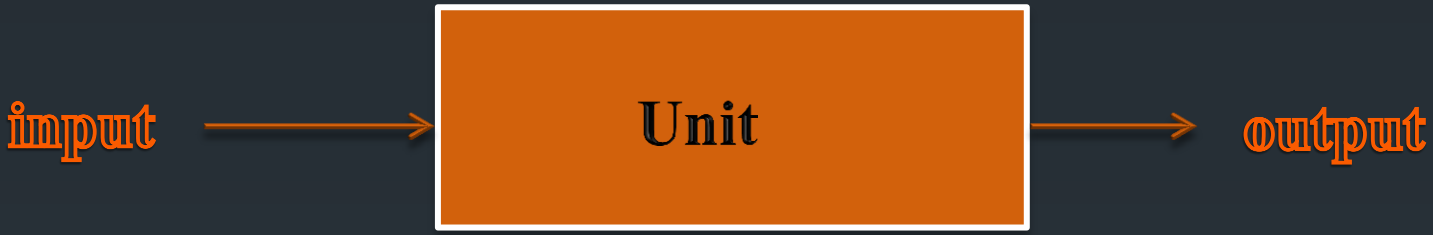
Dynamic Black box testing

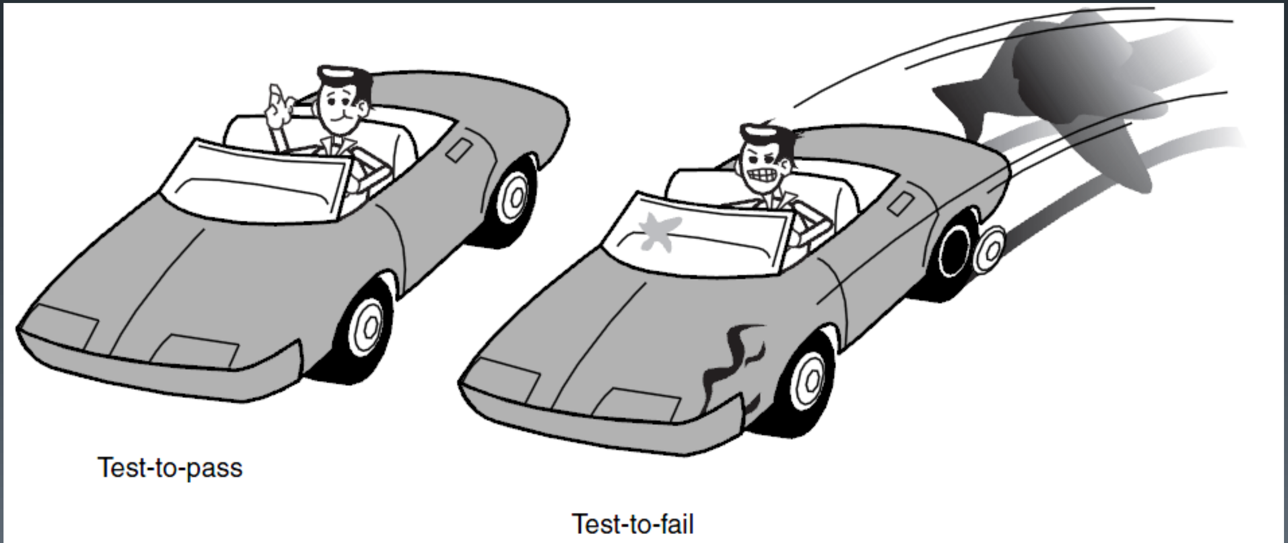
What is Black-Box testing ?

- Testing without having an insight into the details of underlying code
- Testing based on specification
- It's *dynamic because the program is running—you're using it as a customer*
- You're entering inputs, receiving outputs, and checking the results
- ***behavioral testing*** *because you're testing* how the software actually behaves when it's used.



