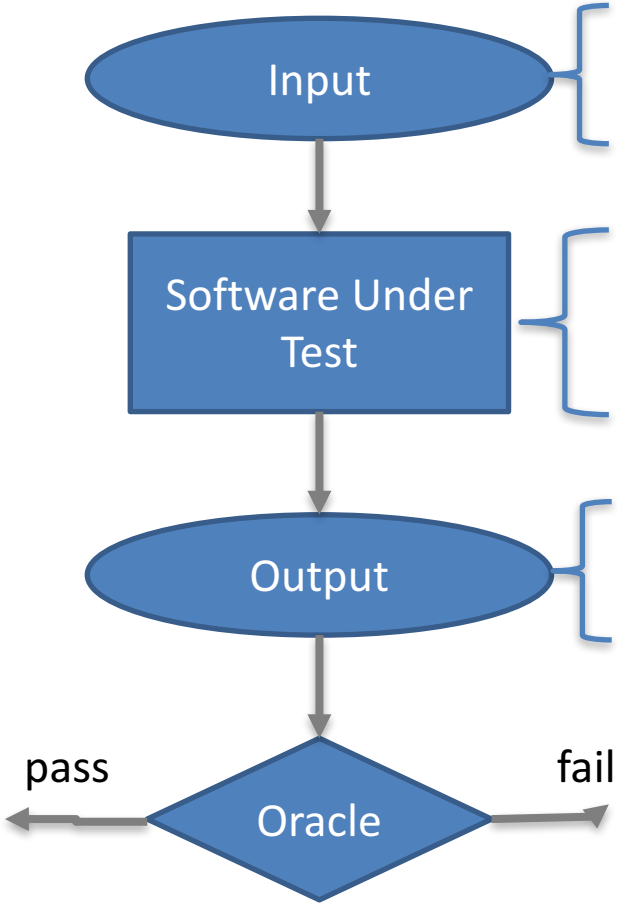
TDDD04: Black box testing

Lena Buffoni lena.buffoni@liu.se

LINKÖPING UNIVERSITY

Black- box testing



Based on requirements or specifications, valid and invalid inputs are chosen

Black box testing can be applied on all levels of the system: Unit, Integration, System, Acceptance.

Actual outputs are compared to expected outputs

! Never can be sure how much of the software has been tested

Types of black box testing

1. **Exhaustive testing**
2. Equivalence class testing (chapter 3)
3. Boundary value analysis (chapter 4)
4. Decision table testing (chapter 5)
5. Pairwise testing (chapter 6)
6. State transition testing (chapter 7)
7. Domain analysis testing (chapter 8)
8. Use case testing (chapter 9)

Example program to determine employability

Age	Employment status
0-16	Don’t hire
16-18	Can hire part-time
18-55	Can hire full-time
55-99	Don’t hire

To test exhaustively we must write tests for all values : 0, 1 … 98, 99

Time consuming and expensive!

Types of black box testing

1. Exhaustive testing
2. **Equivalence class testing (chapter 3)**
3. Boundary value analysis (chapter 4)
4. Decision table testing (chapter 5)
5. Pairwise testing (chapter 6)
6. State transition testing (chapter 7)
7. Domain analysis testing (chapter 8)
8. Use case testing (chapter 9)

Equivalence class (EC) testing

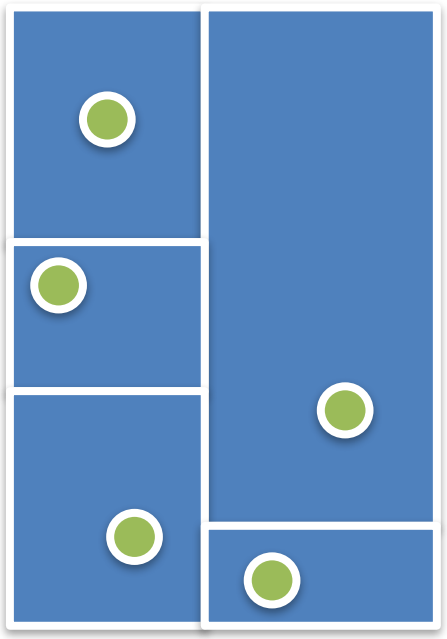
- If one case in the EC detects a defect all other test cases in the EC are likely to detect this defect

Age	Employment status
0- ~~16~~ 15	Don't hire
16-~~18~~ 17	Can hire part-time
18-~~55~~ 54	Can hire full-time
55-99	Don't hire

How many equivalence classes?
Is it enough to test one example per class?

EC testing

- Equivalence Class (EC) testing is a technique used to reduce the number of test cases to a manageable level while still maintaining reasonable test coverage
- Each EC consists of a set of data that is treated the same by the module or that should produce the same result. Any data value within a class is *equivalent*, in terms of testing, to any other value
- Choose one point within each EC to test



Example: NextDate

Problem statement: NextDate is a function of three variables: month (M), day (D) and year (Y). It returns the date of the day after the input date. The month, day and year variables have integer values subject to these conditions:

- C1:1 <= month <=12;
- C2: 1<= day <= 31;
- C3: 1850 <= year <= 2050

Valid ECs	
M1 = {month: 1 <= month <= 12}	M2 = {month: month < 1}
D1={day:1<=day<=31}	M2 = {month: month > 12}
Y1 = {year: 1850 <= year <= 2050}	D2={day:day<1}

What about combinations of M,D & Y?

