The Decision Model #3: Revolutionizing the Testing Process for Business Logic

Barbara von Halle and Larry Goldberg

Abstract

This is the third in a three-part series introducing the Decision Model. The first of these three articles reveals how the Decision Model, as a model for business logic, becomes a strategic instrument of business agility for business leaders. The second article explains how the Decision Model revolutionizes today's business processes and requirements management. The target audience for this article is business people, developers, and testers. "The Decision Model gives a specific form, function, and tangible visual representation to business logic as expressed by business people."(Adapted from (von Halle and Goldberg 2009)*) As such, the Decision Model can shorten the testing and development cycle drastically. That's because the Decision Model separates the business logic from other concerns and therefore test cases can be driven directly from the content of a Decision Model. Test cases can be developed for individual Rule Families within a Decision Model or for part or all of a Decision Model. The Decision Model also assists in scenario testing for a business process.*

Background

The Relational Model changed the way we manage, leverage, and store data. The discovery of this technology independent-model of data brought about the information revolution by enabling us to separate data and realize and leverage its considerable value as an organizational asset. Today, the Decision Model, as a technology-independent model of business logic, promises to release perhaps even greater value from our technology assets and business processes.

The Decision Model organizes business rules and business logic in a similarly rigorous manner to the way the Relational Model organizes data. It enables innovation in a wide range of business endeavors, including business planning and management, business architecture, business transformation and re-engineering, and business process management. The Decision Model also brings significant opportunity and agility in information technology practices, including requirements capturing, application development and testing, and application maintenance.

The Decision Model recognizes that business logic has its own existence, independent of how executed, where executed, and whether or not it's implemented in automated systems. It has a recognizable structure different from other model structures. Decision Model normalization principles minimize redundancies and anomalies in the business logic. Details of the Decision Model and commentaries on its place in the world of business and IT are provided in the book The Decision Model: A Business Logic Framework Linking Business and Technology (von Halle and Goldberg 2009). This article consists, in part, of abstracts from the book; directly quoted passages, diagrams and tables are cited, and are copyright © 2009 Auerbach Publications/Taylor & Francis LLC, and Reprinted with the permission of the Publisher.

To gain a rapid understanding of the Decision Model as insight into the ideas in this article, please read or refer to the Decision Model Primer.

The Decision Model and Testing against the Functional Requirements

"Program testing in development projects traditionally takes place after a module has been programmed. The writing of test cases is vital and is considered to be an integral part of developing functional requirements. Test cases are designed to test the inputs and outputs of the module as a whole. Their purpose is to exercise the business logic within the program for all the circumstances likely to occur.

However, there are two common problems in this conventional method of testing:

- Because the tests are conducted after the module is completely programmed, test failures often result in significant rework.
- Because the business logic is not separated from the rest of the code, test results don't necessarily measure the accuracy of the original business logic, because errors in it may be shielded by program manipulations. This can lead to failure of the code in unanticipated circumstances, or as a result of future maintenance." (von Halle and Goldberg 2009)

Directly Testing the Decision Model

"The Decision Model can shorten the testing and development cycle drastically. The rigor of the Decision Model offers opportunities for testing strategies that can be used to exhaustively test the business logic in the Decision Model. Because Decision Models are designed for reuse, their test cases can be maintained as a part of the Decision Model. In this way, the test cases are available for reuse when the corresponding Decision Model is reused in other systems or updated. Also, test cases are driven directly by the logic in the Decision Model: consequently, when the logic changes, there is a direct traceability from those changes to the test cases that can then be changed accordingly." (The Decision Model, von Halle & Goldberg, © 2009 Auerbach)

