05 Specification-based test design (Cont.)

PowerPoint by Wanida Khamprapai

State transition testing

Many times, application output for the same input is different depending on what has happened before. State transition testing (Carvalho and Tsuchiya, 2014) is one of the most well-known methods for black-box, specification-based software testing. In this testing method, test cases are found using state charts or other state transition-based requirements as a guide. A test case could be a sequence of inputs or a sequence of signals.

Example:

- An ATM system function locks the account if the user enters the wrong PIN three times.
- If your account has 300 baht, you request to withdraw 300 baht. You are unable to submit the same request again, due to insufficient amounts in your account. This is because the initial withdrawal request changes the status of your account.
- A word processing application has two states – Open and Close. You may utilize the 'Close' menu while the document is open. If you select 'Close', you are unable to do it again.

Ref.: C. de Souza Carvalho and T. Tsuchiya, "Coverage Criteria for State Transition Testing and Model Checker-Based Test Case Generation," 2014 Second International Symposium on Computing and Networking, Shizuoka, Japan, 2014, pp. 596-598, doi: 10.1109/CANDAR.2014.111.

State transition testing

Objective:

- To evaluate how the system reacts to different inputs.
- To evaluate the impact of previous values.
- To evaluate how the transition state of application has changed.
- To evaluate the effectiveness of system.

State transition testing

Meeting the requirements

- When a tester has to test an application for a limited set of input values.
- When the tester tries to test the order in which events happen in the program being tested. In other words, this will enable the tester to examine how the application behaves when given a series of input data.
- If the system under test is dependent on prior occurrences or values.

State transition testing

A state transition diagram, sometimes referred to as a state machine diagram or a state chart diagram, is a graphical representation used in software engineering to show how an object or system behaves as it experiences different states or conditions and transitions between them in response to events.

States

These depict different conditions or modes in which the system may be. Typically, each state is represented as a box or node with labels.

Transitions

These are the arrows or lines between the states that indicate the change from one state to another. Transitions are labeled with the triggering event or condition that causes the transition.

Events

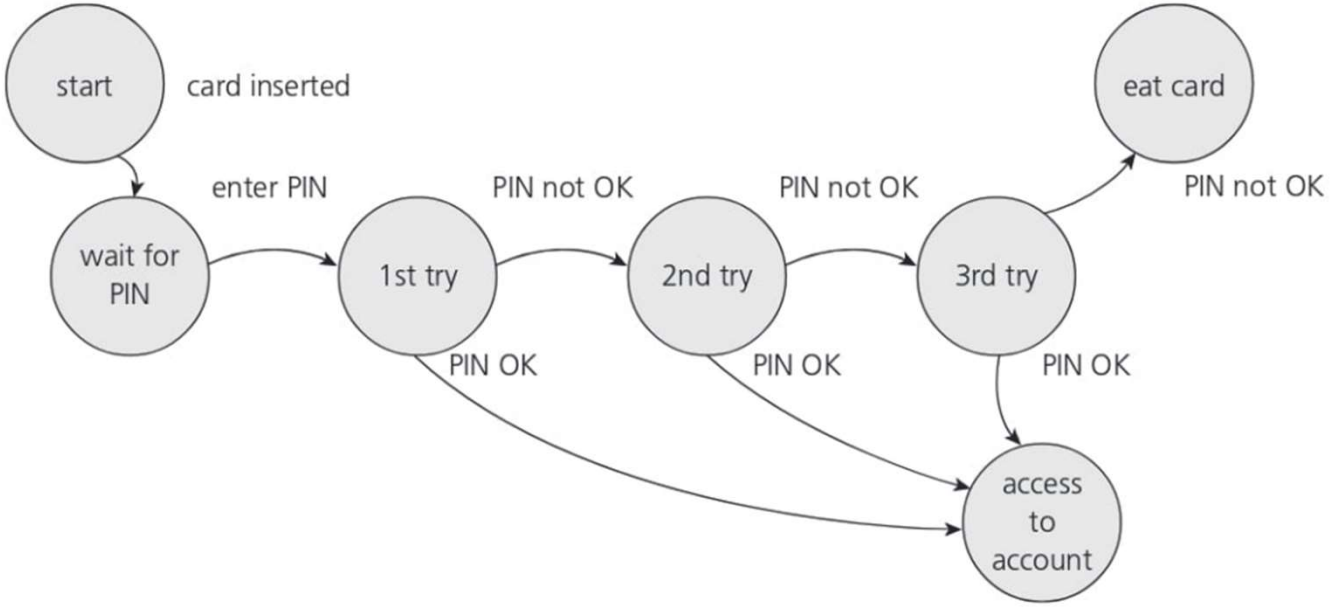
These are the triggers that cause the system to transition from one state to another. Events can be internal (e.g., a timer reaching a certain value) or external (e.g., user input).