Guidelines

- If an input condition specifies a *range* of values; identify one valid EC and two invalid EC.
- If an input condition specifies the *number* (e.g., one through 6 owners can be listed for the automobile); identify one valid EC and two invalid EC (no owners; more than 6 owners).
- If an input condition specifies a *set* of input values and there is reason to believe that each is handled differently by the program; identify *a valid EC for each* and *one invalid EC*.
- If an input condition specifies a "*must be*" situation (e.g., first character of the identifier must be a letter); identify *one valid* EC (it is a letter) and *one invalid* EC (it is not a letter)
- If there is any reason to believe that elements in an EC are not handled in an identical manner by the program, split the equivalence class into smaller equivalence classes.

Applicability and limitations

- Most suited to systems in which much of the **input data** takes on values **within ranges or within sets**.
- It makes the **assumption** that data in the same EC is, in fact, processed in the same way by the system. The simplest way to validate this assumption is to ask the programmer about their implementation.
- EC testing is equally applicable at the **unit, integration, system, and acceptance test levels**. All it requires are inputs or outputs that can be partitioned based on the system's requirements.

Types of black box testing

1. Exhaustive testing
2. Equivalence class testing (chapter 3)
3. **Boundary value analysis (chapter 4)**
4. Decision table testing (chapter 5)
5. Pairwise testing (chapter 6)
6. State transition testing (chapter 7)
7. Domain analysis testing (chapter 8)
8. Use case testing (chapter 9)

Boundary value testing

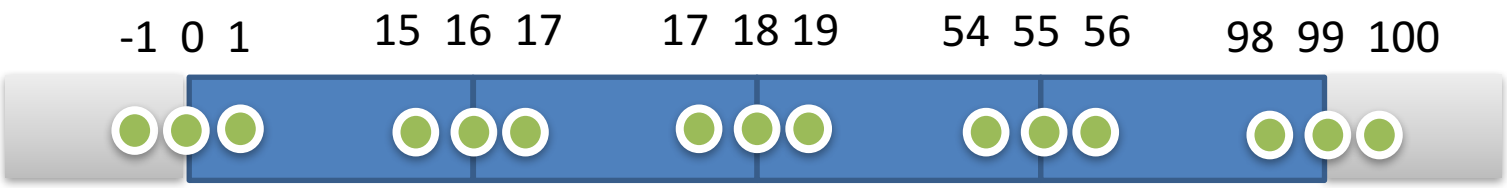
Boundary value testing focuses on the boundaries between equivalence classes, simply because that is where so many defects hide. The defects can be in the requirements or in the code

Method

- Identify the equivalence classes.
- Identify the boundaries of each EC.
- Create test cases for each boundary value by choosing one point **on** the boundary, one point just **below** the boundary, and one point just **above** the boundary.

Example – boundary value testing

Age	Employment status
0-15	Don't hire
16-17	Can hire part-time
18-54	Can hire full-time
55-99	Don't hire



Other test cases?

NextDate – multiple variables

Remember?

- C1:1 <= month <=12;
- C2: 1<= day <= 31;
- C3: 1850 <= year <= 2050

Can we define better equivalence classes?

Valid ECs
M1 = {month has 30 days} M2 = {month has 31 days} M3 ={February}
D1={1<=day<=27}, D2={day=28} D3= {day = 29}, D4= {day = 30}, D5 {day = 31}
Y1 = {year: year is a non-leap year} Y2 = {year: year is a leap year}

What about combinations of M,D & Y?

Case ID	Month	Day	Year	Expected Output
C1	-1	15	1902	Value of month not in range
C2				
C3				
C4				
…				

Applicability and limitations

- Boundary value testing is equally applicable at **the unit, integration, system, and acceptance test levels**. All it requires are inputs that can be partitioned and boundaries that can be identified based on the system's requirements.
- Most suited to systems in which much of the **input data** takes on values **within ranges or within sets**.
- It makes the **assumption** that the program contains no extra boundaries. These can be found through inspection or white-box texting.
- Boundary value testing subsumes equivalence class testing

Types of black box testing

1. Exhaustive testing
2. Equivalence class testing (chapter 3)
3. Boundary value analysis (chapter 4)
4. **Decision table testing (chapter 5)**
5. Pairwise testing (chapter 6)
6. State transition testing (chapter 7)
7. Domain analysis testing (chapter 8)
8. Use case testing (chapter 9)

Decision Table Testing

- An excellent tool to capture certain kinds of **system requirements** and to **document internal system design**. They are used to record complex **business rules** that a system must implement.
- **In addition**, they can serve as a **guide to creating test cases**.

