Software Quality Assurance

CHAPTER 6

SYSTEM TESTING-BLACK BOX

DR. JAMIL S. ALAGHA

Outline

Definition

Exhaustive Testing

Equivalence partitioning

Boundary value analysis

Cause Effect graphing

Pair Wise Testing

Scenario Testing

System Testing

Of the three levels of testing, the system level testing is closest to everyday experience, that is we evaluate a product with respect to our expectations.

Concerns with the app's **externals** and the goal in not to find faults but to demonstrate performance.

We tend to approach system testing from a functional standpoint rather than from a structural one.

System Testing

In system testing we are more interested in showing that the system, as a whole with all the "major" specifications – especially cross-component, ***is working.***

System Testing Objective: to ensure that the software does what the customer wants it to do.

System Function Testing: the system must perform functions specified in the requirements.

Other non-functional discussed in quality factors.

Black box testing

System Testing takes the whole system as black box.

Expected behavior is specified.

Hence just test for specified expected behavior

How it is implemented is not an issue.

Black Box testing

Specification for the black box is given

The expected behavior of the system is used to design test cases.

Test cases are determined solely from specification.

Internal structure of code **not** used for test case design.

System Test

Functional System Test:

- Exhaustive Testing
- **Equivalen**ce partitioning
- Boundary value analysis
- Cause Effect graphing
- Pair Wise Testing
- Scenario testing

No-Functional System Test:

- Usability Test
- Load Test
- Stress Test
- Compatibility Test

Outline

Definition

Exhaustive Testing

Equivalence partitioning

Boundary value analysis

Cause Effect graphing

Pair Wise Testing

Scenario Testing

Exhaustive Testing

To test a piece of code, such as a method or function, with every possible input to check if the code produces the expected output.

For most of the systems, complete testing is near impossible because The domain of possible inputs of a program is too large to be completely used in testing a system. There are both valid inputs and invalid inputs.

Exhaustive Testing

For example, assume you are testing a text box control that accepts a string variable of Unicode characters between upper case *A and Z with a minimum string length of 1 and a maximum string length of 25* characters.

Exhaustive testing would include each letter one time (26^1) and each letter combination for every possible string length.

So, to test for all possible inputs the number of tests is equal to 26^{25} + 26^{24} +….+ + 26^1

Equivalence partitioning

- *Equivalence class* is a subset of data that is representative of a larger class
- Divide input domain into equivalence classes and attempt to cover classes of errors .
- One test case per equivalence class, to reduce total number of test cases needed.

Outline

Definition

Exhaustive Testing

Equivalence partitioning

Boundary value analysis

Cause Effect graphing

Pair Wise Testing

Scenario Testing

Equivalence partitioning

Every condition specified as input is an equivalent class

Define invalid equivalent classes also

E.g. range 0< value<Max specified

- one range is the valid class
- input < 0 is an invalid class
- input > max is an invalid class

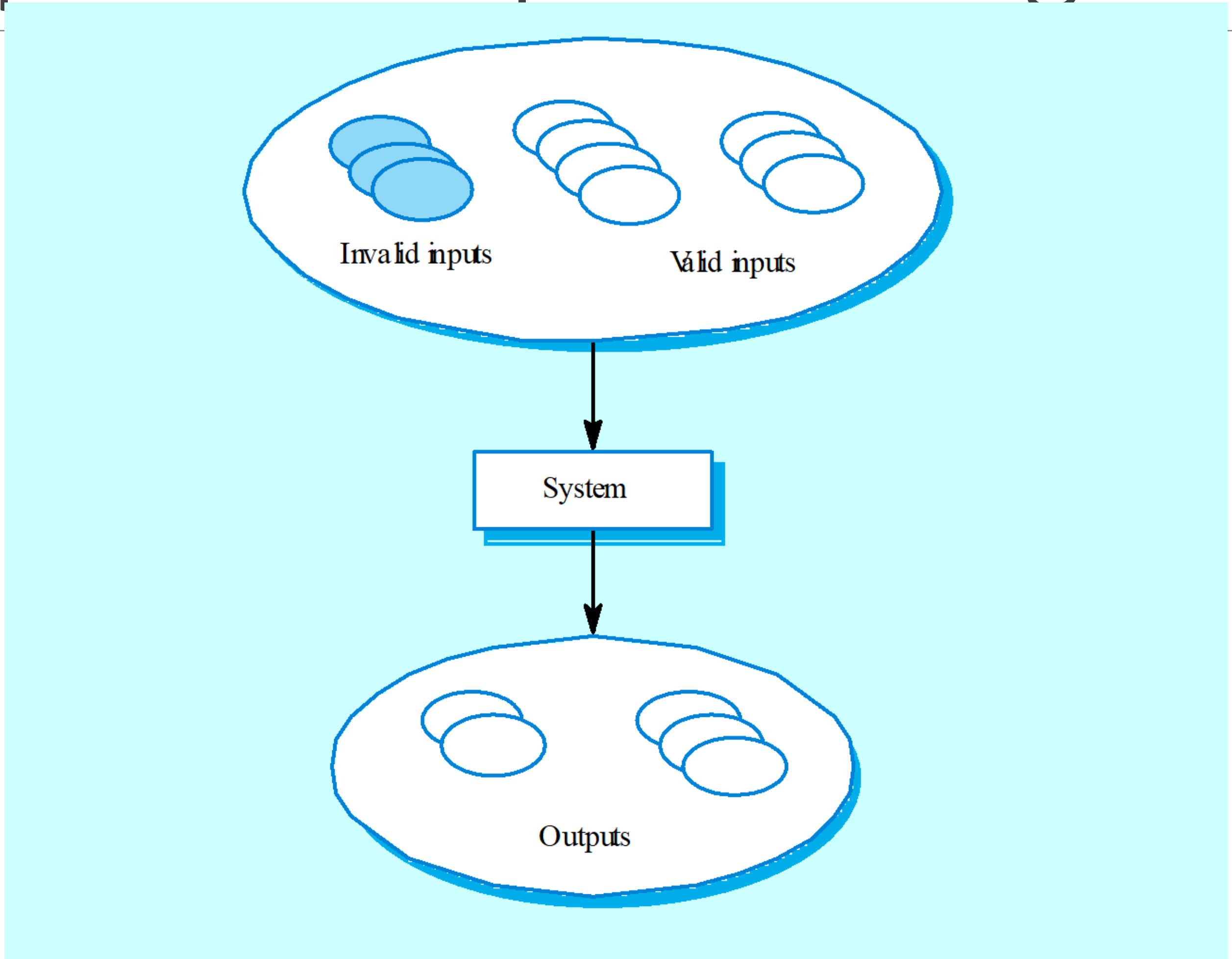
Whenever that entire range may not be treated uniformly - split into classes

Equivalence partitioning

Once Equivalence classes selected for each of the inputs, test cases have to be selected

- Select each test case covering as many valid equivalence classes as possible
- Or, have a test case that covers at most one valid class for each input
- Plus a separate test case for each invalid class

