DataArt

DataArt

Test design techniques

Irina Kuznetsova
Senior QA, QA Leader
DataArt

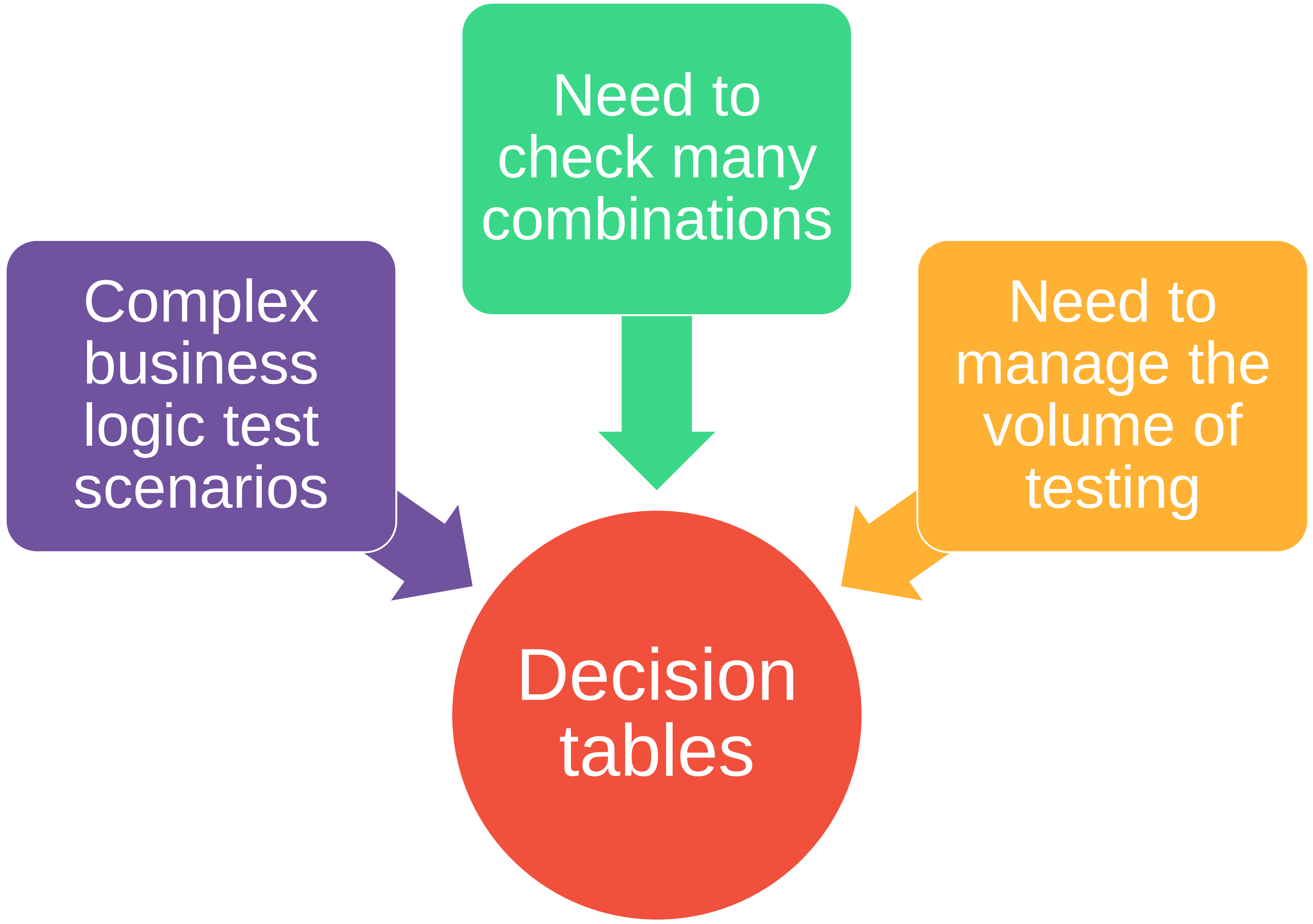
Content

- ✓ Decision tables testing
- ✓ State transition testing
- ✓ Use case testing
- ✓ How to choose the best technique?

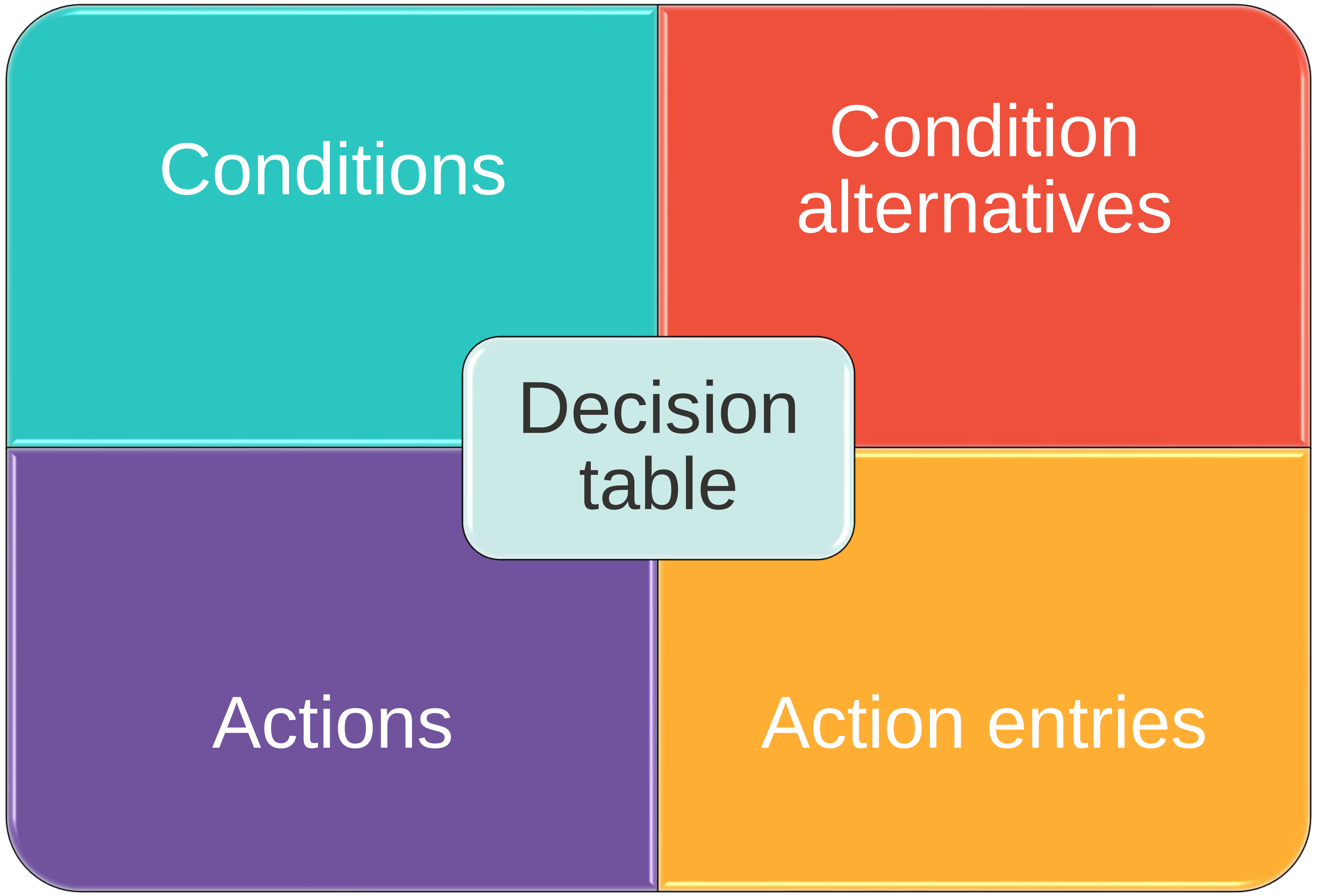
Pair testing gone wrong
ISTQB BASIC LEVEL
"Not so fast Michael! It says here that you should write the test case before you can execute it."

Decision tables testing

Base ideas


Need to check many combinations
Complex business logic test scenarios
Need to manage the volume of testing
Decision tables

Decision table content



- ✓ Each condition corresponds to a variable, relation or predicate
- ✓ Possible values of conditions are listed among the condition alternatives
- ✓ Each action is a procedure or operation to perform
- ✓ The entries specify whether (or in what order) the action is to be performed

The simplest example – discount table

A company sells merchandise to wholesale and retail outlets. Wholesale customers receive a two percent discount on all orders. The company also encourages both wholesale and retail customers to pay cash on delivery by offering an additional two percent discount for this method of payment. Another two percent discount is given on orders of 50 units and tree persent discount is given on orders of 100 or more units.

The simplest example – discount table

Customer type
- Wholesale
- Retail

Customer paid cash
- Yes
- No

Number of units purchased
- >=100
- >=50
- <50

Actions
- No discount
- +2% discount
- +2% discount
- +2% discount
- +3% discount

Number of rules: 2 values * 2 values * 3 values = 12 rules.

The simplest example – discount table

Is a customer wholesale
- Yes
- No

Customer paid cash
- Yes
- No

Number of units purchased
- >=100
- >=50
- <50

Actions
- No discount
- +2% discount
- +2% discount
- +2% discount
- +3% discount

Number of rules: 2 values * 2 values * 3 values = 12 rules.

The simplest example – discount table

Wholesale	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No	No	No
Cash	Yes	Yes	Yes	No	No	No	Yes	Yes	Yes	No	No	No
Units	<50	>=50	>=100	<50	>=50	>=100	<50	>=50	>=100	<50	>=50	>=100

The simplest example – discount table

Wholesale	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No	No	No
Cash	Yes	Yes	Yes	No	No	No	Yes	Yes	Yes	No	No	No
Units	<50	>=50	>=100	<50	>=50	>=100	<50	>=50	>=100	<50	>=50	>=100
No												
+2%												
+2%												
+2%												
+3%												

The simplest example – discount table

Wholesale	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No	No	No
Cash	Yes	Yes	Yes	No	No	No	Yes	Yes	Yes	No	No	No
Units	<50	>=50	>=100	<50	>=50	>=100	<50	>=50	>=100	<50	>=50	>=100
No										X		
+2%	X	X	X	X	X	X						
+2%	X	X	X				X	X	X			
+2%		X			X			X			X	
+3%			X			X			X			X

The simplest example – discount table

Rule

Wholesale	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No	No	No
Cash	Yes	Yes	Yes	No	No	No	Yes	Yes	Yes	No	No	No
Units	<50	>=50	>=100	<50	>=50	>=100	<50	>=50	>=100	<50	>=50	>=100
No										X		
+2%	X	X	X	X	X	X						
+2%	X	X	X				X	X	X			
+2%		X			X			X			X	
+3%			X			X			X			X

The simplest example – discount table

Wholesale	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No	No	No
Cash	Yes	Yes	Yes	No	No	No	Yes	Yes	Yes	No	No	No
Units	<50	>=50	>=100	<50	>=50	>=100	<50	>=50	>=100	<50	>=50	>=100
No										X		
2%				X			X				X	
3%												X
4%	X				X			X				
5%						X			X			
6%		X										
7%			X									

