Basic Specification-Based Techniques

A World of Difference

By **Son Pham**

July, 2012

Duration: 4 hours

www.globalcybersoft.com

Contents

Specification-Based Techniques

1. Equivalence Class testing
2. Boundary Values testing
3. Decision Table testing
4. State Transition testing
5. Use case-based testing
6. Pairwise testing

Practices/Exercises

Course Objectives

At the end of the course, you will be able to know how to...

- Develop generic test cases by using specification-based techniques

Overview

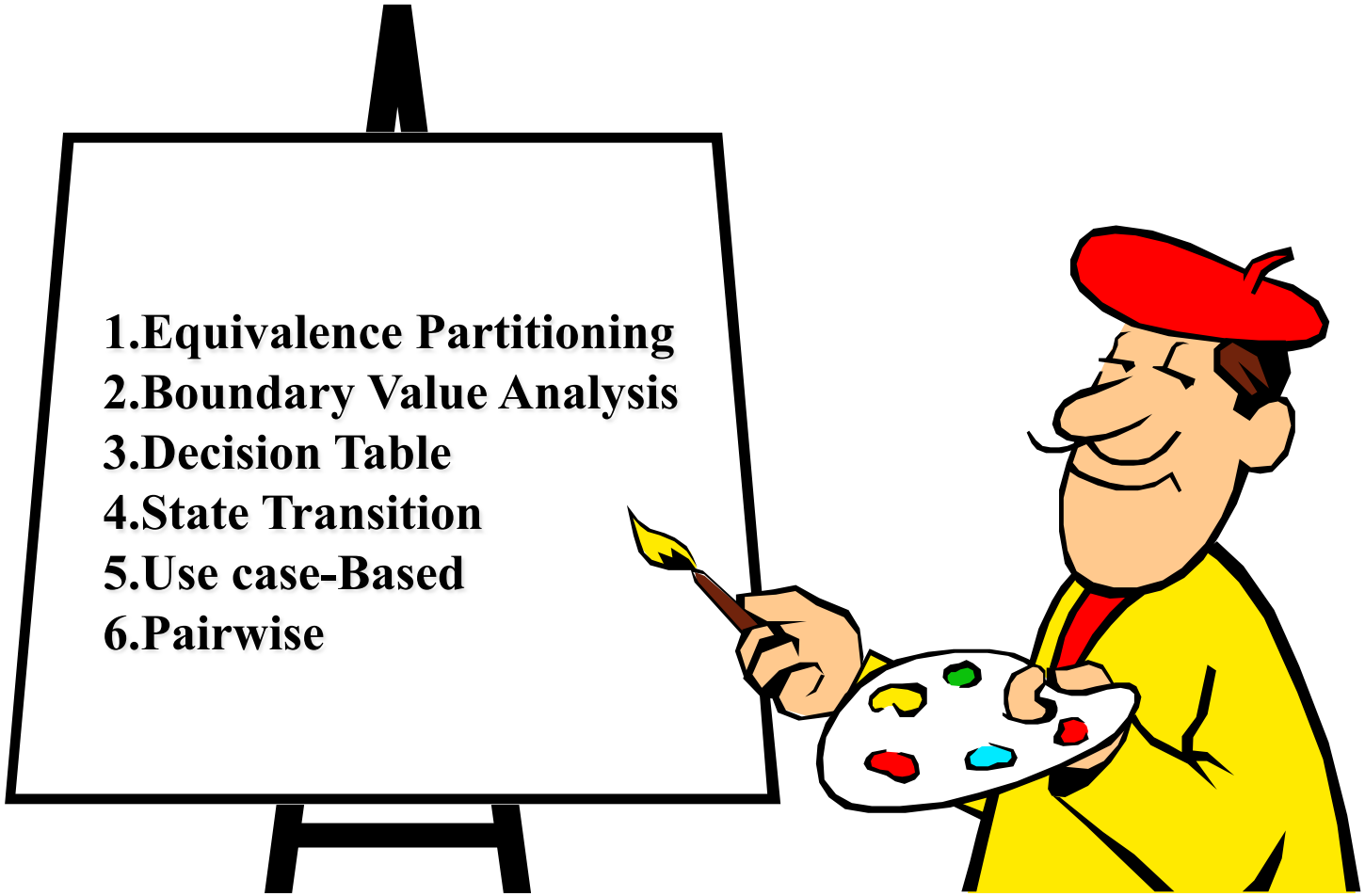
Specification-Based Techniques

- Overview
- Type of Techniques

Specification-Based testing (Dynamic Testing)

- **What is Specification-Based Testing?**
  - Testing, either functional or non-functional, without reference to the internal structure of the component or system. (also called behavioral testing, or Black-box testing)
- **Purpose of Specification-Based Testing?**
  - The main purpose is to verify functionality.
- **Specification-Based Techniques:**
  - Equivalence Partitioning
  - Boundary Value
  - Decision Table
  - State Transition
  - Use case-based
  - Pairwise

Specification-Based Techniques



Impossible to test everything

Equivalence Class Testing Technique

1. Equivalence Partitioning Technique

Definitions

Reasons to use

Steps to do

Example

Further questions

Summary

Definitions

- 1. What is an equivalence class?

  An equivalence class consists of a set of data that is treated the same by the module or that should produce the same result. Any data value within a class is *equivalent* to any other value.

- 2. What is Equivalence Class Testing?

  Equivalence class testing is a specification-based technique used to reduce the number of test cases to a manageable level by testing on equivalence classes.

Definitions - Key Point

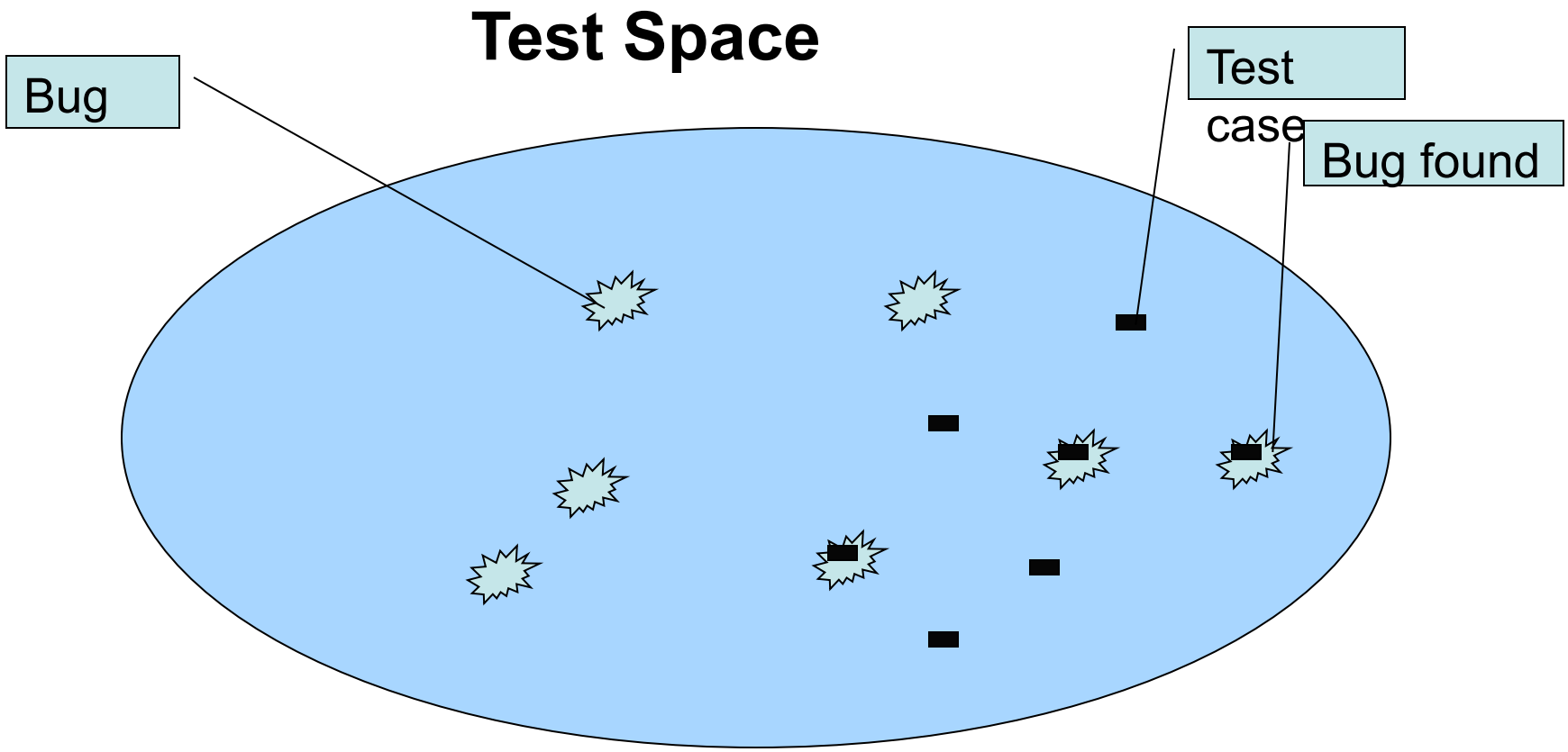
Key Point:

A group of tests forms an equivalence class if you believe that:

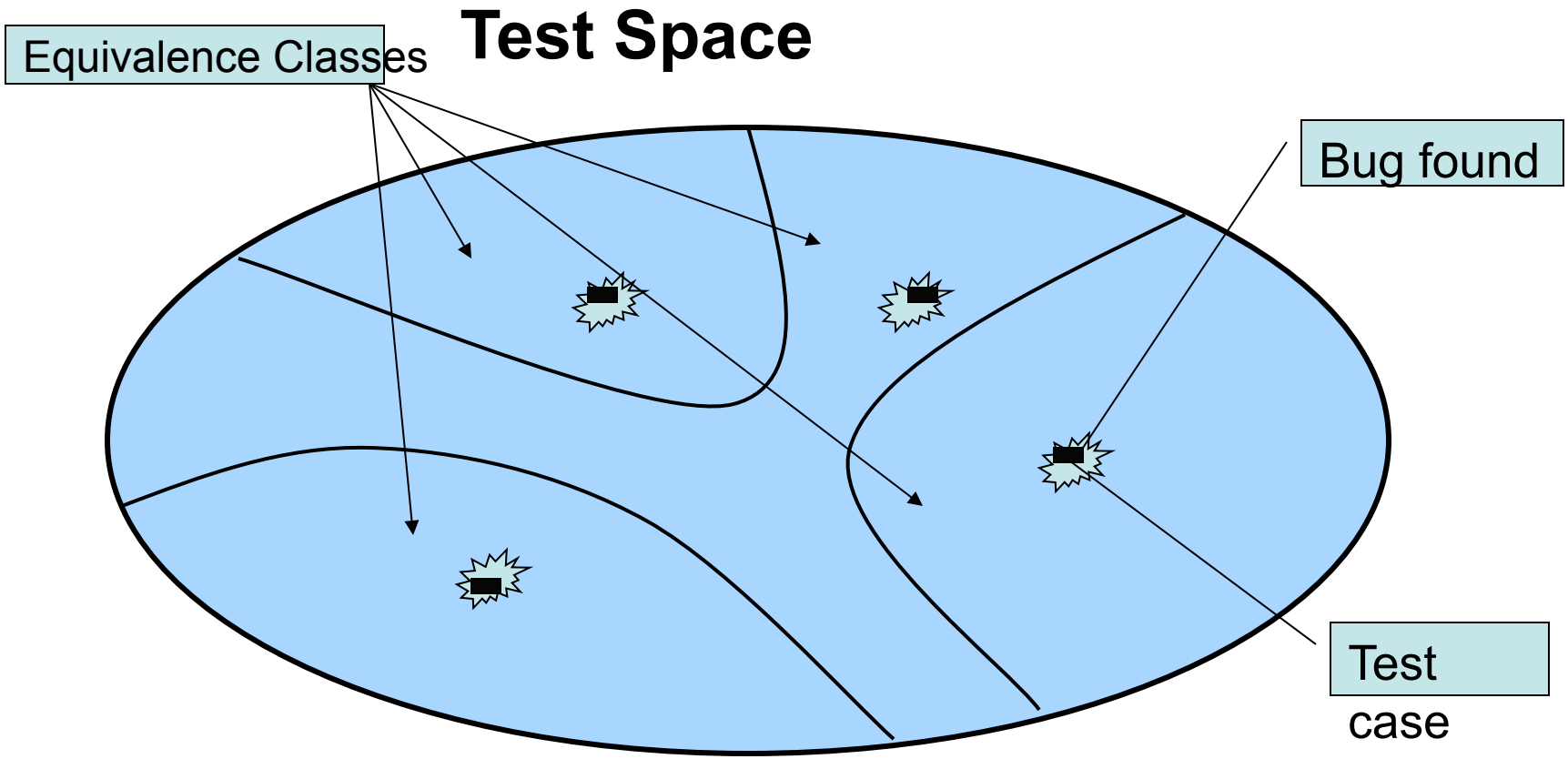
- They all test the same thing.
- If one test catches a bug, the others probably will too.
- If one test doesn't catch a bug, the others probably won't either.

Cem Kaner *Testing Computer Software*

Reasons to use Equivalence Class Technique



Reasons to use Equivalence Class Technique



Reasons to use Equivalence Class Technique

Why use Equivalence Class Testing?

There are at least three reasons why we should use Equivalence Class Testing:

1. It eliminates the need for exhaustive testing, which is not feasible.
2. It guides a tester in selecting a subset of test inputs with a high probability of detecting a defect.
3. It allows a tester to cover a larger domain of inputs/ outputs with a smaller subset selected from an equivalence class.

Steps to do

First, identify the equivalence classes.

Second, create a test case for each equivalence class.

Steps to do – Define an equivalence class

- How does the tester identify equivalence classes for the input domain?

- The input conditions usually come from a description in the specification of the software to be tested. The tester uses the conditions to partition the input domain into equivalence classes and then develops a set of tests cases to cover (include) all the classes.
- Different types of input require different types of equivalence classes.

Steps to do – Key Point

- If an input condition for the software-under-test is specified as a range of values, select one valid equivalence class that covers the allowed range and two invalid equivalence classes, one outside each end of the range.

- If an input condition for the software-under-test is specified as a *set* of valid input values, then select one valid equivalence class that contains all the members of the set and one invalid equivalence class for any value outside the set.

Example

Example:

You have to test a function which will print out the season's names of the input months. If you enter the number:

- From 1 to 6: (print out →) Dry season
- From 7 to 12 : Rainy season

How many test cases we have to execute without Equivalence Class Technique?

12 test cases

Example

- With Equivalence Class technique, we just have to execute 2 test cases:

Test Cases	Input	Expected Results
1	4	Dry season
2	10	Rainy season

Further Questions

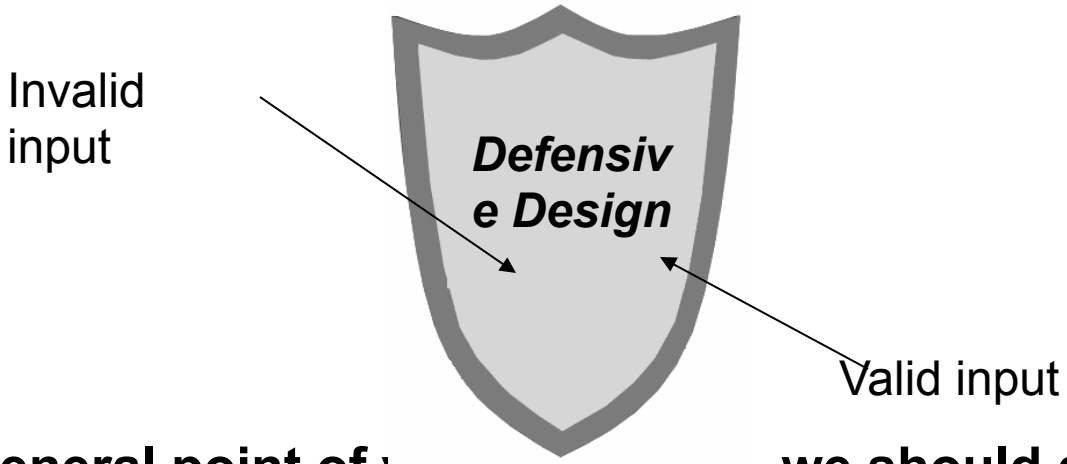
- The question now is that are those test cases enough?
- How about the cases of input values are -5, 128, ABC, #$%,… ?
- Do we have to test invalid input?

The answer is ‘No’ if we are doing **Testing-by-contract**.

The answer is ‘Yes’ if we are doing **Defensive-Testing**.

Further Questions

- **Testing-by-contract** is based on the design-by-contract philosophy. Its approach is to create test cases only for the situations in which the pre-conditions are met.
- **Defensive testing** is an approach that tests under both normal and abnormal pre-conditions. In this case the module is designed to accept any input (called defensive-design).



In the general point of view of testing, we should do defensive-testing.

Example – Summary

- So now in this example we must have at least 2 classes:
  - Valid class (includes valid values)
    - 2 sub classes including the range of values from “1 to 6” and “7 to 12”
  - Invalid class (includes invalid values)
    - Many sub classes including characters, decimal numbers, negative numbers….
- As we can see now, the number of test cases is reduced yet still covers the ability of finding bugs by using Equivalence Class Technique. But this can be strengthened by the use of another technique called Boundary Value.

