| [Print](javascript:window.print()) | [Close](javascript:parent.window.close()) |

**Course Transcript**

Black-Box Software Testing Techniques

**Equivalence Partitioning and Boundary Values**

| [1. Equivalence Partitioning in Software Testing](http://xlibrary.skillport.com/courseware/Content/cca/sd_sftf_a04_it_enus/output/html/course_transcript.html#t2) |

| [2. Boundary Value Analysis in Software testing](http://xlibrary.skillport.com/courseware/Content/cca/sd_sftf_a04_it_enus/output/html/course_transcript.html#t5) |

| [3. Advanced Partitioning and Boundary Value Analysis](http://xlibrary.skillport.com/courseware/Content/cca/sd_sftf_a04_it_enus/output/html/course_transcript.html#t24) |

| [4. Writing Software Test Cases](http://xlibrary.skillport.com/courseware/Content/cca/sd_sftf_a04_it_enus/output/html/course_transcript.html#t17) |

**Decision Tables, State Transitions, and Use Cases**

| [1. Decision Tables in Software Testing](http://xlibrary.skillport.com/courseware/Content/cca/sd_sftf_a04_it_enus/output/html/course_transcript.html#t21) |

| [2. State Transition Software Testing](http://xlibrary.skillport.com/courseware/Content/cca/sd_sftf_a04_it_enus/output/html/course_transcript.html#t22) |

| [3. Use Cases in Software Testing](http://xlibrary.skillport.com/courseware/Content/cca/sd_sftf_a04_it_enus/output/html/course_transcript.html#t23) |

| [4. Creating Software Test Cases](http://xlibrary.skillport.com/courseware/Content/cca/sd_sftf_a04_it_enus/output/html/course_transcript.html#t29) |

Equivalence Partitioning in Software Testing

Learning Objectives

*After completing this topic, you should be able to*

* *recognize how basic equivalence partitioning works*
* *identify some of the most common black-box software testing techniques*

**1. Equivalence partitioning**

The software testing process is driven by a number of test design techniques. *Black-box* software testing, one of the specification-based testing techniques, is a form of dynamic testing.

Black-box testing derives test cases directly from the specifications of the software being tested and focuses on the behavior of the software and the efficiency of its performance. It does not focus on the structural details. That is, a tester using black-box testing only checks the input and output of the software, not how inputs and outputs are generated by the system.

Note

*Black-box testing can be applied for both the functional and non-functional modes of software testing.*

There are five commonly used black-box test techniques.

**Equivalence partitioning**

Equivalence partitioning is a black-box technique where test conditions for software are divided into sets of logical groups called partitions if they exhibit similar behavior when processed. Each partition covers a specific aspect of the software. This type of testing does not require you to create test cases covering all test conditions from the partitions – a process that can become time consuming and tedious. Instead, only one condition from each partition is tested, as its behavior is likely to be similar to the behavior of the other conditions within the same partition.

**Boundary value analysis**

To assess the inputs at the edge or boundary of each equivalence partition, you use the boundary value analysis technique. This type of testing helps yield the most defects because the behavior of test inputs are more likely to be incorrect at the boundaries, where the partition changes from one to another.  
  
For example, while testing software that calculates the interest rates for a bank account, you've defined partitions as $10 to $1000. So while you can test any value between this range, in boundary value analysis, you will design tests that cover 10 and 1000, as both set boundaries to the partition.

**Decision table testing**

Decision tables help you validate system requirements that contain logical conditions, and this type of testing is used to record complex business rules that a system needs to implement. In this technique, the specifications of the system are analyzed, and its conditions and actions are identified. The input conditions and actions for this analysis are then stated as either true or false (as boolean values) in a table format.  
  
The strength of this testing technique is that it creates combinations of conditions that might not have otherwise been exercised during testing.

**State transition technique**

The state transition technique uses the state transition diagram to test a software application that modifies its response depending on its current conditions or previous history of its state. These state changes may be represented in a state transition diagram. This technique allows the tester to view the software in terms of its states, transitions between states, the inputs or events that trigger state changes, and the actions that may result from those transitions.  
  
A state table shows the relationship between the states and inputs, and can highlight possible transitions that are invalid. Tests can be designed to cover these states as well as these transitions.

**Use case testing**

Testing based on use cases, which help you identify the different types of interactions between actors (users or systems), is use case testing. Because use cases describe process flows through a system based on its actual likely use, test cases derived from use cases are most useful in detecting defects in the process flow during the real-world use of the system. This type of technique helps in acceptance tests.

Question

Match each black-box test technique with its usage in a testing environment.

**Options:**

1. Equivalence partitioning
2. Use case testing
3. Decision table technique
4. Boundary value analysis
5. State transition technique

**Targets:**

1. Validates system requirements that contain logical conditions with associated actions
2. Classifies inputs as logical classes if they exhibit similar behavior
3. Tests the values at the edge of an equivalence partition
4. Helps create test cases that are based on an event that cause different states in the software
5. Uncovers defects in the process flows of a system

Answer

*In the Decision table technique, system specifications are analyzed and its conditions and actions are identified.*

*Equivalence partitioning classifies inputs into logical partitions, for which test cases are created. Each partition covers a specific aspect of the software.*

*You use the boundary value analysis technique to test inputs at the edge of a partition because behaviors of these inputs are likely to be erroneous at these boundaries.*

*The state transition technique allows you to view the software in terms of its states, transitions between the states, and the actions which may result from those transitions.*

*Use cases describe the process flows through a system based on its actual likely use.*

**Correct answer(s):**

Target 1 = Option C

Target 2 = Option A

Target 3 = Option D

Target 4 = Option E

Target 5 = Option B

If software behaves in an identical way for a set of values, then the set is termed as an equivalence class or a partition. The functionality of the software will be the same for any data value within the equivalence class or partition. In equivalence partitioning, input data is analyzed and divided into equivalent classes, which produce the same output.  
  
To implement equivalence partitioning, you first identify the equivalence classes or partitions for the system and then pick values from each partition for a complete coverage.

Note

*Equivalence partitioning can be applied for all levels of testing.*

Equivalence partitions include both valid and invalid partitions. Valid partitions consist of values that invoke the correct output expected from the software. An invalid partition on the other hand, includes values that are not specified to be checked, but can be entered within a system. The system would process these values to provide an error response.  
  
For example, a system awards grades to students based on their test marks, where anyone scoring between 1 - 39 is graded F, 40 - 74 is graded C, 75 - 89 is B, and between 90 - 100 gets an A as the grade. The mark ranges specified provide four partitions in this example. If a student scores anything in the specified range, the system should invoke the associated grade for that student, so values within that partition are considered equivalent.

In the grade example, suppose Carla has scored 73% on her test. Her mark comes under the partition 40 - 74 and the output for the same would be the C grade. In addition, inputs that aren't part of any of the partitions in this example, such as 105 or a non-numerical value, would be classified as invalid inputs.

In any program that accepts an integer value, zero is best left as a separate partition, as it is an undefined value that is neither part of the positive nor the negative integers. In addition, it would not be part of the specifications of most software systems.

Suppose you want to use the equivalence partitioning technique to prepare a test case for movie ticketing software that charges customers for movie tickets – $5, $8, and $10 – based on their age.

For children under 12 years of age, the ticket charge is $5. For those over 65 years, the charge is $8, and $10 is charged for all others. Using equivalence partitioning, you can classify these age values into partitions as:

**0 to 12**

Children in the age group 0 to 12 would be charged $5 for the movie tickets, hence this would be the first partition. Any age value in this range would produce the output of $5 when this function works correctly in the software.  
  
For this partition, your sample inputs can be 2, 4, 12, and 10 as valid inputs and 13, -2, and "Children" as invalid inputs. These invalid inputs are not considered as part of this partition - that is, they do not produce the same result as values within the 0 - 12 partition.