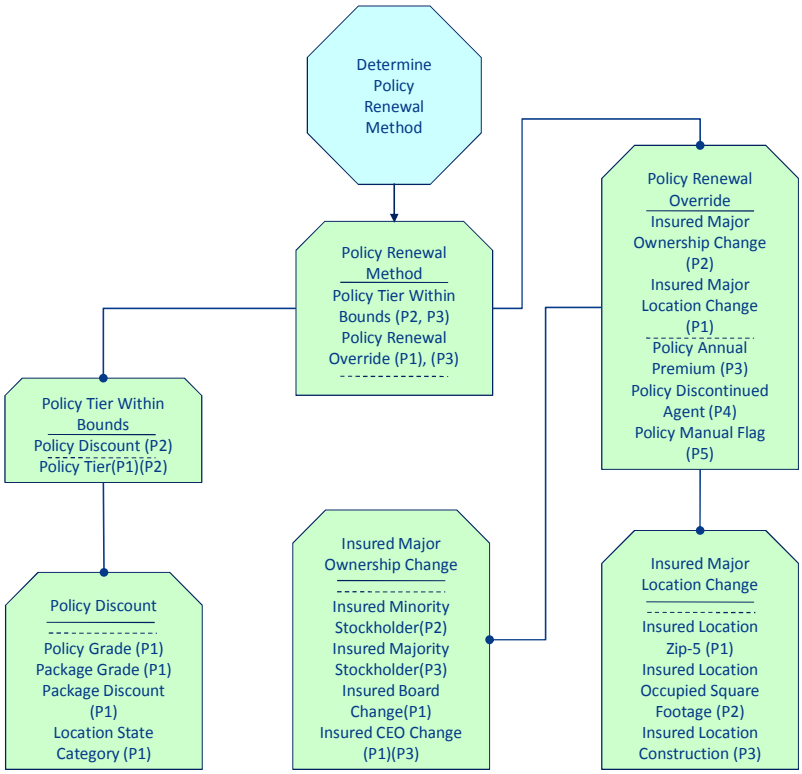

Figure 1 Decision Model Diagram

Source: The Decision Model (von Halle & Goldberg) © 2009 Auerbach Publications/Taylor & Francis LLC. Reprinted with the permission of the Publisher.)

Building Test Cases for the Decision Model

Consider the Decision Model example found in the Decision Model Primer, and contained in

Figure 1. The Decision Rule Family for this Decision Model is shown in Table 1.

	Conditions				Conclusion	
Pattern	Policy Manual Override		Policy Tier Within Bounds		Policy Renewal Method	
1	Is	Yes			is	Manual Renewal Process
2			is	No	is	Manual Renewal Process
3	Is	No	is	Yes	is	Automatic Renewal Process

Table 1 Decision Rule Family

Source: The Decision Model (von Halle & Goldberg) © 2009 Auerbach Publications/Taylor & Francis LLC. Reprinted with the permission of the Publisher.)

A brief inspection of this Rule Family is necessary to develop a test case that has all the possible logical inputs of this Rule Family, ensuring that the test case has all possible logical outputs of the Rule Family.

This test case is shown in Table 2, illustrating the simple technique of providing every single possible combination of values that are within the domain of the two condition fact types, Policy Manual Override and Policy Tier Within Bounds.

Inputs		Output
Policy Manual Override	Policy Tier Within Bounds	Policy Renewal Method
Yes	Yes	Manual Renewal Process
Yes	No	Manual Renewal Process
No	Yes	Automatic Renewal Process
No	No	Manual Renewal Process

Table 2 Test Case for Decision Rule Family

Source: The Decision Model (von Halle & Goldberg) © 2009 Auerbach Publications/Taylor & Francis LLC. Reprinted with the permission of the Publisher.)

The general approach for developing test cases that directly test business logic may be seen from these two tables. However, if we review the test case in Table 2 we soon spot a problem: the inputs, (Policy Manual Override and Policy Tier Within Bounds) are *interim knowledge,* that is to say that this information doesn't exist except at the time the Decision Model for the business decision is actually executed. In other words, the information is not available from persistent data, such as in a data base or another source of stored information. We could have seen this by examining the Decision Model in

Figure 1 because both condition fact types in the Decision Rule Family are below the dotted line, that is to say they are dependent on other Rule Families for their operand values.

