A Review Paper on
Decision Table-Based Testing

Cai Ferriday
345399

Tutors:
Dr. Roggenbach
Prof. Schlingloff

January 7th, 2007

Abstract

An investigation, and review on Decision Table-Based Testing. Discussing the way in which it operates and generates test cases. An insight into the Functional Testing Strategies that surround it, and how they relate to each other.

Table of Contents

1 Introduction

From the beginning of software development testing has always been incorporated into the final stages. Over the years the technicality of software has increased dramatically. As this complexity increases, programmers realise that testing is just as important as the development stages.

Nowadays there are two main types of stages, White box testing and Black box testing. Grey box testing is another type, but it's not so well known and is sometimes used with Decision Table-Based Testing:

- **White box** – testing concerned with the internal structure of the program.
- **Black box** – testing concerned with input/output of the program.
- **Grey box** – using the logical relationships to analyse the input/output of the program.

Testing has been modularised into different categories as it's been practised and researched since the 1970's. This paper is going to discuss and analyse Decision Table-Based Testing which is a Functional Testing method, also known as Black box testing.

In this paper I aim to explore the fundamental concepts of Decision Table-Based Testing and how it differs from other functional testing methods. Using examples I will also explain how Decision Table-Based Testing operates and how to use it.

The remainder of this document is split up into 3 areas:-

- **Background** – An overview of DT-BT's origin and its relationship with other Functional Testing methods.

- **Applications** – A discussion on the ways DT-BT can be used, along with examples and how it compares with different Functional Testing methods.

- **Summary & Further Work** – A brief outline of the material covered and further work that will complement it.

2 Background

2.1 Origin

Decision Table-Based Testing has been around since the early 1960's; it is used to depict complex logical relationships between input data. There are two closely related methods of Functional Testing:

- The Cause-Effect Graphing (Elmendorf, 1973; Myers, 1979), and
- The Decision Tableau Method (Mosley, 1993).

These methods are a little different to Decision Table-Based Testing, but use similar concepts of which I will explain later on. I won't go into great detail as these methods are awkward and unnecessary with the use of Decision Tables.

2.2 Definitions

2.2.1 Decision Table-Based Testing?

A Decision Table is the method used to build a complete set of test cases without using the internal structure of the program in question. In order to create test cases we use a table to contain the input and output values of a program. Such a table is split up into four sections as shown below in fig 2.1.

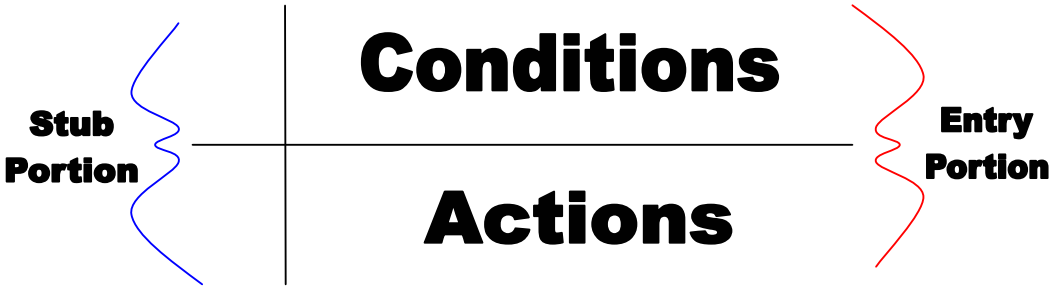

Figure 2.1 The Basic Structure of a Decision Table.

In fig 2.1 there are two lines which divide the table into its main structure. The solid vertical line separates the Stub and Entry portions of the table, and the solid horizontal line is the boundary between the Conditions and Actions. So these lines separate the table into four portions, Condition Stub, Action Stub, Condition Entries and Action Entries.

A column in the entry portion of the table is known as a *rule*. Values which are in the condition entry columns are known as inputs and values inside the action entry portions are known as outputs. Outputs are calculated depending on the inputs and specification of the program.

In fig 2.2 there is an example of a typical Decision Table. The inputs in this given table derive the outputs depending on what conditions these inputs meet. Notice the use of "-"in the table below, these are known as *don't care entries*. Don't care entries are normally viewed as being false values which don't require the value to define the output.