Equivalence partitioning



Equivalence partitioning

Example

A program which **accepts** credit limits with a given range Say,

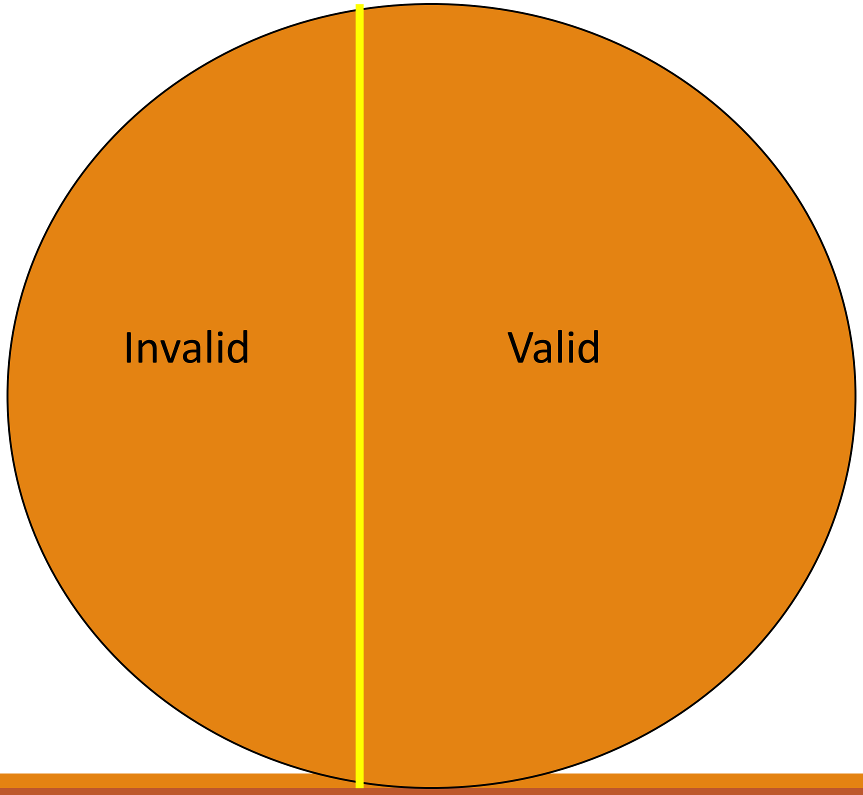
$10,000 – $15,000

This would have three equivalence classes:-

1. Less than $10,000 (Invalid)
2. Between $10,000 and $15,000 (Valid)
3. Greater than $15,000 (Invalid)

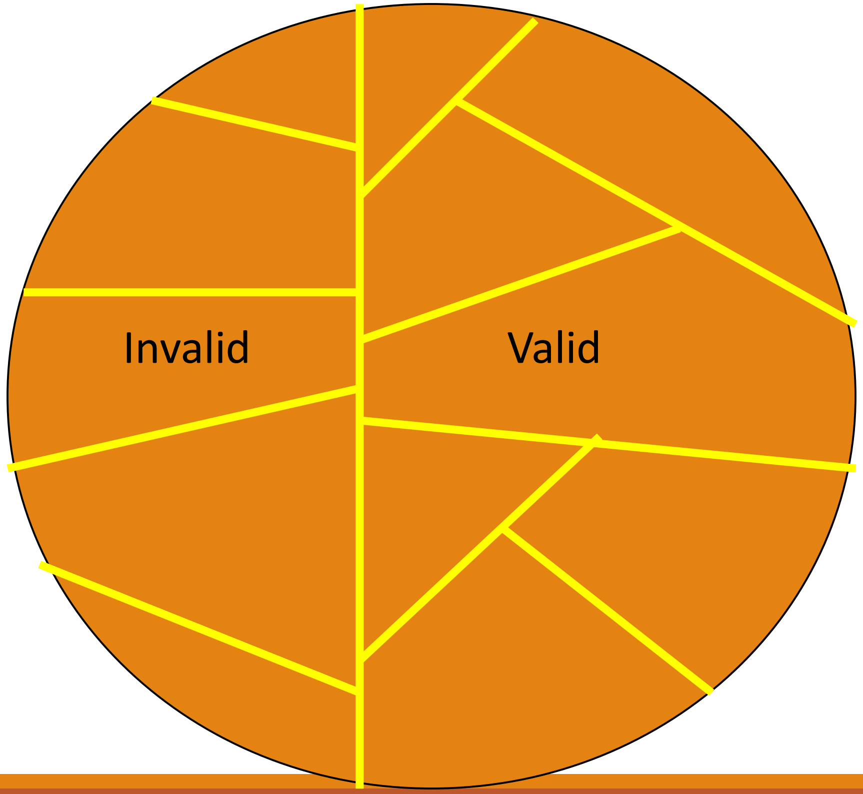
Equivalence Partitioning

First-level partitioning: Valid vs. Invalid values



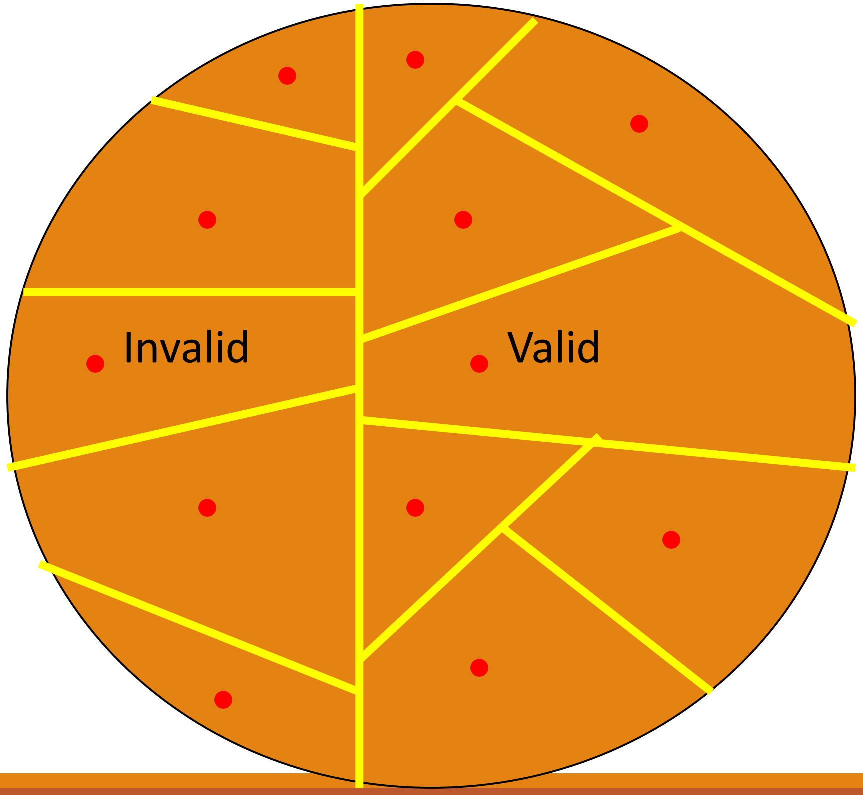
Equivalence Partitioning

Partition valid and invalid values into equivalence classes



Equivalence Partitioning

Create a test case for at least one value from each equivalence class



Equivalence Partitioning - examples

Input	Valid Equivalence Classes	Invalid Equivalence Classes
A integer N such that: -99 <= N <= 99	?	?
Phone Number Area code: [200, 999] Prefix: (200, 999] Suffix: Any 4 digits	?	?

