Chapter 4:
Test Design Techniques

Dynamic

Black box

Equivalence partitioning
Boundary value analysis
State transition testing
Decision tables
Use case based testing

Experience-based techniques

White box

Statement Coverage
Branch Coverage
Condition Coverage
Path Coverage

Static

Reviews/ walkthroughs
Control flow analysis
Data flow analysis
Compiler metrics/ analysis

Boundary Value Analysis:

Boundary analysis /1

- Boundary analysis extends equivalence class partitioning by introducing a **rule** for the **choice of representatives**
- The **edge values** of the **equivalence class** are to be tested **intensively**
- Why put more attention to the edges?
    - Often, the boundaries of values ranges are **not well defined** or lead to different interpretations
    - Checking if the boundaries were **programmed correctly**

- Please note:

  Experience shows that **error** occur **very frequently** on the **boundaries** of value ranges!

Defining boundary values

- If the **EC** is defined as **single numerical value**, for example **x= 5**, the neighboring values will be used as well
  - the representatives (of the class and its neighboring values) are: **4,5 and 6**

Boundary analysis example 3a

- Example 3a:

- value range for a discount in % : 0,00 ≤ x ≤ 100,00

- Definition of EC

3 classes:

1. EC: x < 0
2. EC: 0,00 ≤ x ≤ 100,00
3. EC: x > 100

- Boundary analysis

extends the representatives to:

2.EC: -0.01; 0.00; 0.01; 99.99; 100.00; 100.01

- Please note:

Instead of one representative for the valid EC, there are now six representatives (four valid and two invalid)

Boundary analysis example 3b

- **Basic scheme:** choose **three values** to be tested
  the exact boundary and the two neighboring values (within and outside the EC)

- **Alternative point of view**: since the boundary value belongs to the EC, only **two values** are needed for testing: **one within and one outside the EC**

- Example 3b:

- value range for a discount in %: 0.00 ≤ x ≤ 100.00
- **valid EC**: 0.00 ≤ x ≤ 100.00
- **boundary analysis**

  additional representatives are: **-0.01; 0.00; 100.00; 100.01**

  0.01 – same behavior as 0.00

  99.99 - same behavior as 100.00

- A programming error caused by a wrong comparison operator will be found with the two boundary values

Decision Table

Decision table testing

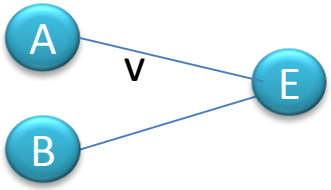
- Equivalence class partitioning and boundary analysis deal with **isolated input** conditions.
- However, an input condition may have an effect only in **combination** with other input conditions.
- All previously described methods do not take into account the effects of **dependencies and combinations.**
- Using the **full set of combinations** of all input equivalence classes usually leads to a very **high number of test cases** (test case explosion).
- With the help of **cause-and-effect graphs** and the decision tables derived from them, **the amount of possible combinations** can be **reduced** systematically to a subset.

- A cause-and-effect **diagram** uses a formal language
- A cause-and-effect diagram is created by translating the (mostly informal) specification of a test object into a formal language
- The test object is subject to a certain amount of effects that are traced back to their respective causes

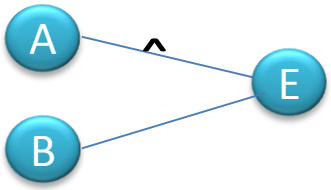
- Elements/symbols:
  - Assertion (If cause A – then effect E)

  - Negation (If cause A – then no effect E))

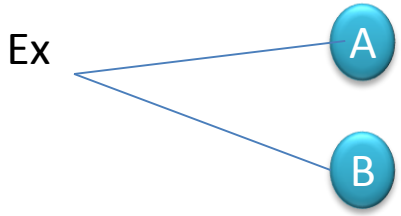
  - Or (If cause A or B – then effect E)



  - And (If cause A and B – then effect E)

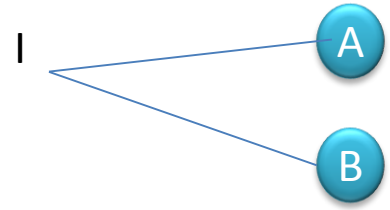


- Other elements of a cause-and-effect diagram:
  - Exclusive (Either cause A or cause B)