Decision table methodology

Determine conditions

Determine values for each condition

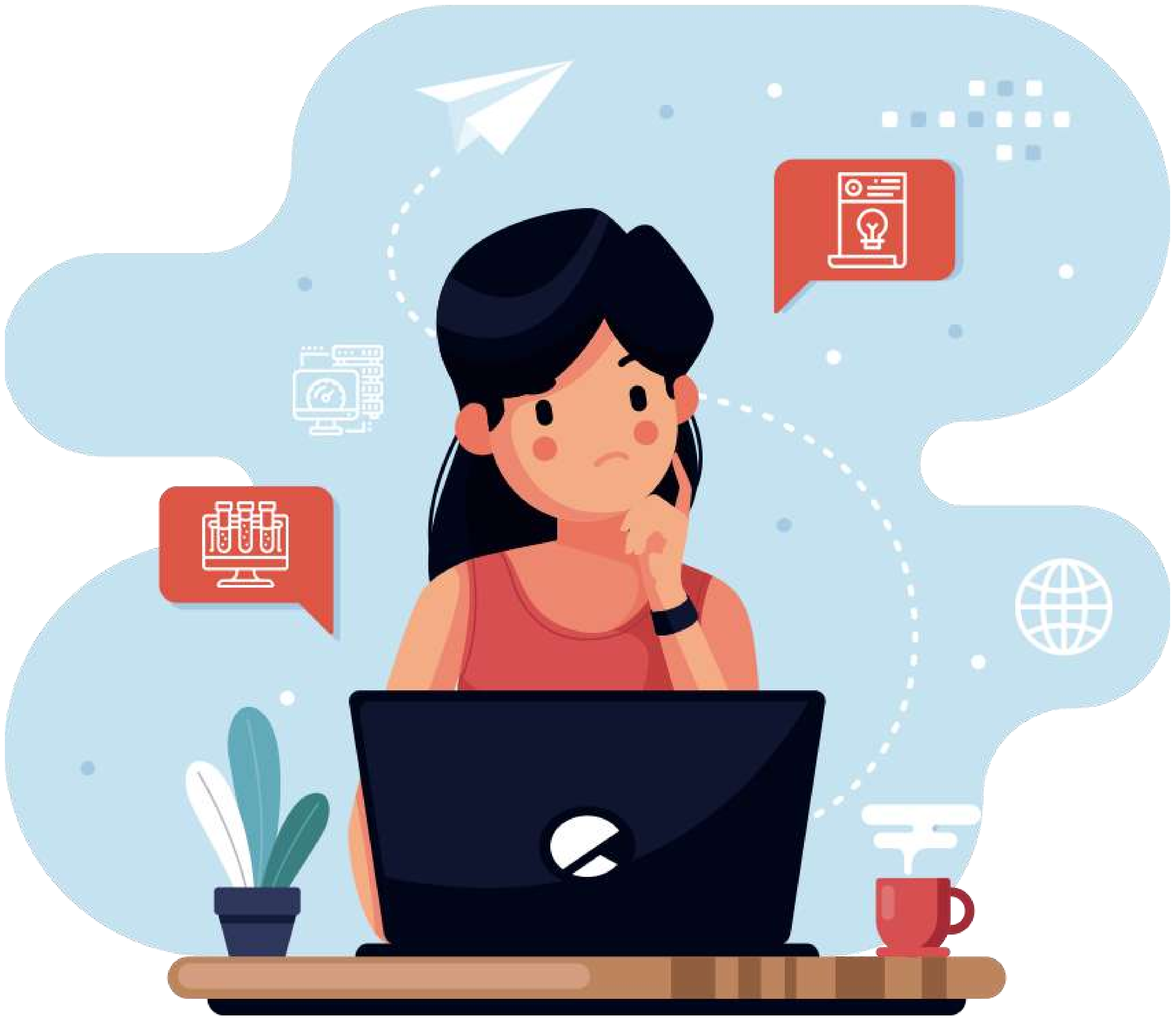
Determine actions

Encode possible rules

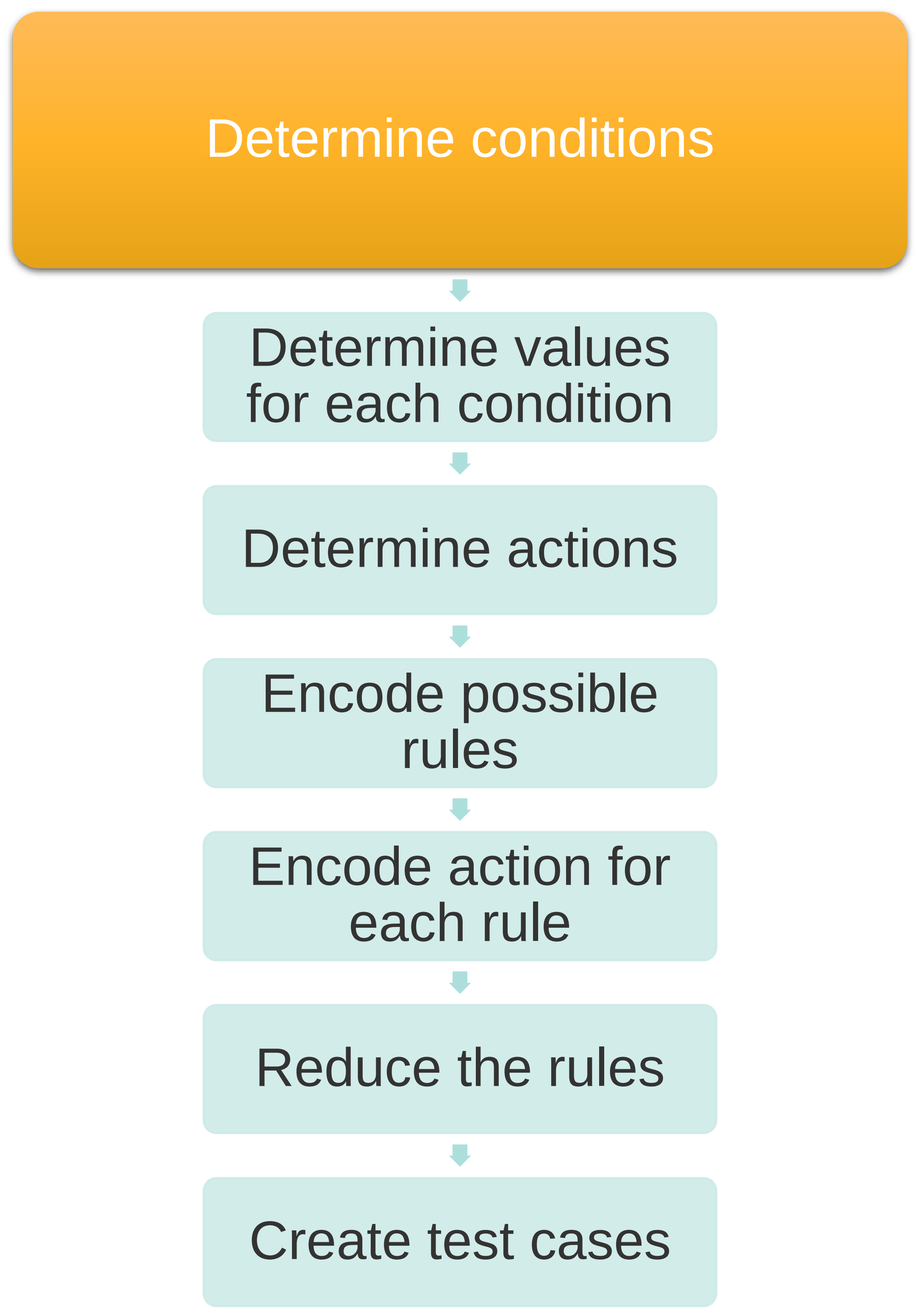
Encode action for each rule

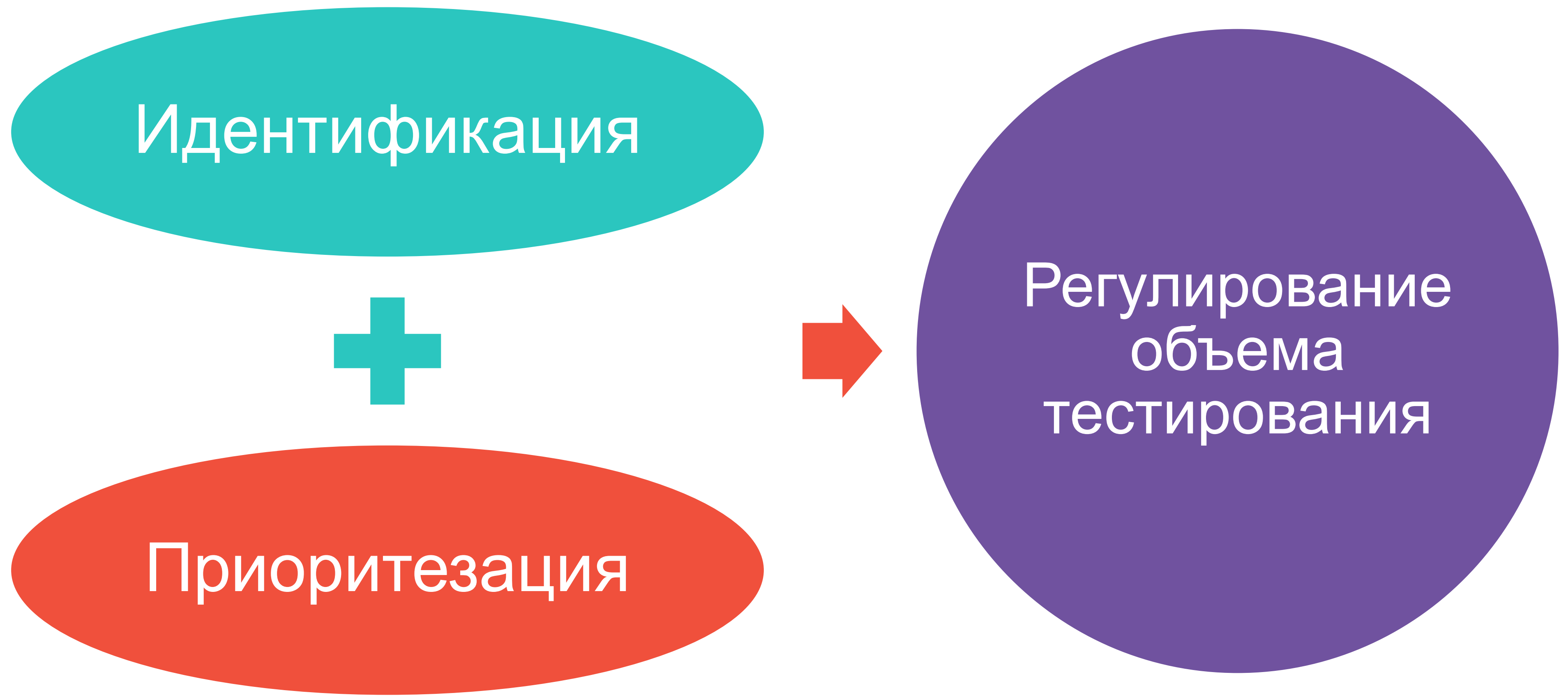
Reduce the rules

Create test cases

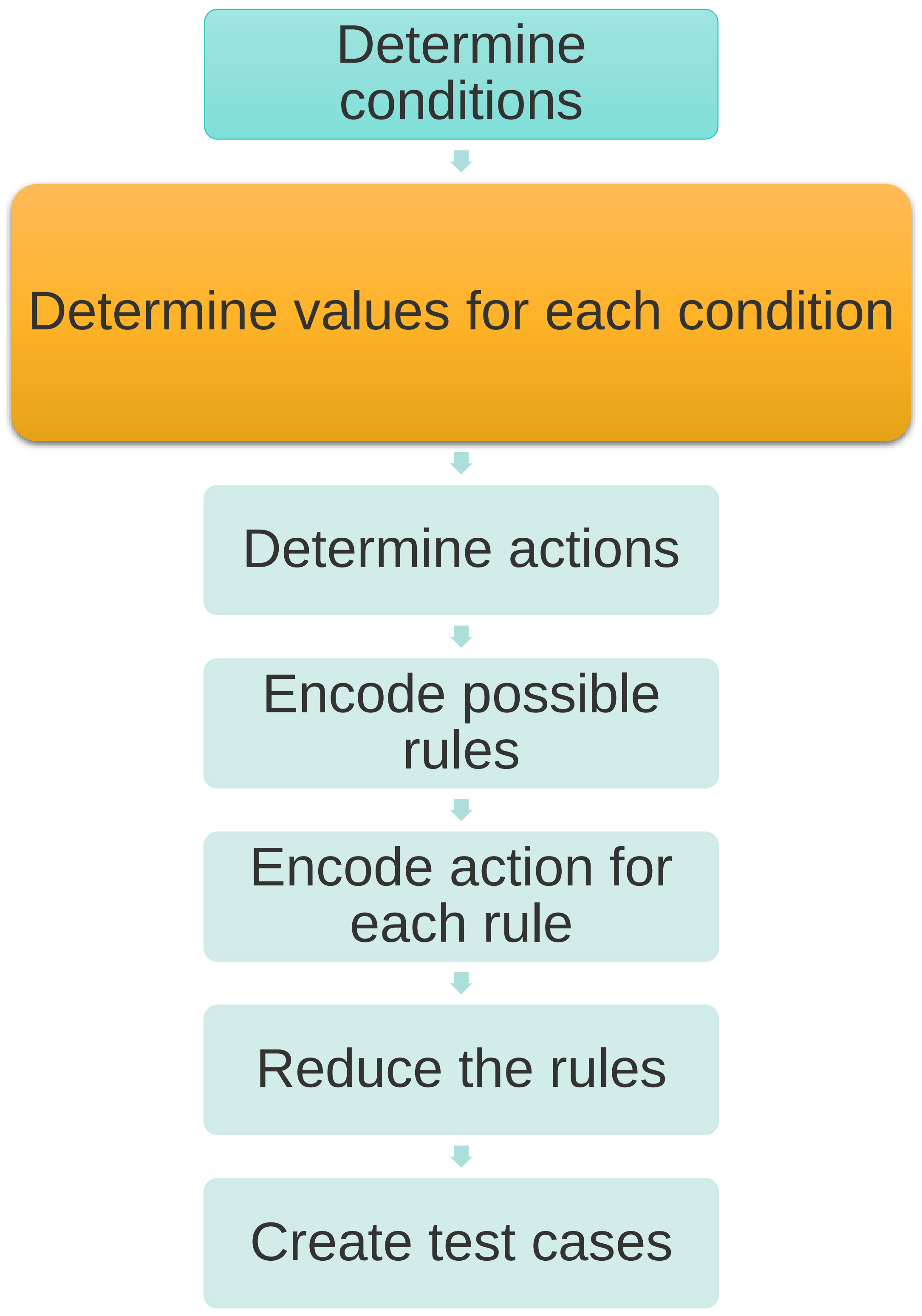


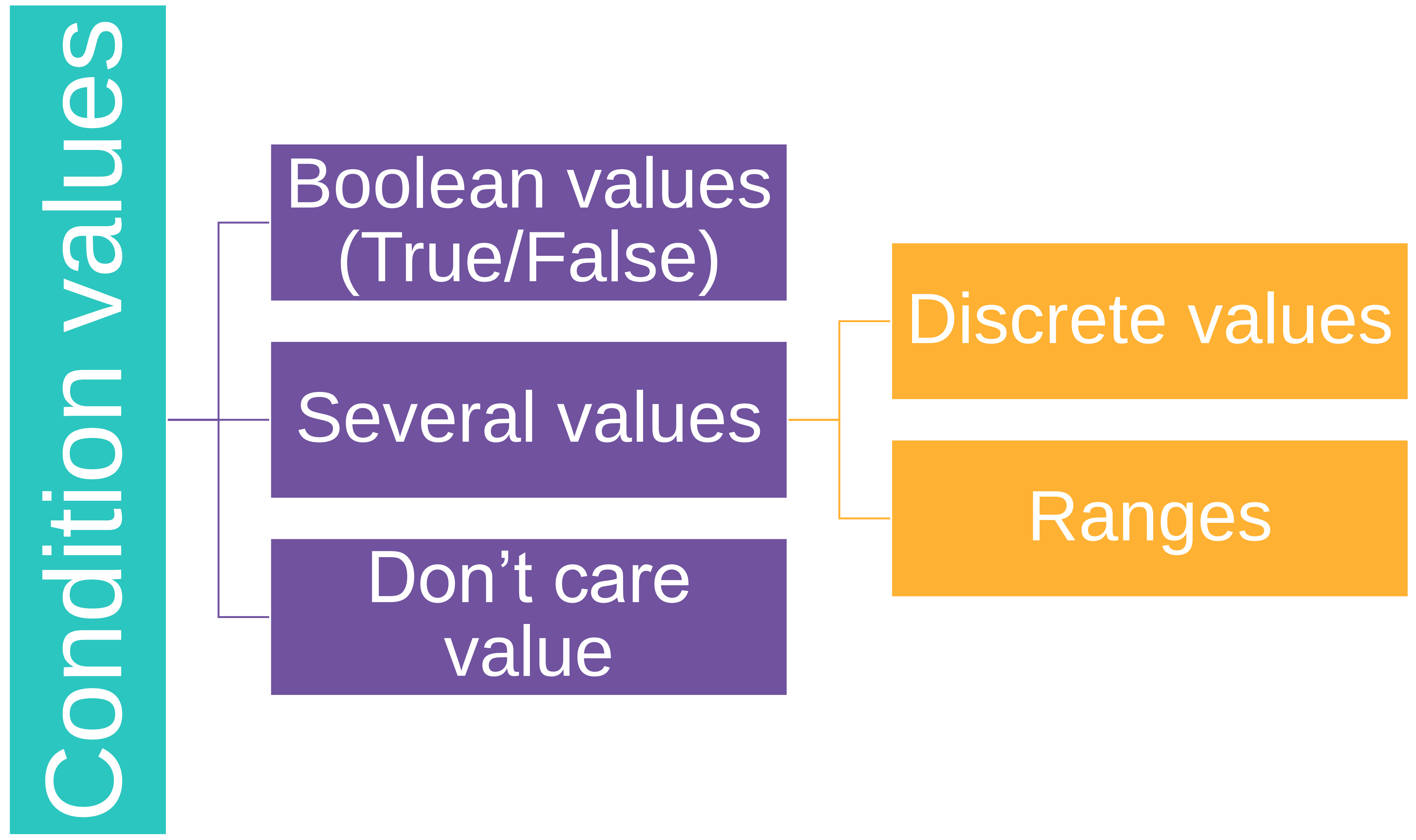
Decision table methodology



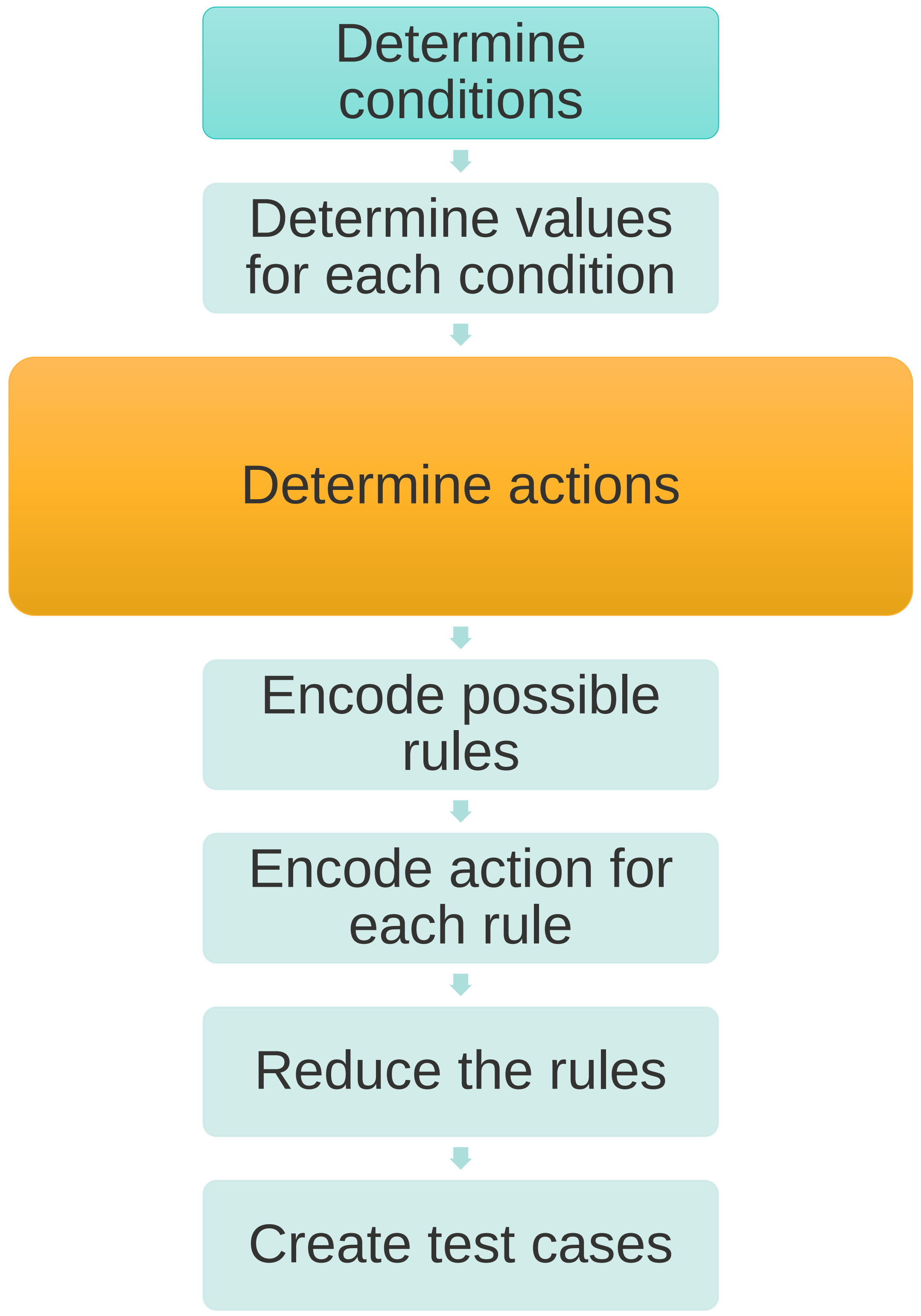


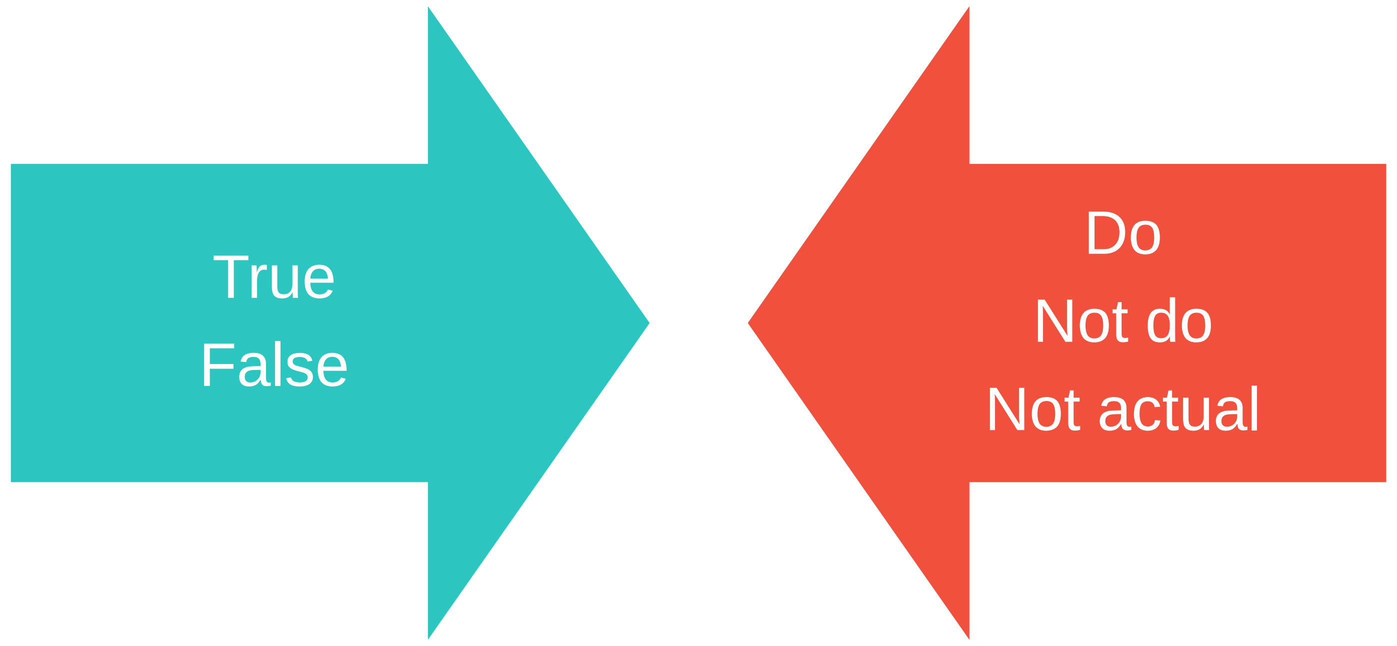
Decision table methodology





Decision table methodology

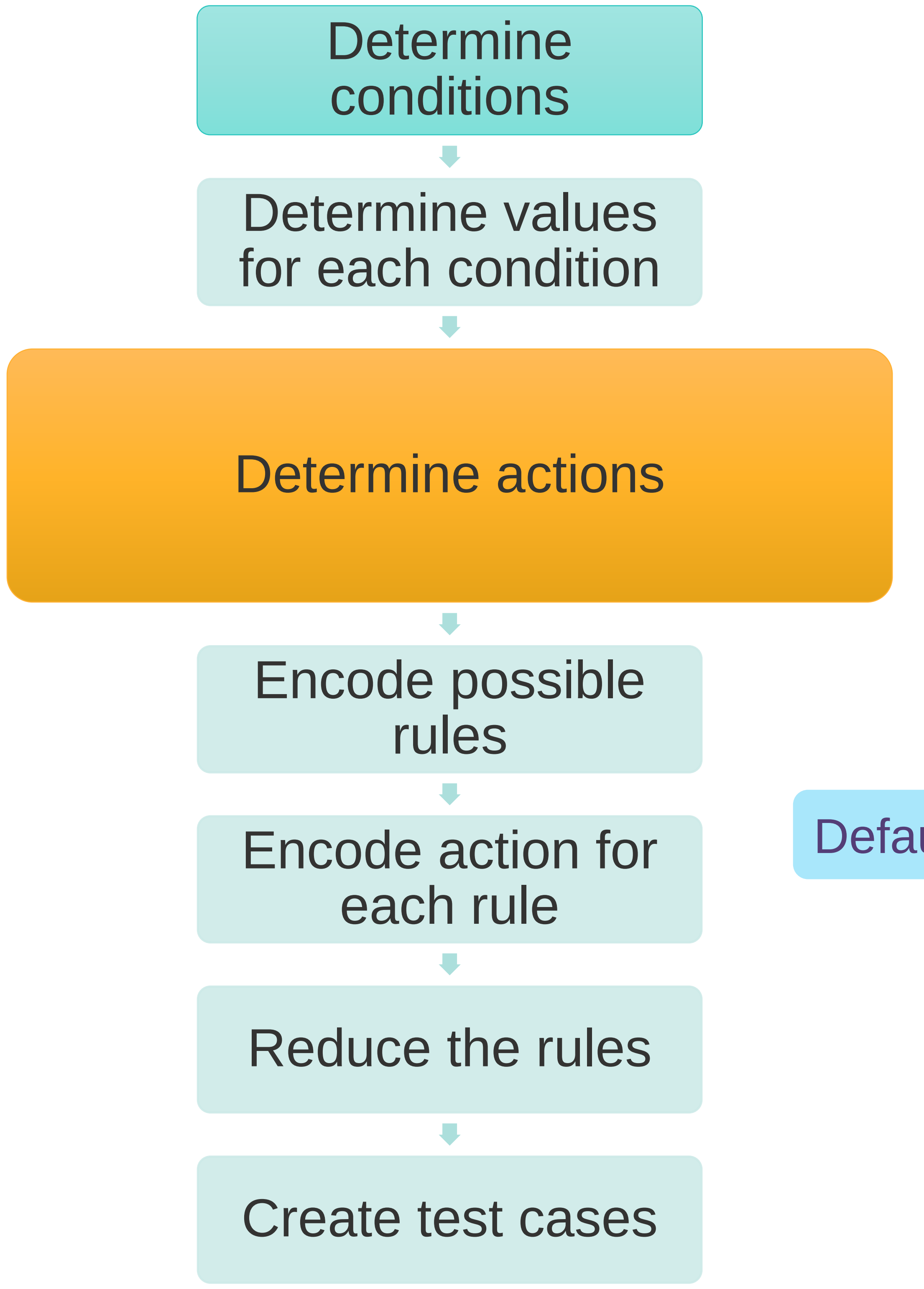


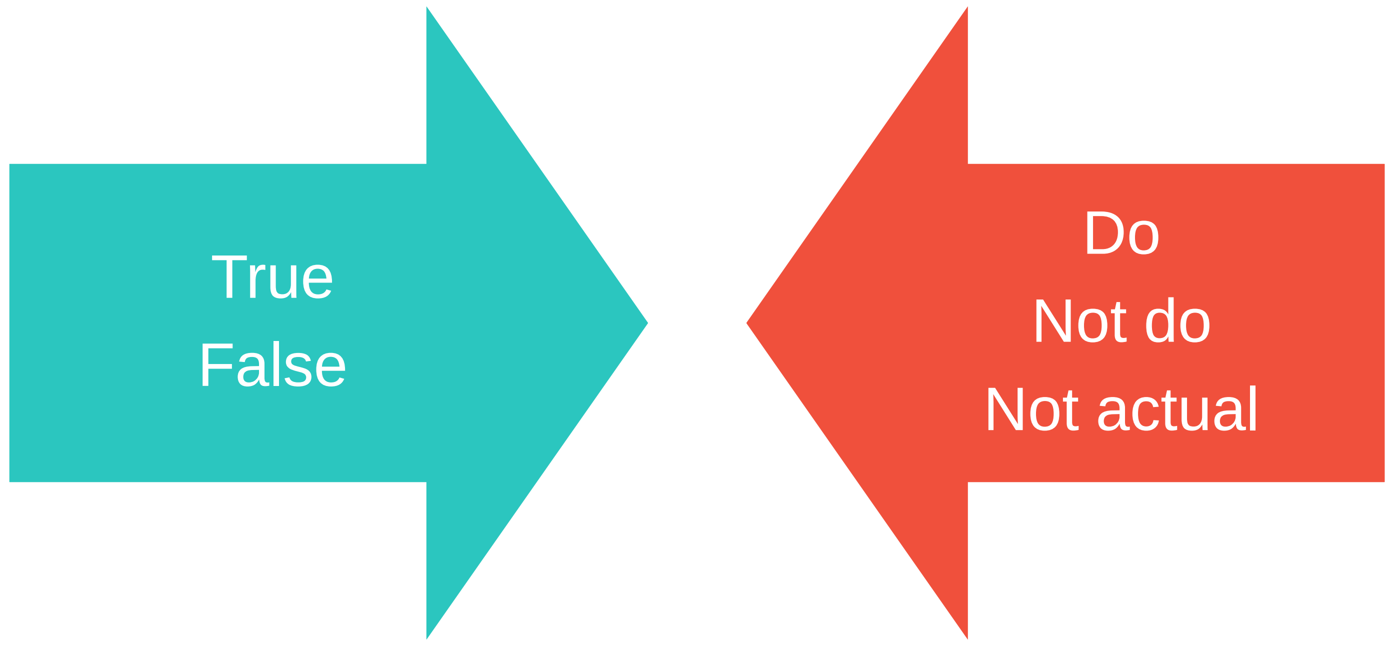


Employee type	S	S	S	S	M	M	M	M	J	J	J	J
Hours worked >40	Y	Y	N	N	Y	Y	N	N	Y	Y	N	N
Overtimes agreed	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N
Pay base salary	+	+	+	+	+	+	+	+	+	+	+	+
Pay for overtime	+	+	*	*	+	–	*	*	+	–	*	*
Pay bonus	+	–	+	–	+	–	+	–	+	–	+	–

*Project salary calculation
All employees who receive overtime approval receive bonuses, as their tasks are critical to the release.
All employees with the Senior grade receive payment for overtime. The rest are paid for overtime only if the overtime has been approved by the team leader.

Decision table methodology





	Employee type	S	S	S	S	M	M	M	M	J	J	J	J
	Hours worked >40	Y	Y	N	N	Y	Y	N	N	Y	Y	N	N
	Overtimes agreed	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N
Default Action	Pay base salary	+	+	+	+	+	+	+	+	+	+	+	+
	Pay for overtime	+	+	*	*	+	–	*	*	+	–	*	*
	Pay bonus	+	–	+	–	+	–	+	–	+	–	+	–

Decision table methodology

1. Determine conditions
