Test Design Techniques

Versión	2009-01-29

Version History

Date	Author	Changes Resume	Modify Sections
2009-01-29	Katherine Prendice	Initial Draft	All

Approvals

Nombre	Puesto	Firma	Fecha

TABLE CONTENTS

1 Introdution

1.1. Purpose

The purpose of this document is defined what is a test design technique, how and when to used and finally to develop by examples the most used techniques.

1.2. Audience

The audience of this document is the QA Director, QA Manager, QA Engineers and any other member that wants to know about test design technique.

1.3. Sope

The scope of this document is to develop the idea of test design techniques and learn about the most used techniques.

1.4. Teminology

Término	Definición
QAD	QA Director
QAM	QA Manager
QAE	QA Engineers

2 Test design technique

In general words a test design technique is a method used to create test cases from any kind of information or references like requirements, specifications, manual, workflows, etc.

The big challenge of these techniques is to indentify and select effective test cases that will allow users to find as much error as they can without an excessive number of executions.

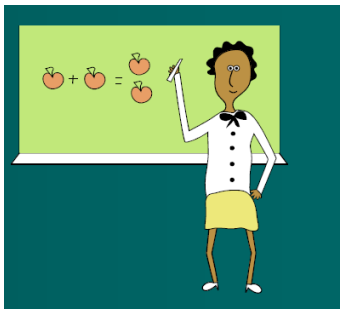
In order to affront this challenge many groups of definitions appears like the well-known black and white box, however these simple definitions can carry on a lot of confusions because many authors put the same techniques in both classification or we get used to related white boxes only with development code.

So the inevitable question appears: How and when used a specific test design technique? …

"How and when apply a technique depend on the model we can build"

*"How and when apply a technique **doesn't** depend on where we take the information (black or white box)"*

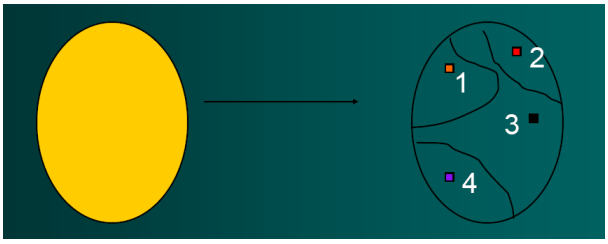
2.1 Model #1: "Domains with inputs and outputs that returns equivalent behaviors"



The groups of techniques used for this model is called "Domain Testing" and the most well-known are:

1. Equivalent Partitions
2. Boundary Value Analysis

A domain testing allows user to take an input data domain and divide it in equivalent parts and then choose a specific test case for each part.



These techniques can be used in the next scenarios:

- When the testing base has inequalities, equalities or domains (e.g. 0<x<=20)
- When there is a lot of logical conditions (e.g. it will rain, it will not rain)
- When there are validations and classification of input values (e.g. input correct values, input incorrect values)

2.1.1 Equivalent Classes

In order to explain how the test cases can be derivate using this technique we need first understand the idea of equivalent partition and then we will proceed to define the heuristic for test cases derivation.

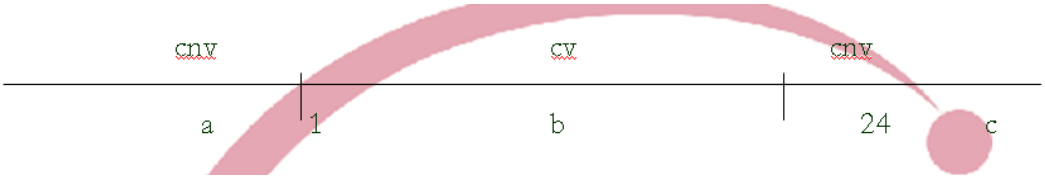
1. *What is an Equivalent Partition?*

   An equivalent partition is a group of data, which have an analog or equivalent behavior that means each member of the group has the same meaning for a specific situation.

   For example, if "x" represent the range of values 0<x<26 is possible say the range [1..24] represents the group of value that belong to that condition so it is consider as an equivalent partition.

   Also is possible identify the range of value]-∞,0] and [25,∞[represent equivalent partitions that don't belong to that condition

   The situation decrypted can be represented by the graphic:



   "a": It is a value representative of the invalid class]-∞,0]

   "b": It is a value representative of the valid class [1..24]

   "c": It is a value representative of the invalid class [25,∞[

   Additionally there are some persons who consider a no enter number (real number or alpha characters) can be considered as other invalid class.

2. *Considerations to identify Equivalent Partitions*

There are three main considerations:

a) If there is a value or range of value then you will have two invalid classes (CNV) and one valid class (CV)

For example: Number of exams (X)

CV: X=2 CNV: X>2 CNV: X<2

b) If there are enumerate values you will have as many valid classes (CV) as elements you will enumerated and only one invalid class (CNV)

For example: Thesis final grades (X)

CV: X= Fail CV: X= Pass CV: X= Pass with honors CNV: X= Others

c) If there is about a situation that must occur you will have one valid class (CV) and one invalid classs (CNV)

For example: Files name (X)

CV: X= File has a name CNV: X= File doesn't have a name

3. *The Heuristic*

Test cases derivation is based on the heuristic to determinate which test cases are really necessary.

Suppose you have two parameters:

Parameter 1: CV: a CNV: b

Parameter 2: CV: c CNV: d

At beginning you will consider all the possible combination in order to create your test cases, that means:

(a,c) (a,d) (b,c) (b,d)

But the question is: Are all these combination really necessary? Why?

Case	Discussion
(a,c)	Knowing “a” and “c” correspond to a valid class you can then expect the result of this process will be pass successfully.
(a,d)	Knowing “a” is a valid class and “d” is an invalid class then this case will fail because of “d”
(b,c)	Knowing “c” is a valid class and “b” is an invalid class then this case will fail because of “b”
(b,d)	Knowing “b” and “c” are invalid classes then how you can know which one make your case fail? Also you have already tested b and d in the previous cases, so there are not reasons to believe this a recommendable case.

Just to consolidate what we have seen, we will check another example with parameters that have more options:

Parameter 1: CV: a,c,e CNV: b,d

Parameter 2: CV: f CV:g

The possible combination will be:

(a,f)(b,f)(c,f)(d,f)(e,f) (a,g)(b,g)(c,g)(d,g)(e,g)

Now we will analyze each case

#	Case	Analysis	Results
1	(a,f)	Both are CV	Useful. Pass successfully (+)
2	(b,f)	B is CNV and F is CV	Useful. Fail b(-)
3	(c,f)	Both are CV	Useful. Pass successfully (+)
4	(d,f)	D is CNV and F is CV	Useful. Fail d(-)
5	(e,f)	Both are CV	Useful. Pass successfully (+)
6	(a,g)	A is CV and G is CNV	Useful. Fail g(-)
7	(b,g)	Both are CNV	Not useful. You don’t know if fail because of b or g
8	(c,g)	C is CV and G is CNV	Not useful. Because even when fail because of “g”, this has been tested in case 6
9	(d,g)	Both are CNV	Not useful. You don’t know if fail because of d or g
10	(e,g)	E is CV and G is CNC	Not useful. Because even when fail because of “g”, this has been tested in case 6

After see this example the next question is: *How do I know the number of cases functional correct (+) and functional incorrect (-)?*

a) For the functional correct cases, you have to identify the parameter which has the biggest number of valid classes (CV)

b) For the functional incorrect cases, you have to count the total number of invalid classes (CNV)

So in our example we have three functional correct cases (+) and three functional incorrect cases (-)

Generalizing the Heuristic we have:

- Identify the equivalent partitions for each parameter and assign an identification value for each one
- Write one or more cases that cover the most quantity of CV. Remember the number of relevant functional correct cases is determinate by the parameter which have the biggest number of valid cases
- Write one case for each CNV considering the rest of values must be CV. Remember that for a negative case has sense, only one CNV must be considered in the combination.

4. *Test Catalog*

A test catalog is a list of several test cases that will be considered during the testing process. The catalog could have several columns depending on organization's necessities; however a common test catalog has the next distribution:

# Test Case	Equivalent Partition	Values	Expected Result	Observation
1	(a,b)	(a=100,b=20)	The process is success	Any possible observation
2	(a,c)	(a=100, c=5)	The process fails because of C	

5. Application

A bank process required the next values to complete a process:

Bank Code: Enter number xyz/ x>1

Branch Code: Enter number xyzt / x>0

Personal Code: An alphanumeric string with at least 5 positions

- Identify the parameters and their valid and invalid class:

  Bank Code:

cv	200 <=xyz<=999	1
cnv	xyz >999	2
cnv	xyz <200	3

Branch Code:

cv	1000<=xyzt<=9999	4
cnv	xyzt >9999	5
cnv	xyzt <1000	6

Personal Code:

cv	5 character	7
cnv	5 characters	8

- Identify the number of functional correct and incorrect cases:

  # Functional Correct Cases: 1

  # Functional Incorrect Cases: 5

- Develop the Test Catalog

#Caso	Clases Equivalentes	Valores	Resultado Esperado	Observación
1	1,4,7	359 (1), 2000 (4), evalu(7)	OK. Record	
2	2,4,7	1000 (2), 2000(4), evalu(7)	No Record: “Invalid Bank Code”	
3	3,4,7	50(3), 2000(4), evalu(7)	No Record: “Invalid Bank Code”	
4	5,1,7	aaa(5), 359(1),evalu(7)	No Record: “Invalid Branch Code””	Branch Code must be numbers
5	6,1,7	10(6),359(1), evalu(7)	No Record: “Invalid Branch Code”	Branch Code must be more than 1000
6	8,1,4	yyy(8), 359(1),1500(4)	No Record: “Invalid Personal Code”	Personal code must have 5 characters

2.1.2 Boundary Value Analysis (BVA)

Boundary value analysis is a methodology for designing test cases that concentrates software-testing effort on cases near to limits of valid ranges.

