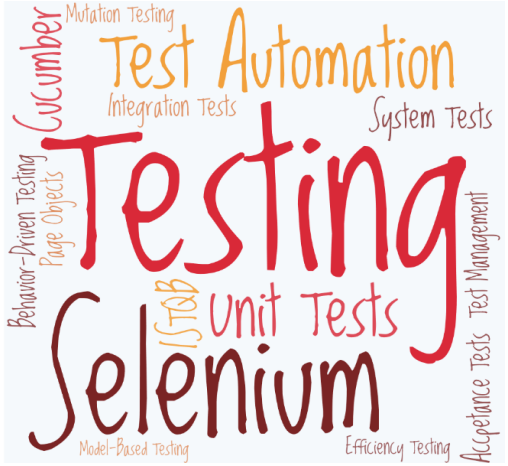
183.290 Software Testing



Test Design Techniques

WS 2020

Markus Zoffi

peso@inso.tuwien.ac.at
https://peso.inso.tuwien.ac.at

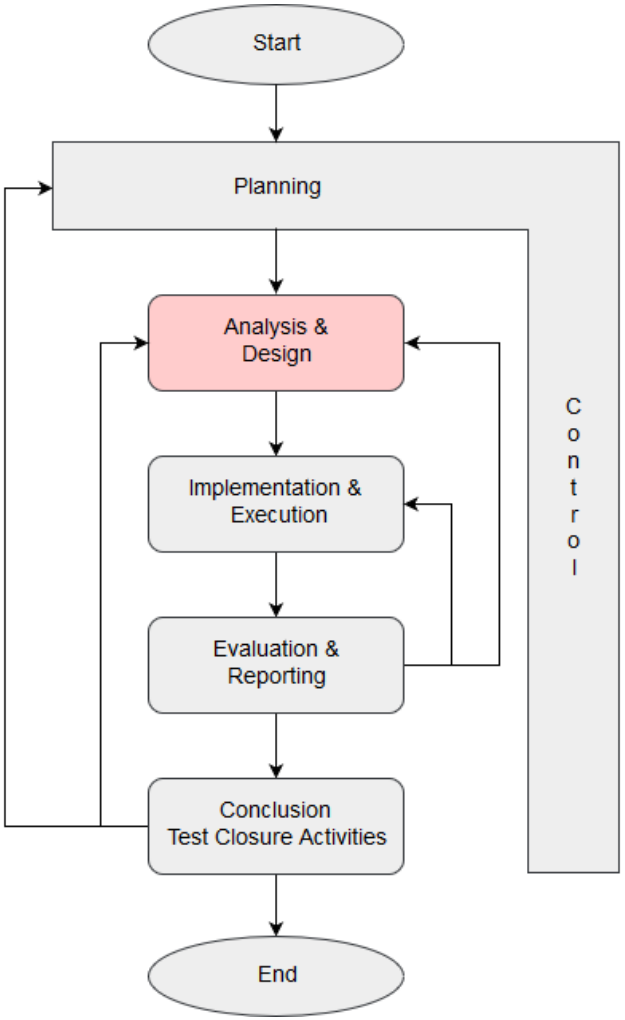
INSO - Industrial Software
Institut für Information Systems Engineering | Fakultät für Informatik | Technische Universität Wien

Content

Test Design Techniques

- **The purpose of test techniques is to help in identifying test conditions, test cases, and test data**
- **Some techniques are more applicable to certain situations and test levels - others are applicable to all test levels**
- **When creating test cases in general a combination of test techniques is used to achieve the best results**
- **The choice of test techniques depends on:**
  - Complexity of the System under Test (SUT)
  - Time and budget
  - Customer or contractual requirements
  - Risk levels / Types
  - Available documentation
  - Tester experience, knowledge and skills, …

ISTQB Fundamental Test Process



Test Design is …

- Designing and prioritizing high level test cases
- Identifying necessary test data to support the definition of test cases

… using Test Design Techniques

Quality Assurance Methods

- **Static Methods**
  - Identifies errors while the SUT is NOT executed

180.764 Software Qualitätssicherung

not part of this lecture, but part of ISTQB Foundation Level

- **Dynamic Methods**
  - Test during execution of the SUT

Reviews
data analysis
...

black box

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

white box

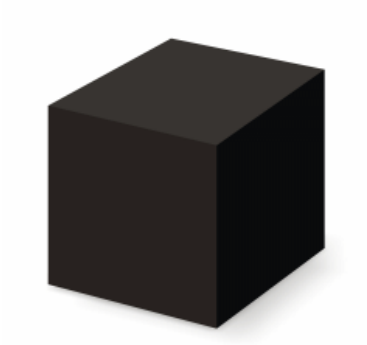
Coverage methods
- statement
- branch
- decision
- condition

Experience based testing

Testing Techniques

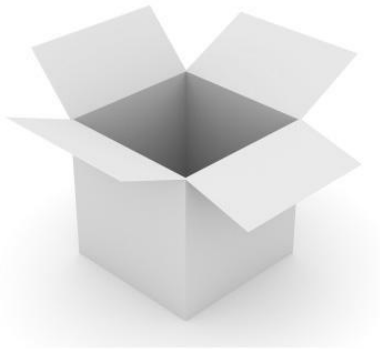
Black Box Tests

- Based on specification
- No knowledge on internal structure available
- Data driven
- Coverage of specification can be determined



White Box Tests

- Based on internal structure (source code, modules, …)
- Knowledge on internal structure required
- Logic driven
- Coverage can be measured (statement, branch, path, condition)



Testing Techniques

Experience based test design techniques

- Derive test cases from knowledge and experience
- Usage of knowledge and experience about the test objects and test environment
- This knowledge and experience includes
  - expected use of the software
  - defects
  - distribution of those defects

- Can be combined with black- and white box tests

Content

Black Box Tests

- Based on Specification
- No knowledge on internal structure available
- Data driven
  - Variation of input parameters
  - Comparing output to expecting results

- **Black Box testing methods**
  - Equivalence partitioning
  - Boundary value analysis
  - Decision tables
  - State based testing
  - Use case Testing

Equivalence Partitioning

- Dividing possible input values into partitions (equivalence classes)
- All the members of a given partition lead to the same expected outcome
- Equivalence partitions for both valid and invalid values
- Each value must belong to one and only one equivalence partition

- Sufficient to select one representative value per partition
- Usage in combination with boundary value analysis

Equivalence Partitioning Example

Example: Car Insurance

- The price of a car insurance is calculated by the age of the driver and the car's HP amount
  - Drivers up to the age of 30 receive a 10% mark up
  - An additional 10% mark up is added for vehicles over 150 HP
  - Drivers have to be at least 18 years old to conclude the insurance

Age 0 - 17		Age 18 - 30		Age >30	
HP <= 150	HP > 150	HP <= 150	HP > 150	HP <= 150	HP > 150
x	x	10%	20%	0%	10%