2. Determine values for each condition
3. Determine actions
4. **Encode possible rules**
5. Encode action for each rule
6. Reduce the rules
7. Create test cases

$2^4 = 16?$

	PIN															
User inserts valid card																
User enters valid PIN																
3 invalid PINs																
Sufficient balance for the request																
Reject card																
Prompt to reenter																
Eat card																
Dispense cash																

Decision table methodology

DataArt

- Determine conditions
- Determine values for each condition
- Determine actions
- **Encode possible rules**
- Encode action for each rule
- Reduce the rules
- Create test cases

PIN					
User inserts valid card	N	Y	Y	Y	Y
User enters valid PIN	N	N	N	Y	Y
3 invalid PINs	N	N	Y	N	N
Sufficient balance for the request	N	N	N	N	Y
Reject card	+	-	-	-	-
Prompt to reenter	-	+	+	-	-
Eat card	-	-	+	-	-
Dispense cash	*	*	*	-	+

Decision table methodology

- Determine conditions
- Determine values for each condition
- Determine actions
- Encode possible rules
- **Encode action for each rule**
- Reduce the rules
- Create test cases

PIN					
User inserts valid card	N	Y	Y	Y	Y
User enters valid PIN	N	N	N	Y	Y
3 invalid PINs	N	N	Y	N	N
Sufficient balance for the request	N	N	N	N	Y
Reject card	+	-	-	-	-
Prompt to reenter	-	+	+	-	-
Eat card	-	-	+	-	-
Dispense cash	*	*	*	-	+

Decision table methodology

Determine conditions

Determine values for each condition

Determine actions

Encode possible rules

Encode action for each rule

Reduce the rules

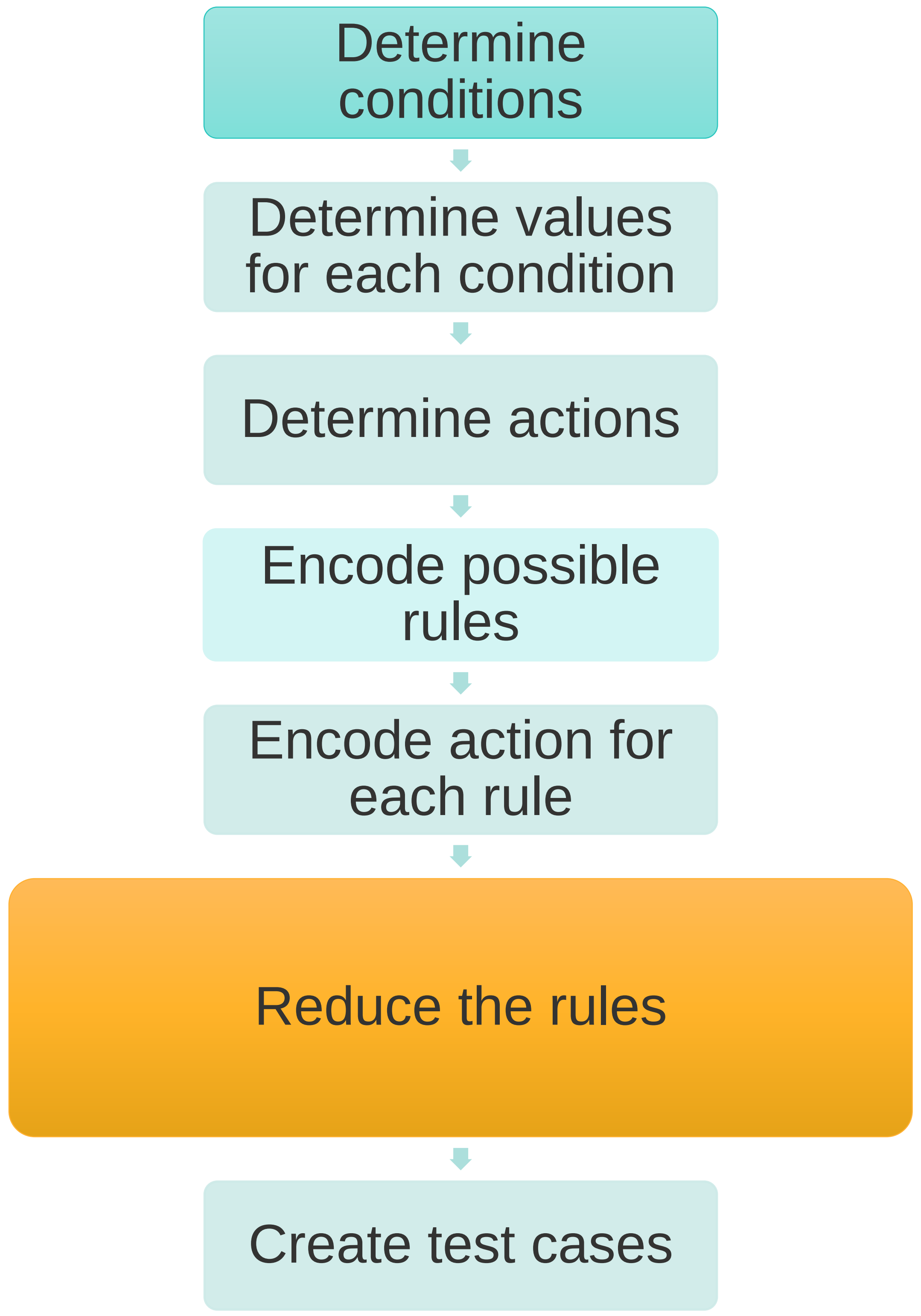
Create test cases

$2^6 = 64$?

