Test case

I. How to build basic test cases

1. Test case concept

Test case is a set of condition cases that Testers rely on to determine whether an application, software system or one of its functions is working as expected.

Test case development process can help find bugs in the requirements or design of the application, as it requires thinking through the application's operations completely. For this reason, it is very helpful to prepare test cases as early as possible in the software development process.

The test cases should cover the entire functional flow described in the analysis and design documentation; requirements for information security, system performance requirements.

2. Structure of Test case

•Test Case ID : The value needed to determine the number of cases needed for testing.

•Test Items : Based on the function of the system, the functions can be broken down to create clearer TCs.

•Pre-condition : Pre-condition if any

•Test Data: The data to prepare for testing

•Test Steps : Describe the test execution steps

•Expected results: Expected results from the above steps

•Actual result: Usually pass, fail

•Note : This column is used to note down relevant information when executing test cases.

3. **Steps to determine TCs**

   B1: Determine the test purpose: need to understand the customer's requirement specification.

   B2: Defining functional testing: need to know how the software is to be used including different activities and functional organization.

   The implementation steps only describe the execution steps from the end user's side including data entry, button press. Checking the data in the DB against the display on the screen is in the desired result. Commonly used for test cases that test save, update, delete

   B3: Identify non-functional requirements: hardware requirements, operating system, security aspects

   B4: Define form for TCs: including UI interface, functionality, compatibility and performance...

   B5: Determining the influence of modular principles: TCs should be designed to cover the maximum extent of the influence of modules on each other.

4. **Define test cases.**

   •Normal case: Common test cases

   •Abnormal case: Abnormal test cases

   •Boundary case: Boundary test cases.

5. **Effective Test cases Needed**

•Accurate, complete system operations

• Independent (can be done without depending on other test cases, easily divided among many testers)

• The content is simple, has a clear purpose and is understood by all readers in a unique way. (input, output, clear steps) Coherent presentation of the entire document.

•Reusable (can be easily updated and modified).

6. **Notes when writing test cases**

The test case file needs to have simple, transparent and easy-to-understand test steps

Step test must be written in clear details so that even if other testers read it, it can be done

The purpose and scope of test cases are also described in detail.

Precondition, test data must also be recorded in each test case if necessary.

Test cases should be cross-reviewed by team members.

Should not combine too many confirmation results into 1 case, but should separate each confirm result into each case.

When creating test cases should stand at End use position.

Test cases should cover test cases such as: Equivalence classification, boundary values, normal and abnormal conditions.

Neither the free test cases are included in the requirements specification.

II. Apply testing techniques during Test case creation

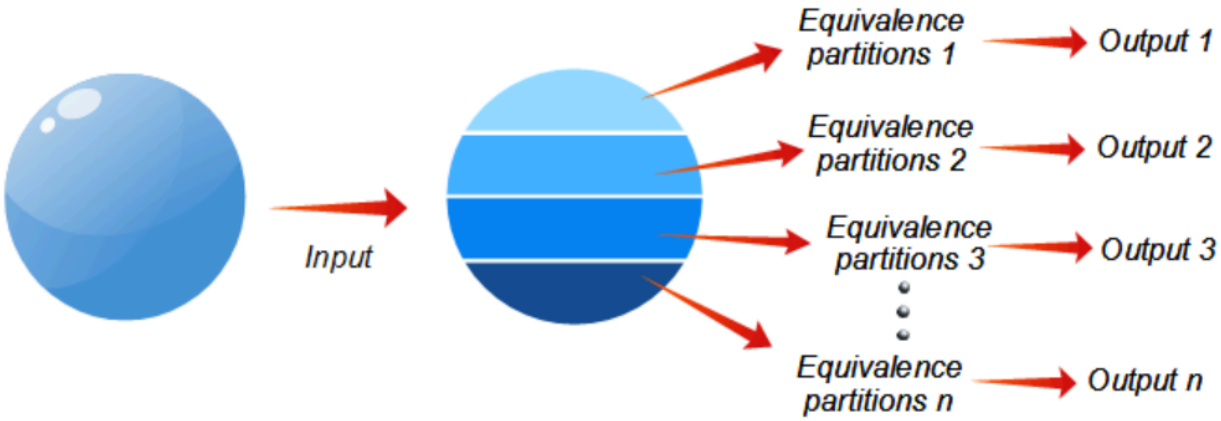
1. Equivalence Classification Method

Equivalence classification has the following characteristics:

A test method that divides the input domain of a program into equivalent classes of data

All values in an equivalence zone will give the same output

It is possible to test a representative value in the equivalence region



Design Test-case by equivalence classification according to the following principles:

Determine the number of valid equivalence classes
Determine the number of equivalence classes that are not valid
Example: Testing the login form includes: User is a text box and Password is a text box

Requirements: Design test cases so that when the user enters the user in the text box, only the number of characters can be entered [6 - 20]
Analyze the test cases as follows:

The equivalence classes are defined as: 6,[6-20] and 20. Thus, the test

cases according to this method are:

+/ Enter the first invalid case: enter 5 characters

+/ Enter a valid case: enter 7 characters

+/ Enter the second invalid case: enter 21 characters

2. **Marginal Value Method**

Marginal price analysis has the following characteristics:

A method that has boundary conditions as input values for test cases

As a special case of equivalence classification, based on the equivalence partitions the tester will determine the boundary value between these partitions and select the appropriate test case.

This method adds test cases to the above equivalence classifier method.

Some rules that can define test cases are:

Smallest value.
The adjacent value is greater than the smallest value
Normal value
The adjacent value is less than the maximum value
The maximum marginal value.
Example: A text box that allows to enter a person's age in the range [0-100].

Designing test cases according to this method is:

+/ Minimum value: 0

+/ The adjacent value is larger than the smallest value: 1

+/ Normal value: 50

+/ The adjacent value is less than the largest value: 99

+/ Maximum boundary value: 100

3. **Guess the error**

The error prediction method has the following characteristics:

As a method based on both intuition and experience, testers can write a list of possible error types or error prone cases and then write test cases based on that list.

This is a method of adding test cases to the above methods based on the tester's experience mainly.
For this method, there is no specific process to apply, mainly depends on each tester's testing experience.

4. **State transition technique (0 - switch)**

The state transition method has the following characteristics:

- An observation method that tracks the transition from one state to another when something is done to the software program.
- Allows testers to view software development under some of its state conditions, transitions between states and input conditions, and state change trigger events
- Add test cases to detect errors that might not be detected by input and output conditions by the above methods.

Example: Test an online buying and selling process as follows:

1. The shopping cart on an online shopping site is started as empty (no items).

2. When you select a product, it will be added to the shopping cart. You can also deselect items in the cart.

3. When you decide to purchase, a screen will appear that summarizes the items in the cart along with information about the price, quantity and total amount of the cart, to let you confirm whether it is correct or not. .

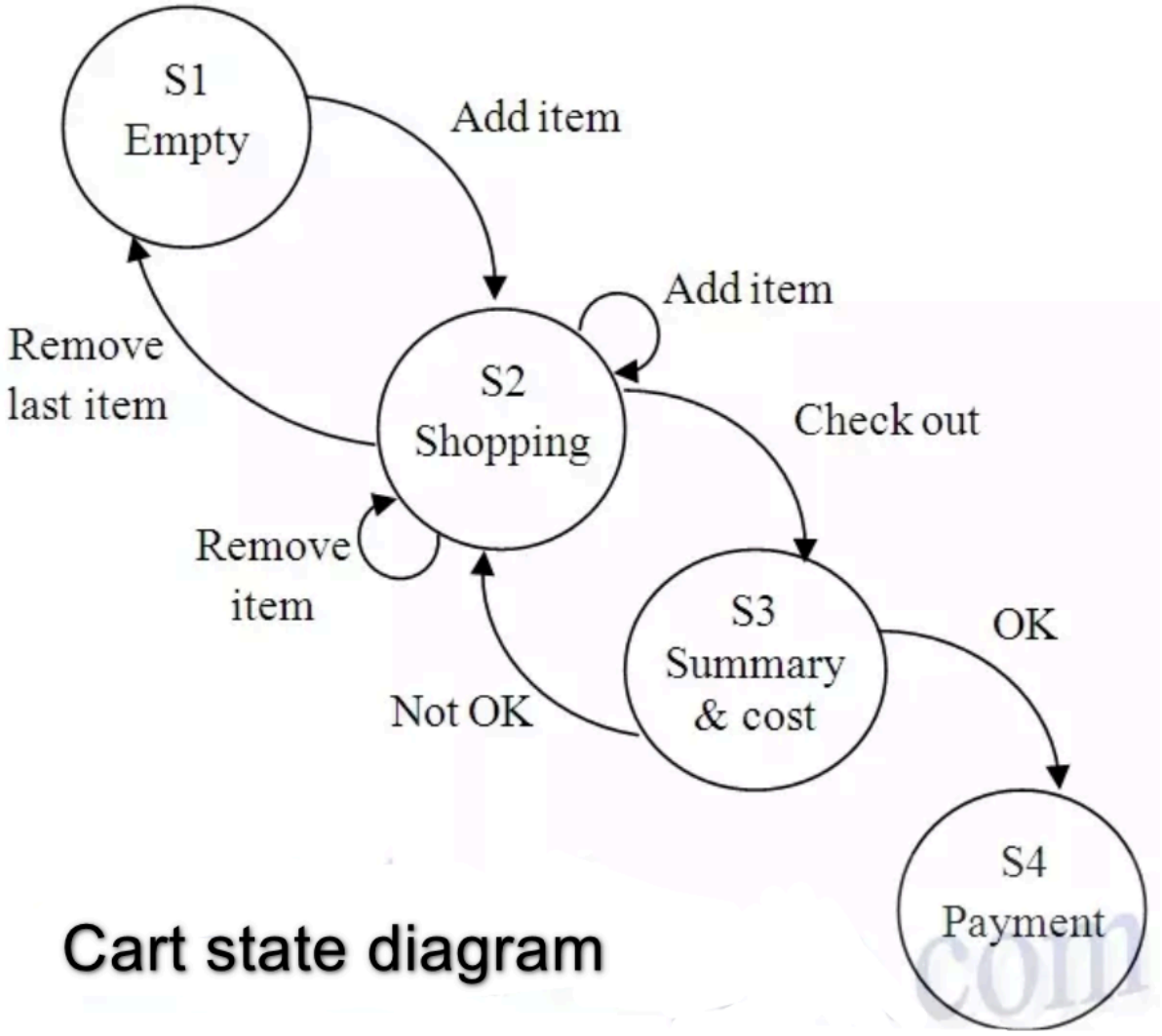
4. If you see the quantity and price OK then you will be redirected to the payment page.