Equivalence Partitioning Example

Example: Car Insurance

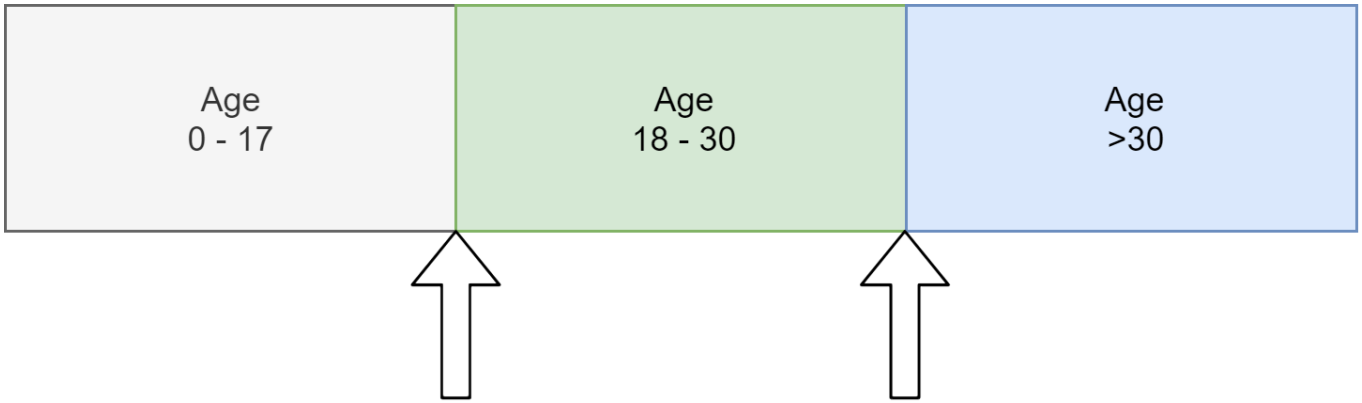
- Class 1: Drivers Age
  - A1: Age 0 - 17
  - A2: Age 18 - 30
  - A3: Age 30+
- Class 2: HP
  - B1: HP <= 150
  - B2: HP >150
- Expected Result

Age 0 - 17		Age 18 - 30		Age >30	
HP <= 150	HP > 150	HP <= 150	HP > 150	HP <= 150	HP > 150
x	x	10%	20%	0%	10%

- Possible Testcases
  - T1: Age 18(A2), HP 149(B1) – expected mark up: 10%
  - T2: Age 30(A3), HP 149(B1) – expected mark up : 10%
  - T3: Age 31(A3), HP 150(B2) – expected mark up : 0%
  - T4: Age 17(A1), HP 149(B1) – expected discount: none / invalid
  - …

Boundary Value Analysis

- **Extension of equivalence partitioning**
- **The minimum and maximum values of a partition are its boundary values**



- Boundary Values: [17] [18, 30] [31]
  Two (or more) Value Boundary Analysis: [16, 17] [18, 30] [31, 32]

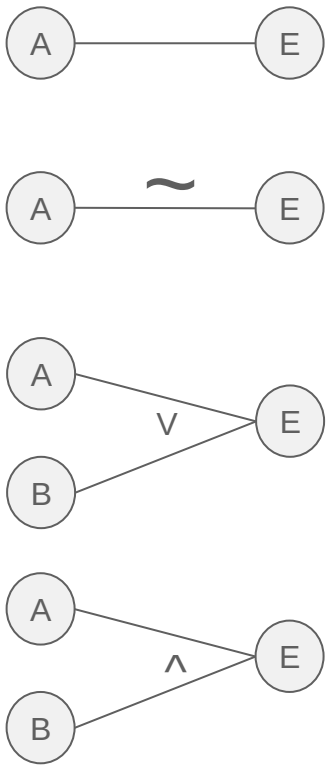
- Behavior at the boundaries of equivalence partitions is more likely to be incorrect than behavior within the partitions

Decision Testing

- **Decision Testing**
  - Good way to record complex business rules
  - Dependencies between input values are not considered in boundary value analysis or equivalence classes
  - Complete testing is not possible
    decision tables should help to reduce the amount of all possible combinations to a subset of combinations
- **Cause Effect Diagram**
  - A graphical representation used to display interrelationships of possible root causes of a problem
- **Decision Table**
  - A table used to show sets of conditions (often inputs) and the actions (often outputs) resulting from them

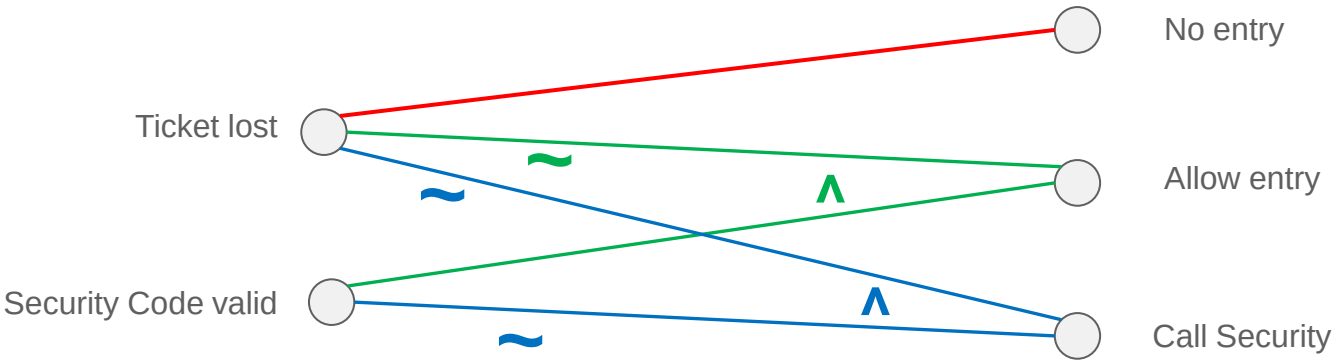
Decision Making – Cause Effect Diagram

- **Cause – effect diagram**
- Elements of a cause effect diagram
  - identity
    - If cause A then effect (E)
  - negation
    - If **not** cause A then effect
  - or
    - If cause A **or** cause B then effect
  - and
    - If cause A **and** cause B then effect



Decision Making – Cause Effect Diagram

People get access to a conference if they have a ticket with a valid security code. If the security code is invalid the event security has to be called. If someone has no ticket, no entry is allowed



Decision Table

	1	2	3
Ticket „lost“	N	N	Y
Security code valid	Y	N	-
Allow Entry	Y		
Security		Y	
No Entry			Y

Exercise: Decision Table

- **Book publisher offers the following discount :**
  - Customer „Bookstore“: If ordering 6 or more books, grant 25% discount. Otherwise no discount.
  - Customer „Library“: Discount with order size
    - 6-19 → 5%
    - 20-49 → 10%
    - 50+ → 15%