Actions

These are the activities or behaviors associated with states or transitions. They specify what happens when a system enters or exits a particular state, or when a transition occurs.

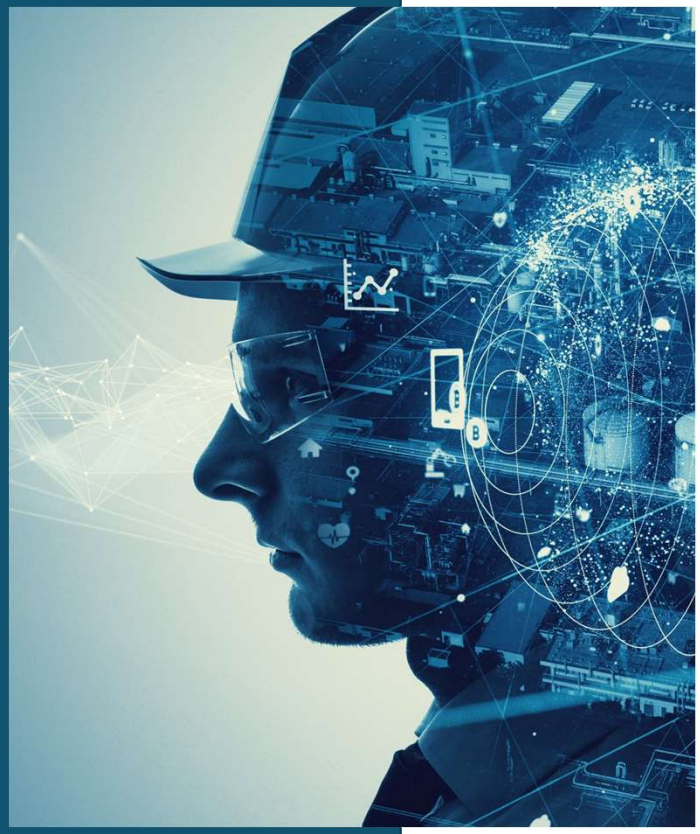
State transition testing

Example: An ATM system function locks the account if the user enters the wrong PIN three times.



State diagram for PIN entry (Graham et.al., 2021)

Ref.: Graham, D., Black, R., van Veenendaal, E. 2021. Foundations of Software Testing ISTQB certification. 4th edition. Cengage Learning.



State transition testing

The state table (ATM system) may be summarized as follows:

State/Event	Event1 (Insert card)	Event2 (Valid PIN)	Event3 (Invalid PIN)
S1: Start	S2		
S2: Wait for PIN		S6	S3
S3: 1st try invalid		S6	S4
S4: 2nd try invalid		S6	S5
S5: 3rd try invalid			S7
S6:Access account		?	?
S7: Eat card	S1 (for new card)		

State transition testing

Test case for ATM system

Test case 1: Successful to access the account with 1st try valid PIN

1) Start in the card inserted
2) Enter the valid PIN
3) Access the account

Test case 2: Successful to access the account with 2nd try valid PIN

1) Start in the card inserted
2) Enter the invalid PIN
3) Try to enter PIN again, enter the valid PIN
4) Access the account

Test case 3: .....

Test case 4: .....

State/Event	Event1 (Insert card)	Event2 (Valid PIN)	Event3 (Invalid PIN)
S1: Start	S2		
S2: Wait for PIN		S6	S3
S3: 1st try invalid		S6	S4
S4: 2nd try invalid		S6	S5
S5: 3rd try invalid			S7
S6:Access account		?	?
S7: Eat card	S1 (for new card)		

Use case testing

A use case is a description of a particular use of the system by an actor. The actor may be something that the system interfaces with. Actors are generally users (people) but they may also be communication links or subsystems or other systems. Each use case provides a sequence of steps that describes the interactions the actor has with the system in order to achieve a specific task (transaction) or, at least, produce something of value to the actor.

