

Orthogonally

Speaking

A method for deriving a suitable set of test cases

by *Elfriede Dustin*

It's a laudable goal: to conduct a thoroughly exhaustive test of a system. In the real world, however, it's generally not possible, feasible, or cost effective to test a system using all the possible variations and combinations of test parameter inputs.

I was reminded of this recently while testing a tax management system. This system contained a calculation engine that computed the depreciation of fixed assets (e.g., computers, airplanes, office furniture) based on user-supplied information about the assets of the company. These types of computations are complex, and sensitive to different combinations of a large number of possible input parameters. Some of these parameters are the "placed in service date" of the fixed asset, methods of depreciation, life of the asset, business use percentages, fixed asset costs, and calendar years—only a few of many possible parameters. Each parameter could have numerous values. As a result, there were tens of thousands of potential input variations, each producing different results, and making exhaustive testing of the calculation engine's computations nearly impossible.

Our test engineers selected test input parameters that were based on high-risk calculations provided by an accountant "subject matter expert" or on issues experienced in the past. Under this approach, the testers could never really measure the completeness and coverage of their test suite. This manual testing process was

mind numbing and error prone. Worse, the software engineers were wary of making any major changes to the calculation engine; they were afraid that any major source code change could result in incorrect computations, compromising the accuracy of the output of the asset management system.

There was no efficient way to test any calculation engine source

▶▶ QUICK LOOK

- **Determining test parameters when exhaustive testing is not practical**
- **Ways to automate an orthogonal test array**
- **5 mistakes to avoid**

code changes; too many of the variables depended on each other.

So how do you derive a suitable set of test cases when it's just not feasible to use all the possible combinations and variations of test parameters? This article addresses one solution: the test technique called the Orthogonal Array Testing System (OATS). Based upon my experience, this technique is very useful for finding a small set of tests (from a large number of possibilities) that exercises key combinations.

The OATS Solution

OATS is derived from manufacturing techniques developed as part of the industrial engineering discipline. Orthogonal arrays (more on the term **orthogonal** in a moment) are used as a mathematical tool in the Robust Design methodology described in Madhan Phadke's *Quality Engineering Using Robust Design* and other books. The Robust Design methodology, created by Professor Genichi Taguchi, is in use

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

TABLE 1 Sample array

TYPE OF CREDIT CARD	CREDIT CARD NUMBER	CREDIT CARD EXPIRATION DATE (YEARS FROM TODAY)	PRODUCT TYPE PURCHASED	QUANTITY PURCHASED
Amex	Correct	50	Book	1
Discover	Incorrect Length	Invalid Year	Video	0
Visa	Invalid Digits	Today	Software	-1
MasterCard		Yesterday	Book, Software, Videos	10
		Invalid Character	Book, Software	1

TABLE 2 Bookstore purchase parameters and values

in many modern areas of engineering.

The OATS technique supports the system test effort by enabling test cases to be determined efficiently and uniformly. With this test technique you'll be able to select the combinations of test parameters that will provide maximum coverage from test procedures, while using a minimum number of test cases. The assumption here is that *tests that maximize the interactions between parameters will find more faults*.

The technique works. In our case, for example, OATS made it possible for my team's tax management developers to change their application's calculation engine with more confidence. They fed automatically generated OATS test parameters into a test harness, which in return exercised the calculation engine. The engine's outputs were captured and became the baseline for any future changes to the calculation engine. This test harness has proven to be very valuable, as it has uncovered many calculation differences caused by calculation engine source code changes. Moreover, the OATS procedure has given us an objective measure of testing completeness.

What Is an Orthogonal Array?

We'll introduce the idea of orthogonal arrays with an example. Suppose there are three parameters (A, B, and C), each of which has one of three possible values (1, 2, or 3). The effort to test all possible combinations involving the three parameters would require twenty-seven test cases.

Do you need all twenty-seven of those tests? Yes, if you think there's a fault that depends on the precise val-

ues of all three parameters (a fault, for example, that would occur only for the case $A=1, B=1, C=1$). But, because of the way programming works, it's probably more likely that a fault will depend on the values of only *two* of the parameters. In that case, the fault might occur for each of these three test cases: $A=1, B=1, C=1$, $A=1, B=1, C=2$, and $A=1, B=1, C=3$. Since the value of C in this example seems to be irrelevant to the occurrence of this particular fault, any one of the three tests will suffice.