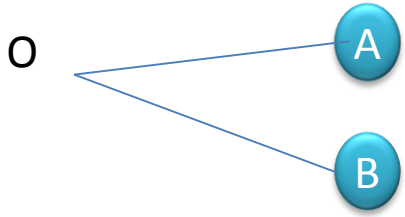
A	B	out
0	0	0
0	1	1
1	0	1
1	1	0

  - Inclusive (At least one of two causes: A or B)



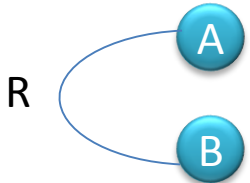
A	B	out
0	0	0
0	1	1
1	0	1
1	1	1

  - One and only one (One and exactly one of two causes: A or B)



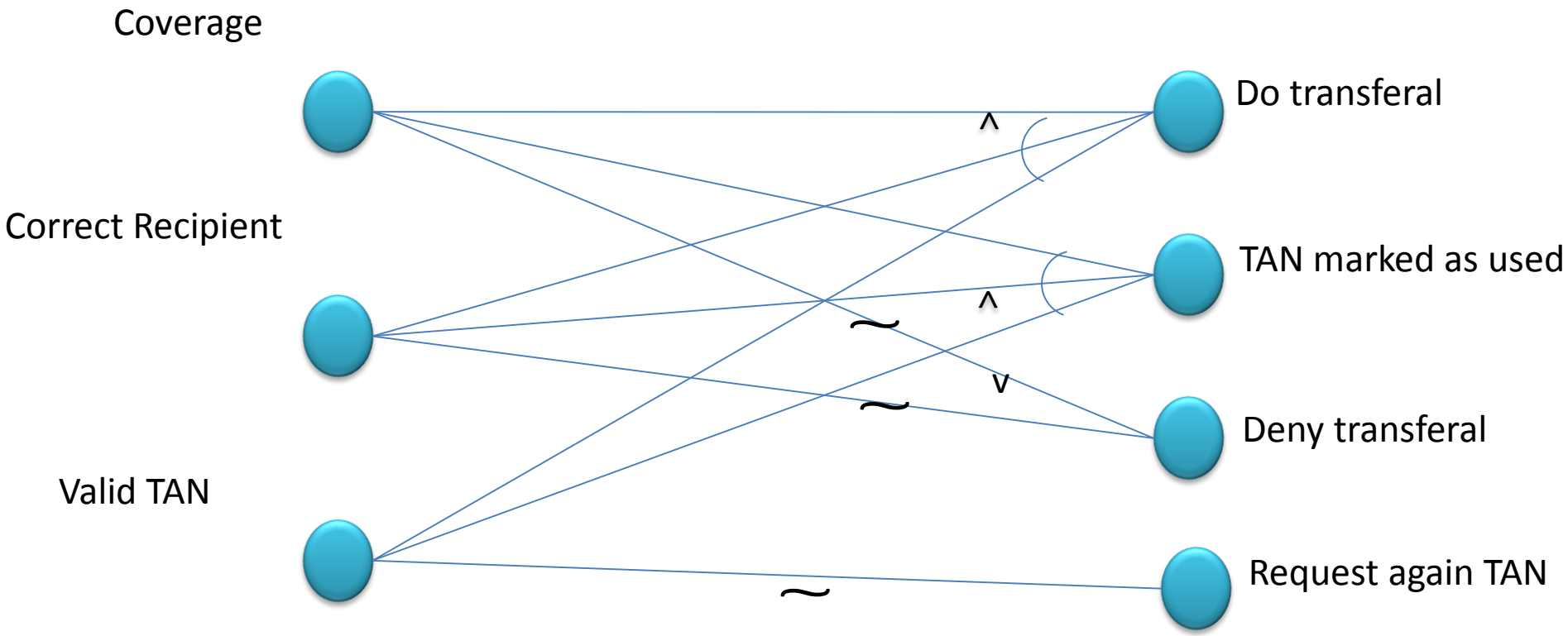
A	B	out
0	0	0
0	1	0
1	0	0
1	1	0
	1	1
1		1

  - Required (if cause A then also cause B)



- Example 5: Online-Banking

The user has identified himself via account number an PIN.
If having sufficient **Coverage**, he is able to set a transferal. To do this he must input the correct details of the **Recipient** and a **valid TAN.**



		T01	T02	T03	T04	T05
Preconditions (Causes)	Enough coverage	Yes	Yes	No	-	-
	Correct recipient	Yes	Yes	-	No	-
	Valid TAN	Yes	Yes	-	-	No
Activities (Effects)	Do transferal	Yes	No	No	No	No
	Mark TAN as used	No	Yes	No	No	No
	Deny transferal	No	No	Yes	Yes	No
	Request again TAN	No	No	No	No	Yes

