UNIT-3

Creating Test Cases from Requirements and Use Cases

USE CASE DIAGRAM AND USE CASES

Use case diagram is also used along with use cases to explain the functionality of the system. This is a graphical representation and gives the top view of the system along with its users and use cases. Use case diagram may be decomposed into a further level of abstraction. Use cases and use case diagrams are normally used together to define the behaviour of a system.

A use case diagram visually explains what happens when an actor interacts with the system. Actor represents the role of a user that interacts with the system. They are outsiders to the system and can be human beings, other systems, devices, etc. We should not confuse the actors with the devices they use. Devices are mechanisms that actors use to communicate with the system, but they are not actors themselves. We use the computer keyboard for interaction; in such a case, we are the actors, and not the keyboard that helps us to interact with the computer. We use the printer to generate a report; in such case, the printer does not become an actor because it is only used to convey the information. However, if we want to take information from an external database, then, this database becomes an actor for our system.

A use case is started by a user for a specific purpose and completes when that purpose is satisfied. It describes a sequence of actions a system performs to produce an observable output for the interacting user (actor). The importance of a use case is effectively given by Greg Fournier [FOUR09] as:

> "The real value of a use case is the dynamic relationship between the actor and the system. A well written use case clarifies how a system is used by the actor for a given goal or reason. If there are any questions about what a system does to provide some specific value to someone or something outside the system, including conditional behaviour and handling conditions of when something goes wrong, the use case is the place to find the answers."

A use case describes who (any user) does what (interaction) with the system, for what goal, without considering the internal details of the system. A complete set of use cases explains the various ways to use the system. Hence, use cases define expected behaviours of the system and helps us to define the scope of the system.

- ### Identification of Actors

An actor represents the role of a user that interacts with the system. An actor may be a human being or a system that may interact with a use case keeping in view a particular goal in mind. Some of the examples of the actors used in the case study of 'University registration system' (discussed in Section 5.7) are given as:

(i) Administrator
(ii) Student
(iii) Faculty
(iv) Data entry operator

The URS will allow the above actors to interact with the system with their specific roles. Depending upon the role, an actor will be able to access only the defined information from the system. We may define the role of every actor as:

(i) Administrator: Able to add, modify or delete a programme, school, scheme, paper, student, faculty and login information. Able to generate student registration card and other reports.
(ii) Student: Able to add and modify his/her details and register for papers to be studied in the current semester. Able to generate student registration card.

(iii) Faculty: Able to generate desired reports.
(iv) Data entry operator: Able to add, modify or delete student and faculty information.

The identification of actors with their specified roles may define the scope for every actor and its expected actions. Every actor may interact with one or more use cases designed for the specified purpose.

• Identification of Use Cases

Whenever we design a system, we expect some functionalities from the system. To achieve such functionalities, many actors interact with the system with some specified expectations. The actor acts from the outside and may provide some inputs to the system and expect some outputs from the system. After the finalization of requirements, we expect to create use cases for the system. Some guidelines for the creation of use cases are given as:

(i) Every use case should have a specified functionality.
(ii) Every use case will have a name. Every name should be unique, meaningful and purposeful to avoid confusion in the system.
(iii) One or more actors will interact with a use case.
(iv) An actor will initiate a use case.
(v) The role of actors should always be clearly defined for every use case. Who will initiate the use case and under which conditions, should be clearly specified.

We should always remember that use cases describe who (actor) does what (interaction) with the system, for what goal, without considering the internal details of the system.

In the URS, we may identify the following use cases for each of the actors.

S. No.	Use Case	Actors	Description
1.	Login	Administrator, student, faculty, DEO	Login
			Change password
2.	Maintain School Details	Administrator	Add School
			Edit School
			Delete School
			View School
3.	Maintain Programme Details	Administrator	Add Programme
			Edit Programme
			Delete Programme
			View Programme
4.	Maintain Scheme Details	Administrator	Add Scheme
			Edit Scheme
			Delete Scheme
			View Scheme

(*Contd.*)

(*Contd.*)

S. No.	Use Case	Actors	Description
5.	Maintain Paper Details	Administrator	Add Paper Edit Paper Delete Paper View Paper
6.	Maintain Student Details	Administrator, DEO	Add Student Edit Student Delete Student View Student
7.	Maintain Faculty Details	Administrator, DEO	Add Faculty Edit Faculty Delete Faculty View Faculty
8.	Maintain Student Registration Details	Administrator, student	Add Student Information Select Papers offered by the programme
9.	Generate Report	Administrator, faculty	Roll number wise Programme wise Semester wise Paper wise
10.	Generate Registration Card	Administrator, student	Printing of Registration card

We should identify use cases very carefully, because it has serious implications on the overall design of the system. Use cases should not be too small or too big. The basic flow and all alternative flows should also be specified. Identifying and writing good use cases means providing better foundations for the intended system.

• Drawing of Use Case Diagram

The use case diagram shows actors, use cases and the relationship between them. It gives the pictorial view of the system. In use case diagram, actors are represented as stick figures and use cases are represented as ovals. The relationship between an actor and a use case is represented by a solid arrow. The components of the use case diagram are given in Figure 6.1.

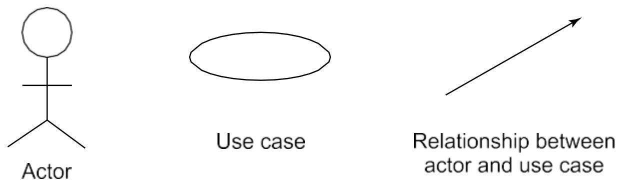
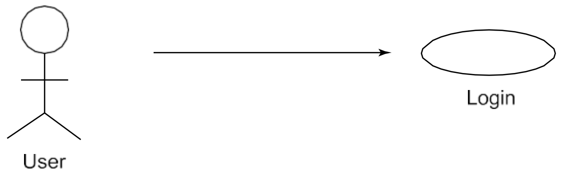


Figure 6.1. Components of use case diagram

Actors appear outside of a system. A relationship is shown by an arrow and is between the actor and a use case and vice versa. A relationship between a ‘user’ (actor) and ‘login’ use case is shown as:



If the system is small, one diagram may be sufficient to represent the whole system, but for large systems, we may require to represent the whole system in many diagrams. The use case diagram of the URS is given in Figure 6.2. There are ten use cases and four actors. The administrator interacts with all use cases, whereas a student may interact only with ‘Login’, ‘Maintain student registration’ details and ‘Generate registration card’ use cases.

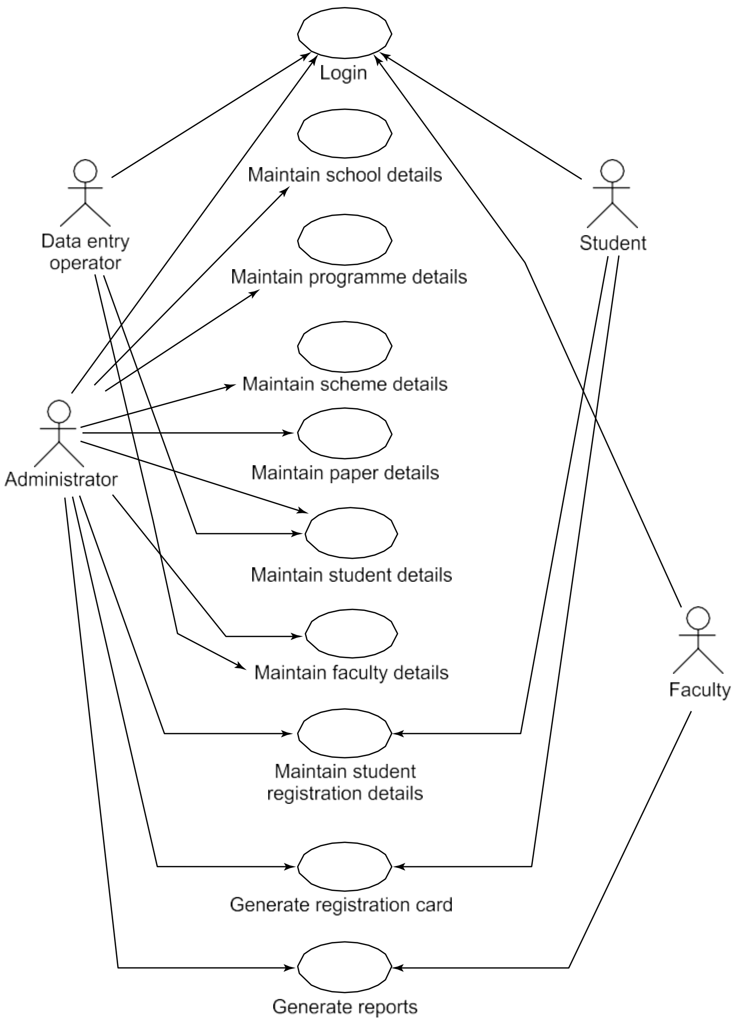


Figure 6.2. Use case diagram of the URS

• Writing of Use Case Description

Actors interact with the use cases for predefined purposes. Hence, each actor does something with the system and the system responds accordingly. Each step is considered as a sequence of events and is called a flow. There are two types of flows:

(i) Basic Flow: It is the main flow and describes the sequence of events that takes place most of the time between the actor and the system to achieve the purpose of the use case.

(ii) Alternative Flows: If the basic flow is not successful due to any condition, the system takes an alternative flow. An alternative flow may occur due to failure of an expected service because of occurrence of exceptions/errors. There may be more than one alternative flow of a use case, but may not occur most of the time. Any alternative flow takes place under certain conditions in order to fulfil the purpose of a use case.

There is no standard method for writing use cases. Jacobson et al. [JACO99] has given a use case template which is given in Table 6.1. This captures the requirements effectively and has become a popular template. Another similar template is given in Table 6.2 which is also used by many companies [COCK01, QUAT03]. All pre-conditions that are required for the use case to perform should be identified. Post conditions, which will emerge after the execution of a use case, should also be defined. The pre-condition is necessary for the use case to start but is not sufficient to start the use case. The use case must be started by an actor when the pre-condition is true. A post-condition describes the state of the system after the ending of the use case. A post-condition for a use case should be true regardless of which flow (basic or any alternative flows) is executed.