The reason why boundary conditions are very important for testing is because defects could be introduced at the boundaries very easily and this technique allows users to make sure the behavior of system is predictable for the input and output boundary conditions.

1. *Applying Boundary Value Analysis*

   Similar than equivalent partitions the boundary value analysis also begging with the identification of our parameters and their valid and invalid classes.

   Suppose you have the next parameters:

   Parameter 1: $20 \leq X \leq 40$

   Identifying their valid and invalid classes we will have:

   Parameter 1:

CV	$20 \leq X \leq 40$
CNV	$X > 40$
CNV	$X < 20$

   Once you have indentified this, the boundary value analysis determines that you need to focus on extremities values. These extremities values are:

   - Min ..... Minimal
   - Min - ..... Just above Minimal
   - Nom ..... Average
   - Max - ..... Just below Maximum
   - Max ..... Maximum
   - Max + ... Just above Maximum

Following our case we will have the next cases:

Parameter 1:

Min: 20 Max - : 39

Min - : 19 Max: 40

Nom: 21 Max +: 41

Finally the test cases used for the testing will be

# Test Case	Cases	Expected Result	Observation
1	X=20	The test pass	N.A
2	X=19	The test failure	
3	X = 21	The test pass	
4	X = 39	The test pass	
5	X = 40	The test pass	
6	X = 41	The test failure	

Notice obviously the Expected Result and Observation columns will change according with the user expects.

2. *Generalizing the Steps:*

Just to consolidate what we have seen lets consolidate the require steps to use the boundary analysis value technique:

- Identify your equivalent partitions

  Indentify your parameters

  Indentify their valid and invalid classes
- From the classes obtained select the boundary values
- Focus on the values close to the edge of those ranges (Min -, Min, Nom, Max -, Max and Max +)
- Always remember: *" If your limits values is N, then the cases you can select are from N+1 to N-1"*

3. *Application:*

   The Next Date problem is a function of three variables: day, month and year. Upon the input of a certain date it returns the date of the day after that of the input.

   The input variables have the obvious conditions:

   $1 \leq Day \leq 31$ $1 \leq month \leq 12$ $1812 \leq Year \leq 2012$

   - Identify your equivalent partitions and their valid and invalid classes:

     Day

CV	$1 \leq Day \leq 31$
CNV	$Day < 1$
CNV	$Day > 31$

     Month

CV	$1 \leq month \leq 12$
CNV	$month < 1$
CNV	$month > 12$

     Year

CV	$1812 \leq Year \leq 2012$
CNV	$Year < 1812$
CNV	$Year > 2012$

- Select boundary values for each class obtained. Focus on the values (Min -, Min, Nom, Max -, Max and Max +)

Day:	Month:	Year
Min =1	Min = 1	Min = 1812
Min + = 2	Min + = 2	Min + = 1813
Nom = 15	Nom = 6	Nom = 1912
Max - = 30	Max - = 11	Max = 2011
Max = 31	Max = 12	Max = 2012
Max+ = 32	Max+ = 13	Max+ = 2013

- In this case the variables interacted between them, so it is necessary create a combination with the boundary values obtained to create our test cases.

  An extract of the combination's table will result like this:

Boundary Value Analysis Test Cases

Case	month	day	year	Expected Output
1	6	15	1812	June 16, 1812
2	6	15	1813	June 16, 1813
3	6	15	1912	June 16, 1912
4	6	15	2011	June 16, 2011
5	6	15	2012	June 16, 2012
6	6	1	1912	June 2, 1912
7	6	2	1912	June 3, 1912
8	6	30	1912	July 1, 1912
9	6	31	1912	error
10	1	15	1912	January 16, 1912
11	2	15	1912	February 16, 1912
12	11	15	1912	November 16, 1912
13	12	15	1912	December 16, 1912

2.2 Model #2: “Table Decision, Decision Trees, Karnaugh maps, etc”

The groups of techniques used for this model is called “Combination Testing” and the most well-known are:

1. Decision table test
2. Cause effect graphing

We use the combination testing to find a several kinds of errors like:

- Missing scenarios
- Errors because ambiguities specifications
- Assign incorrect values to a decision variable.

These techniques can be used in the next scenarios:

- For scenarios where based on an input, parameter or variable combination a specific action or response is chosen.
- For scenarios where the response or action doesn’t depend on the order in which the values of parameter or variables are evaluated.
- For scenarios where the response or action doesn’t depend on previous inputs or outputs.

2.2.1 Decision table test

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 as table is split up into four sections as shown belong:

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

This table should be read by columns of rules: Rule 1 says that when all conditions except c3 are true then actions a1 and a3 occurs

Also 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.

1. *Applying Decision table test*

The triangle problem accepts three integers (a, b and c) as its input, each of which are taken to be sides of a triangle. The values of these inputs are used to determine the type of the triangle (Equilateral, Isosceles, Scalene or not a triangle)

For the inputs to be declared as being a triangle they must satisfy the six conditions:

a < b+ c; b< a+c; c<b+a; a=b; a=c; b=c

The triangles types are: Scalene, Isosceles and Equilateral

To create the decision table we follow the next steps:

i. List all causes in the decision table
ii. Calculate the number of possible combinations
iii. Fill columns with all possible combinations
iv. Reduce test combinations
v. Check covered combinations
vi. Add effects to the table

i. *List all causes in the decision table*

The causes we need to considerer are:

C1: a<b+c C4: a = b

C2: b<a+c C5: a = c

C3: c<a+b C6: b = c

To write all these causes in the decision table we need to consider the next tips:

- Write down the values the cause/condition can assume
- Cluster related causes
- Put the most dominating cause first
- Put multi valued causes last

Then our decision table will look like this:

Cause	**Values**		
C1: a<b+c			
C2: b<a+c			
C3: c<a+b			
C4: a = b			
C5: a = c			
C6: b = c			
Effects			
A1: Not triangle			
A2 : Scalene			
A3 : Isosceles			
A4: Equilateral			
A5 : Impossible			

ii. *Calculate the number of possible combinations*

If all causes are simply Y/N values then the number of possibilities will be:

$2^{\text{number of causes}}$

If 1 cause with 3 values and 3 with 2 the number of possibilities will be:

$3^1 * 2^3 = 24$

For our case we will have: $2^6 = 64$ rules

iii. *Fill columns with all the possible combinations*

Cause	Values										
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
C1: a<b+c	F	T	T	T	T	T	T	T	T	T	T
C2: b<a+c	F	F	T	T	T	T	T	T	T	T	T
C3: c<a+b	F	F	F	T	T	T	T	T	T	T	T
C4: a = b	F	F	F	T	T	T	T	F	F	F	F
C5: a = c	F	F	F	T	T	F	F	T	T	F	F
C6: b = c	F	F	F	T	F	T	F	T	F	T	F
Effects											
A1: Not triangle											
A2 : Scalene											
A3 : Isosceles											
A4: Equilateral											
A5 : Impossible											

Notice in this fillet we have in consideration that if the three first options are true, then they are definitely a triangle, so you don't need to complete any other possibilities, but you start with the combination of the next three options to determine which kind of triangle it is.

iv. *Reduce combinations*

- Find “don’t care” entries and put “-“ on them
- Join columns where their values are identical

Cause	Values										
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
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
Effects											
A1: Not triangle											
A2 : Scalene											
A3 : Isosceles											
A4: Equilateral											
A5 : Impossible											

Notice in column 1, 2 and 3, we have set “don’t care” entries because these options determine if we are talking about a triangle or not. In case one of these options is false then it doesn’t matter if the rest options are False or true, it is definitely not a triangle.

v. *Checked coverage combinations*

When the calculating rule counts the “don’t care” values play a big part to the rule count of that rule (or check sum). 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 ($2^{\text{don't care}}$). 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

In our example we will have:

Cause	**Values**										
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
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

Rule Count = 32+16+8+1+1+1+1+1+1+1+1 = 64 = 2^6

vi. *Add effects to the table*

- Read column by column and determine the effects.
- One effect can occur in multiple test combinations.

Cause	Values										
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
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
Effects											
A1: Not triangle	X	X	X								
A2 : Scalene											X
A3 : Isosceles							X		X	X	
A4: Equilateral				X							
A5 : Impossible					X	X		X			

2. *Application*

Analysis the next case:

A mailing is to be sent out to customers. The content of the mailing is about the current level of discounting and potential levels of discounting. The content is different for different types of customers.

Customers type A, B and C get a normal letter except customer type C, who get a special letter.

Any customer with 2 or more current lines or with a credit rating of "X" get a special paragraph added with an offer to subscribe to another level of discounting.