General form:

		Rule 1	Rule 2	…	Rule p
Conditions	Condition 1				
	Condition 2				
	…				
	Condition n				
Actions	Action 1				
	Action 2				
	…				
	Action m				

Example: car insurance discount

		Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	Rule 7	Rule 8
C	Married?	Yes	Yes	Yes	Yes	No	No	No	No
	# years employed	>= 3	>= 3	<3	<3	>= 3	>= 3	<3	<3
	Good student?	Yes	No	Yes	No	Yes	No	yes	No
A	Discount	70	70	50	20	60	60	40	0

Example - simplified

		Rule 1-2	Rule 3	Rule 4	Rule 5-6	Rule 7	Rule 8
C	Married?	Yes	Yes	Yes	No	No	No
	# years employed	>= 3	<3	<3	>= 3	<3	<3
	Good student?	-	Yes	No	-	Yes	No
A	Discount	70	50	20	60	40	0

Definitions

		Rule 1-2	Rule 3	Rule 4	Rule 5-6	Rule 7	Rule 8
C	Married?	Yes	Yes	Yes	No	No	No
	# years employed	>= 3	<3	<3	>= 3	<3	<3
	Good student?	-	Yes	No	-	Yes	No
A	Discount	70	50	20	60	40	0

Limited entry table – conditions are binary only

Extended entry table – conditions may take on multiple values

Don't-care-entry – condition is irrelevant or not applicable; any value is OK

Decision tables are **declarative** – order between entries does not matter

Using decision tables for testing

- **Rules** become **test cases**
- **Conditions** become **inputs**
- **Actions** become **expected results:**

		Test Case 1	Test Case 2	…	Test Case p
Inputs	Condition 1				
	Condition 2				
	…				
	Condition n				
Expected Results	Action 1				
	Action 2				
	…				
	Action m				

Triangle program

The program accepts three integers, a, b, and c as input. The three values are interpreted as representing the lengths of sides of a triangle. The integers a, b, and c must satisfy the following conditions:

a, b, c forms a triangle, i.e.

- a<b+c
- b<a+c
- c<a+b

The program must determine whether the input is not a triangle, an isosceles, a scalene or an equilateral tringle.

Decision table for the triangle program

		Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	Rule 7	Rule 8	Rule 9
C	a,b,c forms a triangle									
	a=b									
	a=c									
	b=c									
A	NaT									
	Isosceles									
	Scalene									
	Equilateral									

Decision table for the triangle program

		Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	Rule 7	Rule 8	Rule 9
C	a,b,c forms a triangle	N	Y	Y	Y	Y	Y	Y	Y	Y
	a=b	_	Y	Y	Y	Y	N	N	N	N
	a=c	_	Y	Y	N	N	Y	Y	N	N
	b=c	_	Y	N	Y	N	Y	N	Y	N
A	NaT									
	Isosceles									
	Scalene									
	Equilateral									

Decision table for the triangle program

		Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	Rule 7	Rule 8	Rule 9
C	a,b,c forms a triangle	N	Y	Y	Y	Y	Y	Y	Y	Y
	a=b	_	Y	Y	Y	Y	N	N	N	N
	a=c	_	Y	Y	N	N	Y	Y	N	N
	b=c	_	Y	N	Y	N	Y	N	Y	N
A	NaT	X		Impossible	Impossible		Impossible			
	Isosceles					X		X	X	
	Scalene									X
	Equilateral		X							

Testing the triangle program

		1	2	3	4	5	6	7	8	9	10	11
C	a<b+c											
	b<c+a											
	c<b+a											
	a=b											
	a=c											
	b=c											
A	NaT											
	Isosceles											
	Scalene											
	Equilateral											

a	b	c	Exp

Testing the triangle program

		1	2	3	4	5	6	7	8	9	10	11
C	a<b+c	N										
	b<c+a	-										
	a<b+a	-										
	a=b	-										
	a=c	-										
	b=c	-										
A	NaT	X										
	Isosceles											
	Scalene											
	Equilateral											

a	b	c	Exp

Testing the triangle program

		1	2	3	4	5	6	7	8	9	10	11
C	a<b+c	N										
	b<c+a	-										
	c<b+a	-										
	a=b	-										
	a=c	-										
	b=c	-										
A	NaT	X										
	Isosceles											
	Scalene											
	Equilateral											

a	b	c	Exp
4	2	1	NaT

Testing the triangle program

		1	2	3	4	5	6	7	8	9	10	11
C	a<b+c	N	Y	Y	T							
	b<c+a	-	N	Y	T							
	c<b+a	-	-	N	T							
	a=b	-	-	-	T							
	a=c	-	-	-	T							
	b=c	-	-	-	T							
A	NaT	X										
	Isosceles											
	Scalene											
	Equilateral											

a	b	c	Exp
4	2	1	NaT

Testing the triangle program

		1	2	3	4	5	6	7	8	9	10	11
C	a<b+c	N	Y	Y	T							
	b<c+a	-	N	Y	T							
	c<b+a	-	-	N	T							
	a=b	-	-	-	T							
	a=c	-	-	-	T							
	b=c	-	-	-	T							
A	NaT	X	X	X								
	Isosceles											
	Scalene											
	Equilateral				X							

a	b	c	Exp
4	2	1	NaT
2	4	1	NaT
1	4	2	NaT
3	3	3	Eq

Testing the triangle program

		1	2	3	4	5	6	7	8	9	10	11
C	a<b+c	N	Y	Y	T	T	T					
	b<c+a	-	N	Y	T	T	T					
	c<b+a	-	-	N	T	T	T					
	a=b	-	-	-	T	T	T					
	a=c	-	-	-	T	F	T					
	b=c	-	-	-	T	T	F					
A	NaT	X	X	X								
	Isosceles											
	Scalene											
	Equilateral				X							