Table 6.1. Jacobson's use case template

1. **Brief Description.** Describe a quick background of the use case.
2. **Actors.** List the actors that interact and participate in this use case.
3. **Flow of Events.**
   3.1. **Basic flow.** List the primary events that will occur when this use case is executed.
   3.2. **Alternative flows.** Any subsidiary events that can occur in the use case should be separately listed. List each such event as an alternative flow. A use case can have as many alternative flows as required.
4. **Special Requirements.** Business rules for the basic and alternative flow should be listed as special requirements in the use case narration. These business rules will also be used for writing test cases. Both success and failure scenarios should be described here.
5. **Pre-conditions.** Pre-conditions that need to be satisfied for the use case to perform should be listed.
6. **Post-conditions.** Define the different states in which we expect the system to be in, after the use case executes.
7. **Extension Points.** List of related use cases, if any.

Table 6.2. Alternative use case template

1. **Introduction.** Describe the brief purpose of the use case.
2. **Actors.** List the actors that interact and participate in this use case.
3. **Pre-condition.** Define the condition that needs to be satisfied for the use case to execute.
4. **Post-condition.** After the execution of the use case, different states of the systems are defined here.
5. **Flow of Events.**
   5.1. **Basic flow.** List the primary events that will occur when this use case is executed.
   5.2. **Alternative flow.** Any other possible flow in this use case should be separately listed. A use case may have many alternative flows.
6. **Special Requirements.** Business rules for the basic and alternative flows should be listed as special requirements. Both success and failure scenarios should be described.
7. **Associated use cases.** List the related use cases, if any.

We may write a 'Login' use case description of the URS using the template given in Table 6.2 and the same is given below:

Use Case Description of login use case

1 Introduction

This use case documents the steps that must be followed in order to log into the URS

2 Actors

- Administrator
- Student
- Faculty
- Data Entry Operator

3 Pre-Condition

The user must have a valid login Id and password.

4 Post-Condition

If the use case is successful, the actor is logged into the system. If not, the system state remains unchanged.

5 Basic Flow

It starts when the actor wishes to login to the URS.

(v) The system requests that the actor specify the function he/she would like to perform (either Login, Change Password).

(vi) Once the actor provides the requested information, one of the flows is executed.
- If the actor selects 'Login', the Login flow is executed.
- If the actor selects 'Change Password', the Change Password flow is executed.

Basic Flow 1: Login

(i) The system requests that the actor enters his/her login Id and password information.
(ii) The actor enters his/her login Id and password.
(iii) The actor enters into the system.

Basic Flow 2: Change Password

(i) The system requests that the actor enter his/her login Id, old password, new password and confirm the new password information.
(ii) The actor enters login Id, old password and new password, and confirms the new password information.
(iii) The system validates the new password entered and the password change is confirmed.

(*Contd.*)

Use Case Description of login use case

6 Alternative flows

Alternative Flow 1: Invalid login Id/password

If in the Login basic flow, the actor enters an invalid login Id and/or password or leaves the login Id and /or password empty, the system displays an error message. The actor returns to the beginning of the basic flow.

Alternative Flow 2: Invalid Entry

If in the Change Password basic flow, the actor enters an invalid login Id, old password, new password or the new password does not match with the confirmed password, the system displays an error message. The actor returns to the beginning of the basic flow.

Alternative Flow 3: User Exits

This allows the user to exit during the use case. The use case ends.

7 Special Requirement

None

8 Associated use cases

None

The use cases describe the flow of events which include the basic flow and alternative flows and this description should be long enough to clearly explain its various steps. The basic flow and alternative flows are written in simple and clear sentences in order to satisfy all the stakeholders. A login use case, which allows entering the correct login Id and password, has two basic flows (the user is allowed to enter after giving the correct login Id and password and change password) and many alternative flows (incorrect login Id and/or password, invalid entry and user Exits). If an alternative flow has other alternative flows, the use case may have a longer description of the flows and may become a complex use case.

We should write the basic flow independently of the alternative flows and no knowledge of alternative flows is considered. The basic flow must be complete in itself without reference to the alternative flows. The alternative flow knows the details of when and where it is applicable which is opposite to the basic flow. It inserts into the basic flow when a particular condition is true [BITT03].

GENERATION OF TEST CASES FROM USE CASES

We may start writing the test cases as soon as use cases are available. This may happen well before any source code is written. It is always advisable to follow a systematic approach for the generation of test cases. These test cases may give us better coverage of the source code during testing. Any adhoc way may generate many duplicate test cases that may result in to poor coverage of the source code. A systematic approach may include the following steps:

(i) Generation of scenario diagrams
(ii) Creation of use case scenario matrix
(iii) Identification of variables in a use case
(iv) Identification of different input states of available variables
(v) Design of test case matrix
(vi) Assigning actual values to variables

If all steps are followed in the above mentioned sequence, we may have a good number of planned and systematic test cases which will result in an efficient and effective testing process.

- **Generation of Scenario Diagrams**

A use case scenario is an instance of a use case or a complete path through the use case [HEUM01]. The basic flow is one scenario and every alternative path gives another scenario. Use case scenarios may also be generated due to various combinations of alternative flows. The basic and alternative flows for a use case are shown in Figure 6.3.

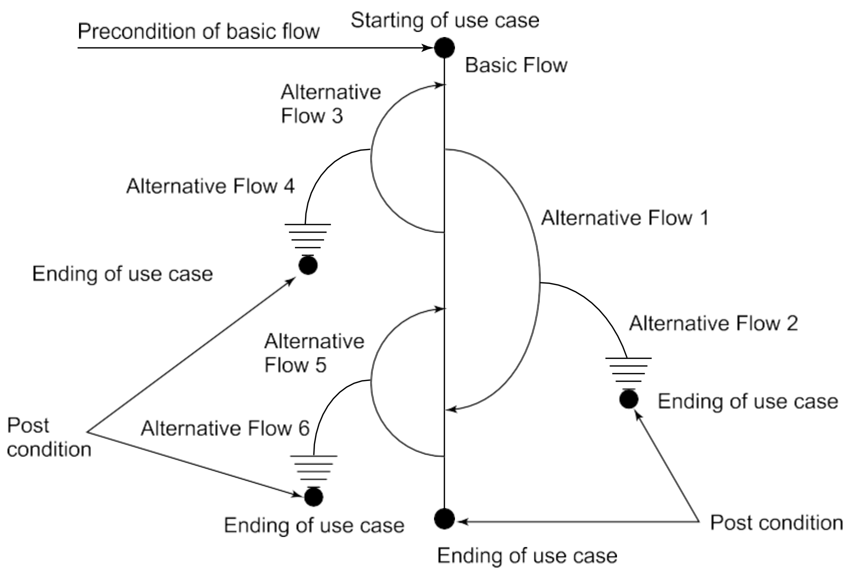


Figure 6.3. Basic and alternative flows with pre- and post-conditions

The basic flow is represented by a straight arrow and the alternative flows by the curves. Some alternative flows return to the basic flow, while others end the use case. At the end of the basic flow, a post-condition is generated while at the starting of the basic flow, a pre-condition is required to be set.

There are the following basic and alternative flows in login use case:

Basic flow:

(i) Login
(ii) Change password

Alternative flows:

(i) Invalid Login Id/password
(ii) Invalid entry
(iii) User exits

The basic and alternative flows for login use case are given in Figure 6.4. In Figure 6.4 (a), there is one basic flow which will be executed when the correct login Id and password are given. This basic flow is expected to be executed most of the time. If any input (Login Id or password) is invalid, then the alternative flow will be executed and the actor will return to the beginning of the basic flow. If at any time, the user decides to exit, then alternative flow 3 will be executed.

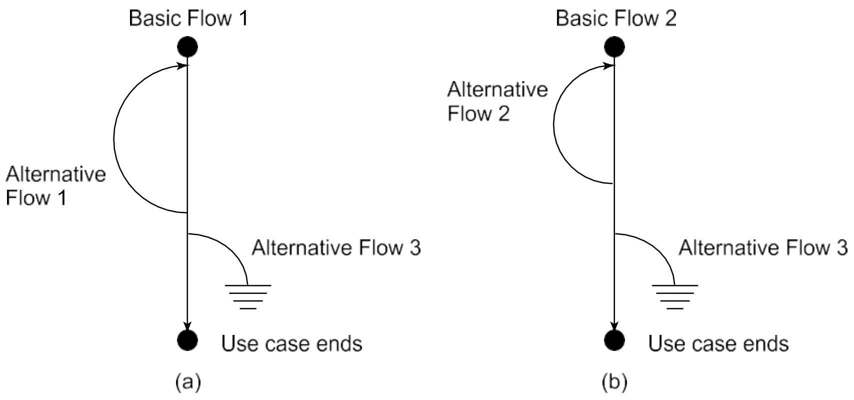


Figure 6.4. Basic and alternative flows for login use case (a) Login (b) Change password
Alternative Flow 1: Invalid login Id/password
Alternative Flow 2: Invalid Entry
Alternative Flow 3: User exits

• Creation of Use Case Scenario Matrix

Use case scenario diagrams generate many scenarios due to the basic flow, every alternative flow along with the basic flow and various combinations of the basic and alternative flows. A scenario matrix gives all possible scenarios of the use case scenario diagram. The scenario matrix given in Table 6.3 gives all possible scenarios for the diagram given in Figure 6.3.

Table 6.3. Scenario matrix for the flow of events shown in Figure 6.3

Scenario 1	Basic Flow			
Scenario 2	Basic Flow	Alternative Flow 1		
Scenario 3	Basic Flow	Alternative Flow 1	Alternative Flow 2	
Scenario 4	Basic Flow	Alternative Flow 3		
Scenario 5	Basic Flow	Alternative Flow 3	Alternative Flow 4	
Scenario 6	Basic Flow	Alternative Flow 3	Alternative Flow1	
Scenario 7	Basic Flow	Alternative Flow 3	Alternative Flow 1	Alternative Flow 2
Scenario 8	Basic Flow	Alternative Flow 5		

(*Contd.*)

(*Contd.*)

Scenario 9	Basic Flow	Alternative Flow 5	Alternative Flow 6	
Scenario 10	Basic Flow	Alternative Flow 3	Alternative Flow 5	
Scenario 11	Basic Flow	Alternative Flow 3	Alternative Flow 5	Alternative Flow 6

In the basic and alternative flows scenario diagram of the login use case, there are six possible paths (see Figure 6.4). These six paths become six scenarios of login use case and are given in Table 6.4. Moreover, the path 'Basic Flow 1, Alternative Flow 1 and Alternative Flow 3' is impossible as per the use case description, because after giving incorrect login ID/ password, the actor returns to the beginning of the basic flow 1. Similarly, both 'Basic Flow 2, Alternative Flow 2 and Alternative Flow 3' are also impossible. All valid combinations of the basic flow and the alternative flows may be generated as per given use case description.