Given that assumption, the array in Table 1 shows the nine test cases required to catch all such faults, in the most economical arrangement that will show all possible pairs within all three variables. The array is *orthogonal* because, for each pair of parameters, all combinations of their values occur once. That is, all possible pairwise combinations between parameters A and B, B and C, and C and A are shown. Since we're thinking in terms of pairs, we say this array has a strength of 2. It doesn't have a strength of 3 because not all three-way combinations occur; $A=1, B=2, C=3$, for example, doesn't appear. But it covers the pairwise possibilities, which is what we're concerned about.

Applying the Technique

Before implementing orthogonal array testing, the test engineer needs to determine the size of the array required for the specific system test effort. The size of the orthogonal array is based upon the maximum number of values for all possible parameters. For demonstration purposes, let's look at a simplified example of how we might use OATS to test our favorite tech bookstore's applications.

Table 2 shows example parameters we believe might interact. They are: *Type of Credit Card*, *Credit Card Number*, *Credit Card Expiration Date*, *Product Type Purchased*, and *Quantity Purchased*.

Each parameter has its own possible values that need to be tested in combination with the values of other parameters. The possible values pertaining to the

ID	CREDIT CARD	CREDIT CARD NUMBER	EXPIRATION DATE	PRODUCT	QUANTITY
0	Amex	402901517	2/13/2001	Books	1
1	Amex	123456789	2/15/2001	Software	0
2	Amex	11111111%	5/15/2001	Videos	-1
3	Amex	WER11212p	5/28/2050	Books, Software, Videos	10
4	Amex	542212345	3/16/2001	Books, Software	1
5	Discover	402901517	2/15/2001	Videos	10
6	Discover	123456789	5/15/2001	Books, Software, Videos	1
7	Discover	11111111%	5/28/2050	Books, Software	1
8	Discover	WER11212p	3/10/2001	Book	0
9	Discover	542212345	2/13/2001	Software	-1
10	Visa	402901517	5/15/2001	Books, Software	0
11	Visa	123456789	5/28/2050	Books	-1
12	Visa	11111111%	2/22/2001	Software	10
13	Visa	WER11212p	2/13/2001	Videos	1
14	Visa	542212345	2/15/2001	Books, Software, Videos	1
15	MasterCard	402901517	5/28/2050	Software	1
16	MasterCard	123456789	3/08/2001	Videos	1
17	MasterCard	11111111%	2/13/2001	Books, Software, Videos	0
18	MasterCard	WER11212p	2/15/2001	Books, Software	-1
19	MasterCard	542212345	5/15/2001	Books	10
20	Visa	402901517	3/26/2001	Books, Software, Videos	-1
21	Visa	123456789	2/13/2001	Books, Software	10
22	Visa	11111111%	2/15/2001	Books	1
23	Visa	WER11212p	5/15/2001	Software	1
24	Visa	542212345	5/28/2050	Videos	0

TABLE 3 Test case definitions

Type of Credit Card used by a customer might include American Express, Discover, Visa, and MasterCard. The *Credit Card Number* entries could be correct or incorrect. All correct numbers are assumed to interact in the same way; that is, if one correct number reveals a fault when combined with a Discover card, all correct numbers will reveal the same fault when combined with a Discover card. However, incorrect numbers are assumed to interact differently, depending on whether they have an incorrect length or some invalid digits.

Once the parameters and the values have been derived, we have to de-

cide how parameters are likely to interact. If only pairwise interactions are likely, the array should have a strength of 2. In this case, it seems reasonable to say that pairwise testing is sufficient (“good enough”) for this type of application testing, so three-way testing doesn’t seem necessary. (Note that with a higher risk application, one might want to consider selecting an array that allows for three-way or *n*-way input parameter testing.)

An orthogonal array tool can be used to produce an orthogonal array such as the one in Table 3. Each resulting row in the orthogonal array specifies one specific test case (without expected results). For example, in row number 0, a test case will be executed using American Express as the credit card, with a credit card number value

of 402901517, with an expiration date of 2/13/2001, involving the purchase of one (1) book. [Author’s note: The credit card numbers used here and in the accompanying table are truncated fictional numbers, so as to avoid any similarity to actual accounts.] Collectively, twenty-five test cases exercise all pairwise combinations. Exhaustive testing would have required 5⁵, or 3125 test cases.

You’ll see that the test cases contain specific values, rather than markers like “incorrect length” or “invalid year.” To construct this example, we used a script that replaced the respective values of the orthogonal array with the actual parameters and values needed for the project.

