

Chapter 3

Functional (Black Box) Testing

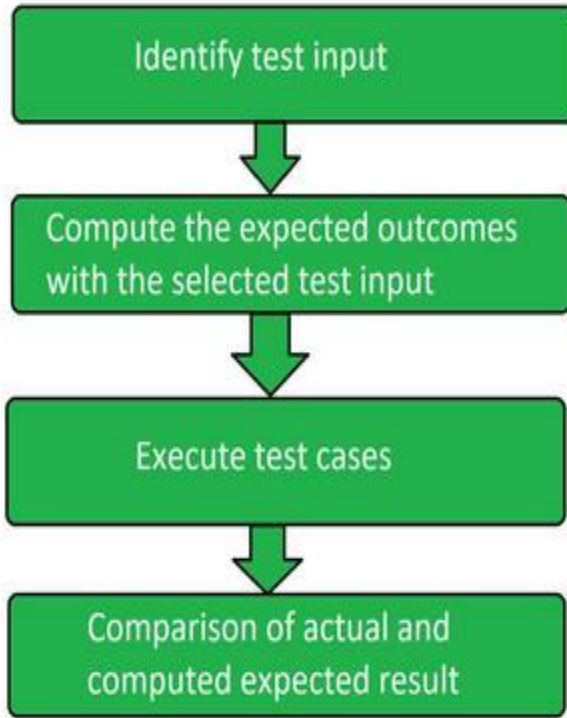
- At the end of this chapter you will be able to cover the following concepts.
 - Define functional testing
 - Discuss about types of functional testing Techniques.

What is Functional Testing?

- Functional testing is defined as a type of testing that verifies **that each function of the software application** works in conformance with the **requirement** and **specification**.
- This testing is not concerned with the **source code of the application**.
- Each functionality of the software application is tested by providing **appropriate test input**, **expecting the output**, and **comparing the actual output** with the expected output.
- This testing focuses on checking
 - the user interface,
 - APIs ,
 - database ,
 - security , client or server application , and
 - functionality of the Application Under Test.

Functional Testing Process

- Functional testing involves the following steps:



Step 1. Identify test input: This step involves identifying the functionality that needs to be tested.

- This can vary from testing the **usability functions**, and **main functions** to error conditions.

Step 2. Compute expected outcomes:

Create **input data** based on the **specifications of the function** and determine the output based on these specifications.

- Step 3. Execute test cases:** This step involves executing the designed test cases and recording the output.
- Step 4. Compare the actual and expected output:** the **actual output obtained after executing** the test cases is compared with **the expected output**

Functional Testing Techniques

- The functional test case design techniques covered here are:
 - Equivalence partitioning and boundary value analysis
 - (Domain analysis—not part of the ISTQB syllabus)
 - Decision tables
 - Cause-effect graph
 - State transition testing
 - Classification tree method
 - Pairwise testing
 - Use case testing
 - (Syntax testing—not part of the ISTQB syllabus)

Equivalence Partitioning and Boundary Value Analysis

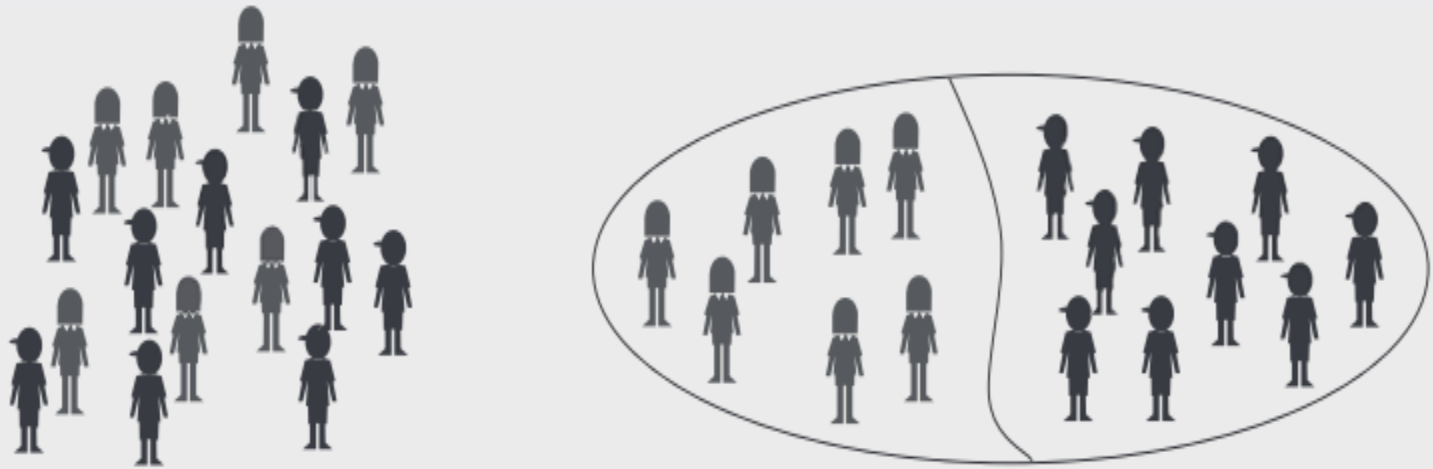
- Designing test cases is about finding the input to cover something we want to test.
 - few to a huge amount of possibilities . Example:-
 - ✓ A product may have only one button and it can be either on or off = 2 possibilities.
 - ✓ A field must be filled in with the name of a valid postal district = thousands of possibilities.
- The equivalence partitioning test technique can help us handle situations with many input possibilities.

Equivalence Partitioning

- partition the **input** or **output domain** into **equivalence classes**.
 - A class is a portion of the domain.
 - The domain is said to be partitioned into classes if all members of the domain belong to exactly one class.
 - ✓ no member belongs to more than one class and
 - ✓ no member falls outside the classes.
- The term equivalence refers to the assumption that **all the members in a class behave in the same way**.
- the equivalence partitioning is that all members in an equivalence class will either **fail or pass the same test**.
 - One member represents all!

Equivalence Partitioning Cont.

- If we take a class of pupils and the requirement says that all the girls should have an e-mail every Thursdays reminding them to bring their swimsuits, we can partition the class into a partition of girls and a partition of boys and use one representative from each class in our test cases.