Table 6.4. Scenario matrix for the login use case

Scenario 1- Login	Basic Flow 1	
Scenario 2- Login alternative flow: Invalid login Id/password	Basic Flow 1	Alternative Flow 1
Scenario 3- Login alternative flow: User Exits	Basic Flow 1	Alternative Flow 3
Scenario 4- Change Password	Basic Flow 2	
Scenario 5- Change password alternative flow: Invalid Entry	Basic Flow 2	Alternative Flow 2
Scenario 6- Change password alternative flow: User Exits	Basic Flow 2	Alternative Flow 3

• Identification of Variables in a Use Case

We have to identify all input variables which have been used in every use case. For a login use case, we use 'login Id' and 'password' as inputs for entering into the use case. These are two input variables for the 'Login' use case. A variable may also be used as a selection variable where many options are available for a variable. A selection variable may be values of buttons available which provide input to the use case at some intermediate step of the use case. For example, 'Updation confirmed?' will provide two options to an actor 'Yes/No' and thus based on this selection input, the decision on whether updation is to be made or not, is made. We may select a semester from a drop down menu. The following variables are used in the login use case:

(i) Login Id
(ii) Password
(iii) Old password
(iv) New password
(v) Confirm password

These variables are inputs to the system and when an input or combination of specified inputs is given, a particular behaviour (output) is expected from the system. Hence, identification of these variables is important and helps in designing the test cases.

- **Identification of Different Input States of a Variable**

An input variable may have different states and the behaviour of the system may change if the state of a variable is changed. Any variable may have at least two states i.e. valid state and invalid state. If we consider the 'Login Id' variable of the login use case, it is expected that the "Login Id should be alphanumeric of length 11 characters and only digits from 0 to 9 are allowed. Alphabets, special characters and blank spaces are not allowed." Hence, one state is the valid login Id as per the given directions and another state is the invalid login Id which is different from the given directions. There may be many different states of invalid variable. If a variable is in an invalid state, then a different process flow is executed (different alternative flow) or the system gives an unexpected output. The invalid variable should be given different inputs and appropriate values should be given at the time of designing the test cases.

- **Design of Test Case Matrix**

We identify all variables and their different states for the purpose of designing test cases. One way to do so is to create a test case matrix where rows of the matrix contain test cases and the first column contains the scenario name and description and the remaining columns may contain the various input variables including the selection variables. The last column contains the expected output when these inputs are given to the system. A typical test case matrix is given in Table 6.5.

Table 6.5. A typical test case matrix

Test Case Id	Scenario Name and Description	Input 1	Input 2	Input 3 (selection variable)	Expected Output
TC1					
TC2					
TC3					
TC4					

The test case matrix for login use case is given in Table 6.6.

- **Assigning Actual Values to Variables**

In test case matrix, we have written only 'valid input', 'invalid input' and 'not applicable (n/a)' in the input value columns of various variables. Now, we want to assign actual values in these columns in order to use them at the time of execution to get the actual output. We may also add two additional columns with titles 'Actual output' and 'Pass/fail' which will be used at the time of executing these test cases. There should be at least one test case for each scenario, but more test cases may be designed, depending upon availability, time and resources. These test cases may be very useful, effective and are also designed at an early stage of the software development life cycle. The test cases for the 'Login' use case are given in Table 6.7.

Table 6.6. Test case matrix for the login use case

Test case Id	Scenario Name and description	Input 1	Input 2	Input 3	Input 4	Input 5	Expected output	Remarks (if any)
		Login id	Password	Old password	New password	Confirm password		
TC1	Scenario 1- Login	Valid input	Valid input	n/a	n/a	n/a	User is allowed to login	--
TC2	Scenario 2- Login alternative flow: Invalid Entry	Invalid input	Valid input	n/a	n/a	n/a	Login id invalid	Login Id is not in the specified format
TC3		Valid input	Valid input	n/a	n/a	n/a	Login id invalid	Login id does not exist in database
TC4		Valid input	Invalid input	n/a	n/a	n/a	Password invalid	Password is not in the specified format
TC5		Valid input	Valid input	n/a	n/a	n/a	Password invalid	Password does not exist in database
TC6		Invalid input	Invalid input	n/a	n/a	n/a	Login id and password invalid	Login id and Password are not in the specified format
TC7	Scenario 3- Login alternative flow: Exit	Valid /Invalid input	Valid /Invalid input	n/a	n/a	n/a	User comes out of the system	--
TC8	Scenario 4- Change password	Valid input	n/a	Valid input	Valid input	Valid input	User is allowed to change password	Password is changed in the database
TC9	Scenario 5- Change password alternative flow: Invalid entry	Invalid input	n/a	Valid /Invalid input	Valid /Invalid input	Valid /Invalid input	Old password invalid	Login Id is not in the specified format
TC10		Valid input	n/a	Invalid input	Valid /Invalid input	Valid /Invalid input	Old password invalid	If old password is not valid, other entries become 'do not care' entries
TC11		Valid input	n/a	Valid input	Invalid input	Valid /Invalid input	New password invalid	Password is not in the specified format
TC12		Valid input	n/a	Valid input	Valid input	Valid input	Confirm password does not match new password	New and confirm password entries are different
TC13	Scenario 6- Change password alternative flow: Exit	Valid /Invalid input	n/a	Valid /Invalid input	Valid /Invalid input	Valid /Invalid input	User is allowed to exit and returns to login screen	--

Table 6.7. Test case matrix with actual data values for the login use case

Test case Id	Scenario Name and description	Login Id	Password	Old password	New password	Confirm password	Expected output	Remarks (if any)
TC1	Scenario 1- Login	01164521657	Abc123	n/a	n/a	n/a	User is allowed to login	--
TC2	Scenario 2- Login alternative flow: Invalid Entry	1234	Abc123	n/a	n/a	n/a	Login id invalid	Login id is not in specified format which is less than 11 characters
TC3		01164521658	Abc123	n/a	n/a	n/a	Login id invalid	Login id does not exist in database
TC4		01164521657	R34	n/a	n/a	n/a	Password invalid	Password is not in specified format which is less than 4 characters
TC5		01164521657	Abc124	n/a	n/a	n/a	Login id invalid	Password does not exist in database
TC6		1234	R34	n/a	n/a	n/a	Login id/password invalid	Login id and password are not in the specified format. Login id is less than 11 characters and password is less than 4 characters.
TC7	Scenario 3- Login alternative flow: Exit	*	*	n/a	n/a	n/a	User comes out of the system	--
TC8	Scenario 4- Change password	01164521657	n/a	Abc123	Abc124	Abc124	User is allowed to change password	--
TC9	Scenario 5- Change password alternative flow: Invalid entry	01165	n/a	*	*	*	Login Id invalid	Login Id is not in the specified format.
TC10		01164521657	n/a	Abc1	*	*	Old password invalid	Old password does not match the corresponding password in the database. Other entries (new password and confirm password) become 'do not care'.
TC11		01164521657	n/a	Abc123	R12	*	New password invalid	New password is not in the specified format which is less than 4 characters. Other entries (confirm password) become 'do not care'.
TC12		01164521657	n/a	Abc123	Abc124	Abc125	Confirm password does not match new password	--
TC13	Scenario 6- Change password alternative flow: Exit	*	n/a	*	*	*	User is allowed to exit and returns to login screen	--

*: 'do not care' conditions (valid/invalid inputs); n/a: option(s) not available for respective scenario

The use cases are available after finalizing the SRS document. If we start writing test cases in the beginning, we may be able to identify defects at the early phases of the software development. This will help to ensure complete test coverage as a complete test suite will be designed directly from the use cases. This technique is becoming popular due to its applicability in the early phases of software development. The technique is simple and directly applicable from the use cases that are part of the SRS which is designed as per IEEE standard 830-1998.

Example 6.1: Consider the problem statement of the URS as given in chapter 5. Write the use case description of use cases and generate test cases from these use cases.

Solution:

The use case description of 'maintain school details' use case is given below:

1 Introduction

Allow the administrator to maintain details of schools in the university. This includes adding, updating, deleting and viewing school information.

2 Actors

Administrator

3 Pre-Conditions

The administrator must be logged onto the system before this use case begins.

4 Post-Conditions

If the use case is successful, the school information is added/updated/deleted/viewed from the system. Otherwise, the system state is unchanged.

5 Basic Flow

This use case starts when the administrator wishes to add/edit/delete/view school information.

(i) The system requests that the administrator specify the function he/she would like to perform (either Add a school, Edit a school, Delete a school or View a school).

(ii) Once the administrator provides the requested information, one of the flows is executed.

- If the administrator selects 'Add a School', the **Add a School** flow is executed.
- If the administrator selects 'Edit a School', the **Edit a School** flow is executed.
- If the administrator selects 'Delete a School', the **Delete a School** flow is executed.
- If the administrator selects 'View a School', the **View a School** flow is executed.

Basic Flow 1: Add a School

The system requests that the administrator enter the school information. This includes:

(i) The system requests the administrator to enter the:

1. School name
2. School code

(ii) Once the administrator provides the requested information, the school is added to the system.

Basic Flow 2: Edit a School

(i) The system requests the administrator to enter the school code.

(ii) The administrator enters the code of the school. The system retrieves and displays the school name information.

(iii) The administrator makes the desired changes to the school information. This includes any of the information specified in the 'Add a School' flow.

(iv) The system prompts the administrator to confirm the updation of the school.

(v) After confirming the changes, the system updates the school record with the updated information.

(*Contd.*)

(*Contd.*)

Basic Flow 3: Delete a School

(i) The system requests the administrator to specify the code of the school.
(ii) The administrator enters the code of the school. The system retrieves and displays the school information.
(iii) The system prompts the administrator to confirm the deletion of the school.
(iv) The administrator confirms the deletion.
(v) The system deletes the school record.