- List all causes in the decision table

  C1: Customer Type — Values: A, B, C, O (Other customer)

  C2: 2 or more lines — Values: Y / N

  C3: Credit rating (X) — Values: Y/N

- Calculate the number of possible combinations

  $2^{2*} 4^{1} = 16$ combinations

- Fill columns with all possible combinations

Cause	**Combinations**															
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Customer Type	A	A	A	A	B	B	B	B	C	C	C	C	O	O	O	O
2 or more line	Y	Y	N	N	Y	Y	N	N	Y	Y	N	N	Y	Y	N	N
Credit Rating	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N

- Check covered combinations

Cause	**Combinations**															
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Customer Type	A	A	A	A	B	B	B	B	C	C	C	C	O	O	O	O
2 or more line	Y	Y	N	N	Y	Y	N	N	Y	Y	N	N	Y	Y	N	N
Credit Rating	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N
Rule Count	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Rule Count = $1*16 = 2^{2*} 4^{1} = 16$ combinations

- Add effects to the table

Cause	Combinations															
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Customer Type	A	A	A	A	B	B	B	B	C	C	C	C	O	O	O	O
2 or more line	Y	Y	N	N	Y	Y	N	N	Y	Y	N	N	Y	Y	N	N
Credit Rating	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N
Rule Count	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Effects																
Normal Letter	X	X	X	X	X	X	X	X					?	?	?	?
Special Letter									X	X	X	X	?	?	?	?
Add. Paragraph	?	X	X		?	X	X		?	X	X		?	?	?	?
No letter													?	?	?	?

Notice “?” are behavior that have not been defined and help us to wonder if they are useful scenarios or not.

3. Additional Information

It is consider there are 3 main functional techniques:

- Boundary Value Analysis (BVA)
- Equivalence Classes
- Decision Table

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

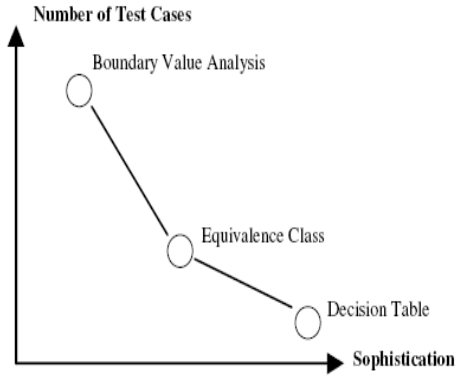
Although all three testing strategies have similar properties and all work towards the same goal, each of the methods is different in terms of application and effort.

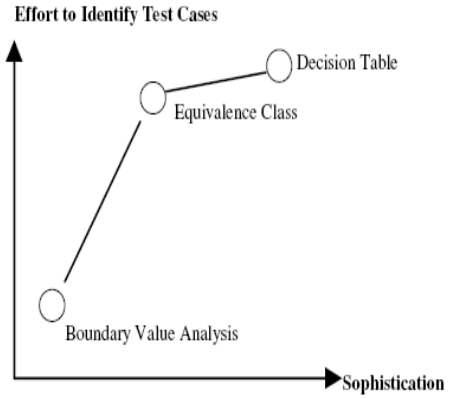
Boundary Value Analysis; is not concerned with the data or logical dependencies as it's a domain based testing technique. It requires low effort to develop test cases for this method, but on the other hand its sophistication is low and the number of test cases generated is high compared with other functional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

Equivalent Classes; 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; uses similar traits as Equivalent Partition; it test logical dependencies, which increases the effort in identifying test cases, which increases the sophistications of these test cases. Because Decision table 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.

The graph below is showing the relationship of these three techniques:





2.2.2 Cause effect graphing

Cause-Effect graphing is very similar to Decision Table 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 and conditions, those symbols are similar to the symbols used in propositional logic.

Cause-Effect graphing also has the ability to detect defects that cancel each other out, and the ability to detect defects hidden by other things going right.

Any type of logic can be modeled using a Cause-Effect diagram. Each cause (or input) in the requirement is expressed in the cause-effect graph as a condition, which is either true or false. Each effect (or output) is expressed as a condition, which is either true or false.

4. Applying Cause-Effect Graphing

We have a requirement that say: "If A or B, then C"

The following rules hold for these requirements:

If A is true and B is true, then C is true

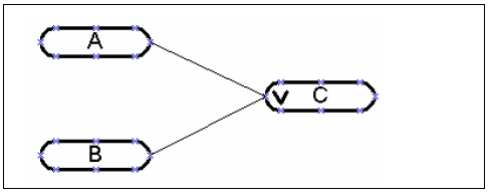
If A is true and B is false, then C is true

If A is false and B is true, then C is true

If A is false and B is false, then C is false

i. Generated the cause effect graph

The cause effect graph that represents this requirement will be:



In this figure A, B and C are called nodes. Nodes A and B are the causes, while C is an effect. Each node can have a true or false condition. Then lines, called vectors, connect the cause nodes A and B to the effect node C

In general (for any case) all requirements are translated into nods and relationships on the cause-effect graph. There are four possible relationships among nodes, and they are indicated by the following symbols:

Where A always leads to C, a straight line: ——————

Where A **OR** B lead to C, a "**v**" at the intersection mean "or" (similar than in our example)

Where A **AND** B lead to C, a "□" at the intersection mean "and"

A tilde **~** means "**NOT**" as in "If not A, then C"

ii. *Generated the Decision Table:*

The cause-effect graph is then converted into a decision table representing the logical relationship between the causes and effects. Each column of the decision tables is a test case. Each test case corresponds to a unique possible combination of inputs that are either in a true state, a false state, or a masked state.

Cause	**Values**		
	1	**2**	**3**
A	T	F	F
B	F	T	F
Effect			
C	T	T	F

Since there are two inputs to this example, there are $2^{\text{number of causes}}$: $2^2 = 4$.

Notice that there are only three test cases. However, these three test cases cover 100% of the functionality for this example. The fourth combination (A=True, B=True), does not add any additional functional coverage, and is a redundant test case.

Also notice if we have entered all the possibilities and then apply the "Rule count", review in Decision table technique, this would matched with the number of combination found = 4.

5. *Application*

Suppose you are planning a discount's campaign for two different products (PA and PB).

The rules of this campaign are:

- If a client buys less than 10 units of PA or less than 7 units of PB there will be not discounts
- If a client buys more than 10 units of PA or more than 7 units of PB then we will have a discount of 12%
- If a client buys more than 10 units of PA and also more than 7 units of PB then we will have a discount of 15%
- If a client buys more than 20 units of PA then we will have a discount of 18%

Also there is a policy for clients who buy PA with their gold card:

- If a client buys more than 20 units of PA with his gold card then we will have a discount of 20%

i. Generated the cause effect graph

Causes

A: less 10 units PA

B: less 7 units PB

C: more 10 units PA

D: more 7 units PB

E: Golden Card

F: more 20 units PA

Actions

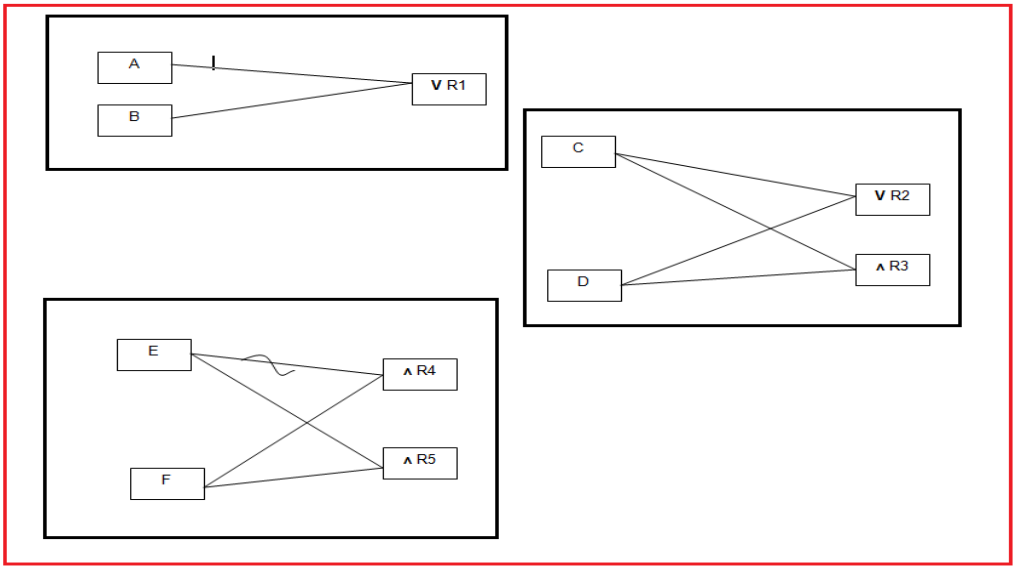
R1: No discount

R2: 12% discount

R3: 15% discount

R4: 18% discount

R5: 20% discount


A
B
V R1
C
D
V R2
^ R3
E
F
^ R4
^ R5

ii. *Generated the decision table*

Remember that to generate this decision table you must use the cause-effect graph you have created. Other possibilities are not considered because they are not important for our case.

Cause	**Values**								
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
A: less 10 units PA	T	T	F	-	-	-	-	-	-
B: less 7 units PB	T	F	T	-	-	-	-	-	-
C: more 10 units PA	-	-	-	T	T	F	T	-	-
D: more 7 units PB	-	-	-	T	F	T	T	-	-
E: Golden Card	-	-	-	-	-	-	-	F	T
F: more 20 units PA	-	-	-	-	-	-	-	T	T
Effects									
R1: No discount	X	X	X						
R2: 12% discount				X	X	X			
R3: 15% discount							X		
R4: 18% discount								X	
R5: 20% discount									X

Remember the symbol "-" represents the "don't care"

2.3 Model #3: “State transactions diagrams”

The groups of techniques used for this model is called “State transition testing” and the most well-known is the one who has the same name:

1. State transition testing

This technique allow user to view the software under test in term of its states, transitions between states, and the inputs or events that trigger state changes.

