**Chapter 6: Pairwise Testing**

*Anton was attracted to Angela like a moth to a flame - not just any moth, but one of the giant silk moths of the genus Hyalophora, perhaps Hyalophora euryalus, whose great red*-*brown wings with white basal and postmedian lines flap almost languorously until one ignites in the flame, fanning the conflagration to ever greater heights until burning down to the hirsute thorax and abdomen, the fat-laden contents of which provide a satisfying sizzle to end the agony.*

— Andrew Amlen

**Introduction**

As they used to say on *Monty Python,* "And now for something completely different."

Consider these situations:

* A Web site must operate correctly with different browsers—Internet Explorer 5.0, 5.5, and 6.0, Netscape 6.0, 6.1, and 7.0, Mozilla 1.1, and Opera 7; using different plug-ins—RealPlayer, MediaPlayer, or none; running on different client operating systems—Windows 95, 98, ME, NT, 2000, and XP; receiving pages from different servers—IIS, Apache, and WebLogic; running on different server operating systems—Windows NT, 2000, and Linux.

**Web Combinations**

8 browsers

3 plug-ins

6 client operating systems

3 servers

3 server OS

1,296 combinations.

* A bank has created a new data processing system that is ready for testing. This bank has different kinds of customers—consumers, very important consumers, businesses, and non-profits; different kinds of accounts—checking, savings, mortgages, consumer loans, and commercial loans; they operate in different states, each with different regulations—California, Nevada, Utah, Idaho, Arizona, and New Mexico.

**Bank Combinations**

4 customer types

5 account types

6 states

120 combinations.

* In an object-oriented system, an object of class A can pass a message containing a parameter P to an object of class X. Classes B, C, and D inherit from A so they too can send the message. Classes Q, R, S, and T inherit from P so they too can be passed as the parameter. Classes Y and Z inherit from X so they too can receive the message.

**OO Combinations**

4 senders

5 parameters

3 receivers

60 combinations.

|  |  |  |
| --- | --- | --- |
|  | Insight | Students in my classes often have a very difficult time thinking of bad ways to do things. Cultivate the skill of choosing poorly. It will be invaluable in evaluating others' ideas. |

**Can You Believe This?**

A student in one of my classes shared this story: His organization uses a process they call "Post-Installation Test Planning." It sounds impressive until you decipher it. Whatever tests they happen to run that happen to pass are documented as their Test Plan.

What do these very different situations all have in common? Each has a large number of combinations that should be tested. Each has a large number of combinations that may be risky if we do not test. Each has such a large number of combinations that we may not have the resources to construct and run all the tests, there are just too many. We must, somehow, select a reasonably sized subset that we could test given our resource constraints. What are some ways of choosing such a subset? This list starts with the worst schemes but does improve:

|  |  |
| --- | --- |
| [Image from book](javascript:PopImage('IMG_17','http://images.books24x7.com/bookimages/id_7898/fig81_01_0.jpg','108','996')) | * Don't test at all. Simply give up because the number of input combinations, and thus the number of test cases, is just too great. * Test all combinations [once], but delay the project so it misses its market window so that everyone quits from stress, or the company goes out of business. * Choose one or two tests and hope for the best. * Choose the tests that you have already run, perhaps as part of programmer-led testing. Incorporate them into a formal test plan and run them again. * Choose the tests that are easy to create and run. Ignore whether they provide useful information about the quality of the product. * Make a list of all the combinations and choose the first few. * Make a list of all the combinations and choose a random subset. |
| Image from book | * By magic, choose a specially selected, fairly small subset that finds a great many defects—more than you would expect from such a subset. |

This last scheme sounds like a winner (but it is a little vague). The question is—what is the "magic" that allows us to choose that "specially selected" subset?

|  |  |  |
| --- | --- | --- |
|  | Insight | Random selection can be a very good approach to choosing a subset but most people have a difficult time choosing truly randomly. |

The answer is not to attempt to test all the combinations for all the values for all the variables but to test **all pairs** of variables. This significantly reduces the number of tests that must be created and run. Consider the significant reductions in test effort in these examples:

* If a system had four different input parameters and each one could take on one of three different values, the number of combinations is 34 which is 81. It is possible to cover all the pairwise input combinations in only nine tests.
* If a system had thirteen different input parameters and each one could take on one of three different values, the number of combinations is 313 which is 1,594,323. It is possible to cover all the pairwise input combinations in only fifteen tests.
* If a system had twenty different input parameters and each one could take on one of ten different values, the number of combinations is 1020. It is possible to cover all the pairwise input combinations in only 180 tests.

There is much anecdotal evidence about the benefit of pairwise testing. Unfortunately, there are only a few documented studies:

* In a case study published by Brownlie of AT&T regarding the testing of a local-area network-based electronic mail system, pairwise testing detected 28 percent more defects than their original plan of developing and executing 1,500 test cases (later reduced to 1,000 because of time constraints) and took 50 percent less effort.
* A study by the National Institute of Standards and Technology published by Wallace and Kuhn on software defects in recalled medical devices reviewed fifteen years of defect data. They concluded that 98 percent of the reported software flaws could have been detected by testing all pairs of parameter settings.
* Kuhn and Reilly analyzed defects recorded in the Mozilla Web browser database. They determined that pairwise testing would have detected 76 percent of the reported errors.

Why does pairwise testing work so well? I don't know. There is no underlying "software physics" that requires it. One hypothesis is that most defects are either single-mode defects (the function under test simply does not work and any test of that function would find the defect) or they are double-mode defects (it is the pairing of this function/module with that function/module that fails even though all other pairings perform properly). Pairwise testing defines a minimal subset that guides us to test for all single-mode and double-mode defects. The success of this technique on many projects, both documented and undocumented, is a great motivation for its use.

|  |  |  |
| --- | --- | --- |
|  | Note | Pairwise testing may not choose combinations which the developers and testers know are either frequently used or highly risky. If these combinations exist, use the pairwise tests, then add additional test cases to minimize the risk of missing an important combination. |

## Technique

Two different techniques are used to identify all the pairs for creating test cases—orthogonal arrays and the Allpairs algorithm.

### Orthogonal Arrays

What are orthogonal arrays? The origin of orthogonal arrays can be traced back to Euler, the great mathematician, in the guise of Latin Squares. Genichi Taguchi has popularized their use in hardware testing. An excellent reference book is *Quality Engineering Using Robust Design* by Madhav S. Phadke.

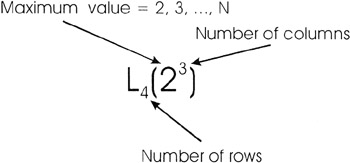
Consider the numbers 1 and 2. How many pair combinations (combinations taken two at a time) of '1' and '2' exist? {1,1}, {1,2}, {2,1} and {2,2}. An orthogonal array is a two-dimensional array of numbers that has this interesting property—choose any two columns in the array. All the pairwise combinations of its values will occur in every pair of columns. Let's examine an L4(23) array:

| Table 6-1: L4(23) Orthogonal Array  [[http://www.books24x7.com/images/b24-bluearrow.gif](http://www.books24x7.com/outputobject.asp?bookid=7898&chunkid=362579270&objectid=ch06table01&objecttype=spreadsheet)Open table as spreadsheet](http://www.books24x7.com/outputobject.asp?bookid=7898&chunkid=362579270&objectid=ch06table01&objecttype=spreadsheet) | | | |
| --- | --- | --- | --- |
|  | **1** | **2** | **3** |
| 1 | **1** | **1** | **1** |
| 2 | **1** | **2** | **2** |
| 3 | **2** | **1** | **2** |
| 4 | **2** | **2** | **1** |

The gray column headings and row numbers are not part of the orthogonal array but are included for convenience in referencing the cells. Examine columns 1 and 2—do the four combinations of 1 and 2 all appear in that column pair? Yes, and in the order listed earlier. Now examine columns 1 and 3—do the four combinations of 1 and 2 appear in that column pair? Yes, although in a different order. Finally, examine columns 2 and 3—do the four combinations appear in that column pair also? Yes they do. The L4(23) array is orthogonal; that is, choose any two columns, all the pairwise combinations will occur in all the column pairs.

|  |  |  |
| --- | --- | --- |
|  | Important Note | As a tester you do **not** have to create orthogonal arrays, all you must do is locate one of the proper size. Books, Web sites, and automated tools will help you do this. |

A note about the curious (but standard) notation: L4 means an orthogonal array with four rows, (23) is not an exponent. It means that the array has three columns, each with either a 1 or a 2.

