



New Spreadsheet Tool Helps Determine Minimal Set of Test Parameter Combinations

Gregory T. Daich

Software Technology Support Center

Combinatorial testing is a method for identifying incorrect interactions between various parameters called test factors, usually with a goal to run a minimum number of tests. "Give us your tired, your poor, your huddled ... (testers) yearning to breathe free" [1] from executing endless and senseless combinations of test cases. This article explains how to minimize test parameter combinations using a new spreadsheet tool called ReduceArray2.

Your manager assigns you a new testing project. "I want you to take over the system integration testing of the Web Time Charging System (TCS). We've got three weeks to get it out the door and we're concerned about the integration of all the Web-TCS components."

The neural cogs in your head start churning. You know the Web TCS must run on your standard Brand X and Brand Y central processing units and the company's current operating systems (OS): Win 98, Win NT, Win 2000, and Win XP. Each of these platforms must support Microsoft Internet Explorer Version 5.5 and 6.0 and Netscape Version 7.0.

Your manager interrupts your daze and says, "And I don't have to remind you about what that last delivered bug cost us."

"What? Oh yeah! I'll get right on it," you profess, while your manager hurries off to another meeting.

The Web TCS has two operational network modes: internal intranet and modem remote. Employees can log their time in both modes. Various default parameters are established depending on the user's type of employee classification, including salaried, hourly, part-time, or contractor. These parameters include default shift, available paid holidays, etc. Also, the user can set the time increment in minutes to six, 10, 15, 30, or 60.

These features and parameters will be combined into various test configurations, however, one key question is, "What is the most effective, smallest set of test configurations that will find the majority of serious parameter interaction defects?" (I'll answer that soon.)

The TCS is defined by five principal use cases¹. System-level test scenarios will be defined to exercise each use-case in the various test configurations chosen. The use-cases are listed as follows:

- 1-Login
- 2-Log Time
- 3-Submit Time Sheet
- 4-Maintain Charge Codes
- 5-Select Time Period

The test group has already defined 15 test scenarios to use to test each test configuration. Test scenarios for the Login use-case include (1) successful login on first attempt, (2) successful login after one failed attempt, and (3) unsuccessful login after three failed attempts. Twelve similar test scenarios were defined for the other four use-cases.

Management has expressed concern about integration defects delivered in recent releases. Any seriously defective interactions between features and various user-assigned and system configuration parameters could prove fatal to the Web TCS upgrade effort and the future of your group (you've heard this before). At any rate, you need to test each parameter paired with every other parameter to be sure that there are no incompatibilities. I will discuss a simple, straightforward approach for obtaining or getting very close to a minimum set of test configurations that the reader will be able to immediately use on his or her project.

This approach or technique will answer the question posed earlier, "What is the most effective, smallest set of test configurations that will find the majority of serious parameter interaction defects?" Notice the qualification "majority of serious ... defects." Remember that no amount of testing can find all defects. However, most people accept as self-evident that effective testing techniques can lead to increased confidence and to fewer delivered defects and happier customers. This does not mean that these techniques should displace other effective and efficient means for improving or assuring the quality of the system.

What Have We Got?

Essentially, there are six parameters called test factors that are of most interest from a system integration testing perspective. Table 1 lists the six test factors with their associated options. If all combinations of these factors were tested, that would require the following:

2x4x3x2x4x5 = 960 test configurations

Since each test configuration requires 15 system-level test scenarios, the result is a total of $960 \times 15 = 14,400$ test scenarios that must be executed. There is not time to execute all 14,400 test-scenarios in three weeks. Say that it takes about three hours to execute the 15 test scenarios for each configuration, which includes setup and reporting. If you consider that there are about six hours per day of productive test execution time, not counting unpaid overtime that you covertly plan to minimize, that gives you 90 hours or 30 test configurations that you have time to perform. Is there hope? Can you test all important combinations of parameters in less than 30 test configurations?

One approach in minimizing the number of test configurations that some organizations use is to test the most common – or important – configuration and then vary one or more parameters for the next test configuration and then test that. Looking at Table 1, you can see that a minimum of five test configurations (rows) are required to test all options at least once. Just look at each row and pick the assigned values. For columns that have dashes, use the preceding value. However, the concern is to test for possible bad interactions between parameters. This method will not be adequate. Madhav Phadke and others have focused on combinatorial testing techniques that arguably "have the highest effectiveness, measured in terms of the number of faults detected per test" [3].