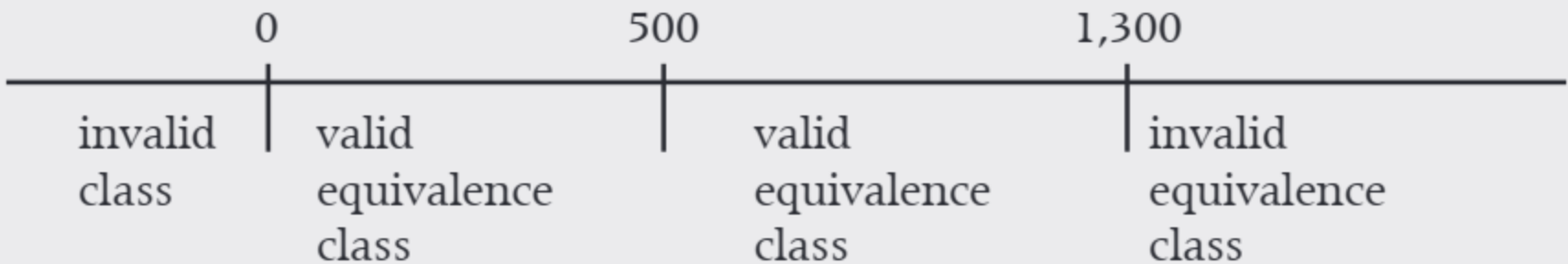
- When we partition a domain into equivalence classes, we will usually get both valid and invalid classes.
- The most common types of equivalence class partitions are intervals and sets of possibilities (unordered list or ordered lists).

Equivalence Partitioning Cont.

- Intervals may be illustrated by a requirement stating:

Income in €	Tax percentage
Up to and including 500	0
More than 500, but less than 1,300	30
1,300 or more, but less than 5,000	40

If this is all we know, we have:



Another invalid equivalence class may be inputs containing letters.

- To illustrate a set of possibilities we may use the **unordered list** of hair colors: (blond, brown, black, red, gray).

Equivalence Partitioning Cont.

- To exercise an equivalence class we need to pick one value in the equivalence class and make a test case.
 - Test cases for the tax percentage could be based on the input values: -5; 234; 810; and 2,207.
 - For the hair colors we could choose: **black** and **green**, as a valid and on invalid input value, respectively.

Boundary Value Analysis

- A boundary value is the value on a boundary of an equivalence class.
- The boundary values require extra attention because defects are often found on or immediately around these.
 - Choosing test cases based on boundary value analysis insures that the test cases are effective.
- For interval classes with precise boundaries, it is not difficult to identify the boundary values. Example:
 - The interval given as: $0 \leq \text{income} \leq 500$, is one equivalence class with the two boundaries 0 and 500.
- If a class has an imprecise boundary ($>$ or $<$) the boundary value is one increment inside the imprecise boundary. Example:

Boundary Value Analysis Cont.

- If the above interval had been specified as: $0 \leq \text{income} < 500$, and the smallest increment is given as 1, we would have an equivalence class with the two boundaries 0 and 499.
- The smallest increment should always be specified; otherwise we must ask or guess based on common or available information.
- For each boundary we test two values.
 - boundary value
 - one value inside the boundary in the equivalence class
- In traditional testing it was also recommended to choose
 - boundary value
 - one value inside the boundary in the equivalence class
 - One value out side the boundary in the equivalence class

Equivalence Partitioning and Boundary Value Analysis Test Design Template

Test design item number:		Traces:	
Based on: Input/Output		Assumptions:	
Type	Description	Tag	BT

- The fields in the table are:
 - Test design item number: Unique identifier of the test design item
 - Traces: References to the requirement(s) or other descriptions covered by this test design
 - Based on: Input/Output: Indication of which type of domain the design is based on
 - Assumptions: Here any assumption must be documented.

Equivalence Partitioning and Boundary Value Analysis Test Design Template

- For each test condition we have the following fields:
 - **Type:** Must be one of
 - ✓ VC—Valid class
 - ✓ IC—Invalid class
 - ✓ VB—Valid boundary value
 - ✓ IB—Invalid boundary value
 - **Description:** The specification of the test condition
 - **Tag:** Unique identification of the test condition
 - **BT = Belongs to:** Indicates the class a boundary value belongs to. This can be used to cross-check the boundary values.

Equivalence Partitioning and Boundary Value Analysis Test Design Template Example

In this example we shall find test conditions and test cases for the testing of this user requirement.

[UR 631] The system shall allow shipments for which the price is less than or equal to € 100.

The first thing we'll do is fill in the header of the design table.

Test design item number: 11	Traces: [UR 631]
Based on: Input	Assumptions: The price cannot be negative The smallest increment is 1 cent

The next thing is identifying the valid class(es).

Type	Description	Tag	BT
VC	0 <= shipment price <= 100		

We then consider if there are any invalid classes. If we only have the single requirement given above, we can identify two obvious and two special invalid equivalence classes. The new rows are indicated in bold.

Type	Description	Tag	BT
IC	shipment price < 0		
VC	0 <= shipment price <= 100		
IC	shipment price > 100		
IC	shipment price is empty		
IC	shipment price contains characters		

Our boundary value analysis gives us two boundary values.

Equivalence Partitioning and Boundary Value Analysis Test Design Template Example Cont.

Type	Description	Tag	BT
IC	shipment price < 0		
IB	shipment price = -0.01		
VB	shipment price = 0.00		
VC	0 <= shipment price <= 100		
VB	shipment price = 100.00		
IC	shipment price > 100		
IB	shipment price = 100.01		
IC	shipment price is empty		
IC	shipment price contains characters		

This concludes the equivalence class partitioning and boundary value analysis for the first requirement. We will complete the table by adding tags and indicating to which classes the boundary values belong.

Test design item number: 11		Traces: [UR 631]	
Based on: Input		Assumptions: The price cannot be negative The smallest increment is 1 cent	
Type	Description	Tag	BT
IC	shipment price < 0	11-1	
IB	shipment price = -0.01	11-2	11-1
VB	shipment price = 0.00	11-3	11-4
VC	0 <= shipment price <= 100	11-4	
VB	shipment price = 100.00	11-5	11-4
IC	shipment price > 100	11-6	
IB	shipment price = 100.01	11-7	11-6
IC	shipment price is empty	11-8	
IC	shipment price contains characters	11-9	

We can now make low-level test cases. If we want 100% equivalence partition coverage and two value boundary value coverage for the requirement, assuming that invalid values are rejected, we get the following test cases:

Equivalence Partitioning and Boundary Value Analysis Test Design Template Example Cont.

Tag	Test case	Input Price=	Expected output
11-1	TC1-1	-25.00	rejection
11-2	TC1-2	0.02	rejection
11-2	TC1-3	-0.01	rejection
11-3	TC1-4	0.00	OK
11-3	TC1-5	0.01	OK
11-4	TC1-6	47.00	OK
11-5	TC1-7	99.99	OK
11-5	TC1-8	100.00	OK
11-7	TC1-9	100.01	rejection
11-7	TC1-10	100.02	rejection
11-6	TC1-11	114.00	rejection
11-8	TC1-12	" "	rejection
11-9	TC1-13	"abcd.nn"	rejection

We could choose to omit some of the test cases, especially since we actually get five different test cases covering the same equivalence class.

Equivalence Partitioning and Boundary Value Analysis Test Design Template Example Cont.

