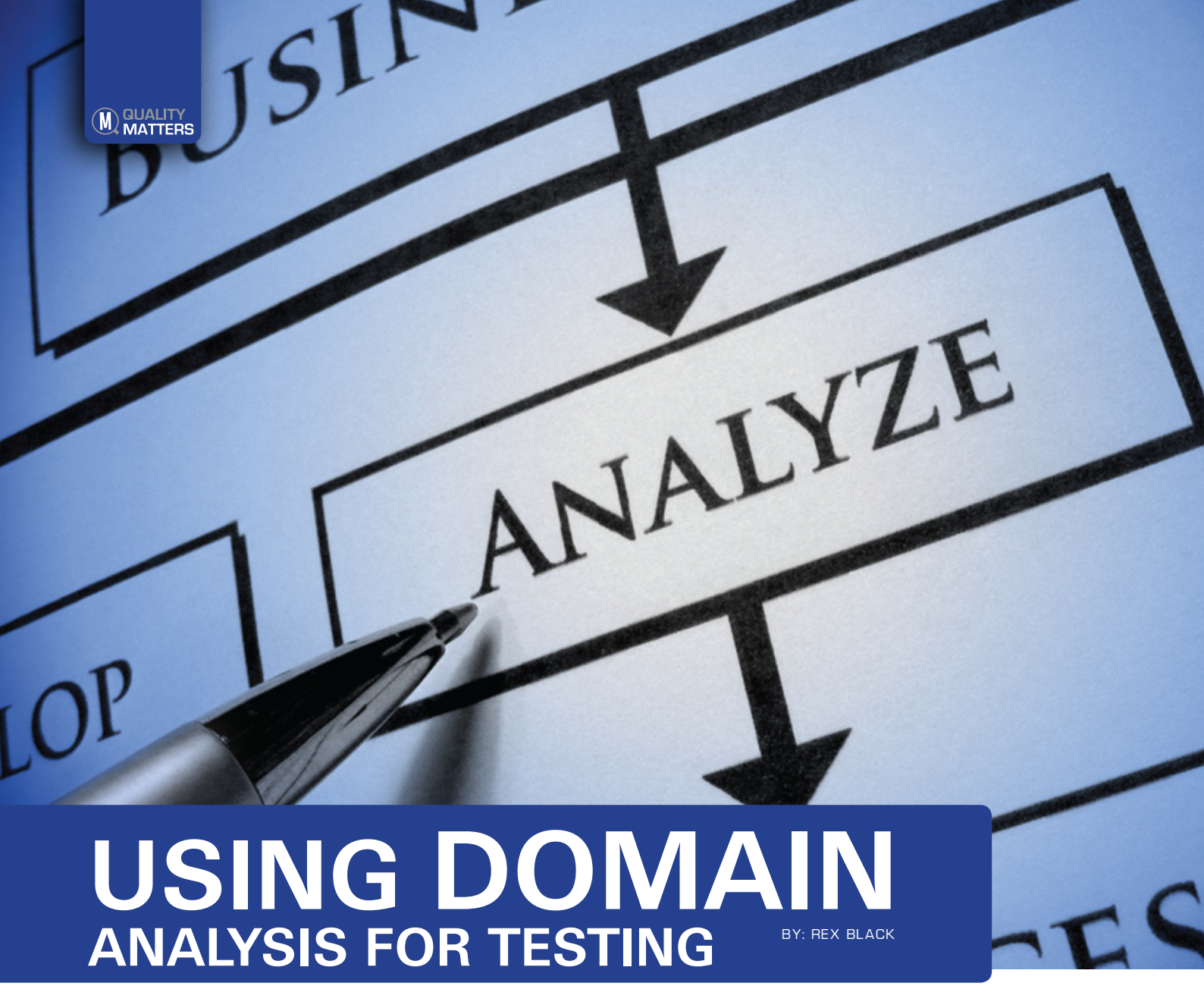


USING DOMAIN ANALYSIS FOR TESTING

BY: REX BLACK

This article is excerpted from Chapter 16 of Rex Black's book Pragmatic Software Testing.

Many of you are probably familiar with basic test techniques like equivalence partitioning and boundary value analysis. In this article, I'll present an advanced technique for black-box testing called domain analysis. Domain analysis is an analytical way to deal with the interaction of factors or variables within the business logic layer of a program. It is appropriate when you have some number of factors to deal with. These factors might be input fields, output fields, database fields, events, or conditions. They should interact to create two or more situations in which the system will process data differently. Those situations are the domains. In each domain, the value of one or more factors influences the values of other factors, the system's outputs, or the processing performed.

