Software Test Design Technique - Decision Tables

Conceptually, decision tables express the rules that govern handling of transactional situations. By their simple, concise structure, decision tables make it easy for us to design tests for those rules, usually at least one test per rule.

When I said "transactional situations," what I meant was those situations where the conditions—inputs, preconditions, etc.—that exist at a given moment in time for a single transaction are sufficient by themselves to determine the actions the system should take. If the conditions are not sufficient, but we must also refer to what conditions have existed in the past, then we'll want to use state-based testing, which we'll cover in a moment.

The underlying model is either a table (most typically) or a Boolean graph (less typically). Either way, the model connects combinations of conditions with the action or actions that should occur when each particular combination of conditions arises.

If the graph is used, this technique is also referred to as a cause-effect graph, because that is the formal name of the graph. However, it's important to keep in mind that any given decision table can be converted into a cause-effect graph, and any given cause-effect graph can be converted into a decision table. I prefer decision tables, and they are more commonly used, so I'll focus on that here.

To create test cases from a decision table or a cause-effect graph, we design test inputs that fulfill the conditions given. The test outputs correspond to the action or actions given for that combination of conditions. During test execution, we check that the actual actions taken correspond to the expected actions.

We create enough test cases that every combination of conditions is covered by at least one test case. Frequently, that coverage criterion is relaxed to say, we cover those combinations of conditions that can determine the action or actions. If that's a little confusing, the distinction I'm drawing will become clear to you when we talk about collapsed decision tables.

With a decision table, the coverage criterion boils down to an easy-to-remember rule of at least one test per column in the table. For cause-effect graphs, you have to generate a so-called "truth table" that contains all possible combinations of conditions and ensure you have one test per row in the truth table.

So, what kind of bugs are we looking for with decision tables? There are two. First, under some combination of conditions, the wrong action might occur. In other words, there is some action that the system is not to take under this combination of conditions, yet it does. Second, under some combination of conditions, the system might not take the right action. In other words, there is some action that the system is to take under this combination of conditions, yet it does not.

Consider an e-commerce application like the one found on our RBCS Web site, www.rbcs-us.com. At the user interface layer, we need to validate payment information, specifically credit card type, card number, card security code, expiration month, expiration year, and cardholder name. You can use

boundary value analysis and equivalence partitioning to test the ability of the application to verify the payment information, as much as possible, before sending it to the server.

So, once that information goes to the credit card processing company for validation, how can we test that? Again, we could handle that with equivalence partitioning, but there are actually a whole set of conditions that determine this processing:

- Does the named person hold the credit card entered, and is the other information correct?
- Is it still active or has it been cancelled?
- Is the person within or over their limit?
- Is the transaction coming from a normal or a suspicious location?

The decision table in Table 1 shows how these four conditions interact to determine which of the following three actions will occur:

- Should we approve the transaction?
- Should we call the cardholder (e.g., to warn them about a purchase from a strange place)?
- Should we call the vendor (e.g., to ask them to seize the cancelled card)?

Take a minute to study the table to see how this works. The conditions are listed at the top left of the table, and the actions at the bottom left. Each column to the right of this left-most column contains a business rule. Each rule says, in essence, "Under this particular combination of conditions (shown at the top of the rule), carry out this particular combination of actions (shown at the bottom of the rule)."

Conditions	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Real account?	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N	N	N
Active account?	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	N	N	N	N
Within limit?	Y	Y	N	N	Y	Y	N	N	Y	Y	N	N	Y	Y	N	N
Location okay?	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N
Actions																
Approve?	Y	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
Call cardholder?	N	Y	Y	Y	N	Y	Y	Y	N	N	N	N	N	N	N	N
Call vendor?	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Table 1: Decision Table Example (Full)

Notice that the number of columns—i.e., the number of business rules—is equal to 2 (two) raised to the power of the number of conditions. In other words, 2 times 2 times 2 times 2, which is 16. When the conditions are strictly Boolean—true or false—and we're dealing with a full decision table (not a collapsed one), that will always be the case.

Did you notice how I populated the conditions? The topmost condition changes most slowly. Half of the columns are Yes, then half No. The condition under the topmost changes more quickly but more slowly than all the others. The pattern is quarter Yes, then quarter No, then quarter Yes, then quarter No. Finally, for the bottommost condition, the alternation is Yes, No, Yes, No, Yes, etc. This pattern makes it easy to ensure you don't miss anything. If you start with the topmost condition, set the left half of the rule columns to Yes and the right half of the rule columns to No, then following the pattern I showed, if you get to the bottom and the Yes, No, Yes, No, Yes, etc., pattern doesn't hold, you did something wrong.