In the next example we will test the following requirement:

[UR 627] The system shall allow the packing type to be specified as either “Box” or “Wrapping paper.”

The test design table looks like this after the analysis.

Test design item number: 15		Traces: [UR 627]	
Based on: Input		Assumptions:	
Type	Description	Tag	BT
VC	“Box” “Wrapping paper”	15-1	
IC	All other texts	15-2	

This type of equivalence class does not have boundaries.

The test cases could be:

Tag	Test case	Input packing type	Expected output
15-1	TC3-1	“Box”	OK
15-2	TC3-2	“Paper”	rejection

It is only necessary to test one of the valid packing types, because it is a member of an equivalence class.

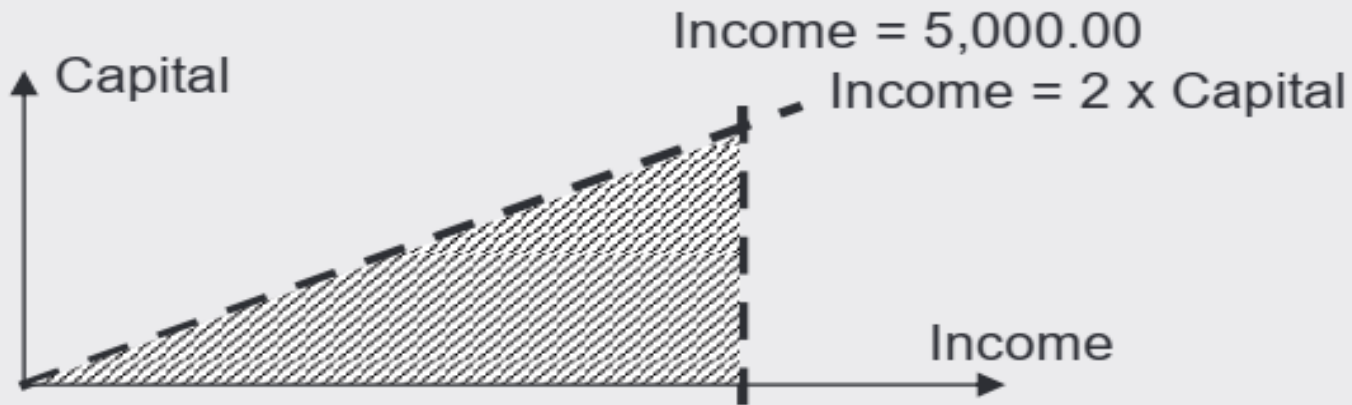
Domain Analysis

- In equivalence partitioning of intervals where the **boundaries** are given by **one-dimensional partitions**.
- The domain analysis test case design technique is used when our **input partitions** are **multidimensional**.
- The **principles** are the same as in equivalence partitioning and **boundary value analysis**.
- Example:-

For equivalence partitioning we had the example of intervals of income groups, where $0.00 \leq \text{income} < 5,000.00$ is tax-free. This is a one-dimensional domain. If people's capital counts in the calculation as well, so that income is only tax-free if it is also less than twice the capital held by the person in question, we have a two-dimensional domain.

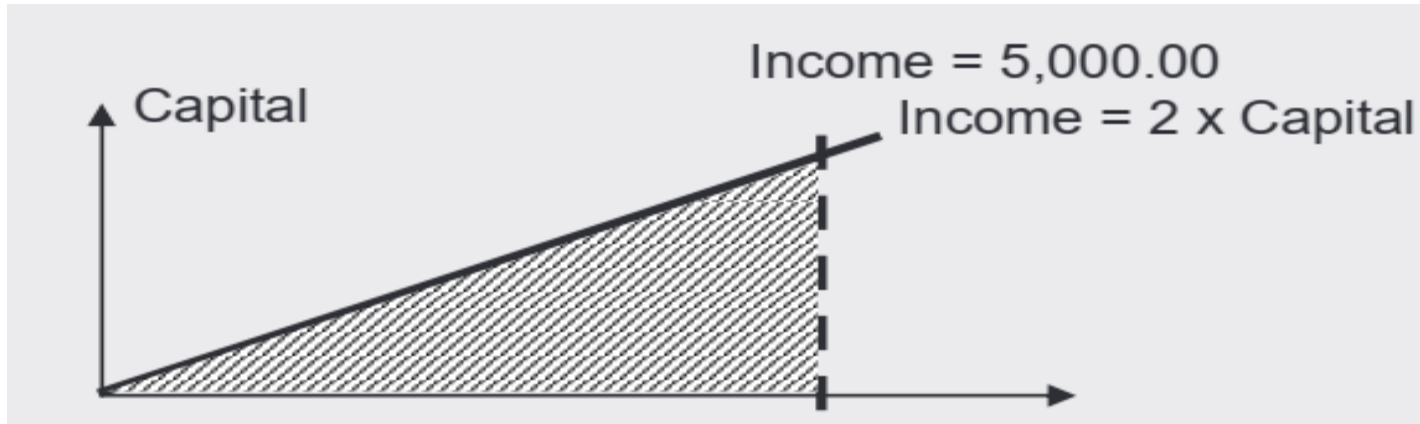
Domain Analysis Cont.

The two-dimensional domain for tax-free income is shown as the striped area (assuming that the capital and the income are both ≥ 0.00):



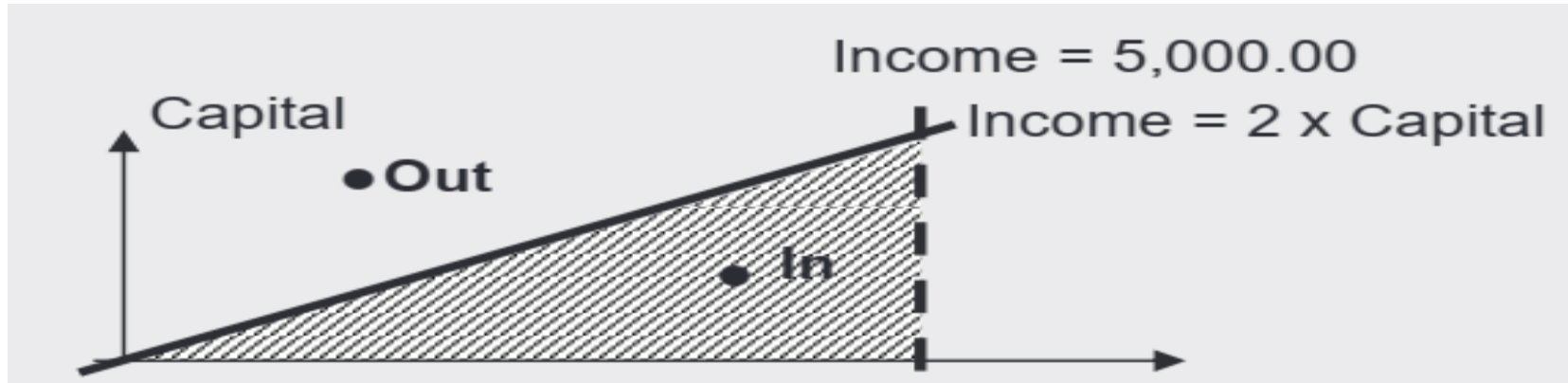
- Borders may be either **open** or **closed**
 - A border is **open** if a value on the **border does not belong to the domain**. Example: $\text{income} < 5,000.00$ and $\text{income} < 2 \times \text{capital}$
 - A border is **closed** if a value on the **border belongs to the domain**. Example: $\text{income} < 5,000.00$ and $\text{income} \leq 2 \times \text{capital}$

Domain Analysis Cont.