Equivalence Class - Summary

1. Equivalence class partitions a test value in a particular class is equivalent to a test value of any other member of that class.

2. Equivalence Class Technique is most suited to systems in which much of the input data takes on values within ranges or within sets

3. It requires inputs or outputs can be partitioned based on the system's requirements.

4. It can be applicable at unit, integration, system and acceptant test levels.

5. Dual combinations (refer to Pairwise technique) instead of complete combinations (between the representative of one equivalence class with the one of other equivalence class).

6. Representatives of “invalid equivalence classes” should not be combined with representatives of other “invalid equivalence classes”. This will cause defect-masking.

Questions?

Boundary Value Testing Technique

2. Boundary Value Analysis Technique

Definitions

Reasons to use

Steps to do

Example

Summary

Definitions

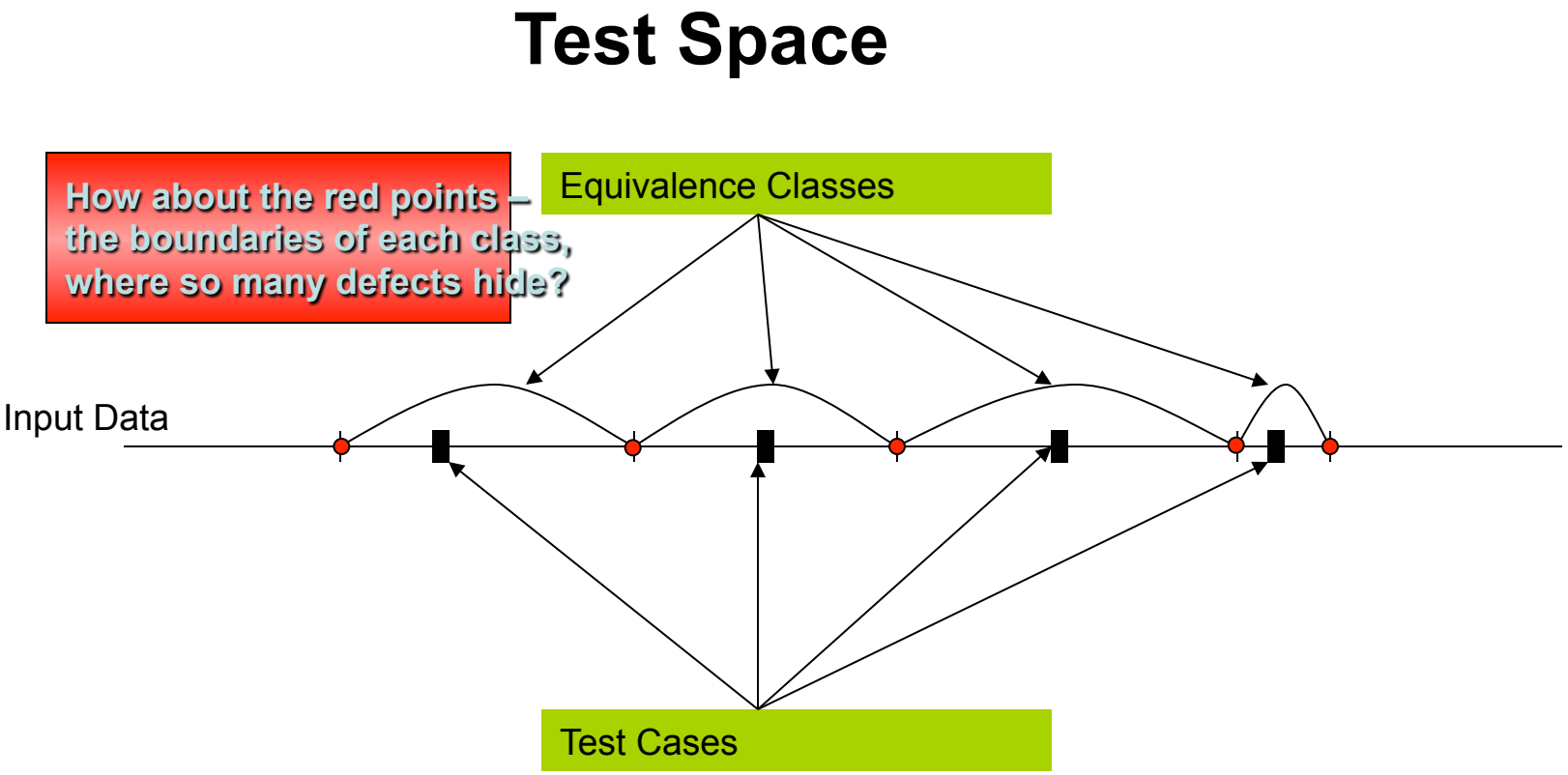
- 1. What is a boundary value?

  A boundary value is an element close to the edges, both the upper and lower edges of an equivalence class.

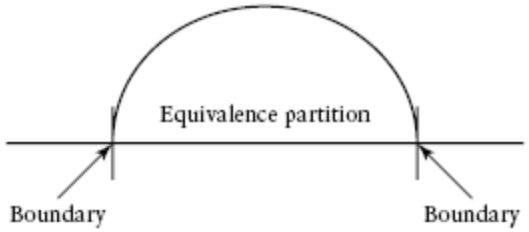
- 2. What is Boundary Value Testing?

  Boundary Value testing is the second key test design technique (after Equivalence Class technique) that focus testing effort on boundaries of each equivalence class.

Reasons to use Boundary Value Technique



Reasons to use Boundary Value Technique



❖ Boundary value testing focuses on the boundaries because that is where so many defects hide.

❖ Experienced testers have encountered this situation many times. Inexperienced testers may have an intuitive feel that mistakes will occur most often at the boundaries.

❖Boundary value testing is most suited to systems in which much of the input data takes on values within ranges or within sets.

Steps to do

First, identify the equivalence classes.

Second, identify the boundaries of each equivalence class.

Third, create test cases for each boundary value.

Steps to do – Key point

‘ ‘If an input condition for the software-under-test is specified as a range of values, develop valid test cases for the ends of the range, and invalid test cases for possibilities just above and below the ends of the range .’ ’

‘ ‘If the input or output of the software-under-test is an ordered set, such as a table or a linear list, develop tests that focus on the first and last elements of the set.”

Steps to do – Define boundary values

- How does the tester identify the boundary values for the input domain?

- **By choosing one point on the boundary, one point just below the boundary, and one point just above the boundary**.

Example

Example: we return to the example of Equivalence Class Testing

The function prints out the season's names of the input months.

If you enter the number:

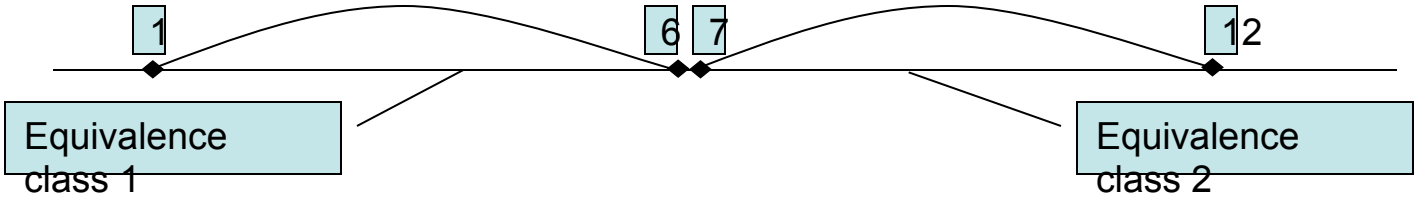
- From 1 to 6: (print out →) Dry season
- From 7 to 12 : Rainy season

Test cases (Using Equivalence Class Technique):

Test Cases	Input	Expected Results
1	4	Dry season
2	10	Rainy season

Example

Let's add more test cases by using Boundary value technique:



1. Input value is 1 (boundary of class 1)
2. Input value is 6 (boundary of class 1)
3. Input value is 0 (below boundary of class 1)
4. Input value is 7 (above boundary of class 1)
5. Input value is 12 (boundary of class 2). The case of 7 is removed because it is duplicated.
6. Input value is 13 (above boundary of class 2). The case of 6 is removed because it is duplicated.

Example

- Test cases designed by using Equivalence Class and Boundary Value technique.

Test Cases	Input	Expected Results
1	4	Dry season
2	10	Rainy season
3	1	Dry season
4	6	Dry season
5	0	*Error*
6	7	Rainy season
7	12	Rainy season
8	13	*Error*

- Note: we need more test cases for invalid input class.

Boundary Value Technique - Summary

1. Boundary Value Technique requires both the upper and lower boundaries of an equivalence class are covered by test cases.

2. Boundary Value Technique is most suited to systems in which much of the input data takes on values within ranges or within sets.