Stub	*Rule 1*	*Rule 2*	*Rule 3*	*Rule 4*	*Rule 5*	*Rule 6*
c1	T	T	T	F	-	T
c2	F	T	T	T	-	T
c3	T	T	-	T	T	T
c4	T	F	F	T	T	T
a1	X	X		X	X	X
a2		X				
a3	X			X		
a4			X			X

Figure 2.2 a Typical Structure of a Decision Table

Figure 2.2 shows its values from the inputs as true(T) or false(F) values which are binary conditions, tables which use binary conditions are known as *limited entry decision tables*. Tables which use multiple conditions are known as *extended entry decision tables*. One important aspect to notice about decision tables is that they aren't imperative as that they don't apply any particular order over the conditions or actions.

2.2.2 Cause-Effect Graphing?

Cause-Effect Graphing is very similar to Decision Table-Based Testing, where logical relationships of the inputs produce outputs; this is shown in the form of a graph. The graph used is similar to that of a Finite State Machine (FSM). Symbols are used to show the relationships between input conditions, those symbols are similar to the symbols used in propositional logic.

2.3 Functional Relationships

There are 3 main functional methods:

- Boundary Value Analysis (BVA)
- Equivalence Class Testing (ECT)
- Decision Table-Based Testing (DT-BT)

All three functional testing methods compliment each other; the functional testing outcome can not be completed to a satisfactory level using just one of these functional testing strategies, or even two.

Decision Table-Based Testing has evolved from Equivalence Class Testing in some way; Equivalence Class Testing groups together inputs of the same manner which behave similarly. DT-BT follows on from ECT by grouping together the input and output behaviours into an "equivalence" rule and testing the logical dependencies of these rules. These rules are regarded as test cases, therefore redundant rules are discarded.

2.3.1 Effort

Although all three testing strategies have similar properties and all work towards the same goal, each of the methods is different in terms on application and effort.

Boundary Value Analysis is not concerned with the data or logical dependencies as it's a domain based testing technique. It requires low effort to develop test cases for this method but on the other hand its sophistication is low and the number of test cases generated is high compared with other functional methods.

Equivalence Class Testing is more concerned with data dependencies and treating similar inputs and outputs the same by grouping them in classes. This reduces the test cases and increases the effort used to create test cases due to the effort required to group them. This is a more sophisticated method of test case development as it's more concerned with the values inside of the domain.

Decision Table-Based Testing on the other hand uses similar traits as Equivalence Class Testing; it tests logical dependencies, which increases the effort in identifying test cases, which increases the sophistication of these test cases. Because DT-BT relies more on the logical dependencies of the equivalence classes in the decision table this reduced the amount of rules required to complete the set of test cases.

Boundary Value Analysis: - A functional testing strategy which is concerned with the limits of an input/output domain.

Equivalence Class Testing: - A functional testing strategy where the inputs/outputs that behave similarly are grouped together into equivalence partitions, in order to decrease test cases.

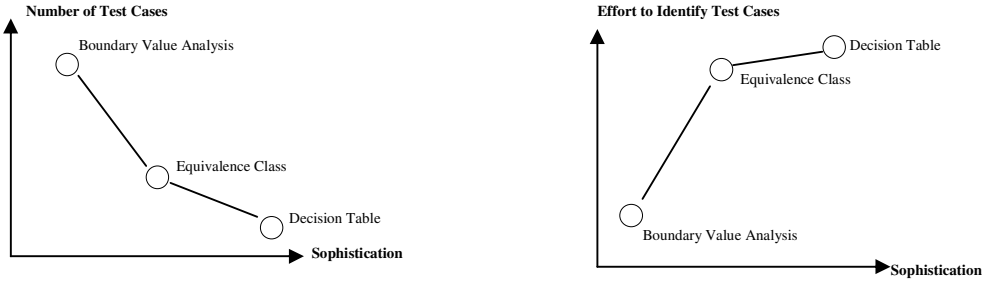

Figure 2.3 Graphs showing the relationships of all functional methods

2.3.2 Efficiency

In order to give a sense of how efficient Decision Table-Based Testing is with respect to other functional methods, Boundary Value Analysis and Equivalence Class Testing have to be examined.

On average Boundary Value Analysis yields 5 times as many test cases as Decision Table- Based Testing, and Equivalence Class Testing 1½ times as many test cases. On this basis we can say that there exists either test case redundancy or impossible test cases, either way this reduces the efficiency of these testing strategies and shows how efficient Decision Table-Based Testing is.

But as stated above, we cannot totally disregard the other functional testing methods as they complement each other and are not totally redundant in all testing cases.

3 Applications

In order to demonstrate and aid the understanding of Decision Tables I will show the some of the many applications it has and aid them with examples. I am going to use the Triangle Problem to explore decision tables in more depth.