Basic Flow 4: View a School

(i) The system requests that the administrator specify the school code.
(ii) The system retrieves and displays the school information.

6 Alternative Flows

Alternative Flow 1: Invalid Entry

If in the **Add a School or Edit a School** flows, the actor enters an invalid school name/code or the actor leaves the school name/code blank, the system displays an error message. The actor returns to the basic flow and may re-enter the invalid entry.

Alternative Flow 2: School Code Already Exists

If in the **Add a School** flow, a specified school code already exists, the system displays an error message. The administrator returns to the basic flow and may re-enter the school code.

Alternative Flow 3: School Not Found

If in the **Edit a School or Delete a School or View a School** flows, a school with the specified school code does not exist, the system displays an error message. The administrator returns to the basic flow and may re-enter the school code.

Alternative Flow 4: Edit Cancelled

If in the **Edit a School** flow, the administrator decides not to edit the school, the edit is cancelled and the **Basic Flow** is re-started at the beginning.

Alternative Flow 5: Delete Cancelled

If in the **Delete a School** flow, the administrator decides not to delete the school, the delete is cancelled and the **Basic Flow** is re-started at the beginning.

Alternative Flow 6: Deletion not allowed

If in the **Delete a School** flow, a programme detail of the school code exists then the system displays an error message. The administrator returns to the basic flow.

Alternative Flow 7: User Exits

This allows the user to exit during the use case. The use case ends.

7 Special Requirements

None.

8 Associated Use cases

Login

The Use Case Scenario diagram of ‘Maintain school details’ use case is given in Figure 6.5 and the scenario matrix is given in Table 6.8. The test case matrix is given in Table 6.9 and corresponding matrix with actual data values is given in Table 6.10.

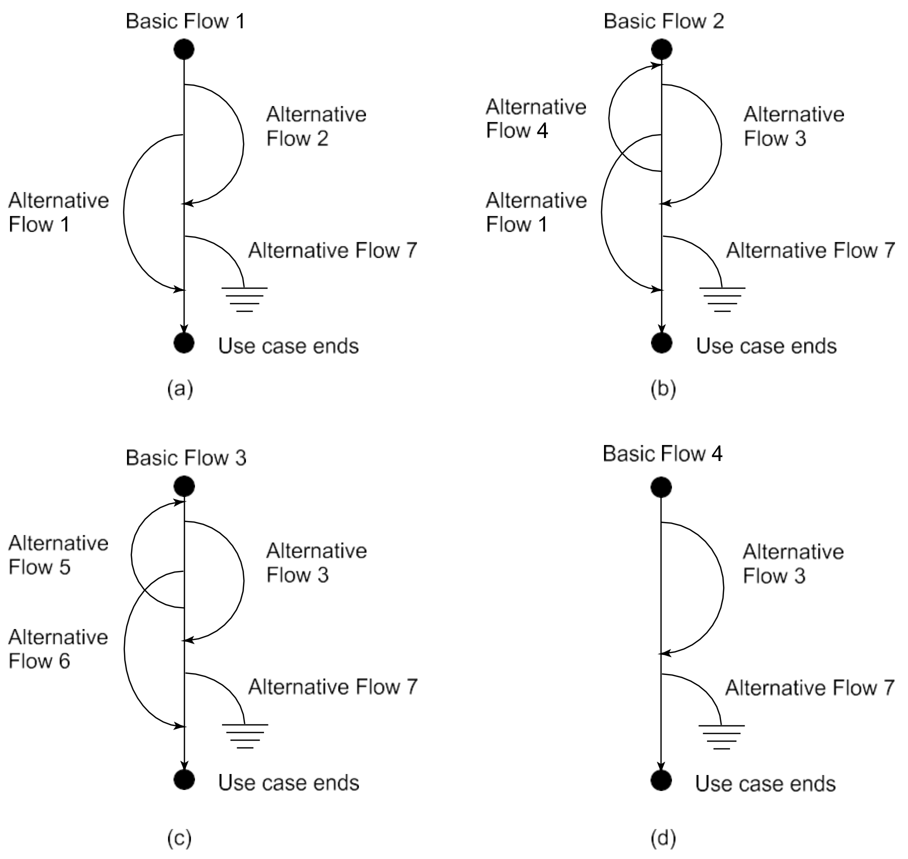


Figure 6.5. Basic and alternative flows for 'maintain school', 'programme', 'scheme', 'paper', or 'student details' use cases (a) Add details (b) Edit details (c) Delete details (d) View details

The scenario diagram is the same for Maintain Programme details, 'Maintain Scheme details', 'Maintain Paper details', and 'Maintain Student details'.

Maintain School Details	**Maintain Programme Details**
Alternative Flow 1: Invalid Entry	**Alternative Flow 1:** Invalid Entry
Alternative Flow 2: School already exists	**Alternative Flow 2:** Programme already exists
Alternative Flow 3: School not found	**Alternative Flow 3:** Programme not found
Alternative Flow 4: Edit cancelled	**Alternative Flow 4:** Edit cancelled
Alternative Flow 5: Delete cancelled	**Alternative Flow 5:** Delete cancelled
Alternative Flow 6: Deletion not allowed	**Alternative Flow 6:** Deletion not allowed
Alternative Flow 7: User exits	**Alternative Flow 7:** User exits
Maintain Scheme Details	**Maintain Paper Details**
Alternative Flow 1: Invalid Entry	**Alternative Flow 1:** Invalid Entry
Alternative Flow 2: Scheme already exists	**Alternative Flow 2:** Paper already exists

(*Contd.*)

(*Contd.*)

Maintain Scheme Details	Maintain Paper Details
Alternative Flow 3: Scheme not found	**Alternative Flow 3:** Paper not found
Alternative Flow 4: Edit cancelled	**Alternative Flow 4:** Edit cancelled
Alternative Flow 5: Delete cancelled	**Alternative Flow 5:** Delete cancelled
Alternative Flow 6: Deletion not allowed	**Alternative Flow 6:** Deletion not allowed
Alternative Flow 7: User exits	**Alternative Flow 7:** User exits

Maintain Student Details
Alternative Flow 1: Invalid Entry
Alternative Flow 2: Roll number already exists
Alternative Flow 3: Student not found
Alternative Flow 4: Edit cancelled
Alternative Flow 5: Delete cancelled
Alternative Flow 6: Deletion not allowed
Alternative Flow 7: User exits

Table 6.8. Scenario matrix for the 'maintain school details' use case

Scenario 1- Add a school	Basic Flow 1	
Scenario 2- Add a school alternative flow: Invalid Entry	Basic Flow 1	Alternative Flow 1
Scenario 3- Add a school alternative flow: School code already exists	Basic Flow 1	Alternative Flow 2
Scenario 4- Add a school alternative flow: User exits	Basic Flow 1	Alternative Flow 7
Scenario 5- Edit a school	Basic Flow 2	
Scenario 6- Edit a school alternative flow: Invalid Entry	Basic Flow 2	Alternative Flow 1
Scenario 7- Edit a school alternative flow: School not found	Basic Flow 2	Alternative Flow 3
Scenario 8- Edit a school alternative flow: Edit cancelled	Basic Flow 2	Alternative Flow 4
Scenario 9- Edit a school alternative flow: User exits	Basic Flow 2	Alternative Flow 7
Scenario 10- Delete a school	Basic Flow 3	
Scenario 11- Delete a school alternative flow: School not found	Basic Flow 3	Alternative Flow 3
Scenario 12- Delete a school alternative flow: Delete cancelled	Basic Flow 3	Alternative Flow 5
Scenario 13- Delete a school alternative flow: Delete not allowed	Basic Flow 3	Alternative Flow 6
Scenario 14- Delete a school alternative flow: User exits	Basic Flow 3	Alternative Flow 7
Scenario 15- View a school	Basic Flow 4	
Scenario 16- View a school alternative flow: School not found	Basic Flow 4	Alternative Flow 3
Scenario 17- View a school alternative flow: User exits	Basic Flow 4	Alternative Flow 7

As shown in Table 6.8, there are 17 scenarios for 'Maintain School Details' use case. For 'Maintain School Details' use case, we identify four input variables for various basic flows in the use case. There are two input variables (school code, school name) and two selection variables (edit confirmed, delete confirmed) in this use case. These inputs will be available for the respective flows as specified in the use case.

Table 6.9. Test case matrix for the ‘maintain school details’ use case

Test case Id	Scenario and description	Input 1 School code	Input 2 School name	Edit confirmed	Deletion confirmed	Expected result	Remarks (if any)
TC1	**Scenario 1- Add a school**	Valid input	Valid input	n/a	n/a	School is added successfully	--
TC2	**Scenario 2- Add a school alternative flow: Invalid entry**	Invalid input	Valid/invalid input	n/a	n/a	Invalid school code	School code is not in the specified format. School name becomes do not care.
TC3	**Scenario 2- Add a school alternative flow: Invalid entry**	Valid input	Invalid input	n/a	n/a	Invalid school name	School name is not in the specified format
TC4	**Scenario 3- Add a school alternative flow: School code already exists**	Valid input	Valid input	n/a	n/a	School code already exist	The school with the same code is already present in the database
TC5	**Scenario 4- Add a school alternative flow: User exits**	Valid / Invalid input	Valid /Invalid input	n/a	n/a	User is allowed to exit and returns to Main menu	--
TC6	**Scenario 5- Edit a school**	Valid input	Valid input	Yes	n/a	School is updated successfully	--
TC7	**Scenario 6- Edit a school alternative flow: Invalid entry**	Invalid input	Valid/invalid input	n/a	n/a	Invalid school code	School code is not in the specified format
TC8	**Scenario 7- Edit a school alternative flow: School not found**	Valid input	n/a	n/a	n/a	School not found	School with the specified code does not exist in the database
TC9	**Scenario 8- Edit cancelled**	Valid input	Valid input	No	n/a	Main screen of school appears	--
TC10	**Scenario 9- Edit a school alternative flow: User exits**	Valid / Invalid input	Valid /Invalid input	n/a	n/a	User is allowed to exit and returns to Main menu	--

(*Contd.*)

(*Contd.*)

Test case Id	Scenario and description	Input 1 School code	Input 2 School name	Edit confirmed	Deletion confirmed	Expected result	Remarks (if any)
TC11	**Scenario 10- Delete a school**	Valid input	n/a	n/a	Yes	School is deleted successfully	--
TC12	**Scenario 11- Delete a school alternative flow: School not found**	Valid input	n/a	n/a	n/a	School not found	School with the specified code does not exist in the database
TC13	**Scenario 12- Delete a school alternative flow: Delete cancelled**	Valid input	n/a	n/a	No	Main screen of school appears	User does not confirm the delete operation
TC14	**Scenario 13- Delete a school alternative flow: Deletion not allowed**	Valid input	n/a	n/a	n/a	Deletion not allowed	Programme of the school exists
TC15	**Scenario 14- Delete a school alternative flow: User exits**	Valid / Invalid input	Valid /Invalid input	n/a	n/a	User is allowed to exit and returns to Main menu	--
TC16	**Scenario 15- View a school**	Valid input	n/a	n/a	n/a	School is displayed successfully	The school name with the specified code is displayed on the screen
TC17	**Scenario 16- View a school alternative flow: School not found**	Valid input	n/a	n/a	n/a	School not found	The school with the specified code does not exist in the database
TC18	**Scenario 17- View a school alternative flow: User exits**	Valid / Invalid input	Valid /Invalid input	n/a	n/a	User is allowed to exit and returns to Main menu	--

n/a: option(s) not available for respective scenario

There are 18 test cases created for the given 17 scenarios as shown in Table 6.9. Two test cases are designed for scenario 2. After constructing these test cases, actual input values are given to all the variables in order to generate actual output and verify whether the test case passes or fails (refer Table 6.10).