3. All it requires are inputs that can be partitioned and boundaries that can be identified based on the system's requirements .

4. It can be applicable at unit, system, integration and acceptant test levels.

Questions?

Decision Table Testing Technique

3. Decision Table Testing Technique

Definitions

Reasons to use

Steps to do

Example

Summary

Definitions

1. What is a Decision Table?

   A decision table includes rows listing **Cause** and **Effect** and columns listing possible combinations. Each combination (column) can be considered as a test case.

   Cause also considered as *Condition*.

   Effect also considered as *Action* or *Expected Results*.

2. What is Decision Table Testing Technique?

   Decision Table is one of the useful testing design techniques used to record complex rules/ flows. It also can serve as a guide to create test cases.

Reasons to use Decision Table Technique

Reasons to use Decision Table Testing Technique:

1. Decision Table is a useful black-box technique for scenario testing.
2. Not to miss any combinations when there are complex rules or flows.
3. Decision Table Technique is used as a guide to create test cases logically with more ability to find bugs.

Steps to do

1, Calculate the number of possible combinations.

2, List all causes in the decision table.

3, Fill columns with all possible combinations.

4, Edit and reduce test combinations.

(5, Add the checksum row)

6, Add effects to the table.

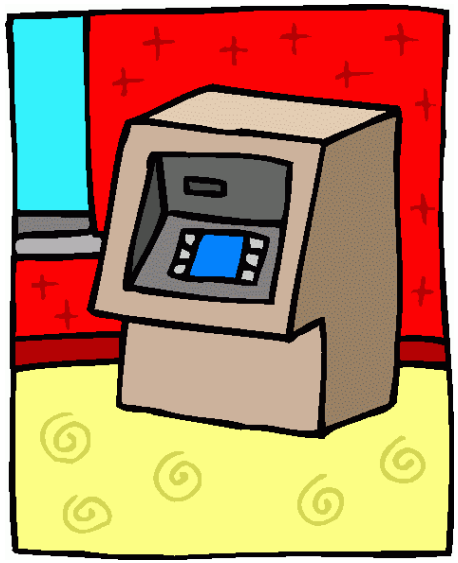
Example

Example 1

We are going to test the function to withdraw money from an ATM.

Here are the conditions for withdrawing money:

1. ATM Card is valid
2. PIN entered is correct (within 3 times)
3. Balance in the account & the machine is enough



Here are the Actions:

- Reject card (if condition 1 failed)
- Eat card (if condition 2 failed)
- Ask new withdrawal amount (if condition 3 failed)
- Pay money (if conditions 1,2 and 3 passed)

Example - Step 1 – Calculate the number of combinations

- To calculate the number of combinations, base on the **Condition/ Cause.**

Here are the Conditions:

1. *ATM Card is valid*
2. *PIN entered is correct (within 3 times)*
3. *Balance in the account & in the machine is enough*

In this example, we have 3 causes, each cause has 2 values (Yes/No). So the combinations should be:

$2^3 = 8$ combinations

OR: 2 x 2 x 2 = 8 combinations

Example - Step 2 – List all causes in Decision Table

The form of tables can be altered. But the following form is recommended.

- On the half-top of the table, list the **condition** or **cause.**

- On the half-bottom of the table, list the **effect** or **expected results.**

Conditions/ Causes	Value	1	2	3	4	5	6	7	8
Card is valid	Yes/No								
PIN entered is correct (within 3 times)	Yes/No								
Balance in the account & in machine is enough to withdraw	Yes/No								
Actions/ Effects									
Reject card	Yes/No								
Eat card	Yes/No								
Ask new withdrawal amount	Yes/No								
Pay money	Yes/No								

- In the column Value, list the values of each cause & effect. In this example, '*Yes*' for the case of conditions are met and reverse to '*No*'.

- Next columns (1- 8) are the combinations or test cases.

Example - Step 3 – Fill columns with all possible combinations

Conditions/ Causes	Value	1	2	3	4	5	6	7	8
Card is valid	Yes/No	Y	Y	Y	Y	N	N	N	N
PIN entered is correct (within 3 times)	Yes/No	Y	Y	N	N	Y	Y	N	N
Balance in the account & in machine is enough to withdraw	Yes/No	Y	N	Y	N	Y	N	Y	N
Actions/ Effects									
Reject card	Yes/No								
Eat card	Yes/No								
Ask new withdrawal amount	Yes/No								
Pay money	Yes/No								

Example - Step 4 – Edit & reduce test combinations

Conditions/ Causes	Value	1	2	3	4	5	6	7	8
Card is valid	Yes/No	Y	Y	Y	Y	N	N	N	N
PIN entered is correct (within 3 times)	Yes/No	Y	Y	N	N	-	-	-	-
Balance in the account & in machine is enough to withdraw	Yes/No	Y	N	-	-	-	-	-	-
Actions/ Effects									
Reject card	Yes/No								
Eat card	Yes/No								
Ask new withdrawal amount	Yes/No								
Pay money	Yes/No								

Replace invalid causes with '-'

Duplicated combinations

Example - Step 5 – Add the checksum row

Conditions/ Causes	Value	1	2	3	4	5	6	7	8
Card is valid	Yes/No	Y	Y	Y	Y	N	N	N	N
PIN entered is correct (within 3 times)	Yes/No	Y	Y	N	N	-	-	-	-
Balance in the account & in machine is enough to withdraw	Yes/No	Y	N	-	-	-	-	-	-
Actions/ Effects									
Reject card	Yes/No								
Eat card	Yes/No								
Ask new withdrawal amount	Yes/No								
Pay money	Yes/No								
Checksum		**1**	**1**	**2**		**4**			

Duplicated combinations will be deleted

Checksum number indicates the number of combinations covered.

Total of checksum number must be equal to the total number of combinations.

1 + 1 + 2 + 4 = 8 → This means 8 combinations are stilled covered.

Example - Step 6 – Add effects to the table

Conditions/ Causes	Value	1	2	3	4
Card is valid	Yes/No	Y	Y	Y	N
PIN entered is correct (within 3 times)	Yes/No	Y	Y	N	-
Balance in the account & in machine is enough to withdraw	Yes/No	Y	N	-	-
Effects/ Actions					
Reject card	Yes/No	N	N	N	Y
Eat card	Yes/No	N	N	Y	N
Ask new withdrawal amount	Yes/No	N	Y	N	N
Pay money	Yes/No	Y	N	N	N

This is the Decision Table for the example. Each combination should be executed as a test case.

For example: test case 1 (or combination 1) is when ATM card + PIN are valid + enough money in the account and in the ATM → then the ATM pays money.

Example

Example 2

A function calculates weekly salary for employees.

Here are the conditions/ causes:

1. Employee type (Project Manager, Team Lead, Team Member)
2. Working status (full time, part time)
3. Hours worked in week (40, >40, <40)

Here are the actions/ effects:

- Pay base salary (40 hours full time working)
- Calculate hourly wage (part time working) (only team members can work part time)
- Calculate overtime (> 40 hours full time working)
- Produce absence report (< 40 hours full time working)

Example

- In this example, we have 3 causes (2 causes have 3 values, 1 cause has 2 values). And the combinations should be:

$$3^2 \times 2^1 = 18 \text{ combinations}$$

OR: 3 x 3 x 2 = 18 combinations

- Using Decision Table technique, we will have a table listing 12 combinations but still covers all.

- Please do step 1 to 6, then see the result.

Example

Decision Table for calculating weekly salary function

Conditions/ Causes	**Values**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Employee Type	PM, TL, TM	PM	PM	PM	TL	TL	TL	TM	TM	TM	TM	TM	TM
Working status	Full time, Part time	Full	Full	Full	Full	Full	Full	Full	Full	Full	Part	Part	Part
Hours worked	40, >40, <40	40	>40	<40	40	>40	<40	40	>40	<40	40	>40	<40
Actions/ Effects													
Pay base salary	Yes/No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No
Calculate hourly wage	Yes/No	No	No	No	No	No	No	No	No	No	Yes	Yes	Yes
Calculate overtime	Yes/No	No	Yes	No	No	Yes	No	No	Yes	No	No	No	No
Produce absence report	Yes/No	No	No	Yes	No	No	Yes	No	No	Yes	No	No	No

Decision Table Technique - Summary

1. Decision Table is a very useful technique to design test cases logically.

2. Decision Table testing can be used whenever the system must implement complex rules/ flows when these rules/ flows can be represented as a combination of conditions and when these conditions have discrete actions associated with them.

3. It can be applicable at unit, integration, system and acceptant test levels.

4. The risk is that mistakes easily occur when filling information in Decision Table. So carefulness is a must to have trustable complete test cases.

Questions?