a	b	c	Exp
4	2	1	NaT
2	4	1	NaT
1	4	2	NaT
3	3	3	Eq

Testing the triangle program

		1	2	3	4	5	6	7	8	9	10	11
C	a<b+c	N	Y	Y	T	T	T					
	b<c+a	-	N	Y	T	T	T					
	c<b+a	-	-	N	T	T	T					
	a=b	-	-	-	T	T	T					
	a=c	-	-	-	T	F	T					
	b=c	-	-	-	T	T	F					
A	NaT	X	X	X								
	Isosceles											
	Scalene											
	Equilateral				X							

a	b	c	Exp
4	2	1	NaT
2	4	1	NaT
1	4	2	NaT
3	3	3	Eq

Testing the triangle program

		1	2	3	4	5	6	7	8	9	10	11
C	a<b+c	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	b<c+a	-	N	Y	Y	Y	Y	Y	Y	Y	Y	Y
	c<b+a	-	-	N	Y	Y	Y	Y	Y	Y	Y	Y
	a=b	-	-	-	Y	Y	Y	Y	N	N	N	N
	a=c	-	-	-	Y	N	Y	N	Y	Y	N	N
	b=c	-	-	-	Y	Y	N	N	Y	N	Y	N
A	NaT	X	X	X								
	Isosceles							X		X	X	
	Scalene											X
	Equilateral				X							

a	b	c	Exp
4	2	1	NaT
2	4	1	NaT
1	4	2	NaT
3	3	3	Eq
5	5	3	Iso
5	3	5	Iso
3	5	5	Iso
3	4	5	Sca

How many test cases

- If there are n conditions, there must be 2^n rules
- Each *don't care* makes that rule count double
- If there are m *don't care* entries in a rule, that rule counts as 2^m rules

		1	2	3	4
C	a<b+c	F	T	T	T
	b<c+a	-	F	T	T
	c<b+a	-	-	F	T
	a=b	-	-	-	T
	a=c	-	-	-	T
	b=c	-	-	-	T
Number of rules		?	?	8	1

The NextDate Function

- **Triangle program:** relationships between inputs and correct outputs.
- **NextDate function:** logical relationships among the input variables.

Problem statement: **NextDate** is a function of three variables: *month*, *day* and *year*. It returns the date of the day after the input date. The month, day and year variables have integer values subject to these conditions:

- C1: 1 <= month <=12
- C2: 1<= day <= 31
- C3: 1850 <= year <= 2050

NextDate – Reminder

Valid ECs
M1 = {month has 30 days} M2 = {month has 31 days} M3 ={February}
D1={1<=day<=27}, D2= {day = 28}, D3= {day = 29}, D4= {day = 30}, D5 {day = 31}
Y1 = {year: year is a non-leap year} Y2 = {year: year is a leap year}

What about combinations of M,D & Y?

The NextDate Function

		1	2	3	4					
C	month in M30									
	month in M31									
	February									
	December									
	1<=day<28									
	Day=28									
	Day=29									
	Day=30									
	Day=31									
	Leap year									
A	Increment Year									
	Increment Month									
	Increment Day									
	Reset year									
	Reset month									
	Reset day									

The NextDate Function

		1	2	3	4					
C	month in M30	T	-	-	-					
	month in M31	-	T	-	-					
	February	-	-	T	-					
	December	-	-	-	T					
	1<=day<28	T	T	-	T					
	Day=28	-	-	T	-					
	Day=29	-	-	-	-					
	Day=30	-	-	-	-					
	Day=31	-	-	-	-					
	Leap year	F	F	F	F					
A	Increment Year									
	Increment Month			X						
	Increment Day	X	X		X					
	Reset year									
	Reset month									
	Reset day			X						

Mutually exclusive conditions

		1	2	3
Conditions	C1	T	-	-
	C2	-	T	-
	C3	-	-	T
Actions	A1	X		
	A2		X	
	A3			X

There should be eight rules total

Rule count is 4 for each rule

Gives total of 12 rules! Why?

Which actions apply to T,F,T

- A1 or A3 or both?

Solution :

- Be explicit to avoid overlaps
- Include “impossible” pseudo-action
- Use extended entries if possible

The NextDate Function

		1	2	3	4					
C	month									
	day									
	year									
A	Increment Year									
	Increment Month									
	Increment Day									
	Reset year									
	Reset month									
	Reset day									
	Error									

M31 = { 1,3,5,7,8,10 }; M30 = { 4,6,9,11 }; M2 = { 2 }; M12 = { 12 };D27={1,2,...,27}; D28={28},D29={29 },D30={30},D31={31} YN = *not a leap year*; YL = *leap year*

		1	2	3	4		5	6	7	8	9			
C	month	M30	M30	M30	M31	M31	M12	M12	M2	M2	M2	M2	M2	M2
	day	D27 D28 D29	D30	D31	D27 D28 D29 D30	D31	D27 D28 D29 D30	D31	D30 D31	D27	D28	D28	D29	D29
	year	-	-	-	-	-	-	-	-	-	YL	YN	YL	YN
A	Increment Year							X						
	Increment Month		X			X						X	X	
	Increment Day	X			X		X			X	X			
	Reset year													
	Reset month		X			X		X						
	Reset day							X				X	X	
	Error			X					X					X