Components of scenario:

Pre-conditions

- Anything that must happen before the scenario can start
- It describes the state in which the system must be before the scenario start

Steps

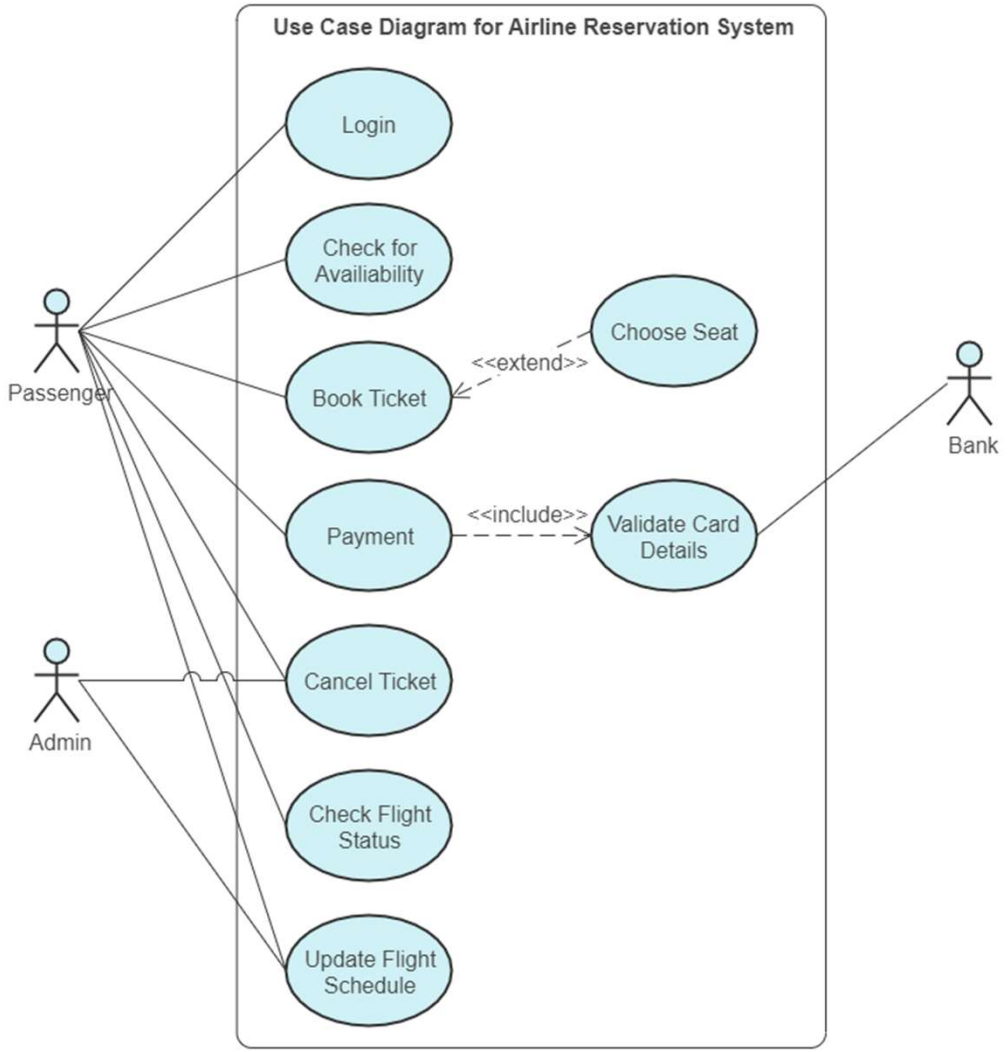
- All interactions between the system and actors that is necessary to complete the scenario

Post-conditions

- Anything that must be true after the scenario is completed
- What state the system acquires after the scenario has been completed successfully