Equivalence Partitioning - examples

Input	Valid Equivalence Classes	Invalid Equivalence Classes
A integer N such that: -99 <= N <= 99	[-99, -10] [-9, -1] 0 [1, 9] [10, 99]	?
Phone Number Area code: [200, 999] Prefix: (200, 999] Suffix: Any 4 digits	?	?

Equivalence Partitioning - examples

Input	Valid Equivalence Classes	Invalid Equivalence Classes
A integer N such that: -99 <= N <= 99	[-99, -10] [-9, -1] 0 [1, 9] [10, 99]	< -99 > 99 Malformed numbers {12-, 1-2-3, …} Non-numeric strings {junk, 1E2, $13} Empty value
Phone Number Area code: [200, 999] Prefix: (200, 999] Suffix: Any 4 digits	?	?

Equivalence Partitioning - examples

Input	Valid Equivalence Classes	Invalid Equivalence Classes
A integer N such that: -99 <= N <= 99	[-99, -10] [-9, -1] 0 [1, 9] [10, 99]	< -99 > 99 Malformed numbers {12-, 1-2-3, …} Non-numeric strings {junk, 1E2, $13} Empty value
Phone Number Area code: [200, 999] Prefix: (200, 999] Suffix: Any 4 digits	555-5555 (555)555-5555 555-555-5555 200 <= Area code <= 999 200 < Prefix <= 999	?

Equivalence Partitioning - examples

Input	Valid Equivalence Classes	Invalid Equivalence Classes
A integer N such that: -99 <= N <= 99	[-99, -10] [-9, -1] 0 [1, 9] [10, 99]	< -99 > 99 Malformed numbers {12-, 1-2-3, …} Non-numeric strings {junk, 1E2, $13} Empty value
Phone Number Area code: [200, 999] Prefix: (200, 999] Suffix: Any 4 digits	555-5555 (555)555-5555 555-555-5555 200 <= Area code <= 999 200 < Prefix <= 999	Area code < 200 Area code > 999 Area code with non-numeric characters *Similar for Prefix and Suffix* Invalid format 5555555, (555)(555)5555, etc.

Outline

Definition

Exhaustive Testing

Equivalence partitioning

Boundary value analysis

Cause Effect graphing

Pair Wise Testing

Scenario Testing

Boundary value analysis

Programs often fail on special values

These values often lie on boundary of equivalence classes

Test cases that have boundary values have *high yield*

These are also called *extreme cases*

A Boundary value test case is a set of input data that lies on the edge of a **Equivalen**ce class of input/output

Boundary value analysis

For each equivalence class

- choose values on the edges of the class
- choose values just outside the edges

E.g. if 0 <= x <= 1.0

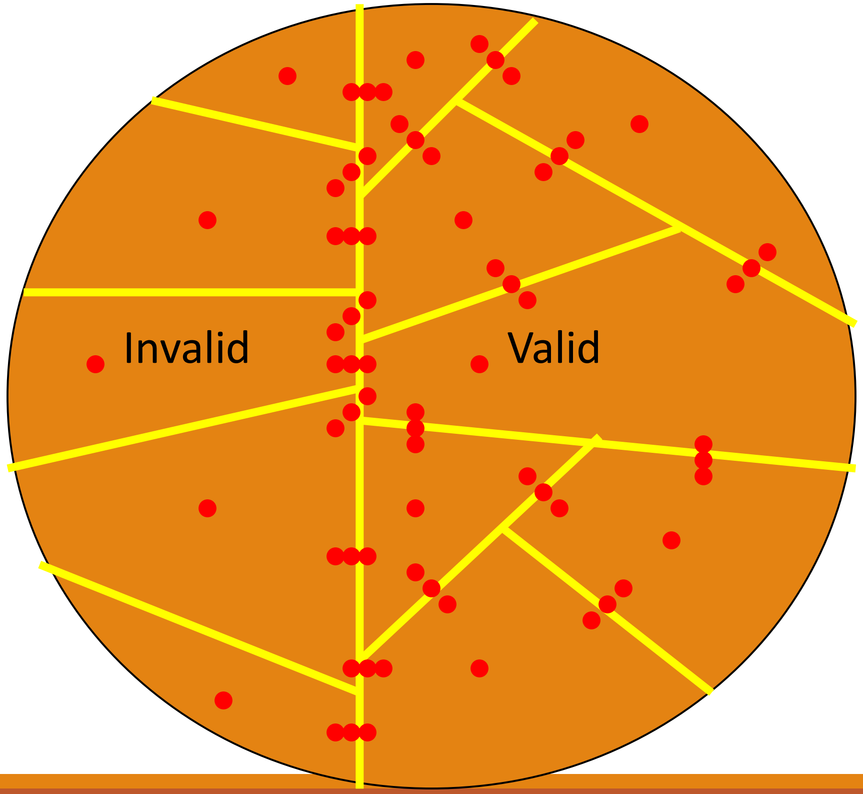
- 0.0 , 1.0 are edges inside
- -0.1,1.1 are just outside

E.g. a bounded list - have a null list , a maximum value list

Consider outputs also and have test cases generate outputs on the boundary

Boundary Value Analysis

Create test cases to test boundaries between equivalence classes



Boundary Value Analysis - examples

Input	Boundary Cases
A number N such that: -99 <= N <= 99	?
Phone Number Area code: [200, 999] Prefix: (200, 999] Suffix: Any 4 digits	?

Boundary Value Analysis - examples

Input	Boundary Cases
A number N such that: -99 <= N <= 99	-100, -99, -98 -10, -9 -1, 0, 1 9, 10 98, 99, 100
Phone Number Area code: [200, 999] Prefix: (200, 999] Suffix: Any 4 digits	?