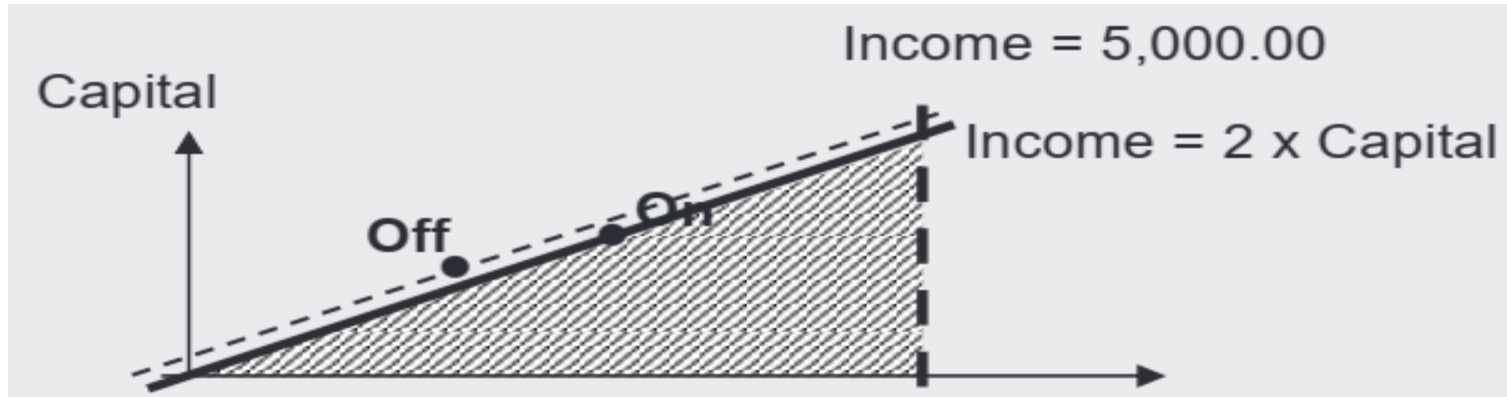
- If we change the border for tax-free income to become: $\text{income} \leq 2 \times \text{capital}$, we have a closed border.
- **In equivalence partitioning**
 - A point is an **In point** in the domain we are considering, **if it is inside** and **not on the border**
 - A point is **an Out point** to the domain we are considering, if it is **outside** and **not on the border** (it is then in another domain)

Domain Analysis Cont.



- An **In point** and an **Out point** relative to the border income $\leq 2 \times \text{capital}$.
- In the **boundary value analysis** related to equivalence partitioning
 - In **domain analysis** we operate with **On** and **Off** points relative to each border.
 - ✓ A point is an **On point** if it is **on the border** between partitions
 - ✓ A point is an **Off point** if it is “slightly” off the border

Domain Analysis Cont.



❖ Domain Analysis Strategy

- The **number of test cases** we can design based on a domain analysis depends on the test strategy.
 - $N\text{-On} * N\text{-Off}$
 - ✓ $N\text{-On}$ is the number of **On points** we want to test for **each border** and
 - ✓ $N\text{-Off}$ correspondingly is the number of **Off points** we want to test for each border for the domains

Domain Analysis Cont.

- If we choose a $1 * 1$ strategy we set out to test one On point and one Off point for each border of the domains
- If our strategy is $2 * 1$, we set out to test two On points and one Off point for each of the borders for all the domains.
- In a $1 * 1$ strategy we will get two test cases for each border.
- If we are testing adjacent domains, we will get equivalent test cases because an Off point in one domain is an In point in the adjacent domain.

Domain Analysis Cont.

❖ Domain Analysis Coverage

- The coverage elements for the identified domains are **the In points** and the **Out points**.
- The coverage is measured as the **percentage of In points** and **Out points** that have been exercised by a test.
- **Do not count an In point** in one partition being an **Out point** in another partition to be tested **twice**.
- The coverage elements for the borders are the **On points** and **Off points**.
- The coverage is measured as the **percentage of On points** and **Off points** that have been exercised by a test **relative to what the strategy determines as the number of points to test**.
 - Again do not count **duplicate points twice**.

Domain Analysis Cont.

❖ Domain Analysis Test Design Template

- The design of the test conditions based on domain analysis and with the aim of getting On and Off point coverage can be captured in a table like this one.

Tag						
Border 1 condition	ON	OFF				
Border 2 condition			ON	OFF		
Border n condition					ON	OFF

- The table is for one domain; and it **must be expanded** both **in the length** and **width to accommodate** all the borders our domain may have.
- The rule is: **divide** and **conquer**.
- For **each of the borders** involved we should:
 - Test an On point
 - Test an Off point

Domain Analysis Cont.

- When we start to make **low-level test cases** we add a row for each variable to select values for.
- In a **two-dimensional domain** we will have to **select values for two variables**.

Tag						
Border 1 condition	ON	OFF				
Border 2 condition			ON	OFF		
Border n condition					ON	OFF
Variable X						
Variable Y						

- For each column we **select a value that satisfies** what we want.
- In the **first column of values** we must select **a value for X** and **a value for Y** that gives us a point On border 1.

Domain Analysis Cont.

- Domain Analysis Test Design Example

In this example we shall test this user requirement:

[UR 637] The system shall allow posting of envelopes where the longest side (l) is longer than or equal to 12 centimeters, but not longer than 75 centimeters. The smallest side (w) must be longer than or equal to 1 centimeter. The length must be twice the width and must be greater than or equal to 10 centimeters. Measures are always rounded up to the nearest centimeter. All odd envelopes are to be handled by courier.

We can rewrite this requirement to read:

$\text{length} \geq 12$

$\text{length} < 75$

$\text{width} \geq 1$

$\text{length} - 2 \times \text{width} \geq 10$

This can be entered in our template:

Domain Analysis Cont.