We use the state transition testing to find errors like:

- Identify missing transactions
- Dead states (state where once entered can not be exited)

The typical scenarios where this technique is used are:

- For bank transactions
- Electronic devices
- Reservation systems

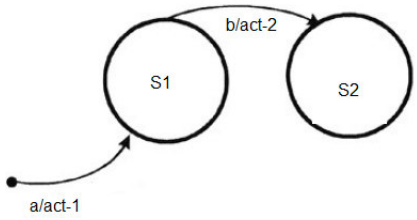
2.3.1 State transition testing

State transition diagrams are an excellent tool to capture certain types of system requirements and to document internal system design.

It views the software under test in term of its state, transitions between states and the inputs or events that trigger state changes.

A states transition graph (STG) can be designed for the whole software system or for its specific modules. The STG graph consist of nodes (circles, ovals, rounded, rectangles) that represent states and arrows between nodes that indicate what input (event) will cause a transition between two linked states.

The next figure is a typical state transition graph:



After create the state transition graph is possible to design the state transition table. The advantage of state-transition table is that it lists all possible state-transition combination, not just the valid ones.

Creating a state-transition table often unearth combinations that were not identified, documented, or dealt with the requirements. It is highly beneficial to discover these defects before coding begins.

Using a state-transition table can help detect defects in implementation that enable invalid paths from one state to another.

1. *Applying State transition testing*

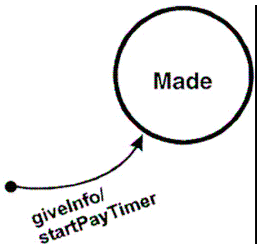
   Make a Reservation:

   - Provide information including departure and destination cities, dates and times.
   - A reservation agent uses that information to make a reservation
   - At that point, the reservation is in the Made state. The system creates and starts a timer
   - If this timer expires before reservation is paid for, the reservation is cancelled by the system.
   - Also is possible the customer cancel the ticket :
     - Before paid
     - After paid and before the ticket was printed
     - After paid and after the ticket was printed.

i. *Identify states*

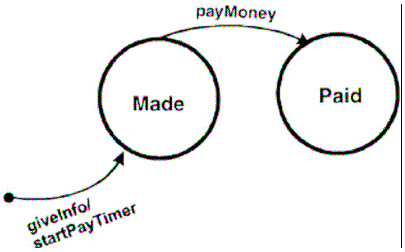
A. The reservation is made:

In the figure bellow “giveInfo”, is an event that comes into the system from the outside world. The command after the “/” denotes an action of the system; in our case “startPayTimer”

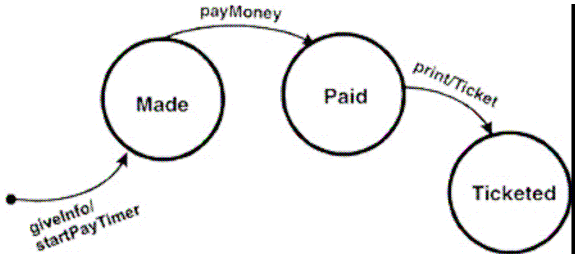


B. Pay for the reservation – PayMoney

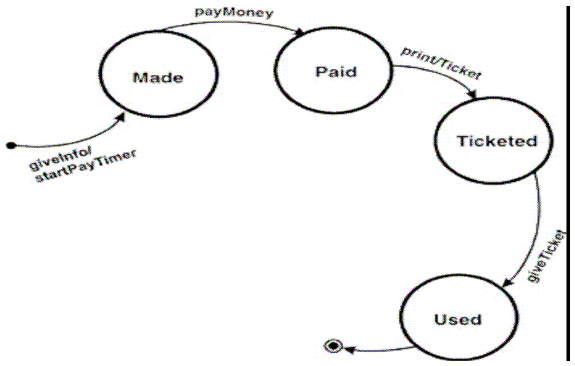
Events may have parameters associated with them. For example: PayMoney, may indicate Cash, Check, Debit Card or Credit Card.



C. Print the ticket – Print

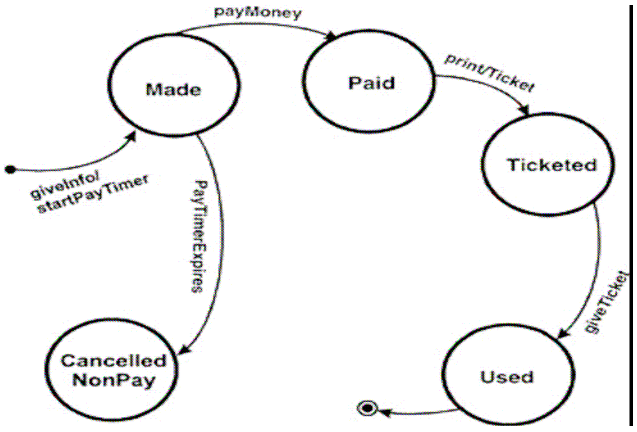


D. Give ticket (Give ticket to the gate agent to board the plane)

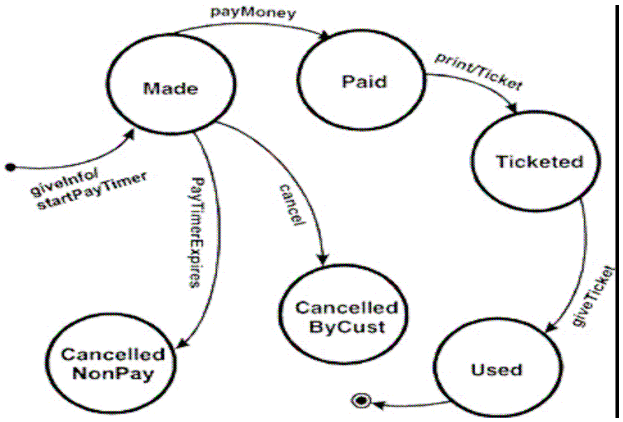


E. Cancel for no payment by the “Pay timer”

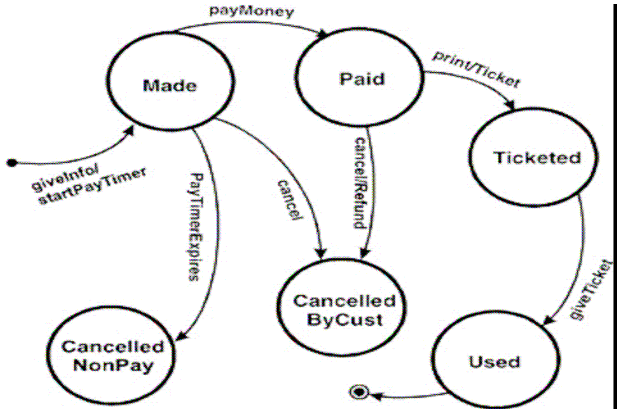
If the reservation is not paid the pay timer expires and the reservation is cancelled for nonpayment.



F. Cancel by Customer



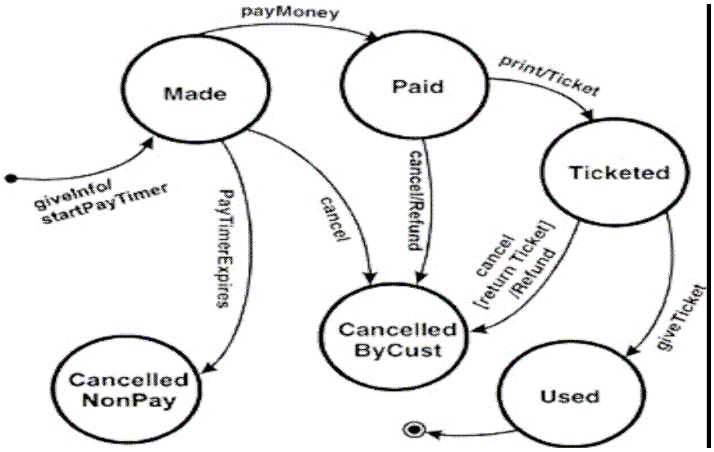
G. Cancel by customer after payment



H. Customer cancels after receiving the ticket:

In this case the airline will generate a refund but only when it receives the printed ticket from the customer. This introduces one new notational element " [] " that contain a conditional that can be evaluated either True or False.

This conditions acts as a guard allowing the transition only if the conditions is true.



ii. *State Transition Tables*

Using the complete state graph we will continue to design the state transition. Remember this table will show all the possibilities between the states, events and actions.

Like was mentioned before, the idea to create a state transition table is uncover combinations that were not identified, documented or dealt with in the requirements.