4. State Transition Testing Technique

Definitions

Reasons to use

Steps to do

Example

Summary

Definitions

1. What is a state?

   A state is a condition (attribute) in which a system or a component is waiting for one or more events.

2. What is a state transition?

Changes in the attributes (from a state to another) of an object caused by an event.

3. What is State Transition Testing?

A test case design technique in which test cases are designed to execute state transitions. State Transition testing uses State Diagram or State Table.

Reasons to use State Transition Technique

- State-Transition is an excellent testing technique to capture certain system requirements.
- State-Transition Technique directs our testing efforts by identifying the states, events, actions, and transitions that should be tested.

Steps to do

First, draw a transition diagram or design a transition table.

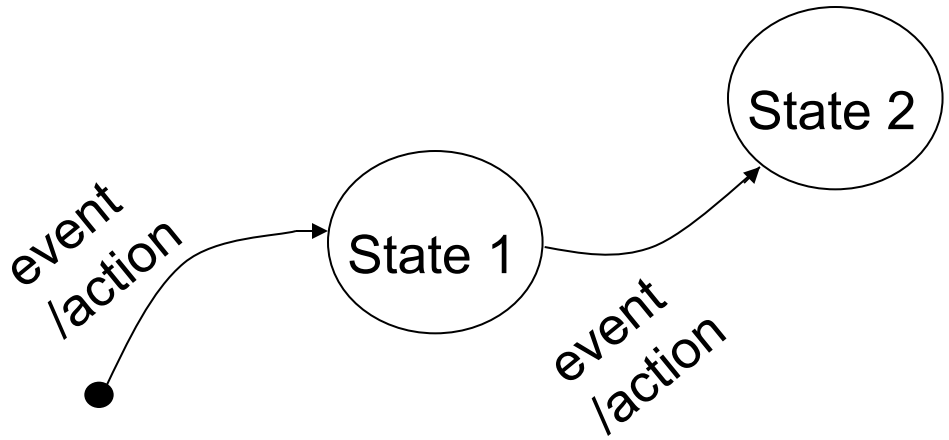
Second, design tests based on the transition diagram or the transition table.

Steps to do - Key point

‘ ‘The diagrams may be easier to comprehend, but state-transition tables may be easier to use in a complete and systematic manner.’ ’

‘ ‘The advantage of a state-transition table is that it lists all possible state-transition combinations, not just the valid ones. ”

Steps to do - How to draw a state-transition diagram?



❖ *An event* is something that causes the system to change state.

❖ *An action* is an operation initiated because of a state change. Note that action only occurs on transitions between states.

Steps to do - How to design a state-transition table?

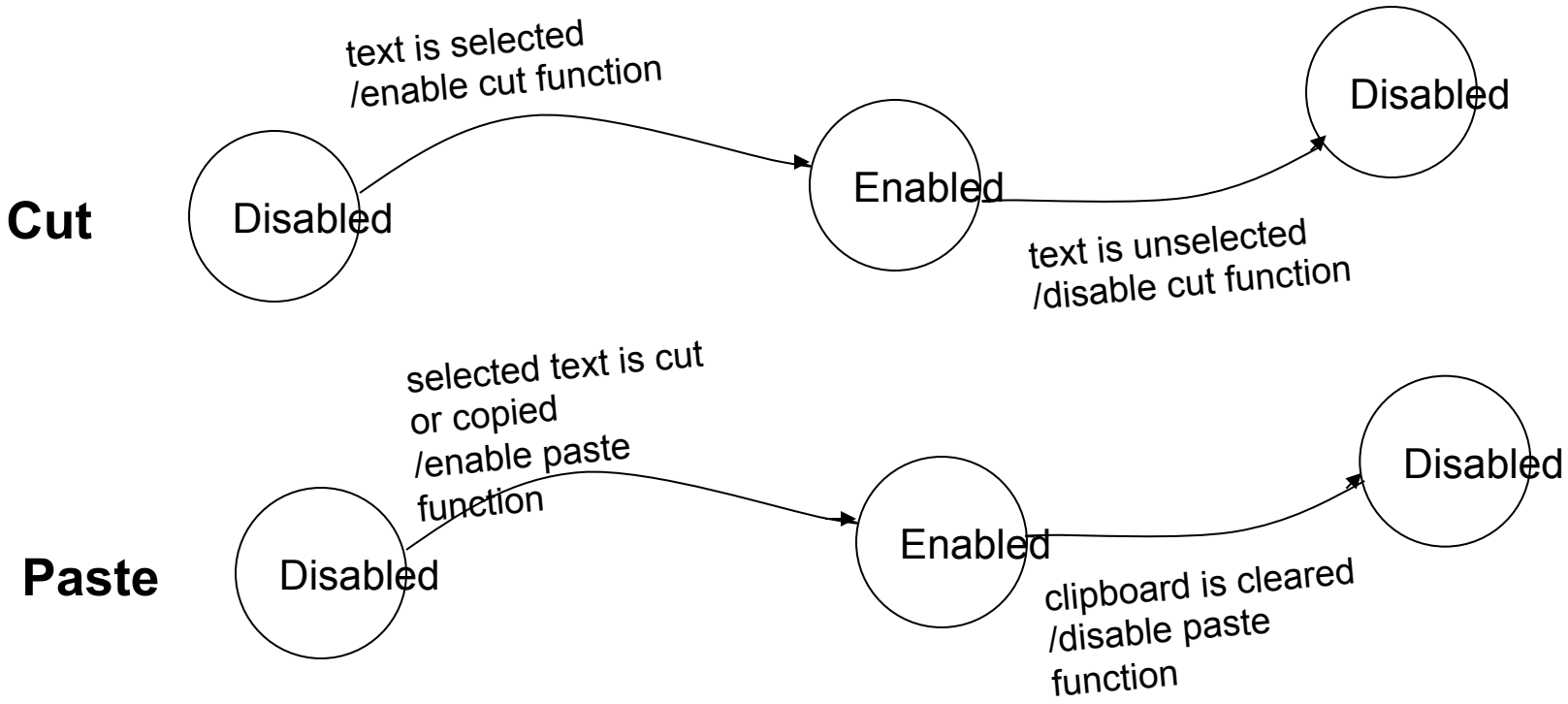
- The form of State-transition table can be altered.
- State-transition tables consist of four columns: *Component* (if needed) *Current State*, *Event*, *Action*, and *Next State*

Component	Current State	Event	Action	Next State
Component 1		Event	Action	State
Component 2	State	Event	Action	State

Example

Example 1: Test functions Cut/ Paste of notepad

State-transition Diagram



Example

Example 1:

State-transition Table

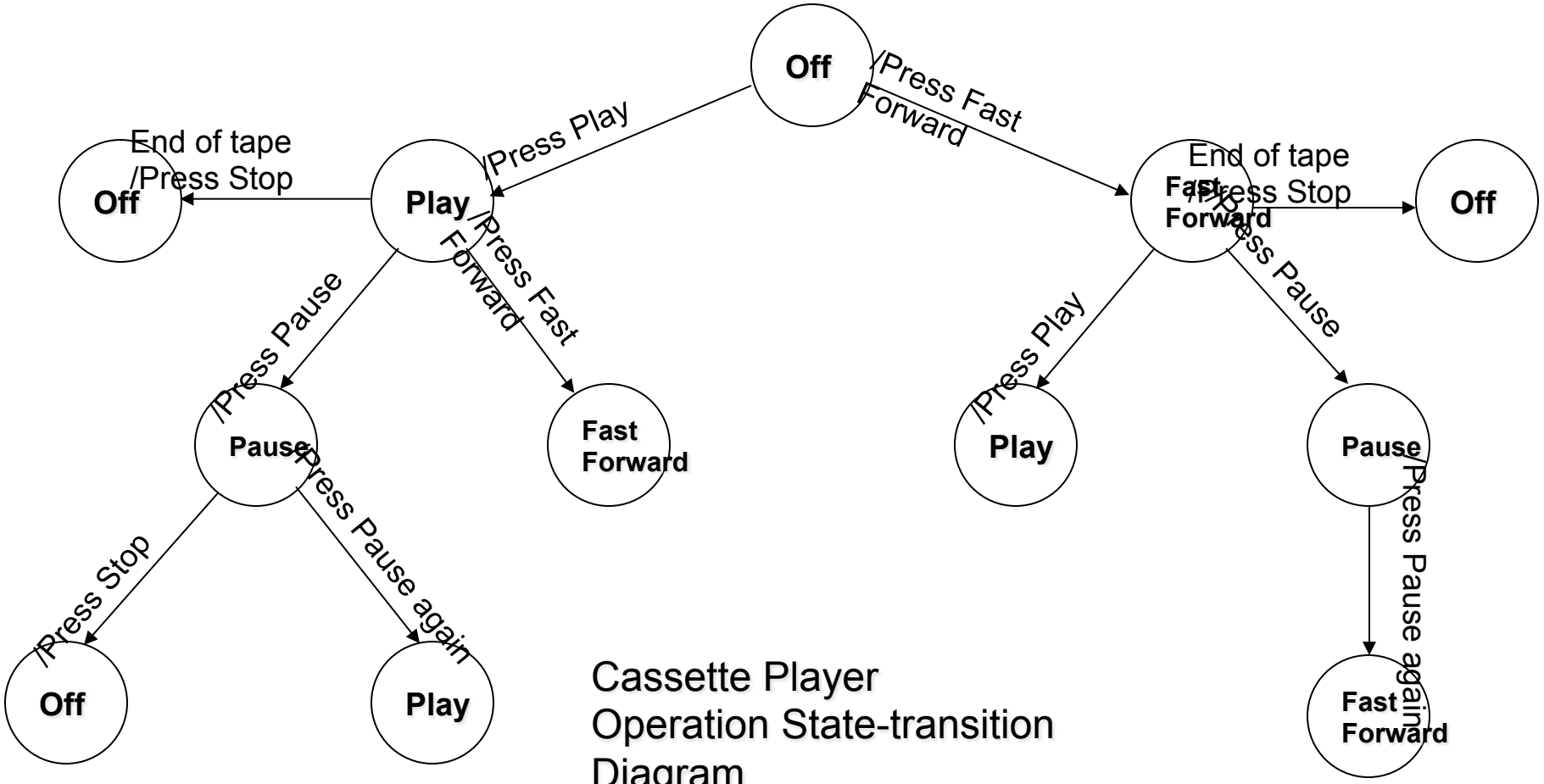
Function	Current state	Event	Action	Next state
Cut	Disabled	Text selected	Enable function	Enabled
	Enabled	Text un-selected	Disable function	Disabled
Paste	Disabled	Selected text is cut or copied	Enable function	Enabled
	Enabled	Clipboard is cleared	Disable function	Disabled