[](javascript:PopImage('IMG_19','http://images.books24x7.com/bookimages/id_7898/fig85_01_0.jpg','727','339'))  
Figure 6-1: Orthogonal array notation

Let's consider a larger orthogonal array. Given the numbers 1, 2 and 3, how many pair combinations of 1, 2, and 3 exist? {1,1}, {1,2}, {1,3}, {2,1}, {2,2}, {2,3}, {3,1}, {3,2}, and {3,3}. Below is an L9(34) array:

| Table 6-2: L9(34) Orthogonal Array  [[http://www.books24x7.com/images/b24-bluearrow.gif](http://www.books24x7.com/outputobject.asp?bookid=7898&chunkid=362579270&objectid=ch06table02&objecttype=spreadsheet)Open table as spreadsheet](http://www.books24x7.com/outputobject.asp?bookid=7898&chunkid=362579270&objectid=ch06table02&objecttype=spreadsheet) | | | | |
| --- | --- | --- | --- | --- |
|  | **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 |

Examine columns 1 and 2—do the nine combinations of 1, 2, and 3 all appear in that column pair? Yes. Now examine columns 1 and 3—do the nine combinations of 1, 2, and 3 appear in that column pair? Yes, although in a different order. Examine columns 1 and 4—do the nine combinations appear in that column pair also? Yes they do. Continue on by examining other pairs of columns—2 and 3, 2 and 4, and finally 3 and 4. The L9(34) array is orthogonal; that is, choose any two columns, all the combinations will occur in all of the column pairs.

**Tool**

The rdExpert tool from Phadke Associates implements the orthogonal array approach. See <http://www.phadkeassociates.com>

Note that not all combinations of 1s, 2s, and 3s appear in the array. For example, {1,1,2}, {1,2,1}, and {2,2,2) do not appear. Orthogonal arrays only guarantee that all the pair combinations exist in the array. Combinations such as {2,2,2} are triples, not pairs.

The following is an L18(35) orthogonal array. It has five columns, each containing a 1, 2, or 3. Examine columns 1 and 2 for the pair {1,1}. Does that pair exist in those two columns? Wait! Don't look at the array. From the definition of an orthogonal array, what is the answer? Yes, that pair exists along with every other pair of 1, 2, and 3. The pair {1,1} is in row 1. Note that {1,1} also appears in row 6. Returning to the original description of orthogonal arrays,

An orthogonal array is a two-dimensional array of numbers that has this interesting property—choose any two columns in the array. All the pairwise combinations of its values will occur in every column pair.

This definition is not totally complete. Not only will all the pair combinations occur in the array, but if any pair occurs multiple times, all pairs will occur that same number of times. This is because orthogonal arrays are "balanced." Examine columns 3 and 5—look for {3,2}. That combination appears in rows 6 and 17.

| Table 6-3: L18(35) Orthogonal Array  [[http://www.books24x7.com/images/b24-bluearrow.gif](http://www.books24x7.com/outputobject.asp?bookid=7898&chunkid=362579270&objectid=ch06table03&objecttype=spreadsheet)Open table as spreadsheet](http://www.books24x7.com/outputobject.asp?bookid=7898&chunkid=362579270&objectid=ch06table03&objecttype=spreadsheet) | | | | | |
| --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** |
| 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 1 | 2 | 3 | 3 | 1 |
| 3 | 1 | 3 | 2 | 3 | 2 |
| 4 | 1 | 2 | 2 | 1 | 3 |
| 5 | 1 | 3 | 1 | 2 | 3 |
| 6 | 1 | 1 | 3 | 2 | 2 |
| 7 | 2 | 2 | 2 | 2 | 2 |
| 8 | 2 | 3 | 1 | 1 | 2 |
| 9 | 2 | 1 | 3 | 1 | 3 |
| 10 | 2 | 3 | 3 | 2 | 1 |
| 11 | 2 | 1 | 2 | 3 | 1 |
| 12 | 2 | 2 | 1 | 3 | 3 |
| 13 | 3 | 3 | 3 | 3 | 3 |
| 14 | 3 | 1 | 2 | 2 | 3 |
| 15 | 3 | 2 | 1 | 2 | 1 |
| 16 | 3 | 1 | 1 | 3 | 2 |
| 17 | 3 | 2 | 3 | 1 | 2 |
| 18 | 3 | 3 | 2 | 1 | 1 |

In orthogonal arrays not all of the columns must have the same range of values (1..2, 1..3, 1..5, etc.). Some orthogonal arrays are mixed. The following is an L18(2137) orthogonal array. It has one column of 1s and 2s, and seven columns of 1s, 2s, and 3s.

| Table 6-4: L18(2137) Orthogonal Array  [[http://www.books24x7.com/images/b24-bluearrow.gif](http://www.books24x7.com/outputobject.asp?bookid=7898&chunkid=362579270&objectid=ch06table04&objecttype=spreadsheet)Open table as spreadsheet](http://www.books24x7.com/outputobject.asp?bookid=7898&chunkid=362579270&objectid=ch06table04&objecttype=spreadsheet) | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 |
| 3 | 1 | 1 | 3 | 3 | 3 | 3 | 3 | 3 |
| 4 | 1 | 2 | 1 | 1 | 2 | 2 | 3 | 3 |
| 5 | 1 | 2 | 2 | 2 | 3 | 3 | 1 | 1 |
| 6 | 1 | 2 | 3 | 3 | 1 | 1 | 2 | 2 |
| 7 | 1 | 3 | 1 | 2 | 1 | 3 | 2 | 3 |
| 8 | 1 | 3 | 2 | 3 | 2 | 1 | 3 | 1 |
| 9 | 1 | 3 | 3 | 1 | 3 | 2 | 1 | 2 |
| 10 | 2 | 1 | 1 | 3 | 3 | 2 | 2 | 1 |
| 11 | 2 | 1 | 2 | 1 | 1 | 3 | 3 | 2 |
| 12 | 2 | 1 | 3 | 2 | 2 | 1 | 1 | 3 |
| 13 | 2 | 2 | 1 | 2 | 3 | 1 | 3 | 2 |
| 14 | 2 | 2 | 2 | 3 | 1 | 2 | 1 | 3 |
| 15 | 2 | 2 | 3 | 1 | 2 | 3 | 2 | 1 |
| 16 | 2 | 3 | 1 | 3 | 2 | 3 | 1 | 2 |
| 17 | 2 | 3 | 2 | 1 | 3 | 1 | 2 | 3 |
| 18 | 2 | 3 | 3 | 2 | 1 | 2 | 3 | 1 |

**Reference**

Neil J.A. Sloane maintains a very comprehensive catalog of orthogonal arrays at [http://www.research.att.com/~njas/oadir/index.html](http://www.research.att.com/%7Enjas/oadir/index.html)

### Using Orthogonal Arrays

The process of using orthogonal arrays to select pairwise subsets for testing is:

1. Identify the variables.
2. Determine the number of choices for each variable.
3. Locate an orthogonal array which has a column for each variable and values within the columns that correspond to the choices for each variable.
4. Map the test problem onto the orthogonal array.
5. Construct the test cases.

If this seems rather vague at this point it's time for an example.

Web-based systems such as Brown & Donaldson and the Stateless University Registration System must operate in a number of environments. Let's execute the process step-by-step using an orthogonal array to choose test cases. Consider the first example in the introduction describing the software combinations a Web site must operate with.

1. **Identify the variables.**

The variables are Browser, Plug-in, Client operating system, Server, and Server operating system.

1. **Determine the number of choices for each variable.**

Browser - Internet Explorer 5.0, 5.5, and 6.0, Netscape 6.0, 6.1, and 7.0, Mozilla 1.1, and Opera 7 (8 choices).

Plug-in - None, RealPlayer, and MediaPlayer (3 choices).

Client operating system - Windows 95, 98, ME, NT, 2000, and XP (6 choices).

Server - IIS, Apache, and WebLogic (3 choices).

Server operating system - Windows NT, 2000, and Linux (3 choices).

Multiplying 8 x 3 x 6 x 3 x 3 we find there are 1,296 combinations. For "complete" test coverage, each of these combinations should be tested.

1. **Locate an orthogonal array that has a column for each variable and values within the columns that correspond to the choices of each variable.**

What size array is needed? First, it must have five columns, one for each variable in this example. The first column must support eight different levels (1 through 8). The second column must support three levels (1 through 3). The third requires six levels. The fourth and the fifth each require three levels. The perfect size orthogonal array would be 816133 (one column of 1 through 8, one column of 1 through 6, and three columns of 1 through 3). Unfortunately, one of this exact size does not exist. When this occurs, we simply pick the next larger array.

|  |  |  |
| --- | --- | --- |
|  | Important Note | As a tester you do not have to create orthogonal arrays. All you must do is locate one of the proper size and then perform the mapping of the test problem onto the array. |

The following orthogonal array meets our requirements. It's an L64(8243) array. Orthogonal arrays can be found in a number of books and on the Web. A favorite book is *Quality Engineering Using Robust Design* by Madhav S. Phadke. In addition, an excellent catalog is maintained on the Web by Neil J.A. Sloane of AT&T. See [http://www.research.att.com/~njas/oadir/index.html](http://www.research.att.com/%7Enjas/oadir/index.html).