- **Define the decision table**
- **Reduce the decision table, if possible**

Exercise: Decision Table

	test case	1	2	3	4	5	6
Conditions	Customer Type	Bookstore	Bookstore	Library	Library	Library	Library
	Order	5	6	5	19	49	51
Actions	Discount	0%	25%	0%	5%	10%	15%

	test case	1	2	3	4	5	6
Conditions	Customer Type	Bookstore	Bookstore	Library	Library	Library	Library
	Order >5	N	Y	N	Y	Y	Y
	Order >= 20			N	N	Y	Y
	Order >= 50			N	N	N	Y
Actions	No Discount	Y		Y			
	5% Discount				Y		
	10% Discount					Y	
	15% Discount						Y
	25% Discount		Y				

Exercise: Decision Table

	tetst case	1	2	3	4	5	6
Conditions	Bookstore	Y	Y	N	N	N	N
	Order > 5	N	Y				
	Library	N	N	Y	Y	Y	Y
	Order >5			N	Y	N	N
	Order >= 20			N	N	Y	N
	Order >= 50			N	N	N	Y
Actions	No Discount	Y		Y			
	5% Discount				Y		
	10% Discount					Y	
	15% Discount						Y
	25% Discount		Y				

Which test cases are reducible?

Test case 1 and 3 have equal actions → reducible

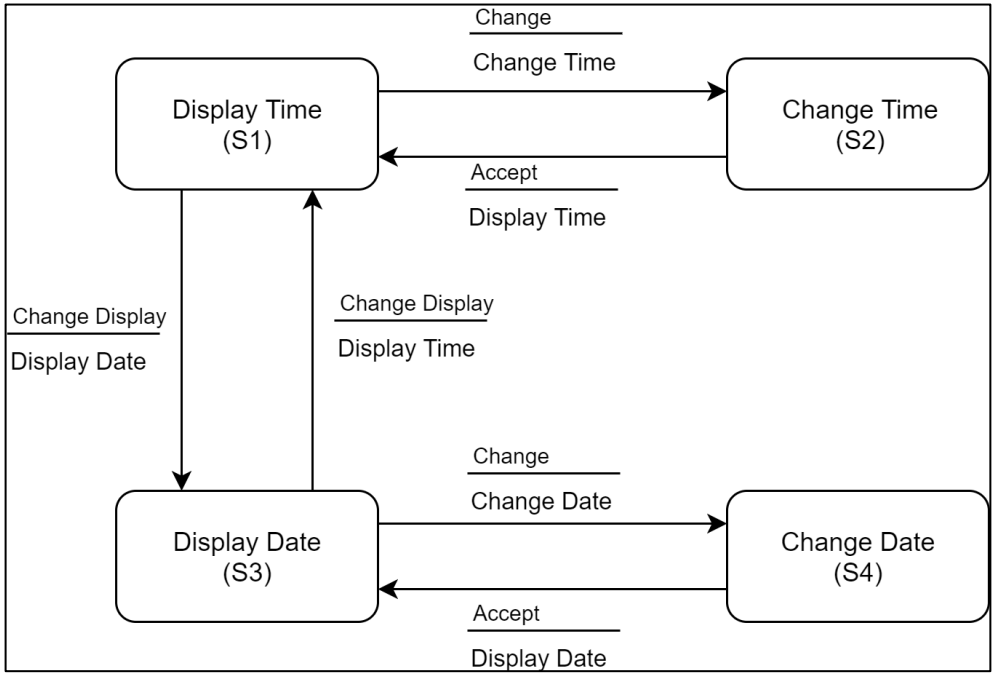
It does not matter if the the customer is a bookstore or a library when ordering less than 5 books!

State Transition Testing

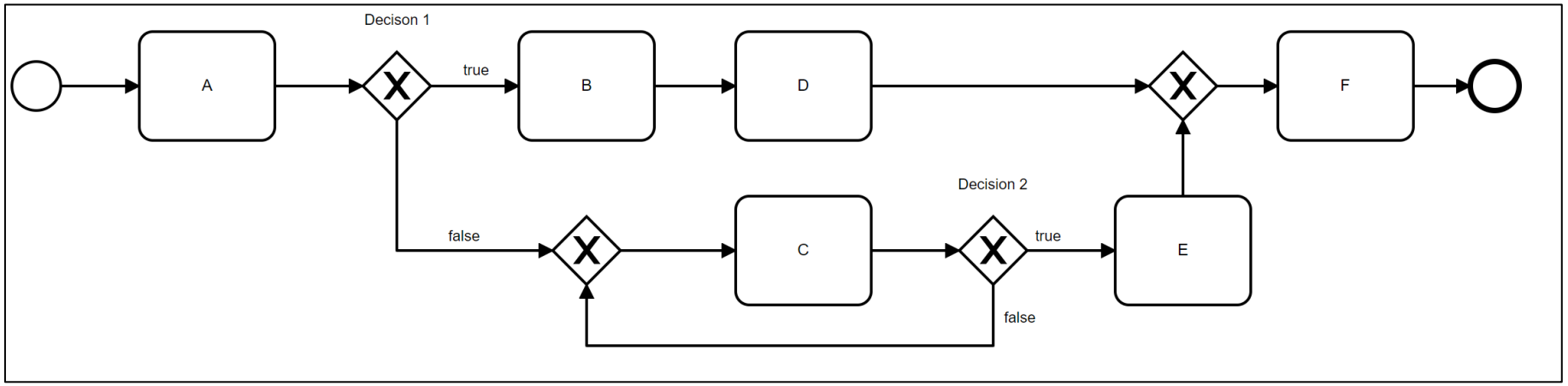
- **Representation of a system's behavior as states**
- **Often modeled as UML state transition diagram**
- **Usually shows only valid transitions**
- **Used for:**
  - Menu-based Applications
  - Process flows (e.g. BPMN)

- **Different levels of tests**
  - 0-Switch-Coverage
    - From one state to another state
  - 1-Switch-Coverage
    - From one state to another with an intermediate state (the intermediate is the one that is counted)