Triangle problem											
$a < b + c$											
$b < a + c$											
$c < a + b$											
$a = b$											
$a = c$											
$b = c$											
Not a triangle											
Scalene											
Isosceles											
Equilateral											
Impossible											

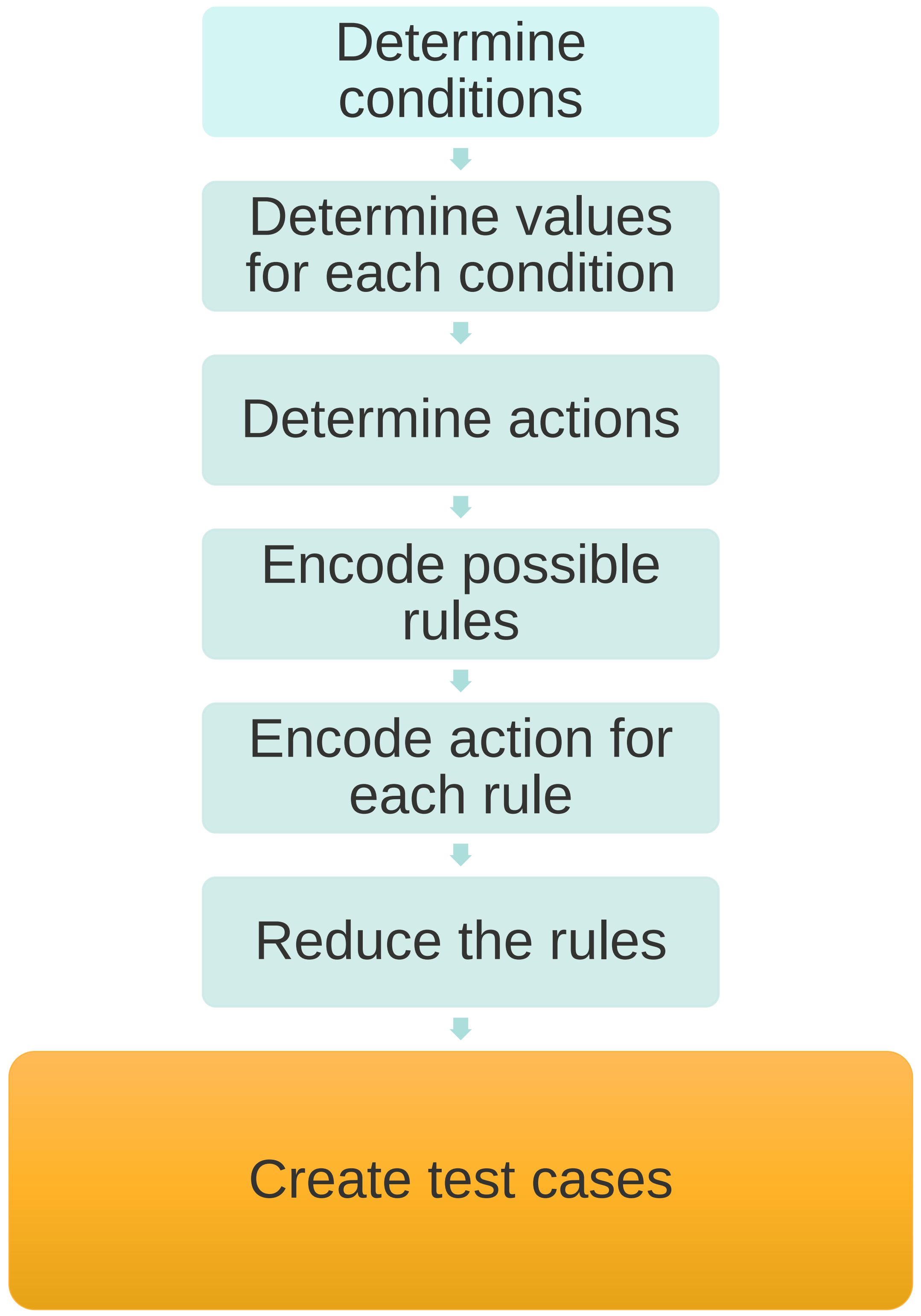
Decision table methodology

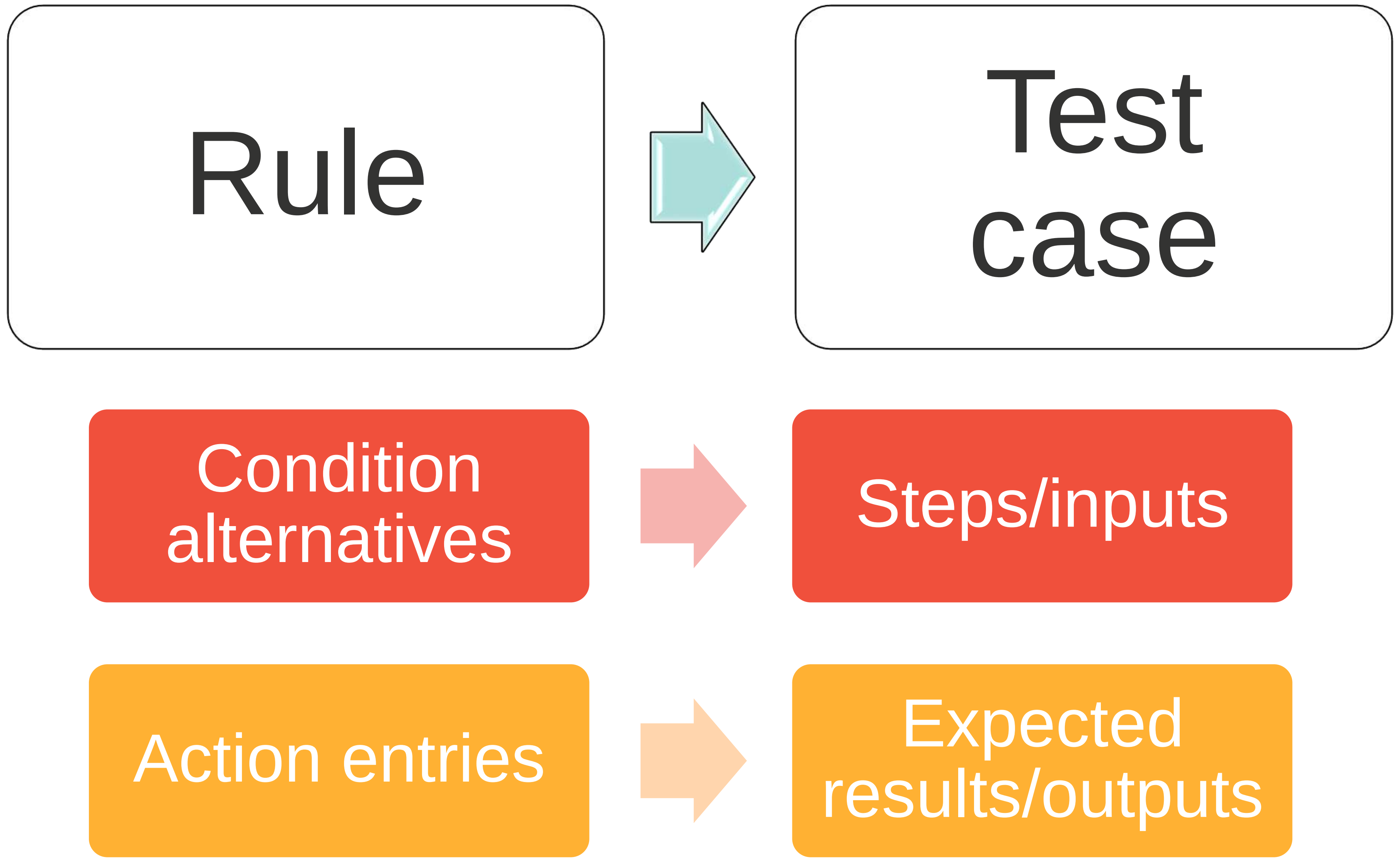
DataArt



Triangle problem											
a < b + c	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
b < a + c	*	N	Y	Y	Y	Y	Y	Y	Y	Y	Y
c < a + b	*	*	N	Y	Y	Y	Y	Y	Y	Y	Y
a = b	*	*	*	N	N	N	N	Y	Y	Y	Y
a = c	*	*	*	N	N	Y	Y	N	N	Y	Y
b = c	*	*	*	N	Y	N	Y	N	Y	N	Y
Not a triangle	Y	Y	Y								
Scalene				Y							
Isosceles					Y	Y		Y			
Equilateral											Y
Impossible							Y		Y	Y	

Decision table methodology





How many tests do we need?

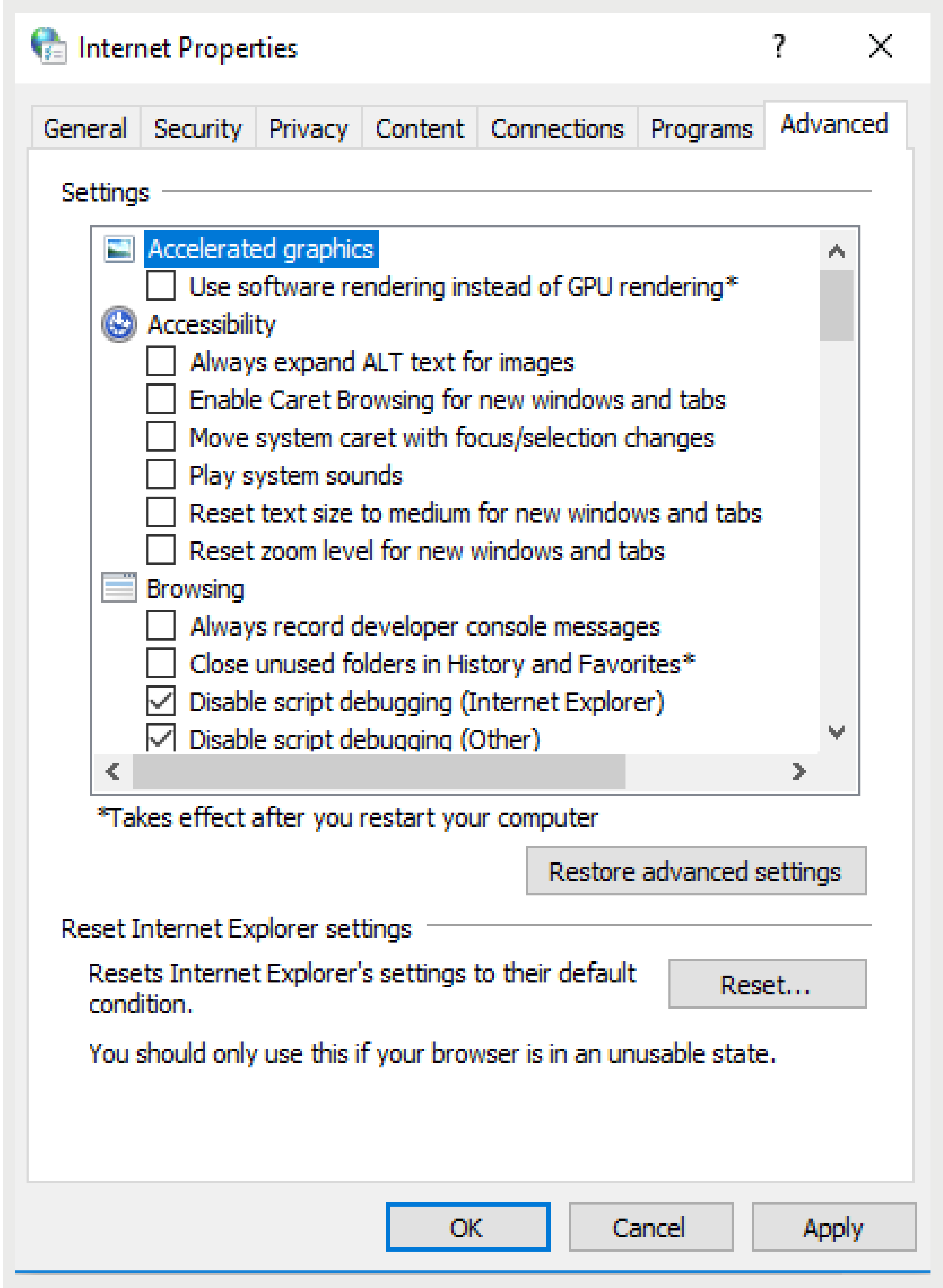
- ✓ 53 binary conditions
- ✓ 1 condition with 3 options
- ✓ 1 condition with 4 options

2^{53} * 3 * 4 = 108 086 391 056 891 904 possible combinations of conditions

1 second per test execution:

108086391056891904 sec = 30023997515803.067 hours

= 3427396976.9 years to test all possible combinations.


Internet Properties

General | Security | Privacy | Content | Connections | Programs | Advanced

Settings

Accelerated graphics
- [ ] Use software rendering instead of GPU rendering*

Accessibility
- [ ] Always expand ALT text for images
- [ ] Enable Caret Browsing for new windows and tabs
- [ ] Move system caret with focus/selection changes
- [ ] Play system sounds
- [ ] Reset text size to medium for new windows and tabs
- [ ] Reset zoom level for new windows and tabs

Browsing
- [ ] Always record developer console messages
- [ ] Close unused folders in History and Favorites*
- [✓] Disable script debugging (Internet Explorer)
- [✓] Disable script debugging (Other)

*Takes effect after you restart your computer

Restore advanced settings

Reset Internet Explorer settings

Resets Internet Explorer's settings to their default condition.

Reset...

You should only use this if your browser is in an unusable state.

OK | Cancel | Apply

Pro and cons of decision tables

- Universal technique
- Easy-to-use
- Simple to read and understand
- Work with complex business logic
- Requirements testing

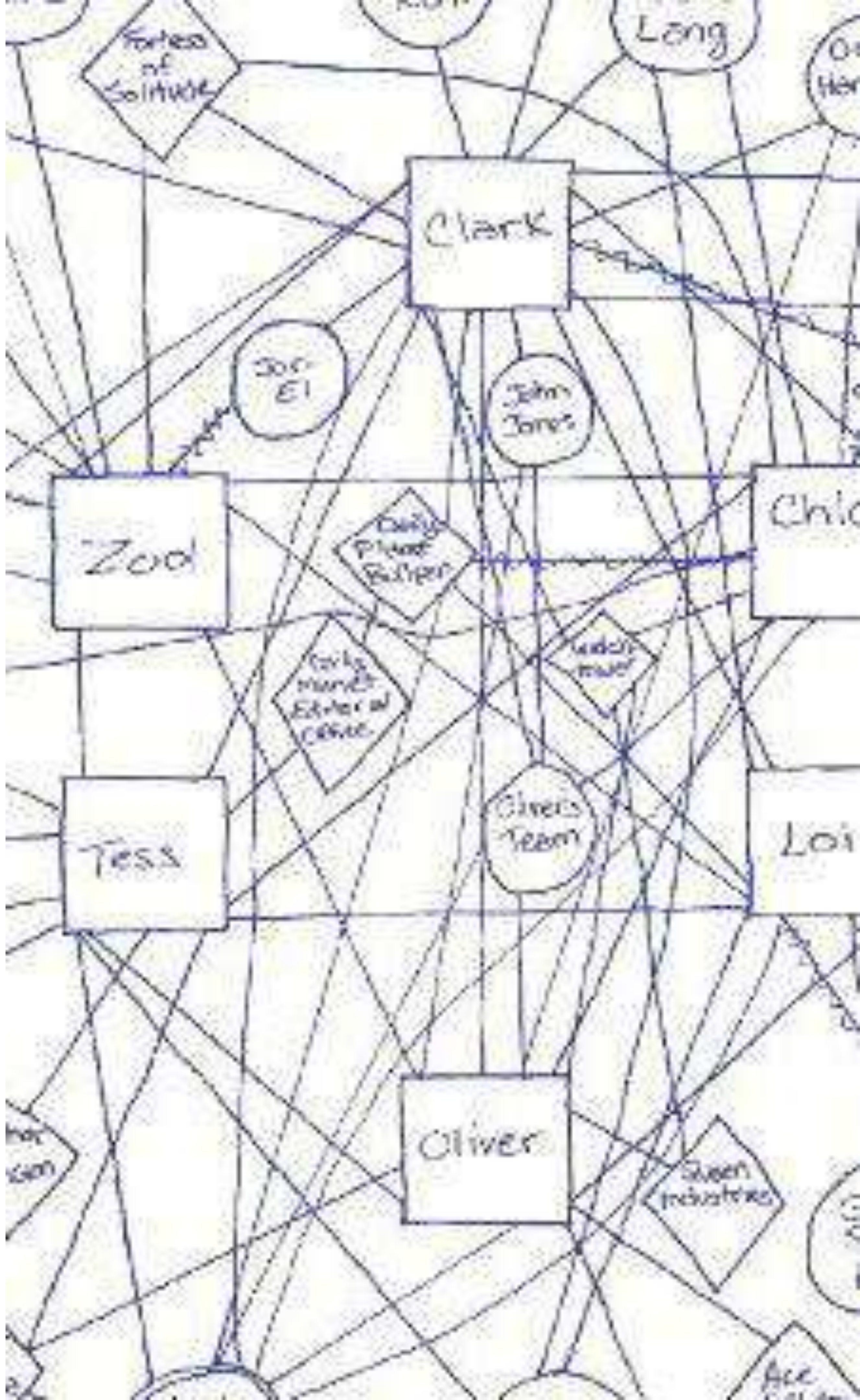
- Too many combinations
- Requires caution when reducing
- Need detailed requirements/well knowledge of business logic