The table fully filled in may look like this:

Tag	TC1	TC2	TC3	TC4	TC5	TC6	TC7	TC8
Length ≥ 12	ON 12	OFF 11	75 in	74 in	15 in	15 in	30 in	31 in
Length < 75	12 IN	11 in	ON 75	OFF 74	15 in	15 in	30 in	31 in
Width ≥ 1	1 in	1 in	30 in	30 in	ON 1	OFF 0	10 in	11 in
$1 - 2 \times w \geq 10$	10 on	9 out	15 in	14 in	13 in	15 in	ON 10	OFF 9
Length	12	11	75	74	15	15	30	31
Width	1	1	30	30	1	0	10	11

We now need to determine the expected results, and then we have our test cases ready.

Decision Tables

- A decision table is a table showing the actions of the system depending on **certain combinations of input conditions**.
- Decision tables are often used to **express rules** and **regulations** for **embedded systems** and **administrative systems**.
- Decision tables are **brilliant for overview** and also for **determining if the requirements are complete**.
- Decision tables are useful to provide an overview of **combinations of inputs** and **the resulting output**.
 - The combinations are **derived from requirements**,
- which are expressed as something that is either **true** or **false**.
- The *coverage measure* for decision tables is the **percentage of the total number of combinations of input** tested in a test.

Decision Tables Cont.

❖ Decision Table Templates

Test design item number:	Traces:			
Assumptions:				
	TC1	TC2		TCn
Input condition 1				
Input condition n				
Action 1				
Action n				

- The fields in the table are:
 - **Test design item number:** Unique identifier of the test design item
 - **Traces:** References to the requirement(s) or other descriptions covered by this test design
 - **Assumptions:** Here any assumption must be documented

Decision Tables Cont.

- The table must have a row for each input and each action, and 2^n columns, where **n is the number of input conditions**.
- The cells are filled in with either **True** or **False** to indicate if the **input conditions**, respectively the actions are **true** or **false**.
- The **values for the resulting actions** must be extracted from **the requirements!**

❖ Decision Table Example

In this example we are going to test the following requirements.

[76] The system shall only calculate discounts for members.

[77] The system shall calculate a discount of 5% if the value of the purchase is less than or equal to € 100. Otherwise the discount is 10%.

[78] The system shall write the discount % on the invoice.

[79] The system must write in the invoices to nonmembers that membership gives a discount.

Decision Tables Cont.

Test design item number: 82

Traces: Req. [76]–[79]

Assumptions: The validity of the input is tested elsewhere

Note that we are only going to test the calculation and printing on the invoice, not the correct calculation of the discount.

	TC1	TC2	TC3	TC4
Purchaser is member	T	T	F	F
Value <= € 100	T	F	T	F
No discount calculated	F	F	T	T
5% discount calculated	T	F	F	F
10% discount calculated	F	T	F	F
Member message on invoice	F	F	T	T
Discount % on invoice	T	T	F	F

The test cases to execute can now be created from these test conditions and written directly into a test procedure.

Decision Table Cont.

Test procedure: 11

Purpose: This test procedure tests the calculation of discount for members.

Traces: Req. [76]–[79]

Tag	TC	Input	Expected output
TC1	1	Choose a member and create a purchase with a value less than € 100	A discount of 5% is calculated and this is written on the invoice.
TC2	2	Choose a member and create a purchase with a value of more than € 100	A discount of 10% is calculated and this is written on the invoice.
TC3	3	Choose a nonmember and create a purchase with a value less than € 100	No discount is calculated and the “membership gives discount” statement is written on the invoice.

Decision Table Cont.

Test procedure: 11

Purpose: This test procedure tests the calculation of discount for members.

Traces: Req. [76]–[79]

Tag	TC	Input	Expected output
TC1	1	Choose a member and create a purchase with a value less than € 100	A discount of 5% is calculated and this is written on the invoice.
TC2	2	Choose a member and create a purchase with a value of more than € 100	A discount of 10% is calculated and this is written on the invoice.
TC3	3	Choose a nonmember and create a purchase with a value less than € 100	No discount is calculated and the “membership gives discount” statement is written on the invoice.

Decision Tables Cont.

❖ Collapsed Decision Tables

- Sometimes it seems evident in a decision table that **some conditions** are without effect because one decision is decisive.
- For example if **one condition is False** an action seems to be **False** no matter what the values of all the other conditions are
- This could lead us to collapse the decision table, that is **reduce the number of combinations** by **only taking one of those** where the rest will give the same result.
- The **decision** as to whether **to collapse a decision table** or **not** should be based on **a risk analysis**.

Cause-Effect Graph

- A cause-effect graph is a graphical way of showing inputs or stimuli (causes) with their associated outputs (effects).
- The graph is a result of an analysis of requirements.
- Test cases can be designed from the cause-effect graph.
- The technique is a semiformal way of expressing certain requirements, namely requirements that are based on Boolean expressions.
- The cause-effect graphing technique is used to design test cases for functions that depend on a combination of more input items.
- In principle any functional requirement can be expressed as:
 - $f(\text{old state, input}) \rightarrow (\text{new state, output})$
- This means that a specific treatment (f = a function) for a given input transforms an old state of the system to a new state and produces an output.

Cause-Effect Graph Cont.

- We can also express this in a more practical way as:
 - $f(\text{ops1, ops2, ..., i1, i2, ..., i}) \rightarrow (\text{ns1, ns2, ..., o1, o2, ...})$
- where the **old state** is split into **a number of old partial states**, and **the input** is split into **a number of input items**.
 - The same is done for the **new state** and **the output**.
- The **causes in the graphs** are characteristics of **input items** or **old partial states**.
- The **effects in the graphs** are characteristics of **output items** or **new partial states**.
- Both **causes** and **effects** have to be statements that are either **True** or **False**.
 - **True** indicates that the characteristic is **present**;
 - **False** indicates its **absence**.

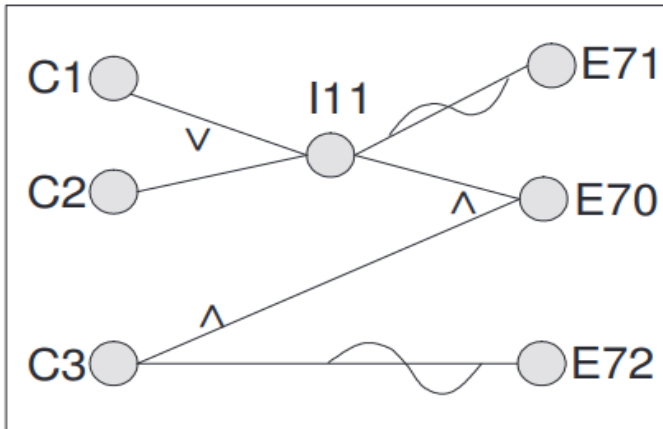


Cause-Effect Graph Cont.