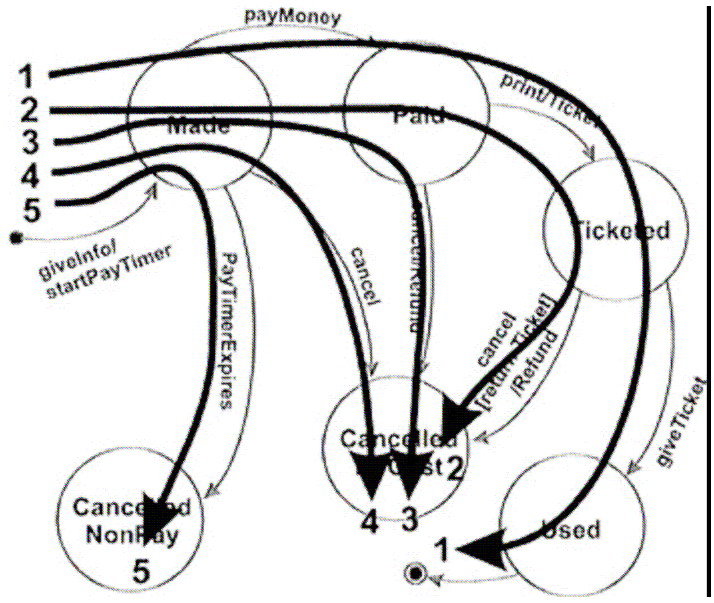
Current State	Event	Action	Next State
Null	giveInfo	startPayTime	Made
Null	payMoney	--	Null
Null	Print	--	Null
Null	giveTicket	--	Null
Null	Cancel	--	Null
Null	PayTimerExpires	--	Null
Made	giveInfo	--	Made
Made	payMoney	--	Paid
Made	Print	--	Made
Made	giveTicket	--	Made
Made	Cancel	--	Cancel by Customer
Made	PayTimerExpires	--	Can – Non Pay
Paid	giveInfo	--	Paid
Paid	payMoney	--	Paid
Paid	Print	Ticket	Ticketed
Paid	giveTicket	--	Paid
Paid	Cancel	Refund	Cancel by Customer
Paid	PayTimerExpires	--	Paid
Ticketed	giveInfo	--	Ticketed
Ticketed	payMoney	--	Ticketed
Ticketed	Print	--	Ticketed
Ticketed	giveTicket	--	Used
Ticketed	Cancel	Refund	Cancel by Customer
Ticketed	PayTimerExpires	--	Ticketed
Used	giveInfo	--	Used
Used	payMoney	--	Used
Used	Print	--	Used
Used	giveTicket	--	Used

Used	Cancel	--	Used
Used	PayTimerExpires	--	Used
Cancel-NonPay	giveInfo	--	Cancel-NonPay
Cancel-NonPay	payMoney	--	Cancel-NonPay
Cancel-NonPay	Print	--	Cancel-NonPay
Cancel-NonPay	giveTicket	--	Cancel-NonPay
Cancel-NonPay	Cancel	--	Cancel-NonPay
Cancel-NonPay	PayTimerExpires	--	Cancel-NonPay
Cancel by Customer	giveInfo	--	Cancel by Customer
Cancel by Customer	payMoney	--	Cancel by Customer
Cancel by Customer	Print	--	Cancel by Customer
Cancel by Customer	giveTicket	--	Cancel by Customer
Cancel by Customer	Cancel	--	Cancel by Customer
Cancel by Customer	PayTimerExpires	--	Cancel by Customer

iii. *Creating Test Cases*

Using the state transaction graph did it in “i” is recommended to create a set of test cases such that all transactions are exercised at least one under test.

This level of testing provides a good level of coverage without generating large number of tests.



These test cases can also be read directly from the state-transition table.

Current State	Event	Action	Next State
Null	giveInfo	startPayTime	Made
Made	payMoney	--	Paid
Made	Cancel	--	Cancel by Customer
Made	PayTimerExpires	--	Can – Non Pay
Paid	Print	Ticket	Ticketed
Paid	Cancel	Refund	Cancel by Customer
Ticketed	giveTicket	--	Used
Ticketed	Cancel	Refund	Cancel by Customer

2.4 Model #4: "Data flow graphics"

The groups of techniques used for this model is called "Data flow testing" and the most well-known is:

1. Data Cycle test / Entity Life Cycle Test

This technique is employed during System testing where validation is done for an entity which under gone several processes before it reaches the current scenario under focus. This mean the entity life cycle allows user to test all the scenarios the entity travel during its life, before it reaches the current scenario.

This kind of testing is more suitable for multi-complete application like: Banking and Insurance where the financial implications are wider in most of the stages.

This technique also is helping for testing not only the current cycle also captures regression, if any, after integration testing is done.

2.4.1 Entity Life Cycle Test

An entity in application software is the least granular object that can be indentified. For example an account or a customer Id is an entity in a Banking Application. In cases like financial application, it is very important to identify the entities involved and clearly define the path each entity is going to traverse during its life cycle. Different business scenarios could evolve at each life cycle stage for the entities.

Defining the life cycle stages, indentifying and mapping business scenarios at each life cycle stage and tracking the entity trough the entire life cycle forms the crux of Entity Life Cycle Testing.

The inter-dependencies of different business life cycle have to be clearly understood and paths have to be drawn to ensure that all the paths have been coverage. Data accumulation/redundancy, if any, in all the life stages of the Entity are to be checked so that wrong updates/no updates should not result in wrong results in other operations or reports. This is more so when an existing application is migrated into a new system or upgraded into a higher version.

For this cases the conventional methods of System Testing may not cover all the processes/path covered by an entity during it life cycle in their entirety and that is when Entity Life Cycle test can help us.

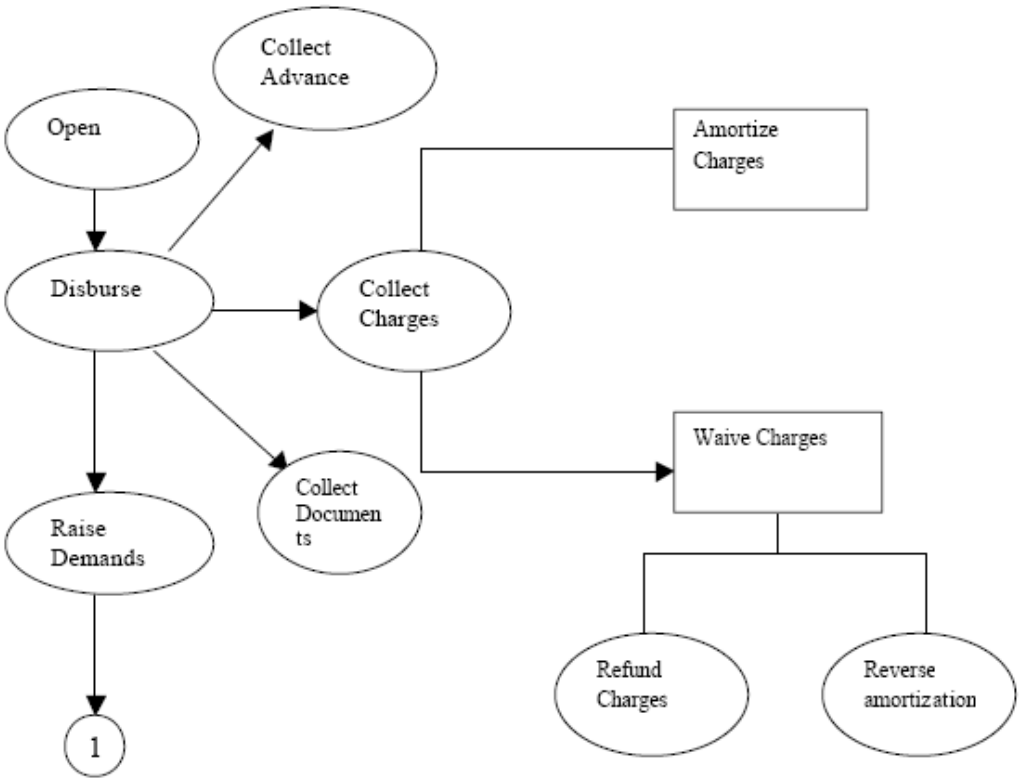
1. *Applying State transition testing*

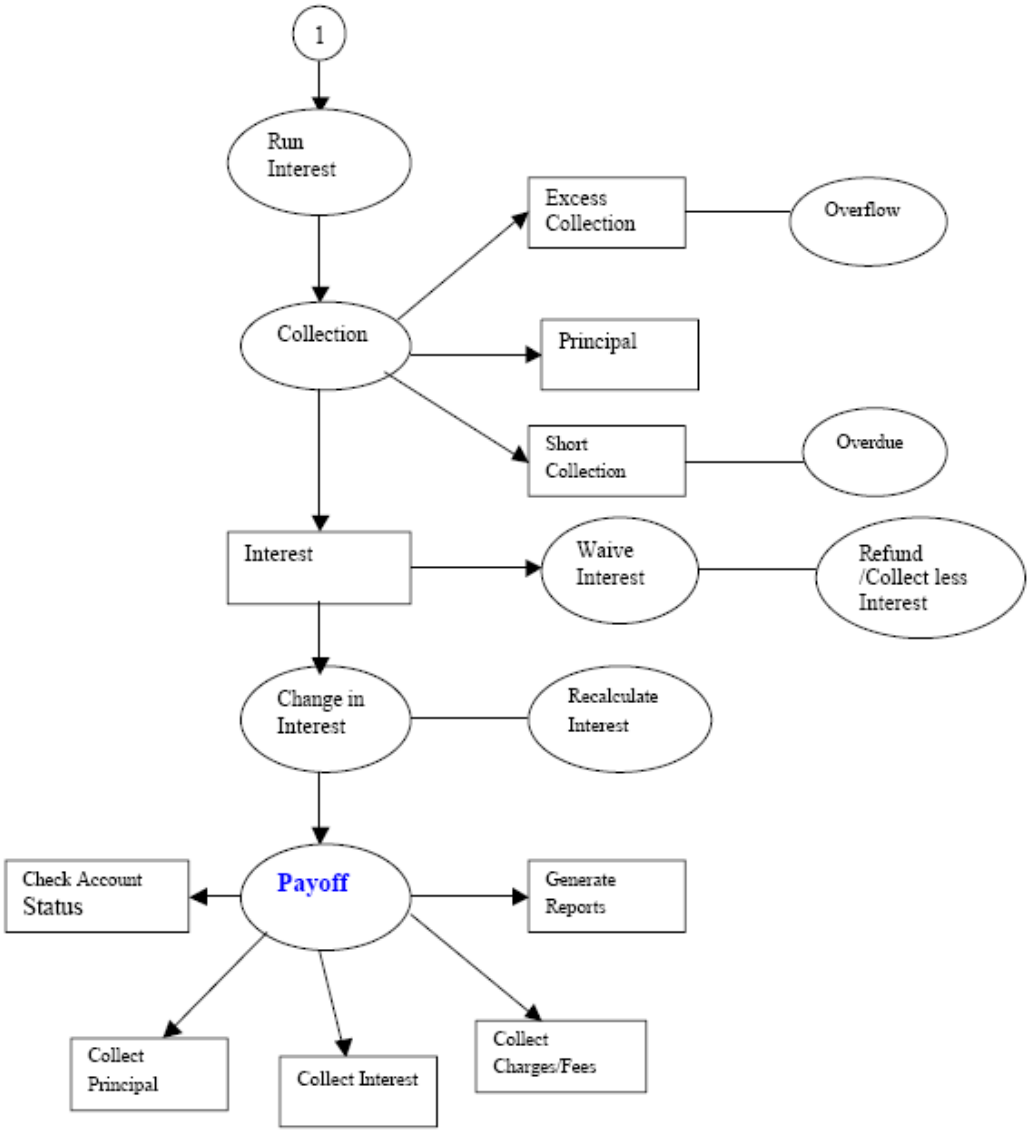
The steps involved in Entity Life Cycle Testing are:

i. Identification of Business entities
ii. Identification of Life Cycle Stages
iii. Identification of Business Scenarios at each Life Cycle Stage
iv. Mapping of Business Scenarios for cross-functional dependencies and designing test cases
v. Execution the Life Cycle testing by taking the entity through the life cycle stages by pre-defined stages

Example:

Suppose you have a bank entity which is completing the migration of its existing system to another one. The process that we need to test in order to find possible regression defects and serious functionality loss is the Payoff of a Loan which has the next entity flow:




1
Run Interest
Excess Collection
Overflow
Collection
Principal
Short Collection
Overdue
Interest
Waive Interest
Refund /Collect less Interest
Change in Interest
Recalculate Interest
Check Account Status
Payoff
Generate Reports
Collect Principal
Collect Interest
Collect Charges/Fees

With this flow we can identify the Business entities and Life Cycle Stages.

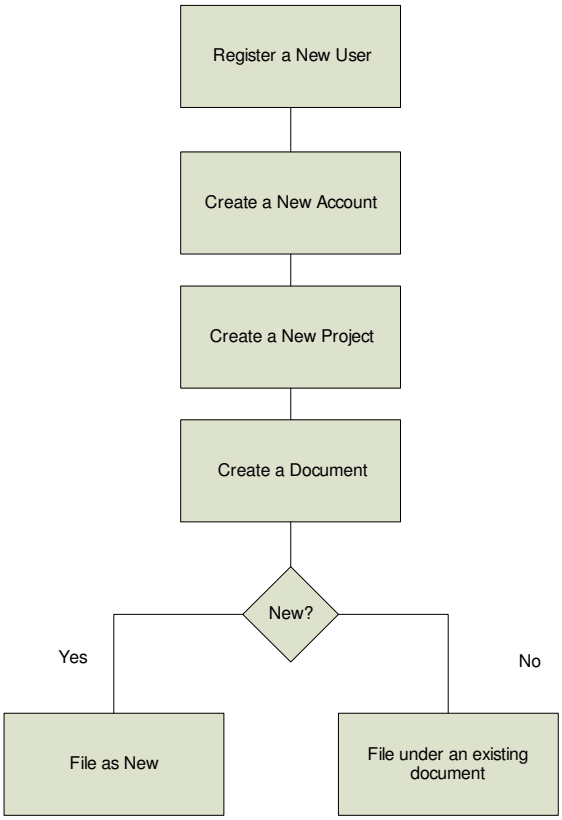
Now our test cases will be written to check the results of all the process involved during the every stage of the Entity (for example Open a new account), during one of the final Stage/Scenario of the Entity after (for example Generate Customer's Account) which link between the Customer and the Bank comes to an end, as for as this Entity is concerned.

2. *Application*

You have Document's repository with the next flow:

i. First is indispensable to have a user register in the system
ii. Second is indispensable the user belongs to an account
iii. Second is also indispensable user belongs to an specific project
iv. If user create a new document, this document will be entered in the project as a new document with non previous version
v. If user creates a new document's version, the system has to indentify which document's version is and then file the document under the same name.

Taking in consideration the previous information, we can create the next flow:



Now our possible test cases are:

# Test Case	Test Case Name
1	File a new document without login as user
2	File a new document's version without login as user
3	After login, file a document into a non-existing account
4	After login, file a document into a non-existing project
5	After login, file a New document
6	After login, file a new document's version

Notice all the test cases are related with an end-to-end flows and also validating the response on each step according to a previous scenario.

3. Other collateral advantages we derived by using this method:

- Different Business Scenarios of different customers, which were hitherto undocumented, were detected-helps while deploying a release to the customer.
- Better understanding of product steps for completing a business scenario
- Helped in continuous improvement of the products – by increasing the test coverage
- Identification of commonly used scenarios – helps in finding the impact of changes being made to a module – Any defect found will have larger impact
- Identification of different paths a business entity could take – helps in developing scripts for automation of regression testing.

2.5 Other Techniques

The next techniques are not going to be develop in detail, but we will define them and mention what they do and how can be used.

2.5.1 Function Point Analysis and Test Point Analysis

i. Function Point Analysis (FPA)

Function Point Analysis is a structured technique of classifying components of a system. It is a method used to break systems down into smaller components so that they can be better understood and analyzed.

In Function Point Analysis, systems are divided into five large classes:

Data Functions

- *Internal Logical Files* - The first data function allows users to utilize data they are responsible for maintaining. For example, a pilot may enter navigational data through a display in the cockpit prior to departure. The data is stored in a file for use and can be modified during the mission. Therefore the pilot is responsible for maintaining the file that contains the navigational information. Logical groupings of data in a system, maintained by an end user, are referred to as Internal Logical Files (ILF).

- *External Interface Files* - The second Data Function a system provides an end user is also related to logical groupings of data. In this case the user is not responsible for maintaining the data. The data resides in another system and is maintained by another user or system. The user of the system being counted requires this data for reference purposes only. For example, it may be necessary for a pilot to reference position data from a satellite or ground-based facility during flight. The pilot does not have the responsibility for updating data at these sites but must reference it during the flight. Groupings of data from another system that are used only for reference purposes are defined as External Interface Files (EIF).

  The remaining functions address the user's capability to access the data contained in ILFs and EIFs. This capability includes maintaining, inquiring and outputting of data. These are referred to as Transactional Functions.

Transactional Functions

- *External Input* - The first Transactional Function allows a user to maintain Internal Logical Files (ILFs) through the ability to add, change and delete the data. For example, a pilot can add, change and delete navigational information prior to and during the mission. In this case the pilot is utilizing a transaction referred to as an External Input (EI). An External Input gives the user the capability to maintain the data in ILF's through adding, changing and deleting its contents.

- *External Output* - The next Transactional Function gives the user the ability to produce outputs. For example a pilot has the ability to separately display ground speed, true air speed and calibrated air speed. The results displayed are derived using data that is maintained and data that is referenced. In function point terminology the resulting display is called an External Output (EO).

- *External Inquiries* - The final capability provided to users through a computerized system addresses the requirement to select and display specific data from files. To accomplish this user inputs selection information that is used to retrieve data that meets the specific criteria. In this situation there is no manipulation of the data. It is a direct retrieval of information contained on the files. For example if a pilot displays terrain clearance data that was previously set, the resulting output is the direct retrieval of stored information. These transactions are referred to as External Inquiries (EQ).

In addition to the five functional components described above there are two adjustment factors that need to be considered in Function Point Analysis.

Functional Complexity

The first adjustment factor considers the Functional Complexity for each unique function. Functional Complexity is determined based on the combination of data groupings and data elements of a particular function. The number of data elements and unique groupings are counted and compared to a complexity matrix that will rate the function as low, average or high complexity. Each of the five functional components (ILF, EIF, EI, EO and EQ) has its own unique complexity matrix. The following is the complexity matrix for External Outputs (EO)

	1-5 DETs	**6-19DETs**	**20+ DETs**
0 or 1 FTRs	L	L	A
2 or 3 FTRs	L	A	H
4 + FTRs	A	H	H

Complexity	**UFP**
L (Low)	4
A (Average)	5
H (High)	7

Using these tables in the pilot's example, the function point count for these functions may be:

Function Name	Function Type	Record Element Type	Data Element Type(DET)	File Type Referenced (FTR)	Unadjusted FPs (UFP)
Ground Speed display	EO	n/a	20	3	7 (High)
Air speed displayed	EO	n/a	20	3	7 (High)
Calibrate air	EO	n/a	10	2	5 (Average)
Total Unadjusted count					50 UFPs

All the functional components are analyzed in this way and added together to derive an Unadjusted Function Point count

Value Adjustment Factor

The unadjusted function point count is now multiplied by the second adjustment factor called the "Value Adjustment Factor". This factor considers the system's technical and operational characteristics and is calculated by reviewing 14 factors:

1. Data Communications: The data and control information used in the application are sent or received over communication facilities.

2. Distributed Data Processing: Distributed data or processing functions are a characteristic of the application within the application boundary.

3. Performance: Application performance objectives, stated or approved by the user, in either response or throughput, influence (or will influence) the design, development, installation and support of the application.

