Pairwise Testing

Contents

[Orthogonal Array 2](#_Toc444155683)

[Step 1 2](#_Toc444155684)

[Step 2 2](#_Toc444155685)

[Step 3 3](#_Toc444155686)

[Step 4 3](#_Toc444155687)

[Step 5 3](#_Toc444155688)

[Step 6 4](#_Toc444155689)

[Pairwise Testing in NUnit 5](#_Toc444155690)

[Combinatorial Testing in NUnit 5](#_Toc444155691)

[References 7](#_Toc444155692)

(Naik & Tripathy, 2008)

|  |
| --- |
|  |

Consider system, which has three input parameters,, and . has two possible values: . has two possible values: . has three possible values: . There are possible test cases, if we are to choose all combinations. Pairwise testing is a special case of all-combination testing where each possible combination of values for every pair of input values is covered by at least one test case.

# Orthogonal Array

Orthogonal arrays allows you select a subset of combinations to ensure that all levels of factors are considered equally. It guarantees testing the pairwise combinations of all the selected variables. It generates fewer test cases than the all-combination approach. It generates a test suite that has an even distribution of all pairwise combinations.

## Step 1

Identify the maximum number of independent input variables with which a system will be tested. This maps to the ***factors*** in the array – each input variable maps to a different factor. The factors map to the columns of the array.

In system there are three independent input variables: , , and .

## Step 2

Identify the maximum number of values that each independent variable will take. This maps to the ***levels*** of the array.

* X has a maximum of 2 values
* Y has a maximum of 2 values
* Z has a maximum of 3 values

## Step 3

Find a suitable orthogonal array with the smallest number of runs. A suitable array is one that has at least as many factors as needed from step 1 and has at least as many levels for each of those factors as identified in step 2.

<http://www.freequality.org/documents/tools/Tagarray_files/tamatrix.htm>

Choice:

|  |
| --- |
|  |

## Step 4

Map the variables to the factors and values of each variable to the levels on the array.

Factors:

Values:

Z: 1 = p, 2 = q, 3 = r

X: 1 = True, 2 = False

Y: 1 = 0, 2 = 5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Runs** | **Z** | **X** | **Y** | **Not used** |
| 1 | p | True | 0 |  |
| 2 | p | False | 5 |  |
| 3 | p | **3** | **3** |  |
| 4 | q | True | 5 |  |
| 5 | q | False | **3** |  |
| 6 | q | **3** | 0 |  |
| 7 | r | True | **3** |  |
| 8 | r | False | 0 |  |
| 9 | r | **3** | 5 |  |

## 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.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Runs** | **Z** | **X** | **Y** | **Not used** |
| 1 | p | True | 0 |  |
| 2 | p | False | 5 |  |
| 3 | p | **True** | **5** |  |
| 4 | q | True | 5 |  |
| 5 | q | False | **5** |  |
| 6 | q | **False** | 0 |  |
| 7 | r | True | **0** |  |
| 8 | r | False | 0 |  |
| 9 | r | **True** | 5 |  |

## Step 6

Transcribe the runs into test cases.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case** | **X** | **Y** | **Z** |
| 1 | **True** | **0** | **p** |
| 2 | **False** | **5** | **p** |
| 3 | **True** | **5** | **p** |
| 4 | **True** | **5** | **q** |
| 5 | **False** | **5** | **q** |
| 6 | **False** | **0** | **q** |
| 7 | **True** | **0** | **r** |
| 8 | **False** | **0** | **r** |
| 9 | **True** | **5** | **r** |

# Pairwise Testing in NUnit

(Nunit, 2015)

The **PairwiseAttribute** is used on a test to specify that NUnit should generate test cases in such a way that all possible pairs of values are used. This is a well-known approach for combatting the combinatorial explosion of test cases when more than two features (parameters) are involved.

|  |
| --- |
| [Test, **Pairwise**]  public void PairwiseTesting(  **[Values(true, false)]** bool X,  **[Values(0, 5)]** int Y,  **[Values('p', 'q', 'r')]**char Z  )  {  Assert.That(true);  } |
|  |

# Combinatorial Testing in NUnit

The **CombinatorialAttribute** is used on a test to specify that NUnit should generate test cases for all possible combinations of the individual data items provided for the parameters of a test.

|  |
| --- |
| [Test, **Combinatorial**]  public void CombinatorialTesting(  [Values(true, false)] bool X,  [Values(0, 5)] int Y,  [Values('p', 'q', 'r')] char Z  )  {  Assert.That(true);  } |
|  |

When used on a generic method the programmer must ensure that all possible combinations of arguments are valid. When multiple parameters use the same generic type (e.g.: T) this may not be possible and the attribute may generate invalid test cases.

# References

(2015). Retrieved from Nunit: http://www.nunit.org/index.php?p=pairwise&r=2.5.10

Naik, K., & Tripathy, P. (2008). *Software Testing and Quality Assurance Theory and Practice.* Wiley.