Identifying a minimum set of tests that check each parameter interacting with every other parameter (i.e., all pairs of parameters) is often a very difficult venture if pursued in an ad hoc, non-systematic fashion. Orthogonal arrays (OAs) provide a systematic means for identifying a minimal set of highly effective tests. Unfortunately, some training in combinatorial software testing techniques available in the industry today is not very helpful in teaching this. But before discussing how to use OAs effectively to

find that minimal set, I need to outline a basic fault model of interest when conducting integration testing, and introduce a little terminology.

Basic Fault Model

Jeremy Harrell published a technique that he calls the Orthogonal Array Testing Strategy (OATS) for manually computing a set of tests from published OAs that is very effective [4]. He does a good job of characterizing a basic fault model that is the foundation for using the OATS technique, which is as follows [4]:

- Interactions and integrations are a major source of defects.
- Most ... defects are not a result of complex interactions such as, "When the background is blue and the font is Arial and the layout has menus on the right and the images are large and it's a Thursday then the tables don't line up properly." Most of these defects arise from simple pair-wise interactions such as, "When the font is Arial and the menus are on the right the tables don't line up properly."
- With so many possible combinations of components or settings, it is easy to miss one.
- Randomly selecting values to create all of the pair-wise combinations is bound to create inefficient test sets and test sets with random, senseless distribution of values.

Phadke adds to this basic fault model with his discussion about the following testing techniques [3]:

- **One-Factor-at-a-Time Testing.** This method varies one factor at a time and would require more than the minimum five tests mentioned earlier. But this makes it easier to identify the defective parameter. However, this technique does not expect to encounter any bad interactions between the given parameters since it does not attempt to cover all the pairs of parameters. It only finds what Phadke calls single-mode faults.
- **Exhaustive Testing.** For any non-trivial system, this will not be possible. Even if all 960 test configurations were tested, which would find nearly every bad interaction between the given parameters, there will be many more tests with varying circumstances that could be conceived that could take a lifetime and more to conduct.
- **Deductive Analytical Method.** This method attempts to cover all important paths in the code. Any testing strategy should be augmented by some of this type of testing. In fact, this is one type of testing that developers should con-

A	B	C	D	E	F
CPU	OS	Browser	Network	Type of Employee	Time Increment
Brand Y	NT	IE 6.0	Modem	Salaried	6
Brand X	98	IE 5.5	Internal	Hourly	10
--	2000	NS 7.0	--	Part-Time	15
--	XP	--	--	Contractor	30
--	--	--	--	--	60

Table 1: Test Factors and Options

- duct on their new components prior to integration testing.
- **Random/Intuitive Method.** This is the most common method used by independent test organizations. This method can be very effective at finding defects but the level of coverage is often questionable.
- **Orthogonal Array-Based Testing.** This method finds all double-mode faults that are two parameters conflicting with each other. An example of a double-mode fault is one parameter overshadowing another, inhibiting required processing of that other parameter.

Terminology

Some introductory terms for understanding OAs include the following [4]:

- **Orthogonal Array.** Two-dimensional arrays that possess the interesting quality that by choosing any two columns in the array you receive an even distribution of all the pair-wise combinations of values in the array.
- **Runs.** The number of rows that are the potential test configurations or test cases.
- **Factors.** The number of columns that are variables or parameters of interest.
- **Levels.** The number of options or values for each factor.
- **Strength.** The number of columns it takes to see each option equally often.
- **OAs.** These are named OA(N, s^k, t) or OA(N, s^k) if its strength is two. This indicates an OA with N runs, k factors, s levels, and strength t.
- **Mixed Arrays.** These are named MA(N, s1^{k1}, s2^{k2}, etc.). This indicates a mixed-level, asymmetric OA with N runs, k1 factors at s1 levels, k2 factors at s2 levels, etc., and with strength 2 (assume strength 2 if it is not stated). Note that the number of runs is dependent on the number of factors, levels, and strength. For example, OA(9,4³) means you have 9 runs required to cover 4 test factors with 3 options each (see Table 2). Since the strength is assumed 2, this OA covers all pairs of parameters. OA(64,6⁴,3) means you have 64 runs

required to cover all three-way combinations (strength 3) of six test factors with four options each. Look these up on the Web or in a book on statistics².