Example

Example 2: A tape player has four operations: **play**, **fast forward,** **pause** and **stop**.

- *Play*, *fast forward,* and pause are activated using the play, fast forward and pause button respectively. These operations can be canceled using the stop button.
- When in play mode, the fast forward and pause functions can be used by pressing each button.
- When in fast forward play mode, the play and pause functions can be used by pressing each button.
- When in Pause mode, press pause button again to return to the previous state (play mode or fast forward mode).

Example


Off
/Press Fast Forward
/Press Play
End of tape
/Press Stop
Off
Play
Fast Forward
End of tape
/Press Stop
Off
/Press Pause
/Press Fast Forward
/Press Play
/Press Pause
Pause
Fast Forward
Play
Pause
/Press Stop
/Press Pause again
/Press Pause again
Off
Play
Fast Forward
Cassette Player
Operation State-transition
Diagram

Example

Cassette Player Operation State-transition Table

Current state	Event	Action	Next State
Stop		Press Play button	Play
Stop		Press Fast Forward button	Fast Forward
Stop		Press Pause button	Stop
Play		Press Fast Forward button	Fast Forward
Play		Press Pause button	Pause (1)
Play		Press Stop button	Stop
Play	End of tape	Activate Stop button	Stop
Fast Forward		Press Play button	Play
Fast Forward		Press Pause button	Pause (2)
Fast Forward		Press Stop button	Stop
Fast Forward	End of tape	Activate Stop button	Stop
Pause (1)		Press Pause button again	Play
Pause (2)		Press Pause button again	Fast Forward
Pause		Press Stop button	Stop

State Transition Technique - Summary

1. State transition testing focuses on the testing of transitions from one state of an object to another state.
2. State transition testing is one of the most useful tools to test the operation of a device.
3. State-Transition Technique is best suitable to be applied *whenever* there is a state-transition of any component. And, of course, it is not applicable when the system has no state.

Questions?

5. Use Case Based Testing Technique

Definitions

Reasons to use

Steps to do

Example

Summary

Definitions

1. What is a use case?

   A use case is a sequence (scenario) of interactions between *an actor* and the system that accomplish some goal. Use case holds a wealth of information useful to testers.

2. What is an actor?

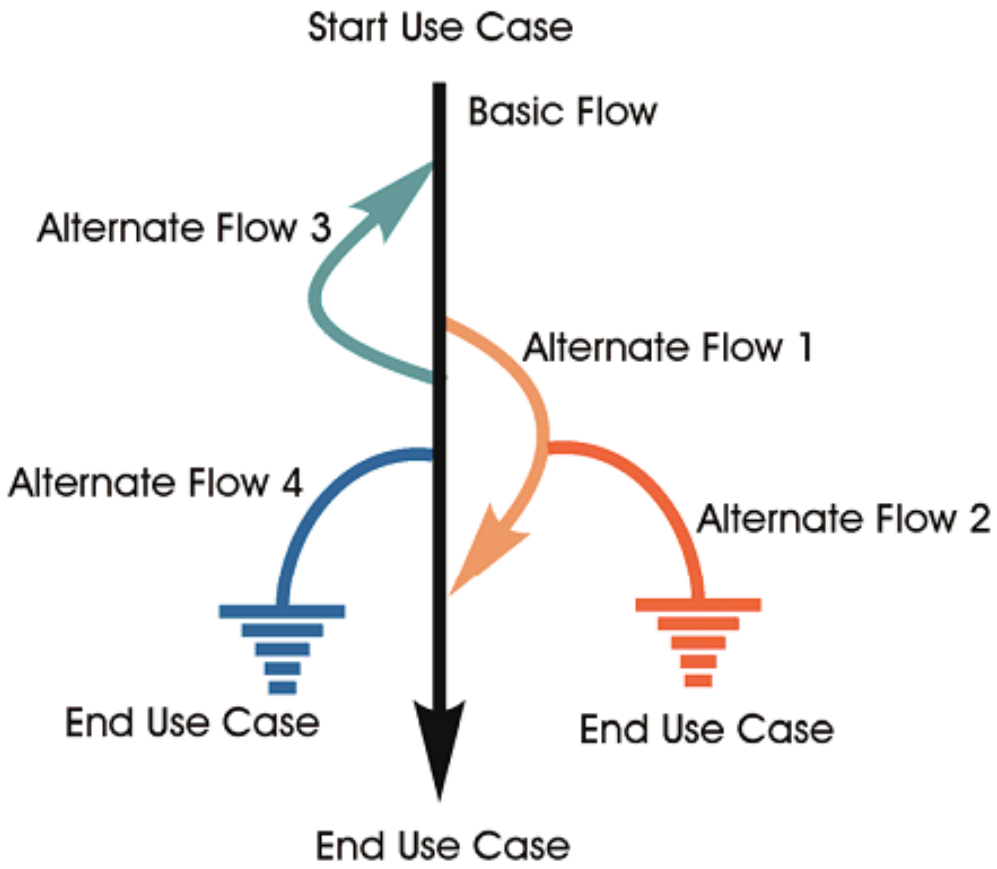
An actor is something or someone which interacts with the system under test. Actors may be end users, other systems, or hardware devices.

3. What is Use case based Testing (UCBT)?

A testing technique in which test cases are designed to execute user scenarios.

Definitions

A Use Case is based on flows of event (basic flows and alternative flows)



Reasons to use Use Case Based Technique

- To discover the objects that will construct a system to satisfy all functional requirements.
- To construct the scenarios that ensure the functionality can be supported.

Steps to do

(1, Draw a use case diagram – this will require UML knowledge and take more time.)
1, Design a use case table with basic & alternative flows.

2, Develop the table by adding more information for test cases execution.

Steps to do - Key point

‘ “To do UCBT, the tester needs to identify two things:

1. The actors involved in using the system,
2. The input, output, and system effects for the use cases.”

Example

Example: A simple mail application

Normal cases (Basic flows):

Use Case	Actor	Steps	Expected Result
Login	User	1. Type user name 2. Type password	Go to Inbox folder
Read new mails	User	1. Press check mail button 2. Click the new mail	1. List inbox mails 2. New mail is opened.
Reply mail	User	1. Press reply button 2. Enter send address 3. Enter mail title 4. Type the content 5. Press send button	1. A new empty mail opens. 5. "The mail has been sent" appears on the screen

Abnormal cases (Alternative flows):

Use Case	Actor	Steps	Expected Result
Login	User	1. Type the wrong user name 2. Leave password blank	"User invalid" appears on the screen
Login	User	1. Press correct user name 2. Type the wrong password	"Password is not correct" appears on the screen

And go on with normal and abnormal cases till the requirements are covered.

Use Case Based Technique - Summary

1. **Use Cases are used to specify the required functionality of an system.**

2. **UCBT is best used in the phase of *System test* or wherever the tester is interested in exploring behavior that flows through multiple use cases (scenarios).**

Questions?