COMBINATORIAL EXPLOSIONS

In some cases, the number of possible test cases becomes very large due to the number of variables or factors and the potentially interesting test values or options for each variable or factor. For example, suppose you have 10 integer input fields that accept a number from 0 to 99. There are 10 billion billion valid input combinations.

Equivalence class partitioning and boundary value analysis on each field will reduce but not resolve the problem. You have four boundary values for each field. The illegal values are easy, because you have only 20 tests for those. However, to test each legal combination of fields, you have 1,024 test cases.

But do you need to do so? And would testing combinations of boundary values necessarily make for good tests? Are there smarter options for dealing with such combinatorial explosions?

There are two general cases:

• The factors should interact. One or more factors should partition the system's behavior into two or more domains, situations in which the system processes data differently. In addition, one or more factors interact in the system data processing. The value of one or more factors influences the values of other factors, the system's outputs, or the processing performed. For example, on typical multiscreen forms or business logic, you have different fields and operations, you can fill the fields in various orders, and so forth. The value of one field can determine the way subsequent fields are handled.

• The factors should not interact. The system should process data in substantially the same way no matter what value the factors take on. For example, to test application compatibility, you have to consider platforms, operating systems, printers, cohabitating applications, networks, and modems. The application should work the same in all supported configurations.

If you're looking for a test technique that works when the factors should not interact, then take a look at Chapter 18 of my book Pragmatic Software Testing, which covers orthogonal arrays and all pairs testing. In this article, we'll look at domain analysis, a technique that works when they should interact.

A DOMAIN EXAMPLE USING FREQUENT-FLYER PROGRAMS

One of the occupational hazards of being an international consultant is spending too much time on airplanes. To attempt to gather some benefit from that wasted time and (especially lately) aggravation, I collect frequent-flyer points, usually enough to fly my family around during vacations at the airlines' expense.

Airlines don't give my family free tickets because they like me. Airlines give frequent-flyer points to ensure customer loyalty. So the more loyal I am, the more richly they reward me. Each airline assigns me a status level based on the distance I traveled in their planes in the last year. Table 1 shows the way some airlines calculate bonus points based on the domains formed by the status levels.

STATUS LEVEL	NONE	SILVER	GOLD	PLATINUM
TRIP BONUS	0%	25%	50%	100%
DISTANCE TRAVELED	d	d	d	d
POINTS AWARDED	d	1.25*d	1.5*d	2*d

Table 1: Status levels forming domains of point calculations

The points awarded for any trip is a function of the traveler's status level and the distance traveled.

points = (1+bonus)*distance

The distance traveled in the last year determines the status level, which creates four domains of processing. The distance and trip bonus variables interact, and the result is the points variable.

Certainly, from the point of view of input and output verification, you might want to try maximum and minimum distances and points. Boundary values are useful for that. Systems are often built in layers. At the user interface and the data access layers, checking fields and variables at the boundaries is smart. However, notice that testing a distance of zero with each status level is not only nonsensical, it is also unlikely to tell us whether the formula that calculates the points is incorrect.

So, suppose we are concerned that the system will miscalculate the points awarded? We need to test at least one trip for each traveler status. In Figure 1, you see four points that covers the four domains (i.e., the four calculations). Like the decision table technique, domain analysis focuses on central layers of the system architecture that implement the processing, not the user or data interface layers. Good system design principles says that, as much as possible, accepting valid inputs, displaying valid outputs, rejecting invalid inputs, and marking invalid outputs should reside in the user or data interface(s). You can test those interfaces using boundary values or equivalence partitioning.