**13 to 64**

The second partition in this test case would be for customers who fall in the age group of 13 to 64. These people would be charged $10 for the ticket.  
  
Sample inputs for this partition can include 24, 47, 14, and 64 as valid inputs, which should result in the software charging $10. Values such as 70, 2, and "Adults" are considered invalid inputs.

**65 to 110**

People between the ages of 65 to 110 would become the third valid partition in the test case. The third ticket rate of $8 is meant for this group. Your test cases would then check if values between 65 and 110 return the output of $8.  
  
The sample inputs for this partition can be 78, 99, 109, and 65 as valid inputs and 300, 111, and "Seniors" as invalid inputs.

In most real-world scenarios, partitions can become highly complex due to multiple ranges in the input values and the different parameters that decide these values. For example, consider an inventory tracking system for clothing in a multi-brand boutique. To prepare test cases for this system, the simplest type of partition will be to begin with two partitions – men's wear and women's clothing.

But the range of clothes does not stop with these two classes. Men's wear can be further partitioned into shirts, trousers, and undergarments, for instance. And women's clothing partitions can increase to include tops, trousers, skirts, and dresses. There can also be further partitions based on brand, price range, size, and so on.

For a complete coverage of this inventory tracking software, you should prepare partitions for all these levels, identifying inputs – valid and invalid – for each partition, and then run tests for each partition.

Question

You want to write a test case for software that tracks the discount provided for photocopying in bulk. According to the discount chart, up to nine copies cost 10 cents each. Further, ten copies or more cost 9 cents each, while 8 cents per copy is the cost for 100 copies or more and 6 cents each for 1000 copies or more. What valid equivalence partitions would you identify for this test case?

**Options:**

1. 1 to 9
2. 10 to 99
3. 10 to 100
4. 100 to 1000

Answer

***Option 1:****Correct. 1 to 9 is a valid partition that includes values covering the number of copies within this range that returns the output of 10 cents charged per copy.*

***Option 2:****Correct. The 10 to 99 partition is valid because the software implements a discount charge of 9 cents each for photocopying ten copies or more. Any value within this range then should return the output of 9 cents.*

***Option 3:****Incorrect. The 10 to 100 partition is faulty as all values included within this range are not equivalent. Ten copies cost 9 cents, while 100 copies cost 8 cents.*

***Option 4:****Incorrect. The 100 to 1000 partition is faulty as all values included within this range are not equivalent. 100 copies cost 8 cents, while 1000 copies cost 6 cents.*

**Correct answer(s):**

1. 1 to 9  
2. 10 to 99

**2. Summary**

Among the different test techniques that drive the software test process, black-box testing is a dynamic, specification-based test technique. It derives test cases directly from the specifications of the software being tested. There are five major black-box test techniques – equivalence partitioning, boundary value analysis, state transition technique, use case testing, and decision table technique.  
  
Equivalence partitioning is based on the idea of grouping the inputs or outputs of the system under testing in equivalent partitions. Values within each partition are considered equivalent in that when used as an input, they will result in the same behavior as expected for that partition. Consequently, this technique requires only one test case for each partition. To ensure that most specifications of the system are tested, you should then identify as many valid partitions for your software as possible.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_sftf_a04_it_enus/output/html/course_transcript.html#top)

Boundary Value Analysis in Software testing

Learning Objective

*After completing this topic, you should be able to*

* *determine boundary values for a boundary analysis*

**1. Determining boundary values**

Boundary value analysis is a black-box test technique that is often executed along with equivalence partitioning. Both these techniques, when implemented together, help you prepare stringent test cases that can uncover most defects in a software application.

In equivalence partitioning, you classify the input or the output values into equivalent partitions. These partitions include boundary values, which mark the beginning and the end of a partition. Boundaries are the input/output values found at the edge of an equivalence partition. Boundary values also include the smallest incremental values found at either side of these boundaries. Creating test cases for verifying these values is termed as *boundary value analysis*.

Note

*Boundary value analysis is commonly known as range checking and can be applied at all levels of testing.*

Boundary value analysis is very important for any system whose values are divided into partitions because the system's function is most likely to change when these boundary values are hit.   
  
For example, suppose a software grades student – F, P, D, and O – based on their marks and on this basis you assign four partitions – 0 to 39, 40 to 74, 75 to 89, and 90 to 100 respectively for each specified grade.

The boundary values for the first partition in this example are 0 and 39, with 0 as the lowest boundary value and 39 the highest. Similarly, marks specified at the edges of other partitions become boundary values for the remaining partitions. The grade changes at the edges of these partitions – for instance, from F to P when a student scores 40 – define a change of outcome in the test environment. This change makes boundary zones prone to errors. So, you should specifically concentrate on the boundary values while implementing test suites.

While it is easy to identify two boundary values for each partition, some practitioners of the boundary value analysis technique note that every boundary in a partition includes three boundary values. To identify multiple boundary values for both lower and upper boundaries, you identify the next precision value in the sequence of the partition. By identifying all these boundary values and executing test suites for the same, you ensure that most errors that occur in this zone can be uncovered.

In the student grade example, for the partition of 75 to 89, the numbers are to a precision of 1. So the lower boundary values can be identified as 74, 75, and 76 so that the values to the precision of 1 preceding and succeeding the actual boundary are taken into account. Similarly, the upper boundary values are identified as 88, 89, and 90. In this example, you should prepare six test cases, one for each boundary value identified.

Boundary value analysis is applied to values such as numbers, text, dates, time, and currency. Among these, two values that benefit are

**integers**

Boundary value analysis for integer values is simple and most effective. You want to test a software system that tracks the number of telephone calls a customer care helpline receives per day. Past records mention 250 as the maximum number of calls received. In this case, you can have three partitions, one valid partition being 1 to 250 and two invalid partitions being less than 0 to 0 and 251, to the theoretical maximum value. The valid boundary values for the partition 1 to 250 can include three lower boundary values – 0, 1, and 2 and three upper boundary values – 249, 250, and 251. Testing these boundaries would suffice unless these calls can be further partitioned based on other parameters such as calls from specific geographical regions.  
*Description of the An example with the Boundary Value Analysis for Integers table:  
  
This table is a single row table with four columns – Conditions, Valid partitions, Invalid partitions, and Valid boundaries. The condition for this boundary value analysis is calls per day. The valid partition is specified as 1 to 250 and the invalid partitions are mentioned as < 0 to 0 and 251 to maximum value. The Valid boundaries column includes the values of boundaries as 0, 1, 2 and 249, 250, and 251.  
  
Description ends.*

**real numbers**

Applying boundary value analysis for real numbers is more complicated than integer value evaluation. Real numbers include floating-points, due to which you should identify the smallest difference between the values that form the input sequence. This difference is termed as epsilon – ∑. You are testing a system that groups people, based on their height as short, average, and tall. People up to the height of 4 ft are grouped as short, those with the height of 5.5 ft are average, and those measuring above 5.5 ft are tall. In this sequence, ∑ for height calculation is 0.1, which becomes the basic unit value for identifying boundaries. So for the average group in this case, the lower boundaries are 4.0, 4.1, and 4.2 and the upper boundaries are 5.4, 5.5, and 5.6.  
*Description of the An example with the Boundary Value Analysis for Real numbers table:  
  
This table consists of four columns - Conditions, Partitions, Lower boundary values, and Upper boundary values, and three rows – Short, Average, and Tall. For the condition Short, the partition is 0 to 4.0. It's lower boundary values are -0.1, 0, and 0.1 and upper boundary values are 3.9, 4.0, and 4.1. For the condition Average, the partition is 4.1 to 5.5. It's lower boundary values are 4.0, 4.1, and 4.2 and it's upper boundary values are 5.4, 5.5, 5.6. For the condition Tall, the partition is 5.6 to the maximum value accepted by the system. Its lower boundary values are 5.5, 5.6, and 5.7 and it's upper boundary values are not defined.  
  