4. Heavily Used Configuration: A heavily used operational configuration, requiring special design considerations, is a characteristic of the application.

5. Transaction Rate: The transaction rate is high and influences the design, development, installation and support.

6. On-line Data Entry: On-line data entry and control information functions are provided in the application.

7. End -User Efficiency: The on-line functions provided emphasize a design for end-user efficiency.

8. On-line Update: The application provides on-line update for the internal logical files.

9. Complex Processing: Complex processing is a characteristic of the application.

10. Reusability: The application and the code in the application have been specifically designed, developed and supported to be usable in other applications.

11. Installation Ease: Conversion and installation ease are characteristics of the application. A conversion and installation plan and/or conversion tools were provided and tested during the system test phase.

12. Operational Ease: Operational ease is a characteristic of the application. Effective start-up, backup and recovery procedures were provided and tested during the system test phase.

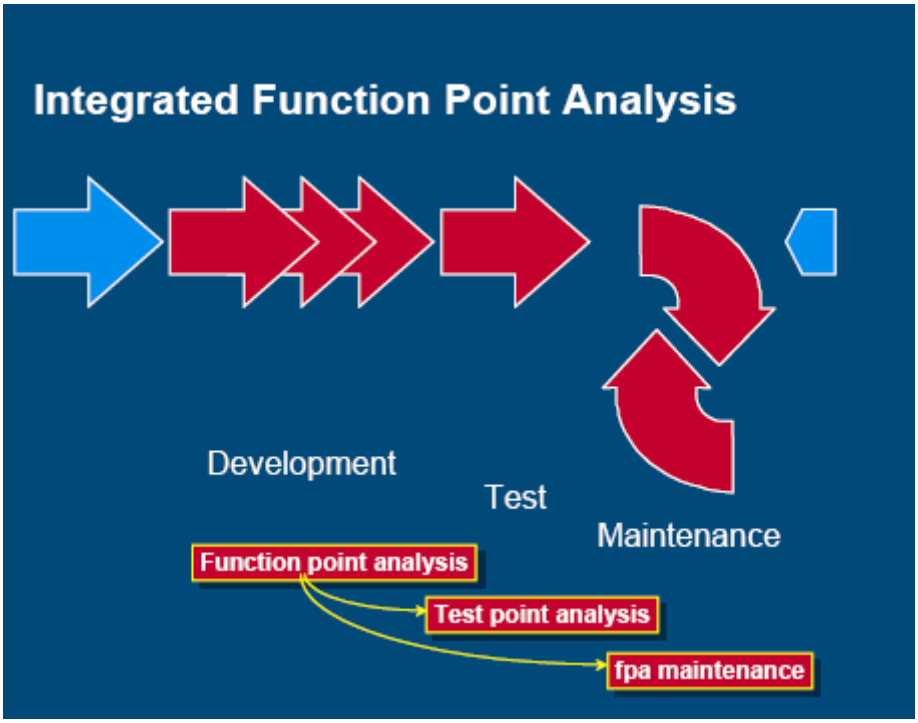
13. Multiple Sites: The application has been specifically designed, developed and supported to be installed at multiple sites for multiple organizations.

14. Facilitate Change: The application has been specifically designed, developed and supported to facilitate change.

Each of these factors is scored based on their influence on the system being counted. The resulting score will increase or decrease the Unadjusted Function Point count by 35%. This calculation provides us with the Adjusted Function Point count

Finally after the functional point count has been developed this value can help us to estimate the number of test cases our application could have, that methodology is called “Test Point Analysis”

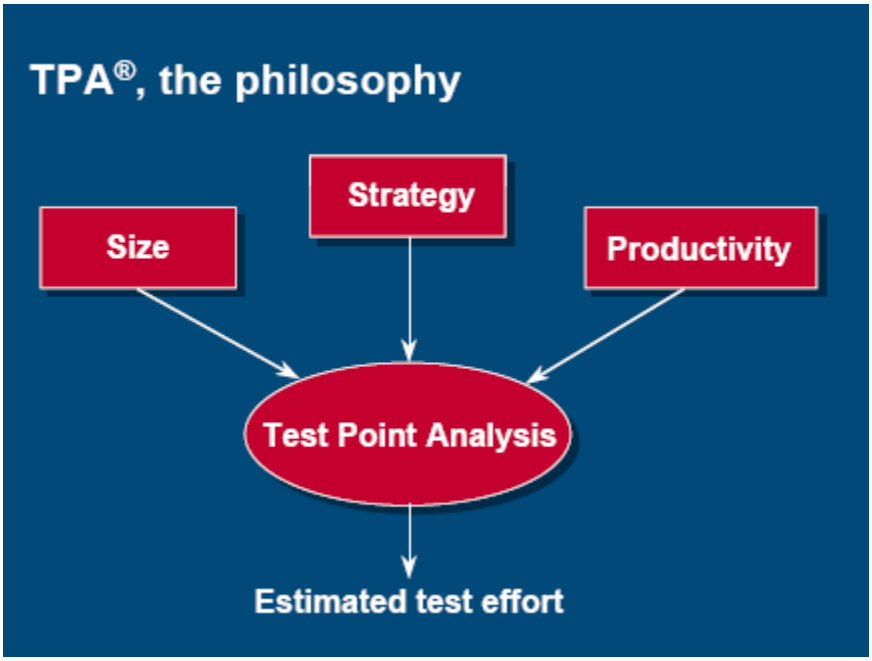
ii. Test Point Analysis (TPA)



Like you can see in this image while Function point analysis is used in Development and white-box testing (Unit testing for example) the test point analysis covers the black box testing, allowing user to estimate the test activities and focus on system testing or acceptance testing.

With TPA is also possible to determine or calculate the relative importance of various functions, with a view to using the available testing time as efficiently as possible.

Test Point Analysis use three relevant elements like you can see in the next image:



Size:

The first element to be considered is the size of the information system. For TPA purposes, the size of an information system is determined mainly by the number of function points assigned to it. However, a number of additions or amendments need to be made, because certain factors, which have little or no influence on the number of function points, are pertinent to testing.

The factors in question are the following:

- *Complexity;* complexity relates to the number of conditions in a function. More conditions almost always mean more test cases and therefore a greater volume of testing work.

- *Interfacing;* the degree of interfacing of a function is determined by the number of data sets maintained by a function and the number of other functions, which make use of those data sets. Interfacing is relevant because these “other” functions will require testing if the maintenance function is modified.

- *Uniformity;* it is important to consider the extent to which the structure of a function allows it to be tested using existing or slightly modified specifications, i.e. the extent to which the information system contains similarly structured functions.

Test Strategy:

During development or maintenance, quality requirements will have been specified for the information system. The test activities must determine the extent to which these requirements have been satisfied. In liaison with the client, the system and/or subsystem quality characteristics to be tested are identified and their relative importance determined. The importance of each characteristic influences the thoroughness of the related test activities. The more important a quality characteristic, the more exacting and thorough the tests have to be and the greater the volume of work. The importance of the various characteristics should be determined in consultation with the client when the test strategy is being formulated; the

Information can then be used as TPA input. In the course of the TPA process, the volume of testing work is calculated on the basis of the test strategy.

While certain general quality requirements apply to the information system as a whole, there are also differences between the various functions in terms of the requirements to be met. From the user's point of view, a function which is utilized throughout the day may be much more important than a processing function which operates only at night. For each function, therefore, there are two (subjective) factors that influence the thoroughness of testing namely the user-importance of the function and the usage-intensity. The thoroughness of testing reflects the degree of certainty or insight into system quality sought by the customer. The user- importance and usage-intensity factors are, of course, based on the test strategy.

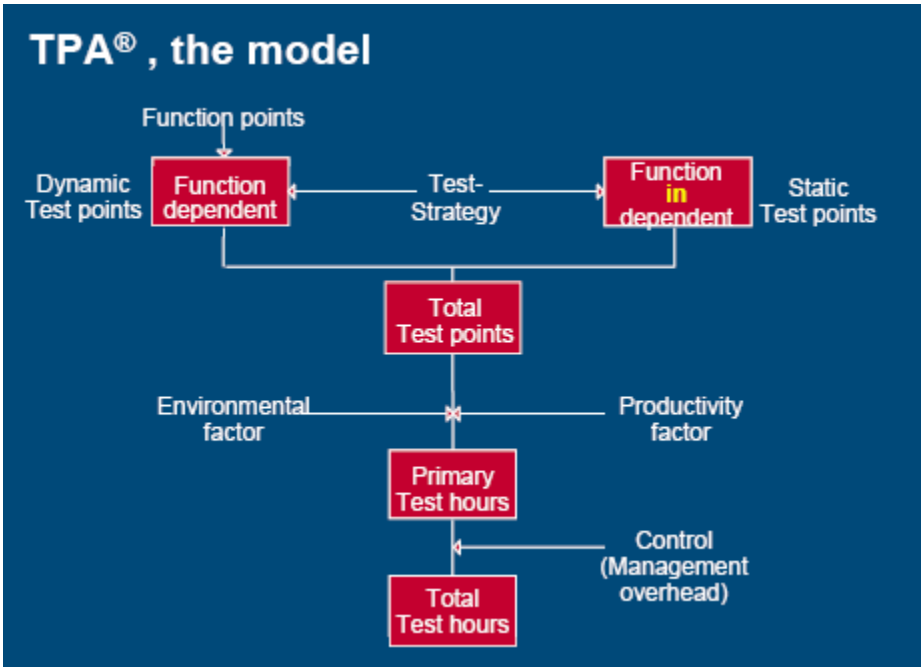
As previously indicated, the importance attached to the various quality characteristics for testing purposes and the importance of the various subsystems and/or functions determine the test strategy. The test strategy specifies which quality characteristics are to be tested for each subsystem or function, and the relevant degree of coverage. TPA and strategy determination are closely related and in practice are often performed at the same time.