- The graph shows the **connections** and **relationships** between the causes and the effects.
- ❖ **Cause-Effect Graph Coverage**
 - The coverage of the cause-effect graph can be measured as the **percentage of all the possible combinations of inputs tested in a test suite.**
- ❖ **Cause-Effect Graphing Process and Template**
 - A cause-effect graph is **constructed** in the following way based on **an analysis of selected suitable requirements**:
 - List and assign an ID to all causes
 - List and assign an ID to all effects
 - For each effect make **a Boolean expression** so that the **effect is expressed in terms of relevant causes**
 - Draw the cause-effect graph


Cause-Effect Graph Cont.


- An example of a cause-effect graph is shown here.



- Identified cause or effect: Must be labeled with the corresponding ID.
- — ● Connection between cause(s) and effect: The connection always goes from the left to the right.

- ^ This means that the causes are combined with AND, **that is all causes must be True for the effect to be True.**
- v This means that the causes are combined with OR, **that is only one cause needs to be True for the effect to be True.**

 This is a **negation**, meaning that a **True should be understood as a False**, and **vice versa**.

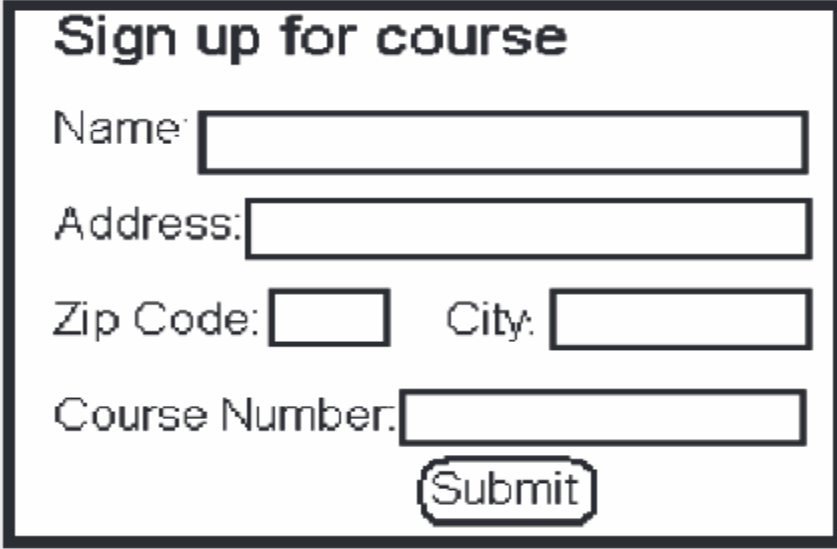
 The **arch** shows that all the causes must be combined with the Boolean operator; in this case the **three causes must be AND**.

Cause-Effect Graph Cont.

- Test cases may be derived directly from the graph.
- The graph may also be converted into a decision table, and the test cases derived from the columns in the table.
- Sometimes constraints may apply to the causes and these will have to be taken into consideration as well.

❖ Cause-Effect Graph Example

- In this example we are going to test a Web page, on which it is possible to sign up for a course. The Web page looks like this:

- 

Sign up for course

Name:

Address:

Zip Code: City:

Course Number:

Cause-Effect Graph Cont.

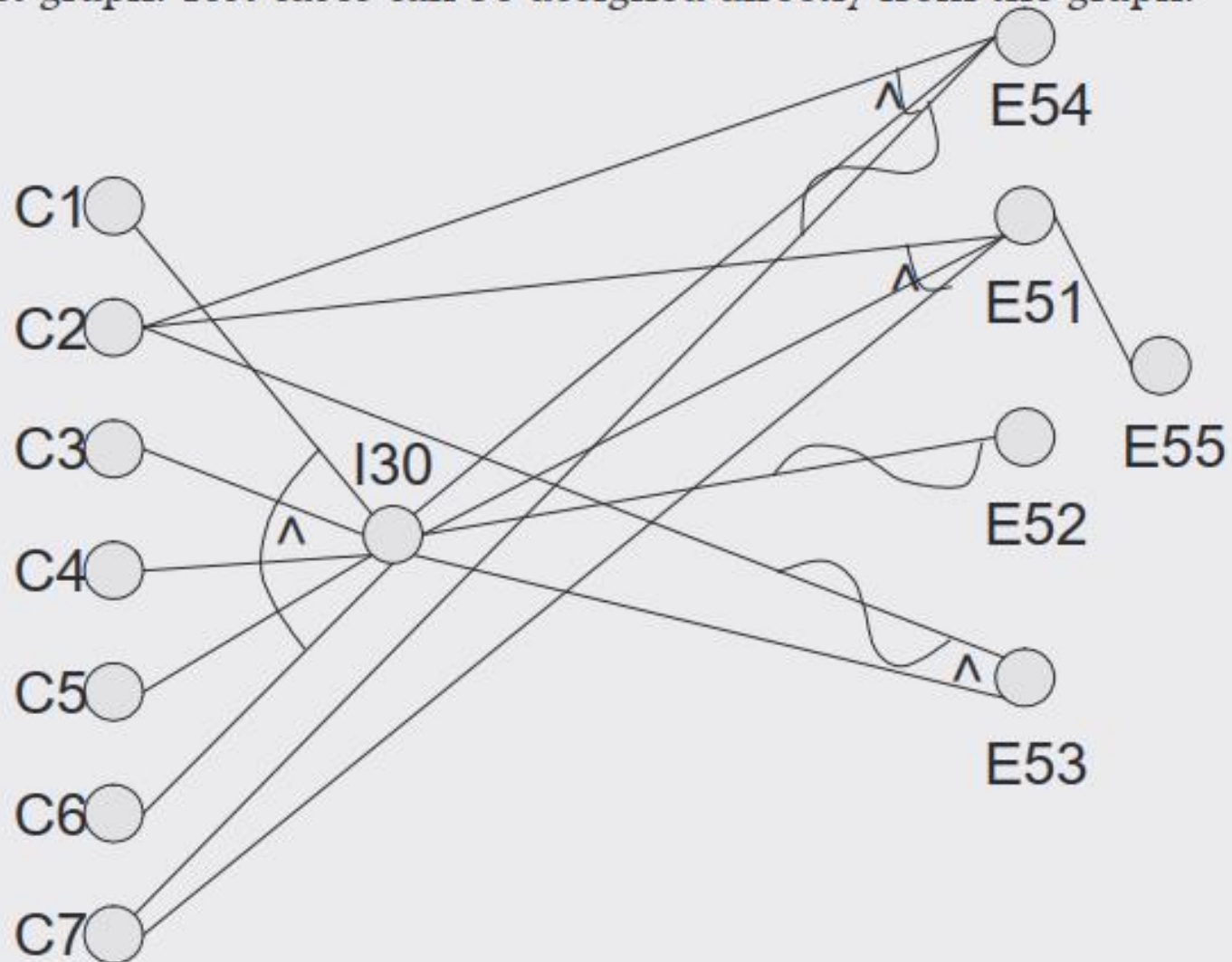
- First we make a complete **list of causes with identification**.
- The **causes** are derived from **a textual description of the form** (not included here):
 - C1. Name field is filled in
 - C2. Name contains only letters and spaces
 - C3. Address field is filled in
 - C4. Zip code is filled in
 - C5. City is filled in
 - C6. Course number is filled in
 - C7. Course number exists in the system

Cause-Effect Graph Cont.