Test cases for The NextDate Function

Case ID	Month	Day	Year	Expected output
1-3	April	15	2001	April 16, 2001
4	April	30	2001	May 1, 2001
5	April	31	2001	Error
6-9	January	15	2001	January 16, 2001
10	January	31	2001	February 1, 2001
11-14	December	15	2001	December 16, 2001
15	December	31	2001	January 1, 2002
16	February	15	2001	February 16, 2001
17	February	28	2004	February 29, 2004
18	February	28	2001	March 1, 2001
19	February	29	2004	March 1, 2004
20	February	29	2001	Error
21, 22	February	30	2001	Error

Applicability and Limitations

- Decision table testing can be used whenever the **system must implement complex business rules** when these rules can be represented as a combination of conditions and when these conditions have discrete actions associated with them.

Types of black box testing

1. Exhaustive testing
2. Equivalence class testing (chapter 3)
3. Boundary value analysis (chapter 4)
4. Decision table testing (chapter 5)
5. **Pairwise testing (chapter 6)**
6. State transition testing (chapter 7)
7. Domain analysis testing (chapter 8)
8. Use case testing (chapter 9)

Example a web site

Must work with:

- 8 browsers
- 3 plugins
- 6 client operating systems
- 3 servers
- 3 server operating systems

= 1,296 combinations!

→ Test all combinations?
→ Test a random set of combinations?
→ Test an “intelligently selected” set of combinations?

How does it work?

Test pairs only, and ensure each pair of parameter values exists at least once:

- Latin square
- Orthogonal array
  - Web site example : 1296 test cases to exercise all variable combinations, 64 tests for pair-wise coverage - 95% reduction in test cases
- All-pairs algorithm
  - 48 test cases for all pair coverage

Types of black box testing

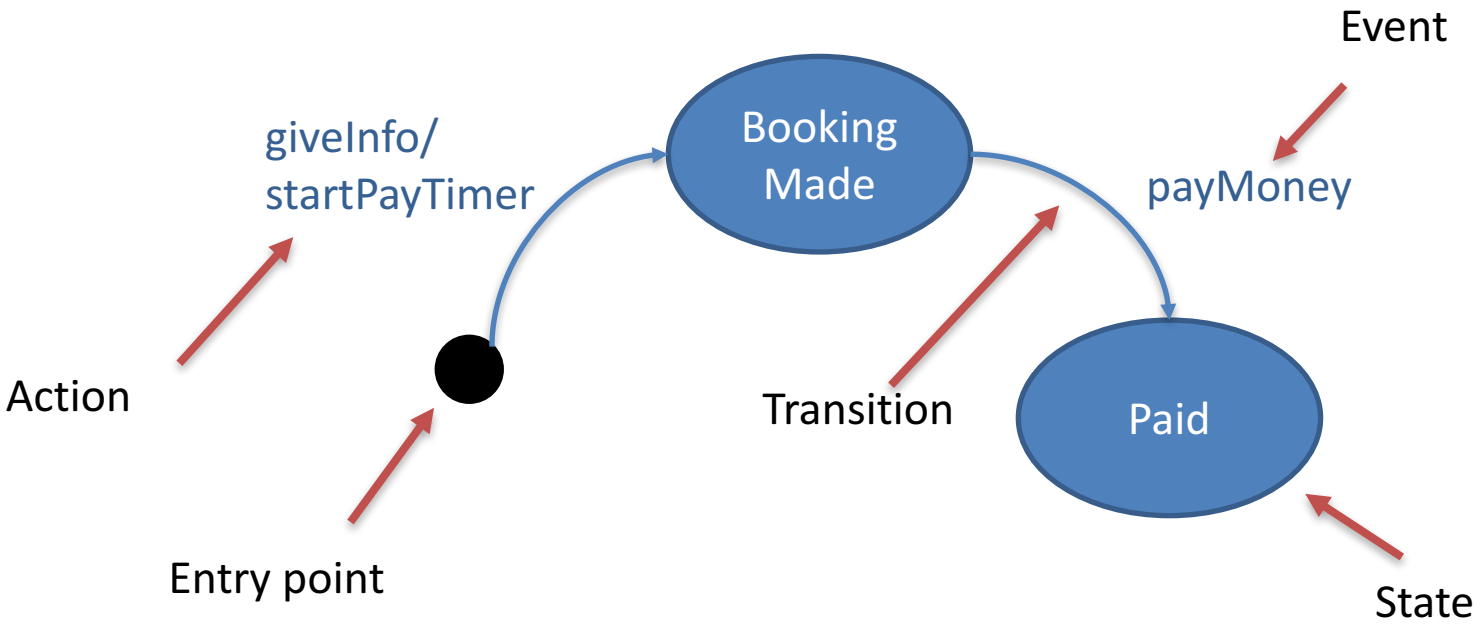
1. Exhaustive testing
2. Equivalence class testing (chapter 3)
3. Boundary value analysis (chapter 4)
4. Decision table testing (chapter 5)
5. Pairwise testing (chapter 6)
6. **State transition testing (chapter 7)**
7. Domain analysis testing (chapter 8)
8. Use case testing (chapter 9)

State-transition testing

- When a system must remember something about what has happened before or when valid and invalid orders of operations exist.
- State-Transition diagrams direct testing efforts by identifying the states, events, actions, and transitions that should be tested.