Boundary Value Analysis - examples

Input	Boundary Cases
A number N such that: -99 <= N <= 99	-100, -99, -98 -10, -9 -1, 0, 1 9, 10 98, 99, 100
Phone Number Area code: [200, 999] Prefix: (200, 999] Suffix: Any 4 digits	Area code: 199, 200, 201 Area code: 998, 999, 1000 Prefix: 200, 199, 198 Prefix: 998, 999, 1000 Suffix: 3 digits, 5 digits

Outline

Definition

Exhaustive Testing

Equivalence partitioning

Boundary value analysis

Cause Effect graphing

Pair Wise Testing

Scenario Testing

Cause Effect graphing

- Equivalence classes and boundary value analysis consider each input separately.
- To handle multiple inputs, different combinations of equivalent classes of inputs can be tried.
- Number of combinations can be large – if n diff input conditions such that each condition is valid/invalid, total: 2^n
- Cause effect graphing helps in selecting combinations as input conditions

Cause Effect graphing

Identify causes and effects in the system

- Cause: distinct input condition which can be true or false
- Effect: distinct output condition (T/F)

Identify which causes can produce which effects; can combine causes

Causes/effects are nodes in the graph and arcs are drawn to capture dependency; and/or are allowed

Cause Effect graphing

From the Cause Effect graph, can make a decision table

- Lists combination of conditions that set different effects
- Together they check for various effects

Decision table can be used for forming the test cases

Cause Effect graphing

Cause & Effect Graphing is done through the following steps:

Step – 1: For a module, identify the input conditions (causes) and actions (effect).

Step – 2: Develop a cause-effect graph.

Step – 3: Transform cause-effect graph into a decision table.

Step – 4: Convert decision table rules to test cases. Each column of the decision table represents a test case.

Cause Effect graphing Example

Consider the following set of requirements as an example:

Requirements for Calculating Car Insurance Premiums:

R00101 For females less than 65 years of age, the premium is $500

R00102For males less than 25 years of age, the premium is $3000

R00103 For males between 25 and 64 years of age, the premium is $1000

R00104 For anyone 65 years of age or more, the premium is $1500

Cause Effect graphing Example

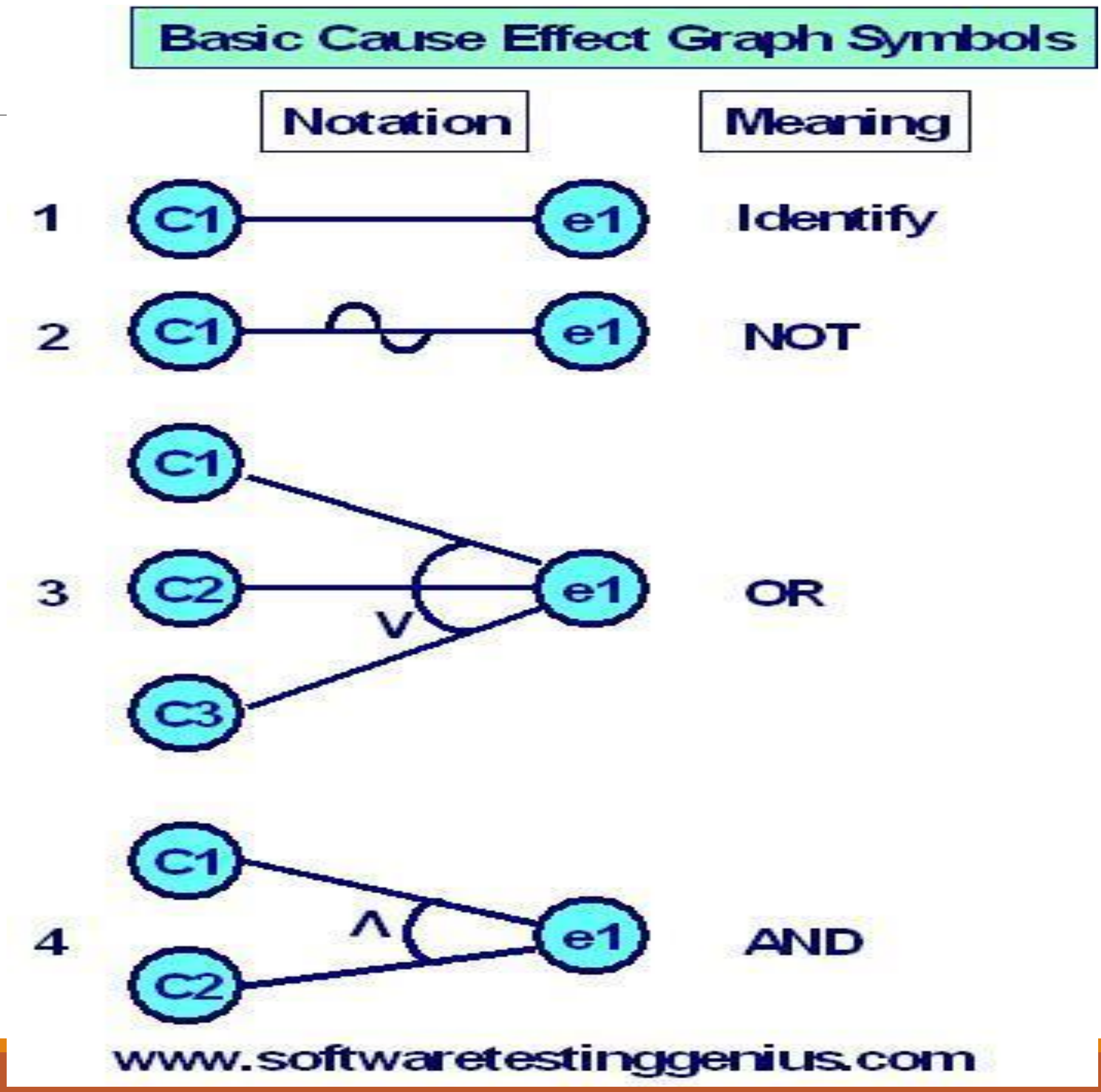
Step – 1: For a module, identify the input conditions (causes) and actions (effect).

Causes (input conditions)	Effects (output conditions)
1. Sex is Male	100. Premium is $1000
2. Sex is Female	101. Premium is $3000
3. Age is <25	102. Premium is $1500
4. Age is >=25 and < 65	103. Premium is $500
5. Age is >= 65	

Table 1 – Causes and Effects

As shown in Table 1 , each cause and each effect is assigned an arbitrary unique number as part of this process step.

Cause Effect graphing



Cause Effect graphing Example

Step – 2:
Develop a cause-effect graph.

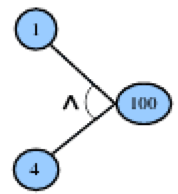
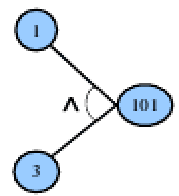
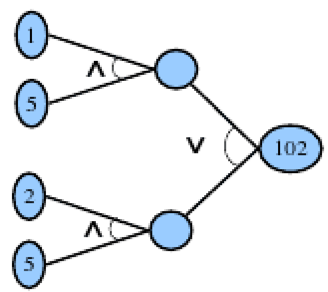
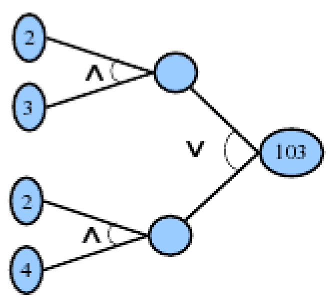
CEG	Interpretation
CEG #1: 	Causes: 1. Sex is Male and (∧) 4. Age is >=25 and < 65 Effect: 100: Premium is $1000
CEG #2: 	Causes: 1. Sex is Male and (∧) 3. Age is <25 Effect: 101: Premium is $3000
CEG #3: 	Causes: 1. Sex is Male and (∧) 5. Age is >= 65 or (∨) 2. Sex is Female and (∧) 5. Age is >= 65 Effect: 102: Premium is $1500
CEG #4: 	Causes: 2. Sex is Female and (∧) 3. Age is <25 or (∨) 2. Sex is Female and (∧) 4. Age is >=25 and < 65 Effect: 103: Premium is $500

Table 2 – Cause-Effect Graphs

Cause Effect graphing Example

Step – 3: Transform cause-effect graph into a decision table.