A Powerful Technique

It is not easy to create non-trivial OAs, which are OAs with more than three factors. Furthermore, some currently available automated tools that produce sets of tests covering all pairs of parameters do not create actual OAs or a minimum set of tests. After all, it is a very difficult, discrete mathematics problem to create OAs. These tools may come fairly close to a minimum set but if you can save yourself from having to run even one more test, you are going to want to find that minimal set, especially if it does not require investing any more time. There are tools that can compute OAs or all pair combinations resulting in a minimal set of tests for all reasonable numbers of factors and options but they may cost much more than you may want to spend³.

The OATS technique for manually generating a minimal or near minimal set of tests is actually better than some tools I have seen. In other words it produces a smaller set of tests that exercise all pair-wise combinations of parameters. OATS is simple and straightforward and should be used by a lot more organizations to help with software and system integration testing efforts. Briefly, it includes the following steps [4]:

1. Decide the number factors to test.
2. Decide which options to test for each factor.
3. Find a suitable OA with the smallest number of runs to cover all factors and options.

Table 2: OA(9,4³)

	A	B	C	D
1	0	0	0	0
2	0	1	1	2
3	0	2	2	1
4	1	0	1	1
5	1	1	2	0
6	1	2	0	2
7	2	0	2	2
8	2	1	0	1
9	2	2	1	0

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Array					TP	54	Results						
2	A	B	C	D		TC#	RP	UP	A:B	A:C	A:D	B:C	B:D	C:D
3	0	0	0	0		1	6	6	1	1	1	1	1	1
4	0	1	1	2		2	6	6	1	1	1	1	1	1
5	0	2	2	1		3	6	6	1	1	1	1	1	1
6	1	0	1	1		4	6	6	1	1	1	1	1	1
7	1	1	2	0		5	6	6	1	1	1	1	1	1
8	1	2	0	2		6	6	6	1	1	1	1	1	1
9	2	0	2	2		7	6	6	1	1	1	1	1	1
10	2	1	0	1		8	6	6	1	1	1	1	1	1
11	2	2	1	0		9	6	6	1	1	1	1	1	1
12						SP	54							

TP = Total Pairs (54) RP = Number of New Pairs on a Row (Six on Each Row)
 UP = Unique Pairs (Six on Each Row) A:B = Column A Paired With Column B, etc.
 TC# = Test Case Number SP = Sum of Pairs (54)

Table 3: Results of Analyzing OA(9,4³)

- Map the factors and options onto the array.
- Choose values for any options that are not needed (call them *leftovers*) from the valid remaining options and delete any columns (factors) that are not needed. A given test situation may not need all the factors or all the options that a particular OA provides.
- Transcribe the runs into test cases, adding additional combinations as needed.

The selection of which OA to use can pose a bit of a challenge. The good news is that you do not generally need to compute a new OA for many test situations. As the technique discusses, if there is not a specific OA for your test situation, you can use one that is similar, delete extra factors (columns), and choose values for the extra options consistent with your test situation. In other words, the real work in creating the arrays has already been done. And it only takes a few minutes to apply the array to a specific test

situation. That is pretty powerful, but not enough people know about it.

Techniques vs. Tools

When using the OATS technique to identify a minimal set of tests, you may encounter an OA where each pair appears more than once but the same number of times (evenly). This may be considered overkill from a software testing perspective since all you usually want is to test each pair once. However, identifying the actual minimum set and eliminating any overkill is generally difficult and time consuming without the aid of a tool to quickly and easily see the pair combination counts. Thus, I have prepared an Excel spreadsheet tool called ReduceArray2 that computes the total number of pairs of parameters possible based on the array's factors and options, and it identifies any missing pairs⁴. It counts the number of unique pairs in the array for each row from top to bottom so you can compare it with the total number of pairs. It also computes the number of occur-

rences of each pair in the array and displays them in the spreadsheet in a concise and easily analyzed form.

Table 3 shows the results of analyzing OA(9,4³) using the ReduceArray2 tool. The top row identifies the columns in the spreadsheet. The left column identifies the rows in the spreadsheet. This makes it fairly simple to identify any extra test cases so that rows can be deleted to reduce the size of the array. Extra test cases are rows with no unique pairs. Note that every pair in OA(9,4³) is unique, thus every one is needed. However, the OATS technique will often have values that are not needed (called leftovers) that will create redundant pairs that can be rearranged to create rows with no unique pairs that can then be deleted.