State Transition Testing



State Transition Diagram

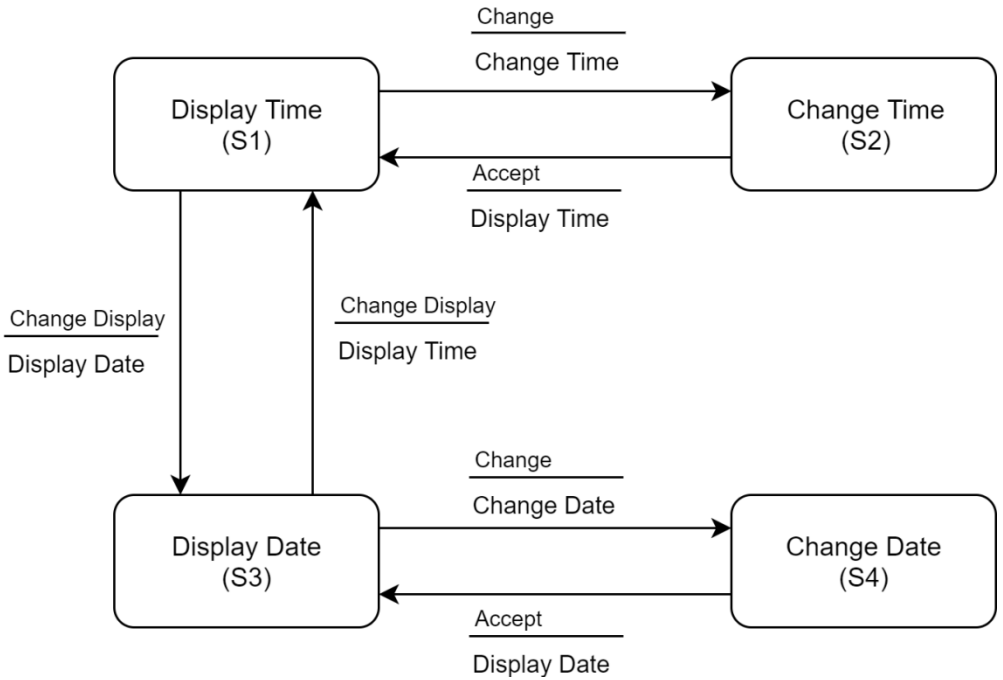


BPMN Diagram

State Transition Testing

0-Switch

	T1	T2	T3	T4	T5	T6
Start state	S1	S1	S2	S3	S3	S4
event	Change	change display	accept	change display	change	accept
impact	Change Time	Display date	display time	display time	change date	display date
end state	S2	S3	S1	S1	S4	S3



State Transition Testing

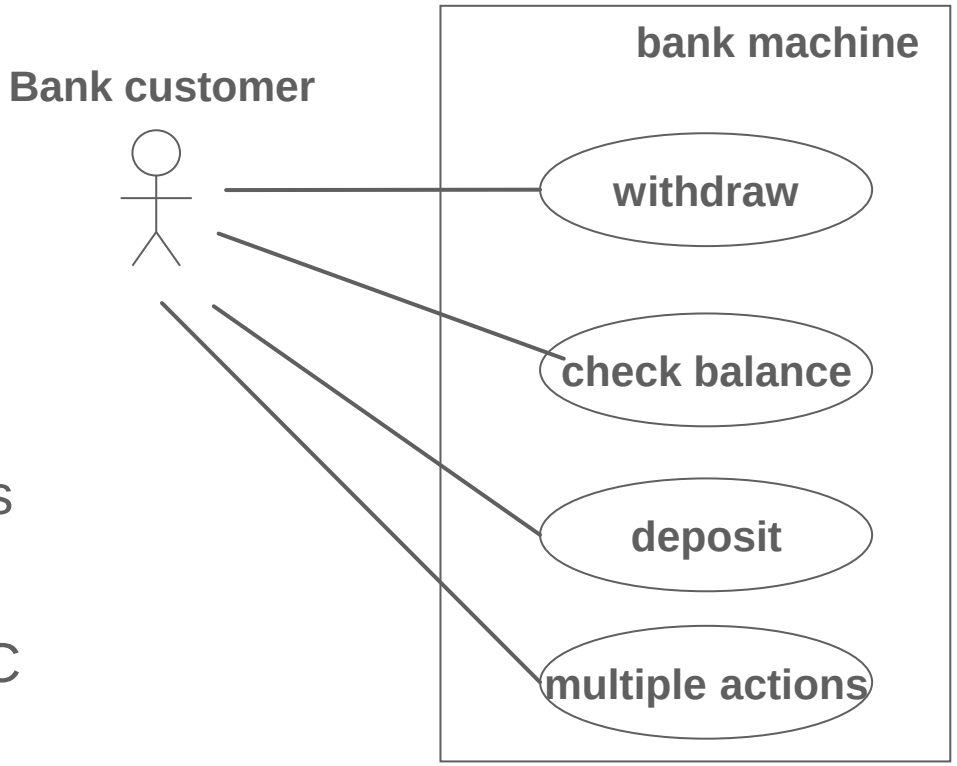
1-Switch

Testcase	start state	switching state	endstate
A	S1	S2	S1
B	S1	S3	S1
C	S1	S3	S4
D	S2	S1	S2
E	S2	S1	S3
F	S3	S1	S3
G	S3	S1	S2
H	S3	S4	S3
I	S4	S3	S1
J	S4	S3	S4

Use Case Testing

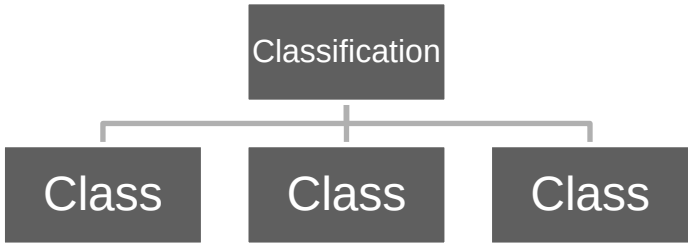
Derive Test Cases from a UML – Use Case Diagram

- Every use case describes a task, which interacts with the system
- Use cases includes:
  - Preconditions
  - Expected results
  - Postconditions
- Every use case can be transformed to a test case
- Tests are designed to exercise the defined behaviors
- Coverage
  UC tested / total number of UC