Description ends.*

Suppose you want to prepare a test case for software that charges buyers for movie tickets based on their age. So if a buyer is in the age group of 12 or under, the software charges $5, if they are between 13 and 64, it charges $10 and $8 for people above age group of 64.  
  
If employing the boundary value analysis technique to evaluate the software, you would first identify the partitions, which in this example are set as 0 to 12, 13 to 64, and 65 to a theoretical maximum for an age value.

Try It

You now want to identify the boundary values for these partitions to write specific test cases for the same. You want to determine the lower boundary values for the first partition – 0 to 12 – with the ∑ value identified as one.

*Description of the Boundary value analysis for movie theater charges table:  
This table consists of four columns – Conditions, Partitions, Lower boundary values, and Upper boundary values. For the condition $5, the partition is 0 to 12. For the condition $10, the partition is 13 to 64. For the condition $8, the partition is 65 to 110. All lower and upper boundary values appear empty.  
Description ends.*

To complete the task

1. Type -1, 0, 1 in the Lower boundary values column for the 0 to 12 partition and click **Submit**

The lower boundary values for the partition 0 to 12 are identified.

After identifying the lower boundary values for the 0 to 12 partition, you determine its upper boundary values. Similarly, for the remaining partitions – 13 to 64 and 65 to a theoretical maximum – you identify the lower and upper boundary values.

Graphic

*Description of the Boundary value analysis for movie theater charges table:  
This table consists of four columns – Conditions, Partitions, Lower boundary values, and Upper boundary values. For the condition $5, the partition is 0 to 12. It's lower boundary values are -1, 0, 1 and upper boundary values are 11, 12, 13. For the condition $10, the partition is 13 to 64. It's lower boundary values are 12, 13, 14 and upper boundary values are 63, 64, 65. For the condition $8, the partition is 65 to 110. It's lower boundary values are 64, 65, 66 and upper boundary values are 109, 110, 111.  
Description ends.*

In the example of the movie theatre, for the partition 65 to 110, the boundary value of 110 is assumed as a theoretical maximum for age. However, identifying a specific theoretical maximum boundary for a partition isn't always possible. Consider the height measurement example where the partition categorizing people as tall is over 5.6 ft. The maximum value accepted by the system is left without identifying any theoretical maximum value because there is no established limit to what the maximum height can be. In this case, the upper boundary value is not defined. This is an example of an *open boundary*, where one edge of the partition is left open.  
  
To test such boundaries, you should check system specifications to identify if any possible value can take the place of the theoretical maximum.

**Question Set**

Understanding how boundary value analysis can be applied takes time. Answer these questions to check your knowledge on this black-box technique.

**Question 1 of 3**

Question

You want to prepare a test case for the software that tracks the discount provided for bulk photocopying. According to the discount chart, up to nine copies cost 10 cents each. Ten copies or more cost 9 cents each. 100 or more copies cost 8 cents each while an order of 1000 copies or more costs 6 cents each. Four partitions based on the number of copies to be photocopied are identified for this test case: 1 to 9, 10 to 99, 100 to 999, and 1000 to a theoretical maximum value. You now want to identify the three boundary values for these partitions taking into consideration the ∑ value, which is 1 in this case.  
  
Enter the first lower boundary value for the first partition 1 to 9.

*Description of the Boundary value analysis for photocopying charges table:  
This table consists of four columns – Conditions, Partitions, Lower boundary values, and Upper boundary values. The four conditions are specified in four rows based on the amount charged per copy – 10 cents, 9 cents, 8 cents, and 6 cents, in the order listed. Against each condition, partition values are specified though the columns for lower and upper boundary values for each are currently empty.  
Description ends.*

Answer

The lower boundary values for the partition 1 to 9 include 0, 1, and 2. The ∑ value for this sequence is 1 and the values from this partition would return the output 10 cents.

**Correct answer(s):**

1. 0

**Question 2 of 3**

Question

Enter the second lower boundary value for the first partition 1 to 9.

Answer

The lower boundary values for the partition 1 to 9 include 0, 1, and 2. The ∑ value for this sequence is 1 and the values from this partition would return the output 10 cents.

**Correct answer(s):**

1. 1

**Question 3 of 3**

Question

Enter the second upper boundary value for the first partition 1 to 9.

Answer

The upper boundary values for the partition 1 to 9 include 8, 9, and 10. The ∑ value for this sequence is 1 and the values from this partition would return the output 10 cents.

**Correct answer(s):**

1. 9

**2. Summary**

Boundary values analysis is an important black-box testing technique that helps create test cases for different types of values. In boundary value analysis, you identify the input/output values found at the edge of a partition as boundaries before analyzing them, as these values are often error-prone in most applications.  
  
This technique is especially useful for integer and real number values, where you identify a minimum of three boundary values for lower and upper boundary values. You can then create test cases for these boundary values.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_sftf_a04_it_enus/output/html/course_transcript.html#top)

Advanced Partitioning and Boundary Value Analysis

Learning Objective

*After completing this topic, you should be able to*

* *recognize how to construct software test cases using advanced equivalence partitioning and boundary value analysis*

**1. Using EP and BVA in software testing**

Equivalence partitioning and boundary value analysis are two popular black-box testing techniques. In equivalence partitioning (EP), input data is divided into equivalent classes or partitions based on the conditions you want to test. It helps to create partitions so that you need to test only a couple of values from a partition and not all the values. If those values are valid for the partition, all values in that partition would be valid.

In boundary value analysis (BVA), you test the values at the boundary of an equivalent partition. It is important to perform boundary value analysis because the system's functions might change when exercising the values at the boundaries.

You use equivalence partitioning and boundary value analysis techniques when you want to perform rigorous and thorough testing.

If you perform only BVA, it means you've tested the boundary values for each partition and every boundary falls in some partition. However, if the test fails, you would not know whether the boundary values or the partitions failed. Also, you may miss out on checking some values in the partitions.

Because the values at the boundaries are extreme, it would not be appropriate to test only boundary values. You should use a combination of the techniques to create minimum, but efficient test cases.

Equivalence partitioning and boundary value analysis techniques can be used for all levels of testing. At component level, it is easier to create and test the partitions and boundaries. However, at system-level testing, it would be more complex. For example, if you are installing a software application, there are three installation types available – typical, custom, and advanced. Each of these types, along with its options, is a partition. You need to test each installation type along with all the options to adequately cover the application.

When creating test sets you should select test values from within the partitions but not the boundary values.  
  
Consider a situation where you want 1000 photocopies of a document. For 500 to 1000 copies, you have to pay 8 cents and if you want less than 500 copies, you have to pay 10 cents. In this example, one of the valid partitions is 500 to 1000.

When performing boundary value analysis, you need four test values: 500, 501, 999, and 1000. Therefore, a separate equivalent partition value is not required as it would be redundant. However, you can replace the values 501 and 999 with one value from the equivalent partition: 750. This would prove more efficient because it covers the boundaries as well as the partition with the fewest possible test cases.

You can use both EP and BVA to create effective test cases for a given situation. For example, you can use these test techniques in situations, such as

**calculating student grades based on marks**

You want to test an application that prints the students' grades based on their scores. Students who score above 80 are assigned A, who score from 61 to 80 are assigned B, and those who score from 41 to 60 are assigned C. Those who score 40 or less will fail the test. For this example, equivalent and valid partitions are 0 to 40, 41 to 60, 61 to 80, and 81 to 100. Invalid partitions would be the values less than 0 and values greater than 100. To test these partitions and boundary values, you use test values, such as 39, 40, 41, 59, 60, 61, 79, 80, and 81. This test set contains optimum values because it covers the boundaries and also some values in each partition.