In order to identify a set of tests for the factors and options in Table 1 using the OATS technique, you could use OA(25,5⁶). This would result in 25 tests. Additional rows could be deleted after rearranging some pairs but that would require some fairly labor-intensive study and manual effort without tool support. Also, using one automated tool with which I am familiar produced 26 tests for the test situation in Table 1.

Using ReduceArray2 to assist in finding extra tests, I was able to reduce the set to 20, which is the minimum number of configurations to test all pairs for this situation. The minimum count of 20 was derived from the two factors with the most options. The Time Increment factor has five options and the OS factor has four options. Thus, we know that there must be at least 5 x 4 = 20 runs to cover all combinations of those two options. The trick is to cover all the other pair combinations in those 20 runs. With the visibility provided by the ReduceArray2 tool, this became a much easier task.

Demonstration

The following uses the OATS technique augmented with a few extra steps to identify a minimal set of tests to cover all pairs. To simplify the demonstration and reduce the number of tests, only look at the test factors and options A, C, D, and E in Table 4. The following lists each OATS step with our actions in italics:

- Decide the number of factors to test. *We chose the four test factors in Table 4.*
- Decide which options to test for each factor. *The options for test factors A, C, D, and E are listed in Table 4.*
- Find a suitable OA with the smallest number of runs to cover all factors and options. *A suitable OA was selected that is OA(9,4⁴), see Table 2. A suitable array has at least the number of factors and options with*

Table 4: Subset Test Situation

A	C	D	E
CPU	Browser	Network	Type of Employee
Brand X	IE 5.5	Internal	Salaried
Brand Y	NS 7.0	Modem	Hourly
--	--	--	Part-Time

Table 5: Subset Test Situation With Leftovers Highlighted

	A	B	C	D
1	Array			
2	A	C	D	E
3	0	0	0	0
4	0	1	1	2
5	0	≥ 0	≥ 0	1
6	1	0	1	1
7	1	1	≥ 1	0
8	1	≥ 1	0	2
9	≥ 0	0	≥ 0	2
10	≥ 1	1	0	1
11	≥ 0	≥ 0	1	0

Table 6: Subset Test Situation With Leftovers Assigned

	A	B	C	D
1	Array			
2	A	C	D	E
3	0	0	0	0
4	0	1	1	2
5	0	0	0	1
6	1	0	1	1
7	1	1	1	0
8	1	1	0	2
9	0	0	1	2
10	1	1	0	1
11	0	0	1	0

a minimum of leftovers. In this case, there are no leftover factors (columns) and factors A, C, and D each have one leftover option.

4. Map the factors and options onto the array. Table 5 identifies the leftovers that are the highlighted boxes.
5. Choose values for any leftovers from the valid remaining options and delete any columns that are not needed. Harrell [4] rightly suggests that we assign alternating values for each factor as shown in Table 5. The array with the newly assigned options is shown in Table 6.

5a. (Extra step) Delete any rows that have no unique pairs. Table 7 shows the results of analyzing our subset test situation. Spreadsheet row 11 contains no new pairs in the row (see under RP heading). This row can be deleted. After deleting row 11, reanalyze the array to produce Table 8.

5b. (Extra step) Rearrange cell values to create a row with no unique pairs. If this cannot be done, then quit; otherwise, go back to step 5a. Row 10 column M shows that the B:D column pair is unique (only one occurrence). Move that pair combination to row five by assigning the value one to B5 (note that this is the spreadsheet cell location) as shown in Table 8. The results of reanalysis after reassigning cell B5 are shown in Table 9. Continuing steps 5a and 5b produces Table 10 with six resulting test cases.

6. Transcribe the runs into test cases, adding combinations as needed. Table 11 (see page 30) shows the transcribed values that define the configurations for testing interactions between test factors A, C, D, and E in Table 4.

“Wait a minute!” you say. “Steps 5a and 5b are easier said than done.” You are right, especially if you have more factors! The ReduceArray2 tool displays the number of unique pairs automatically in each row so that you can identify any rows that are not adding new pairs of parameters. They are, in other words, overkill, so you can delete them. After deleting a row, you need to recompute the pair counts so you can identify other rows with no unique pairs that can be deleted. If no rows can be deleted, then you would try to rearrange cell values to create a row with no unique pairs.