Table 6.10. Test case matrix with actual data values for the 'maintain school details' use case

Test case Id	Scenario and description	School ID	School Name	Edit confirmed	Deletion confirmed	Expected result	Remarks (if any)
TC1	**Scenario 1- Add a school**	101	University School of Information technology	n/a	n/a	School is added successfully	--
TC2	**Scenario 2- Add a school alternative flow: Invalid entry**	1001	*	n/a	n/a	Invalid school code	School code is not of specified length
TC3	**Scenario 2- Add a school alternative flow : Invalid entry**	101	12univ	n/a	n/a	Invalid school name	School name is not in the specified format i.e. it contains digits in the beginning
TC4	**Scenario 3- Add a school alternative flow: School code already exists**	102	University School of Management Studies	n/a	n/a	School code already exists	Entry with the same school code already exists in the database
TC5	**Scenario 4- Add a school alternative flow: User exits**	*	*	n/a	n/a	User is allowed to exit and returns to Main menu	--
TC6	**Scenario 5- Edit a school**	102	University School of Management Studies	Yes	n/a	School is updated successfully	--
TC7	**Scenario 6- Edit a school alternative flow: Invalid entry**	101	univ	n/a	n/a	Invalid school name	School name is not in the specified format which is less than 10 characters
TC8	**Scenario 7- Edit a school alternative flow: School not found**	103	n/a	n/a	n/a	School not found	School code does not exist in the database
TC9	**Scenario 8- Edit cancelled**	101	University School of Information technology	No	n/a	Main screen of school appears	User does not confirm the edit operation

(*Contd.*)

(*Contd.*)

Test case Id	Scenario and description	School ID	School Name	Edit confirmed	Deletion confirmed	Expected result	Remarks (if any)
TC10	**Scenario 9- Edit a school alternative flow: User exits**	*	*	n/a	n/a	User is allowed to exit and returns to Main menu	--
TC11	**Scenario 10- Delete a school**	101	n/a	n/a	Yes	School is deleted successfully	--
TC12	**Scenario 11- Delete a school alternative flow: School not found**	103	n/a	n/a	n/a	School not found	School code does not exist in the database
TC13	**Scenario 12- Delete a school alternative flow: Delete cancelled**	102	n/a	n/a	No	Main screen of school appears	--
TC14	**Scenario 13- Delete a school alternative flow: Deletion not allowed**	102	n/a	n/a	n/a	Deletion not allowed	Programme of the school exists
TC15	**Scenario 14- Delete a school alternative flow: User exits**	*	*	n/a	n/a	User is allowed to exit and returns to Main menu	--
TC16	**Scenario 15- View a school**	101	n/a	n/a	n/a	School is displayed successfully	--
TC17	**Scenario 16- View a school alternative flow: School not found**	103	n/a	n/a	n/a	School not found	School code does not exist in the database
TC18	**Scenario 17- View a school alternative flow: User exits**	*	*	n/a	n/a	User is allowed to exit and returns to Main menu	--

*: 'do not care' conditions (valid/invalid inputs)
n/a: option(s) that are not available for respective scenario

The use case description of 'Maintain programme details' use case is given below:

1. **Introduction**
   Allow the administrator to maintain details of the programme in the school. This includes adding, updating, deleting and viewing programme information.
2. **Actors**
   Administrator
3. **Pre-Conditions**
   The administrator must be logged onto the system and school details for which the programme details are to be added/updated/deleted/viewed must be available in the system before this use case begins.
4. **Post-Conditions**
   If the use case is successful, the programme information is added/updated/deleted/viewed from the system. Otherwise, the system state is unchanged.
5. **Basic Flow**
   This use case starts when the administrator wishes to add/edit/delete/view programme information
   (i) The system requests that the administrator specify the function he/she would like to perform (either 'Add a programme', 'Edit a programme', 'Delete a programme' or 'View a programme')
   (ii) Once the administrator provides the requested information, one of the flows is executed.
   - If the administrator selects 'Add a Programme', the **Add a Programme** flow is executed.
   - If the administrator selects 'Edit a Programme', the **Edit a Programme** flow is executed.
   - If the administrator selects 'Delete a Programme', the **Delete a Programme** flow is executed.
   - If the administrator selects 'View a Programme', the **View a Programme** flow is executed.

   Basic Flow 1: Add a Programme
   The system requests that the administrator enters the programme information. This includes:
   (i) The system requests the administrator to select an already existing school and also enter:
   1. Programme name
   2. Duration (select through drop down menu)
   3. Number of semesters
   4. Programme code

   (ii) Once the administrator provides the requested information, the programme is added to the system.

   Basic Flow 2: Edit a Programme
   (i) The system requests that the administrator enters the programme code.
   (ii) The administrator enters the programme code. The system retrieves and displays the programme information.
   (iii) The administrator makes the desired changes to the programme information. This includes any of the information specified in the **Add a Programme** flow.
   (iv) The system prompts the administrator to confirm the updation of the programme.
   (v) After confirming the changes, the system updates the programme record with the updated information.

   Basic Flow 3: Delete a Programme
   (i) The system requests that the administrator specify the programme code.
   (ii) The administrator enters the programme code. The system retrieves and displays the programme information.
   (iii) The system prompts the administrator to confirm the deletion of the programme. The system deletes the programme record.
   The administrator confirms the deletion.

   Basic Flow 4: View a Programme
   (i) The system requests that the administrator specify the programme code.
   (ii) The system retrieves and displays the programme information.

(*Contd.*)

(*Contd.*)

6. **Alternative Flows**

Alternative Flow 1: Invalid Entry

If in the 'Add a Programme' or 'Edit a Programme' flows, the actor enters invalid programme/duration/number of semesters/programme code or the actor leaves the programme/duration/number of semesters/programme code empty, the system displays an error message. The actor returns to the basic flow and may re-enter the invalid entry.

Alternative Flow 2: Programme code already exists

If in the 'Add a Programme' flow, a programme with a specified programme code already exists, the system displays an error message. The administrator returns to the basic flow and may renter the programme code.

Alternative Flow 3: Programme not found

If in the 'Edit a Programme' or 'Delete a Programme' or 'View a Programme' flows, a programme with the specified programme code does not exist, the system displays an error message. The administrator returns to the basic flow and may re-enter the programme code.

Alternative Flow 4: Edit cancelled

If in the 'Edit a Programme' flow, the administrator decides not to edit the programme, the edit is cancelled and the Basic Flow is re-started at the beginning.

Alternative Flow 5: Delete cancelled

If in the 'Delete a Programme' flow, the administrator decides not to delete the programme, the delete is cancelled and the Basic Flow is re-started at the beginning.

Alternative Flow 6: Deletion not allowed

If in the 'Delete a Programme' flow, a scheme detail of the programme code exists then the system displays an error message. The administrator returns to the basic flow.

Alternative Flow 7: User exits

This allows the user to exit during the use case. The use case ends.

7. **Special Requirements**

None.

8. **Associated use cases**

Login, Maintain School Details, Maintain Scheme Details.

The Use Case Scenario of 'Maintain programme details' use case is given in Figure 6.5 and the scenario matrix is given in Table 6.11.

Table 6.11. Scenario matrix for the 'maintain programme details' use case

Scenario 1- Add a programme	Basic Flow 1
Scenario 2- Add a programme alternative flow: Invalid entry	Basic Flow 1 Alternative Flow 1
Scenario 3- Add a programme alternative flow: Programme code already exists	Basic Flow 1 Alternative Flow 2
Scenario 4- Add a programme alternative flow: User exits	Basic Flow 1 Alternative Flow 7
Scenario 5- Edit a programme alternative flow: Edit a programme	Basic Flow 2
Scenario 6- Edit a programme alternative flow: Invalid entry	Basic Flow 2 Alternative Flow 1
Scenario 7- Edit a programme alternative flow: Programme not found	Basic Flow 2 Alternative Flow 3
Scenario 8- Edit a programme alternative flow: Edit cancelled	Basic Flow 2 Alternative Flow 4
Scenario 9- Edit a programme alternative flow: User exits	Basic Flow 2 Alternative Flow 7
Scenario 10- Delete a programme	Basic Flow 3
Scenario 11- Delete a programme alternative flow: Programme not found	Basic Flow 3 Alternative Flow 3
Scenario 12- Delete a programme alternative flow: Deletion cancelled	Basic Flow 3 Alternative Flow 5
Scenario 13- Delete a programme alternative flow: Deletion not allowed	Basic Flow 3 Alternative Flow 6
Scenario 14- Delete a programme alternative flow: User exits	Basic Flow 3 Alternative Flow 7
Scenario 15-View a programme	Basic Flow 4
Scenario 16- View a programme alternative flow: Programme not found	Basic Flow 4 Alternative Flow 3
Scenario 17- View a programme alternative flow: User exits	Basic Flow 4 Alternative Flow 7

From the use case, we identify seven input variables, out of which four are selection variables. The input variables are programme name, duration, number of semesters and programme code. The selection variables include school name, duration, edit confirmed and delete confirmed. The test case matrix is given in Table 6.12 and the corresponding matrix with actual data values is given in Table 6.13.