Fundamental approaches to testing software :Test-to-Pass and Test-to-Fail

- Test to pass
  - assure only that the software minimally works.
  - don’t push its capabilities.
  - don’t try to break it
  - assure yourself that the software does what it’s specified to do in ordinary circumstances,
- Test to fail
  - Then attempt to find bugs by trying things that should force them out.
  - Designing and running test cases with the sole purpose of breaking the software is called testing-to-fail or *error-forcing.*



Black box Approaches

- Equivalence Partitioning
- Boundary Value Analysis
- Decision tables
- Cause-Effect Graphing
- State Transition
- Tests derived from functional requirements/Use case testing

Equivalence partitioning

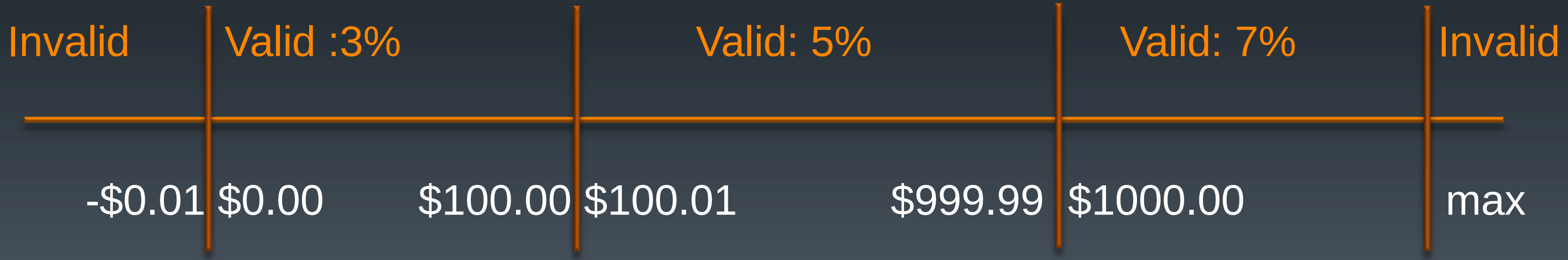
What is Equivalence partitioning?

- Equivalence partitioning is the process of methodically reducing the huge (infinite) set of possible test cases into a much smaller, but still equally effective, set
- Partition input domain in equivalence classes according to specification
- Ideally ECs should be
  - such that each input is a class :test only one condition from each partition
  - disjointed
  - elements of same class mapped to output similarly

RP-5, ISTQB-4

Example

- a savings account in a bank earns a different rate of interest depending on the balance in the account.
- In order to test the software that calculates the interest due, we can identify the ranges of balance values that earn the different rates of interest.
- For example,
  - if a balance is in the range $0 up to $100 has a 3% interest rate,
  - a balance over $100 and up to $1000 has a 5% interest rate
  - balances of $1000 and over have a 7% interest rate,



Strategy

- Separate the domain in equivalent classes
- Choose a representative from each class

Find the equivalence class for absolute value function

Ans

1. `[Integer.MIN_VALUE, -1] [0] [1,Integer.MAX_VALUE]`
2. `[Integer.MIN_VALUE, -1]`: Choose **-34**

   `[0]`: Choose **0**

   `[1,Integer.MAX_VALUE]`: Choose **+42**

Boundary Value Analysis

What is BVA?

- Errors tend to occur near extreme values (*boundaries*)

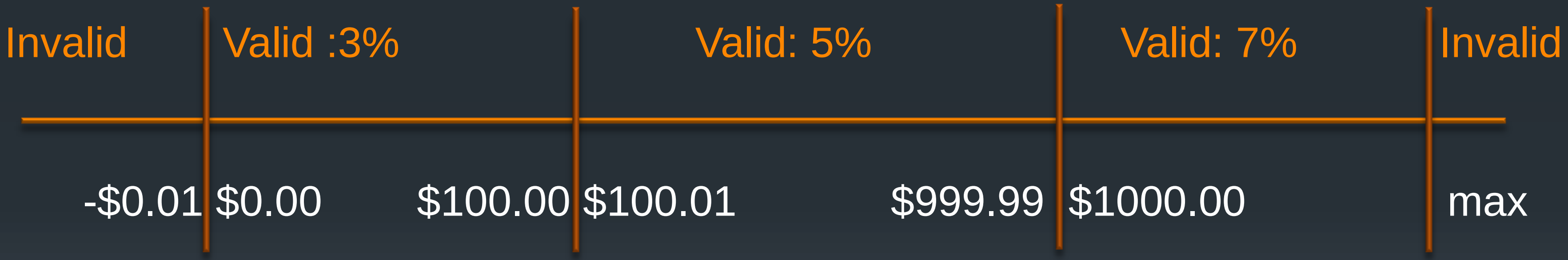
- BVA enhances Equivalence Partitioning by
  - selecting elements just on, and just beyond the borders of each EC
  - deriving test cases by considering ECs from output as well