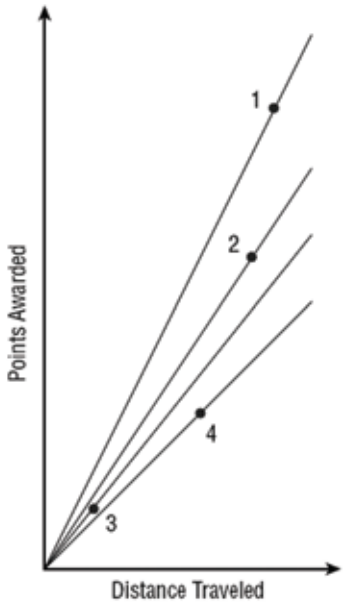


Figure 1: Graphical view of the bonus-points domains

REX BLACK

With a quarter-century of software and systems engineering experience, Rex Black is President of RBCS (www.rbcs-us.com), a leader in software, hardware, and systems testing. For over a dozen years, RBCS has delivered services in consulting, outsourcing and training for software and hardware testing. Employing the industry's most experienced and recognized consultants, RBCS conducts product testing, builds and improves testing groups and hires testing staff for hundreds of clients worldwide. Ranging from Fortune 20 companies to start-ups, RBCS clients save time and money through improved product development, decreased tech support calls, improved corporate reputation and more. As the leader of RBCS, Rex is the most prolific author practicing in the field of software testing today. His popular first book, Managing the Testing Process, has sold over 35,000 copies around the world, including Japanese, Chinese, and Indian releases. His five other books on testing, Advanced Software Testing: Volume I, Advanced Software Testing: Volume II, Critical Testing Processes, Foundations of Software Testing, and Pragmatic Software Testing, have also sold tens of thousands of copies, including Hebrew, Indian, Chinese, Japanese and Russian editions. He has written over twenty-five articles, presented hundreds of papers, workshops, and seminars, and given about thirty keynote speeches at conferences and events around the world. Rex is the past President of the International Software Testing Qualifications Board and the American Software Testing Qualifications Board.

POSSIBLE DOMAIN TEST VALUES

Much of the original work in the area of domain testing was done by Boris Beizer and lately has been carried on by Robert Binder and Lee Copeland[1]. They have written that in order to test a domain, there are four types of interesting test values, as shown in Table 2. This table also includes an analysis of the four points shown in Figure 1.

NAME	DESCRIPTION	FREQUENT-FLYER EXAMPLE
ON	A value on a domain boundary which may be inside or outside of the domain	1 (ON PLATINUM) 2 (ON GOLD) 3 (ON SILVER) 4 (ON NONE)
OFF	A value just off a domain boundary by the smallest recognizable amount and outside of the domain	1 (OFF GOLD) 2 (OFF PLATINUM) 2 (OFF SILVER) 3 (OFF GOLD) 3 (OFF NONE) 4 (OFF SILVER)
IN	A value inside the domain, not on or off	NO SUCH VALUE POSSIBLE
OUT	A value outside the domain, not on or off	NO SUCH VALUE POSSIBLE

Table 2: Domain test values

In case you're wondering about the in and out values for this example, because the domains are actually lines, you can only be on or off. Being off one line means you are on another because those are the only possible values. So, if you look at point 1, you see it is on the platinum line and off the nearest next line, which is gold[2].

Because this is the point where a lot of people get lost when they learn domain analysis, let's back up for a minute to the single-factor situation. For equivalence partitioning and boundary values, imagine a very simple example: a "quantity ordered" field in an e-commerce application. The application accepts quantities that are greater than zero and less than 100. Boundary values and equivalence partitioning say you need to test at least 0, 1, 99, and 100, with the added possibility of an interior value in each class like –7, 15, and 126.

Let's apply domain analysis. First, I identify three domains, which are simply the three equivalence classes.

Invalid too low

Valid

Invalid too high

Now, in Table 3, I identify the four different types of domain test values, reusing them where I can to keep the testing efficient.

DOMAIN	ON	OFF	IN	OUT
INVALID TOO LOW	0	1	-7	15
VALID (LOWER BOUNDARY)	0	-1	15	-7
VALID (UPPER BOUNDARY)	100	101	15	126
INVALID TOO HIGH	100	99	126	15

Table 3: Domain test values for quantity ordered

1 - See, for example, Beizer's book Software Testing Techniques, Binder's book Testing Object-Oriented Systems, and Copeland's book A Practitioner's Guide to Software Test Design.

2 - I thank Judy McKay for this neat way of explaining this fact.

Notice that the domain analysis is applied boundary by boundary so that the valid domain has two analyses: one at the boundary with the invalid too high domain and one at the boundary with the invalid too low domain. Also, notice that you can use the same set of tests you used with boundary values and equivalence partitioning, with the addition of the –1 and 101 values[3]. Finally, notice that, had you defined the quantity ordered field as accepting values that are greater than or equal to 1 and less than or equal to 99, you would still test with the same boundary value and equivalence partitioning data, but in this case you would add 2 and 98 rather than –1 and 101.