Table 6.12. Test case matrix for the 'maintain programme details' use case

Test case Id	Scenario Name and description	Input 1 School selected	Input 2 Programme name	Input 3 Duration	Input 4 Number of semesters	Input 5 Programme code	Edit confirmed	Deletion confirmed	Expected output	Remarks (if any)
TC1	Scenario 1- Add a programme	Yes	Valid input	Valid input	Valid input	Valid input	n/a	n/a	User is allowed to add a programme	--
TC2	Scenario 2- Add a programme alternative flow: Invalid entry	No	Valid/ Invalid input	Valid/ Invalid input	Valid/ Invalid input	Valid/ Invalid input	n/a	n/a	School not selected	User did not select a school
TC3	—do—	Yes	Invalid input	Valid/ Invalid input	Valid/ Invalid input	Valid/ Invalid input	n/a	n/a	Programme name invalid	Programme name is not in the specified format
TC4	—do—	Yes	Valid input	Invalid input	Valid/ Invalid input	Valid/ Invalid input	n/a	n/a	Duration invalid	Duration is not selected
TC5	—do—	Yes	Valid input	Valid input	Invalid input	Valid/ Invalid input	n/a	n/a	Number of semesters invalid	--
TC6	—do—	Yes	Valid input	Valid input	Valid input	Invalid input	n/a	n/a	Programme code invalid	Programme code is not in the specified format
TC7	Scenario 3- Add a programme alternative flow: Programme code already exist	Yes	Valid input	Valid input	Valid input	Valid input	n/a	n/a	Programme code already exists	Entry with the same programme code already exists in the database

(*Contd.*)

(*Contd.*)

Test case Id	Scenario Name and description	Input 1 School selected	Input 2 Programme name	Input 3 Duration	Input 4 Number of semesters	Input 5 Programme code	Edit confirmed	Deletion confirmed	Expected output	Remarks (if any)
TC8	Scenario 4- Add a programme alternative flow: User exits	Yes	Valid / Invalid input	Valid / Invalid input	Valid / Invalid input	Valid / Invalid input	n/a	n/a	User is allowed to exit and returns to Main menu	--
TC9	Scenario 5- Edit a programme alternative flow	n/a	Valid input	Valid input	Valid input	Valid input	Yes	n/a	Programme is suc-cessfully updated	--
TC10	Scenario 6-Edit a programme alternative flow: Invalid entry	n/a	Invalid input	Valid/ Invalid input	Valid/ Invalid input	Valid/ Invalid input	n/a	n/a	Programme name invalid	Programme name is not in the speci-fied format
TC11	—do—	n/a	Valid input	Invalid input	Valid/ Invalid input	Valid/ Invalid input	n/a	n/a	Duration invalid	Duration is not selected
TC12	—do—	n/a	Valid input	Valid input	Invalid input	Valid/ Invalid input	n/a	n/a	Number of semesters invalid	--
TC13	Scenario 7- Edit a programme alternative flow: Programme not found	n/a	n/a	n/a	n/a	Valid input	n/a	n/a	Programme not found	Programme with the specified programme code does not exist in the database
TC14	Scenario 8- Edit a programme alternative flow: Edit cancelled	n/a	Valid input	Valid input	Valid input	Valid input	No	n/a	Sub menu of pro-gramme appears	User does not confirm the edit operation

(*Contd.*)

(*Contd.*)

Test case Id	Scenario Name and description	Input 1 School selected	Input 2 Programme name	Input 3 Duration	Input 4 Number of semesters	Input 5 Programme code	Edit confirmed	Deletion confirmed	Expected output	Remarks (if any)
TC15	Scenario 9- Edit a programme alternative flow: User exits	n/a	Valid / Invalid input	Valid / Invalid input	Valid / Invalid input	Valid / Invalid input	n/a	n/a	User is allowed to exit and returns to Main menu	--
TC16	Scenario 10- Delete a programme	n/a	n/a	n/a	n/a	Valid input	n/a	Yes	Programme is successfully deleted	--
TC17	Scenario 11- Delete a programme alternative flow: Programme not found	n/a	n/a	n/a	n/a	Valid input	n/a	n/a	Programme not found	Programme with the specified programme code does not exist in the database
TC18	Scenario 12- Delete a programme alternative flow: Deletion cancelled	n/a	n/a	n/a	n/a	Valid input	n/a	No	Deletion cancelled	User does not confirm the deletion operation
TC19	Scenario 13- Delete a programme alternative flow: Deletion not allowed	n/a	n/a	n/a	n/a	Valid input	n/a	n/a	Deletion not allowed	Scheme of the programme exists
TC20	Scenario 14- Delete a programme alternative flow: User exits	n/a	n/a	n/a	n/a	Valid / Invalid input	n/a	n/a	User is allowed to exit and returns to Main menu	--

(*Contd.*)

(*Contd.*)

Test case Id	Scenario Name and description	Input 1 School selected	Input 2 Programme name	Input 3 Duration	Input 4 Number of semesters	Input 5 Programme code	Edit confirmed	Deletion confirmed	Expected output	Remarks (if any)
TC21	Scenario 15-View a programme	n/a	n/a	n/a	n/a	Valid input	n/a	n/a	Programme details are displayed	--
TC22	Scenario 16- View a programme alternative flow: Programme not found	n/a	n/a	n/a	n/a	Valid input	n/a	n/a	Programme not found	Programme with the specified programme code does not exist in the database
TC23	Scenario 17- View a programme alternative flow: User exits	n/a	n/a	n/a	n/a	Valid / Invalid input	n/a	n/a	User is allowed to exit and returns to Main menu	--

n/a: option(s) not available for respective scenario

Table 6.13. Test case matrix with actual data values for the programme use case

Test case Id	Scenario name and description	School selected	Programme name	Duration	Number of semesters	Programme code	Edit confirmed	Deletion confirmed	Expected output	Remarks (if any)
TC1	Scenario 1- Add a programme	Yes	MCA	3	6	12	n/a	n/a	User is allowed to add a programme	--
TC2	Scenario 2- Add a programme alternative flow: Invalid entry	No	*	*	*	*	n/a	n/a	School not selected	User did not select a school
TC3	–do—	Yes	M12Ca	*	*	*	n/a	n/a	Programme name invalid	Programme name is not in the specified format i.e. it contains digits
TC4	–do—	Yes	MCA		*	*	n/a	n/a	Duration invalid	Duration not selected
TC5	–do—	Yes	MCA	3	12	*	n/a	n/a	Number of semesters invalid	Number of semesters should be 6 as duration of the programme is 3
TC6	–do—	Yes	MCA	3	6	12d	n/a	n/a	Programme code invalid	Programme code is a numeric field and cannot contain alphabets
TC7	Scenario 3- Add a programme alternative flow: Programme code already exists	Yes	BTech(IT)	4	8	13	n/a	n/a	Programme code already exists	Programme with the same programme code exists in the database
TC8	Scenario 4- Add a programme alternative flow: User exits	Yes	*	*	*	*	n/a	n/a	User is allowed to exit and returns to Main menu	--

(*Contd.*)

(*Contd.*)

Test case Id	Scenario name and description	School selected	Programme name	Duration	Number of semesters	Programme code	Edit confirmed	Deletion confirmed	Expected output	Remarks (if any)
TC9	Scenario 5- Edit a programme alternative flow	n/a	BTech(IT)	4	8	13	Yes	n/a	Programme is successfully edited	--
TC10	Scenario 6-Edit a programme alternative flow: Invalid entry	n/a	Mca123	*	*	*	n/a	n/a	Programme name invalid in 'Edit a programme' flow	Programme name is not in the specified format and contains digits
TC11	—do—	n/a	BTech(IT)		*	*	n/a	n/a	Duration invalid	Duration not selected
TC12	—do—	n/a	BTech(IT)	4	13	*	n/a	n/a	Number of semesters	Number of semesters should be 8 as duration is 4
TC13	Scenario 7- Edit a programme alternative flow: Programme not found	n/a	n/a	n/a	n/a	14	n/a	n/a	Programme not found	Programme with the specified programme code does not exist in the database
TC14	Scenario 8- Edit a programme alternative flow: Edit cancelled	n/a	BTech(IT)	4	8	13	No	n/a	Blank form appears	User does not confirm the edit operation
TC15	Scenario 9- Edit a programme alternative flow: User exits	n/a	*	*	*	*	n/a	n/a	User is allowed to exit and returns to Main menu	--
TC16	Scenario 10- Delete a programme	n/a	n/a	n/a	n/a	12	n/a	Yes	Programme is successfully deleted	--
TC17	Scenario 11- Delete a programme alternative flow: Programme not found	n/a	n/a	n/a	n/a	16	n/a	n/a	Programme not found	Programme with the specified programme code does not exist in the database

(*Contd.*)

(*Contd.*)

Test case Id	Scenario name and description	School selected	Programme name	Duration	Number of semesters	Programme code	Edit confirmed	Deletion confirmed	Expected output	Remarks (if any)
TC18	Scenario 12- Delete a programme alternative flow: Deletion cancelled	n/a	n/a	n/a	n/a	13	n/a	No	Sub menu of programme appears	User does not confirm the delete operation
TC19	Scenario 13- Delete a programme alternative flow: Deletion not allowed	n/a	n/a	n/a	n/a	13	n/a	n/a	Deletion not allowed	Scheme of the record already exists, hence the programme with the specified code cannot be deleted
TC20	Scenario 14- Delete a programme alternative flow: User exits	n/a	n/a	n/a	n/a	*	n/a	n/a	User is allowed to exit and returns to Main menu	--
TC21	Scenario 15-View a programme	n/a	n/a	n/a	n/a	13	n/a	n/a	Programme details are displayed	--
TC22	Scenario 16- View a programme alternative flow: Programme not found	n/a	n/a	n/a	n/a	16	n/a	n/a	Programme not found	Programme with the specified programme code does not exist in the database
TC23	Scenario 17- View a programme alternative flow: User exits	n/a	n/a	n/a	n/a	*	n/a	n/a	User is allowed to exit and returns to Main menu	--

*: 'do not care' conditions (valid/invalid inputs)
n/a: option(s) not available for respective scenario

The test cases for other use cases of the URS case study are given in Appendix II.