The requirement of 8161 (one column of 1 through 8 and 1 column of 1 through 6) is met by 82 (two columns of 1 through 8). The requirement of 33 (three columns of 1 through 3) is met by 43 (three columns of 1 through 4).

The number of combinations of all the values of all the variables is 1,296 and thus 1,296 test cases should be created and run for complete coverage. Using this orthogonal array, all pairs of all the values of all the variables can be covered in only sixty-four tests, a 95 percent reduction in the number of test cases.

| Table 6-5: L64(8243) Orthogonal Array  [[http://www.books24x7.com/images/b24-bluearrow.gif](http://www.books24x7.com/outputobject.asp?bookid=7898&chunkid=362579270&objectid=ch06table05&objecttype=spreadsheet)Open table as spreadsheet](http://www.books24x7.com/outputobject.asp?bookid=7898&chunkid=362579270&objectid=ch06table05&objecttype=spreadsheet) | | | | | |
| --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** |
| 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 1 | 4 | 3 | 4 | 4 |
| 3 | 1 | 4 | 2 | 4 | 4 |
| 4 | 1 | 1 | 4 | 1 | 1 |
| 5 | 1 | 3 | 5 | 3 | 3 |
| 6 | 1 | 2 | 7 | 2 | 2 |
| 7 | 1 | 2 | 6 | 2 | 2 |
| 8 | 1 | 3 | 8 | 3 | 3 |
| 9 | 3 | 4 | 1 | 3 | 3 |
| 10 | 3 | 1 | 3 | 2 | 2 |
| 11 | 3 | 1 | 2 | 2 | 2 |
| 12 | 3 | 4 | 4 | 3 | 3 |
| 13 | 3 | 2 | 5 | 1 | 1 |
| 14 | 3 | 3 | 7 | 4 | 4 |
| 15 | 3 | 3 | 6 | 4 | 4 |
| 16 | 3 | 2 | 8 | 1 | 1 |
| 17 | 2 | 3 | 1 | 2 | 1 |
| 18 | 2 | 2 | 3 | 3 | 4 |
| 19 | 2 | 2 | 2 | 3 | 4 |
| 20 | 2 | 3 | 4 | 2 | 1 |
| 21 | 2 | 1 | 5 | 4 | 3 |
| 22 | 2 | 4 | 7 | 1 | 2 |
| 23 | 2 | 4 | 6 | 1 | 2 |
| 24 | 2 | 1 | 8 | 4 | 3 |
| 25 | 4 | 2 | 1 | 4 | 3 |
| 26 | 4 | 3 | 3 | 1 | 2 |
| 27 | 4 | 3 | 2 | 1 | 2 |
| 28 | 4 | 2 | 4 | 4 | 3 |
| 29 | 4 | 4 | 5 | 2 | 1 |
| 30 | 4 | 1 | 7 | 3 | 4 |
| 31 | 4 | 1 | 6 | 3 | 4 |
| 32 | 4 | 4 | 8 | 2 | 1 |
| 33 | 5 | 2 | 1 | 4 | 2 |
| 34 | 5 | 3 | 3 | 1 | 3 |
| 35 | 5 | 3 | 2 | 1 | 3 |
| 36 | 5 | 2 | 4 | 4 | 2 |
| 37 | 5 | 4 | 5 | 2 | 4 |
| 38 | 5 | 1 | 7 | 3 | 1 |
| 39 | 5 | 1 | 6 | 3 | 1 |
| 40 | 5 | 4 | 8 | 2 | 4 |
| 41 | 7 | 3 | 1 | 2 | 4 |
| 42 | 7 | 2 | 3 | 3 | 1 |
| 43 | 7 | 2 | 2 | 3 | 1 |
| 44 | 7 | 3 | 4 | 2 | 4 |
| 45 | 7 | 1 | 5 | 4 | 2 |
| 46 | 7 | 4 | 7 | 1 | 3 |
| 47 | 7 | 4 | 6 | 1 | 3 |
| 48 | 7 | 1 | 8 | 4 | 2 |
| 49 | 6 | 4 | 1 | 3 | 2 |
| 50 | 6 | 1 | 3 | 2 | 3 |
| 51 | 6 | 1 | 2 | 2 | 3 |
| 52 | 6 | 4 | 4 | 3 | 2 |
| 53 | 6 | 2 | 5 | 1 | 4 |
| 54 | 6 | 3 | 7 | 4 | 1 |
| 55 | 6 | 3 | 6 | 4 | 1 |
| 56 | 6 | 2 | 8 | 1 | 4 |
| 57 | 8 | 1 | 1 | 1 | 4 |
| 58 | 8 | 4 | 3 | 4 | 1 |
| 59 | 8 | 4 | 2 | 4 | 1 |
| 60 | 8 | 1 | 4 | 1 | 4 |
| 61 | 8 | 3 | 5 | 3 | 2 |
| 62 | 8 | 2 | 7 | 2 | 3 |
| 63 | 8 | 2 | 6 | 2 | 3 |
| 64 | 8 | 3 | 8 | 3 | 2 |

1. **Map the test problem onto the orthogonal array.**

The Browser choices will be mapped onto column 1 of the orthogonal array. Cells containing a 1 will represent IE 5.0; cells with a 2 will represent IE5.5; cells with a 3 will represent IE 6.0; etc. The mapping is:

* + 1 ↔ IE 5.0
  + 2 ↔ IE 5.5
  + 3 ↔ IE 6.0
  + 4 ↔ Netscape 6.0
  + 5 ↔ Netscape 6.1
  + 6 ↔ Netscape 7.0
  + 7 ↔ Mozilla 1.1
  + 8 ↔ Opera 7

Partially filling in the first column gives:

| Table 6-6: L64 (8243) with a partial mapping of its first column.  [[http://www.books24x7.com/images/b24-bluearrow.gif](http://www.books24x7.com/outputobject.asp?bookid=7898&chunkid=362579270&objectid=ch06table06&objecttype=spreadsheet)Open table as spreadsheet](http://www.books24x7.com/outputobject.asp?bookid=7898&chunkid=362579270&objectid=ch06table06&objecttype=spreadsheet) | | | | | |
| --- | --- | --- | --- | --- | --- |
|  | **Browser** | **2** | **3** | **4** | **5** |
| 1 | IE 5.0 | 1 | 1 | 1 | 1 |
| 2 | 1 | 4 | 3 | 4 | 4 |
| 3 | 1 | 4 | 2 | 4 | 4 |
| 4 | 1 | 1 | 4 | 1 | 1 |
| 5 | 1 | 3 | 5 | 3 | 3 |
| 6 | 1 | 2 | 7 | 2 | 2 |
| 7 | 1 | 2 | 6 | 2 | 2 |
| 8 | 1 | 3 | 8 | 3 | 3 |
| 9 | IE 6.0 | 4 | 1 | 3 | 3 |
| 10 | 3 | 1 | 3 | 2 | 2 |
| 11 | 3 | 1 | 2 | 2 | 2 |
| 12 | 3 | 4 | 4 | 3 | 3 |
| 13 | 3 | 2 | 5 | 1 | 1 |
| 14 | 3 | 3 | 7 | 4 | 4 |
| 15 | 3 | 3 | 6 | 4 | 4 |
| 16 | 3 | 2 | 8 | 1 | 1 |
| 17 | IE 5.5 | 3 | 1 | 2 | 1 |
| 18 | 2 | 2 | 3 | 3 | 4 |
| 19 | 2 | 2 | 2 | 3 | 4 |
| 20 | 2 | 3 | 4 | 2 | 1 |
| 21 | 2 | 1 | 5 | 4 | 3 |
| 22 | 2 | 4 | 7 | 1 | 2 |
| 23 | 2 | 4 | 6 | 1 | 2 |
| 24 | 2 | 1 | 8 | 4 | 3 |
| 25 | Net 6.0 | 2 | 1 | 4 | 3 |
| 26 | 4 | 3 | 3 | 1 | 2 |
| 27 | 4 | 3 | 2 | 1 | 2 |
| 28 | 4 | 2 | 4 | 4 | 3 |
| 29 | 4 | 4 | 5 | 2 | 1 |
| 30 | 4 | 1 | 7 | 3 | 4 |
| 31 | 4 | 1 | 6 | 3 | 4 |
| 32 | 4 | 4 | 8 | 2 | 1 |
| 33 | Net 6.1 | 2 | 1 | 4 | 2 |
| 34 | 5 | 3 | 3 | 1 | 3 |
| 35 | 5 | 3 | 2 | 1 | 3 |
| 36 | 5 | 2 | 4 | 4 | 2 |
| 37 | 5 | 4 | 5 | 2 | 4 |
| 38 | 5 | 1 | 7 | 3 | 1 |
| 39 | 5 | 1 | 6 | 3 | 1 |
| 40 | 5 | 4 | 8 | 2 | 4 |
| 41 | Moz 1.1 | 3 | 1 | 2 | 4 |
| 42 | 7 | 2 | 3 | 3 | 1 |
| 43 | 7 | 2 | 2 | 3 | 1 |
| 44 | 7 | 3 | 4 | 2 | 4 |
| 45 | 7 | 1 | 5 | 4 | 2 |
| 46 | 7 | 4 | 7 | 1 | 3 |
| 47 | 7 | 4 | 6 | 1 | 3 |
| 48 | 7 | 1 | 8 | 4 | 2 |
| 49 | Net 7.0 | 4 | 1 | 3 | 2 |
| 50 | 6 | 1 | 3 | 2 | 3 |
| 51 | 6 | 1 | 2 | 2 | 3 |
| 52 | 6 | 4 | 4 | 3 | 2 |
| 53 | 6 | 2 | 5 | 1 | 4 |
| 54 | 6 | 3 | 7 | 4 | 1 |
| 55 | 6 | 3 | 6 | 4 | 1 |
| 56 | 6 | 2 | 8 | 1 | 4 |
| 57 | Opera 7 | 1 | 1 | 1 | 4 |
| 58 | 8 | 4 | 3 | 4 | 1 |
| 59 | 8 | 4 | 2 | 4 | 1 |
| 60 | 8 | 1 | 4 | 1 | 4 |
| 61 | 8 | 3 | 5 | 3 | 2 |
| 62 | 8 | 2 | 7 | 2 | 3 |
| 63 | 8 | 2 | 6 | 2 | 3 |
| 64 | 8 | 3 | 8 | 3 | 2 |