Classification Tree Method

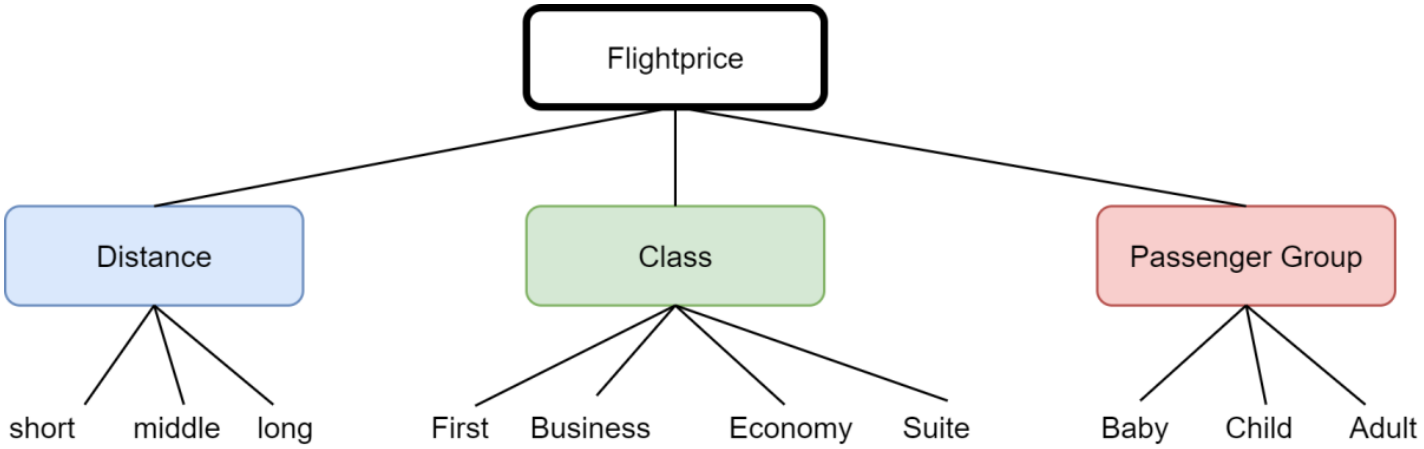
- **Combinatorial Testing Techniques**
- Graphical visualization of combinations which should be tested
- Used when there are **more combinations** than are feasible to test in the **time allowed**
- Identifies a suitable subset of combinations to achieve a predetermined level of coverage



CTM – Class Coverage

- **Full combination (*)**
  - Every class is combined with all classes from other classifications
- **Minimal combination (+)**
  - Every class is used at least once
- **Pairwise combination**
  - Two classifications are fully combined. Further classes are combined minimally.
- **n-wise combination**
  - Same as pairwise but full combination with n classifications

Classification Tree Method – Example



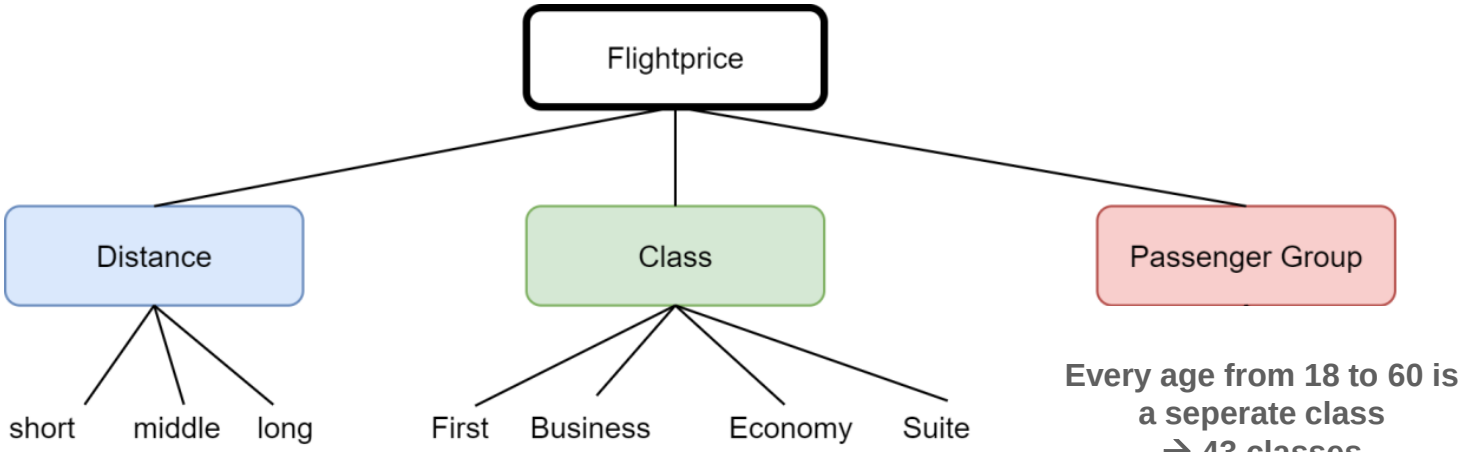
How many combinations with **full combination**?

3 x 4 x 3 = 36 test cases

Pairwise with **flight distance** and **flight class** and minimal **passenger group**?

3 x 4 = 12 minimal combined with passenger group
→ 12 test cases

Classification Tree Method – Example



Pairwise **flight distance** with **flight class** and minimal **passenger group**?

3 x 4 = 12 minimal combined with 43

→ 43 test cases necessary

Classification Tree Method – Example

3-wise combination:

	Distance	Class	Passanger Group
1	short	First Class	baby
2	short	First Class	child
3	short	First Class	adult
4	short	Business	baby
5	short	Business	child
6	short	Business	adult
7	short	Economy	baby
8	short	Economy	child
9	short	Economy	adult
10	short	Suite Class	baby
11	short	Suite Class	child
12	short	Suite Class	adult
13	middle	First Class	baby
14	middle	First Class	child
15	middle	First Class	adult
16	middle	Business	baby
17	middle	Business	child
18	middle	Business	adult
19	middle	Economy	baby
20	middle	Economy	child
21	middle	Economy	adult
22	middle	Suite Class	baby
23	middle	Suite Class	child
24	middle	Suite Class	adult
25	long	First Class	baby
26	long	First Class	child
27	long	First Class	adult
28	long	Business	baby
29	long	Business	child
30	long	Business	adult
31	long	Economy	baby
32	long	Economy	child
33	long	Economy	adult
34	long	Suite Class	baby
35	long	Suite Class	child
36	long	Suite Class	adult

Classification Tree Method – Example

- **Constraint**
  First and **Suite Class** - only available at **long haul flights**
- Eliminate **non-valid** combinations
  2 Flight Classes
  x 2 Flight Distances
  x 3 Passenger Groups
  = 12 invalid combinations
- 36 – 12 = 24 valid test cases

	Distance	Class	Passenger Group
1	short	Business	baby
2	short	Business	child
3	short	Business	adult
4	short	Economy	baby
5	short	Economy	child
6	short	Economy	adult
7	middle	Business	baby
8	middle	Business	child
9	middle	Business	adult
10	middle	Economy	baby
11	middle	Economy	child
12	middle	Economy	adult
13	long	First Class	baby
14	long	First Class	child
15	long	First Class	adult
16	long	Business	baby
17	long	Business	child
18	long	Business	adult
19	long	Economy	baby
20	long	Economy	child
21	long	Economy	adult
22	long	Suite Class	baby
23	long	Suite Class	child
24	long	Suite Class	adult

Classification Tree Method – Example

- Usage mostly in automated tests
- It is not guaranteed that maximum combination brings full path coverage
- In some cases it is too complicated to generate a classification tree (and the respective classes)
- Tools usually are used to find the minimum set of combinations

Content

White Box Tests

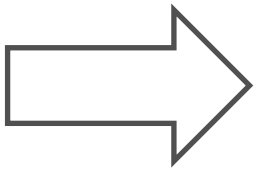
- Based on internal structure (Sourcecode, modules, …)
- Knowledge on internal structure required
- Coverage can be measured as percentage
- Applicable for all test levels – mostly used at component test level