- Situation at the edge of the planned operational limits of the software
- Data types with boundary conditions
  - Numeric, Character, Position, Quantity, Speed, Location, Size
- Boundary condition characteristics
  - First/Last, Start/Finish, Min/Max, Over/Under, Empty/Full, Slowest/Fastest, Largest/Smallest, Next-To/Farthest-From, Shortest/Longest, Soonest/Latest, Highest/Lowest

RP-5, ISTQB-4

Boundary Value Analysis -Guidelines

- For each boundary condition
  - include boundary value in at least one valid test case
  - include value just beyond boundary in at least one invalid test case
- Use same guidelines for each output condition
  - include test cases with input such that output at boundaries are produced



-$0.01
$0.00
$100.00
$100.01
$999.99
$1000.00
max

RP-5, ISTQB-4

Strategy

- Choose all values on a boundary.
- Choose all values that are “one off” from a boundary.

- Example :
- If a text entry field allows 1 to 255 characters, try entering 1 character and 255 characters as the valid partition. You might also try 254 characters as a valid choice. Enter 0 and 256 characters as the invalid partitions.
- If a program reads and writes to a small capacity chip, try saving a file that’s very small, maybe with one entry. Save a file that’s very large—just at the limit for what that chip holds. Also try saving an empty file and a file that’s too large to fit on the disk.
- If a program allows you to print multiple pages onto a single page, try printing just one (the standard case) and try printing the most pages that it allows. If you can, try printing zero pages and one more than it allows.

Find the BVA for absolute value function

ANS

▪ `[Integer.MIN_VALUE, -1]`:	Choose `-2, -1, min,min+1`
▪ `[0]`:	Choose `0`
▪ `[1,Integer.MAX_VALUE]`:	Choose `1, 2 , max-1,max`

Problem solving example

- The software module in question calculates entrance ticket prices for the Golden Splash Swimming Center.
- The Center’s ticket price depends on four variables: day (weekday, weekend),visitor’s status (OT = one time, M = member), entry hour (6.00–19.00,19.01–24.00) and visitor’s age (up to 16, 16.01–60, 60.01–120).

Equivalence Classes -example

- Check a phone number with the following format:
  - (XXX)   XXX – XXXX

Area Code (optional)   **Prefix**   **Suffix**

- In addition:
  - **Area Code:** 3-digit number [`200 … 999`] except `911`
  - **Prefix:** 3-digit number not beginning with `0` or `1`
  - **Suffix:** 4-digit number

Equivalence Classes -example

- Area Code: 3-digit number [`200 … 999`] except `911`
  - Five Classes:
    - [-∞ …199], [200 … 910], [911], [912 … 999], [1000 … ∞]
- Prefix: 3-digit number not beginning with `0` or `1`
  - Three Classes:
    - [-∞ … 199], [200 … 999], [1000 … ∞]
- Suffix: 4-digit number
  - Three Classes:
    - [-∞ … -1], [0000 … 9999], [10000 … ∞]

Decision table

- Ideal for situations where:
  - combinations of actions taken under varying set of conditions
  - conditions depends on input variables
  - response produced doesn't depend on the order in which input variables are set or evaluated, and
  - response produced doesn't depend on prior input or output

Decision Table - Development

- In order to build decision tables, you need to determine the maximum size of the table, eliminate any impossible situations, inconsistencies, or redundancies, and simplify the table as much as possible. The following steps provide offer some guidelines to developing decision tables:

- Identify decision variables and conditions
- Identify resultant actions to be selected or controlled
- Identify which action should be produced in response to particular combinations of actions
- finding problems and ambiguities in the specification.
- It is a technique that works well in conjunction with equivalence partitioning. The combination of conditions explored may be combinations of equivalence partitions.

Example: loan application

- If you are a new customer opening a credit card account, you will get a 15% discount on all your purchases today. If you are an existing customer and you hold a loyalty card, you get a 10% discount. If you have a coupon, you can get 20% off today (but it can't be used with the 'new customer' discount). Discount amounts are added, if applicable.