3.1 The Basics

As explained above, there are two types of decision table, limited and extended entry tables. Below, in fig 3.1 is an example of a limited entry decision table where the inputs are depicted using binary values.

c1: a<b+c?	F	T	T	T	T	T	T	T	T	T	T
c2: b<a+c?	-	F	T	T	T	T	T	T	T	T	T
c3: c<a+b?	-	-	F	T	T	T	T	T	T	T	T
c4: a = b?	-	-	-	T	T	T	T	F	F	F	F
c5: a = c?	-	-	-	T	T	F	F	T	T	F	F
c6: b = c?	-	-	-	T	F	T	F	T	F	T	F
a1: Not a Triangle	X	X	X								
a2: Scalene											X
a3: Isosceles							X		X	X	
a4: Equilateral				X							
a5: Impossible					X	X		X			

Fig 3.1 Decision Table for the Triangle Problem

When creating a decision table there are many techniques people adopt to improve the construction. Most testers add two main techniques, the use of the "impossible" action stub and don't care entries. The impossible action stub entry is used as a form of error catching, if out of range values are inputted then the impossible action entry is checked. Don't care entries are also another useful procedure, they are used when no other checks are required in the table, and therefore we don't care what the rest of the values are. Often, these don't care entries are referred to as false values.

3.2 Rule Counts

Rule counts are used along with don't care entries as a method to test for decision table completeness; we can count the amount of test cases in a decision table using rule counts and compare it with a calculated value. Below is a table which illustrates rule counts in a decision table.

c1: a<b+c?	F	T	T	T	T	T	T	T	T	T	T
c2: b<a+c?	-	F	T	T	T	T	T	T	T	T	T
c3: c<a+b?	-	-	F	T	T	T	T	T	T	T	T
c4: a = b?	-	-	-	T	T	T	T	F	F	F	F
c5: a = c?	-	-	-	T	T	F	F	T	T	F	F
c6: b = c?	-	-	-	T	F	T	F	T	F	T	F
Rule Count	32	16	8	1	1	1	1	1	1	1	1
a1: Not a Triangle	X	X	X								
a2: Scalene											X
a3: Isosceles							X		X	X	
a4: Equilateral				X							
a5: Impossible					X	X		X			

Fig 3.2 an example of Rule Counts in a Decision Table

The table above has a total rule count of 64; this can be calculated using the limited entry formula as it's a limited entry table.

$$\text{Number of Rules} = 2^{\text{Number of Conditions}}$$

So therefore, $\text{Number of Rules} = 2^6 = 64$

When calculating rule counts the don't care values play a big part to the rule count of that rule. Each "don't care" entry in a rule doubles the rule count of that rule; so, each rule has a rule count of 1 initially, if a "don't care" entry exists then this rule count doubles for every "don't care" entry. So using both ways of computing the rule count brings us to the same value which shows we have a complete decision table.

Where the Rule Count value of the decision table does not equal the number of rules computed by the equation we know the decision table is not complete, and therefore needs revision.

3.3 Redundancy & Inconsistency

When using "don't care" entries a level of care must be taken, using these entries can cause redundancy and inconsistency within a decision table.

Using rule counts to check the completeness of the decision table can help to eliminate redundant rules within the table. An example of a decision table with a redundant rule can be seen in figure 3.3.

From the table you can see that there is some conflict between rules 1-4 and rule 9, rules 1-4 use "don't care" entries as an alternative to false, but rule 9 replaces those "don't care" entries with "false" entries. So when condition 1 is met rules 1-4 or 9 may be applied, luckily in this particular instance these rules have identical actions so there is only a simple correction to be made to complete the following table.

Conditions	1 to 4	5	6	7	8	9
c1	T	F	F	F	F	T
c2	-	T	T	F	F	F
c3	-	T	F	T	F	F
a1	X	X	X	-	-	X
a2	-	X	X	X	-	-
a3	X	-	X	X	X	X

Figure 3.3 an example of a Redundant Rule

If on the other hand the actions of the redundant rule differ from that of rules 1-4 then we have a problem. A table showing this can be seen in figure 3.4.

Conditions	1 to 4	5	6	7	8	9
c1	T	F	F	F	F	T
c2	-	T	T	F	F	F
c3	-	T	F	T	F	F
a1	X	X	X	-	-	-
a2	-	X	X	X	-	X
a3	X	-	X	X	X	-

Figure 3.4 an example of Inconsistent Rules

From the above decision table, if condition 1 was to be true and conditions 2 and 3 were false then rules 1-4 and 9 could be applied. This would result in a problem because the actions of these rules are inconsistent so therefore the result is non-deterministic and would cause the decision table to fail.