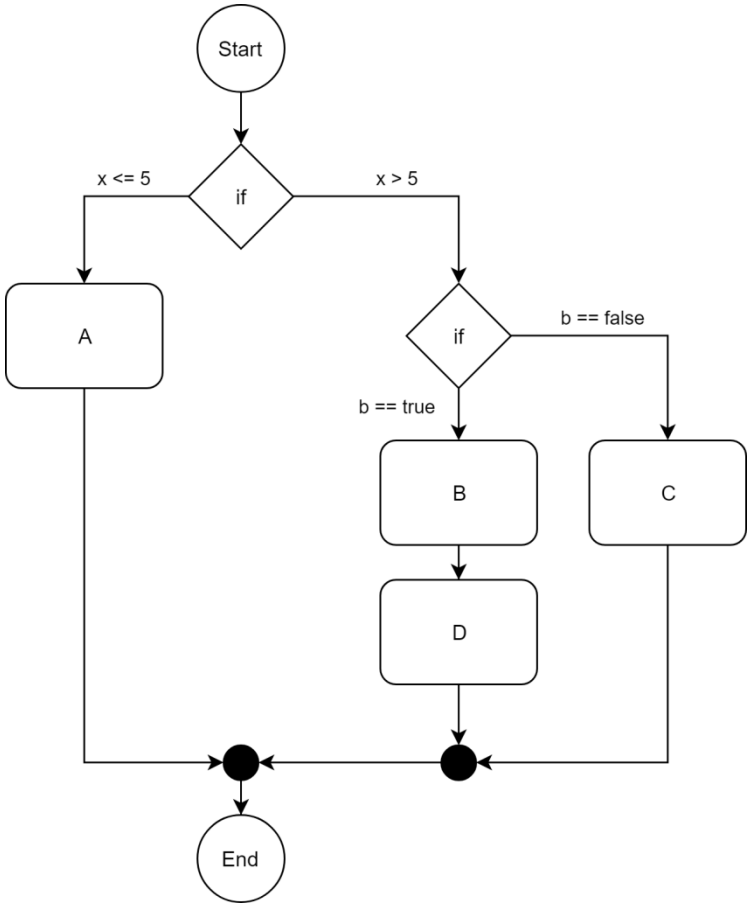
- **White box testing methods**
  - Coverage based testing methods
    - Statement Coverage (C0)
    - Branch Coverage (C1)
    - Path Coverage (C2)
    - Condition Coverage (C3)

Coverage Based Testing Methods

- Based on **control-flow diagram**

```
if(x <= 5) {
    A();
}
else {
    if(b) {
        B();
        D();
    }
    else {
        C();
    }
}
```





Statement Coverage

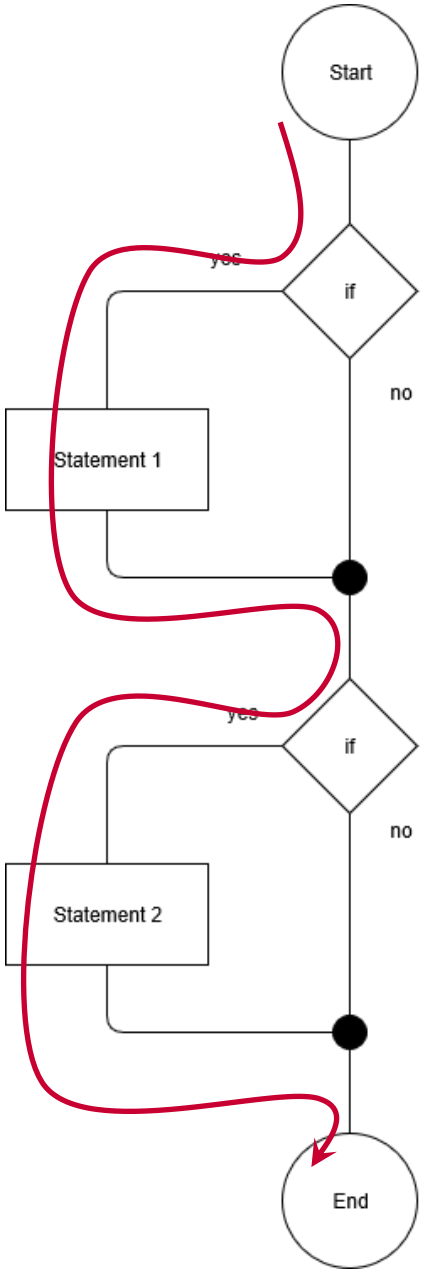
- Goal: **every statement** has to be executed at least once
- Complete statement coverage is achieved, when every statement was executed at least once
  - Used to identify „dead code“
    - Statement that is never reached
- Very weak criterion for reliable test execution

Statement Coverage

- **Example**
  - Statement Coverage

- Each statement should be executed at least once

- 1 test case

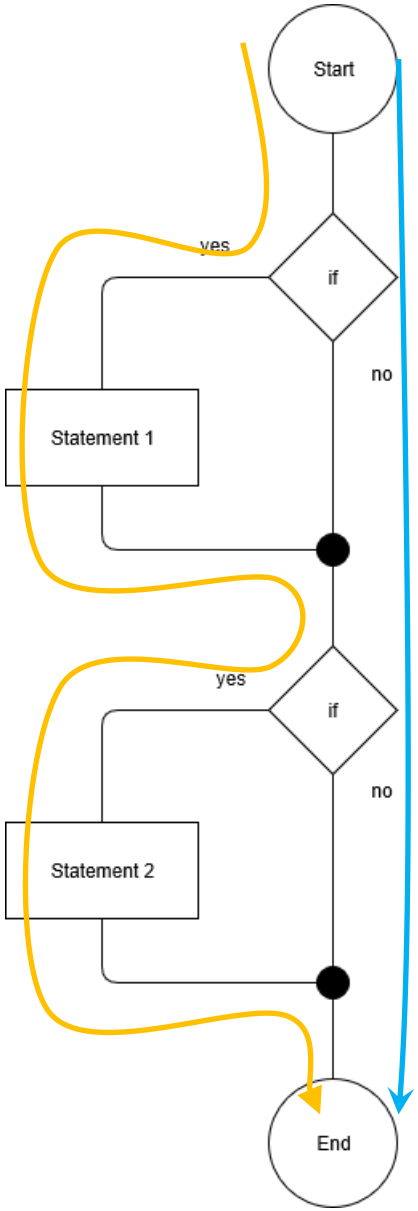


Branch Coverage

- Includes complete statement coverage
- Goal: **every edge** has to be executed at least once
  - Identifies non executable branches
  - Supports optimization of frequent branches
- Every condition must evaluate to **true** and **false**
- Disadvantages
  - Inadequate method for loops
  - Complex logic in condition statements is not considered