Decision table testing

- Example 5: Online-Banking

		T01 T02	T03	T04	T05
Preconditions (Causes)	Enough coverage	Yes	No	-	-
	Correct recipient	Yes	-	No	-
	Valid TAN	Yes	-	-	No
Activities (Effects)	Do transferal	Yes	No	No	No
	Mark TAN as used	Yes	No	No	No
	Deny transferal	No	Yes	Yes	No
	Request again TAN	No	No	No	Yes

- Each table column represents a test case
- Creating a design table:
  - Choose an effect
  - Trace back along the diagram to identify the cause
  - Each combination of causes represents a column of the decision table (a test case)
  - Identical combinations of causes, leading to different effects, may be merged, to form a single test case

State Transition

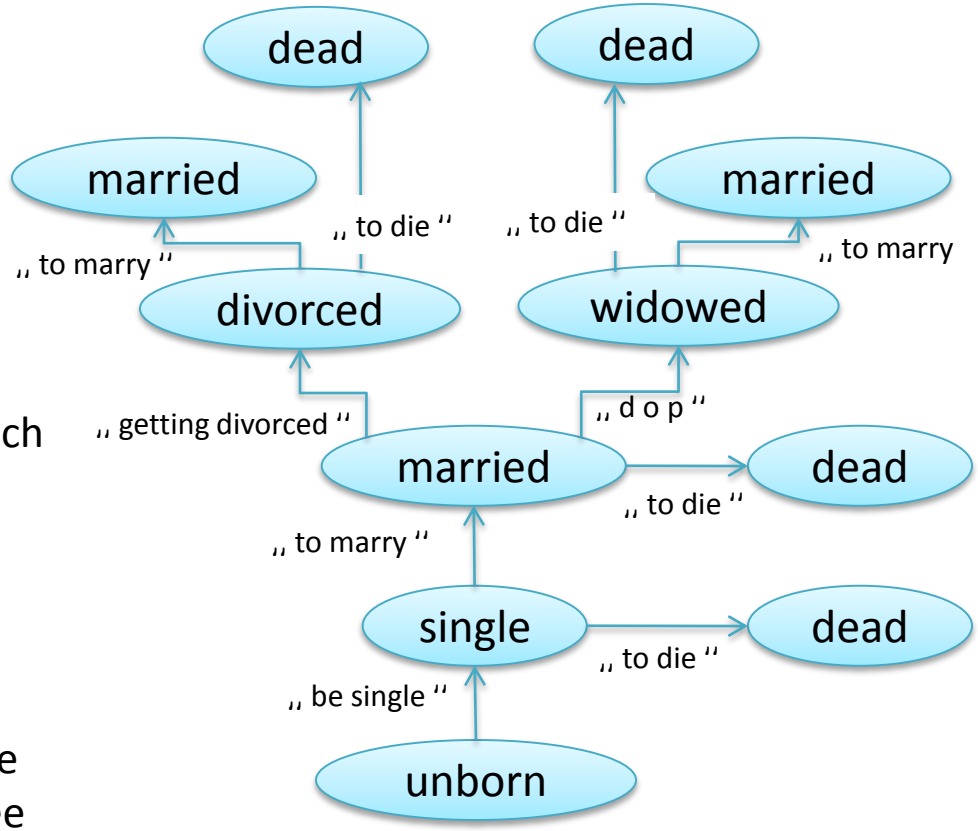
State transition testing

- Many methods only take into account the system behavior in terms of **input data** and **output data**
- Different states that a test object might take on are not taken into account
  - for example, results of actions that **happened in the past** actions, that caused the test object to be in a certain internal state

- The different states that a test object can take on are modeled using **state transition diagrams**
- **State transition analysis** is used to define state transition based test cases

State transition testing

- To determine test cases using a state transition diagram a **transition tree** is constructed:
- The initial state is the **root** of the tree
- For every state that may be reached from the initial state, A node is created which is connected to the root by a branch
- This operation is repeated and comes to an end if
  - the state of the node is an end state (a leaf of the tree) Or
  - the same node with the same state is already part of the tree