State Transition Testing Technique

6. Pairwise Testing Technique

Definitions

Reasons to use

Steps to do

Examples

Using tools

Summary

Definitions

What is Pairwise Testing?

Pairwise Testing is a methodology that enables us to generate as few test cases as possible by pairing values of different variables.

Reasons to use Pairwise Technique

- Testing every combinations is impossible.
- Every combination of valid values of the two parameters will be covered by at least one test.
- Many faults are caused by the interactions between two parameters (Double-mode fault).

Steps to do

Pairwise Technique works on the rule of testing every unique pair of combination. *Any duplicated ones will be removed*.

Examples

Example 1: Test these simple combinations:

Parameter A has two values A1 and A2

Parameter B has two values B1 and B2

Parameter C has two values C1 and C2

How many test cases should be generated to cover all the combinations? 2 x 2 x 2 = 8 test cases

With Allpairs Technique, we just need to execute 4 test cases

Examples

Example 1:

Test cases with all combinations

Test cases	Parameter A	Parameter B	Parameter C
1	A1	B1	C1
2	A1	B1	C2
3	A1	B2	C1
4	A1	B2	C2
5	A2	B1	C1
6	A2	B1	C2
7	A2	B2	C1
8	A2	B2	C2

1. We don't need case 2 because:
[A1;B1] is duplicated in case 1, [A1;C2] is duplicated in case 4,
[B1;C2] is duplicated in case 6;
2. The same to cases 3, 5, 8

Examples

Example 1:

After removing duplicated pairs, we have 4 test cases left:

Test cases	Parameter A	Parameter B	Parameter C
1	A1	B1	C1
2	A1	B2	C2
3	A2	B1	C2
4	A2	B2	C1

Examples

Example 2: Test these another simple combinations:

Parameter A has three values A1, A2 and A3

Parameter B has three values B1, B2 and B3

Parameter C has three values C1, C2 and C3

- How many test cases should be generated to cover all the combinations? 3 x 3 x 3 = 27 test cases

- With Allpairs Technique, we just need to execute 9 test cases

Examples

Example 2:

Test cases	Parameter A	Parameter B	Parameter C
1			
2			
3			
4			
5			
6			
7			
8			
9			

Using tools

- But what will you do when you need to generate test cases for the large scale of combinations?

  Using tool for these cases is smarter than trying to design test cases by hand.

  Tools recommended:

1. **Allpairs** or **McDowell**
2. **Jenny**

Comparison of Efficiency

The number of test cases produced by different tools for the same model:

Parameter Sizes	AETG [1]	IPO [2]	TConfig [3]	CTS [4]	Jenny [5]	TestCover [6]	DDA [7]	AllPairs [McDowell] [5]	PICT
3^4	9	9	9	9	11	9	?	9	9
3^{13}	15	17	15	15	18	15	18	17	18
$4^{15}\ 3^{17}\ 2^{29}$	41	34	40	39	38	29	35	34	37
$4^1\ 3^{39}\ 2^{35}$	28	26	30	29	28	21	27	26	27
2^{100}	10	15	14	10	16	10	15	14	15
10^{20}	180	212	231	210	193	181	201	197	210

Using tools

- Jenny and Allpairs use different algorithms. Each tool will generate different results. So using both Jenny and Allpairs to have different test cases then we make decision to select the biggest one to have a full complete test set is a good experience.

- Reference for how to use and for other tools:
  http://www.pairwise.org/tools.asp

Pairwise Testing Technique - Summary

1. **The main idea of Pairwise Testing is that most bugs are found only when two variable values conflict, not when all conflict at the same time. This reduces significantly the number of test cases.**

2. **The problem is that Pairwise will not be effective if the defect requires three or more variables to be presented. We call this is Triple-mode fault and Multi-mode fault.**

Questions?

State Transition Testing Technique

7. Course Summary

Use testing techniques – Key Point

Equivalence Classes & Boundary Values

Table Decision vs Pairwise Testing

Use Case Based Testing & the others

Use Testing Techniques – Key Point

- You never find all your bugs by using each testing technique respectively.
- Never think that you have found enough bugs (for any software) or you got enough testing techniques for your career.
- Focus more testing effort on the weakness of the application.
- Know where and when to apply and combine the best suitable testing techniques.

Equivalence and Boundary Values

- Equivalence Classes and Boundary Values should always can be used in any cases whose much of the input data (values) are within ranges or sets and the boundaries of each can be determined based on requirements.

- Equivalence Classes and Boundary Values should always be combined to have as few test cases as possible that still cover a large scale of input.

- “Valid input” and “Invalid input” are two popular equivalence classes that you should have. Then divide each class into smaller equivalence classes.

Table Decision vs Pairwise Testing

- **Table Decision and Pairwise use tables and their methods seem to be the same. But their applications are very different:**

  - Table Decision should be used for the cases of rules or flows. That means this combination may be the pre-condition of another event (or function) to be happened.
    - Ex: For the ATM case. If the card and password is valid, then system will check the available money in the machine. If it is okay, functions money withdrawal will be activated… and so on…

  - Pairwise is best applied for the cases of "free" combinations. That means whatever the combinations you make, it won't cause any change in the way system processes.
    - Ex: you have a website offering life assurance to individuals, and user can enter the following personal information:
      - Sex (male/female)
      - Age (18-25, 26-35, 36-45, 46-55, 56+)
      - Smoker (no, under 10 a day, 10-20 a day, 20+)
      - Employed (yes/no)
      - Ever had an operation (yes/no)

Use Case Based Testing & the others

- Just ask yourself: If there are no Use cases, then how can you apply other techniques?
- Use Case Based may be the most popular testing technique we have ever used. It gives us the over view of a system or all the behaviors that system will do in both normal and abnormal cases. Based on that, we can applied other testing techniques for each case or flow.
- Use Case Diagram or Use Case Table is the best tool to make sure that we test the whole system and our testing effort is enough or not for the software requirements.

Questions?

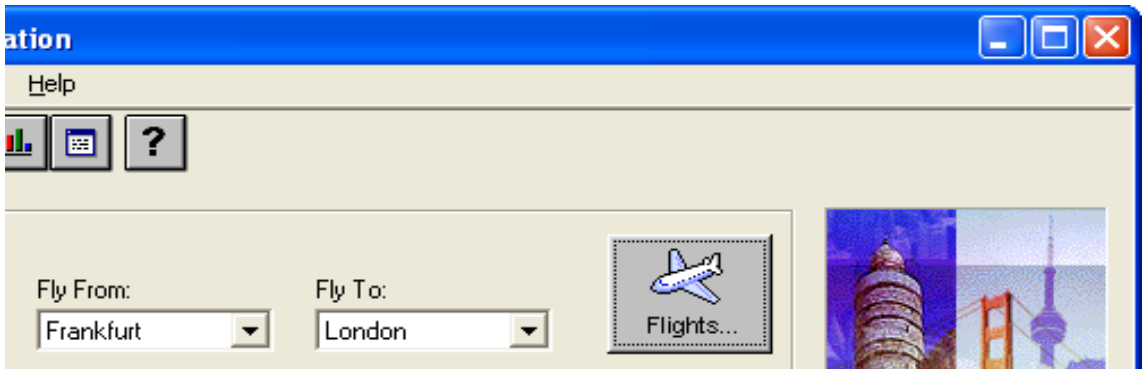
Practices

Practices

Example 1 – Analyze Requirement

Requirements as below:

1. The “Flights” button would function only if the values are selected from the dropdown lists.
2. “Fly From” dropdown list would have 3 values for instance Frankfurt, Colombo, Austin and a blank value.
3. “Fly To” dropdown list would have 3 values for instance Paris, London, Delhi and a blank value.