Branch Coverage

- **Example**
  - Branch Coverage
- Each edge should be executed at least once
- 2 test cases

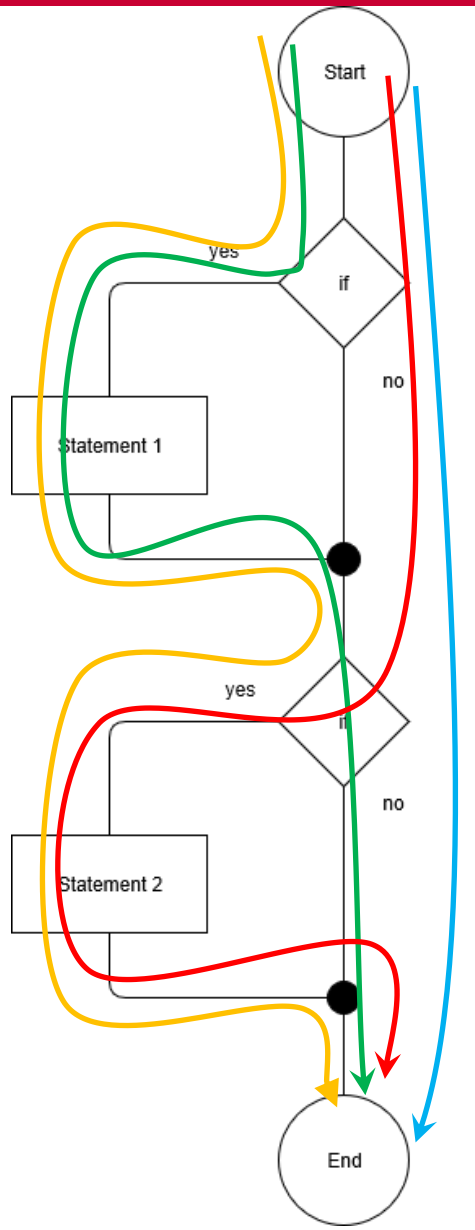


Path Coverage

- Goal: **every path** has to executed at least once
- Not feasible for complex modules
  - Infinite number of possible paths
  - Multiple possible paths for loops

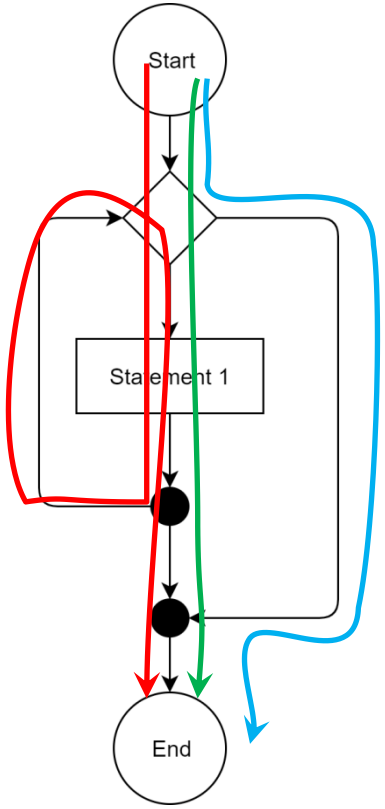
Path Coverage

- **Example**
  - Path Coverage
- Each path should be executed at least once
- 4 test cases



Loop Coverage

- **Many loops have entry and/or exit criteria**
- **Test cases should cover**
  - No execution of the loop
  - One execution of the loop
  - More than one execution of the loop



Condition Coverage

- Validation of aggregated conditions
- Goal: Every **single condition** is required to be executed as t**rue** and **false**

- **Condition Coverage**
  - Every condition has to be true **and** false once
- **Multiple Condition Coverage**
  - Also combinations of conditions will be covered
- **Condition Determination Coverage**
  - Combinations will be reduced. Conditions which do not influence the result of a combination will be deleted

Condition Coverage

- Condition: (((a == 0) || (b < 5)) && ((x < 6) || (y == 0)))
- Or in general: ((A || B) && (C || D))

- If a, b, x and y are self-contained – A, B, C and D can independently become true and false

- Example
  - **Condition Coverage**
  - TC1: A = false, B = false, C = false, D = false
  - TC2: A = true, B = true, C = true, D = true

Condition Coverage

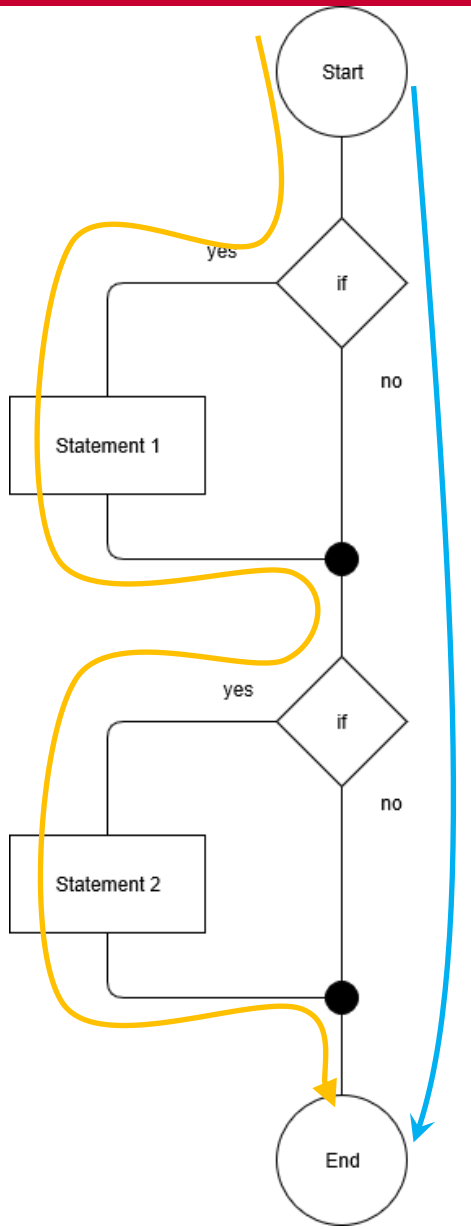
- Condition: (((a == 0) || (b < 5)) && ((x < 6) || (y == 0)))
- Or in general: ((A || B) && (C || D))

- If a, b, x and y are self-contained – A, B, C and D can independently become true and false

- Example
  - **Multiple Condition Coverage**
  - 16 test cases
    - 16 combinations (2^4)