This close relationship between the boundary value and equivalence partitioning technique and domain testing leads to domain analysis being referred to as multivariable or multifactor equivalence partitioning by some.

AN AEROSPACE EXAMPLE

With the concept of on, off, in, and out points firmly in hand, let's go back to the multifactor situation. Figure 2 shows an aerospace example. For a missile defense system, one needs to be able to calculate the current—and projected—vertical and horizontal positions of a large falling object, in this case an incoming ballistic missile. A measurement is taken with a laser or radar, but time elapses during the measurement period. When the radar or laser gets back to the measuring station, the object is no longer where it was when it was measured. It has moved.

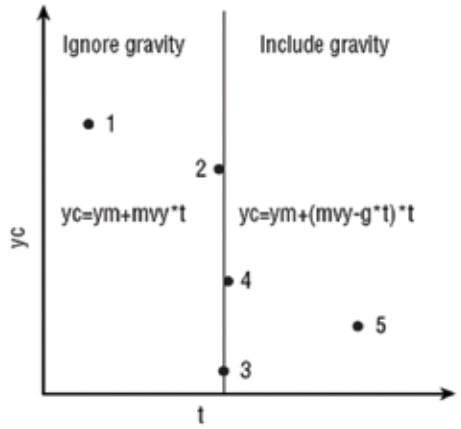


Figure 2: An aerospace domain

3 - In fact, some variants of boundary value and equivalence partitioning test design would include these two values as well.

The situation with horizontal position and velocity is straightforward, as no forces are acting on the missile in this dimension. However, vertically, gravity is accelerating the missile downward. To simplify the calculation, we ignore the effect of gravity if a small enough period of time has elapsed, but we will include the effect of gravity if the time elapsed is 1 millisecond or more. Now we have two domains, as shown in .

To define the five points shown in the figure, I performed the domain analysis shown in Table 4. I assumed that the smallest measurable difference in time is one microsecond. Notice that my choices of the ym and mvy values are arbitrary. What I'm trying to test here is that the boundaries between the two domains are defined properly and that the calculation is done correctly in each domain. Again, notice the similarity between the objectives of domain analysis testing and boundary value and equivalence class testing.

DOMAIN	ON	OFF	IN	OUT
IGNORE GRAVITY	1.000 (POINT 3)	1.001 (POINT 4)	0.372 (POINT 1)	1.752 (POINT 5)
INCLUDE GRAVITY	1.000 (POINT 3)	0.999 (POINT 2)	1.752 (POINT 5)	0.372 (POINT 1)

Table 4: Domain test values for aerospace example

WHEN DOMAIN RULES CHANGE

In the two examples so far, the rules for the domain and the calculations within each domain remain constant. However, it's possible for the rules to change within a domain, creating the need for additional tests.

Suppose we look at the total number of points awarded to a frequent flyer for a whole year. A traveler's status is based on the total distance traveled in the current year or the total distance traveled in the past year. At 25,000, the status increases to silver. At 50,000, the status increases

to gold. At 100,000, the status increases to platinum. Any distance traveled after the status changes accrues bonus points based on the new status, both in the current year and the next year. Basically, these rules split three of the four domains into two or more subdomains.

Domain testing theory says that we should add some additional domain tests in this situation, specifically to cover values in each subdomain or rule segment. This adds one new test for travelers starting with gold status (if they move up to platinum), two new tests for travelers starting with silver status (if they move up to gold or platinum), and three for travelers starting with no status (if they move up to silver, gold or platinum). These values are shown in Figure 3. Notice that, if you test this by adding points for completed flights one flight at a time, a traveler who, for example, started the year with silver status and ended with platinum would, at some point in the year, have silver status, gold status, and platinum status.

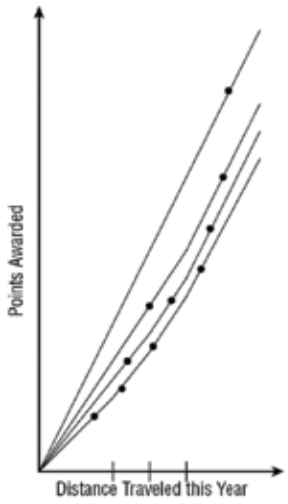


Figure 3: Yearly frequent-flyer point calculations

Notice that what I am testing with these values is only the calculation itself. If I wanted to add a test that checked whether the boundaries where a traveler's status changes were defined correctly I'd have to add tests at these boundaries. To do so, I could apply the concept of domain analysis to each of the four lines. Notice that if I did so, domain analysis for each line would be the same as boundary value and equivalence partitioning[4].