Deriving test cases from this example is easy: Each column of the table produces a test case. When the time comes to run the tests, we'll create the conditions which are each test's inputs. We'll replace the "yes/no" conditions with actual input values for credit card number, security code, expiration date, and cardholder name, either during test design or perhaps even at test execution time. We'll verify the actions which are the test's expected results.

In some cases, we might generate more than one test case per column. I'll cover this possibility in more detail later, as we enlist our previous test techniques, equivalence partitioning and boundary value analysis, to extend decision table testing.

Collapsing Columns in the Table

Notice that, in this case, some of the test cases don't make much sense. For example, how can the account not be real but yet active? How can the account not be real but within limit? This kind of situation is a hint that maybe we don't need all the columns in our decision table.

We can sometimes collapse the decision table, combining columns, to achieve a more concise—and in some cases sensible—decision table. In any situation where the value of one or more particular conditions can't affect the actions for two or more combinations of conditions, we can collapse the decision table.

This involves combining two or more columns where, as I said, one or more of the conditions don't affect the actions. As a hint, combinable columns are often but not always next to each other. You can at least start by looking at columns next to each other.

To combine two or more columns, look for two or more columns that result in the same combination of actions. Note that the actions must be the same for all of the actions in the table, not just some of them. In these columns, some of the conditions will be the same, and some will be different. The ones that are different obviously don't affect the outcome. So, we can replace the conditions that are different in those columns with the dash character ("-"). The dash usually means either I don't care, it doesn't matter, or it can't happen, given the other conditions.

Now, repeat this process until the only further columns that share the same combination of actions for all the actions in the table are ones where you'd be combining a dash with Yes or No value and thus wiping out an important distinction for cause of action. What I mean by this will be clear in the example I present in a moment, if it's not clear already.

Another word of caution at this point: Be careful when dealing with a table where more than one rule can apply at one single point in time. These tables have non-exclusive rules. We'll discuss that further later in this section.

Table 2 shows the same decision table as before, but collapsed to eliminate extraneous columns. Most notably, you can see that columns 9 through 16 in the original decision table have been collapsed into a single column.

Conditions	1	2	3	5	6	7	9
Real account?	Y	Y	Y	Y	Y	Y	N
Active account?	Y	Y	Y	N	N	N	-
Within limit?	Y	Y	N	Y	Y	N	-
Location okay?	Y	N	-	Y	N	-	-
Actions							
Approve?	Y	N	N	N	N	N	N
Call cardholder?	N	Y	Y	N	Y	Y	N
Call vendor?	N	N	N	Y	Y	Y	Y

Table 2: Decision Table Example (Collapsed)

I've kept the original column numbers for ease of comparison. Again, take a minute to study the table to see how I did this. Look carefully at columns 1, 2, and 3. Notice that we can't collapse 2 and 3 because that would result in "dash" for both "within limit" and "location okay." If you study this table or the full one, you can see that one of these conditions must not be true for the cardholder to receive a call. The collapse of rule 4 into rule 3 says that, if the card is over limit, the cardholder will be called, regardless of location. The same logic applies to the collapse of rule 8 into rule 7.

Notice that the format is unchanged. The conditions are listed at the top left of the table, and the actions at the bottom left. Each column to the right of this left-most column contains a business rule. Each rules says, "Under this particular combination of conditions (shown at the top of the rule, some

of which might not be applicable), carry out this particular combination of actions (shown at the bottom of the rule, all of which are fully specified)."

Notice that the number of columns is no longer equal to 2 raised to the power of the number of conditions. This makes sense, since otherwise no collapsing would have occurred. If you are concerned that you might miss something important, you can always start with the full decision table. In a full table, because of the way you generate it, it is guaranteed to have all the combinations of conditions. You can mathematically check if it does. Then, carefully collapse the table to reduce the number of test cases you create.

Also, notice that, when you collapse the table, that pleasant pattern of Yes and No columns present in the full table goes away. This is yet another reason to be very careful when collapsing the columns, because you can't count on the pattern or the mathematical formula to check your work.