5. Otherwise you will return to the purchase page (now you can deselect the items you want to remove).

Requirements: Write Test case for the above example.

Instructions for applying this state transition technique:

Step 1. Draw the state transition diagram according to the above requirements:



Cart state diagram

Step 2. Make a table describing the state corresponding to the above state diagram:

Status/Event	Add item	Remove item	Remove last item	Check out	Not OK	OK
S1 – empty	S2	-	-	-	-	-
S2 – Shoping	S2	S2	S1	S3	-	-
S3 – Summary	-	-	-	-	S2	S4
S4 – Payment	-	-	-	-	-	-

Note: In the table above, the cell containing the - sign is an invalid state

Step 3. From the above state description table, the test cases can be designed as follows:

Test case ID	Precondition	Condition	Expected Result	Note
TC1	Cart is empty	Click the Add item button to add 1 item	Cart contains the selected item	S1=>S2
TC2	Cart has at least 1 item	Click the Add item button to add 1 item	Cart has more items you just selected	S2=>S2
TC3	Cart has at least 2 item	Click the Remove item button to remove 1 item	Deleted item is no longer in cart	S2=>S2
TC4	There is only 1 item in the cart	Click the Remove item button to remove 1 item	Cart is empty	S2=>S1
TC5	Cart has at least 1 item	Click the Check out button to pay	Display the list of currently selected items in the cart and the total amount to be paid	S2=>S3
TC6	Is on the screen to check goods and money	Click the back button to return to the purchase screen	Return to the purchase screen	S3=>S2

TC7	Is on the screen to check goods and money	Click the Payment button to make a purchase	Payment screen display	S3=>S4
TC8	Cart is empty	Click the Check out button	Display an error message or there is no Check out button	S1=>S3
TC9	Cart is empty	Click the Remove item button	Display an error message or the Remove item button is not displayed or is disabled	S1=>S1
TC10	Cart has at least 1 item	Click the Payment button	Display an error message or there is no Payment button	S2=>S4
TC11	Is on the screen to check goods and money	Click the Add item button	Display an error message or there is no Add item button	S3=>S1
TC12	Is on the screen to check goods and money	Click the Remove item button	Display an error message or there is no Remove item button	S3=>S2
TC13	Is on the payment screen	Click the Add item button	Display an error message or there is no Add item button	S4=>S1
TC14	Is on the payment screen	Click the Remove item button	Display an error message or there is no Remove item button	S4=>S2

LOG BUG

I. Bug ticket template

Most of the Agile projects use some common management system like Redmine or Jira, so here I will use Jira as the standard to implement the requirements in the Bug Log. Here is an ideal template for a newly created bug ticket on Jira:

Summary

- <content>

Expected result

- <content>

Actual result

- <content>

Environment

- Hardware: <content>
- Software: <content>

Precondition

- <content>

Reproduction Steps

- Step 1: <content>
- Step 2: <content>
- Step n: <content>

Recover action

- <content>

Remarks

- <content>

Version

- <content>

II. How to log bugs properly?

1. Make sure the ticket type is selected as 'Bug' or 'Defect':

This requirement is to be able to easily filter the search and extract the list of bugs without mixing with other Task types.