Examining the Decision Model even further, we determine all fact types that are derived from persistent data or from user input, and, therefore, they are not *interim knowledge*. These are the fact types that are below the dotted lines in each of the Rule Families. Taken together, they represent the total input for our Decision Model test case.

So we may assume that a complete test case for the Decision Model in

Figure 1 would consist of a table of input values for all fact types below the dotted lines, and would be based on values representing the full domain of values for those fact types. We soon realize that the test case in Table 2 is very simple because the domain (i.e. the valid values) for each of the inputs is only the binary "Yes" or "No", ensuring only a few rows to cover all the combinations of possible values. Because there are 16 fact type input values in the Decision Model in

Figure 1, each with a large range of domain values, the test case becomes very large if we cover every possible combination of every value of every input fact type. A better approach is to use selected boundary values for those input fact types where the boundary values are values close to the point that the input has an impact on the conclusion fact type value. Table 3 is only a segment of the test case necessary to test the Decision Model in

Figure 1, showing just sixteen of potentially tens of thousands of rows. If you look at columns 1,2, 3 and 4 in Table 3 you will see how we use the boundary value to reduce the total size of the test case.

In column 1, Policy Tier is a fact type whose domain is the range of 1-4, in integers. So we expect there to be a value of 1, 2, 3, and 4 for each of the valid test values of each of the other columns. In column 2, Policy Grade is a fact type whose domain spans 75%-100% in integers, so we consult the business and choose to test a range of values that are statistically significant, 78%, 82%, 90% and 95%. In column 5, Package Discount is a fact type whose domain range is 0% to 25%, with important breaks at 10%, 15% and 25%, so we choose values that are close to the boundaries of these breaks and test 0%, 10%, 14% and 22%. In our segment of the test case we have only demonstrated variations for the first 4 columns, but each of these variants will be tested with every variation we choose within the domains for each of the other columns. Some of these columns have only a binary domain of "Yes" or "No", but column 11, Policy Discount is a fact type having a domain range of $2,000 to $150,000 so we can expect a wide range of values. Clearly, testing the entire Decision Model requires extensive test cases when there are a significant number of input values.

A point may well be reached where there are too many input values for practical testing purposes, and the number of variations have to be reduced by statistical methods. In Table 3, allowing for only 4 different values for column 12, there would be over two million rows to exhaustively test every possible combination. So in practice we would reduce the variations to focus on the statistically meaning variations; this may mean more samples in column 11, but fewer in columns 1-5.

	Inputs															Output
	1	2	3	5	6	7	8	9	10	11	12	13	14	15	16	
Test Seq. #	Policy Tier	Policy Grade	Package Grade	Package Discount	Location State Category	Insured Minority Stockholder	Insured Majority Stockholder	Insured Board Change	Insured CEO Change	Policy Annual Premium	Policy Discontinued Agent	Policy Manual Flag	Insured Location Zip-5	Insured Location Occupied Square Footage	Insured Location Occupied Construction	Decision
1	1	82%	78%	0%	2	No	No	No	No	11,000	No	No	Unchanged	Unchanged	Unchanged	Automatic
2	1	90%	82%	10%	2	No	No	No	No	11,000	No	No	Unchanged	Unchanged	Unchanged	Automatic
3	1	95%	90%	14%	2	No	No	No	No	11,000	No	No	Unchanged	Unchanged	Unchanged	Automatic
4	1	99%	95%	22%	2	No	No	No	No	11,000	No	No	Unchanged	Unchanged	Unchanged	Automatic
5	2	82%	78%	0%	2	No	No	No	No	11,000	No	No	Unchanged	Unchanged	Unchanged	Automatic
6	2	90%	82%	10%	2	No	No	No	No	11,000	No	No	Unchanged	Unchanged	Unchanged	Automatic
7	2	95%	90%	14%	2	No	No	No	No	11,000	No	No	Unchanged	Unchanged	Unchanged	Automatic
8	2	99%	95%	22%	2	No	No	No	No	11,000	No	No	Unchanged	Unchanged	Unchanged	Automatic
9	3	82%	78%	0%	2	No	No	No	No	11,000	No	No	Unchanged	Unchanged	Unchanged	Automatic
10	3	90%	82%	10%	2	No	No	No	No	11,000	No	No	Unchanged	Unchanged	Unchanged	Automatic
11	3	95%	90%	14%	2	No	No	No	No	11,000	No	No	Unchanged	Unchanged	Unchanged	Automatic
12	3	99%	95%	22%	2	No	No	No	No	11,000	No	No	Unchanged	Unchanged	Unchanged	Automatic
13	4	82%	78%	0%	2	No	No	No	No	11,000	No	No	Unchanged	Unchanged	Unchanged	Automatic
14	4	90%	82%	10%	2	No	No	No	No	11,000	No	No	Unchanged	Unchanged	Unchanged	Automatic
15	4	95%	90%	14%	2	No	No	No	No	11,000	No	No	Unchanged	Unchanged	Unchanged	Automatic