- An intermediate Boolean may be introduced here, namely **I30** meaning that **all fields are filled in**. This is expressed as:
 - $I30 = \text{and}(C1, C3, C4, C5, C6)$
- The full list of effects with identification is:
 - E51. **Registration of delegate in system**
 - E52. Message shown: **All fields should be filled in**
 - E53. Message shown: **Only letters and spaces in name**
 - E54. Message shown: **Unknown course number**
 - E55. Message shown: **You have been registered**
- We must now express **each effect** as a Boolean expression based on the causes.
- They are:
 - $E53 = \text{and}(I30, \text{not } C2)$
 - $E54 = \text{and}(I30, C2, \text{not } C7)$
 - $E51 = \text{and}(I30, C2, C7)$
 - $E55 = E51$
 - $E52 = \text{not } I30$

Cause-Effect Graph Cont.

Drawing the causes and the effects and their relationships gives us the cause-effect graph. Test cases can be designed directly from the graph.



Cause-Effect Graph Cont.

❖ Cause-Effect Graph Hints

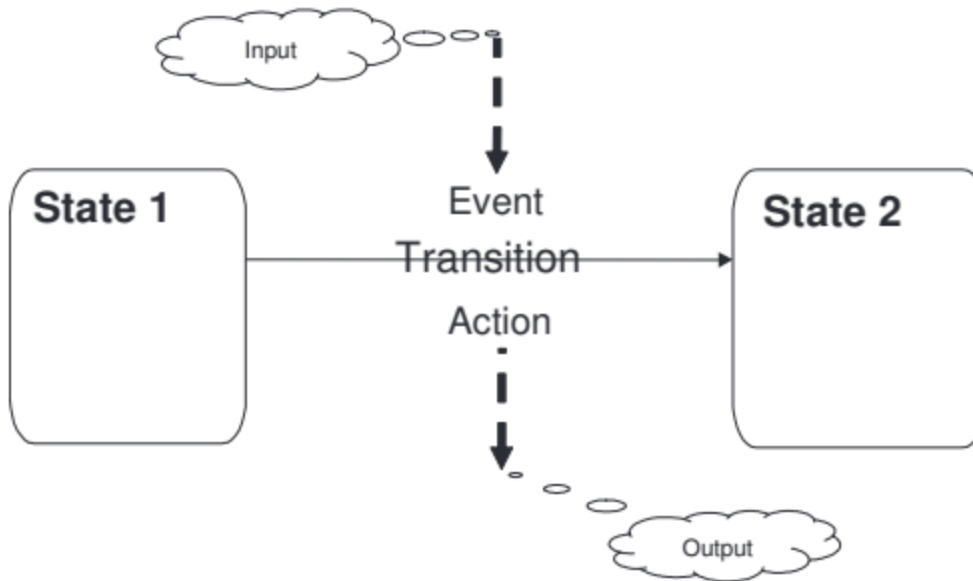
- It help in the **analysis phase** and the basis for the construction of a decision table from which test cases.
- **All the effects** can be filled into the decision table by looking at the cause effect graph or even from **the Boolean expressions directly**.
- **Cause-effect graphs** frequently become **very large** and **therefore difficult to work with**.
- To avoid this divide the specification into **workable pieces of isolated functionality**

State Transition Testing

- State transition testing is based on a state machine model of the test object.
- State machine modeling is a design technique, most often used for embedded software, but also applicable for user interface design.
 - the system can be in a number of well-defined states.
- A state is the collection of all features of the system at a given point in time, including
 - all visible data,
 - all stored data, and
 - any current form and field.
- The transition from one state to another is initiated by an event.
- An event will cause an action and the object will change into another state or stay in the same state.

State Transition Testing Cont.

- A transition = start state + event + action + end state
- The principle in a state machine is illustrated next.



State Transition Testing Cont.

❖ State Transition Testing Coverage

- Transitions can be performed in sequences.
- The smallest “sequence” is one transition at a time.
- The second smallest sequence is a sequence of two transitions in a row.
- Sequences can be of any length.
- The coverage for state transition testing is measurable for different lengths of transition sequences.
- The state transition coverage measure is:
 - Chows n-switch coverage
- where $n = \text{sequential transitions} - 1$.

State Transition Testing Cont.

- We could also say that $N = \text{no. of "in-between-states."}$
- Chows n -switch coverage is the percentages of all transition sequences of $n-1$ transitions' length tested in a test suite.

❖ State Transition Testing Templates

- To obtain Chows 0-switch coverage, we need a table showing all single transitions.
- These transitions are test conditions and can be used directly as the basis for test cases.
- The fields in the table are:
 - **Test design item number**: Unique identifier of the test design item
 - **Traces**: References to the requirement(s) or other descriptions covered by this test design
 - **Assumptions**: Here any assumption must be documented

State Transition Testing Cont.

- The information for each transition must be:
 - **Transition**: The identification of the transition
 - **Start state**: The identification of the start state (for this transition)
 - **Input**: The identification or description of the event that **triggers the transition**
 - **Expected output**: The identification or description of the **action connected to the transition**
 - **End state**: The identification of the end state (for this transition)

Test design item number:			Traces:
Assumptions:			
Transition			
Start state*			
Input			
Expected output			
End state*			

- ✓ * the “start” and “end” states are for each specific transition (**test condition**) **only**, not the state machine

State Transition Testing Cont.

- Testing to 100% Chows 0-switch coverage detects simple faults in transitions and outputs.
- To achieve a higher Chows n-switch coverage we need to describe the sequences of transitions.
- The table to capture test conditions for Chows 1-switch coverage is shown this

Test design item number:			Traces:
Assumptions:			
Transition pair			
Start state*			
Input			
Expected output			
Intermediate state*			
Input			
Expected output			
End state*			

- we have to include the intermediate state and the input to cause the second transition in each sequence.
- If we want an even higher Chows n-switch coverage we must describe test conditions for longer sequences of transitions

State Transition Testing Cont.

- A state table is a **matrix showing the relationships** between all states and events, and resulting states and actions.