GUIDELINES FOR GENERATING VALIDITY CHECKS

We want to have guidelines for generating validity checks for input data. We may have to give many inputs to a program via forms, data files and / or input statement(s). Ideally, we want to enter correct data and for this purpose we should test various conditions with invalid inputs to the program. Some of the guidelines are given in the following sub-sections.

- ### Data Type

If input x is defined as an integer, then x should also be checked for float, char, double, etc. values. We should clearly state what can be accepted as an input. In the login form, (please refer to Figure 6.7), we should clearly state the type of both the inputs i.e. Login Id and password. For example, the Login Id input should be numeric and should not accept alphabets, special characters and blank spaces. Similarly, the password input will accept alphabets, digits, hyphen and underscore but will not accept blank spaces. We should generate validity checks for every 'do' and every 'do not' case.

- ### Data Range

The range of inputs should also be clearly specified. If x is defined as an integer, its range, (say $1 \leq x \leq 100$) should also be defined. Validity checks may be written for conditions when $x \leq 1$ and $x > 100$. For example, in login form, length of the login-id is defined as 11 digits and the password as 4 to 15 digits. We should generate validity checks for both valid and invalid range of inputs.

- ### Special Data Conditions

Some special conditions may need to be checked for specified inputs. For example, in the e-mail address, '@' and '.'symbols are essential and must be checked. We should write validity checks for such special symbols which are essential for any specific input.

- ### Mandatory Data Inputs

Some inputs are compulsory for the execution of a program. These mandatory fields should be identified and validity checks be written accordingly. In the login form, both inputs (login Id and password) are mandatory. Some fields (data inputs) may not be mandatory like telephone number in a student registration form. We should provide validity checks to verify that mandatory fields are entered by the user.

- ### Domain Specific Checks

Some validity checks should be written on the basis of the expected functionality. In the URS, no two semesters should have a common paper. The roll number should be used as a login Id. A student cannot select more than the required number of elective papers in a semester. These domain specific issues should be written as validity checks in order to verify their correctness.

STRATEGIES FOR DATA VALIDITY

What are data validity checks? Are they required to be tested? Why should data validation be given focus in testing? Why do we expect valid data? These questions are important and their answers may motivate us to generate test cases using data validity checks.

Valid data means correct data which is expected in every software. The software should provide checks for validating data entered into the system. Whenever and wherever we attempt to enter invalid data, an appropriate message should be displayed. Ideally, the software should only allow the entry of valid data into the system. If we are able to do so with a good design, we may be able to minimize many problems. In order to give proper focus on data validations, there is a provision of writing data validity checks for every form / screen in the SRS document. These data validity checks may become the basis for the generation of test cases.

Data validity strategies are often influenced by the design of the software. Three popular strategies for data validation are discussed which may be applied at the early phases of the software development life cycle.

- ### Accept Only Known Valid Data

We all want to enter valid data into the system. If our software accepts only correct data, our design is a successful design. If it does not happen, we may enter invalid data into the system, which may further complicate many issues. Invalid data may force the software to behave unexpectedly and may lead to a failure. Hence, software should accept only input(s) that is / are known to be safe and expected.

Consider the SRS document of the URS (refer Appendix I). The login form is given in Figure 6.6 that allows users to enter into the system using a valid login Id and a valid password. Some data validity checks are also given in Table 6.14. Our login form should only accept valid login Id and valid password and allow the user to enter into the system. If we give valid entries for both login ID and password, we should enter into the system, otherwise proper error message(s) should be displayed. In order to ensure validity of data, we should generate test cases to check the validity of the data and to also check the conditions when we enter invalid data. Both valid and invalid data inputs will generate test cases that may check the entry of data into the software. We have identified 8 validity checks shown in Table 6.14 and may generate test cases as given in Table 6.15. If the first input is invalid, the second input automatically becomes ‘do not care’ and an appropriate error message is displayed. Every validity check condition at least generates a test case. In Table 6.15, two test cases (TC1, TC10) accept only valid data. We have identified three test cases for VC4 and two test cases for VC6.

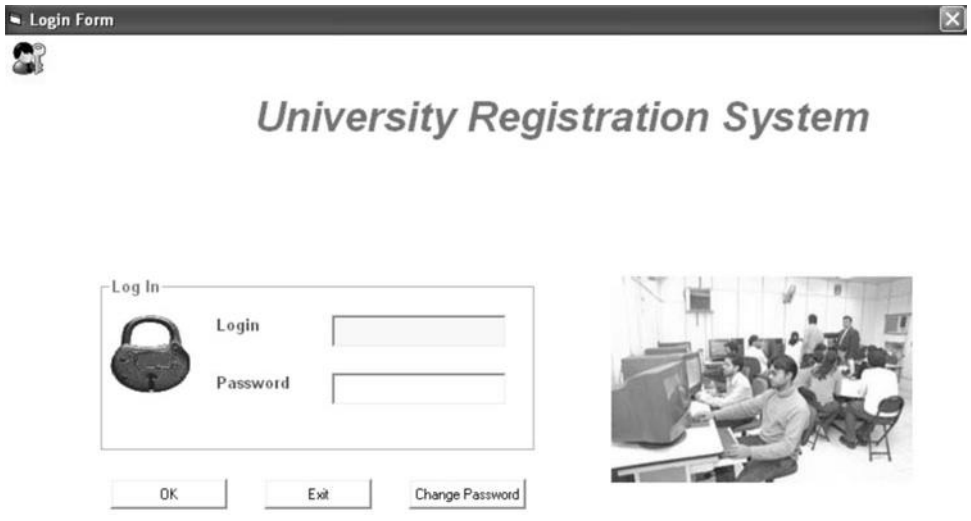


Figure 6.6. Login form

Table 6.14. Validity checks for login form

Validity check Number	Description
VC1	Every user will have a unique login Id.
VC2	Login Id cannot be blank.
VC3	Login Id can only have 11 digits.
VC4	Login Id will not accept alphabetic, special and blank spaces.
VC5	Password cannot be blank.
VC6	Length of password can only be 4 to 15 digits.
VC7	Alphabets, digits, hyphen and underscore characters are allowed in password field.
VC8	Password will not accept blank spaces.

Table 6.15. Test case with actual data values for the login form

Test case Id	Validity check Number	Login id	Password	Expected output	Remarks
TC1	VC1	10234567899	Rkhj7689	User successfully logs into the system	-
TC2	VC2		*	Please Enter Login Id	Login id cannot be blank
TC3	VC3	1234	*	Invalid login id	Login id should have 11 digits
TC4	VC4	Ae455678521	*	Invalid login id	Login id cannot have alphanu-meric characters
TC5	VC4	123$4567867	*	Invalid login id	Login id cannot have special characters
TC6	VC4	123 45667897	*	Invalid login id	Login id cannot have blank spaces
TC7	VC5	10234567899		Please Enter Password	Password cannot be blank
TC8	VC6	10234567899	Ruc	Invalid password	Password cannot be less than 4 characters in length
TC9	VC6	10234567899	Rtyuiopki1123678	Invalid password	Password cannot be greater than 15 characters in length
TC10	VC7	10234567899	Rty_uyo	User successfully logs into the system	Password can have underscore character
TC11	VC8	10234567899	Rt yuii	Invalid password	Password cannot have blank spaces

*: 'do not care' conditions (valid/invalid inputs)

Additional validity checks are designed in order to validate various inputs in the 'Change password' form. The 'Change password' form is given in Figure 6.7 and the validity checks are given in Table 6.16. The corresponding test cases for each validity check are given in Table 6.17.

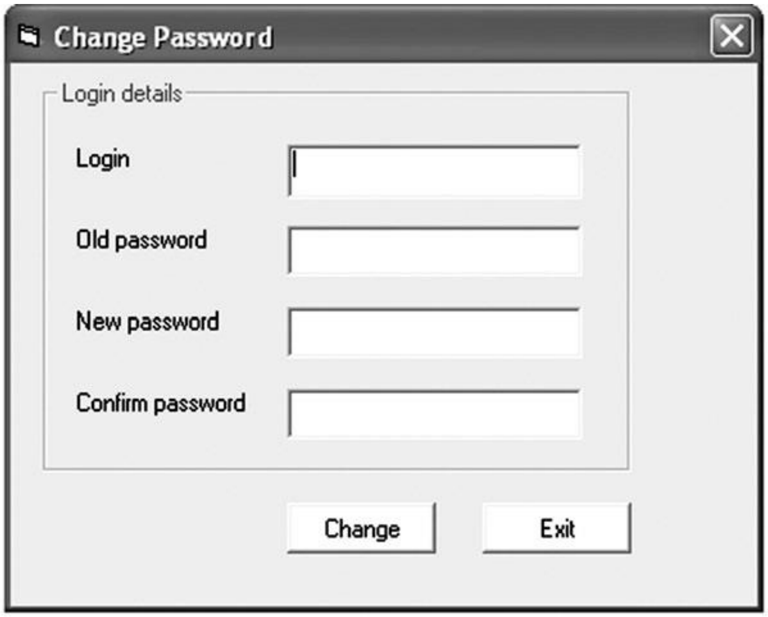


Figure 6.7. Change password form

Table 6.16. Validity checks for change password form

Validity check Number	Description
VC9	Login Id cannot be blank.
VC10	Login Id can only have 11 digits.
VC11	Login Id will not accept alphabetic, special and blank spaces.
VC12	Old password cannot be blank.
VC13	Length of old password can only be 4 to 15 digits.
VC14	Alphabets, digits, hyphen and underscore characters are allowed in old password field.
VC15	Old password will not accept blank spaces.
VC16	New password cannot be blank.
VC17	Length of new password can only be 4 to 15 digits.
VC18	Alphabets, digits, hyphen and underscore characters are allowed in new password field.
VC19	New password will not accept blank spaces.
VC20	'Confirm password' cannot be blank.
VC21	'Confirm password' should match with new password.