Example: loan application

- If you are a **new customer** opening a credit card account, you will get a 15% discount on all your purchases today. If you are an **existing customer** and you **hold a loyalty card**, you get a 10% discount. If you **have a coupon**, you can get 20% off today (but it **can't** be used with the **'new customer'** discount). Discount amounts are **added**, if applicable.

	New customer(15%)	**existing Loyalty Card(10%)**	**Coupon(20%)**	**Discount**
Rule 1	0	0	0	0
Rule 2	0	0	1	20
Rule 3	0	1	0	10
Rule 4	0	1	1	30
Rule 5	1	0	0	15
Rule 6	1	0	1	**20** ←
Rule 7	1	1	0	n/a
Rule 8	1	1	1	n/a

Example: loan application

- you can enter the amount of the monthly repayment or the number of years you want to take to pay it back (the term of the loan). If you enter both, the system will make a compro-mise between the two if they conflict

- Identify conditions:

1. *Repayment amount has been entered*
2. *Term of loan has been entered*

- identify the correct outcome

1. *Process loan amount*
2. *Process term*

Example: loan application

Repayment amount has been entered	y	y	n	n
Term of loan has been entered	y	n	y	n
Process loan amount				
Process term				

Example: decision on non-cash receipts for goods

Repayment amount has been entered	y	y	n	n
Term of loan has been entered	y	n	y	n
Process loan amount	y	y		?
Process term	y		y	?

Example: loan application

Repayment amount has been entered	**y**	**y**	**n**	**n**
Term of loan has been entered	**y**	**n**	**y**	**n**
Process loan amount	**y**	**y**		**?**
Process term	**y**		**y**	**?**
Error message				**y**

Example: loan application

Repayment amount has been entered	**y**	**y**	**n**	**n**
Term of loan has been entered	**y**	**n**	**y**	**n**
Process loan amount		**y**		
Process term			**y**	
Error message	**y**			**y**

Example: loan application

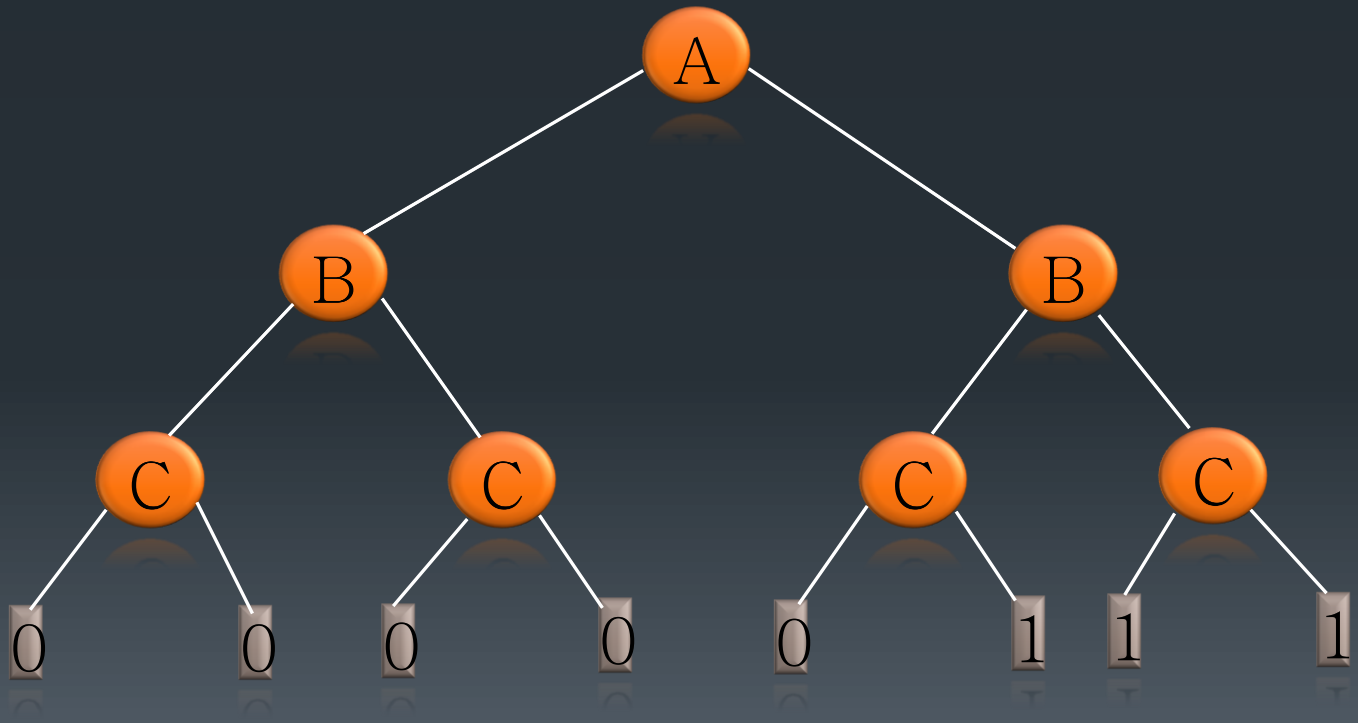
Repayment amount has been entered	y	y	n	n
Term of loan has been entered	y	n	y	n
Process loan amount		y		
Process term			y	
Error message	y			y