Flight booking system

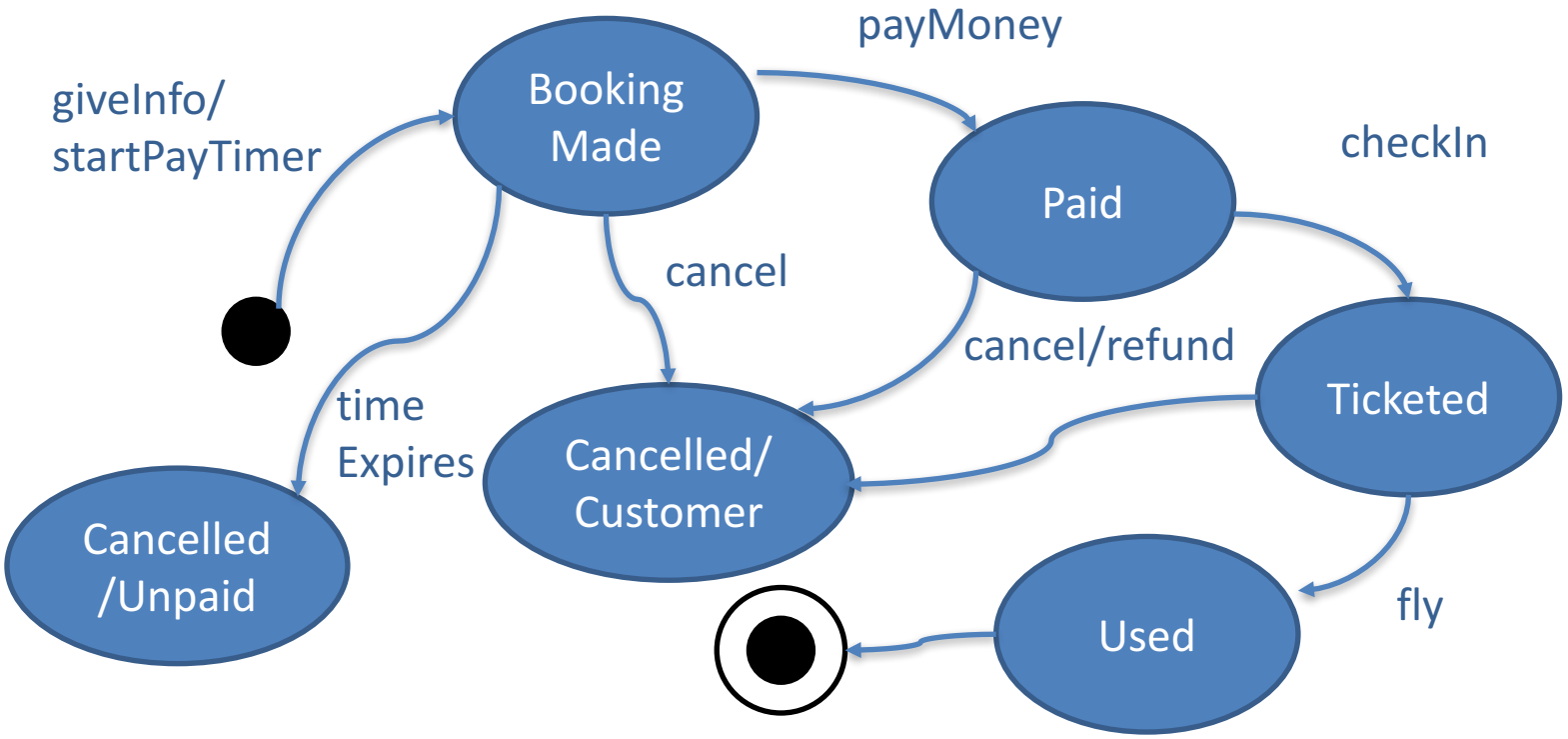
The customer provides some information and makes a booking. He then has a certain amount of time to make the reservation.