Test Case	1	2	3	4	5	6
Causes:						
1 (male)	1	1	1	0	0	0
2 (female)	0	0	0	1	1	1
3 (<25)	1	0	0	0	1	0
4 (>=25 and < 65)	0	1	0	0	0	1
5 (>= 65)	0	0	1	1	0	0
Effects:						
100 (Premium is $1000)	0	1	0	0	0	0
101 (Premium is $3000)	1	0	0	0	0	0
102 (Premium is $1500)	0	0	1	1	0	0
103 (Premium is $500)	0	0	0	0	1	1

Table 4 – Limited-Entry Decision Table

Cause Effect graphing Example

For example, the CEG #1, from Table 2 in step 3, converts into test case column 1 in the table below. From CEG #1, causes 1 and 3 being true result in effect 101 being true.

Some CEGs may result in more than one test case being created. For example, because of the one and only one constraint in the annotated CGE #3 from step 4, this CEG results in test cases 3 and 4 in the decision table above.

Cause Effect graphing Example

Step – 4: Convert decision table rules to test cases. Each column of the decision table represents a test case.

Test Case	1	2	3	4	5	6
Causes:						
1 (male)	1	1	1	0	0	0
2 (female)	0	0	0	1	1	1
3 (<25)	1	0	0	0	1	0
4 (>=25 and < 65)	0	1	0	0	0	1
5 (>= 65)	0	0	1	1	0	0
Effects:						
100 (Premium is $1000)	0	1	0	0	0	0
101 (Premium is $3000)	1	0	0	0	0	0
102 (Premium is $1500)	0	0	1	1	0	0
103 (Premium is $500)	0	0	0	0	1	1

Test Case #	Inputs (Causes)		Expected Output (Effects)
	Sex	Age	Premium
1	Male	<25	$3000
2	Male	>=25 and < 65	$1000
3	Male	>= 65	$1500
4	Female	>= 65	$1500
5	Female	<25	$500
6	Female	>=25 and < 65	$500

Table 5 – Test Cases

Cause effect graph, another example

- File management:
  - If the character of the first column is ‘**A**’ or ‘**B**’, and the second column is a **number**, then the file is considered updated.
  - If the first character is erroneous, print message X12
  - If the second column is not a number, print message X13

Cause effect graph, another example

Causes:

1 - first character is "A"

2 - first character is "B"

3 - second character is a digit

Effects:

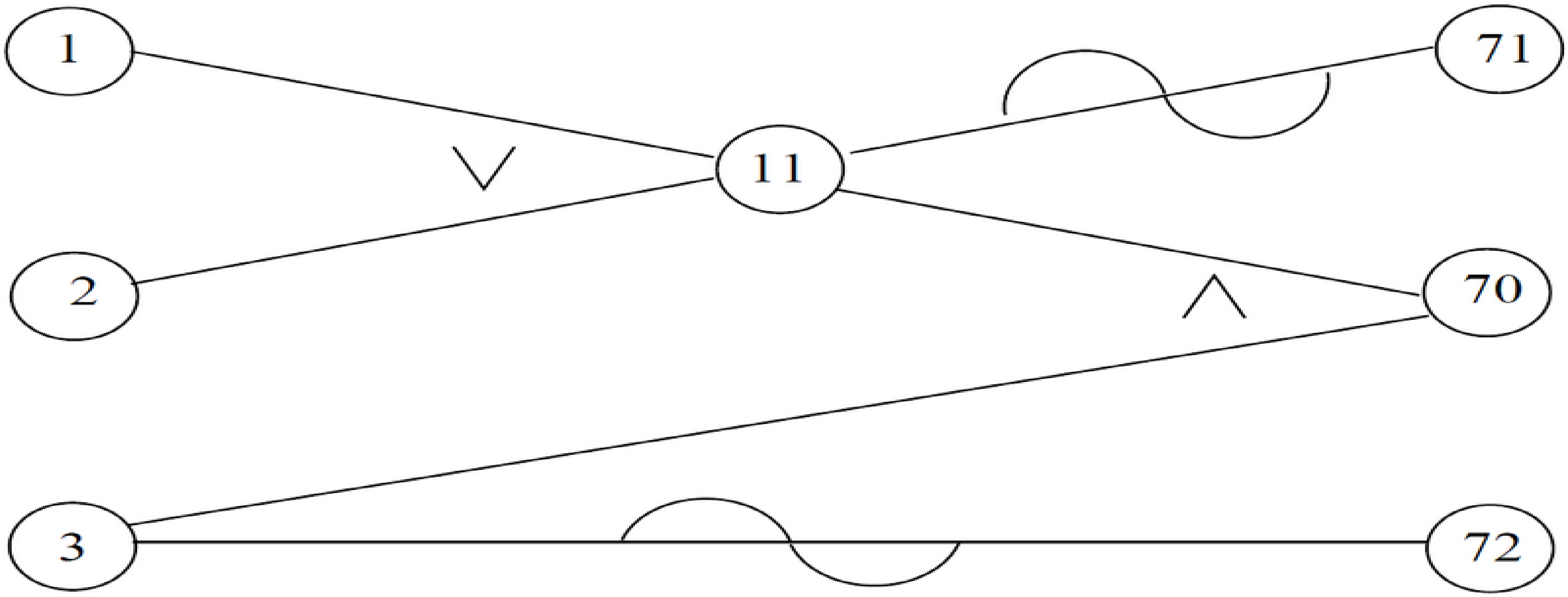
70 - the file is updated

71 - message X is print

72 - message Y is print

Cause effect graph, another example

- If the character of the first column is ‘**A**’ or ‘**B**’, and the second column is a **number**, then the file is considered updated.
- If the first character is erroneous, print message X12
- If the second column is not a number, print message X13



Cause effect graph, another example

Test Case	1	2	3	4	5
1	0	0	1	0	1
2	0	0	0	1	0
3	0	1	1	1	0
70	0	0	1	1	0
71	1	1	0	0	0
72	1	0	0	0	1

Cause effect graph, another example

Test Case	1	2	3	4	5
1	0	0	1	0	1
2	0	0	0	1	0
3	0	1	1	1	0
70	0	0	1	1	0
71	1	1	0	0	0
72	1	0	0	0	1

Test Case	1	2	3	O
1	M	N	L	X
2	M	N	1	X
3	A	N	1	U
4	M	B	12	U
5	A	N	L	Y

Outline

Definition

Exhaustive Testing

Equivalence partitioning

Boundary value analysis

Cause Effect graphing

Pair Wise Testing

Scenario Testing

Pairwise Testing

Pairwise (a.k.a. all-pairs) testing is an effective test case generation technique that is based on the observation that most faults are caused by interactions of at most two factors. Pairwise-generated test suites cover all combinations of two therefore are much smaller than exhaustive ones yet still very effective in finding defects.