**calculating tax for employees**

The tax calculation for employees depends on their incomes. Employees with salaries $0 to $35,000 need to pay 15%, those with salaries from $35,001 to $60,000 need to pay 20%, and those with salaries $60,001 to $100,000 need to pay 25%. All those whose salaries are above $100,001 will be taxed at 30%.   
  
In this example, the valid partitions are $0 to $35,000, $35,001 to $60,000, and $60,001 to $100,000. The values less than 0 are in the invalid partition. The boundary values are $0, $35,000, $35,001, $60,000, $60,001, $100,000, and $100,001. In addition to testing the boundary values, you should test the application for negative and decimal values, such as 0.01 and non-numeric characters.

Depending on the application and your test objective, you can create test cases to cover some or all the boundaries and partitions.  
  
If you want to perform thorough testing, you need to test the valid and invalid partitions and boundaries. However, if you have a limited time you can test only the valid partitions and boundaries. Similarly, if you want to test the reaction of the system to incorrect inputs, you test only the invalid partitions and boundaries.

The examples for calculating student grades and tax liabilities use integer values. In addition to integer values, you can use EP and BVA to create test cases for other inputs, such as

* real numbers
* text strings
* dates
* times
* currency

When testing with real numbers, the situation could become complicated because real numbers include decimal points.

Consider a scenario where you have to test a loan application. If you input loan amount, the tenure, and the rate of interest on loan, the application calculates the monthly installments. The rates of interest have been fluctuating between 9% and 12% and you want to know the average rate of interest for the previous financial period.

In this example, the valid partition is 0 to 12 and the invalid partitions are less than 0 and greater than 12. When creating partitions, you need to consider the epsilon for real numbers. If the epsilon is one thousandth of a dollar, then your boundary values can be 0.001 and 11.999. The invalid boundaries would be -0.001 and 12.001. Invalid boundaries would also include using wrong decimal separators. In such instances you should verify the level of precision required. Based on that, the partitions and boundary values would be affected.

Consider a situation where you want to test a payroll web site. You can log on to the web site by providing your employee ID.  
  
An employee ID is an input string that should begin with the letters D, E, or F depending on whether the employee is a consultant or on contract. The input string should also be between 10 and 99,999. In this example, if you put all the conditions together, you would get eight invalid partitions.

If you perform EP, you use values such as 10, 99, 10,000, and 99,999 because these are values from the valid partition. You would use values such as 9 and 100,000 to test the invalid partitions. So you need 6 tests for performing EP. You need additional 4 test sets to cover the valid boundaries.

Sometimes, such test cases are based on assumptions that are faulty. A good test case involves using values that test whether the employee ID is between 10 and 99,999 and checks whether the ID begins with D, E, or F. You can come up with numerous test cases for invalid partitions and boundaries. However, you can avoid testing for numbers lower than 9 or 8 and numbers that are much greater than 99,999.  
  
In such cases where you allow letters, you should consider checking the largest and smallest possible valid ASCII values.

Consider another example where you're testing the entries in the date field. A date field consists of subfields – day, month, and year – that could influence each other. When testing date values, you need to consider whether the application accepts dates in a day-month-year or month-day-year format.  
  
You want to check the list of people who are eligible for appraisal in the current cycle. Only those employees who've joined after December 31 are eligible for performance appraisal.

Assume that the date format is month-day-year. Depending on this format, you would first test whether the month falls between 1 and 12. The range between 1 and 12 is the valid partition. When performing EP, you could check whether the application accepts 10 as the month. This value falls in the valid-partition range. However, you could also check for 0 and 13; these fall in the invalid-partition range.

The month will determine the number of days to be tested. In this example, because the month is December, the number of days is 31. So when performing EP, the valid partition is 1 to 31. If your test data is 25, for example, it falls in the valid-partition range. You should also check for values from the invalid partitions.

Note that if you had to create test cases to test for the month of February, then you need to consider the year. Based on the year, you need to check if it is leap year and if it is, the number of days in February would be different. Because of this type of complexity, BVA for a date field can become tricky.

In addition to date values, you can perform EP and BVA with time values. When creating test cases based on time values, you need to consider the subfields, such as hours, minutes, and seconds. In addition, you need to know the time format the application uses – 12-hour or 24-hour format. Consider you're testing an auction web site in which you specify that the auction closes at 6:00 p.m.

In this example, to simplify the explanation we won't consider the time in seconds. Assuming that the time will be displayed in the 12-hour format, a valid time format would be 6:00. This would mean a time entered as 18:00 is invalid because it is not in the proper format.

For invalid time, you can check for any values, hour can be anything other than 6 and minutes can be any value between 0 and 59. Testing for time can become complex. For example, some states observe daylight savings and others don't while in some parts of the world, the summertime starts and ends on different days – further complicating the daylight savings time test case. These considerations make performing rigorous tests on date fields challenging.

You can perform EP and BVA on currency fields as well. When performing the tests on currency, you need to determine whether the currency format is dollars, pounds, or any other. You need to also determine whether the application accepts decimals and fractions.

Consider an auction application where you need to enter the bid price between $1 and $500 for an item. Assuming decimals are not allowed, the valid partition would be between $1 and $500. You could have test values from the valid partition and you could also test the boundaries. To test format supported, you could design a test case by inserting a dollar sign ($) with a value from the valid partition.

From invalid boundaries, you could test for values such as 500.01. If the application uses various currencies, you can test for each currency. If the application doesn't support different currencies, you can still test with different currencies. However, they would fall in the invalid partitions. You can apply this combination technique at any level, such as unit and acceptance testing.

Question

You're testing a financial application that calculates the interest rates for the amount in fixed deposit accounts based on tenures. For fixed deposit amounts for a period of 2 to 5 years, the interest is 7%, for the period between 6 and 8 years, the interest is 9%, and for amounts deposited for 9 to 15 years, the interest is 12%.  
  
Identify the test cases that use equivalence partition and boundary value analysis techniques to test all the partitions and boundaries.

**Options:**

1. Test set A with values 1, 16, and 15
2. Test set B with values 1, 2, 4, 5, 6, 7, 8, 9, 14, and 15
3. Test set C with values 0, 1, 2, 3, 4, and 5
4. Test set D with values 3, 5, 10, and 12

Answer

***Option 1:****Incorrect. The values in this test case set tests for only invalid partitions and tests only one boundary – 9 to 15.*

***Option 2:****Correct. This is an exhaustive test set because it tests for boundary values and values in all partitions.*

***Option 3:****Incorrect. This test set contains values that test only one invalid partition and one valid partition. It also checks only the boundary values of one partition – 2 to 5.*

***Option 4:****Incorrect. These values do not test every partition, and they don't test the boundary values, so this can't be considered an exhaustive test set.*

**Correct answer(s):**

2. Test set B with values 1, 2, 4, 5, 6, 7, 8, 9, 14, and 15

Question

You're testing a credit card application in which you need to check whether the customer pays credit card bills by 5:00 p.m. on the 5th of every month. You are using the 12-hour time format and want to currently check only for the time values.  
  
Which is the most appropriate test set that you can create using EP and BVA?

**Options:**

1. Test set A with hour value 17:00
2. Test set B with hour value 5
3. Test set C with minute value -1
4. Test set D with minute value 62

Answer

***Option 1:****Incorrect. You are using the 12-hour time format so if you enter 17:00, it would be considered invalid.*

***Option 2:****Correct. This is the correct test value because it falls within the valid equivalent partition.*

***Option 3:****Incorrect. This cannot be a valid test set because it only checks for a value in the invalid partition.*

***Option 4:****Incorrect. The value of 62 is a value outside the valid partition for minutes.*

**Correct answer(s):**

2. Test set B with hour value 5

**2. Summary**

You can use equivalence partitioning (EP) and boundary value analysis (BVA) to perform rigorous and thorough testing. If you perform only BVA, you may miss out on checking some values in the partitions. Therefore, you should use a combination of the techniques so that you create minimum, but efficient test cases.  
  