3.4 Creating a Decision Table

When creating a decision table care must be taken when choosing your stub conditions, and also the type of decision table you are creating. Limited Entry decision tables are easier to create than extended entry tables. Here are some steps on how to create a simple decision table using the Triangle Problem.

Step One – List All Stub Conditions

In this example we take three inputs, and from those inputs we perform conditional checks to calculate if it's a triangle, if so then what type of triangle it is. The first condition we add must check whether all 3 sides constitute a triangle, as we don't want to perform other checks if the answer is false.

Then the remainder of the conditions will check whether the sides of the triangles are equal or not. As there are only three sides to a triangle means that we have three conditions when checking all of the sides.
So the condition stubs for the table would be:

- a, b, c form a triangle?
- a = b?
- a = c?
- a = c?

Step Two – Calculate the Number of Possible Combinations (Rules)

So in our table we have 4 condition stubs and we are developing a limited entry decision table so we use the following formula:

$$\text{Number of Rules} = 2^{\text{Number of Condition Stubs}}$$

So therefore, $\text{Number of Rules} = 2^4 = 16$

So we have 16 possible combinations in our decision table.

Step Three – Place all of the Combinations into the Table

c1: a,b,c form a triangle?	N	Y	Y	Y	Y	Y	Y	Y	Y
c2: a = b?	-	Y	Y	Y	Y	N	N	N	N
c3: a = c?	-	Y	Y	N	N	Y	Y	N	N
c4: b = c?	-	Y	N	Y	N	Y	N	Y	N

Figure 3.5 a Complete Decision Table

Here we have a complete decision table as there are three don't care entries which gives the first rule a rule count of 8 and the last 8 rules have a rule count of 1 each, so the total rule count for the table is 16. Therefore we know that this table is complete.

Step Four – Check Covered Combinations

This step is a precautionary step to check for errors and redundant and inconsistent rules. We don't want to go any further with the development of the decision table if we have errors because this will complicate matters in the next step.

Step Five – Fill the Table with the Actions

For the final step of creating a decision table we must fill the Action Stub and Entry sections of the table. The final decision table is shown in fig 3.6.

After completing the decision table and adding the actions we notice that each action stub is exercised once, and we have also added the "impossible" action into the table or catching rogue values.

c1: a,b,c form a triangle?	N	Y	Y	Y	Y	Y	Y	Y	Y
c2: a = b?	-	Y	Y	Y	Y	N	N	N	N
c3: a = c?	-	Y	Y	N	N	Y	Y	N	N
c4: b = c?	-	Y	N	Y	N	Y	N	Y	N
a1: Not a Triangle	X								
a2: Scalene									X
a3: Isosceles					X		X	X	
a4: Equilateral		X							
a5: Impossible			X	X		X			

Figure 3.6 the Final Decision Table

The above table can be explored and expanded by refining the first condition stub. Instead of having "a, b, c form a triangle" we can expand this by using 3 conditions rather than one, which will increase accuracy. This would also bring in a logical dependency, because the actions of the first condition stub would affect the remaining condition stubs. This is shown in figure 3.1.

4 Summary & Future Work

Decision Table-Based Testing is an important part of Functional Testing; it explores testing routes that other functional strategies avoid. One key aspect of decision table-based testing is the use of logical dependencies; this enhances the tester's ability to solve inputs in a program which relies upon other inputs to perform its operation, which is a strong characteristic in testing nowadays.

DT-BT is the most complete method of all of the functional testing strategies as it encourages strict logical relationships between conditions. Creating these logical dependencies can be tricky especially for difficult and extensive programs. It works well with the Triangle problem as there are lots of decisions within the problem.

The difference between the functional testing strategies were outlined and shown in this report, we saw the difference in effort, sophistication and number of test cases these functional methods create. This illustrates that decision table-based testing is the final step of the functional testing process. There are many testing tools available for creating decision tables which are excellent for new users to become accustomed to using this functional technique.

This report has outlined the importance of testing and time required when creating software. I aim to make use of the knowledge I have gained while writing this report, and apply to my final year dissertation. For my dissertation I am developing a system which navigates to users to rendezvous with each other using musing as their cue. This insight will aid me to create a reliable software package as I can use the input values from devices which my software uses.

References

[1] Jorgensen, Paul C., Software Testing: A Craftsman's Approach, 2nd Edition, CRC Press. July 2002.

[2] Myers, Glenford J., The Art of Software Testing, 2nd Edition, John Wiley & Sons Canada, Ltd. June 2004.

[3] Mosley, Daniel J., The Handbook of MIS Application Software Testing, Yourdon Press, Prentice Hall. 1993.