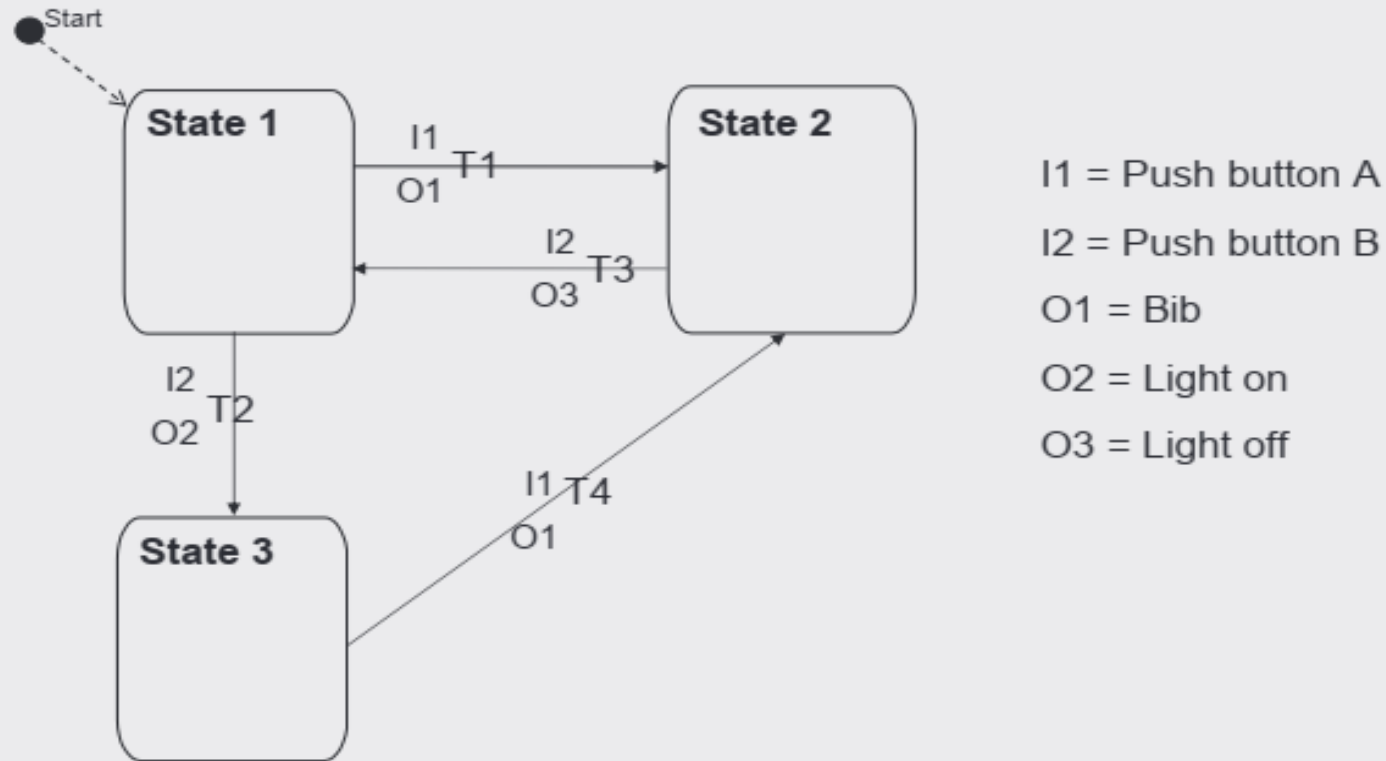
Start state \ Input	
	End state / Action

- The matrix must have **a row for each defined state** and **a column for each input (event)**.
- In the cross cell the corresponding **end state and actions must be given**.
- An **invalid transaction** is defined as a start state where the **end state** and **action is not defined for a specific event**.
- This should result in **the system** staying in the start state and no action or **a null-action being performed**
- The **“End state / Action”** for invalid transitions must be given as the identification of the **start state / “N” or the like**.
- **A test condition** can be identified from this table for each of the invalid transitions

State Transition Testing Cont.

❖ State Transition Testing Example

In this example we are going to identify test conditions and test cases for the state machine shown here.



Don't worry about what the system is doing—that is not interesting from a testing point of view.

State Transition Testing Cont.

The drawing of the state machine shows the identification of the states, the events (inputs), the actions (outputs), and the transitions. The descriptions of the inputs and outputs are given to the right of the drawing.

First of all we have to define test conditions for all single transitions to get Chows 0-switch coverage.

Test design item number: 2.4		Traces: State machine 1.1		
Assumptions: None				
Transition	T1	T2	T3	T4
Start state*	S1	S1	S2	S3
Input	I1	I2	I2	I1
Expected output	O1	O2	O3	O1
End state*	S2	S3	S1	S2

State Transition Testing Cont.

Identification of sequences of two transitions to achieve Chows 1-switch coverage results in the following table.

Test design item number: 2.5		Traces: State machine 1.1			
Assumptions: None					
Transition pair	T1/T3	T1/T3	T3/T2	T2/T4	T4/T3
Start state*	S1	S2	S2	S1	S3
Input	I1	I2	I2	I2	I1
Expected output	O1	O3	O3	O2	O1
Intermediate state*	S2	S1	S1	S3	S2
Input	I2	I1	I2	I1	I2
Expected output	O3	O1	O2	O1	O3
End state*	S1	S2	S3	S2	S1

We will not go further in sequences.

The next thing will be to identify invalid transitions. To do this we fill in the state table. The result is:

State Transition Testing Cont.

Identification of sequences of two transitions to achieve Chows 1-switch coverage results in the following table.

Test design item number: 2.5		Traces: State machine 1.1			
Assumptions: None					
Transition pair	T1/T3	T1/T3	T3/T2	T2/T4	T4/T3
Start state*	S1	S2	S2	S1	S3
Input	I1	I2	I2	I2	I1
Expected output	O1	O3	O3	O2	O1
Intermediate state*	S2	S1	S1	S3	S2
Input	I2	I1	I2	I1	I2
Expected output	O3	O1	O2	O1	O3
End state*	S1	S2	S3	S2	S1

We will not go further in sequences.

The next thing will be to identify invalid transitions. To do this we fill in the state table. The result is:

Start state \ Input	I1	I2
S1	S2/O2	S3/O2
S2	S2/N	S1/O3
S3	S2/O1	S3/N

State Transition Testing Cont.

We have two invalid transitions:

State 2 + Input 1 and State 3 + Input 2.

The test cases to execute can now be created from these test conditions and written directly into a test procedure.

Test procedure: 3.5

Purpose: This test procedure tests single valid and invalid transitions.

Traces: State machine 1.1

Prerequisites: The system is in State 1

Expected duration: 5 minutes

Tag	TC	Input	Expected output
T1	2	Reset to state 1 + push button B	The system bips + state 2
T2		Reset to state 1 + push button B	The light is on + state 3
IV2		Push button B again	Nothing changes
T4		Push button A	The system bips + state 2
IV1		Push button A again	Nothing changes
T3		Push button B	The light is off + state 1

- Reading Assignment
 - Classification tree method
 - Pairwise testing
 - Use case testing
 - Syntax testing