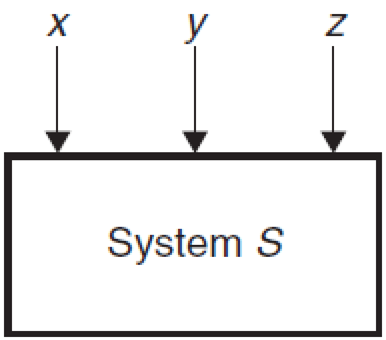
Pairwise Testing

- *Pairwise testing* – an approach to combinatorial testing that executes a pairwise test data set.
- Pairwise test data set - A set of test cases that covers all combinations of the *selected test data values* for every pair of a system's input variables.

Pairwise Testing

Consider the system *S which has three input variables X, Y,* and *Z. Let the notation D(w) denote the set of values for variable w. For the three given variables X, Y, and Z, their value sets are as follows:*

D(X) = {True, False}, D(Y) = {0, 5}, and D(Z) = {Q,R}.



Pairwise Testing

The total number of all-combination test cases is $2 \times 2 \times 2 = 8$. However, a subset of four test cases, covers all pairwise combinations.

Test Case ID	Input X	Input Y	Input Z
TC_1	True	0	Q
TC_2	True	5	R
TC_3	False	0	R
TC_4	False	5	Q

Pairwise Testing

Let us map the values as follows:

In the first column, let 1 = True*, 2 = False.*

In the second column, let 1 = 0, 2 = 5.

In the third column, let 1 = *Q, 2 = R.*

Pairwise Testing

The array has an interesting property: Choose any two columns at random and find all pairs (1,1), (1,2), (2,1), and (2,2); however, not all the combinations of 1's and 2's appear in the table. For example, (2,2,2) is a valid combination, but it is not in the table.

	Factors		
Runs	1	2	3
1	1	1	1
2	1	2	2
3	2	1	2
4	2	2	1

Pairwise Testing

Another Example:

Suppose you have an application that needs to be tested on several hardware and software configurations.

You are told that the application must be able to run on a Windows NT, Windows 2000 and Windows XP machine.

It also has to work on a system with a minimum of 128Meg and a max of 512Meg of RAM.

Finally, the application must also run on a Pentium II, III, and IV processor and work with Oracle, SQL and Access Databases.

Pairwise Testing

Operating System	RAM	PC Processor	Database
Windows NT	128 Meg	Pentium II	Oracle
Windows 2000	256 Meg	Pentium III	SQL
Windows XP	512 Meg	Pentium IV	Access

Combinations of Test Cases 3 x 3 x 3 x 3 = 81 Test Cases

Pairwise Testing

Test Case #	Operating System	RAM	PC Processor	Database
1	Windows NT	128 Meg	Pentium II	Oracle
2	Windows NT	256 Meg	Pentium III	SQL
3	Windows NT	512 Meg	Pentium IV	Access
4	Windows 2000	128 Meg	Pentium III	Access
5	Windows 2000	256 Meg	Pentium IV	Oracle
6	Windows 2000	512 Meg	Pentium II	SQL
7	Windows XP	128 Meg	Pentium IV	SQL
8	Windows XP	512 Meg	Pentium III	Oracle

Pairwise Testing

Definition

Given

Test Case ID	Input X	Input Y	Input Z
TC_1	True	0	Q
TC_2	True	5	R
TC_3	False	0	R
TC_4	False	5	Q

Pairwise Testing

This is an example of *$L_4(2^3)$ orthogonal array.*

The 4 indicates that the array has four rows, also known as *runs. The 2^3 part indicates that the array has three columns, known* as factors, and each cell in the array contains two different values, known as levels. *Levels mean the maximum number of values that a single factor can take on*

Pairwise Testing

Commonly Used Orthogonal Arrays

Orthogonal Array	Number of Runs	Maximum Number of Factors	Maximum Number of Columns at These Levels			
			2	3	4	5
L_4	4	3	3			
L_8	8	7	7			
L_9	9	4	—	4		
L_{12}	12	11	11			
L_{16}	16	15	15			
L'_{16}	16	5	—	—	5	
L_{18}	18	8	1	7		
L_{25}	25	6	—	—	—	6
L_{27}	27	13	—	13		

Pairwise Testing

In the following, the steps of a technique to generate orthogonal arrays.

Using the following example:

Web Example. Consider a website that is viewed on a number of browsers with

various plug-ins and operating systems (OSs) and through different connections

Variables	Values
Browser	Netscape, Internet Explorer (IE), Mozilla
Plug-in	Realplayer, Mediaplayer
OS	Windows, Linux, Macintosh
Connection	LAN, PPP, ISDN

Pairwise Testing

Step 1: Identify the maximum number of independent input variables with which a system will be tested. This will map to the *factors* *.*

Step 1: There are four independent variables, namely, Browser, Plug-in, OS, and Connection

Pairwise Testing

Step 2: Identify the maximum number of values that each independent variable

Step 2: Each variable can take at most three values.

Pairwise Testing

Step 3: **Find a suitable orthogonal array with the smallest number of runs** *$L_{Runs}(X^Y)$,*

Step 3: An orthogonal array ***$L_9(3^4)$ as shown in Table above is good enough for*** the purpose. The array has nine rows, three levels for the values, and four factors for the variables. The ***$L_9(3^4)$ constructed as follows:***

Pairwise Testing

TABLE 9.7 $L_9(3^4)$ Orthogonal Array

	Factors			
Runs	1	2	3	4
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

Pairwise Testing

Step 4: Map the variables to the factors and values of each variable to the levels on the array

Step 4: Map the variables to the factors and values to the levels of the array: the factor 1 to Browser, the factor 2 to Plug-in, the factor 3 to OS, and the factor 4 to Connection. Let 1 = Netscape, 2 = IE, and 3 = Mozilla in the Browser column. In the Plug-in column, let 1 = Realplayer and 3 = Mediaplayer. Let 1 = Windows, 2 = Linux, and 3 = Macintosh in the OS column. Let 1 = LAN, 2 = PPP, and 3 = ISDN in the Connection column.