Further Automation

Now that I have described the OATS technique and my extensions to further reduce the number of test combinations using the ReduceArray2 tool, I have some more good news. ReduceArray2 also does steps 5a and 5b automatically using its embedded ReduceArray2 macro. In other words, do steps one through five and then use the

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Array					TP	30	Result						
2	A	C	D	E		TC#	RP	UP	A:B	A:C	A:D	B:C	B:D	C:D
3	0	0	0	0		1	6	1	4	3	2	3	2	1
4	0	1	1	2		2	6	2	1	2	2	2	2	1
5	0	0	0	1		3	3	1	4	3	1	3	2	2
6	1	0	1	1		4	5	2	1	2	2	2	2	1
7	1	1	1	0		5	4	2	3	2	1	2	1	2
8	1	1	0	2		6	4	1	3	2	1	2	2	2
9	0	0	0	2		7	1	1	4	3	2	3	1	2
10	1	1	0	1		8	1	1	3	2	2	2	1	2
11	0	0	1	0		9	0	0	4	2	2	2	2	2
12						SP	30							

Note: Cells with X can be any valid value (called "do not cares")

Table 7: Subset Test Situation Results of Analysis

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Array					TP	30	Result						
2	A	C	D	E		TC#	RP	UP	A:B	A:C	A:D	B:C	B:D	C:D
3	0	0	0	0		1	6	3	3	3	1	3	1	1
4	0	1	1	2		2	6	3	1	1	2	2	2	1
5	0	0	0	1		3	3	1	3	3	1	3	2	2
6	1	0	1	1		4	5	3	1	2	2	1	2	1
7	1	1	1	0		5	4	3	3	2	1	2	1	1
8	1	1	0	2		6	4	1	3	2	1	2	2	2
9	0	0	0	2		7	1	1	3	3	2	3	1	2
10	1	1	0	1		8	1	1	3	2	2	2	1	2
11						SP	30							

Table 8: Subset Test Situation Reanalysis After Deleting a Row

ReduceArray2 macro to automatically rearrange parameter values and delete extra rows without losing any pair combinations.

ReduceArray2 provides a near minimal set of tests using simple one-cell-at-a-time rearrangements. If you start with a better initial arrangement, you may be able to reduce it further. ReduceArray2 saves a lot of time and effort and will find a minimum set for some configurations. Furthermore, you can define specific combinations that are required and specific combinations that

are to be excluded. I would be happy to walk you through a demonstration on this. Another macro called Names automatically transcribes the OA values to the names in Table 11, page 30.

Epilogue

Go back to the original test situation. Using ReduceArray2, you can create a set of 20 test configurations from the test factors and options in Table 1. You have time to run 15 test scenarios in each of the 20 test config-

Table 9: Subset Test Situation Reanalysis After Reassigning Cell B5

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Array					TP	30	Result						
2	A	C	D	E		TC#	RP	UP	A:B	A:C	A:D	B:C	B:D	C:D
3	0	0	0	0		1	6	3	2	3	1	2	1	1
4	0	1	1	2		2	6	2	2	1	2	2	2	1
5	0	1	0	1		3	4	1	2	3	1	3	2	2
6	1	0	1	1		4	6	4	1	2	2	1	1	1
7	1	1	1	0		5	4	3	3	2	1	2	1	1
8	1	1	0	2		6	3	1	3	2	1	3	2	2
9	0	0	0	2		7	1	1	2	3	2	2	1	2
10	1	1	0	1		8	0	0	3	2	2	3	2	2
11						SP	30							

Table 10: Final Results Six Test Cases

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Array					TP	30	Result						
2	A	C	D	E		TC#	RP	UP	A:B	A:C	A:D	B:C	B:D	C:D
3	0	0	0	0		1	6	4	2	2	1	1	1	1
4	0	0	1	2		2	5	4	2	1	1	2	1	1
5	1	0	1	1		3	5	4	1	2	1	2	1	1
6	1	1	1	0		4	5	4	2	2	1	1	1	1
7	1	1	0	2		5	5	4	2	1	1	2	1	1
8	0	1	0	1		6	4	4	1	2	1	2	1	1
9						SP	30							

A	C	D	E
CPU	Browser	Network	Type of Employee
Brand X	IE 5.5	Internal	Salaried
Brand X	IE 5.5	Modem	Part-Time
Brand Y	IE 5.5	Modem	Hourly
Brand Y	NS 7.0	Modem	Salaried
Brand Y	NS 7.0	Internal	Part-Time
Brand X	NS 7.0	Internal	Hourly

Table 11: *Transcribed Options*

urations within the timeframe mandated by management. And you even have time to rerun some tests if you encounter some defects and must retest the system.