Is it clear what is happening? In column 1 (which we have chosen to represent the Browser) every cell containing a 1 is being replaced with "IE 5.0." Every cell containing a 2 is being replaced with "IE 5.5." Every cell containing an 8 is being replaced with "Opera 7," etc.

We'll continue by completing the mapping (replacement) of all the cells in column 1. Note that the mapping between the variable values and the 1s, 2s, and 3s is totally arbitrary. There is no logical connection between "1" and IE 5.0 or "7" and Mozilla 1.1. But, although the initial assignment is arbitrary, once chosen, the assignments and use must remain consistent within each column.

| Table 6-7: L64 (8243) with a full mapping of its first column.  [[http://www.books24x7.com/images/b24-bluearrow.gif](http://www.books24x7.com/outputobject.asp?bookid=7898&chunkid=362579270&objectid=ch06table07&objecttype=spreadsheet)Open table as spreadsheet](http://www.books24x7.com/outputobject.asp?bookid=7898&chunkid=362579270&objectid=ch06table07&objecttype=spreadsheet) | | | | | |
| --- | --- | --- | --- | --- | --- |
|  | **Browser** | **2** | **3** | **4** | **5** |
| 1 | IE 5.0 | 1 | 1 | 1 | 1 |
| 2 | IE 5.0 | 4 | 3 | 4 | 4 |
| 3 | IE 5.0 | 4 | 2 | 4 | 4 |
| 4 | IE 5.0 | 1 | 4 | 1 | 1 |
| 5 | IE 5.0 | 3 | 5 | 3 | 3 |
| 6 | IE 5.0 | 2 | 7 | 2 | 2 |
| 7 | IE 5.0 | 2 | 6 | 2 | 2 |
| 8 | IE 5.0 | 3 | 8 | 3 | 3 |
| 9 | IE 6.0 | 4 | 1 | 3 | 3 |
| 10 | IE 6.0 | 1 | 3 | 2 | 2 |
| 11 | IE 6.0 | 1 | 2 | 2 | 2 |
| 12 | IE 6.0 | 4 | 4 | 3 | 3 |
| 13 | IE 6.0 | 2 | 5 | 1 | 1 |
| 14 | IE 6.0 | 3 | 7 | 4 | 4 |
| 15 | IE 6.0 | 3 | 6 | 4 | 4 |
| 16 | IE 6.0 | 2 | 8 | 1 | 1 |
| 17 | IE 5.5 | 3 | 1 | 2 | 1 |
| 18 | IE 5.5 | 2 | 3 | 3 | 4 |
| 19 | IE 5.5 | 2 | 2 | 3 | 4 |
| 20 | IE 5.5 | 3 | 4 | 2 | 1 |
| 21 | IE 5.5 | 1 | 5 | 4 | 3 |
| 22 | IE 5.5 | 4 | 7 | 1 | 2 |
| 23 | IE 5.5 | 4 | 6 | 1 | 2 |
| 24 | IE 5.5 | 1 | 8 | 4 | 3 |
| 25 | Net 6.0 | 2 | 1 | 4 | 3 |
| 26 | Net 6.0 | 3 | 3 | 1 | 2 |
| 27 | Net 6.0 | 3 | 2 | 1 | 2 |
| 28 | Net 6.0 | 2 | 4 | 4 | 3 |
| 29 | Net 6.0 | 4 | 5 | 2 | 1 |
| 30 | Net 6.0 | 1 | 7 | 3 | 4 |
| 31 | Net 6.0 | 1 | 6 | 3 | 4 |
| 32 | Net 6.0 | 4 | 8 | 2 | 1 |
| 33 | Net 6.1 | 2 | 1 | 4 | 2 |
| 34 | Net 6.1 | 3 | 3 | 1 | 3 |
| 35 | Net 6.1 | 3 | 2 | 1 | 3 |
| 36 | Net 6.1 | 2 | 4 | 4 | 2 |
| 37 | Net 6.1 | 4 | 5 | 2 | 4 |
| 38 | Net 6.1 | 1 | 7 | 3 | 1 |
| 39 | Net 6.1 | 1 | 6 | 3 | 1 |
| 40 | Net 6.1 | 4 | 8 | 2 | 4 |
| 41 | Moz 1.1 | 3 | 1 | 2 | 4 |
| 42 | Moz 1.1 | 2 | 3 | 3 | 1 |
| 43 | Moz 1.1 | 2 | 2 | 3 | 1 |
| 44 | Moz 1.1 | 3 | 4 | 2 | 4 |
| 45 | Moz 1.1 | 1 | 5 | 4 | 2 |
| 46 | Moz 1.1 | 4 | 7 | 1 | 3 |
| 47 | Moz 1.1 | 4 | 6 | 1 | 3 |
| 48 | Moz 1.1 | 1 | 8 | 4 | 2 |
| 49 | Net 7.0 | 4 | 1 | 3 | 2 |
| 50 | Net 7.0 | 1 | 3 | 2 | 3 |
| 51 | Net 7.0 | 1 | 2 | 2 | 3 |
| 52 | Net 7.0 | 4 | 4 | 3 | 2 |
| 53 | Net 7.0 | 2 | 5 | 1 | 4 |
| 54 | Net 7.0 | 3 | 7 | 4 | 1 |
| 55 | Net 7.0 | 3 | 6 | 4 | 1 |
| 56 | Net 7.0 | 2 | 8 | 1 | 4 |
| 57 | Opera 7 | 1 | 1 | 1 | 4 |
| 58 | Opera 7 | 4 | 3 | 4 | 1 |
| 59 | Opera 7 | 4 | 2 | 4 | 1 |
| 60 | Opera 7 | 1 | 4 | 1 | 4 |
| 61 | Opera 7 | 3 | 5 | 3 | 2 |
| 62 | Opera 7 | 2 | 7 | 2 | 3 |
| 63 | Opera 7 | 2 | 6 | 2 | 3 |
| 64 | Opera 7 | 3 | 8 | 3 | 2 |

Now that the first column has been mapped, let's proceed to the next one. The Plug-in choices will be mapped onto column 2 of the array. Cells containing a 1 will represent None (No plug-in); cells with a 2 will represent RealPlayer; cells with a 3 will represent MediaPlayer; cells with a 4 will not be mapped at the present time. The mapping is:

* + 1 ↔ None
  + 2 ↔ RealPlayer
  + 3 ↔ MediaPlayer
  + 4 ↔ Not used (at this time)