DOMAIN ANALYSIS SUMMARY

Let's summarize what we've covered. In some cases, factors or variables should interact. More precisely, the system's behavior is only correct when the factors or variables interact properly. In the first frequent-flyer example, the status level determined the formula that calculated the number of points awarded from the distance traveled. Each status level formed a domain of processing.

In the aerospace example, the system should use one of two formulas to calculate the current vertical position, depending on the measured position, the measured vertical velocity, time, and in one domain, the acceleration of gravity. The time elapsed since the measurement formed the two domains of processing.

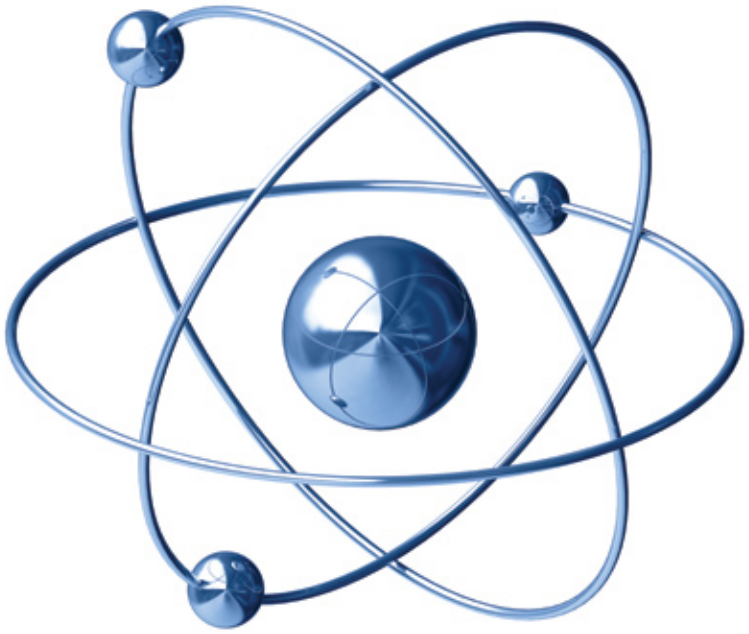
In the second frequent-flyer example, the initial status level (based on the last year) and the total distance traveled for the current year determined both the points awarded and the transition into the new status level. In that example, I analyzed the domains to test the correct points awarded. However, note that by adding more test points, I could have tested whether the promotion to the correct new status level was occurring at the correct points. Specifically, I would have needed on and off points for the boundaries at 25,000, 50,000, and 100,000, on each of the four lines corresponding to the initial status levels. We'll take a look at ways to cover multiple domains in a single test shortly.

In the aerospace example, the interaction took the form of formulas that influence how inputs interacted to create an output. In the frequent-flyer example, the inputs interacted with data about the traveler, likely from a database. One or more of the factors or variables combined to create domains of operation or processing that should be the same. Domains can be based on inputs, outputs, or internal data sets. Domains can involve real numbers, integers, dates, sets, or other intrinsic or programmer-defined data types.

Here are some rules of thumb for constructing domain tests:

- You should cover at least the on and off values for each domain.
- If an on value for one domain is an off value for another, you needn't create additional values.
- In the case where the variable that determines the domain is an integer or real variable, as in the aerospace example, you should also cover in and out values.
- If an in value for one domain is an out value for another, you needn't create additional values.
- If the rules for a domain's boundary change or the processing changes in the domain, consider additional on, off, in, and out values to achieve strong domain coverage.

Finally, remember that typically you are not verifying basic data sanity checking, but rather that the internal core logic of the system is correct.



4 - In Software Testing Techniques, Boris Beizer introduces some new terms for this situation. He calls domain tests that cover each rule segment strong. He refers to domain tests that do not cover each rule segment as weak. He then uses the phrase "strong 1x1 testing" to refer to domain tests that include at least one on and on off point for each subdomain. He calls domain tests that test each domain but not each subdomain "weak 1x1 testing."