16	4	99 %	95%	22%	2	No	No	No	No	11,000	No	No	Unchanged	Unchanged	Unchanged	Automatic

Table 3 Segment of a Sample Test Case for the Decision Model in Figure 1

SOA proponents may notice with satisfaction that the input fact types for the test case are exactly those that we would use as the input variables for a service interface to implement this Decision Model as a decision service. Of course the output values we expect in our test case are exactly those we would use as outputs for that same decision service. In other words, our Decision Model test case treats the Decision Model as a service, and the test case exercises the code as if it were a decision service. It would be most convenient then if the whole Decision Model were implemented as a decision service; not only would this greatly aid in testing but – and more importantly – would contribute to the agility of the SOA environment.

Building Test Cases for the Rule Families

If the business people define and maintain the business logic in their Decision Models, but IT professionals do not implement the business logic in a way that closely follows the Decision Model, then testing the business logic in implemented code would need to occur as a single unit (in other words treating it as a "black box"). Each test case must exercise the entire Decision Model, as in Table 3.

However, if the business logic were to be programmed using a pattern of development that coded each Rule Family as a separate module or segment, then it would be conceptually possible to build a set of test cases, one for each Rule Family. Of course such a pattern of development would best be served by BRMS (Business Rule Management Systems) technology, where implementing a set of logic as a single group of business rules is a well established pattern of development. In this approach, each Rule Family becomes a module, or service, extending even further the agility suggested by the creation of decision services. All implemented Rule Families would have their own defined interface, and dependent Rule Families would have some or all of the inputs provided by supporting Rule Families. This approach may be enhanced by the use of what is sometimes called rule flow in some BRMS technology, where Rule Families are implemented as points in the rule flow.

Implementing each Rule Family as a reusable object is the most flexible approach to automating the business logic in a Decision Model, as it permits the orchestration of Rule Families into a single decision service. Properly implemented, the developer allows each row in the Rule Family to be traceable to specific lines of code (contiguous, ideally.) This would perfect the traceability between the rows of business logic in the Decision Model by the business people, and the code being maintained by IT professionals.

Such a pattern of development allows the testing of interim knowledge generated by each Rule Family, and then of cascading Rule Families until the complete Decision Model is tested. The developer provides an interface so that interim knowledge values may be entered into the Rule Family by human input or by some other means (for example, a test file containing pre-determined values), providing ad hoc testing of Rule Families, which would be particularly helpful during development.

By aligning the code structure with that of the Decision Model, the developer permits grey[1] or white[2] box testing of the business logic: the rewards over time would be threefold; significantly more accurate, maintainable code, business people in control of business logic and its test cases, and applications delivering higher quality at lower risk.

Also, by testing at the Rule Family level, the number of variables in each test will be significantly lower than at the Decision Model level, leading to simpler test cases that are easier to maintain.

Scenario Testing Processes

"One effective way to test the business functionality of a business process is scenario testing. Scenario testing does not test the business logic directly; it tests the whole implemented business process model for a given set of inputs. This approach to testing business logic was originally suggested by Michael Beck and Art Moore, and the examples below are based on their work (Beck and Moore 2006). Figure 2 contains a simplified business process model. A business scenario is described in Table 4.