Event. „ d o p “ : death of partner

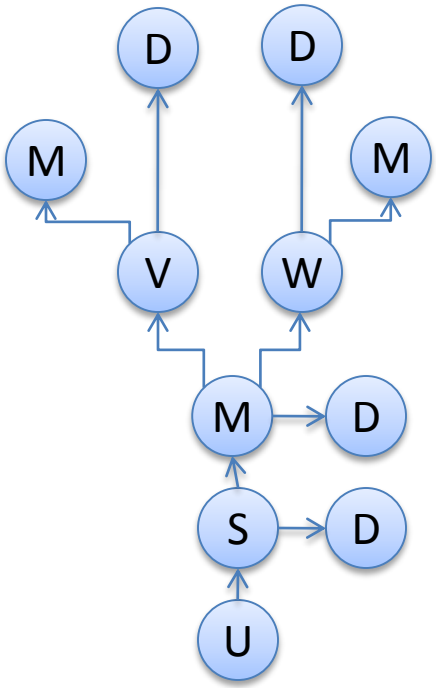
IV – Test Design Technique

03 – Black box techniques

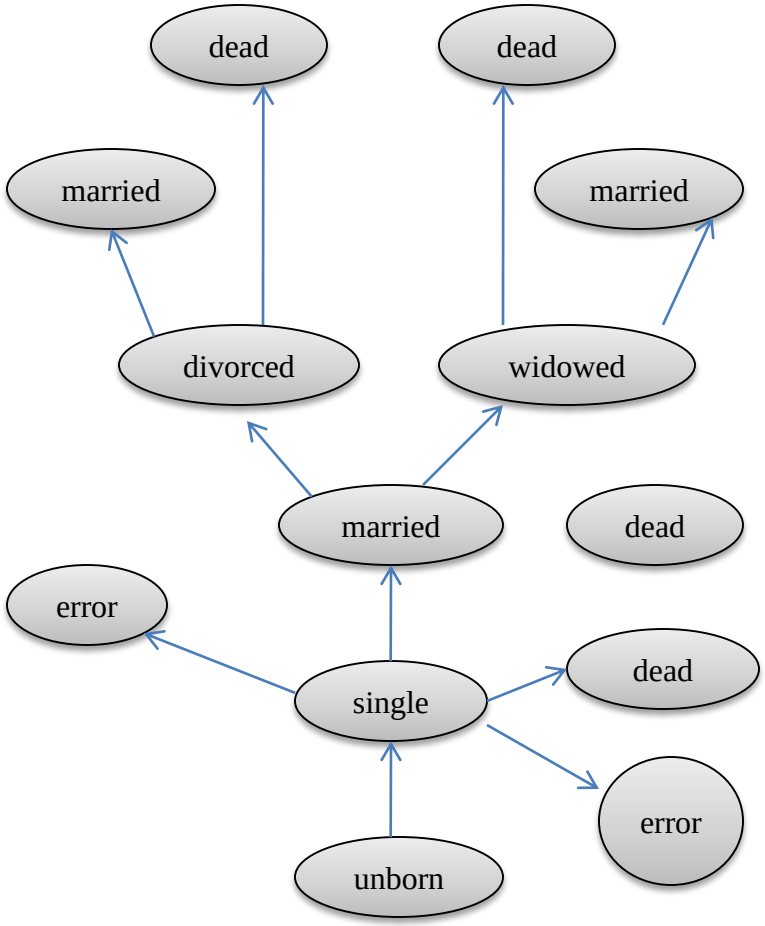
State transition testing

Every path from the root to a leaf then represents
a test case for state transition testing

The state transition tree for this example leads
to the following six test cases:



state1	state2	state3	state4	State5	End State
unborn	single	Dead			Dead
Unborn	Single	married	dead		Dead
Unborn	Single	married	widowed	dead	Dead
Unborn	single	married	widowed	Married	Married
Unborn	Single	married	divorced	Dead	Dead
Unborn	Single	married	divorced	married	married



-The transition tree of our example may now be extended using invalid transition (negative test cases robustness testing)

- Example: two possible **invalid** transitions- there are more
- **impossible** transitions between states can not be tested