Use case testing

Example: The booking for the flight



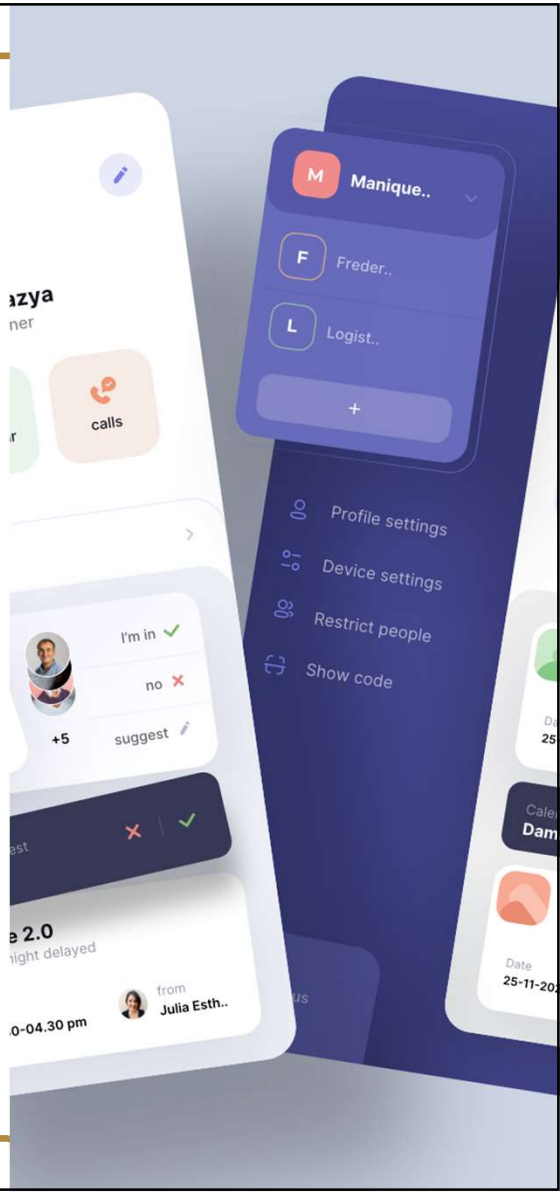
Use case testing

Example: The booking for the flight

The scenario: Customer makes flight reservation

Pre-conditions	Steps	Post-conditions
• Round trip is available • Customer has logged in successfully • Credit card is valid	A - Select trip type (return journey) A - Enter travel details A - Press 'search' button S - Available options displayed A - Select one option S - Passenger information page displayed A - Fill Passenger details S - Payment page displayed A - Fills credit card details and press 'Book Tickets' S - interacts with Accounting system S - Confirmation message is displayed	• Reservation has been made • Seats are assigned and removed from inventory • Credit card transaction is posted • Customer is still logged in • 'Update' and 'Cancel' buttons are enabled

***Note*:** In steps, A- Represents action taken by Actor and S- represents response by the System.



Use case testing

Use case testing

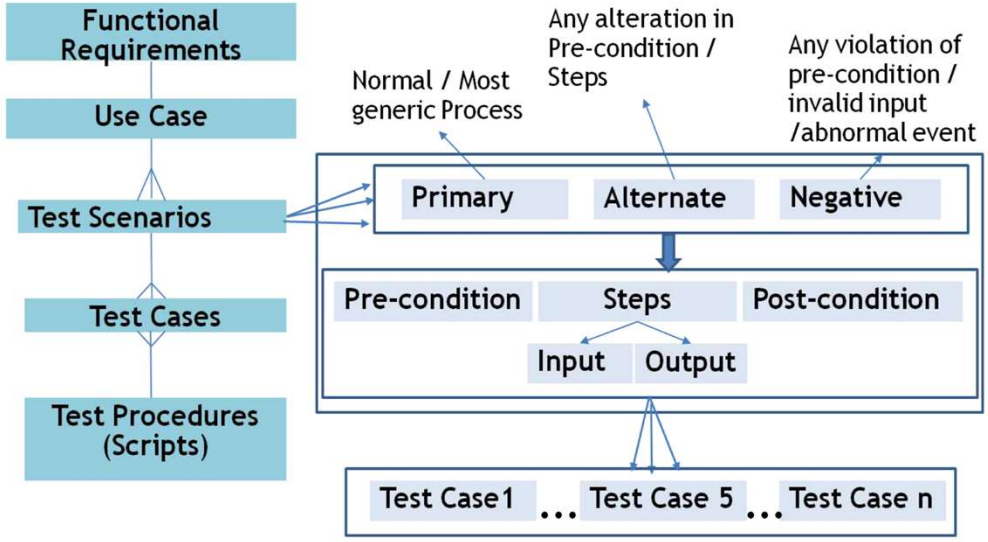
- A black box test design technique in which test cases are designed to execute scenarios of use cases. It helps us identify test cases that exercise the whole system considering all use cases with every scenario from start to finish. They are used mostly at the system and acceptance testing levels & useful for finding defects in the real-world use of the system.
- Use Case-based testing can uncover integration defects caused by the incorrect implementation of a solution by a component considering the inputs provided or processing done by the previous component. Please note that technical integration problems may be identified during integration testing, but business rule-based integration may be identified more accurately using this technique.