State transition testing

Base ideas

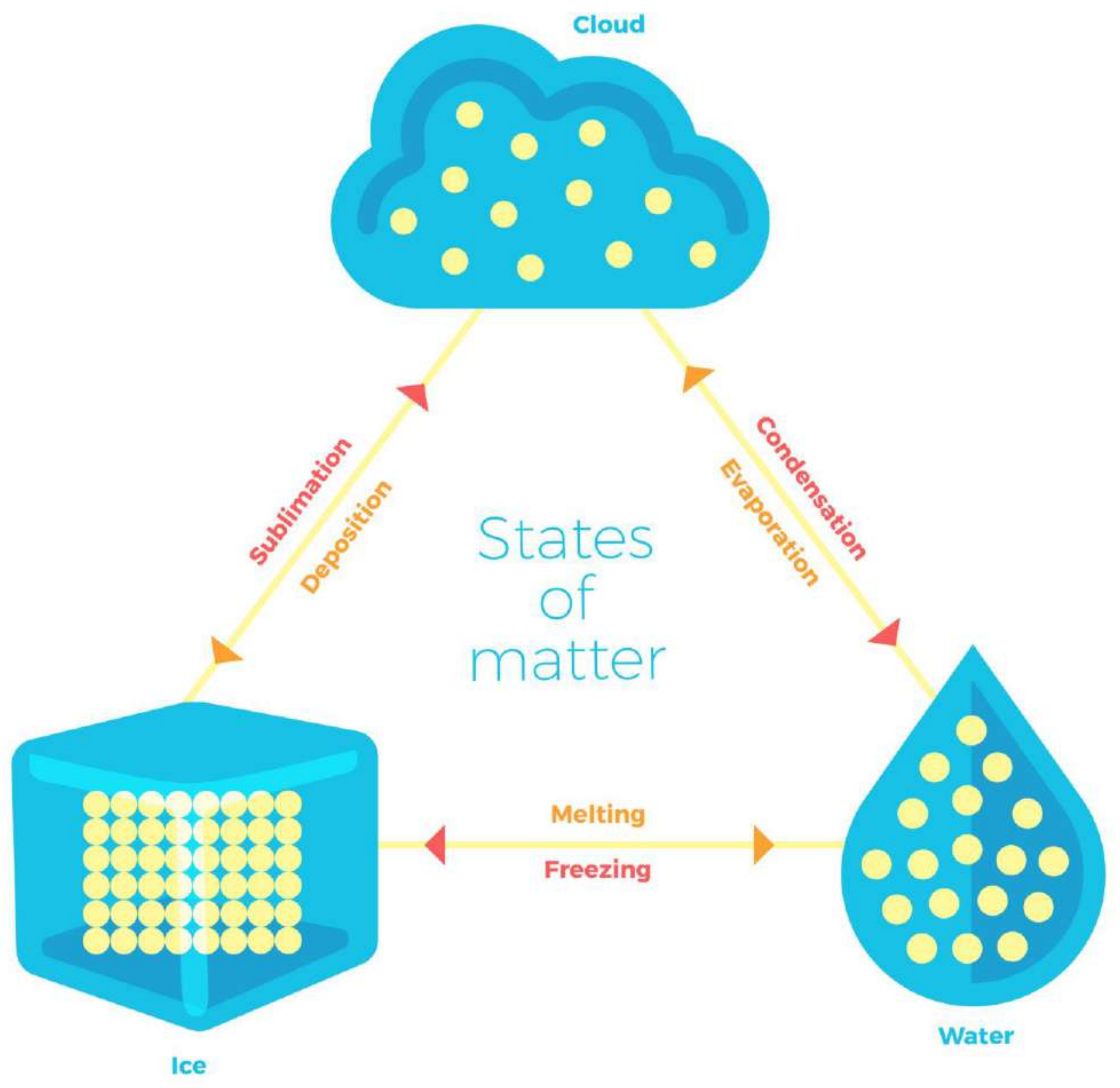
The operation of the system can be represented in the form of a state machine

State transition tables



What do we have to know about state machine?

DataArt



States

- What state does the software might get?

Transition

- Changing a system state from one to another

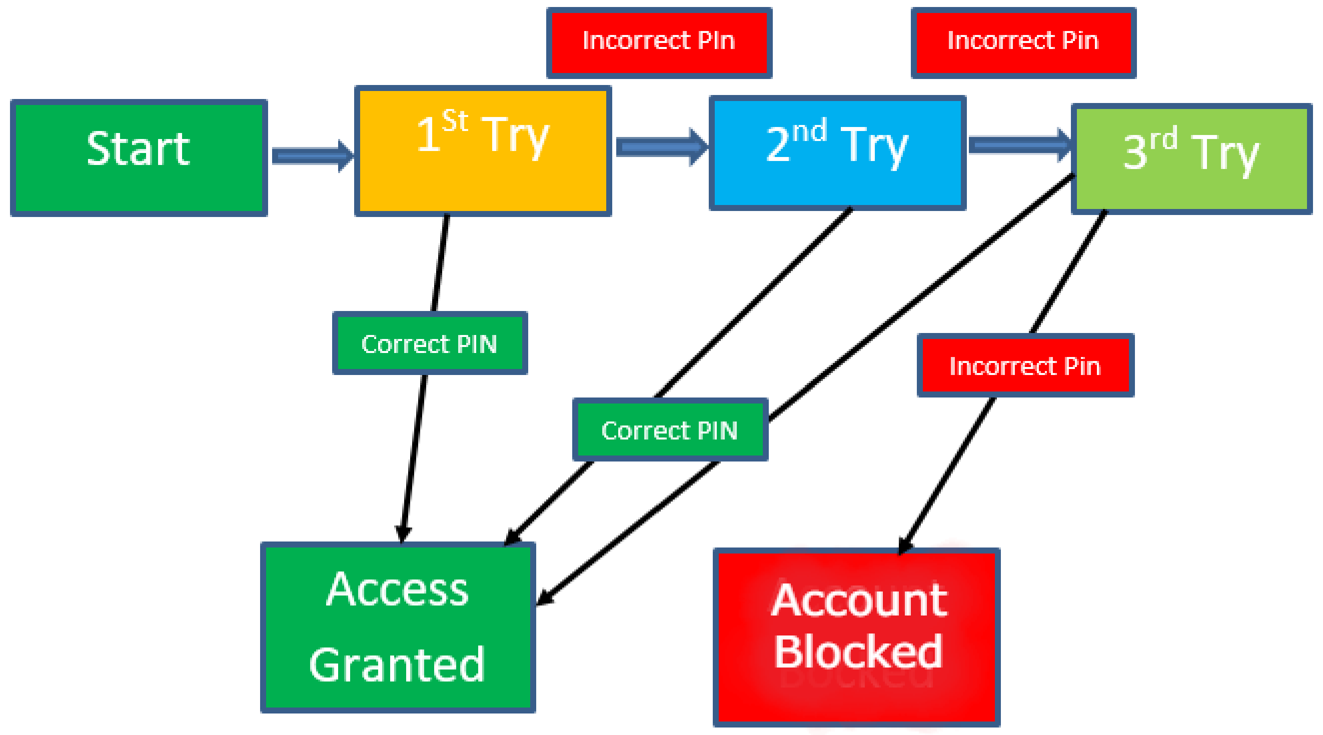
Events

- Reasons for change of state (internal & external)

Actions

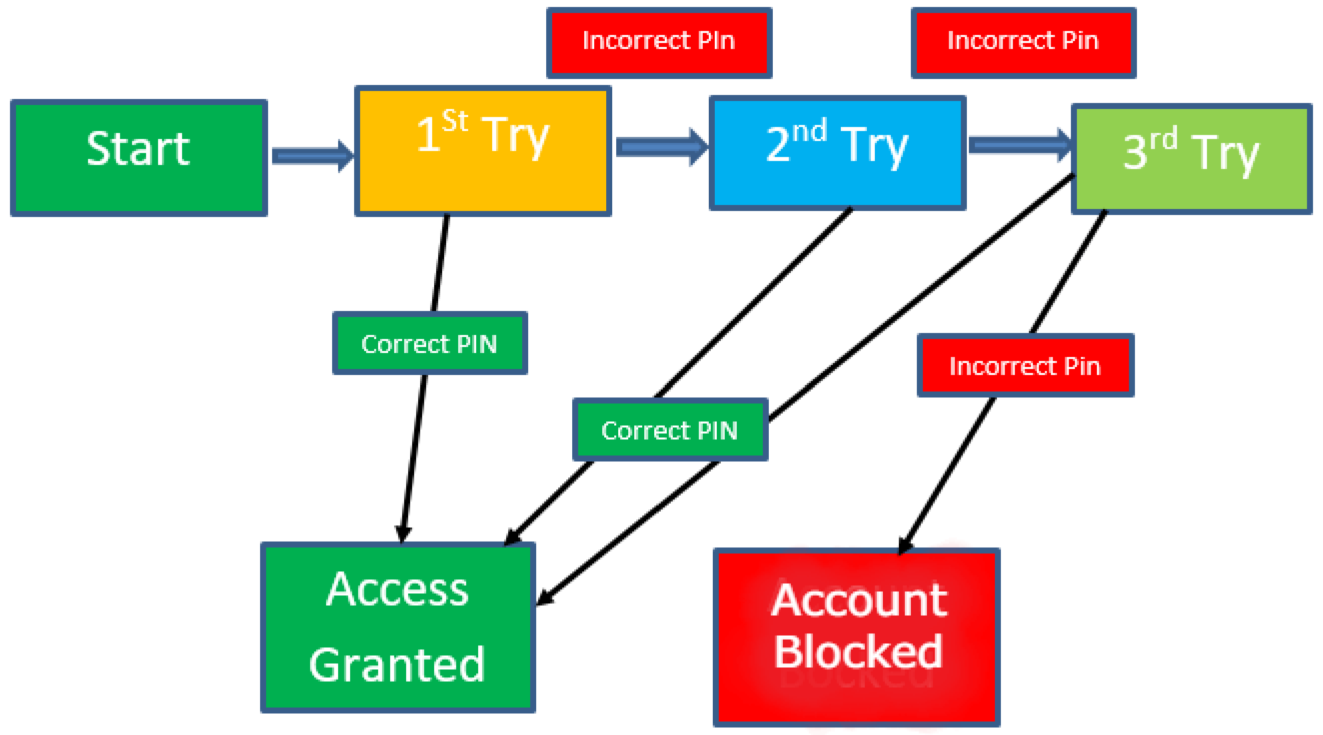
- Operation initiated as a result of a state change

How to represent a state machine?



Initial state	Event	Final state	Action
1st Try	Correct PIN	Access	Home screen
1st Try	Incorrect PIN	2nd Try	PIN Error
2nd Try	Correct PIN	Access	Home screen
2nd Try	Incorrect PIN	3rd Try	PIN Error
3rd Try	Correct PIN	Access	Home screen
3rd Try	Incorrect PIN	Blocked	Blocked Error
Access	Correct PIN	-	-
Access	Incorrect PIN	-	-
Blocked	Correct PIN	-	-
Blocked	Incorrect PIN	-	-

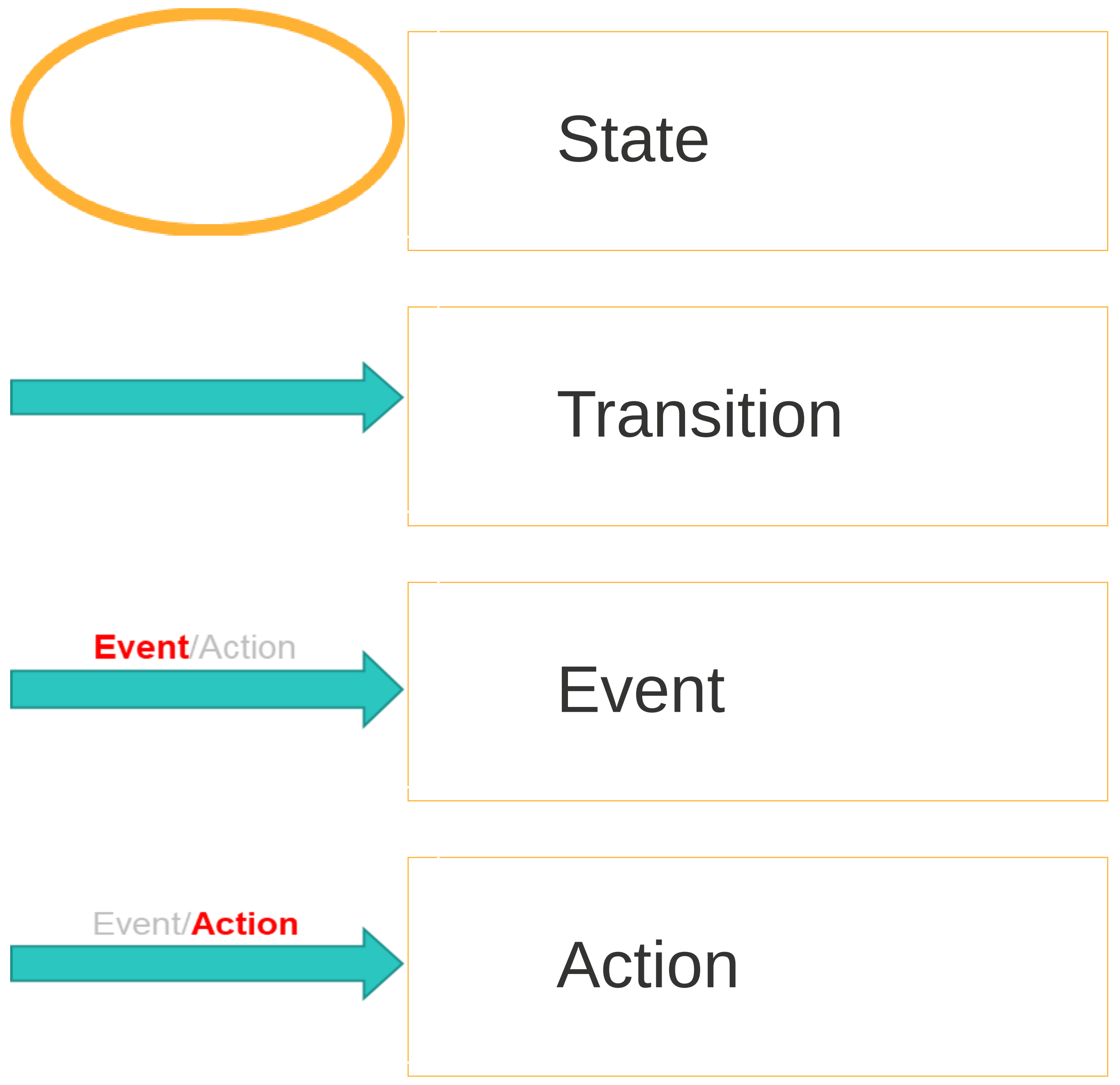
How to represent a state machine?



