ve	E1PDT2
TS4	Ensure that the tickets cannot be reserved if the no of tickets are more than the available tickets	-ve	E1PDT2
TS5	Ensure that the tickets cannot be reserved if the date of travel mentioned is outside the allowed reservation date	-ve	E1PDT2
---	----	--	
TSN			

STEP 4 - Now for each of the scenarios we have to design test cases using actual test data for each input.

Note: Only basic attributes are mentioned here for test case template. There are other attributes like priority, execution stage, execution model etc which can be included as per the requirement/organizational teat case template.

There should be at least one test case for each test scenario.

TCID	Test case objective	Test data	Expected result
TS1.TC1	Verify that ticket can be reserved from SL1 to DL1 with valid details	SL1,DL1,today,SL,(>0 and less than available tickets)	Ticket booked
TS1.TC2	Verify that ticket can be reserved from SL3 to DL3 with valid details	SL3,DL3,today+1,SL,(>0 and less than available tickets)	Ticket booked
--	---		
TS2.TC1			
TS2.TC2			
--			