Filling in the second column gives:

| Table 6-8: L64 (8243) with a full mapping of its first and second columns.  [[http://www.books24x7.com/images/b24-bluearrow.gif](http://www.books24x7.com/outputobject.asp?bookid=7898&chunkid=362579270&objectid=ch06table08&objecttype=spreadsheet)Open table as spreadsheet](http://www.books24x7.com/outputobject.asp?bookid=7898&chunkid=362579270&objectid=ch06table08&objecttype=spreadsheet) | | | | | |
| --- | --- | --- | --- | --- | --- |
|  | **Browser** | **Plug-In** | **3** | **4** | **5** |
| 1 | IE 5.0 | None | 1 | 1 | 1 |
| 2 | IE 5.0 | 4 | 3 | 4 | 4 |
| 3 | IE 5.0 | 4 | 2 | 4 | 4 |
| 4 | IE 5.0 | None | 4 | 1 | 1 |
| 5 | IE 5.0 | MediaPlayer | 5 | 3 | 3 |
| 6 | IE 5.0 | RealPlayer | 7 | 2 | 2 |
| 7 | IE 5.0 | RealPlayer | 6 | 2 | 2 |
| 8 | IE 5.0 | MediaPlayer | 8 | 3 | 3 |
| 9 | IE 6.0 | 4 | 1 | 3 | 3 |
| 10 | IE 6.0 | None | 3 | 2 | 2 |
| 11 | IE 6.0 | None | 2 | 2 | 2 |
| 12 | IE 6.0 | 4 | 4 | 3 | 3 |
| 13 | IE 6.0 | RealPlayer | 5 | 1 | 1 |
| 14 | IE 6.0 | MediaPlayer | 7 | 4 | 4 |
| 15 | IE 6.0 | MediaPlayer | 6 | 4 | 4 |
| 16 | IE 6.0 | RealPlayer | 8 | 1 | 1 |
| 17 | IE 5.5 | MediaPlayer | 1 | 2 | 1 |
| 18 | IE 5.5 | RealPlayer | 3 | 3 | 4 |
| 19 | IE 5.5 | RealPlayer | 2 | 3 | 4 |
| 20 | IE 5.5 | MediaPlayer | 4 | 2 | 1 |
| 21 | IE 5.5 | None | 5 | 4 | 3 |
| 22 | IE 5.5 | 4 | 7 | 1 | 2 |
| 23 | IE 5.5 | 4 | 6 | 1 | 2 |
| 24 | IE 5.5 | None | 8 | 4 | 3 |
| 25 | Net 6.0 | RealPlayer | 1 | 4 | 3 |
| 26 | Net 6.0 | MediaPlayer | 3 | 1 | 2 |
| 27 | Net 6.0 | MediaPlayer | 2 | 1 | 2 |
| 28 | Net 6.0 | RealPlayer | 4 | 4 | 3 |
| 29 | Net 6.0 | 4 | 5 | 2 | 1 |
| 30 | Net 6.0 | None | 7 | 3 | 4 |
| 31 | Net 6.0 | None | 6 | 3 | 4 |
| 32 | Net 6.0 | 4 | 8 | 2 | 1 |
| 33 | Net 6.1 | RealPlayer | 1 | 4 | 2 |
| 34 | Net 6.1 | MediaPlayer | 3 | 1 | 3 |
| 35 | Net 6.1 | MediaPlayer | 2 | 1 | 3 |
| 36 | Net 6.1 | RealPlayer | 4 | 4 | 2 |
| 37 | Net 6.1 | 4 | 5 | 2 | 4 |
| 38 | Net 6.1 | None | 7 | 3 | 1 |
| 39 | Net 6.1 | None | 6 | 3 | 1 |
| 40 | Net 6.1 | 4 | 8 | 2 | 4 |
| 41 | Moz 1.1 | MediaPlayer | 1 | 2 | 4 |
| 42 | Moz 1.1 | RealPlayer | 3 | 3 | 1 |
| 43 | Moz 1.1 | RealPlayer | 2 | 3 | 1 |
| 44 | Moz 1.1 | MediaPlayer | 4 | 2 | 4 |
| 45 | Moz 1.1 | None | 5 | 4 | 2 |
| 46 | Moz 1.1 | 4 | 7 | 1 | 3 |
| 47 | Moz 1.1 | 4 | 6 | 1 | 3 |
| 48 | Moz 1.1 | None | 8 | 4 | 2 |
| 49 | Net 7.0 | 4 | 1 | 3 | 2 |
| 50 | Net 7.0 | None | 3 | 2 | 3 |
| 51 | Net 7.0 | None | 2 | 2 | 3 |
| 52 | Net 7.0 | 4 | 4 | 3 | 2 |
| 53 | Net 7.0 | RealPlayer | 5 | 1 | 4 |
| 54 | Net 7.0 | MediaPlayer | 7 | 4 | 1 |
| 55 | Net 7.0 | MediaPlayer | 6 | 4 | 1 |
| 56 | Net 7.0 | RealPlayer | 8 | 1 | 4 |
| 57 | Opera 7 | None | 1 | 1 | 4 |
| 58 | Opera 7 | 4 | 3 | 4 | 1 |
| 59 | Opera 7 | 4 | 2 | 4 | 1 |
| 60 | Opera 7 | None | 4 | 1 | 4 |
| 61 | Opera 7 | MediaPlayer | 5 | 3 | 2 |
| 62 | Opera 7 | RealPlayer | 7 | 2 | 3 |
| 63 | Opera 7 | RealPlayer | 6 | 2 | 3 |
| 64 | Opera 7 | MediaPlayer | 8 | 3 | 2 |

Now that the first and second columns have been mapped, let's proceed to map the next three columns simultaneously.

The mapping for Client operating system is:

* + 1 ↔Windows 95
  + 2 ↔ Windows 98
  + 3 ↔ Windows ME
  + 4 ↔ Windows NT
  + 5 ↔ Windows 2000
  + 6 ↔ Windows XP
  + 7 ↔ Not used (at this time)
  + 8 ↔ Not used (at this time)

The mapping for Servers is:

* + 1 ↔ IIS
  + 2 ↔ Apache
  + 3 ↔ WebLogic
  + 4 ↔ Not used (at this time)

The mapping for Server operating system is:

* + 1 ↔ Windows NT
  + 2 ↔ Windows 2000
  + 3 ↔ Linux
  + 4 ↔ Not used (at this time)

Filling in the remainder of the columns gives:

| Table 6-9: L64(8243) with a full mapping of all its columns.  [[http://www.books24x7.com/images/b24-bluearrow.gif](http://www.books24x7.com/outputobject.asp?bookid=7898&chunkid=362579270&objectid=ch06table09&objecttype=spreadsheet)Open table as spreadsheet](http://www.books24x7.com/outputobject.asp?bookid=7898&chunkid=362579270&objectid=ch06table09&objecttype=spreadsheet) | | | | | |
| --- | --- | --- | --- | --- | --- |
|  | **Browser** | **Plug-in** | **Client OS** | **Server** | **Server OS** |
| 1 | IE 5.0 | None | Win 95 | IIS | Win NT |
| 2 | IE 5.0 | 4 | Win ME | 4 | 4 |
| 3 | IE 5.0 | 4 | Win 98 | 4 | 4 |
| 4 | IE 5.0 | None | Win NT | IIS | Win NT |
| 5 | IE 5.0 | MediaPlayer | Win 2000 | WebLogic | Linux |
| 6 | IE 5.0 | RealPlayer | 7 | Apache | Win 2000 |
| 7 | IE 5.0 | RealPlayer | Win XP | Apache | Win 2000 |
| 8 | IE 5.0 | MediaPlayer | 8 | WebLogic | Linux |
| 9 | IE 6.0 | 4 | Win 95 | WebLogic | Linux |
| 10 | IE 6.0 | None | Win ME | Apache | Win 2000 |
| 11 | IE 6.0 | None | Win 98 | Apache | Win 2000 |
| 12 | IE 6.0 | 4 | Win NT | WebLogic | Linux |
| 13 | IE 6.0 | RealPlayer | Win 2000 | IIS | Win NT |
| 14 | IE 6.0 | MediaPlayer | 7 | 4 | 4 |
| 15 | IE 6.0 | MediaPlayer | Win XP | 4 | 4 |
| 16 | IE 6.0 | RealPlayer | 8 | US | Win NT |
| 17 | IE 5.5 | MediaPlayer | Win 95 | Apache | Win NT |
| 18 | IE 5.5 | RealPlayer | Win ME | WebLogic | 4 |
| 19 | IE 5.5 | RealPlayer | Win 98 | WebLogic | 4 |
| 20 | IE 5.5 | MediaPlayer | Win NT | Apache | Win NT |
| 21 | IE 5.5 | None | Win 2000 | 4 | Linux |
| 22 | IE 5.5 | 4 | 7 | IIS | Win 2000 |
| 23 | IE 5.5 | 4 | Win XP | IIS | Win 2000 |
| 24 | IE 5.5 | None | 8 | 4 | Linux |
| 25 | Net 6.0 | RealPlayer | Win 95 | 4 | Linux |
| 26 | Net 6.0 | MediaPlayer | Win ME | IIS | Win 2000 |
| 27 | Net 6.0 | MediaPlayer | Win 98 | IIS | Win 2000 |
| 28 | Net 6.0 | RealPlayer | Win NT | 4 | Linux |
| 29 | Net 6.0 | 4 | Win 2000 | Apache | Win NT |
| 30 | Net 6.0 | None | 7 | WebLogic | 4 |
| 31 | Net 6.0 | None | Win XP | WebLogic | 4 |
| 32 | Net 6.0 | 4 | 8 | Apache | Win NT |
| 33 | Net 6.1 | RealPlayer | Win 95 | 4 | Win 2000 |
| 34 | Net 6.1 | MediaPlayer | Win ME | IIS | Linux |
| 35 | Net 6.1 | MediaPlayer | Win 98 | IIS | Linux |
| 36 | Net 6.1 | RealPlayer | Win NT | 4 | Win 2000 |
| 37 | Net 6.1 | 4 | Win 2000 | Apache | 4 |
| 38 | Net 6.1 | None | 7 | WebLogic | Win NT |
| 39 | Net 6.1 | None | Win XP | WebLogic | 1 Win NT |
| 40 | Net 6.1 | 4 | 8 | Apache | 4 |
| 41 | Moz 1.1 | MediaPlayer | Win 95 | Apache | 4 |
| 42 | Moz 1.1 | RealPlayer | Win ME | WebLogic | Win NT |
| 43 | Moz 1.1 | RealPlayer | Win 98 | WebLogic | Win NT |
| 44 | Moz 1.1 | MediaPlayer | Win NT | Apache | 4 |
| 45 | Moz 1.1 | None | Win 2000 | 4 | Win 2000 |
| 46 | Moz 1.1 | 4 | 7 | IIS | Linux |
| 47 | Moz 1.1 | 4 | Win XP | IIS | Linux |
| 48 | Moz 1.1 | None | 8 | 4 | Win 2000 |
| 49 | Net 7.0 | 4 | Win 95 | WebLogic | Win 2000 |
| 50 | Net 7.0 | None | Win ME | Apache | Linux |
| 51 | Net 7.0 | None | Win 98 | Apache | Linux |
| 52 | Net 7.0 | 4 | Win NT | WebLogic | Win 2000 |
| 53 | Net 7.0 | RealPlayer | Win 2000 | IIS | 4 |
| 54 | Net 7.0 | MediaPlayer | 7 | 4 | Win NT |
| 55 | Net 7.0 | MediaPlayer | Win XP | 4 | Win NT |
| 56 | Net 7.0 | RealPlayer | 8 | IIS | 4 |
| 57 | Opera 7 | None | Win 95 | IIS | 4 |
| 58 | Opera 7 | 4 | Win ME | 4 | Win NT |
| 59 | Opera 7 | 4 | Win 98 | 4 | Win NT |
| 60 | Opera 7 | None | Win NT | IIS | 4 |
| 61 | Opera 7 | MediaPlayer | Win 2000 | WebLogic | Win 2000 |
| 62 | Opera 7 | RealPlayer | 7 | Apache | Linux |
| 63 | Opera 7 | RealPlayer | Win XP | Apache | Linux |
| 64 | Opera 7 | MediaPlayer | 8 | WebLogic | Win 2000 |

Were it not for the few cells that remain unassigned, the mapping of the orthogonal array, and thus the selection of the test cases, would be completed. What about the unassigned cells—first, why do they exist?; second, what should be done with them?

The unassigned cells exist because the orthogonal array chosen was "too big." The perfect size would be an 816133 array; that is, one column that varies from 1 to 8; one column that varies from 1 to 6; and three columns that vary from 1 to 3. Unfortunately, that specific size orthogonal array does not exist. Orthogonal arrays cannot be constructed for any arbitrary size parameters. They come in fixed, "quantum" sizes. You can construct one "this big"; you can construct one "that big"; but you cannot necessarily construct one in-between. Famous Software Tester Mick Jagger gives excellent advice regarding this, "You can't always get what you want, But if you try sometimes, You just might find, you get what you need."

**Famous Software Tester**



If the perfect size array does not exist, choose one that is slightly bigger and apply these two rules to deal with the "excess." The first rule deals with extra columns. If the orthogonal array chosen has more columns than needed for a particular test scenario, simply delete them. The array will remain orthogonal. The second rule deals with extra values for a variable. In the current example, column 3 runs from 1 to 8 but only 1 through 6 is needed. It is tempting to delete the rows that contain these cells but DON'T. The "orthogonalness" may be lost. Each row in the array exists to provide at least one pair combination that appears nowhere else in the array. If you delete a row, you lose that test case. Instead of deleting them, simply convert the extra cells to valid values. Some automated tools randomly choose from the set of valid values for each cell while others choose one valid value and use it in every cell within a column. Either approach is acceptable. Using this second approach, we'll complete the orthogonal array. Note that it may be difficult to maintain the "balanced" aspect of the array when assigning values to these extra cells.

| Table 6-10: L64 (8243) with a full mapping of all its columns including the "extra" cells.  [[http://www.books24x7.com/images/b24-bluearrow.gif](http://www.books24x7.com/outputobject.asp?bookid=7898&chunkid=362579270&objectid=ch06table10&objecttype=spreadsheet)Open table as spreadsheet](http://www.books24x7.com/outputobject.asp?bookid=7898&chunkid=362579270&objectid=ch06table10&objecttype=spreadsheet) | | | | | |
| --- | --- | --- | --- | --- | --- |
|  | **Browser** | **Plug-in** | **Client OS** | **Server** | **Server OS** |
| 1 | IE 5.0 | None | Win 95 | IIS | Win NT |
| 2 | IE 5.0 | None | Win ME | IIS | Win NT |
| 3 | IE 5.0 | None | Win 98 | IIS | Win NT |
| 4 | IE 5.0 | None | Win NT | IIS | Win NT |
| 5 | IE 5.0 | MediaPlayer | Win 2000 | WebLogic | Linux |
| 6 | IE 5.0 | RealPlayer | Win 95 | Apache | Win 2000 |
| 7 | IE 5.0 | RealPlayer | Win XP | Apache | Win 2000 |
| 8 | IE 5.0 | MediaPlayer | Win 98 | WebLogic | Linux |
| 9 | IE 6.0 | None | Win 95 | WebLogic | Linux |
| 10 | IE 6.0 | None | Win ME | Apache | Win 2000 |
| 11 | IE 6.0 | None | Win 98 | Apache | Win 2000 |
| 12 | IE 6.0 | None | Win NT | WebLogic | Linux |
| 13 | IE 6.0 | RealPlayer | Win 2000 | IIS | Win NT |
| 14 | IE 6.0 | MediaPlayer | Win 95 | IIS | Win NT |
| 15 | IE 6.0 | MediaPlayer | Win XP | IIS | Win NT |
| 16 | IE 6.0 | RealPlayer | Win 98 | IIS | Win NT |
| 17 | IE 5.5 | MediaPlayer | Win 95 | Apache | Win NT |
| 18 | IE 5.5 | RealPlayer | Win ME | WebLogic | Win NT |
| 19 | IE 5.5 | RealPlayer | Win 98 | WebLogic | Win NT |
| 20 | IE 5.5 | MediaPlayer | Win NT | Apache | Win NT |
| 21 | IE 5.5 | None | Win 2000 | IIS | Linux |
| 22 | IE 5.5 | None | Win 95 | IIS | Win 2000 |
| 23 | IE 5.5 | None | Win XP | IIS | Win 2000 |
| 24 | IE 5.5 | None | Win 98 | IIS | Linux |
| 25 | Net 6.0 | RealPlayer | Win 95 | IIS | Linux |
| 26 | Net 6.0 | MediaPlayer | Win ME | IIS | Win 2000 |
| 27 | Net 6.0 | MediaPlayer | Win 98 | IIS | Win 2000 |
| 28 | Net 6.0 | RealPlayer | Win NT | IIS | Linux |
| 29 | Net 6.0 | None | Win 2000 | Apache | Win NT |
| 30 | Net 6.0 | None | Win 95 | WebLogic | Win NT |
| 31 | Net 6.0 | None | Win XP | WebLogic | Win NT |
| 32 | Net 6.0 | None | Win 98 | Apache | Win NT |
| 33 | Net 6.1 | RealPlayer | Win 95 | IIS | Win 2000 |
| 34 | Net 6.1 | MediaPlayer | Win ME | IIS | Linux |
| 35 | Net 6.1 | MediaPlayer | Win 98 | IIS | Linux |
| 36 | Net 6.1 | RealPlayer | Win NT | IIS | Win 2000 |
| 37 | Net 6.1 | None | Win 2000 | Apache | Win NT |
| 38 | Net 6.1 | None | Win 95 | WebLogic | Win NT |
| 39 | Net 6.1 | None | Win XP | WebLogic | Win NT |
| 40 | Net 6.1 | None | Win 98 | Apache | Win NT |
| 41 | Moz 1.1 | MediaPlayer | Win 95 | Apache | Win NT |
| 42 | Moz 1.1 | RealPlayer | Win ME | WebLogic | Win NT |
| 43 | Moz 1.1 | RealPlayer | Win 98 | WebLogic | Win NT |
| 44 | Moz 1.1 | MediaPlayer | Win NT | Apache | Win NT |
| 45 | Moz 1.1 | None | Win 2000 | IIS | Win 2000 |
| 46 | Moz 1.1 | None | Win 95 | IIS | Linux |
| 47 | Moz 1.1 | None | Win XP | IIS | Linux |
| 48 | Moz 1.1 | None | Win 98 | IIS | Win 2000 |
| 49 | Net 7.0 | None | Win 95 | WebLogic | Win 2000 |
| 50 | Net 7.0 | None | Win ME | Apache | Linux |
| 51 | Net 7.0 | None | Win 98 | Apache | Linux |
| 52 | Net 7.0 | None | Win NT | WebLogic | Win 2000 |
| 53 | Net 7.0 | RealPlayer | Win 2000 | IIS | Win NT |
| 54 | Net 7.0 | MediaPlayer | Win 95 | IIS | Win NT |
| 55 | Net 7.0 | MediaPlayer | Win XP | IIS | Win NT |
| 56 | Net 7.0 | RealPlayer | Win 98 | IIS | Win NT |
| 57 | Opera 7 | None | Win 95 | IIS | Win NT |
| 58 | Opera 7 | None | Win ME | IIS | Win NT |
| 59 | Opera 7 | None | Win 98 | IIS | Win NT |
| 60 | Opera 7 | None | Win NT | IIS | Win NT |
| 61 | Opera 7 | MediaPlayer | Win 2000 | WebLogic | Win 2000 |
| 62 | Opera 7 | RealPlayer | Win 95 | Apache | Linux |
| 63 | Opera 7 | RealPlayer | Win XP | Apache | Linux |
| 64 | Opera 7 | MediaPlayer | Win 98 | WebLogic | Win 2000 |