Note that while OATS is a useful tool, you should consider using addi-

tional testing techniques to derive your data elements when you're determining actual values. Techniques such as boundary value analysis (e.g., selecting the maximum, minimum, one more than maximum, one more than minimum, or zero data) in combination with OATS can be a powerful technique. (It is common for errors to congregate around the boundary values.) In this example, one could pick the *Quantity* 100,000—because that's the maximum number of any item that can be purchased at one time in this example—then try 99,999 (one less than maximum), 100,001 (one more than maximum), etc.

This example also illustrates an issue that often arises: unspecified values. Because there are fewer *Type of Credit Card* and *Credit Card Number* values than there are *Expiration Date* values, not all rows in the orthogonal array are required to exercise all the pairwise combinations involving them. For some of the rows, the value of *Type of Credit Card* or *Credit Card Number* is left to the discretion of the test engineer. The values can be chosen based on risk, highest usage, or highest problem area. Table 4 gives an example. American Express might have been chosen because it has been the card the bookstore's application traditionally has had the most trouble with, or is used most often. A correct *Credit Card Number* might have been chosen because incorrect number input might not have been an issue in the past.

In some cases combinations of test parameters and values can be invalid, and an invalid test case combination is generated (depending on business logic). For example, with three parameters (A, B, and C), it might be invalid for both A and C to have a value of 1, but the OATS tool would still generate that combination. In that case it is up to the test engineer to make a decision. You can execute the test case as is (garbage in and garbage out) and determine how the system handles invalid combinations of input. Or you can decide to not use the invalid test case combinations—in order to shorten the test case evaluation cycle, or in cases where the back-end system doesn't

allow for invalid input combinations. But choose carefully: if you throw out the invalid cases, you might also be throwing out other combinations. For example, the A=1, C=1 case might be the only row containing A=1, B=3 (a valid combination that won't be tested if you throw out the row). And if you throw out the invalid cases, you won't know how the system behaves given these invalid combinations.

In the case study I mentioned earlier, in which we used orthogonal array testing for the calculation engine of the asset management system, we decided to also allow input of invalid test combinations. The calculation engine was expected to produce consistent results among the various builds, whether the input was valid or invalid.

Please note that Table 4 is a simplified excerpt, one that is small and readable, of a sample test case combination. In addition to the parameters illustrated here, the bookstore might also wish to track the number of books that remain in inventory following the purchase, or be able to query the purchase status for a particular customer. Test professionals should review the resulting test cases and add additional test cases based on known risk areas.

In our test program for the asset management calculation engine, we executed a test harness that generated over 17,000 test cases using OATS. A software program was then developed that incorporated these test cases and applied them to the back end of the Web application one by one. (Such a software program might be tailored to create a particular load on the system, or to simply verify baseline functionality.)

Mistakes to Avoid

Although we've used OATS successfully on a variety of projects, we had a false start on a previous project that

taught us several lessons on using the technique effectively. The things we did wrong in that first situation are mistakes you could easily make on your own; let's look at each in turn.

1. Applying OATS manually

We applied OATS manually during our first orthogonal testing effort. We determined the variously sized orthogonal arrays based on our test parameters, and then manually replaced and plugged in the actual values into the table, using an Excel spreadsheet. Doing all this manually was very time consuming, tedious, and prone to inaccuracies—and it required quite a bit of maintenance. We didn't have the extra budget to buy commercial test case generating tools, but had we known then about the availability of orthogonal array freeware, we could have used it to automatically generate various array sizes. These automatically generated arrays could have been used in turn to write a program that automatically replaced the numbers in the table with actual test parameter values.

2. Focusing the testing effort on the wrong area of the application

Some readers might be enthusiastic about implementing orthogonal array testing in their next testing project. While you're exploring those possibilities, temper that eagerness with a sense of perspective. Just because a specific area of the application lends itself perfectly to orthogonal array testing, don't get distracted and shift the whole focus of the testing effort to that area. As with any testing technique, it's important to focus this method on the critical areas. You have to evaluate whether it's feasible to apply OATS testing to a specific area.

For example, one test engineer I know returned from a training class on the OATS technique, impatient to apply it right away. He spent days devising an incredibly elaborate orthogonal array

TYPE OF CREDIT CARD	CREDIT CARD NUMBER	CREDIT CARD EXPIRATION DATE (YEARS FROM TODAY)	PRODUCT TYPE PURCHASED	QUANTITY PURCHASED
Amex	Correct	Invalid Character	Books, Software	1

TABLE 4 Sample combination

testing system to verify possible query combinations applied through a GUI. This specific application used a GUI front end that allowed the user to select various query combinations by selecting parameters on various pull-down menus—basically an in-house modification of a commercial off-the-shelf (or COTS) system that allowed for various application queries. The COTS tool had already been tested thoroughly by the vendor, but in his eagerness to apply his newly learned technique the test engineer lost sight of the big picture. He applied his elaborate OATS scheme to the wrong area of the system: the already-tested COTS front end. It would have been good enough to simply run a variety of GUI tests to verify that the pull-down selections worked and produced correct results. In this case using an elaborate OATS system was excessive, and the testing effort would have been better focused on verifying the accuracy of the SQL queries (the actual code) and on verifying the data migration.