You can use EP and BVA for real numbers, integers, text strings, dates, times, and currencies.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_sftf_a04_it_enus/output/html/course_transcript.html#top)

Writing Software Test Cases

Learning Objectives

*After completing this topic, you should be able to*

* *write a software test case using equivalence partitioning*
* *write a software test case using equivalence partitioning and boundary value analysis*

**1. Exercise Overview**

**2. Task 1: Using EP**

**3. Task 2: Using EP and BVA**

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_sftf_a04_it_enus/output/html/course_transcript.html#top)

Decision Tables in Software Testing

Learning Objective

*After completing this topic, you should be able to*

* *recognize how decision tables work in software testing*

**1. Using decision tables to test software**

In software testing, decision tables are used to create test cases for software that implements complex logic, where different conditions produce different outcomes. They help ensure that all possible test cases are accounted for during the testing process.

A decision table contains four sections.

**conditions**

Conditions are activities that you perform in combinations during testing. Each combination is known as a test case.  
  
In this example, the decision table contains two conditions – C1 and C2.  
*Conditions in a decision table are entered on the top rows of the first column. In this example, the first two rows of the first column have the condition C1 and C2.*

**condition entries**

Condition entries are simple boolean values, such as yes and no or true and false, that are used to create all possible rules. These entries are usually represented through their first initial. For example, yes is represented as Y and false is represented as F.  
  
In this example, the values T and F are used.  
*Condition entries are added in columns to the right of the conditions column. In this example, the second column has T in both the C1 and C2 rows, the third column has T in the C1 row and F in the C2 row, the fourth column has F in the C1 row and T in the C2 row, and the fifth column has F in both the C1 and C2 rows.*

**action**

An action is the result expected from the application when a test case is performed. During testing, if an action for a test case matches the action listed in the decision table, the tester does not log a bug. If the action does not match the action listed in the table, the tester logs a bug.  
  
In this example, when a tester specifies both conditions as true, the software should perform the action A1. If not, the tester should log a bug and test the next test case.  
*Actions in a decision table are placed directly below the conditions in the first column. In this example, the decision table contains two actions – A1 and A2.*

**action entries**

Action entries are values, such as crosses or tick marks that indicate which actions are possible for a test case and if no action entries are present then no action takes place for the condition.  
*Action entries are added in columns to the right of the Actions column. In this example, the second column has X in the A1 row, the third column has X in the A2 row, the fourth column has X in the A2 row, the fifth column has no entries of X.*

Consider that you want to test an application using a decision table. This application calculates the discount that a customer should receive in a department store.  
  
To create a decision table, you first list all conditions and expected actions. In this example, customers will receive a discount when they use a Credit card, a Coupon, or both. So you specify the conditions as Credit card and Coupon and the actions as 0%, 5%, 2%, and 7%.

You then calculate the possible number of test cases. To do so, you use this formula: number of condition entry values raised to the number of conditions. In this example, there are two values – Yes and No – and two conditions – Credit card and Coupon – so you add four columns for the test cases.

You then use the values Y and N to create test cases and use crosses to mark the expected actions for them. The test cases and actions in this example are

* when a customer does not use a Credit card or a Coupon, the application should calculate the discount as 0%   
  *In the second column, you enter N in the Credit card and Coupon rows and an X in the 0% row.*
* when a customer uses a Credit card but not a Coupon, the application should calculate the discount as 5%   
  *In the third column, you enter Y in the Credit card row, N in the Coupon row, and X in the 5% row.*
* when a customer uses a Coupon but not a Credit card, the application should calculate the discount as 2%   
  *In the fourth column, you enter N in the Credit card row, Y in the Coupon row, and X in the 2% row.*

Try It

To complete the decision table, you want to indicate that when a customer makes a payment using both a Credit card and a Coupon, the application should calculate the discount as 7%.

To complete the task

1. In the fifth column, type Y in both the Credit card and Coupon rows and press **Enter**  
   *The Courseware Player cannot recognize input from your keyboard Enter key. Therefore, please click the Enter button provided instead of pressing the Enter key.*
2. In the fifth column, type X in the 7% row and press **Enter**  
   *The Courseware Player cannot recognize input from your keyboard Enter key. Therefore, please click the Enter button provided instead of pressing the Enter key.*

You've finished creating the decision table.

During testing, when you enter the bill amount as $200 and specify that the customer has used both a Credit card and a Coupon, the application should calculate the discount as 7%. If not, you log a bug.

Similarly, when you enter the bill amount as $150 and specify that the customer has used only a coupon, the application should calculate the discount as $3. If this doesn't happen, you log a bug and test another condition defined in the decision table.

Now suppose that the store policy doesn't allow customers to use both a Credit card and a Coupon at the same time. In this case, you would add an additional action called Error to check if the system produces an error when you specify that both a Credit card and a Coupon have been used.

Graphic

*An additional action – Error – is added, and in the fifth column, X is entered in the Error row. The X on the 7% row is removed.*

As an alternative, the software developer might also have programmed the application to accept the Credit card because it provides a higher discount than the Coupon. In this case, you would modify the decision table to check if the application calculates the discount as 5% when you specify that both a Credit card and Coupon have been used.

Graphic

*In the fourth column, X is entered in the 5% row.*

In the previous example, only one action was possible for each test case. In some software, implementing a test case may also result in multiple actions.  
  
For example, consider a decision table for an application that registers users on a web site. The application should perform two actions for users who specify a user name, a password, and an e-mail address – the user should be registered and should receive an e-mail message. So both these actions have been marked for the first test case.

Graphic

*Description of the decision table.  
In the first column, there are three conditions – User name, Password, and E-mail – and two actions – Register and Send E-mail. In the second column, there is Y in the User name, Password, and E-mail rows and X in the Register and Send E-mail rows. In the third column, there is Y in User name and Password rows, N in the E-mail row, and X in the Register row. In the fourth column, there is a hyphen in the User name row, N in the Password row, Y in the E-mail row, and X in the Send E-mail row. In the fifth column, there is a hyphen in the User name row and N in the Password and E-mail rows. In the sixth column, there is N in the User name row, a hyphen in the Password row, Y in the E-mail row, and an X in the Send E-mail row. In the seventh column, there is N in the User name and E-mail rows and a hyphen in the Password row.  
Description ends.*

This decision table also contains hyphens in the condition entry section. These hyphens are known as *don't care* entries. Don't care entries are used when using any regular entry doesn't influence the action. For example, if a user doesn't specify a password, it doesn't matter if the user has specified a user name or not; the action performed by the application should be the same – the application should show an error. Don't care entries reduce test cases, so there are only six test cases instead of eight in this example.

You can also use a decision table with the Boundary Value Analysis (BVA) method for testing software.   
  
Consider that you've created a decision table for an application that calculates the fine that an account should be charged if it contains less than $1,000. If an account contains less than $500, it should be charged a fine of $50, and if the account contains between $500 and $999, it should be charged a fine of $20. However, gold accounts are charged half the fine.