Example: Cancellation of a ticket

Cancellation of a ticket may be technically possible but from a business rule perspective, the ticket can be canceled, and the refund can be given only if the cancellation is done before 24 hours of travel time. Such defects can be uncovered by relevant primary and alternate scenarios.

Use case testing

We can derive multiple test cases from each scenario with different combinations of data inputs. In this case, the pre-condition or steps followed may not change but the data used may change. The system test cases for various transactions are derived based on these use cases as per the process described below.



Each variant in the pre-condition and/or input becomes a test case.

Use case testing

If the use cases are written in detail, they can be directly used for testing. It is however advisable to derive and document test cases from each use case scenario instead of directly doing testing using the Use case. This is because; you can provide actual input data to be used during testing so that if a defect occurs, it is possible to reproduce the defect using the same input data.

It is also likely that you may have to use different input data where expected results don't just depend on the steps followed but also on the data entered. So, it is likely that there are more test cases for a single scenario.

Use case testing

Example: The booking for the flight

Test case ID: TC0001

Description: Verify that travel reservation is completed successfully by a passenger

Pre-condition:

1) Passenger has a valid credit card for booking
2) Round Trip is available between two cities – Phuket and Chiang Mai.
3) A user is successfully logged in with User ID: passenger1 and Password: passenger1 and the Home page is displayed which contains a button 'Reserve Ticket'.

#Step	Step and test data	Expected results
1	Enter trip type: Round trip	Accepts
	From location: Phuket	Accepts
	To location: Chiang Mai	Accepts
	Travel date: August 25, 2023	Accepts
	Return date: August 27, 2023	Accepts
	Number of passenger: 0	Error
	Number of passenger: 2	Accepts
	Select class: Economy	Accepts
	Press 'Search' button	System displays various flight options with seats available and price/cost

Use case testing

Example: The booking for the flight (Cont.)

#Step	Step and test data	Expected results
2	User selects a specific flight option by clicking 'Book' button against that flight	System displays forms to input passenger details
3	Enter passenger details (Name, gender, age) for both passengers 1. Somying Srisamorn, F, 28 2. Fluk Srisamon, M, 26 Press 'Submit' button	System verifies and accepts the details and displays payment form
4	Provide credit card details Credit card agency: Krungthai Credit card number: 9999 9999 9999 9999	Control goes to the accounting system, credit card details are verified and then a confirmation message is displayed.

Post Condition: Verify that

1) Reservation is done,
2) Seats are removed from inventory,
3) Credit card transaction is posted,
4) Customer is still logged into the system,
5) 'Update' and 'Cancel' buttons are enabled (so that user can update or cancel the reservation if required).

Use case testing

Note:

1) Each step in the scenario is written separately with action (including data) and expected result
2) You will notice that steps are at a high level as compared to Unit testing because it is assumed that unit testing was already done and individual field level rules are assumed to be working fine.
3) If the system is not in a preconditioned state then we need to first take some steps to bring it into a preconditioned state.
4) One needs to Identify inputs in the pre-conditions and in steps and provide exact data to be used.
5) Output described in each step and post-conditions are expected results and to be verified. They are all to be considered as separate test cases and in some cases may have to be documented separately.
6) Each variant in the pre-condition or steps becomes a separate test case. For example, the user has a valid debit card instead of a credit card or does not confirm the details after entering details and changes the data in between or enters age such that the passenger becomes a senior citizen

Use case testing

Types of scenarios

Each use case has two types of scenarios – a) Primary and b) Alternate

Primary/Basic Scenario

It is the most common way for a use case to happen; as if everything goes as per normal process. It represents normal functionality described by the use case.

For example, online reservation by the passenger with all valid details using normal payment mode

Alternate Scenario

This is a scenario where the precondition steps, actions, or sequence of actions are different from the one described in the primary scenario. This includes special cases or exceptional conditions or even error conditions.

For example, the User enters all the details, but later on changes their mind and modifies details before confirmation. User cancel's in between or Flight gets full for a given date/slot or Invalid card is used.