2. Ticket Name must be focused:

Finding a bug when the project has been running for a long time takes a lot of effort and is frustrating. So apply the following methods to make it easier to find a specific bug:

2.1 Ticket Name needs to be short, easy to understand and include keywords:

- The Ticket Name needs to be a complete sentence stating the problem and the conditions for that problem (if identified).
- Ticket Name needs to include important and easy-to-remember keywords. For example, if I wanted to find a bug about the image not being fully enlarged in the slideshow, I would just type 'stretch' or 'zoom' into Jira's Searchbox and the tickets would be filtered out. It's easy, isn't it?

II.2 Attach Tags in Ticket Name:

Distinguishing bugs by Tags is a classic but very effective way. Before logging a bug, pay attention to see if the bug is a [LOGIC] bug or a [GUI] bug, a [FRONTEND] or [BACKEND] bug, a [WEB] or [ANDROID] or [IOS] app bug. Depending on the specifics of the project, please choose the appropriate Tags. Putting Tags at the top of the Ticket name will be very helpful in sorting and searching, and it's also easier for PM to assign Tasks to Developers.

3. Summary should not be omitted:

Although this part does not have much value in practical use by developers, it is very valuable for QA and PM when the number of bugs in the project becomes too much or when more than one QA participates in testing. Summarizing the bug content in an easily visualized way in about 1-3 lines will make browsing, filtering and finding bugs a lot easier, avoiding the case of duplicate bugs or developers not being able to visualize the bug.

4. **Expected Result needs to be reasonable:**

- Expected Result is the expected result of an action or combination of actions performed on the product. This expected result is mainly based on the customer's Requirement. However, when the Requirement is not detailed enough or clear or unreasonable, QA needs to rely on the user's point of view (common senses) to evaluate and determine an acceptable response level (compromise) that the customer does not. opposed and the dev also felt sympathy.
- Expected Result in bug ticket should be written clearly, in detail, concisely. Expected Result of the same set of actions on different screens should be clearly noted to avoid confusion.

5. **Actual Result should be described in detail:**

- Ideally, Actual Result should be rewritten based on the sentence pattern of Expected Result so that readers can easily distinguish them from each other. Example: (Expected) Button should return to normal after uploading is done (Actual) Button does not return to normal after uploading is done
- Actual Result on different monitors should also be separated and annotated separately.

6. **Environment should be fully listed:**

- Hardware: On which systems is the bug reproduced (laptop/mobile)? On what screen size (? x ?) does the bug occur?
- Software: On which operating system has the bug been

reproduced? (Windows/Ubuntu/OSX/Android/iOS)? Does the bug occur in any specific browser (Chrome/Firefox/Safari/IE)?

7. **Precondition is no less important:**

- What screen were you on before the bug happened?
- When the bug occurred, which roll account did you log in with (member/admin/...)?
- What function were you using in the background when the bug occurred (playing video/music/...)?
- The bug only occurs when what conditions are satisfied (another error occurs before/switching screens in a particular sequence/change in hardware or device configuration/...)?

8. Reproduction Steps are detailed steps to reproduce the bug:

- Please ensure that the steps are detailed, accurate, concise, and full of information (may include the type of device causing the error, the account where the error occurred, the name of the test data, ...)
- The implementation steps should cover the entire bug reproducibility process, including the precondition reproduction steps
- Need to confirm the results of each step (OK/NG) along with other symptoms that you consider important (if any).

9. Recover Actions are important clues:

- Recover Actions are actions to restore the normal working state of the product or make the bug disappear temporarily. Investigating Recover Actions is not always necessary, but for complex systems it is an important clue to understanding the root cause and how to fix the bug. Spending time investigating Recover Actions also shows that Testers invest a lot of time and care in bug logging.

10. Remarks show similar error symptoms:

In a professional software development system, the ability to reuse code is required, so that one piece of faulty code can cause bugs in many different places. Remarks is a place for Testers to record screens with similar bug behavior, partial or complete similarity, and inconsistent behavior across different environments. Eg:

- Bug also occurs on Template List, Image Set List and Profile page.
- Bug occurs on all browsers accept Chrome.

11. Version is the product version on which the bug was discovered:

Noting the product version where the bug was discovered will help the manager to determine the evolution of the product.
Along with re-marking the version where the bug is fixed will help simplify tracking if a Degrade bug occurs.

Note: Above is the ideal Template on which a Bug Ticket should be written. Of course, in the pure Agile model with a smaller number of team members, it can be simplified or reduced.