Example: loan application

Repayment amount has been entered	y	y	n	n
Term of loan has been entered	y	n	y	n
Result	Error message	Process loan amount	Process term	Error message

Decision tree

Z = A *and* (B *or* C)



Good days to play golf

Day	Outlook	Temp	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Draw Decision tree

State transition testing

State transition testing

- **State transition testing** is used where some aspect of the system can be described in what is called a 'finite state machine'. This simply means that the system can be in a (finite) number of different states, and the transitions from one state to another are determined by the rules of the 'machine'.
- Any system where you get a different output for the same input, depending on what has happened before, is a finite state system.
- A finite state system is often shown as a **state diagram**

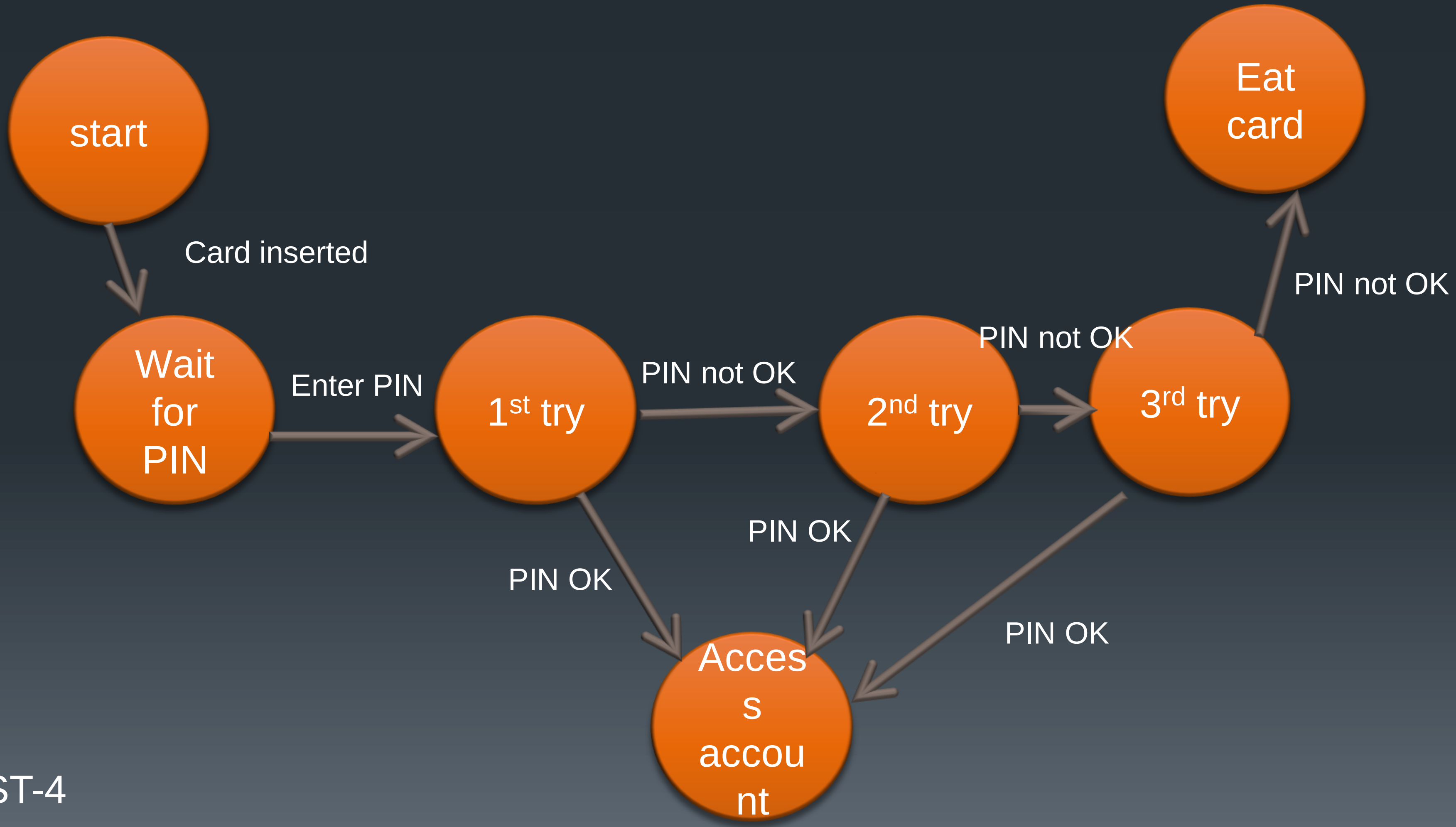
Example

- if you request to withdraw $100 from a bank ATM, you may be given cash. Later you may make exactly the same request but be refused the money (because your balance is insufficient). This later refusal is because the state of your bank account has changed from having sufficient funds to cover the withdrawal to having insufficient funds. The transaction that caused your account to change its state was probably the earlier withdrawal.
- Another example is a word processor. If a document is open, you are able to close it. If no document is open, then 'Close' is not available. After you choose 'Close' once, you cannot choose it again for the same document unless you open that document. A document thus has two states: open and closed.

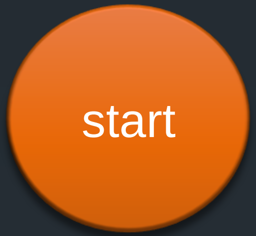
A state transition model has four basic parts:

- the states that the software may occupy (open/closed or funded/insufficient funds);
- the transitions from one state to another (not all transitions are allowed);
- the events that cause a transition (closing a file or withdrawing money);
- the actions that result from a transition (an error message or being given your cash).