Graphic

*Description of the decision table.  
In the first column, there are three conditions – $0 - $499, $500 - $999, and Gold Account – and four actions – $20, $50, $10, and $25. In the second column, there is Y in the $0 - $499 and Gold Account rows and X in the $25 row. In the third column, there is Y in the $0 - $499 row, N in the Gold Account row, and X in the $50 row. In the fourth column, there is Y in the $500 - $999 and Gold Account rows and X in the $10 row. In the fifth column, there is Y in the $500 - $999 row, N in the Gold Account row, and X in the $20 row.  
Description ends.*

Note

*In this example, don't care entries are used to indicate that an account cannot have a balance between $0 and $499 and between $500 and $999 at the same time.*

According to the BVA method, mistakes in software are often made at the boundaries of conditions. In this example, a fine of $10 should be charged for gold accounts that have a balance between $500 and $1,000.  
  
However, it's possible that the developer might have entered the wrong symbol so a fine of $25 is charged for gold accounts with less than or equal to $500. It's also possible that a fine of $25 is charged for gold accounts with a balance of $500.

So you need to test if gold accounts with $499 are charged a fine of $25 and gold accounts with $500 and $501 are charged a fine of $10. You should also test if standard accounts with $499 are charged a fine of $50 and standard accounts with $500 and $501 are charged a fine of $25. Similarly, you should also test values around the other three boundaries – $0, $499, and $999.  
  
So you use the BVA method to increase the scope of the decision table. You specify ten boundary values in the conditions column – $-1, $0, $1, $498, $499, $500, $501, $998, $999, and $1,000.

Graphic

*Description of the decision table.  
In the first column, there are 11 conditions – $-1, $0, $1, $498, $499, $500, $501, $998, $999, $1,000, and Gold Account – and four actions – $20, $50, $10, and $25. In the second column, there is Y in the -$1 row. In the third column, there is Y in the $0 and Gold Account rows and X in the $25 row. In the fourth column, there is Y in the $0 row, N in the Gold Account row, and X in the $50 row. In the fifth column, there is Y in the $1 and Gold Account rows and X in the $25 row. In the sixth column, there is Y in the $1 row, N in the Gold Account row, and X in the $50 row. In the seventh column, there is Y in the $498 and Gold Account rows and X in the $25 row. In the eighth column, there is Y in the $498 row, N in the Gold Account row, and X in the $50 row. In the ninth column, there is Y in the $499 and Gold Account rows and X in the $25 row. In the tenth column, there is Y in the $499 row, N in the Gold Account row, and X in the $50 row. In the eleventh column, there is Y in the $500 and Gold Account rows and X in the $10 row. In the twelfth column, there is Y in the $500 row, N in the Gold Account row, and X in the $20 row. In the thirteenth column, there is Y in the $501 and Gold Account rows and X in the $10 row. In the fourteenth column, there is Y in the $501 row, N in the Gold Account row, and X in the $20 row. In the fifteenth column, there is Y in the $998 and Gold Account rows and X in the $10 row. In the sixteenth column, there is Y in the $998 row, N in the Gold Account row, and X in the $20 row. In the seventeenth column, there is Y in the $999 and Gold Account rows and X in the $10 row. In the eighteenth column, there is Y in the $999 row, N in the Gold Account row, and X in the $20 row. In the nineteenth column, there is only Y in the $1,000 row.  
Description ends.*

Question

Using a decision table, you're testing an application that allows users appropriate access to a database based on their position in the company.  
  
What result should you expect from the application when you test it as a junior employee with no administrative rights?

*Description of the decision table.  
In the first column there are four conditions – Junior Employee, Senior Employee, Administrator Rights, and Vendor – and four actions – Read, Edit, Delete, and Share. In the second column, there is Y in the Junior Employee and Administrator Rights rows and X in the Read, Edit, and Share rows. In the third column, there is Y in the Junior Employee, N in the Administrator Rights rows, and X in the Read and Edit rows. In the fourth column, there is Y in the Senior Employee and Administrator Rights rows and X in the Read, Edit, Delete, and Share rows. In the fifth column, there is Y in the Senior Employee, N in the Administrator Rights rows, and X in the Read, Edit, and Share rows. In the sixth column, there is Y in the Vendor row and X in the Read row.  
Description ends.*

**Options:**

1. You should be able to read and edit files
2. You should be able to read, edit, and share files
3. You should only be able to read files
4. You should be able to read, edit, delete, and share files

Answer

***Option 1:****Correct. You should be able to read and edit files when you test the application as a junior employee with no administrative rights.*

***Option 2:****Incorrect. You should be able to read, edit, and share files when you test the application as a junior employee with administrative rights.*

***Option 3:****Incorrect. You should be able to only read files when you test the application as an external vendor.*

***Option 4:****Incorrect. You should be able to read, edit, delete, and share files when you test the application as a senior employee with administrative rights.*

**Correct answer(s):**

1. You should be able to read and edit files

Question

You are using a decision table to test an application that calculates the amount that a customer should pay for a vacation.  
  
For what types of customers should the application calculate a bill of $4,500?

*Description of decision table.  
In the first column, there are three conditions – More than four people, Regular customer, and Discount card – and five actions – $5,000, $4,750, $4,500, $4,250, and $4,000. In the second column, there is Y in the More than four people, Regular customer, and Discount card rows and X in the $4,000 row. In the third column, there is Y in the More than four people and Regular customer rows, N in the Discount card row, and X in the 4,250 row. In the fourth column, there is Y in the More than four people and Discount card rows, N in the Regular customer row, and X in the $4,500 row. In the fifth column, there is Y in the More than four people row, N in the Regular customer and Discount card rows, and X in the $4,750 row. In the sixth column, there is N in the More than four people row, Y in the Regular customer and Discount card rows, and X in the $4,250 row. In the seventh column, there is N in the More than four people and Discount card rows, Y in the Regular customer row, and X in the $4,500 row. In the eighth column, there is N in the More than four people and Regular customer rows, Y in the Discount card row, and X in the $4,750 row. In the ninth column, there is N in the More than four people, Regular customer, and Discount card rows and X in the $5,000 row.  
Description ends.*

**Options:**

1. Regular customers with more than four people, but no discount card
2. Customers with more than four people and a discount card
3. Regular customers
4. Regular customers with a discount card, but with less than four people

Answer

***Option 1:****Incorrect. The application should calculate a bill of $4,250 for regular customers with more than four people.*

***Option 2:****Correct. The application should calculate a bill of $4,500 for customers with more than four people and a discount card.*

***Option 3:****Correct. The application should calculate a bill of $4,500 for regular customers.*

***Option 4:****Incorrect. The application should calculate a bill of $4,250 for regular customers with a discount card.*

**Correct answer(s):**

2. Customers with more than four people and a discount card  
3. Regular customers

**2. Summary**

You can use decision tables to create all possible test cases for a software application. A decision table is divided into four sections – conditions, condition entries, actions, and action entries.  
  
To create a decision table, you first write the conditions and the actions in the first column and then create columns for all possible test cases. You then use each test case to check if the action performed by the software matches the action listed in the decision table. If the actions don't match, you log a bug.  
  
You can also use a decision table with the Boundary Value Analysis method to test boundary values of conditions.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_sftf_a04_it_enus/output/html/course_transcript.html#top)

State Transition Software Testing

Learning Objectives

*After completing this topic, you should be able to*

* *recognize how state transitions work in software testing*
* *recognize the functions of state transition and finite state machines in software testing*

**1. Testing state transition in software**

A software application can be in various states depending on the action its user performs. An e-mail application, for example, displays your Inbox when you launch it. When you click the control that creates a new e-mail message, the application's state changes to display a blank form. Similarly, when you type a message, the state changes to display text, and when you send the message, the state changes to display your Inbox again.

As a software tester, you need to check if state transitions like these occur correctly. If, for example, the e-mail application closes or shows your Outbox when you send a message, you log a bug. This type of testing is known as *state transition testing*.