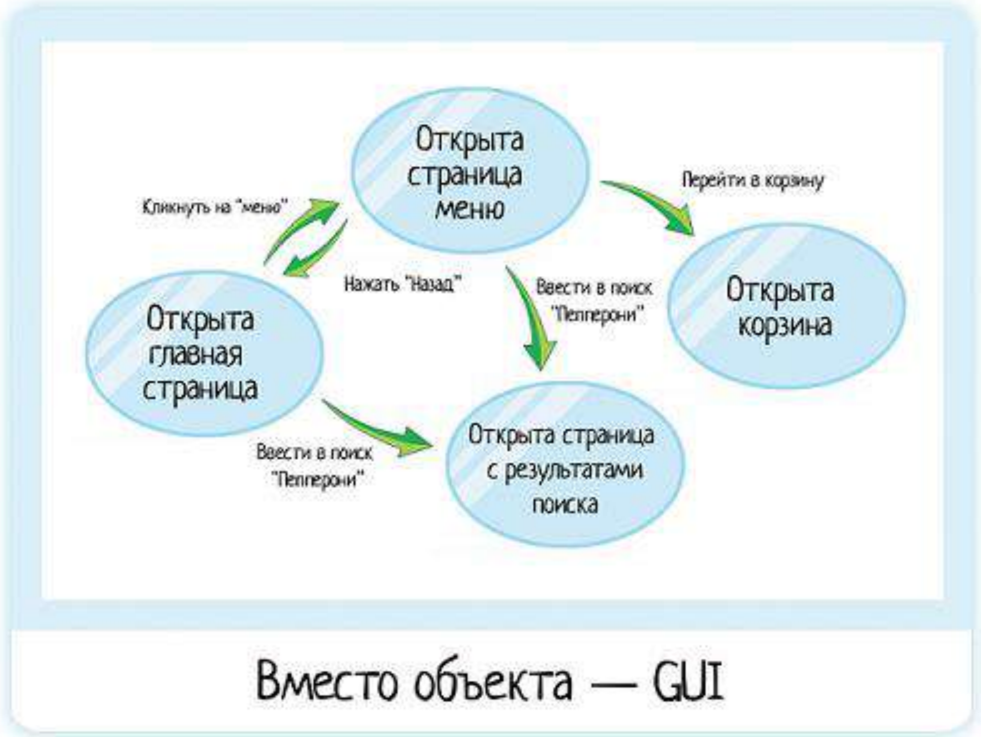
	Correct Pin	Incorrect PIN
1st Try	**Access** Home screen	**2nd Try** PIN Error
2nd Try	**Access** Home screen	**3rd Try** PIN Error
3rd Try	**Access** Home screen	**Blocked** Blocked Error
Access	-	-
Blocked	-	-

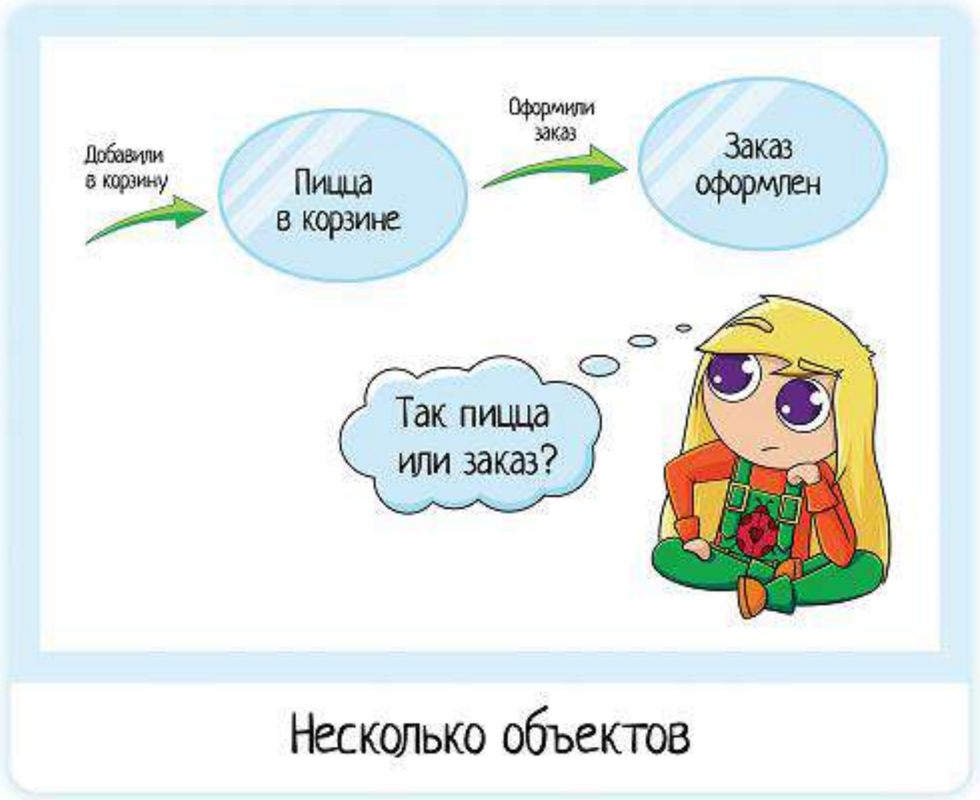
How to draw the diagram?

- ✓ Choose an object
- ✓ Identify states and draw them in circles
- ✓ Define transitions and connect states with arrows
- ✓ Identify events
- ✓ Sign them under/above the arrows.
- ✓ Define actions for each event.
- ✓ Sign them next to the corresponding event



Typical mistakes

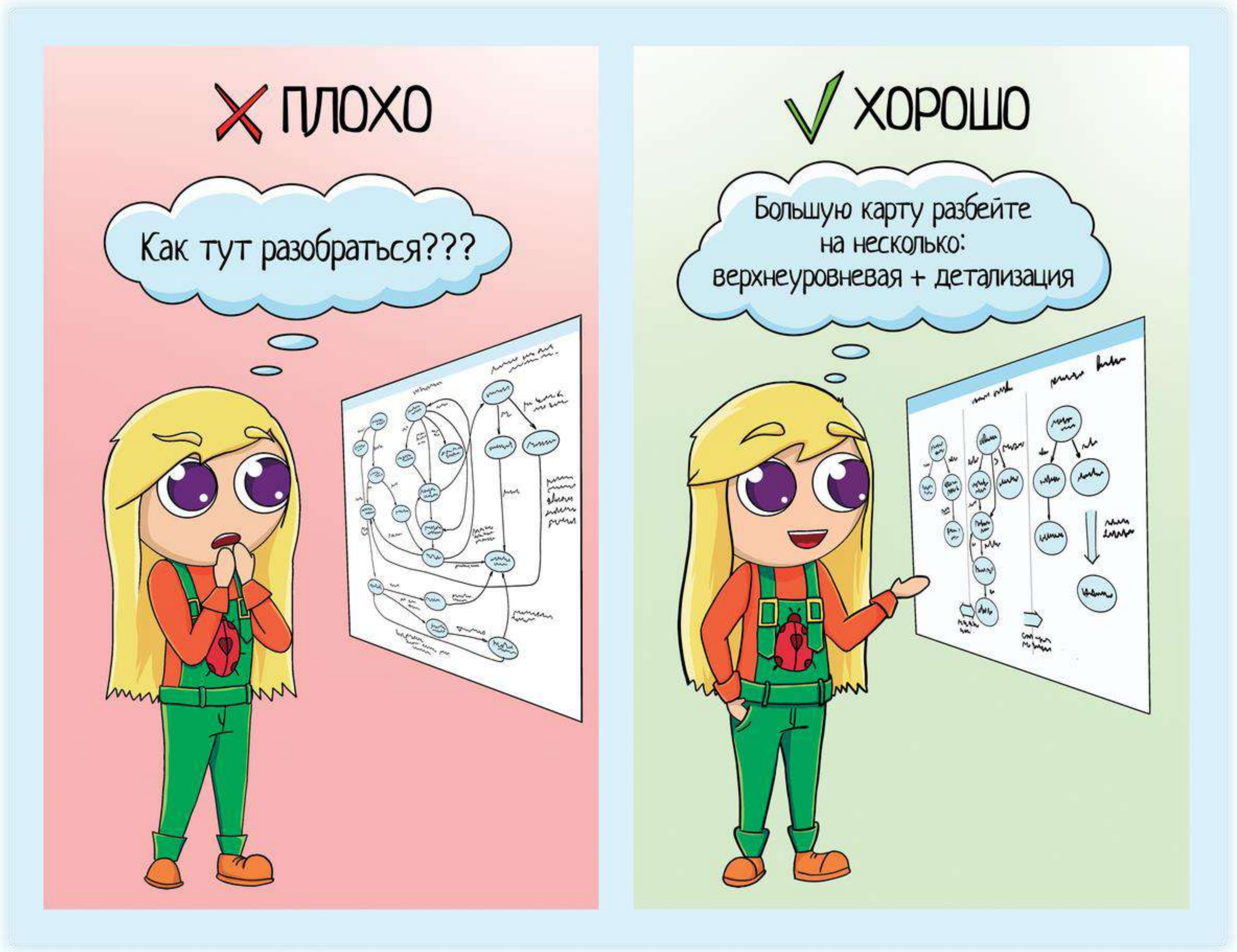




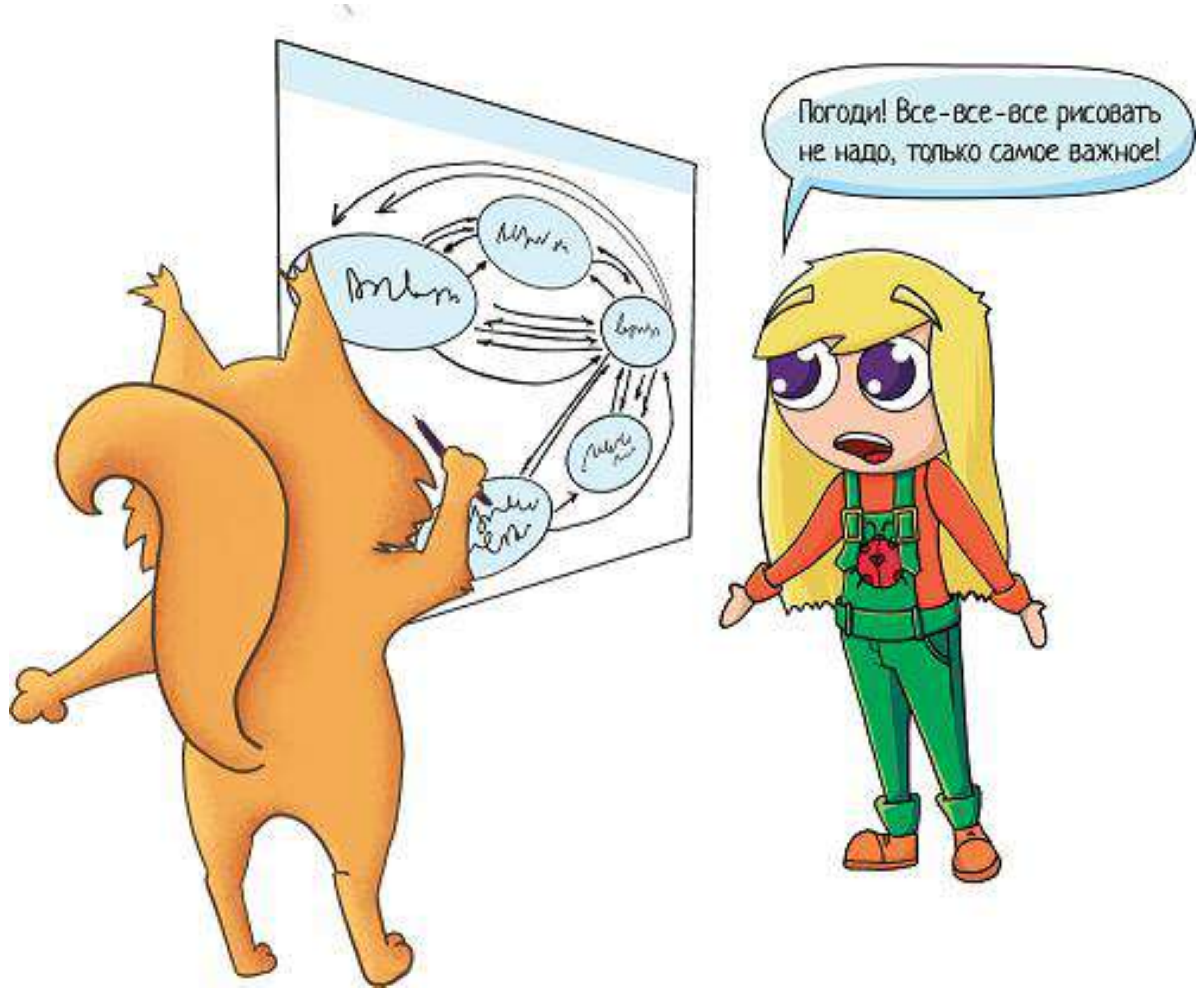
Состояние торта не меняется от перестановок, убирай дублирование!
ТОРТ КУПЛЕН
ТОРТ В ХОЛОДИЛЬНИКЕ

© Ольга @Molechka

Typical mistakes



ПЛОХО
Как тут разобраться???
ХОРОШО
Большую карту разбейте на несколько: верхнеуровневая + детализация