Example 1 – Define Test Data

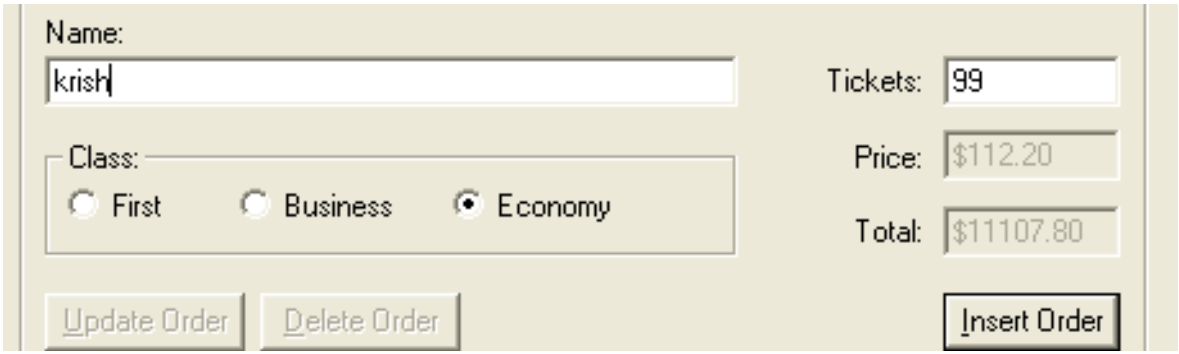
Test Data is analyzed as below:

No.	Short Description for data set	Fly From	Fly To	Action Pass/Fail	Comment
1	"Fly From" is blank, "Fly To" is blank	blank	blank	Fail	→ *Combination technique:* **1. "Fly From":** - Frankfurt - Colombo - Austin - Blank (no data) **2. "Fly To":** - Paris - London - Delhi - Blank (no data) => **All combinations** **= 4 * 4 = 16**
2	"Fly From" is blank, "Fly To" = "Paris"	blank	Paris	Fail	
3	"Fly From" is blank, "Fly To" = "London"	blank	London	Fail	
4	"Fly From" is blank, "Fly To" = "Delhi"	blank	Delhi	Fail	
5	"Fly From" = "Frankfurt", "Fly To" is blank	Frankfurt	blank	Fail	
6	"Fly From" = "Frankfurt", "Fly To" = "Paris"	Frankfurt	Paris	Pass	
7	"Fly From" = "Frankfurt", "Fly To" = "London"	Frankfurt	London	Pass	
8	"Fly From" = "Frankfurt", "Fly To" = "Delhi"	Frankfurt	Delhi	Pass	
9	"Fly From" = "Colombo", "Fly To" is blank	Colombo	blank	Fail	
10	"Fly From" = "Colombo", "Fly To" = "Paris"	Colombo	Paris	Fail	
11	"Fly From" = "Colombo", "Fly To" = "London"	Colombo	London	Pass	
12	"Fly From" = "Colombo", "Fly To" = "Delhi"	Colombo	Delhi	Pass	
13	"Fly From" = "Austin", "Fly To" is blank	Austin	blank	Fail	
14	"Fly From" = "Austin", "Fly To" = "Paris"	Austin	Paris	Pass	
15	"Fly From" = "Austin", "Fly To" = "London"	Austin	London	Pass	
16	"Fly From" = "Austin", "Fly To" = "London"	Austin	Delhi	Pass	

Example 2 – Analyze Requirement

Requirements as below:

1. Name is populated with the default value as “krish”.
2. “Insert Order” button would function only if the “Tickets” edit field is filled with a number and any one of the “Class” radio button is selected.
3. “Tickets” field (integer) would accept any value ranging from 1-99 only.



The test data that could be generated for this are as follows:

Note: “Name”: ignore to check → always using default value

Example 2 – Define Test Data (1)

No.	Short Description for data set	Tickets (field 1)	Class (field 2)	Action Pass/Fail	Comment
1	No data for Ticket with class is First	Blank	First	Fail: Error	***1. No data:*** Ticket = blank
2	No data for Ticket with class is Business	Blank	Business	Fail: Error	
3	No data for Ticket with class is Economy	Blank	Economy	Fail: Error	
4	Valid input data with	1	First	Pass	*2. Valid data:* - *Equivalence class Technique*: **EC1** = Ticket in [1..99] → Ticket = 50 - *BVA technique*: - Ticket = 1 - Ticket = 99
5	Valid input data with	1	Business	Pass	
6	Valid input data with	1	Economy	Pass	
7	Valid input data with	50	First	Pass	
8	Valid input data with	50	Business	Pass	
9	Valid input data with	50	Economy	Pass	
10	Valid input data with	99	First	Pass	
11	Valid input data with	99	Business	Pass	
12	Valid input data with	99	Economy	Pass	

Example 2 – Define Test Data (2)

No.	Short Description for data set	Tickets	Class	Action Pass/Fail	Comment
13	Invalid input data: Tickets is out of range	**0**	**First**	**Fail: Error**	*3. Invalid Data:* → Equivalence class & BVA technique *Invalid EC:* **iEC1** = (-∞, 1) → Ticket = 0 **iEC2** = (99, + ∞) → Ticket = 100
14	Invalid input data: Tickets is out of range	**0**	**Business**	**Fail: Error**	
15	Invalid input data: Tickets is out of range	**0**	**Economy**	**Fail: Error**	
16	Invalid input data: Tickets is out of range	**100**	**First**	**Fail: Error**	
17	Invalid input data: Tickets is out of range	**100**	**Business**	**Fail: Error**	
18	Invalid input data: Tickets is out of range	**100**	**Economy**	**Fail: Error**	
19	Invalid input data: Ticket is not a number	**two**	**First**	**Fail: Error**	Invalid Equivalence class: Ticket field is **n**ot **a** **n**umber **iEC3** = NaN
20	Invalid input data: Ticket is not a number	**Two**	**Business**	**Fail: Error**	
21	Invalid input data: Ticket is not a number	**Two**	**Economy**	**Fail: Error**	
22	Invalid input data: Ticket is not a number	**@#@#**	**First**	**Fail: Error**	
23	Invalid input data: Ticket is not a number	**@#@#**	**Business**	**Fail: Error**	
24	Invalid input data: Ticket is not a number	**@#@#**	**Economy**	**Fail: Error**	

Example 3 – Develop logical test cases

Using Use Case technique:

Basic Flow	This Use Case begins with the ATM in the Ready State. 1. Initiate Withdraw - Customer inserts bank card in the card reader on the ATM machine 2. Verify Bank Card - The ATM reads the account code from the magnetic strip on the bank card and checks if it is an acceptable bank card. 3. Enter PIN - The ATM asks for the customer's PIN code (4 digits) 4. Verify account code and PIN - The account code and PIN are verified to determine if the account is valid and if the PIN entered is the correct PIN for the account. For this flow, the account is a valid account and the PIN is the correct PIN associated with this account. 5. ATM Options - The ATM displays the different alternatives available at this ATM. In this flow, the bank customer always selects "Cash Withdraw." 6. Enter Amount - The ATM the amount to withdraw. For this flow the customer selects a pre-set amount ($10, $20, $50, or $100). 7. Authorization - The ATM initiates the verification process with the Banking System by sending the Card ID, PIN, Amount, and Account information as a transaction. For this flow, the Banking System is online and replies with the authorization to complete the cash withdrawal successfully and updates the account balance accordingly. 8. Dispense - The Money is dispensed. 9. Return Card - The Bank Card is returned. 10. Receipt - The receipt is printed and dispensed. The ATM also updates the internal log accordingly. Use Case ends with the ATM in the Ready State.

Example 3 – Develop logical test cases (2)

Alternate Flow 3 - Insufficient funds in ATM	At Basic Flow Step 6 - Enter Amount, if the ATM contains insufficient funds to dispense the requested amount, an appropriate message will be displayed, and rejoins the basic flow at Step 6 - Enter Amount.

Example 3 – Develop logical test cases (3)

The different use-case scenarios can be identified by beginning with the basic flow and then combining the basic flow with alternate flows:

Scenario 1	Basic Flow			
Scenario 2	Basic Flow	Alternate Flow 1		
Scenario 3	Basic Flow	Alternate Flow 1	Alternate Flow 2	
Scenario 4	Basic Flow	Alternate Flow 3		
Scenario 5	Basic Flow	Alternate Flow 3	Alternate Flow 1	
Scenario 6	Basic Flow	Alternate Flow 3	Alternate Flow 1	Alternate Flow 2
Scenario 7	Basic Flow	Alternate Flow 4		
Scenario 8	Basic Flow	Alternate Flow 3	Alternate Flow 4	

Deriving the test cases for each scenario is done by identifying the specific condition that will cause that specific use-case scenario to be executed.

Example 3 – Develop logical test cases (4)

- For instance: Scenario 4 with Alternative Flow 3, the test cases below are developed accordingly:

Test Case ID	Scenario	Condition	Expected Result
TC x	Scenario 4	Step 6 (Basic Flow) - Withdraw Amount > Account Balance	Rejoin basic flow at Step 6
TC y	Scenario 4	Step 6 (Basic Flow) - Withdraw Amount < Account Balance	Does not execute Alternate Flow 3, takes basic flow
TC z	Scenario 4	Step 6 (Basic Flow) - Withdraw Amount = Account Balance	Does not execute Alternate Flow 3, takes basic flow

Reference

- Refer to: ISTQB Glossary

Appendix: Course detail form

Author	Son Pham	**Duration**	4 hours
Category	Basic Specification-Based Techniques	**Type**	Theory and Practice

Examination	N/A
Intended Audience	Any QC
Pre-requisites	N/A
Completion criteria for the course	Attendee must join at least 90% course length
Criteria for granting training waivers	Those who has experience on Test Case Design area or ISTQB certification

Thank you

THANK YOU

Inquires regarding the above may be directed to:
Someone, Title, email@globalcybersoft.com