The table is a simplified view of the scenario and associated fact type logical expressions. In practice, the scenario would include a selection of inputs, which in this case means policies that exercise the range of relevant values for all fact types across the business process. These scenarios exercise the system as a complete business process model. In the words of Beck and Moore (their references to "business model" mean the business process model together with the relevant business rules): Our project experience has shown that the advantage of this approach, which is based on the information

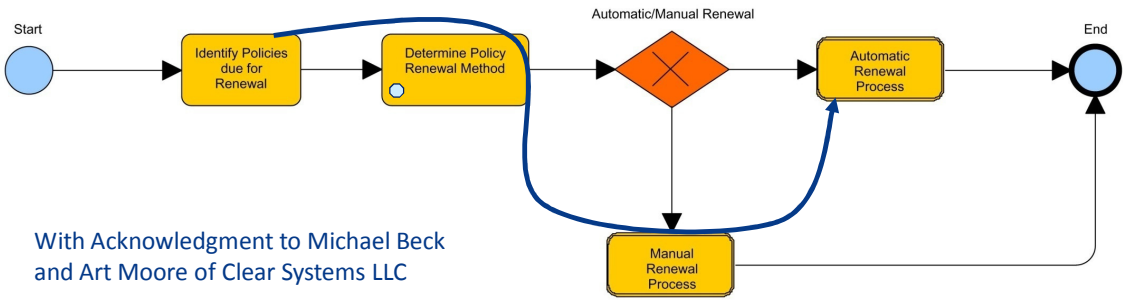

Figure 2 Path through the Process Model of a Single Business Scenario

Source: The Decision Model (von Halle & Goldberg) © 2009 Auerbach Publications/Taylor & Francis LLC. Reprinted with the permission of the Publisher.)

[1] In grey box testing, the tester has access to internal algorithm and variables in the code, but performs tests at the black box, or user level

[2] In white box testing, the tester has access to internal algorithm and variables in the code, and can test these specifically

contained in the business model, is that we can develop the scenarios early in the development process (no later than logical design); giving the test team valuable input on the number and complexity of tests they will be required to perform. The second advantage is that the scenarios and subsequent test scripts based on the content of the business model are readily understood by the business users, which has the benefit of removing ambiguity from the testing and acceptance process. (von Halle and Goldberg 2009)

Scenario	Fact Types Evaluated
A Policy is submitted for renewal and is evaluated to determined whether the policy will be routed to the manual or automotive renewal method The Policy Tier is Within Bounds The conditions meet the requirements for automatic renewal, consequently the policy should be submitted for automatic renewal	Fact types evaluated are: Policy Tier > 2.6 Policy Discount = 0% Policy Manual Override=No

Table 4 Testing Business Scenarios

Summary

The Decision Model offers a range of opportunities to test directly the business logic in systems precisely because the Decision Model separates the business logic from all other aspects of systems. These test cases can be written by business people who define and manage the business logic in the Decision Models and who are familiar with the business intent of that business logic. Test cases can be stored and changed over time with the Decision Models and the code, maintaining a strong traceability between them. Because the Decision Model promotes a shared understanding of the business logic between business people and IT professionals, the value is that of business people generating accurate test cases appropriate for the business logic they govern.

But this has been only the part of the story. The Decision Model plays an important role in linking business and technology, the subject of the first article of this series. The Decision Model also plays a significant role in simplifying and improving business process and the development of functional requirements. These are the subjects of the second article of this series.

To learn more about the Decision Model, readers can order the book at www.TheDecisionModel.com.

Works Cited

Beck, Micheal, and Art Moore. "The Business Rule Revolution." In *The Business Rule Revolution*, by Barbara von Halle and Larry Goldberg, 119-146. Cupertino, CA: Happy About, 2006.

von Halle, Barbara, and Larry Goldberg. *The Decision Model; A Business Logic Framework Linking Business and Technology.* New York: Auerbach, 2009.