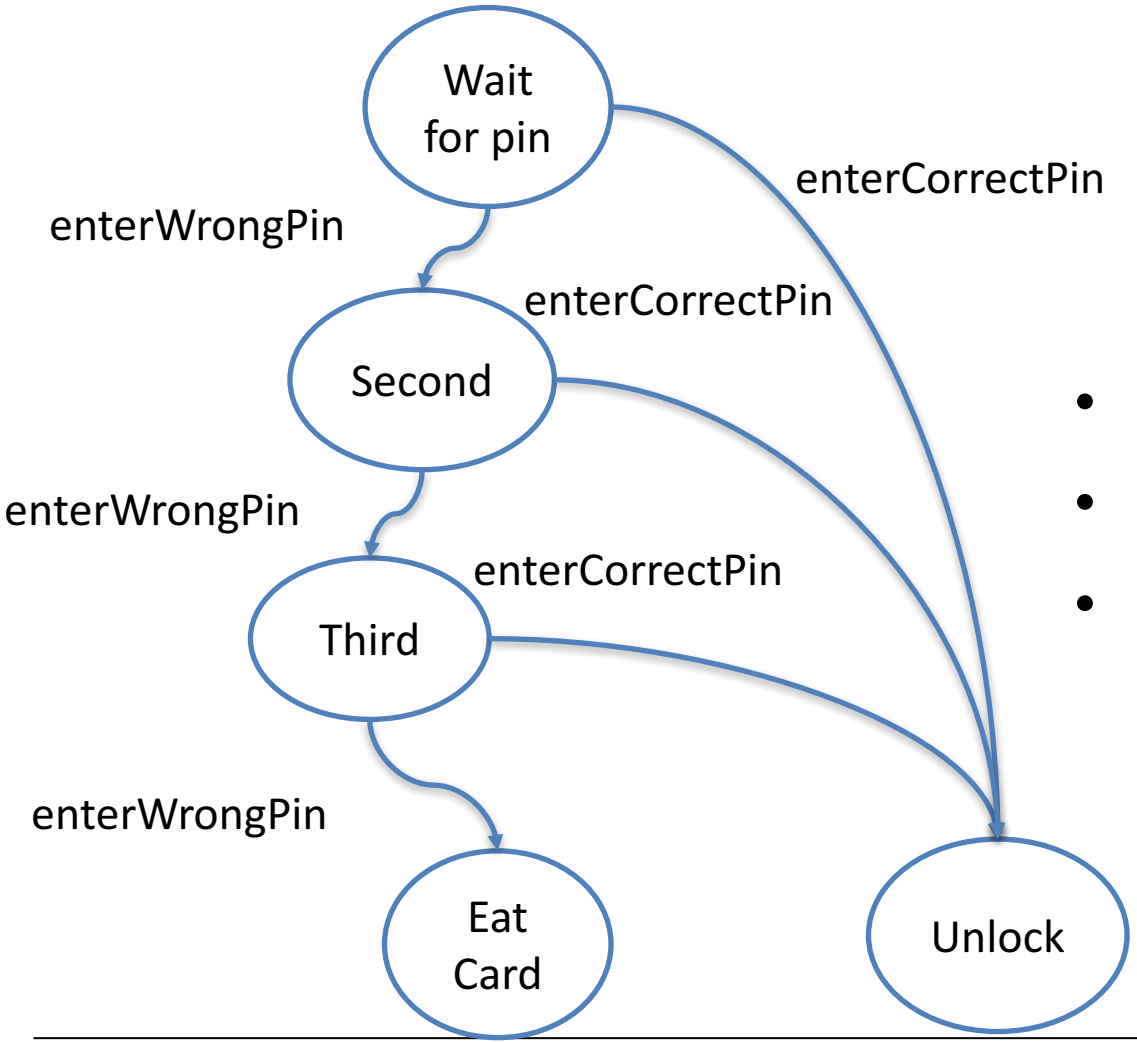
Creating test cases

- Create a set of test cases such that **all states** are visited at least once. May miss important transitions.
- Create a set of test cases such that **all events** are triggered at least once. May miss both states and transitions.
- Create a set of test cases such that all transitions are exercised at least once. Subsumes (includes) all-states and all-events.
  - Stronger: cover *all* possible *pairs* of transitions
  - Even stronger: cover *all* possible *triplets* of transitions
- Create a set of test cases such that all paths are executed at least once. Subsumes all others. Can be infeasible – consider loops.

Flight booking system complete



Authentication system - State transition testing



- Cover all states
- Cover all events
- Cover all transitions

Events	Expected states
…	….

Applicability and Limitations

- Excellent to capture **certain system requirements**.
- Not applicable when the system has no state
- Frequently inapplicable if the system does not need to respond to outside events

Types of black box testing

1. Exhaustive testing
2. Equivalence class testing (chapter 3)
3. Boundary value analysis (chapter 4)
4. Decision table testing (chapter 5)
5. Pairwise testing (chapter 6)
6. State transition testing (chapter 7)
7. **Domain analysis testing (chapter 8)**
8. Use case testing (chapter 9)

Domain analysis testing

- Testing of **multiple variables simultaneously**.
- It builds on and **generalizes equivalence class and boundary value testing**.
- Is usually applied to one input variable or two simple combinations of two variables, based on specifications.

Test cases

- For each relationship (>=, >, <=, or <) choose **one on point** and **one off point**.
- For each strict equality condition (=) choose **one on point** and **two off points**, one slightly less than the conditional value and one slightly greater than the value.

Applicability and Limitations

- Applicable when **multiple variables** should be tested together either for efficiency or because of a logical interaction.
- Best suited to numeric values.