1. **Construct the test cases.**

Now, all that remains is to construct a test case for each row in the orthogonal array. Note that the array specifies only the input conditions. An oracle (usually the tester) is required to determine the expected result for each test.

### Allpairs Algorithm

Using orthogonal arrays is one way to identify all the pairs. A second way is to use an algorithm that generates the pairs directly without resorting to an "external" device like an orthogonal array.

**Reference**

James Bach provides a tool to generate all pairs combinations at <http://www.satisfice.com>. Click on Test Methodology and look for Allpairs.

Ward Cunningham provides further discussion and the source code for a Java program to generate all pairs combinations at <http://fit.c2.com/wiki.cgi?AllPairs>.

James Bach presents an algorithm to generate all pairs in [*Lessons Learned in Software Testing*](http://www.books24x7.com/assetviewer.aspx?bkid=18448&destid=0#0). In addition, he provides a program called "Allpairs" that will generate the all pairs combinations. It is available at <http://www.satisfice.com>. Click on "Test Methodology" and look for Allpairs. Let's apply the Allpairs algorithm to the previous Web site testing problem.

After downloading and unzipping, to use Allpairs create a tab-delimited table of the variables and their values. If you are a Windows user, the easiest way is to launch Excel, enter the data into the spreadsheet, and then SaveAs a .txt file. The following table was created and saved as input.txt.

| Table 6-11: Input to the Allpairs program.  [[http://www.books24x7.com/images/b24-bluearrow.gif](http://www.books24x7.com/outputobject.asp?bookid=7898&chunkid=362579270&objectid=ch06table11&objecttype=spreadsheet)Open table as spreadsheet](http://www.books24x7.com/outputobject.asp?bookid=7898&chunkid=362579270&objectid=ch06table11&objecttype=spreadsheet) | | | | |
| --- | --- | --- | --- | --- |
| **Browser** | **Client OS** | **Plug-in** | **Server** | **Server OS** |
| **IE 5.0** | **Win 95** | **None** | **IIS** | **Win NT** |
| **IE 5.5** | **Win 98** | **Real Player** | **Apache** | **Win 2000** |
| **IE 6.0** | **Win ME** | **Media Player** | **WebLogic** | **Linux** |
| **Netscape 6.0** | **Win NT** |  |  |  |
| **Netscape 6.1** | **Win 2000** |  |  |  |
| **Netscape 7.0** | **Win XP** |  |  |  |
| **Mozilla 1.1** |  |  |  |  |
| **Opera 7** |  |  |  |  |

Then run the Allpairs program by typing: allpairs input.txt > output.txt

where output.txt will contain the list of all pairs test cases. The following table was created:

| Table 6-12: Output from the Allpairs program.  [[http://www.books24x7.com/images/b24-bluearrow.gif](http://www.books24x7.com/outputobject.asp?bookid=7898&chunkid=362579270&objectid=ch06table12&objecttype=spreadsheet)Open table as spreadsheet](http://www.books24x7.com/outputobject.asp?bookid=7898&chunkid=362579270&objectid=ch06table12&objecttype=spreadsheet) | | | | | |
| --- | --- | --- | --- | --- | --- |
|  | **Browser** | **Client OS** | **Plug-in** | **Server** | **Server OS** |
| **1** | **IE 5.0** | **Win 95** | **None** | **IIS** | **Win NT** |
| **2** | **IE 5.0** | **Win 98** | **Real Player** | **Apache** | **Win 2000** |
| **3** | **IE 5.0** | **Win ME** | **Media Player** | **WebLogic** | **Linux** |
| **4** | **IE 5.5** | **Win 95** | **Real Player** | **WebLogic** | **Win NT** |
| **5** | **IE 5.5** | **Win 98** | **None** | **IIS** | **Linux** |
| **6** | **IE 5.5** | **Win ME** | **None** | **Apache** | **Win 2000** |
| **7** | **IE 6.0** | **Win 95** | **Media Player** | **Apache** | **Linux** |
| **8** | **IE 6.0** | **Win 98** | **Real Player** | **IIS** | **Win NT** |
| **9** | **IE 6.0** | **Win ME** | **None** | **WebLogic** | **Win 2000** |
| **10** | **Netscape 6.0** | **Win ME** | **Real Player** | **IIS** | **Linux** |
| **11** | **Netscape 6.0** | **Win NT** | **Media Player** | **IIS** | **Win 2000** |
| **12** | **Netscape 6.0** | **Win 2000** | **None** | **Apache** | **Win NT** |
| **13** | **Netscape 6.1** | **Win NT** | **None** | **WebLogic** | **Linux** |
| **14** | **Netscape 6.1** | **Win 2000** | **Media Player** | **IIS** | **Win 2000** |
| **15** | **Netscape 6.1** | **Win XP** | **Real Player** | **Apache** | **Win NT** |
| **16** | **Netscape 7.0** | **Win NT** | **Real Player** | **Apache** | **Win NT** |
| **17** | **Netscape 7.0** | **Win 2000** | **Media Player** | **WebLogic** | **Linux** |
| **18** | **Netscape 7.0** | **Win XP** | **Media Player** | **IIS** | **Win 2000** |
| **19** | **Mozilla 1.1** | **Win XP** | **Media Player** | **WebLogic** | **Win NT** |
| **20** | **Mozilla 1.1** | **Win 98** | **Media Player** | **Apache** | **Linux** |
| **21** | **Mozilla1.1** | **Win 95** | **Real Player** | **IIS** | **Win 2000** |
| **22** | **Opera 7** | **Win XP** | **None** | **WebLogic** | **Linux** |
| **23** | **Opera 7** | **Win 98** | **Real Player** | **WebLogic** | **Win 2000** |
| **24** | **Opera 7** | **Win ME** | **Media Player** | **Apache** | **Win NT** |
| **25** | **IE 5.5** | **Win 2000** | **Real Player** | **~WebLogic** | **~Linux** |
| **26** | **IE 5.5** | **Win NT** | **Media Player** | **~IIS** | **~Win NT** |
| **27** | **Netscape 6.0** | **Win 95** | **~None** | **WebLogic** | **~Win 2000** |
| **28** | **Netscape 7.0** | **Win 95** | **None** | **~Apache** | **~Linux** |
| **29** | **Mozilla 1.1** | **Win ME** | **None** | **~IIS** | **~Win NT** |
| **30** | **Opera 7** | **Win NT** | **~Real Player** | **IIS** | **~Linux** |
| **31** | **IE 5.0** | **Win NT** | **~None** | **~Apache** | **~Win 2000** |
| **32** | **IE 5.0** | **Win 2000** | **~Real Player** | **~IIS** | **~Win NT** |
| **33** | **IE 5.0** | **Win XP** | **~None** | **~WebLogic** | **~Linux** |
| **34** | **IE 5.5** | **Win XP** | **~Real Player** | **~Apache** | **~Win 2000** |
| **35** | **IE 6.0** | **Win 2000** | **~None** | **~Apache** | **~Win 2000** |
| **36** | **IE 6.0** | **Win NT** | **~Real Player** | **~WebLogic** | **~Win NT** |
| **37** | **IE 6,0** | **Win XP** | **~Media Player** | **~IIS** | **~Linux** |
| **38** | **Netscape 6.0** | **Win 98** | **~Media Player** | **~WebLogic** | **~Win NT** |
| **39** | **Netscape 6.0** | **Win XP** | **~Real Player** | **~Apache** | **~Linux** |
| **40** | **Netscape 6.1** | **Win 95** | **~Media Player** | **~Apache** | **~Win 2000** |
| **41** | **Netscape 6.1** | **Win 98** | **~None** | **~IIS** | **~Win NT** |
| **42** | **Netscape 6.1** | **Win ME** | **~Real Player** | **~WebLogic** | **~Linux** |
| **43** | **Netscape 7.0** | **Win 98** | **~None** | **~WebLogic** | **~Win 2000** |
| **44** | **Netscape 7.0** | **Win ME** | **~Real Player** | **-US** | **~Win NT** |
| **45** | **Mozilla 1.1** | **Win NT** | **~None** | **~Apache** | **~Linux** |
| **46** | **Mozilla 1.1** | **Win 2000** | **~Real Player** | **~WebLogic** | **~Win 2000** |
| **47** | **Opera 7** | **Win 95** | **~Media Player** | **~IIS** | **~Win NT** |
| **48** | **Opera 7** | **Win 2000** | **~None** | **~Apache** | **~Win 2000** |