Table 6.17. Test case with actual data values for the 'Change Password' form

Test case Id	Validity check No.	Login id	Old password	New Password	Confirm Password	Expected output	Remarks
TC1	VC9		*	*	*	Please Enter Login Id	Login id cannot be blank
TC2	VC10	1234	*	*	*	Invalid login id	Login id should have 11 digits
TC3	VC11	Ae455678521	*	*	*	Invalid login id	Login id cannot have alphanumeric characters
TC4	VC11	123$4567867	*	*	*	Invalid login id	Login id cannot have special characters
TC5	VC11	123 45667897	*	*	*	Invalid login id	Login id cannot have blank spaces
TC6	VC12	10234567899		*	*	Please Enter Old Password	Password cannot be blank
TC7	VC13	10234567899	Ruc	*	*	Invalid old password	Password cannot be less than 4 characters long
TC8	VC14	10234567899	Rtyuiopki1123678	*	*	Invalid old password	Password cannot be greater than 15 characters in length
TC9	VC14	10234567899	Rty_uyo	*	*	--	Password can have underscore character
TC10	VC15	10234567899	Rt yuii	*	*	Invalid old password	Password cannot have blank spaces
TC11	VC16	10234567899	Ruc_ui		*	--	Password cannot be blank
TC12	VC17	10234567899	Ruc_ui	Rrk	*	Invalid new password	Password cannot be less than 4 characters long
TC13	VC17	10234567899	Ruc_ui	Rtyuiopki1123678	*	Invalid new password	Password cannot be greater than 15 characters in length
TC14	VC18	10234567899	Ruc_ui	Rty_uyo	*	Invalid new password	New password can have underscore character
TC15	VC19	10234567899	Ruc_ui	Rty uyo	*	Invalid new password	New password cannot have blank spaces
TC16	VC20	10234567899	Ruc_ui	Rty_uyo		--	'Confirm Password' cannot be blank
TC17	VC21	10234567899	Ruc_ui	Rty_uyo	Rty_uyo	Invalid 'Confirm Password'	'Confirm Password' should match with new password

*: 'do not care' conditions (valid/invalid inputs)

- **Reject Known Bad Data**

We should be able to identify the correctness of the data. If the input data is not as expected, the software should reject it and an appropriate error message should be displayed. We should check the data type from the form itself. If the integer type x is the input and we enter x as a float, an error should immediately be displayed. The software should accept values in the specified range. If the input is beyond range, it should not be accepted at all. Many test cases of Table 6.15 check this concept and reject known bad data (refer TC3, TC4, TC5, TC6, TC8, TC9, and TC11) by giving appropriate error messages.

- **Sanitize All Data**

Data sanitization is the process of purifying (filtering) undesirable data in order to make it harmless and safe for the system. We may sanitize data at the input stage where data is entered by the user. We may also sanitize the data at the output stage where data is displayed to the user in such a way that it becomes more useful and meaningful. For example, when an integer variable is used, its lower and upper permissible limits must be specified and provisions should be made in the program to prevent the entry of any value outside the permissible limit. These limits are hardware dependent and may change, if not earlier specified. In case of Boolean variable, provision should be made in the program to reject any value which is not from the following list:
List = (true, false, 0, 1, yes, no)

Hence, we should attempt to make undesired data harmless, especially when dealing with rejecting bad inputs. This may be easy to write but extremely difficult to do in practice. It is advisable to reject undesired data if we want to play safe and secure.

Example 6.2: Consider the 'Maintain School detail' form of the URS as given in Figure 6.8. The validity checks for the 'Maintain school details' form are given in Table 6.18. Generate test cases from these validity checks.

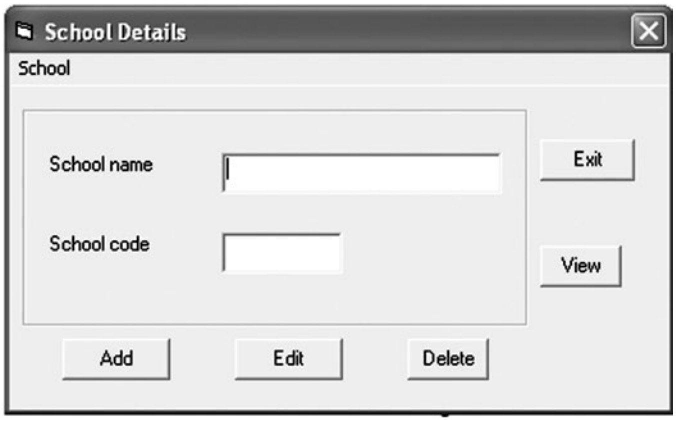


Figure 6.8. Maintain school details form

Table 6.18. Validity checks for school form

Validity check Number	Description
VC1	Only Administrator will be authorized to access the 'Maintain School Details' module. Test case of this validity check cannot be generated as access to the module is provided to the actor at the time of login.
VC2	Every school will have a unique school name.
VC3	School code cannot be blank.
VC4	School code cannot contain alphanumeric, special and blank characters.
VC5	School code will have only 3 digits.
VC6	School name cannot be blank.
VC7	School name will only accept alphabetic characters and blank spaces.
VC8	School name cannot accept special characters and numeric digits.
VC9	School name can have from 10 to 50 characters.

Solution:

Test cases based on validity checks for 'Maintain school details' form are given in Table 6.19.

Table 6.19. Test case with actual data values for the school form

Test case Id	Validity check No.	School ID	School Name	Expected result	Remarks (if any)
TC1	VC2	101	University School of Information Technology	User successfully adds the school record	--
TC2	VC3		*	Please enter school code	School code cannot be blank
TC3	VC4	1rr	*	Invalid school code	School code cannot contain alphanumeric characters
TC3	VC4	1_*	*	Invalid school code	School code cannot contain special characters
TC3	VC4	1 3	*	Invalid school code	School code cannot contain blank characters
TC4	VC5	1012	*	Invalid school code	School code can have length of 3 digits
TC5	VC6	102		Invalid school name	School name cannot be blank
TC6	VC7	102	University School of Management Studies	User successfully adds the school record	--
TC7	VC8	103	University 434	Invalid school name	School name cannot contain digits
TC8	VC8	104	University_school_of_ basic_applied_science	Invalid school name	School name cannot contain special characters
TC9	VC9	105	univer	Invalid school name	School name cannot contain less than 10 characters
TC10	VC9	106	>50	Invalid school name	School name cannot contain more than 50 characters

*: 'do not care' conditions (valid/invalid inputs)

Example 6.3: Consider the ‘Maintain programme detail’ form of the URS as given in Figure 6.9. This form will be accessible only to the system administrator. It will allow him/her to add/edit/delete/view information about new/existing programme(s) for the school that was selected in the ‘Programme Details’ form. Generate the test cases using validity checks given in Table 6.20.

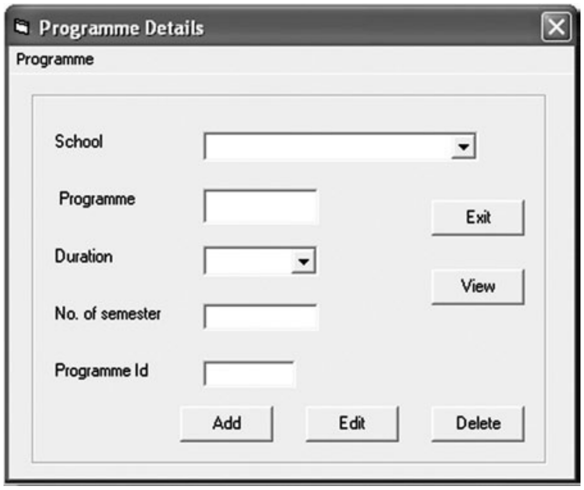


Figure 6.9. Maintain program details form

Table 6.20. Validity checks for program form

Validity check No.	Description
VC1	Only Administrator will be authorized to access the ‘Maintain Programme Details’ module.
VC2	Every programme will have a unique programme code and name.
VC3	School name cannot be blank.
VC4	Programme name cannot be blank.
VC5	Programme name can be of length 3 to 50 characters.
VC6	Programme name can only have alphabets and brackets.
VC7	Programme name cannot have special characters, digits and blank spaces.
VC8	Duration cannot be blank.
VC9	Duration can have a value from 1 to 7.
VC10	Number of semesters cannot be blank.
VC11	Number of semesters can have a value from 2 to 14.
VC12	Programme code cannot be blank.
VC13	Programme code cannot have special characters, digits and blank spaces.
VC14	Programme code can have only 2 digits.

Solution:

The test cases based on validity checks of the ‘Maintain Programme Detail’ form are given in Table 6.21.

Table 6.21. Test case with actual data values for the program form

Test case Id	Validity check No.	School selected	Programme name	Duration	Number of semesters	Programme code	Expected output	Remarks (if any)
TC1	VC2	Yes	MCA	3	6	12	User is allowed to add a programme	--
TC2	VC3		*	*	*	*	Please select school	User should select a school
TC3	VC4	Yes		*	*	*	Please enter programme name	Programme name cannot be blank
TC4	VC5	Yes	MC	*	*	*	Invalid programme name	Programme cannot be less than 3 characters
TC5	VC5	Yes	>50	*	*	*	Invalid programme name	Programme cannot be greater than 50 characters
TC6	VC6	Yes	MCA(SE)	*	*	*	--	Valid programme name
TC7	VC7	Yes	MC_A(Se)	*	*	*	Invalid programme name	Programme cannot contain special characters
TC8	VC7	Yes	MC1234	*	*	*	Invalid programme name	Programme cannot contain digits
TC9	VC8	Yes	MCA		*	*	Please enter duration	Duration cannot be blank
TC10	VC9	Yes	MCA	8	*	*	Invalid duration	Duration can have value between 1 to 7
TC11	VC10	Yes	MCA	3		*	Please enter number of semesters	Number of semesters cannot be blank
TC12	VC11	Yes	MCA	3	15	*	Invalid number of semesters	Number of semesters can have value between 2 to 14
TC13	VC12	Yes	MCA	3	6		Please enter programme code	
TC14	VC13	Yes	MCA	3	6	12a	Invalid programme code	Programme code cannot contain alphanumeric characters
TC15	VC13	Yes	MCA	3	6	1_0	Invalid programme code	Programme code cannot contain special characters
TC16	VC13	Yes	MCA	3	6	1 2	Invalid programme code	Programme code cannot contain blank spaces
TC17	VC14	Yes	MCA	3	6	123	Invalid programme code	Programme code can only contain 2 digits

*: 'do not care' conditions (valid/invalid inputs)

The validity checks for other forms of the URS case study are given in Appendix III.