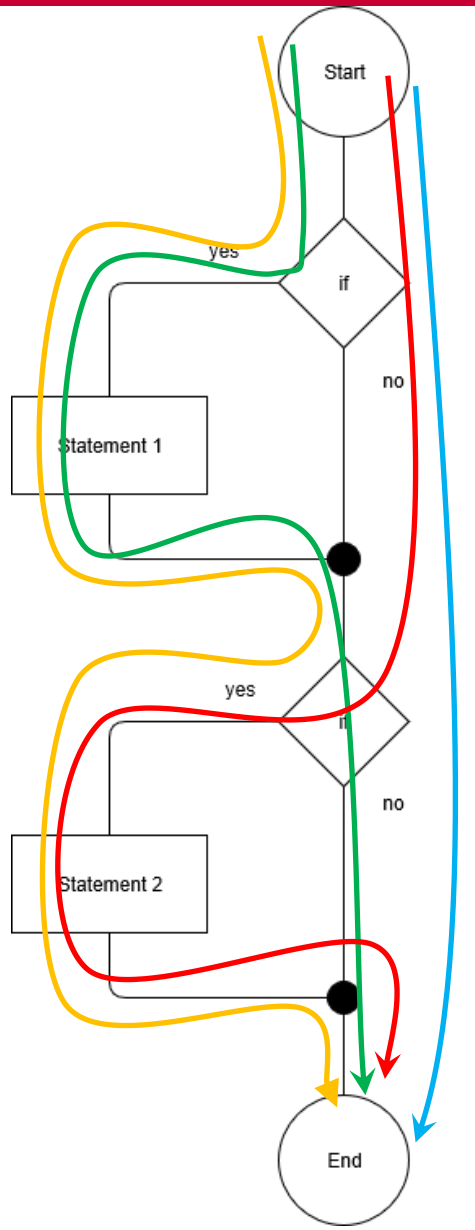
Condition Coverage

- **Example**
  - Condition Coverage
  - Every condition has to be true and false once
- 2 test cases



Condition Coverage

- **Example**
  - Multiple Condition Coverage
  - Combinations of conditions
- 4 test cases



Coverage Based Testing Methods

- How much coverage is sufficent?

80 %

82 %

82,3 %

- Question is about Quality
- Measurements are quantitative

Content

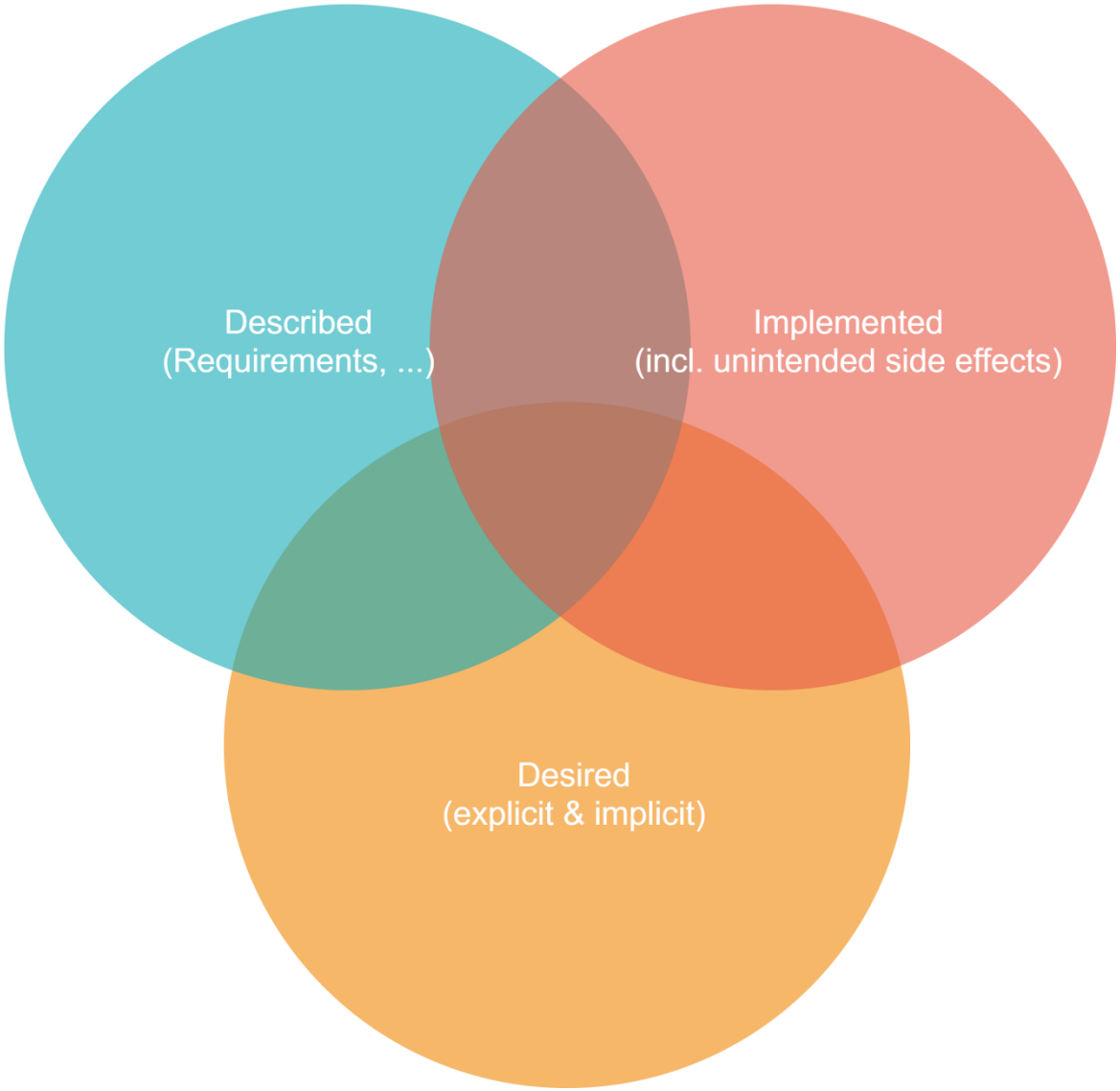
Experience Based Testing

Also known as **Explorative Testing**

usually on top of other test techniques

- Test cases are derived from the testers intuition and their experience
  - Where might be a defect
  - Where were defects in past (or similar projects)
  - Which components were created with less quality assurance
  - …
- Experience based tests needs to be documented

Experience Based Testing



References

- T. Grechenig et al; Softwaretechnik Mit Fallbeispielen aus realen Entwicklungsprojekten, 2010
- ISTQB Foundation Level Syllabus
- A. Spillner and T. Linz; Basiswissen Softwaretest. 5. Aufl. dpunkt Verlag, 2012.
- D. Graham et al; Foundations of Software Testing: ISTQB Certification. Cengage Learning EMEA, 2008.
- IEEE Std 610.12-1990 (R2002), IEEE Standard Glossary of Software Engineering Terminology IEEE, 1990.
- A. Spillner et al; Praxiswissen Softwaretest – Testmanagement: Aus- und Weiterbildung zum Certified Tester, 2014
- House of Test – Ilari Henrik Aegerter