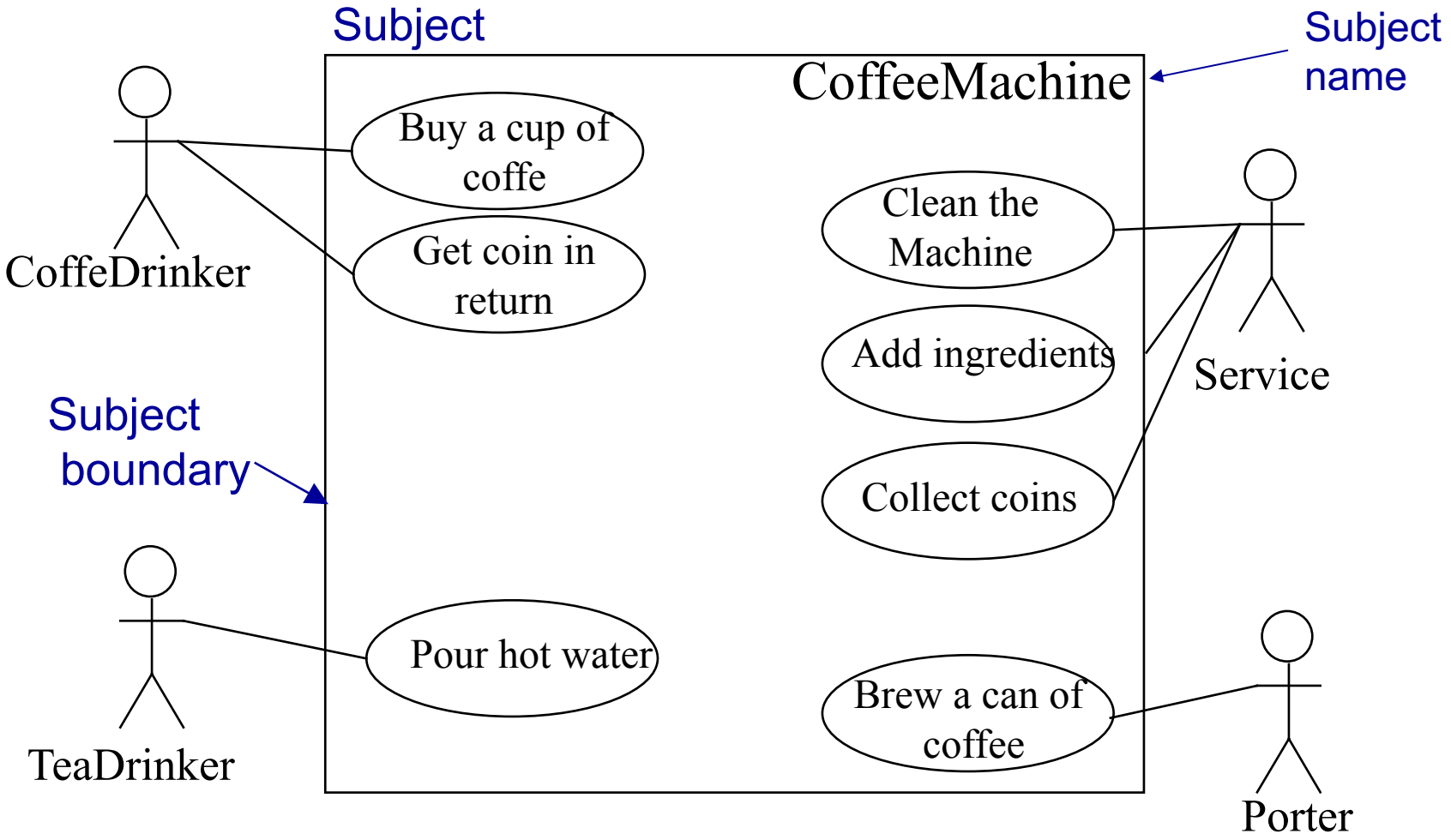
Types of black box testing

1. Exhaustive testing
2. Equivalence class testing (chapter 3)
3. Boundary value analysis (chapter 4)
4. Decision table testing (chapter 5)
5. Pairwise testing (chapter 6)
6. State transition testing (chapter 7)
7. Domain analysis testing (chapter 8)
8. **Use case testing (chapter 9)**

Use-case testing

A use-case is:
"… a particular form or pattern or exemplar of usage, a scenario that begins with some user of the system initiating some transaction of sequence of interrelated events."
Jacobson, m fl 1992: Object-oriented software engineering. Addison-Wesley

Use-case diagram for the coffee-machine



Sufficient information to write a test-case?

Detailed template for describing use cases

Use case component	Description	
Identifier		
Goal in Context	a longer statement of the goal, if needed	
Scope	what system is being considered black-box under design	
Level	Summary, Primary task, Sub-function	
Preconditions	what we expect is already the state of the world	
Success End Condition	the state of the world upon successful completion	
Failed End Condition	the state of the world if goal abandoned	
Primary Actor	a role name for the primary actor, or description	
Trigger	the action upon the system that starts the use case, may be time event	
MAIN SUCCESS SCENARIO	step #	action description
	1	
	2	
	…. For complete template see chapter 9	

Types of black box testing - additional reading

1. Exhaustive testing
2. Equivalence class testing (chapter 3)
3. Boundary value analysis (chapter 4)
4. Decision table testing (chapter 5)
5. Pairwise testing (chapter 6)
6. State transition testing (chapter 7)
7. Domain analysis testing (chapter 8)
8. Use case testing (chapter 9)

Thank you!

Questions?

LINKÖPING UNIVERSITY