Use case testing

Identifying alternate/error scenarios

An alternate scenario can occur if

- Any other action can also be taken at this time E.g. Any time before a customer clicks 'Book', the customer may click 'Cancel'. Post Condition: Customer still logged in. Pages initialized
- Any other event can happen that can cause the system to behave differently E.g. Accounting system can notify you that the credit card has been declined. Post Condition: Allow to correct information
- Anything that can go wrong (by the system, by the user or by another actor)? E.g. System crashes before the system displays a confirmation message. Post condition: system saves partially entered data, such as Ticket entry is done but actual reservation does not happen (seat allocation does not happen)

Developing Negative (Destructive) test cases based on Scenarios

There are two sources for deriving negative test cases

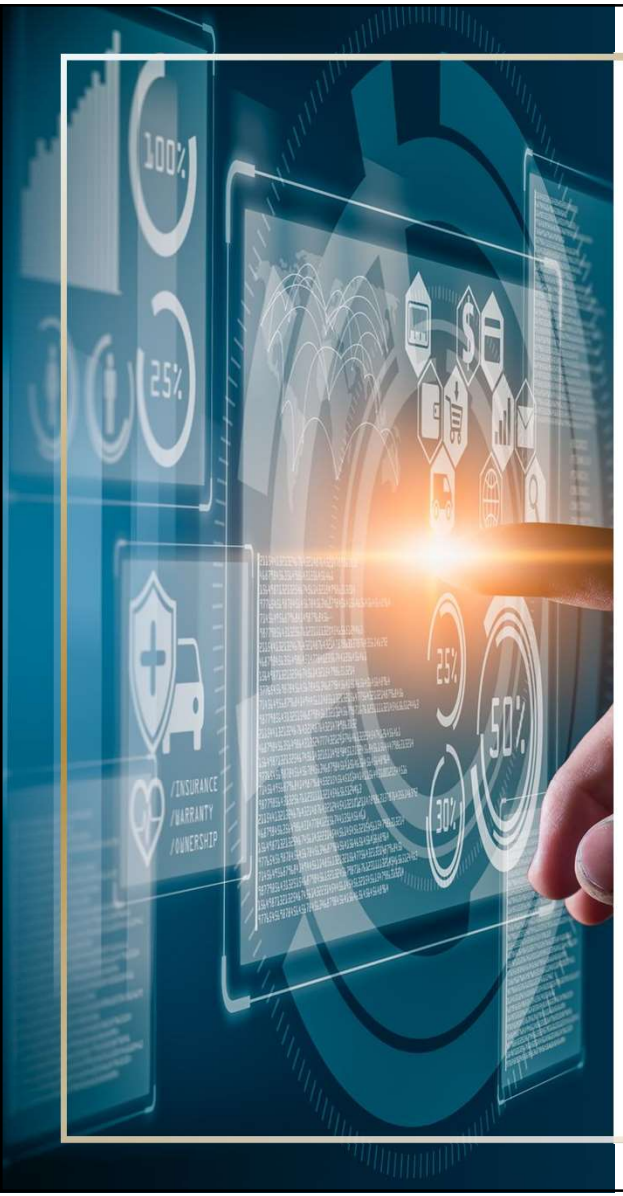
- Test Cases based on alternate scenarios with invalid data
- Test cases that violate items in pre-condition

Example

- Cancel reservation that does not exist
- Cancel reservation where flight time is < 24 hours
- Cancel reservation for which boarding pass was already issued

Experience-Based Test Design

- **Error Guessing**: Test design techniques will help us identify test cases for finding potential defects. The experience and intuition of a person can also find many errors and sometimes those errors may not be even easily possible through various techniques discussed above.
- **Exploratory Testing**: About exploring, finding out about the software (its functionality and features - what it does, what it doesn't do), and also testing to find what works and what doesn't work. It also involves learning about its strengths and its weaknesses.



Experience-Based Test Design

Error Guessing

Error guessing is based on the number of aspects described below

- Domain or application experience or testing experience of the person.
- Defect history – if someone is good at analyzing the defects found in previous modules or some other applications, they may anticipate similar errors in the new module or application.
- Understanding of possible errors or error-prone situations primarily based on technical knowledge. Empty or null lists/strings, blanks or null characters in strings zero occurrences negative numbers
- defects around exception situations. Requirements around exceptional situations are generally missed and some experienced people with required domain knowledge can quickly point out those aspects.
- Defects around changes in functionality – business rules. When the functionality changes or business rules change, the user knows what was earlier valid and now invalid or what is likely to get impacted because of change.

All the tests identified based on various aspects mentioned above should be documented and used during test execution.

Questions & Answers