The OATS testing method is a great resource, but it's important to evaluate where in your application testing effort it will be most effective and efficient to apply, and how much of it to use.

3. Using OATS for minimal testing efforts

In some cases OATS might ask for more test cases than is necessary for a “good enough” testing effort or for the testing time and budget allotted. Here again, it's important to evaluate whether the OATS testing technique is appropriate, or whether other testing techniques such as boundary values, equivalence classes, etc., would be sufficient. If the test engineer decides to go forward with the implementation of orthogonal array testing despite timing or budget concerns, it's important to evaluate the OATS-generated test case combinations and omit any excessive test input combinations.

4. Using OATS for high-risk applications

If you are testing medical device systems or any other life-critical applications, don't pin your hopes on orthogonal array testing to do the job all by itself. You'll need to complement this testing technique with other well-established testing techniques. It's important to keep in mind that the goal of OATS is to allow for minimizing the test combination inputs—it's *not* a technique that allows for exhaustive testing.

5. Picking the wrong parameters to combine

When I'm teaching the OATS testing technique, I often ask my audience members to come up with an example application on which they think they could apply OATS. I then ask for volunteers to present their example application and how they would specifically apply OATS. The most common issue that surfaces during this exercise is that people pick the wrong parameters to combine.

Picking those parameters can be an elaborate effort—especially when a wide variety or huge number of parameter combinations is involved. In the case study of the calculation engine, our search for the right parameters to combine meant doing a detailed analysis, in which we enlisted the help of a subject matter expert, a developer who understood the calculation engine code, and a test engineer.

Summary

Exhaustive testing in most cases is not possible, feasible, or cost effective. The effort to include all possible combinations and variations of test input parameters is generally an impossible undertaking for complex

PERSPECTIVE

Test Coverage without Missing a Beat

Warren Campbell knows the importance of making your entrance on time. He was, after all, a professional opera singer before switching from on-stage vocalizing to developing voice-interactive systems for DECvoice platforms. So when he was assigned to work on a pharmaceutical application two years ago with a crucial product rollout date, he took the schedule constraints seriously.

The problem was that testing the product to meet exacting regulatory standards was going to be impossible in the time allotted. Fortunately, he recalls, two things were working in his team's favor. First, there were still developers around who had designed the original legacy portion of the software, and second, Campbell had been reading up on orthogonal array testing and was eager to try it. “Verification and validation standards for pharmaceutical projects are incredibly rigorous,” he says, “and OATS seemed to be

a way to demonstrate thorough test coverage in a short time period.”

Based on advice from the original engineers, Campbell's team quickly targeted key functionalities and laid out an efficient testing array. “Things came together,” he says. “We didn't necessarily reduce the total amount of testing, but OATS helped cut down on the *complexity* of the testing.” They weighted the array according to risk and core functionalities—and got their testing done two weeks ahead of schedule. “The alternative to using this method would have been chaos and major delays,” says Campbell. “Instead, we had coverage that would stand up to a tough regulatory inspection, and had it early!”

Although Campbell has seen his share of projects over the years that couldn't really take advantage of an OATS approach—because of a lack of traceability, change management, thorough design processes, or clear communication—his success on that first orthogonal project made him a believer in the approach, and, he says, has been key to several of his projects staying on pitch.

—A.W.

systems, and an attempt to do so can result in scope creep. One common practice in situations such as these is to guess, using engineering judgment, on which test parameters to choose. Shooting in the dark like that can yield hit-and-miss results—and that's just not very effective. When your testing effort faces an impossibly wide choice of test parameters,

the OATS technique can be a very useful method for deriving a suitable set of test cases. [STQE](#)

Elfriede Dustin (edustin@bna.com) works as a QA/Test Manager at *BNA Software* (www.bnasoftware.com). *Elfriede* is co-author of the book *Automated Software Testing*, and also co-authored the recently

published book *Quality Web Systems. Her Automated Software Testing white papers are posted on* www.stickyminds.com.

<p><i>STQE</i> magazine is produced by STQE Publishing, a division of Software Quality Engineering.</p>