State transition testing is based on the assumption that each application has a finite number of states, and the transition from one state to another depends on the objectives of the application. In the e-mail application example, the application's objectives include opening a blank form when you choose to create a new message and displaying the Inbox after you send a message. Such applications with finite states are known as *Finite State Machines*.

Question

What is the function of state transition testing?

**Options:**

1. To test if the application has infinite states
2. To test if changes in states are accurate
3. To test if the application has finite states
4. To test how many states an application has

Answer

***Option 1:****Incorrect. State transition testing is based on the assumption that applications have finite states. It cannot be used if an application has infinite states.*

***Option 2:****Correct. The function of state transition testing is to test if an application's state changes accurately when an event is performed.*

***Option 3:****Incorrect. State transition testing is based on the assumption that an application has finite states. You don't use this type of testing to check if an application has finite states.*

***Option 4:****Incorrect. The function of state transition testing is to test if changes in an application's states are accurate. It is not used to test how many states an application has.*

**Correct answer(s):**

2. To test if changes in states are accurate

To test state transitions of an application, you create a state transition model. This model contains four components:

Graphic

*This example shows a state transition model for an e-mail application. According to this model, you can create a new e-mail message from the Inbox state. You can then type an e-mail address, type the message text, and send the e-mail message, or you can close the form to go back to the Inbox state. You cannot send a new e-mail message without specifying an e-mail address or the message text; an error occurs when you do so. You can also open a received e-mail message from the Inbox state by double-clicking it. You can then reply to the received message or close it to return to the Inbox state. You cannot reply to a received message without specifying the reply text; an error occurs when you do so. You can also forward a received message and return to the Inbox state. You cannot forward a received message without specifying an e-mail address; you encounter an error when you do so.*

**states**

You include all states possible in the application. In this example, the e-mail application can be in nine possible states.  
*The nine possible states of this e-mail application are Inbox, New e-mail form, New e-mail form with e-mail address and message, Received e-mail, Reply e-mail form, Reply e-mail form with message text, Forward e-mail form, Forward e-mail form with e-mail address, Error.*

**events**

Events are user actions, such as a button click or text input, that trigger a state transition. There are eight events that can trigger state transition in this e-mail application.  
*The eight events that can trigger state transition in this e-mail application are E-mail double-clicked, New button clicked, E-mail address typed, Message typed, Send button clicked, Reply button clicked, Forward button clicked, and Close button clicked.*

**transitions**

A transition is the process where an application changes from one state to another when an event is performed. You use arrows to depict transitions.

**outcomes**

An outcome is a state that is the result of a transition. Outcomes are displayed near the arrowheads of transition arrows.

To test an application using the state transition model, you perform the possible events from all states to check if they change the application's state accurately.  
  
For example, this e-mail application's state should change to display a New e-mail form when you click the **New** button from the Inbox state, and when you double-click a received e-mail message from the Inbox state, the state should change to display that message. If these changes don't occur, you should log a bug.

In some cases, the same event can trigger different changes. So you should also check if an event triggers an appropriate change depending on the current state of the application.  
  
In this example, when you click the **Send** button in the Forward e-mail form, you can either return to the Inbox state or receive an error. Therefore, you should check if you return to the Inbox state when you click the **Send** button after specifying an e-mail address, and you should also check if you encounter an error when you click the **Send** button without specifying an e-mail address.

This e-mail application also enables you to view and forward sent messages by opening the Outbox. So you want to expand the scope of this state transition model so that you can also test if the state changes correctly when you open the Outbox and view a sent message. You also want to ensure that state changes to the Forward e-mail form when the **Forward** button is clicked on a sent message.

To do so, you add two more states in this example – Outbox and Sent e-mail message. You specify that

* the Outbox is displayed when the **Outbox** button is clicked from the Inbox state
* a sent message is displayed when it is double-clicked from the Outbox state
* the Forward e-mail form is displayed when the **Forward** button is clicked from a sent message
* the Inbox state is displayed when the **Close** button is clicked from the Outbox and Sent e-mail message states

Now consider that you are using a state transition model to test software used by a vending machine. This machine provides a drink worth 25 cents, and it accepts only nickels, dimes, and quarters.

Graphic

*According to this model, the state of the machine changes from 0 cents to 5 cents when you insert a nickel, 10 cents when you insert a dime, and 25 cents when you insert a quarter. The state changes from 5 cents to 10 cents when you insert a nickel and 15 cents when you insert a dime. However, if you insert a quarter when the state is 5 cents, the state doesn't change. The state changes from 10 cents to 15 cents when you insert a nickel and 20 cents when you insert a dime. However, if you insert a quarter when the state is 10 cents, the state doesn't change. The state changes from 15 cents to 20 cents when you insert a nickel and 25 cents when you insert a dime. However, if you insert a quarter when the state is 15 cents, the state doesn't change. The state changes from 20 cents to 25 cents when you insert a nickel. However, if you insert a dime or a quarter when the state is 20 cents, the state doesn't change. When the state is 25 cents and the OK button is pressed, the machine dispenses the drink and its state changes to 0 cents.*

During testing, you should check if the state of the machine's display screen changes depending on its current state and the value of the coin inserted. For example, if the screen's current state is 15 cents and you insert a nickel, the state should change to 20 cents. Similarly, you should check if the screen's state changes accurately from other states provided in the state transition model.

However, if the total value is above 25 cents when you insert a coin, the state should not change and the coin should be returned. So if the screen's current state is 15 cents and you insert a quarter, the state should remain 15 cents and the quarter should be returned.

You should also check if the state changes to display 0 cents and a Drink Dispensed message when you press **OK** after inserting either

* five nickels
* two dimes and a nickel
* three nickels and a dime
* one quarter

Question

You're testing a word processor using a state transition model.  
  
You've created a new file and have clicked the **Save** button to open the Save dialog box. What should the state change to display if you click the **OK** button after specifying a file name?

*According to the state transition model, you can create a new file by clicking the New button. You can then edit the file. You can click the Close button on the edited file to return to the Home screen. Or you can click the Save button on the edited file to open the Save dialog box. You can then click OK to view an error message. Or you can enter a file name and click the OK button to return to the file being edited. When you get an error message, you can click the Close button to return to the Save dialog box. From the Home screen, you can also click the Open button to display the Open dialog box. You can then double-click a file to open it. You can click the Edit button to access the Edit file mode. You can then input text to edit the file and click the Save button to return the Edit file mode. You can return to the Home screen from the Edit file mode by clicking the Close button.*

**Options:**

1. The Home screen
2. An error message
3. The file being edited
4. The Open dialog box

Answer

***Option 1:****Incorrect. The state should change to display the Home screen when you close an open file.*

***Option 2:****Incorrect. The state should change to display an error message when you click the****OK****button without specifying a file name.*

***Option 3:****Correct. The state should change to display the file being edited when you click the****OK****button after specifying a file name.*

***Option 4:****Incorrect. The state should change to display the Open dialog box when you click the****Open****button on the Home screen.*

**Correct answer(s):**

3. The file being edited

**2. Summary**

State transition testing is used to test whether an application changes from one state to another accurately. To use state transition testing, you create a state transition model for the application. This model includes four components – states, events, transitions, and outcomes.  
  
You test an application using the state transition model by performing all possible events from all states to check if the application's state changes accurately or not.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_sftf_a04_it_enus/output/html/course_transcript.html#top)

Use Cases in Software Testing

Learning Objective

*After completing this topic, you should be able to*

* *recognize how to build use cases in software testing*

**1. Using use cases to test software**

It's a good practice to test software applications by operating them as end users. To do this, software testers usually create use cases. A use case is a scenario that contains a series of realistic tasks that end users will frequently perform when using the application.  
  
For a web browser, for example, you can create a use case scenario for a beginner who wants to open a web page and bookmark it. Similarly, you can create another use case scenario for an advanced user who wants to activate the pop-up blocker and open a web site that uses pop-ups.