Pairwise Testing

The mapping of the variables and the values onto the orthogonal array as follows:

Test Case ID	Browser	Plug-in	OS	Connection
TC_1	Netscape	Realplayer	Windows	LAN
TC_2	Netscape	2	Linux	PPP
TC_3	Netscape	Mediaplayer	Macintosh	ISDN
TC_4	IE	Realplayer	Linux	ISDN
TC_5	IE	2	Macintosh	LAN
TC_6	IE	Mediaplayer	Windows	PPP
TC_7	Mozilla	Realplayer	Macintosh	PPP
TC_8	Mozilla	2	Windows	ISDN
TC_9	Mozilla	Mediaplayer	Linux	LAN

Pairwise Testing

Step 5: Check for any "left-over" levels in the array that have not been mapped. Choose arbitrary valid values for those left-over levels.

Step 5: There are left-over levels in the array that are not being mapped. The factor 2 has three levels specified in the original array, but there are only two possible values for this variable. This has caused a level (2) to be left over for variable Plug-in after mapping the factors. One must provide a value in the cell.

Pairwise Testing

The choice of this value can be arbitrary, but to have a coverage, start at the top of the Plug-in column and cycle through the possible values when filling in the left-over levels

Test Case ID	Browser	Plug-in	OS	Connection
TC_1	Netscape	Realplayer	Windows	LAN
TC_2	Netscape	Realplayer	Linux	PPP
TC_3	Netscape	Mediaplayer	Macintosh	ISDN
TC_4	IE	Realplayer	Linux	ISDN
TC_5	IE	Mediaplayer	Macintosh	LAN
TC_6	IE	Mediaplayer	Windows	PPP
TC_7	Mozilla	Realplayer	Macintosh	PPP
TC_8	Mozilla	Realplayer	Windows	ISDN
TC_9	Mozilla	Mediaplayer	Linux	LAN

Outline

Definition

Exhaustive Testing

Equivalence partitioning

Boundary value analysis

Cause Effect graphing

Pair Wise Testing

Scenario Testing

Scenario testing

Scenario testing is a software testing activity that uses scenario tests, or simply scenarios, which are based on a hypothetical story to help a person think through a complex problem or system. They can be as simple as a diagram for a testing environment or they could be a description written in prose.

Scenario testing

These tests are usually different from test cases in that test cases are single steps and scenarios cover a number of steps.

scenarios can be used in concert for complete system testing.

Scenario testing

The ideal scenario has five key characteristics. It is (a) a story that is (b) motivating, (c) credible, (d) complex, and (e) easy to evaluate

Scenario testing

The test is *based on a story about how the program is used, including information about* the motivations of the people involved.

The story is *motivating.* To make the story more motivating, tell the reader *why it is important, why the user is* doing what he's doing, what he wants, and what are the consequences of failure to him.

Scenario testing

The story is *credible. It not only could happen in the real world; stakeholders would* believe that something like it probably *will happen.*

Sometimes you can establish credibility simply by referring to a requirements specification. In many projects, though, you won't have these specs or they won't cover your situation.

Scenario testing

The story involves a *complex use of the program or a complex environment or a complex set of data.*

A *complex story involves many features. You can create simplistic stories that involve only one* feature, but why bother? Other techniques, easy to apply to single features and more focused on developing power in these simple situations. The strength of the scenario is that it can help you discover problems in the relationships *among the features.*

Scenario testing

The test results are *easy to evaluate*. *This is valuable for all tests, but is especially* important for scenarios because they are complex

it should be easy to tell whether the program passed or failed. Of course, every test result should be easy to evaluate. However, the more complex the test, the more likely that the tester will accept a plausible-looking result as correct.

Scenario testing

Scenarios are great ways to capture realism in testing. They are much more complex than most other techniques and they focus on end-to-end experiences that users really have.

Scenario testing

A simple example of a scenario for usability test of the e-mail program:

You have just arrived at your desk after a short vacation. Check to see how many mail messages you have waiting for you. If there are any messages from Mr. Green, a Vice President of your company, read them.

Scenario testing

Another Example:

CARE Hospital uses a Hospital Management System (HMS) to manage patient activities with the hospital. The system includes features to manage outpatients as well as inpatients. An outpatient is someone who consults a doctor, is prescribed medication and leaves the hospital the same day. An inpatient gets admitted to the hospital. One of the features of HMS is "Patient History", which stores patients’ history, both for outpatients and inpatients. The HMS helps doctors in better diagnosis and treatment by providing them access to important patient information. Individual functions of the HMS application have been well tested and each feature works properly. And yet, under scenario testing, the HMS failed the following test.

Scenario testing

Here is the scenario it failed. Bunny, a middle-aged man goes to a hospital with pain in his arm. He is asked to wait for the relevant doctor and his details are entered as an outpatient. While waiting to see the doctor, he starts experiencing a severe pain in his chest. He is immediately admitted in the hospital and his status is changed to inpatient. In the test scenario, Bunny's history was initially entered as an outpatient, and later in the day his status was changed to inpatient. After making the entries in the HMS, the tester logged into the application as a doctor to review Bunny's Patient History – however, the history page for Bunny was blank. This led to the discovery of a bug that when Patient History is first entered as an outpatient, and then the patient status is changed to an inpatient the same day, the Patient History does not transfer over.

Scenario Test from Use Cases

The use case, is a scenario that describes the use of a system by an actor to accomplish work.

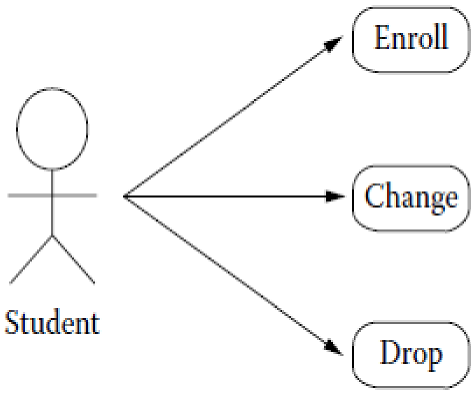
The following are the steps the tester can follow to create effective **Scenario** from use cases.

Scenario Test from Use Cases

Step 1: Draw a Use Case Diagram

Use cases can be represented visually with use case diagrams as shown in the Figure .

The ovals represent use cases, and the stick figures represent "actors," which can be either humans or other systems. The lines represent communication between an actor and a use case.



Scenario Test from Use Cases

Step 2: Write the Detailed Use Case Text

The details of each use case are then documented in text format. The next Table illustrates the “Enroll” use case details consisting of the normal and alternative flows.