© Ольга @Molechka

How to create state transition table

DataArt

Choose object

Identify states

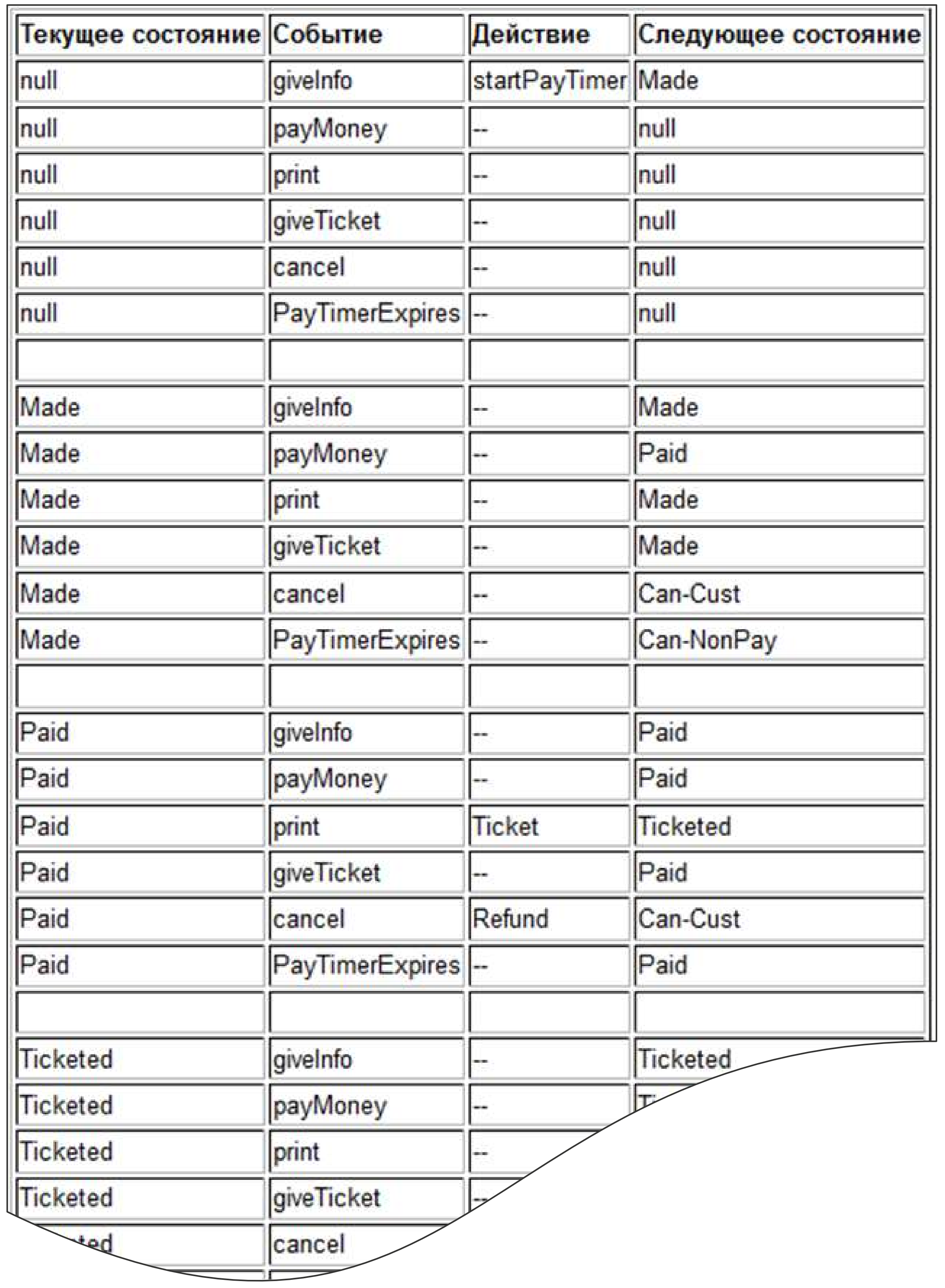
Identify events

Define all combinations state+event

Cross out impossible pairs

Determine the final state for each pair

Determine action for each pair

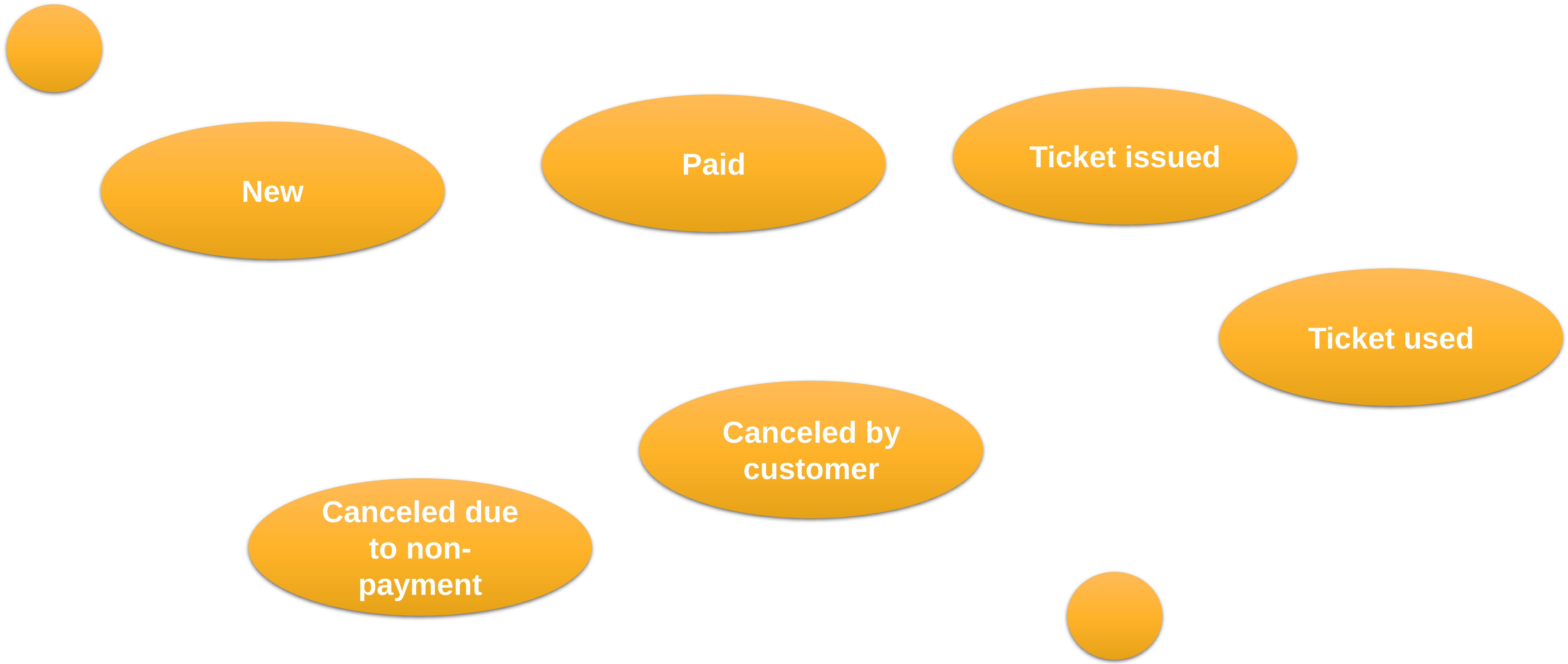


Текущее состояние	Событие	Действие	Следующее состояние
null	giveInfo	startPayTimer	Made
null	payMoney	--	null
null	print	--	null
null	giveTicket	--	null
null	cancel	--	null
null	PayTimerExpires	--	null
Made	giveInfo	--	Made
Made	payMoney	--	Paid
Made	print	--	Made
Made	giveTicket	--	Made
Made	cancel	--	Can-Cust
Made	PayTimerExpires	--	Can-NonPay
Paid	giveInfo	--	Paid
Paid	payMoney	--	Paid
Paid	print	Ticket	Ticketed
Paid	giveTicket	--	Paid
Paid	cancel	Refund	Can-Cust
Paid	PayTimerExpires	--	Paid
Ticketed	giveInfo	--	Ticketed
Ticketed	payMoney	--	T
Ticketed	print	--	
Ticketed	giveTicket	--	
ted	cancel		

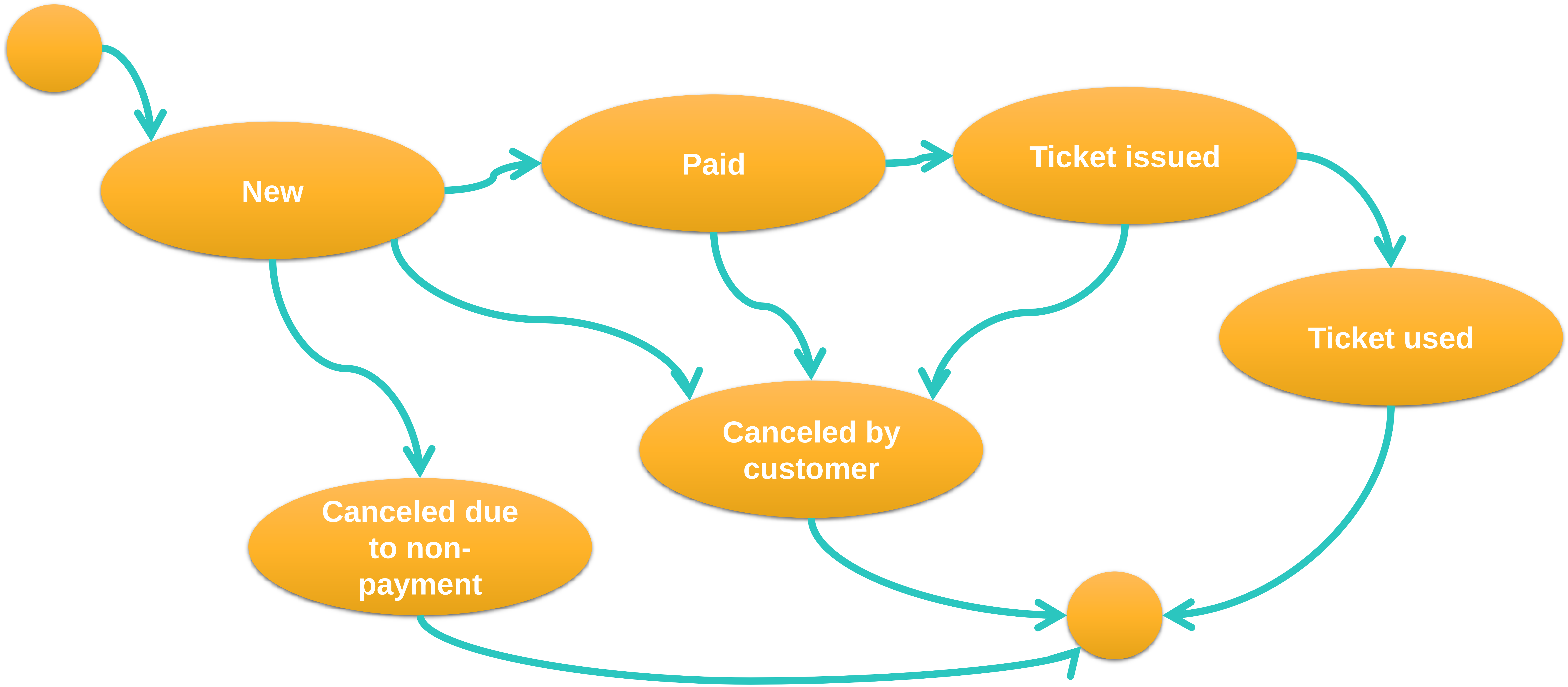
Example – flight reservation

- New
- Ticket used
- Paid
- Canceled by customer
- Ticket issued
- Canceled due to non-payment

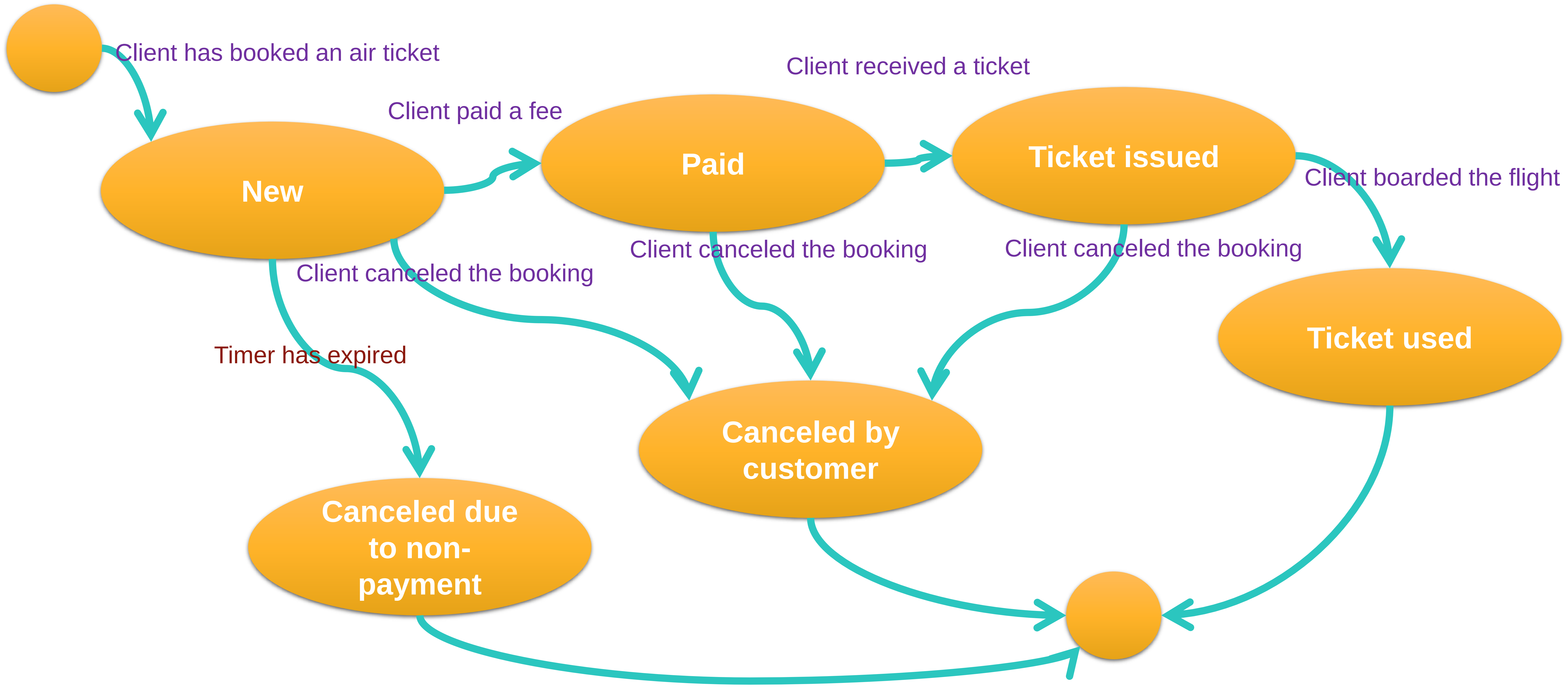
Example – flight reservation



Example – flight reservation

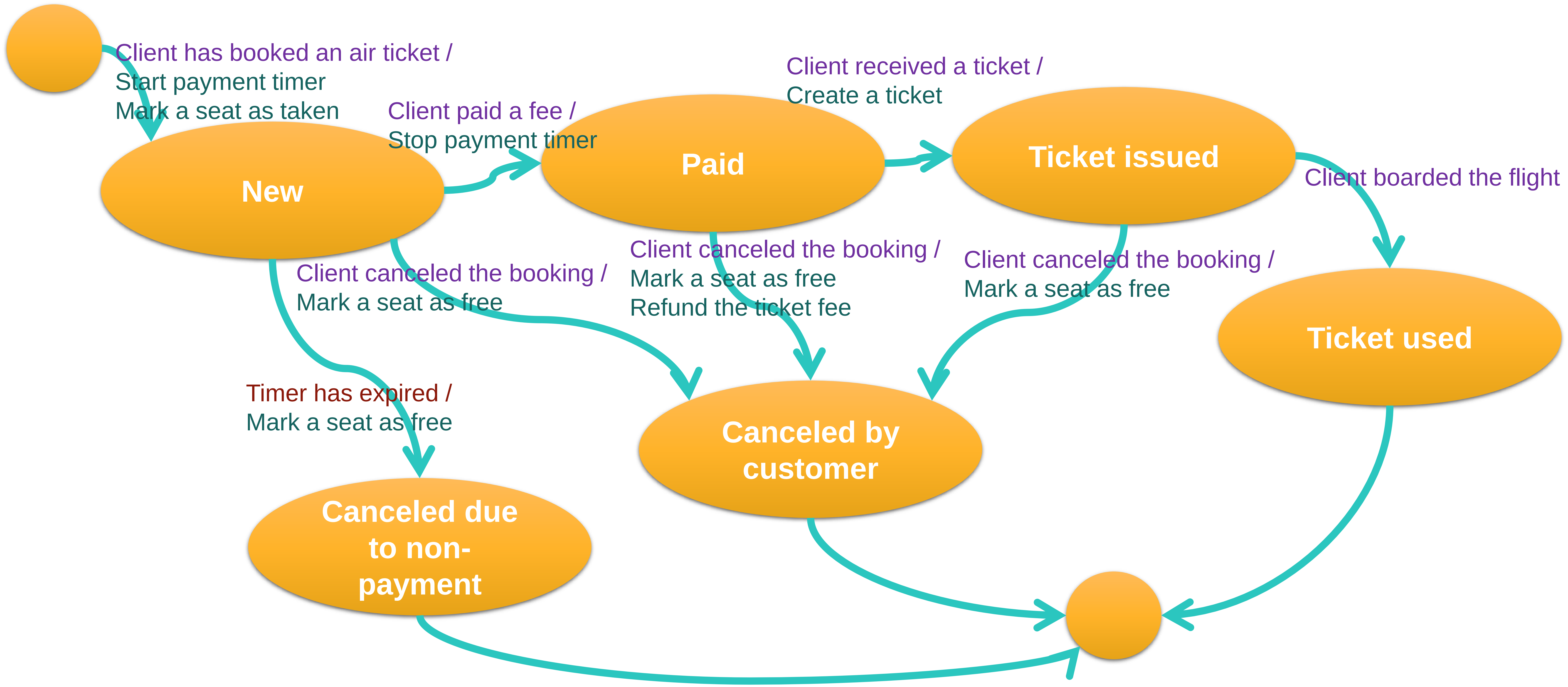


Example – flight reservation

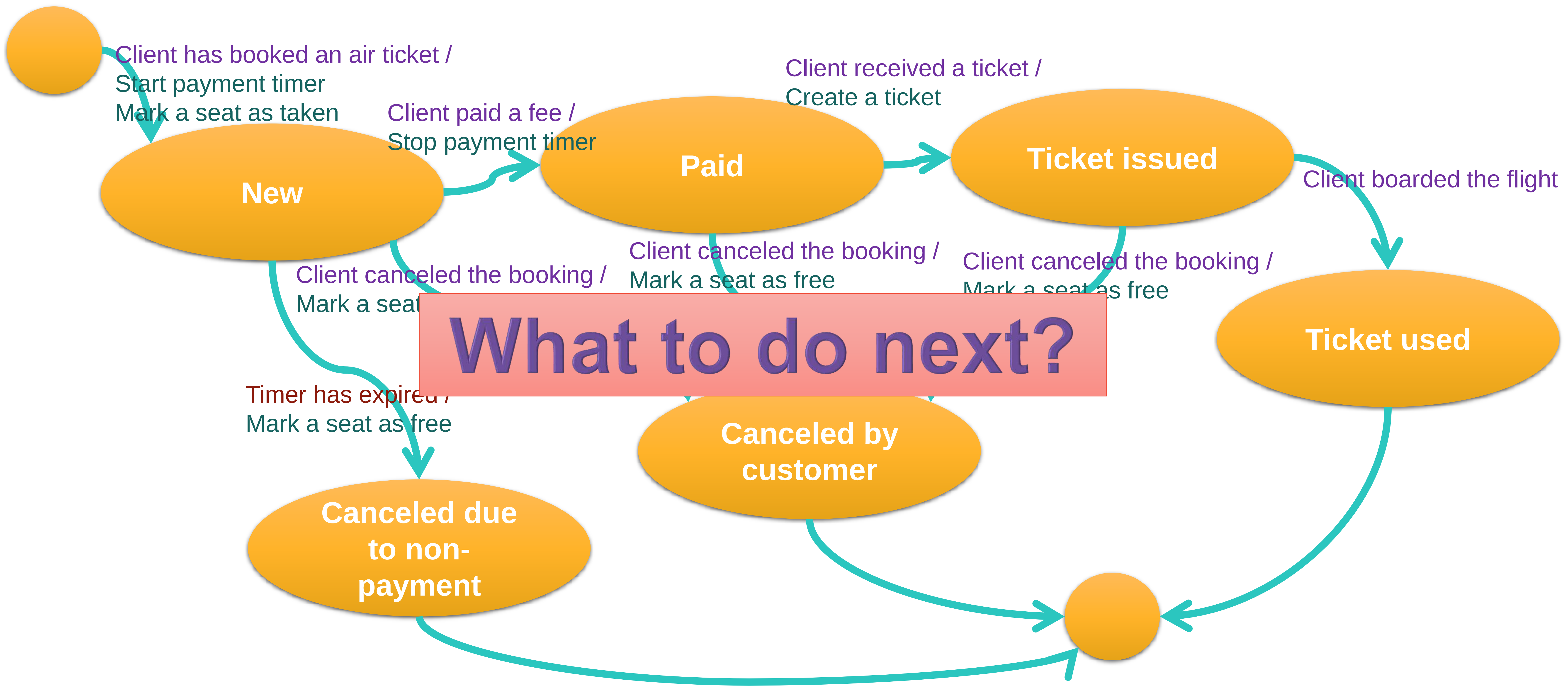


Client has booked an air ticket
Client paid a fee
Client received a ticket
New
Paid
Ticket issued
Client boarded the flight
Client canceled the booking
Client canceled the booking
Client canceled the booking
Ticket used
Timer has expired
Canceled by customer
Canceled due to non-payment