In addition to tasks, use cases include the outcome expected from the system for each task. In the use case for the advanced user, for example, an expected outcome can be that a message appears informing the user that pop-ups for the currently open web site have been blocked.

Use cases should also include error handling methods, which are alternate outcomes for tasks not performed correctly. For example, in the use case for the beginner, you can specify that an error should be displayed when the **Enter** key is pressed without specifying a web address.

In use case modeling, an actor is any person, external computer system, or event that interacts with the solution through a use case. Actors should also be given unique names that describe the role they play in interactions with the system.  
  
Testers will monitor the outcome of interactions between various software components, helping to find integration errors caused by incorrect interactions between components. For example, if the browser asks the actor to enter the web page address when that actor tries to bookmark an already open web page, the interaction between the web page address and the Bookmark service is incorrect and the tester logs a bug.

You can use a table to build a use case. In this table, you can specify imperative tasks using appropriate verbs and nouns.  
  
For example, to create the beginner's use case for the web browser, you specify three tasks for the actor – Type a web address and press the **Enter** key, Click the **Add to Bookmarks** button, and Type a name for the bookmark and click the **Save** button. You enter each task in alternate rows.

You then write the successful outcomes expected from the system for each task – The web page opens, The Add Bookmark dialog box opens with the web address entered, and The Bookmark name is visible in the Bookmarks list. You enter each outcome in the row below its corresponding task.

The column that contains the tasks and their successful outcomes is known as the *Main Success Scenario*.

You then write the possible error handling methods in a new column.   
  
In this example, there are two possible error handling methods for the outcome of the first task – Prompts for an address if the **Enter** key is pressed without specifying a web address and Displays a Page Not Found error if the **Enter** key is pressed after specifying an incorrect web address. The outcome for the third task has one error handling method – Prompts for a name if the **Save** button is clicked without specifying a name in the Add Bookmark dialog box.

You can also use adverbs and adjectives in the outcome statements to describe how each outcome should occur.  
  
For example, you can write that the web page opens quickly in two seconds after the actor types a web address and presses the **Enter** key.

Question

You're building a use case for testing the installation process of an application. You've already specified the main success scenario. To install the application, the actor should click the **Next** button on the Welcome page, agree to the terms and conditions and specify an optional e-mail address on the Install page, and click the **Install** button. Which error handling method is valid for this use case?

*This use case table contains three tasks – Double-click the installation file, Click the Next button, and Specify an e-mail address, select the I agree to the terms and conditions checkbox, and click the Install button. These tasks have one outcome each – The Welcome page of the installation wizard opens, The Install page of the installation wizard opens, and The application is installed.*

**Options:**

1. Displays an error if the **Install** button is clicked without specifying an e-mail address
2. Displays an error if the **Install** button is clicked after agreeing to the terms and conditions
3. Displays an error if the **Install** button is clicked after agreeing the terms and conditions and specifying an e-mail address
4. Displays an error if the **Install** button is clicked without agreeing to the terms and conditions

Answer

***Option 1:****Incorrect. Specifying an e-mail address in this example is optional, so an error cannot be displayed if the****Install****button is clicked without specifying an e-mail address.*

***Option 2:****Incorrect. In this example, the application should be installed if the actor clicks the****Install****button after agreeing to the terms and conditions; an error cannot be displayed if the actor does this.*

***Option 3:****Incorrect. In this example, the application should be installed if the actor clicks the****Install****button after specifying an e-mail address and agreeing to the terms and conditions; an error should not be displayed if the actor does this.*

***Option 4:****Correct. A possible error handling method is that an error should be displayed if the actor clicks the****Install****button without agreeing to the terms and conditions.*

**Correct answer(s):**

4. Displays an error if the **Install** button is clicked without agreeing to the terms and conditions

Consider that you want to create a use case for the process of sending an e-mail message. To do so, you specify the

**tasks**

You specify two tasks in this example – Click the **Send** button and Click the **Sent Items** link.

**outcomes**

You specify two expected outcomes for the first task – Attempts to send message through the mail server and Receives an OK message from the mail server. For the second task, you specify one outcome – Displays message in Sent Items folder.

**error handling methods**

You specify one error handling method for the outcome of the first task – Prompts user for subject if the **Send** button is clicked when the Subject field is blank. You specify two error handling methods for the outcomes of the second task – If the server doesn't respond, displays an error message and tries sending the message again. And if the server doesn't respond a second time, stores message in the Outbox and displays an error message.

Consider another example where you want to create a use case for the process of saving a file in a word processor. To do so, you again specify the

**tasks**

You specify two tasks in this example – Click the **Save** button and Specify a location and click the **OK** button.

**outcomes**

For the first task, you specify the outcome. The Save dialog box opens. For the second task, you specify the outcome. The file is saved in the specified location.

**error handling methods**

You specify one error handling method for the outcome of the first task – Specifies a cannot save message if it is clicked and the file is open in read-only mode. You specify two error handling methods for the outcome of the second task – Prompts for alternate name if file already exists and Prompts for another location if the disk is full or unwritable.

After the use case testing has been completed, you can invite the real customer to test the changes to their software that have been implemented. This strategy is useful for complex business software that can have a large number of Use Cases. Using the real data and actual customers will allow you to implement User Acceptance Testing.

Question

You're building a use case for testing the process of accessing an account in an online banking application. You've already specified that the actor should be able to login if the **Login** button is clicked after specifying a valid user name and password. You now want to specify that an error should be displayed if the **Login** button is clicked after specifying an invalid user name or password. What should you do?

*This use table contains two tasks – Click the Login link and Specify a user name and password and click the Login button. These tasks have one outcome each – The Login page opens and Bank account details are displayed. The first task has the error handling method Display error if the browser version is not compatible, and the second task has the error handling method Prompt for user name and password if either of these two fields are blank when the Login button is clicked.*

**Options:**

1. Create an additional task
2. Create an additional outcome
3. Create an error handling method
4. Create a main success scenario

Answer

***Option 1:****Incorrect. In this example, you want add an alternative outcome for a task, so you must specify this alternative outcome as an error handling method.*

***Option 2:****Incorrect. You create outcomes for tasks that are performed correctly. In this example, you want to specify an alternative outcome for an incorrectly performed task, which should be included as an error handling method.*

***Option 3:****Correct. You want to specify an alternative outcome for an incorrectly performed task, so this alternative outcome should be included as an error handling method.*

***Option 4:****Incorrect. A main success scenario includes successful outcomes of tasks. In this example, you want to specify an alternative outcome for an incorrectly performed task, which should be included as an error handling method.*

**Correct answer(s):**

3. Create an error handling method

**2. Summary**

You can test software from the perspective of the end user by creating use cases. Use cases contain realistic tasks that might be performed by the user and their outcomes. These cases also include error handling methods, which are alternative outcomes for incorrectly performed tasks.  
  
You can build a use case as a table. Starting with the first column, you enter tasks in alternate rows and specify the outcomes below each task. You also specify the error handling methods in a separate column.  
  
Instead of using hypothetical scenarios, you can use real data in a use case. To do this, you can ask customers rather than actors to test a use case, especially for complex applications in which real rather than assumed behavior is crucial.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_sftf_a04_it_enus/output/html/course_transcript.html#top)

Creating Software Test Cases

Learning Objectives

*After completing this topic, you should be able to*

* *create a decision table and evaluate its effectiveness at designing software test cases*
* *create a state transition table and evaluate its effectiveness at designing software test cases*
* *create a use case and evaluate its effectiveness at designing software test cases*

**1. Creating software test cases**

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_sftf_a04_it_enus/output/html/course_transcript.html#top)

© 2016 Skillsoft Ireland Limited