By the way, I meant no disrespect in this article's preface to the poem written by Emma Lazarus enshrined on a plaque near the Statue of Liberty. But since I have seen literally thousands of tired testers yearning for more effective and more efficient test techniques in my software test-consulting career, I thought I would borrow the theme.

The test situation in this article was contrived. Details about actual feature requirements were ignored to simplify the discussion. But it does make the point that, when you know the features and parameters that could possibly interact incorrectly with each other, then you can follow a simple systematic approach to identify a minimal or near minimal set of tests to test all parameter pairings. That is a powerful capability. You can still pick and choose which test runs to ignore and which additional ones to add. However, identifying the test factors and options in the first place is often very difficult. More research is needed in this area.

It is interesting to note that Phadke's perspective back in 1997 was that,

... the number of tests needed for (the OA testing) method is similar to the number of tests needed for the one-factor-at-a-time method, and with a proper software tool (likely his Robust Testing Method tool), the effort to generate the test plan

can be small. [3]

Harrell's article [4] and this article show you how to create tests without expensive tools; if you have Internet access, download some OAs. Also, I have shown you how to augment the OATS technique with a few additional steps to reduce the number of tests for many types of test situations where you have leftovers. This method also works when the OA has multiple occurrences of pairs that are common when the number of factors is higher. Of course, the Reduce Array2 tool makes it even easier to do this.

Whether you are following formal practices, using defined processes advocated by the Software Engineering Institute's Capability Maturity Model®, or you are applying agile exploratory testing [5] methods, this combinatorial testing technique will certainly help you obtain better integration test coverage. When you are concerned about various features and parameters interacting incorrectly with each other, use this OATS technique augmented with the ReduceArray2 tool or purchase and use a tool such as Automatic Efficient Test Generation. It may not be worth the risk of letting those defects get delivered to your customer. ♦

References

1. Lazarus, Emma. "The New Colossus." Statue of Liberty plaque.
2. Rational Unified Process, 2001, Use Case Template.
3. Phadke, Madhav S. "Planning Efficient

Software Tests." CrossTalk Oct. 1997.

4. Harrell, Jeremy M. "Orthogonal Array Testing Strategy (OATS) Technique." Seilevel, 2001 <www.seilevel.com/OATS.html>.
5. Bach, James. "Exploratory Testing Explained." Ver. 1.1. Satisfice, Inc., 19 Jan. 2003 <www.satisfice.com>.

Notes

1. A use case is a description of a sequence of actions that a system performs that yields an observable result of value to a particular actor (user) [2].
2. See Sloane, N. J. A. "A Library of Orthogonal Arrays" <www.research.att.com/~njas/oaddir>. Also see Sherwood, George. "On the Construction of Orthogonal Arrays and Covering Arrays Using Permutation Groups" <<http://home.att.net/~gsherwood/cover.htm>>.
3. See the Automatic Efficient Test Generation System by Telecordia Technologies at <<http://aetgweb2.arggreenhouse.com>>.
4. Available at no cost at <www.stsc.hill.af.mil>.

About the Author



Gregory T. Daich is a senior software engineer with Science Applications International Corporation currently on contract with the Software Technology Support Center (STSC). He supports STSC's Software Quality and Test Group with more than 26 years of experience in developing and testing software. Daich has taught more than 100 public and on-site seminars involving software testing, document reviews, and process improvement. He consults with government and commercial organizations on improving the effectiveness and efficiency of software quality practices. He has a master's degree in computer science from the University of Utah.

Software Technology
Support Center
OO-ALC/MASEA
6022 Fir Ave.
Bldg. 1238
Hill AFB, UT 84056-5820
Phone: (801) 777-7172
E-mail: greg.daich@hill.af.mil

The Software Technology Support Center's No-Cost Service Offer

If you will identify the factors and options for your particular system, the Software Technology Support Center (STSC) can either help you to identify a minimal or near minimal set of tests to test all pair-wise combinations, or the STSC can identify these for you. Information can be easily received by e-mailing a spreadsheet formatted like Table 1 to the STSC HelpDesk. This is a no-cost service offer to Department of Defense (DoD) organizations. Go to <www.stsc.hill.af.mil> for more information. Of course, non-DoD organizations are welcome to inquire also. We have seen the development costs for many organizations reduced because of the information we have shared with DoD organizations and their contractors.