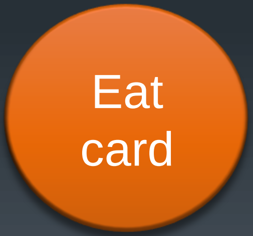
Example : PIN


start
Card inserted
Wait for PIN
Enter PIN
1st try
PIN not OK
2nd try
PIN not OK
3rd try
PIN not OK
Eat card
PIN OK
PIN OK
PIN OK
Acces s accou nt

- State that software may occupy
- Transition from one state to another
- Events that cause transition
- Actions from event



PIN not OK



State diagram -> valid scenario

State table -> invalid scenario

Testing for invalid transitions-State table

		Inset card	enter pin	valid pin	invalid pin
s1	Start state	s2	-	-	-
s2	Wait for PIN	-	s3	-	-
s3	1st try	-	-	s6	s4
s4	2nd try	-	-	s6	s5
s5	3rd try	-	-	s6	s7
s6	access to account		-	?	?
s7	eat card	-	-	-	-

Derive test case for the diagram

Applicability and Limitations

- State-Transition diagrams are excellent tools to capture certain system requirements, namely those that describe states and their associated transitions. These diagrams then can be used to direct our testing efforts by identifying the states, events, and transitions that should be tested.

- State-Transition diagrams are not applicable when the system has no state or does not need to respond to real-time events from outside of the system.

  An example is a payroll program that reads an employee's time record, computes pay, subtracts deductions, saves the record, prints a paycheck, and repeats the process.

Applicability and Limitations

- State-Transition diagrams direct our testing efforts by identifying the states, events, actions, and transitions that should be tested. Together, these define how a system interacts with the outside world, the events it processes, and the valid and invalid order of these events.
- A state-transition diagram is not the only way to document system behaviour. They may be easier to comprehend, but state-transition tables may be easier to use in a complete and systematic manner.
- The generally recommended level of testing using state-transition diagrams is to create a set of test cases such that all transitions are exercised at least once under test.
- In high-risk systems, you may want to create even more test cases, approaching all paths if possible.