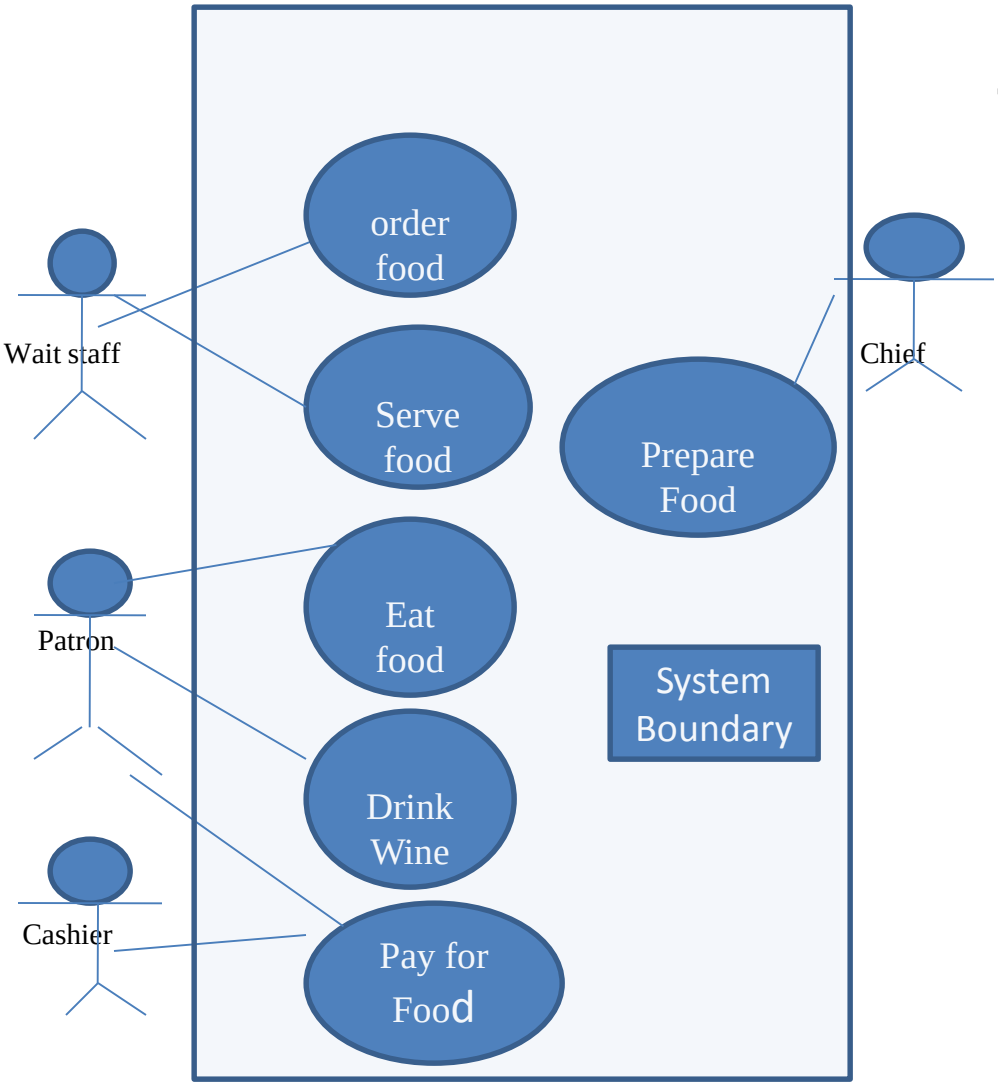
Use case based testing

- **Test cases** are derived directly from the **use cases** of the test object
  - The test object is seen as a **system** reacting with **actors**
  - A use case describes the **interaction** of **all involved actors** leading to an **end result** of the system
  - every use case has **pre-conditions** that have to be met the in order to execute the use case (the test case) successfully
  - Every use case have **post-conditions** describing the system after execution of the use case (the test case)
- Use cases are **elements** of the unified modeling language **UML***
  - **Use case diagram** are one of 13 different types of diagrams used by UML
  - A use case diagram is a diagram is a diagram describing a **behavior**, it does not describe the sequence events
  - It shows the system reaction from the **viewpoint of user**

UML is a non-proprietory specification language for object modeling

Use case based testing

- Example of a simple use case diagram (source: Wikipedia)



The diagram on the left describes the functionality of a simple restaurant system
Use case are represented by **ovals** and the actors are represented by **stick** figures.
The patron actor can eat food. Pay for food or drink wine
Only the chief actor can prepare food. Note that both the patron and the cashier are involved in the pay for food use case
The box defines the boundaries of the restaurant system being modeled, the actors are not

Use case based testing

- Every use case describes a certain task (user-system-interaction)
- **Use case** descriptions include, but are not limited to:
    - Pre conditions
    - Expected results/system behavior
    - Post: conditions
- These descriptive elements are also used to define the corresponding **test cases**
- Every **use case** may be used as the basis for a **test case**
- **Every alternative** within the diagram corresponds to a **separate test case**
- Typically, information provided with a use case has **not enough detail** to define the **test case** directly, Additional data is needed (input data, expected results) to make up a test case