Productivity:

Productivity is not a new concept to anyone who has produced estimates on the basis of function points. In function point analysis, productivity is an expression of the relationship between the number of hours necessary for a task and the measured number of function points. In TPA, productivity relates to the time necessary to realize one test point, as determined by the size of the information system and the test strategy.

Productivity has two components: a productivity figure and an environmental factor. The productivity figure is based on the knowledge and skill of the test team, and is therefore specific to the individual organization. The environmental factor indicates the degree to which the environment influences the test activities to which the productivity is related. Influential environmental considerations include the availability of test tools, the amount of experience the team has with the test environment, the quality of the test basis and the availability of testware.

Finally the TPA procedure model is illustrated below:



Dynamic Test Point:

The number of dynamic test points is the sum of the test points assigned to the individual functions. The calculation is made by the next formula:

TPf = FPf * Df * Qd

Where:

FPf = Number of Function Points
Df = Function Dependent
Qd = Quality requirements relating to the dynamic quality characteristic to be tested.

On the other hand the calculation of Df (Function Dependent) is calculated by the next formula:

Df = ((Ue+Uy+I+C) /16)* U

Where:

Ue= User-importance
Uy = Usage-intesity
I = Interfacing
C = Complexity
U = Uniformity.

Test Points:

The indirect test point count depends mainly on the function point count for the system as a whole. The indirect test point count is also influenced by the requirements regarding the static quality characteristic to be tested (Qi factor)

The total number of test points is calculated by the next formula:

$$TP = \sum TPf + (FP * Qi) / 500$$

Where:

∑ TPf = Sum of the test points assigned to the individual functions (dynamic test points)
FP= total number of function points assigned to the system as a whole
Qi = weighting factor for the indirectly measurable quality characteristics

Primary Test Hours (PT)

The primary test hour count is the number of hours required for carrying out the test activities involved in the test life cycle phases preparation, specification, execution and completion.

The primary test hour is calculated by the next formula:

$$PT = TP * P * E$$

Where

TP = Total number of test poing assigned to the system as a whole
P = Productivity (which is consider in practice between 0.7 and 2.0 hour per test point)
E = Environment factor (= sum rating environmental variables/21)

Total Test Hours (PT)

Since every test process involves tasks, which may be placed under the heading "planning and control" (PC), allowance need to be made for such activities. The number of primary test hour and the planning and control allowance together give the total number of test hours.

The standard (nominal) allowance is 10 per cent. However, the allowance may be increased or decreased, in line with the following two factors:

- Team size
- Management tools

In general the PC (Planning and Control) is calculated by the next formula:

PC = (100+ sum rating variables) /100

Leaving the Total test hour calculation like:

Total test hours = PT * PC

Where
PT = Primary Test Hour
PC = Planning Control hours.

2.5.2 Error Guessing

Design of test cases using error guessing method is based on the testers' past experience and intuition.

It is impossible to give a procedure for an error guessing approach since it is more intuitive and ad hoc process but the basic idea behind is to enumerate a list of possible errors and then writes test cases based on this list.

2.5.3 Error Testing / Exploratory Testing

Exploratory testing is rather difficult to line up with the other test design techniques.

It is not based on any of the described basic techniques, it leaves a free choice of basic techniques to be applied and it provides no guaranteed coverage.

Exploratory testing is often associated with testing in the absence of a formal test basis, such as a functional design. However, this is not necessarily true. It is entirely possible to apply it with a well-described system.

Having said that, the technique lends itself very well to situations in which no described test basis is present. Exploratory testing puts less emphasis on a described test basis and more on other ways of assessing the adequacy of the test object, such as by familiarisation with the system in the course of the test execution

2.5.4 Data Cycle Test

The primary aim of the data cycle test is not to trace functional defects in individual functions, but to find integration defects.

The test focuses on the link between various functions and the way in which they deal with communal data. The Data Cycle Test is most effective if the functionality of the individual functions has already been sufficiently tested.

The most important test basis is the CRUD matrix (a table showing the Functions in an application containing SQL statement affecting parts of a database) and a description of the applicable integrity rules. If this matrix is not present, it must be created to enable the drafting of the test design within the framework of the data cycle test.

2.5.5 Permutation and Random Test

Each software system has an input domain from which input data is selected for testing. If inputs are randomly selected is called permutation testing (also called random testing). The advantage of the method is that it can save time and effort that more detailed and thoughtful test input selections methods require.

Permutation testing may be performed either manually or using automated test tools. This technique is most cost-effective when fully automated as then very many tests can be run without manual intervention. However, to achieve full automaton it must be possible to:

- Automatically generate random test inputs
- Automatically generate expected results from the specifications
- Automatically check test outputs against the specifications

Permutation testing can be considered to solve mathematic problems like transform Cartesian coordinates to their polar equivalent.

2.5.6 Process cycle test

The process cycle test is a technique that is applied in particular to the testing of the quality characteristic of suitability (integration between the administrative organisation and the automated information system).

The test basis should contain structured information on the required system behaviour in the form of paths and decision points. The test basis for the process cycle test consists mainly of descriptions of procedures and related forms. Preferably, flow charts should be added to the procedure descriptions.

2.5.7 Real Life Test

With the real life test, it is not the intention to test the system behaviour in separate situations, but to simulate the realistic usage of the system in a statistically responsible way. This test mainly focuses on characteristics, such as effectively, connectivity, continuity and performance of the system under test. To be able to test whether a system can handle realistic usage of it, that usage should be somehow specified. This also serves as a test basis and, in this context, is often referred to as the profile. The two most common types are: operational profile and load profile.

2.5.8 Semantic Test

The test basis consists of the semantic rules that specify what a datum should comply with in order to be accepted by the system as valid input.

Semantic rules are connected with the relationships between data. These relationships may be between the data within a screen, between data on various screens and between input data and existing data in the database.

Semantic rules may be established in various documents, but are usually described in: functional specifications of the relevant function or input screen and in the business rules that apply to the functions overall.

The following checklist can be used to check the specifications and create the test cases:

- Have any standards for error handling been described at (sub) system level?
- Have the input checks (in particular the relation checks) - including any related error message - been described as part of the function description and can these be implemented?
- Have any specific requirements been set for access security of functions and/or data?
- Have any user profiles been described with regard to security?
- Has it been described which requirements are set for identification (user ID) and authentication (password)?

2.5.9 Syntactic Test

The test basis for the syntactic test consists of the syntactic rules, which specify how an attribute should comply in order to be accepted as valid input/output by the system.

These rules actually describe the value domain for the relevant attribute. If a value outside this domain is offered for the attribute, the system should discontinue the processing in a controlled manner – usually with an error message.

Syntactic rules may be established in various documents, but they are normally described in: the 'data dictionary' and other data models, in which the characteristics of all the data are described and in the functional specifications of the relevant function or input screen, containing the specific requirements in respect of the attributes.

The following checklist can be used to check the specifications and create the test cases:

- Have any applicable standards been described at system level?
- Have any applicable standards been described at subsystem level?
- Have the layouts of the screens, menus and dialogs been described? In this context, has any attention been given to the following aspects:
    - Field length of the items;
    - Location of items on the screen;
    - Distinction between input and output items;
    - Primary input checks (not resulting from domain definition);
    - Error handling;
    - Mandatory and non-mandatory items;
    - Possible function keys, help screens and selections?
- Have the "screen items" and/or attributes been included in the data model?
- Have the types (numeric, alphanumeric, date) and the domains of the input and output data been described?
- Are the specified mandatory and non-mandatory items consistent with the options from the data model?
- Do the described screen layouts comply with the standards?
- Have the layouts of the reports been described? In this context, has any attention been given to the following aspects:
    - Field length of the items;
    - Location of items in the report?
- Have the "report items" and/or attributes been included in the data model?
- Do the described report layouts comply with the standards?

3 References

Equivalent Partition:

http://zweb.iti.upv.es/groups/squac/events/JTS2007/slides/3demayo16.00-TanjaVos.pdf/attach/3demayo16.00-TanjaVos.pdf

Boundary Value Technique:

http://www.cs.swan.ac.uk/~csmarkus/CS339/dissertations/NeateB.pdf

Decision Table Technique:

http://www.cs.arizona.edu/classes/cs436/spring06/Lectures/DecTable.PDF

http://www.cs.swan.ac.uk/~csmarkus/CS339/dissertations/FerridayC.pdf

Cause Effect Technique:

http://fsktm.um.edu.my/~ainon/wkes3107/Black%20Box%20Testing%20Part%20II.pdf

http://softestserv.ca/RBT_Cause-Effect_Graphing2.pdf

State Transition Technique:

http://www.oxon.bcs.org/downloads/state_transition_testing_handout.pdf

Entity Life Cycle Testing:

http://www.tarrani.net/EntityLifeCycleTesting.pdf

Function Point Analysis:

http://www.softwaremetrics.com/fpafund.html

http://www.devshed.com/c/a/Practices/An-Overview-of-Function-Point-Analysis/

Test Point Analysis:

http://www.isqs.nl/documenten/ESCOM%201999%20-%20TPA%20(presentation).pdf

http://www.scribd.com/doc/6958131/ESCOM1999-Test-Point-Analysis-A-method-for-test-estimationtcm15835882