When a particular value in the test case doesn't matter, because all of its pairings have already been selected, it is marked with a ~. Bach's algorithm chooses the value that has been paired the fewest times relative to the others in the test case. Any other value could be substituted for one prefixed with a ~ and all pairs coverage would still be maintained. This might be done to test more commonly used or more critical combinations more often. In addition, Bach's program displays information on how the pairings were done. It lists each pair, shows how many times that pair occurs in the table, and indicates each test case that contains that pair.

Because of the "balanced" nature of orthogonal arrays, that approach required sixty-four test cases. The "unbalanced" nature of the all pairs selection algorithm requires only forty-eight test cases, a savings of 25 percent.

Note that the combinations chosen by the Orthogonal Array method may not be the same as those chosen by Allpairs. It does not matter. What does matter is that all of the pair combinations of parameters are chosen. Those are the combinations we want to test.

Proponents of the Allpairs algorithm point out that given a problem with 100 parameters, each capable of taking on one of two values, 101 test cases would be required using a (balanced) orthogonal array while the un-balanced all pairs approach requires only ten tests. Since many applications have large numbers of inputs that take on only a few values each, they argue the all pairs approach is superior.

**Tool**

The AETG tool from Telcordia implements the all-pairs testing approach. See <http://aetgweb.argreenhouse.com>.

### Final Comments

In some situations, constraints exist between certain choices of some of the variables. For example, Microsoft's IIS and Apple's MacOS are not compatible. It is certain that the pairwise techniques will choose that combination for test. (Remember, it does select **all** the pairs.) When creating pairwise subsets by hand, honoring these various constraints can be difficult. Both the rdExpert and AETG tools have this ability. You define the constraints and the tool selects pairs meeting those constraints.

Given the two approaches to pairwise testing, orthogonal arrays and the Allpairs algorithm, which is more effective? One expert, who favors orthogonal arrays, believes that the coverage provided by Allpairs is substantially inferior. He notes that the uniform distribution of test points in the domain offers some coverage against faults that are more complex than double-mode faults. Another expert, who favors the Allpairs approach, notes that Allpairs does, in fact, test all the pairs, which is the goal. He claims there is no evidence that the orthogonal array approach detects more defects. He also notes that the Allpairs tool is available free on the Web. What both experts acknowledge is that no documented studies exist comparing the efficacy of one approach over the other.

The exciting hope of pairwise testing is that by creating and running between 1 percent to 20 percent of the tests you will find between 70 percent and 85 percent of the total defects. There is no promise here, only a hope. Many others have experienced this significant result. Try this technique. Discover whether it works for you.

Cohen reported that in addition to reducing the number of test cases and increasing the defect find rate, test cases created by the Allpairs algorithm also provided better code coverage. A set of 300 randomly selected tests achieved 67 percent statement coverage and 58 percent decision coverage while the 200 all pairs test cases achieved 92 percent block coverage and 85 percent decision coverage, a significant increase in coverage with fewer test cases.

One final comment—it is possible that certain important combinations may be missed by both pairwise approaches. The 80:20 rule tells us that combinations are not uniformly important. Use your judgment to determine if certain additional tests should be created for those combinations.

In the previous example we can be assured that the distribution of browsers is not identical. It would be truly amazing if 12.5 percent of our users had IE 5.0, 12.5 percent had IE 5.5, 12.5 percent had IE 6.0, etc. Certain combinations occur more frequently than others. In addition, some combinations exist that, while used infrequently, absolutely positively must work properly—"shut down the nuclear reactor" is a good example. In case pairwise misses an important combination, please add that combination to your test cases.

## Applicability and Limitations

Like other test design approaches previously presented, pairwise testing can significantly reduce the number of test cases that must be created and executed. It is equally applicable at the unit, integration, system, and acceptance test levels. All it requires are combinations of inputs, each taking on various values, that result in a combinatorial explosion, too many combinations to test.

Remember, there is no underlying "software defect physics" that guarantees pairwise testing will be of benefit. There is only one way to know—try it.

## Summary

* When the number of combinations to test is very large, do not to attempt to test all combinations for all the values for all the variables, but test **all pairs** of variables. This significantly reduces the number of tests that must be created and run.
* Studies suggest that most defects are either single-mode defects (the function under test simply does not work) or double-mode defects (the pairing of this function/module with that function/module fails). Pairwise testing defines a minimal subset that guides us to test for all single-mode and double-mode defects. The success of this technique on many projects, both documented and undocumented, is a great motivation for its use.
* An orthogonal array is a two-dimensional array of numbers that has this interesting property—choose any two columns in the array, all the combinations will occur in every column pair.
* There is no underlying "software defect physics" that guarantees pairwise testing will be of benefit. There is only one way to know—try it.

## Practice

1. Neither the Brown & Donaldson nor the Stateless University Registration System case studies contain huge numbers of combinations suitable for the pairwise testing approach. As exercises, use the orthogonal array and/or all pairs technique on the other two examples in this chapter. Determine the set of pairwise test cases using the chosen technique.
   1. A bank has created a new data processing system that is ready for testing. This bank has different kinds of customers—consumers, very important consumers, businesses, and non-profits; different kinds of accounts—checking, savings, mortgages, consumer loans, and commercial loans; they operate in different states, each with different regulations—California, Nevada, Utah, Idaho, Arizona, and New Mexico.
   2. In an object-oriented system, an object of class A can send a message containing a parameter P to an object of class X. Classes B, C, and D inherit from A so they too can send the message. Classes Q, R, S, and T inherit from P so they too can be passed as the parameter. Classes Y and Z inherit from X so they too can receive the message.

## References

Brownlie, Robert, et al. "Robust Testing of AT&T PMX/StarMAIL Using OATS," *AT&T Technical Journal*, Vol. 71, No. 3, May/June 1992, pp. 41–47.

Cohen, D.M., et al. "The AETG System: An Approach to Testing Based on Combinatorial Design." *IEEE Transactions on Software Engineering*, Vol. 23, No. 7, July, 1997.

Kaner, Cem,James Bach, and Bret Pettichord (2002). *Lessons Learned in Software Testing: A Context-Driven Approach*. John Wiley & Sons.

Kuhn, D. Richard and Michael J. Reilly. "An Investigation of the Applicability of Design of Experiments to Software Testing," 27th NASA/IEEE Software Engineering Workshop, NASA Goddard Space Flight Center, 4–6 December, 2002. <http://csrc.nist.gov/staff/kuhn/kuhn-reilly-02.pdf>

Mandl, Robert. "Orthogonal Latin Squares: An Application of Experiment Design to Compiler Testing," *Communications of the ACM*, Vol. 128, No. 10, October 1985, pp. 1054–1058.

Phadke, Madhav S. (1989). *Quality Engineering Using Robust Design*. Prentice-Hall.

Wallace, Delores R. and D. Richard Kuhn. "Failure Modes In Medical Device Software: An Analysis Of 15 Years Of Recall Data," *International Journal of Reliability, Quality, and Safety Engineering*, Vol. 8, No. 4, 2001.