Scenario Test from Use Cases

Use case ID	*Enroll_001*		
Use case name	Enroll a Student		
Created by	John Doe	Last updated by:	
Date created	3/15/2008	Date last updated:	
Actors	*Student*		
Description	Enroll a student into classes		
Trigger	Student wishes to enroll before the enrollment deadline		
Preconditions	Student has been accepted to the university Enrollment period has started		
Postconditions	Student's information has been validated and stored in the university enrollment system		

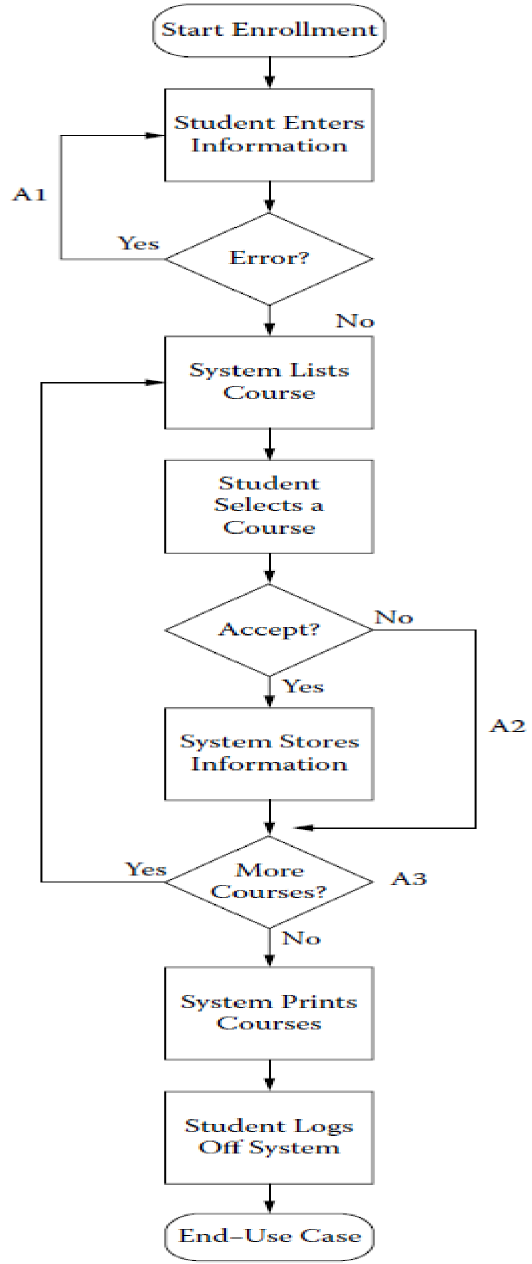
Scenario Test from Use Cases

Basic flow	Enrollment: Student enters his or her name Student enters his or her address Student enters his or her phone number Student enters his or her student number Student presses the “Submit” button Enrollment system lists the available courses from a drop-down list Student selects a course from a drop-down list and presses the “Accept” button The system stores the course information and asks the student if he or she wants to select another course The student selects “Yes,” and the enrollment process continues (Step 6) until all the courses have been selected and the student presses “No” All selected courses and schedule are printed out The student logs off the system

Scenario Test from Use Cases

Step 3: Identify Use Case Scenarios

A use case scenario is an instance of a use case, or a complete “path” through the use case. End users of a system can go down many paths as they execute the functionality specified in the use case. To illustrate this, the Figure below is a flowchart of the enrollment process. The basic (or normal) path is illustrated by the dotted lines.


Start Enrollment
Student Enters Information
A1
Yes
Error?
No
System Lists Course
Student Selects a Course
Accept?
No
Yes
System Stores Information
A2
Yes
More Courses?
A3
No
System Prints Courses
Student Logs Off System
End–Use Case

Scenario Test from Use Cases

The alternate paths (or exceptions) are depicted as A1 and A2. A1 is the case when an error occurs when the student is entering his or her information into the system. A2 depicts the case when the student has selected a particular course but then chooses not to accept it.

The Table below lists some possible combinations of scenarios for the Figure .

Starting with the basic flow combinations, alternative flows are added to define the scenarios.

These scenarios will be used as the basis for creating test cases.

Scenario Test from Use Cases

Scenario 1	Basic flow			
Scenario 2	Basic flow	Alternate flow 1		
Scenario 3	Basic flow	Alternate flow 2		
Scenario 4	Basic flow	Alternate flow 2	Alternate flow 3	

Scenario Test from Use Cases

Step 4: Generating the Test Cases

Once the set of scenarios has been identified, the next step is to identify the test cases. This is accomplished by analyzing the scenarios and reviewing the use case textual descriptions. There should be at least one test case for each scenario. For each invalid test case, there should be only one invalid input.

Scenario Test from Use Cases

To document the test cases, a matrix format can be used, as illustrated in the Table below. The first column of the first row contains the test case ID, and the second column has a brief description of the test case and the scenario being tested.

All the other columns except the last one contain data elements that will be used to implement the tests. The last column contains a description of the test case's expected output.

The "V" depicts a valid test input, and an "I" depicts an invalid test input.

Scenario Test from Use Cases

Test Case ID	*Scenario/Condition*	*Student Name*	*Address*	*Phone Number*	*Student Number*	*Course Rejected*	*Exit Enrollment*	*Expected Result*
Enroll 1	Scenario 1—successful enrollment	V	V	V	V	No	Yes	Selected courses are displayed and exit system
Enroll 2	Scenario 2—unidentified student	I	N/A	N/A	N/A	N/A	No	Error message; back to list of available courses
Enroll 3	Scenario 3—rejects a course	V	V	V	V	Yes	No	Selected course is selected, rejected; back to list of available courses

Scenario Test from Use Cases

Step 5: Generating Test Data

Once all of the test cases have been identified, they should be reviewed and validated to ensure accuracy and to identify redundant or missing test cases. Then, once they are approved, the final step is to substitute actual data values for the I's and V's.

Table below shows a test case matrix with values substituted for the I's and V's in the previous matrix. A number of techniques can be used for identifying data values.

Table 4.6 Enrollment Test Case Details

Test Case ID	Scenario/ Condition	Student Name	Address	Phone Number	Student Number	Course Selected	Expected Result
Enroll 1	Scenario 1—successful registration	John Doe	2719 Brook Avenue, Dallas, Texas 75093	(972) 9832876	G3982	Oceanography	Courses and schedule displayed; exit system
Enroll 2	Scenario 2—unidentified student	Invalid	2719 Brook Avenue, Dallas, Texas 75093	(972) 9832876	G3982	Oceanography	Error message; back to login screen
Enroll 3	Scenario 2—unidentified student	John Doe	Invalid	(972) 9832876	G3982	Oceanography	Error message; back to login screen
Enroll 4	Scenario 2—unidentified student	John Doe	2719 Brook Avenue, Dallas, Texas 75093	Invalid	G3982	Oceanography	Error message; back to login screen
Enroll 5	Scenario 2—unidentified student	John Doe	2719 Brook Avenue, Dallas, Texas 75093	(972) 9832876	Invalid	Oceanography	Error message; back to login screen
Enroll 6	Scenario 2—unidentified student	John Doe	2719 Brook Avenue, Dallas, Texas 75093	(972) 9832876	G3982	Invalid	Error message; back to login screen
Enroll 7	Scenario 3—unidentified student	John Doe	2719 Brook Avenue, Dallas, Texas 75093	(972) 9832876	G3982	Oceanography rejected	Back to login screen

Black Box Testing

- Pros
  - **This is what the product is about.**
  - **Implementation independent.**
- Cons
  - **For complicated products it is hard to identify erroneous output.**
  - **It is hard to estimate whether the product is error-free.**
  - Practically: **Choosing input with high probability of *error detection* is very difficult (e.g. division of two numbers).**