Example – flight reservation

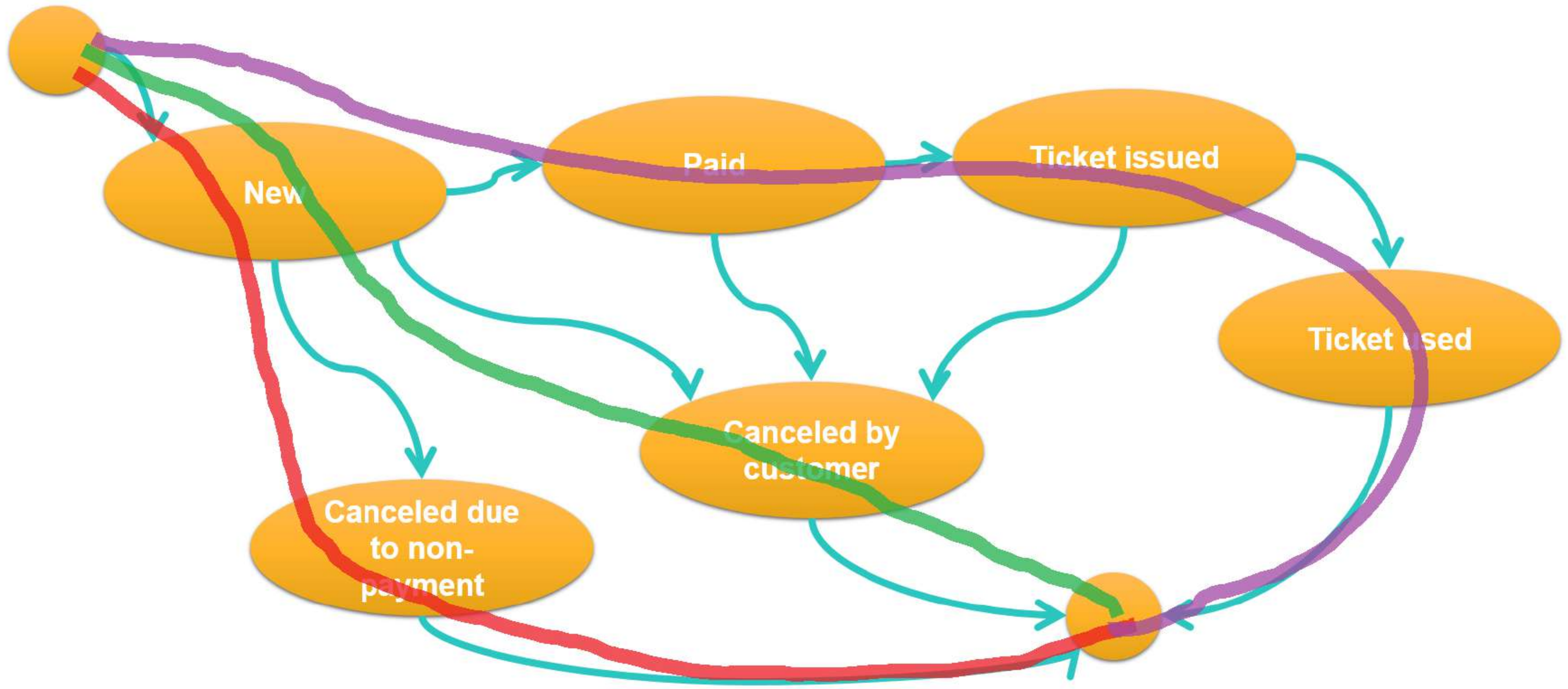


Example – flight reservation



Method 1. States covering

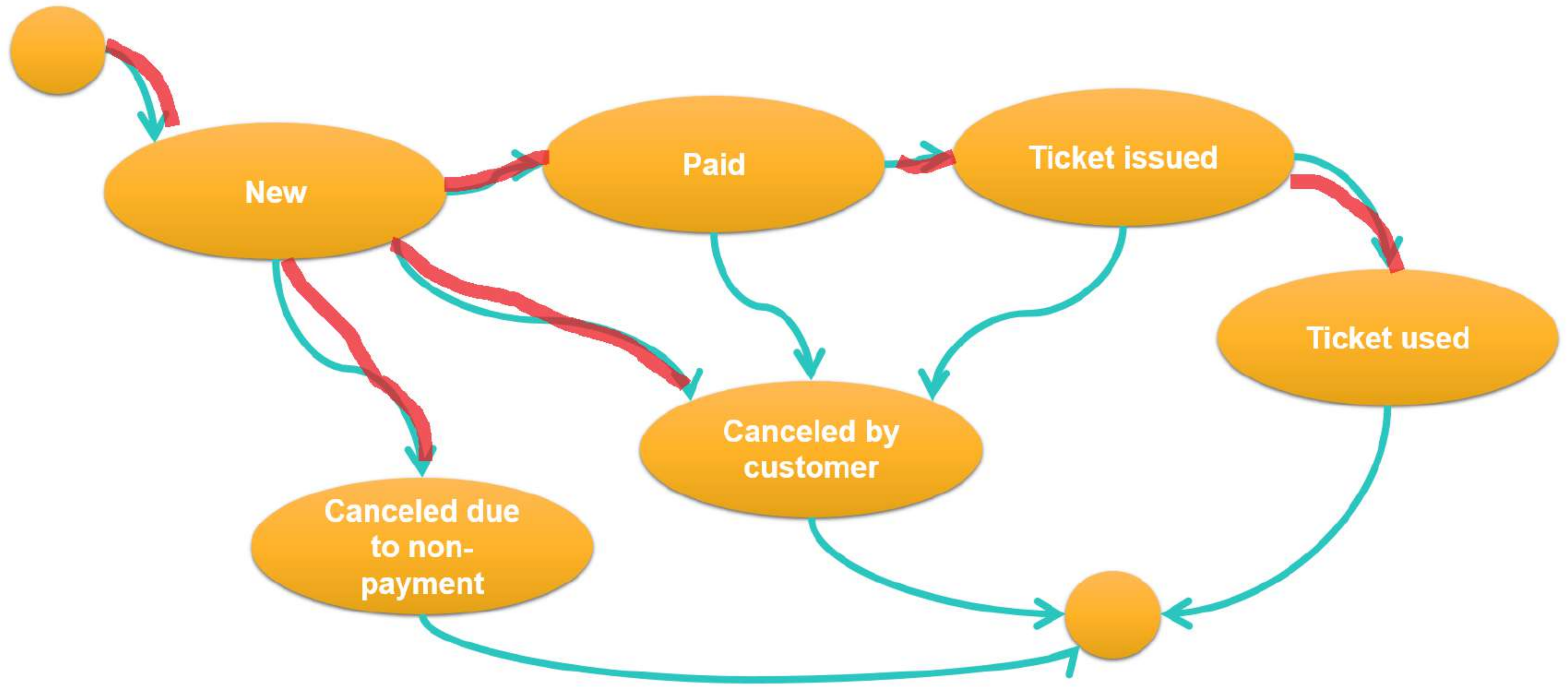
Create sets of test cases so that all states are passed at least once.



A rather weak level of test coverage.

Method 2. Events covering

Create sets of test cases so that all events are triggered at least once.

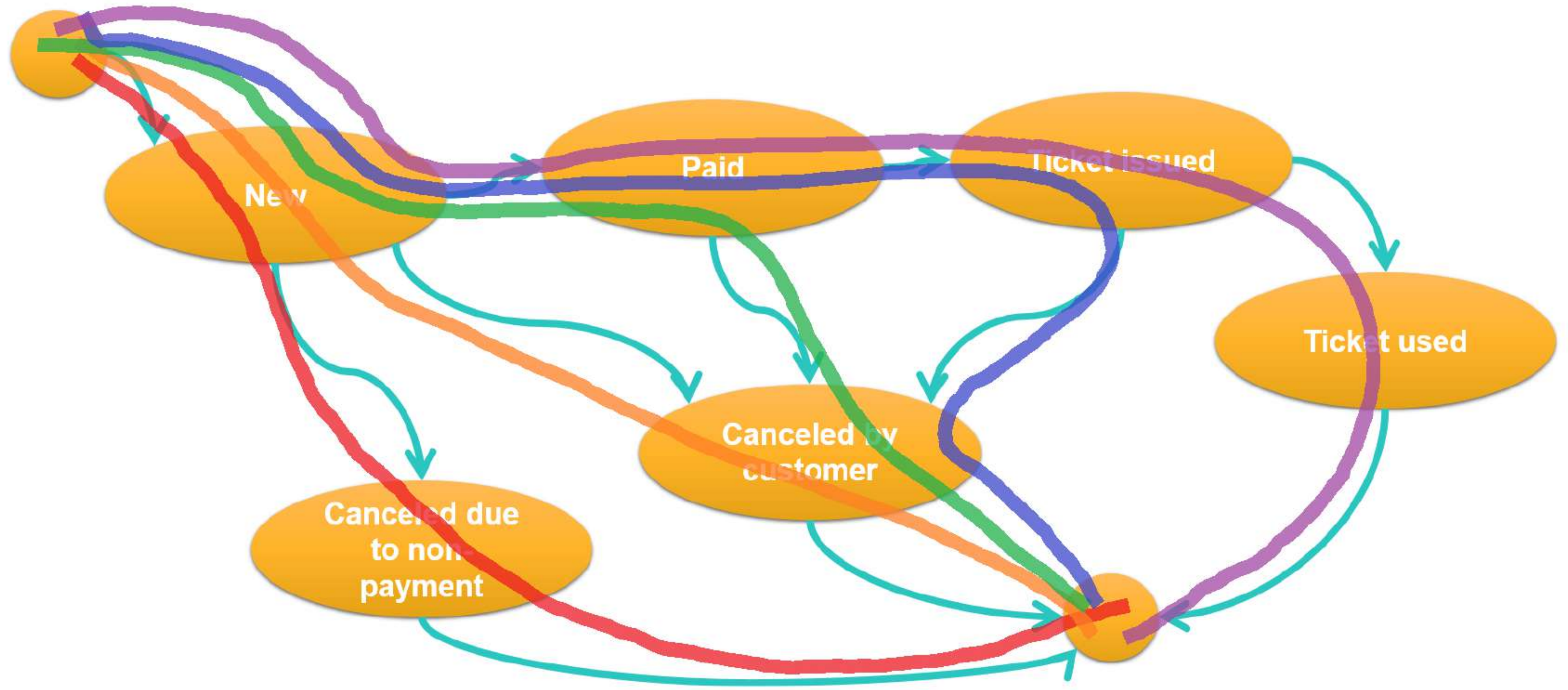


Test cases that cover all events at the same time cover all states.

Again, the level of test coverage is weak.

Method 3. Path covering

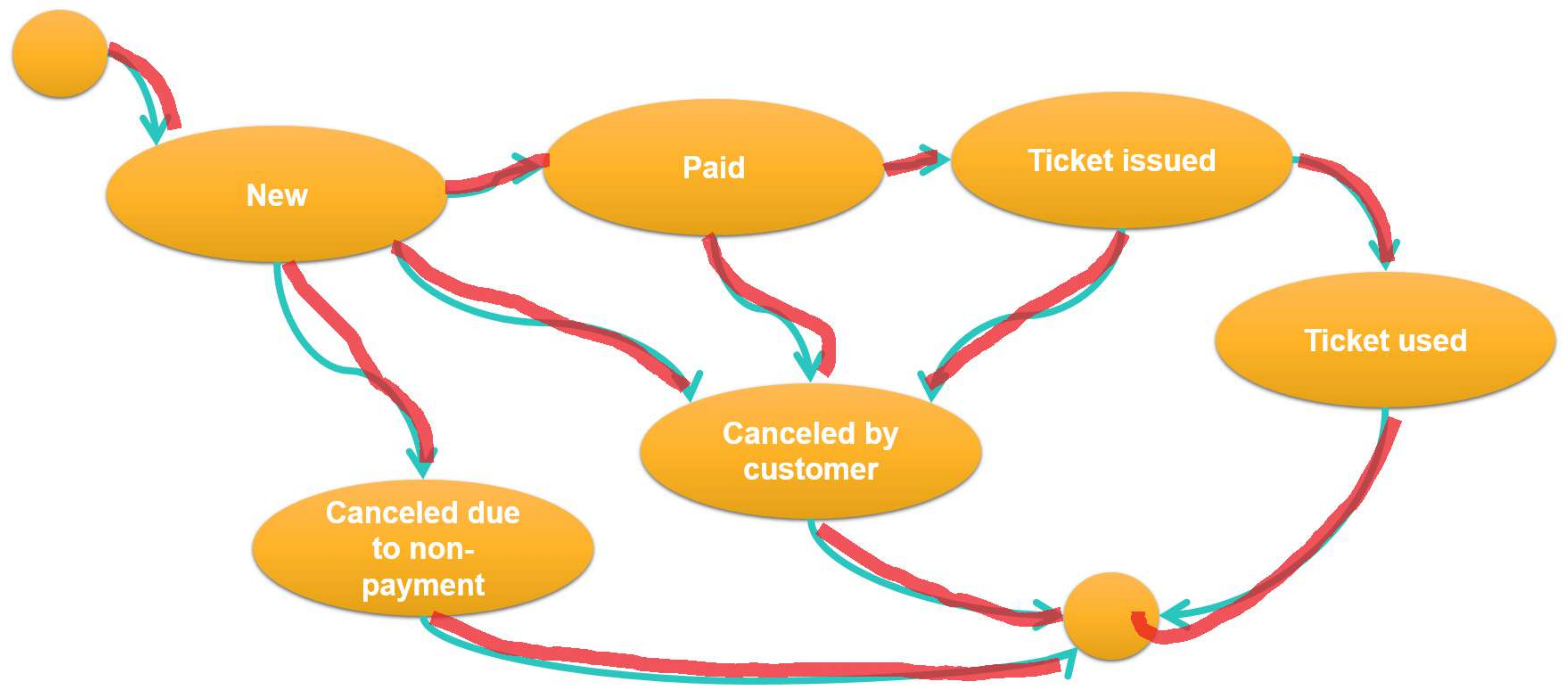
Create sets of test cases so that all paths are traversed at least once.



Good test coverage, but practically impossible sometimes. If the diagram has cycles, then the number of possible paths can be infinite.

Method 4. Transition covering

Create sets of test cases so that all transitions are performed at least once.



This method provides a good level of test coverage, fix volume of testing.

Other approach – state transition table

States
- Start (null)
- New
- Paid
- Ticket issued
- Ticket used
- Canceled by customer
- Canceled due to non-payment

×

Events
- Client has booked an air ticket
- Client paid a fee
- Client received a ticket
- Client boarded the flight
- Client canceled the booking
- Timer has expired

=

?

Other approach – state transition table

DataArt

States

- Start (null)
- New
- Paid
- Ticket issued
- Ticket used
- Canceled by customer
- Canceled due to non-payment

×

Events

- Client has booked an air ticket
- Client paid a fee
- Client received a ticket
- Client boarded the flight
- Client canceled the booking
- Timer has expired

=

State	Event	New state	Action
null	Client has booked an air ticket	New	Start timer Mark a seat as taken
null	Client paid a fee	-	
null	Client received a ticket	-	
null	Client boarded the flight	-	
null	Client canceled the booking	-	
null	Timer has expired	-	
New	Client has booked an air ticket	-	
New	Client paid a fee	Paid	Start timer
New	Client received a ticket	-	

Other approach – state transition table

DataArt

States

- Start (null)
- New
- Paid
- Ticket issued
- Ticket used
- Canceled by customer
- Canceled due to non-payment

×

Events

- Client has booked an air ticket
- Client paid a fee
- Client received a ticket
- Client boarded the flight
- Client canceled the booking
- Timer has expired

=

State	Event	New state	Action
null	Client has booked an air ticket	New	Start timer Mark a seat as taken
null	Client paid a fee	-	
null	Client received a ticket	-	
null	Client boarded the flight	-	
null	Client canceled the booking	-	
null	Timer has expired	-	
New	Client has booked an air ticket	-	
New	Client paid a fee	Paid	Start timer
New	Client received a ticket	-	

Other approach – state transition table

Analyze

- ✓ completeness of requirements
- ✓ correctness of the user interface
- ✓ if transitions are skipped

State	Event	New state	Action
null	Client has booked an air ticket	New	Start timer Mark a seat as taken
null	Client paid a fee	-	
null	Client received a ticket	-	
null	Client boarded the flight	-	
null	Client canceled the booking	-	
null	Timer has expired	-	
New	Client has booked an air ticket	-	
New	Client paid a fee	Paid	Start timer
New	Client received a ticket	-	

Other approach – state transition table

Remaining row → Test case

Initial state & Event → Steps/inputs

New state & Actions → Expected results/outputs

State	Event	New state	Action
null	Client has booked an air ticket	New	Start timer Mark a seat as taken
null	Client paid a fee	-	
null	Client received a ticket	-	
null	Client boarded the flight	-	
null	Client canceled the booking	-	
null	Timer has expired	-	
New	Client has booked an air ticket	-	
New	Client paid a fee	Paid	Start timer
New	Client received a ticket	-	

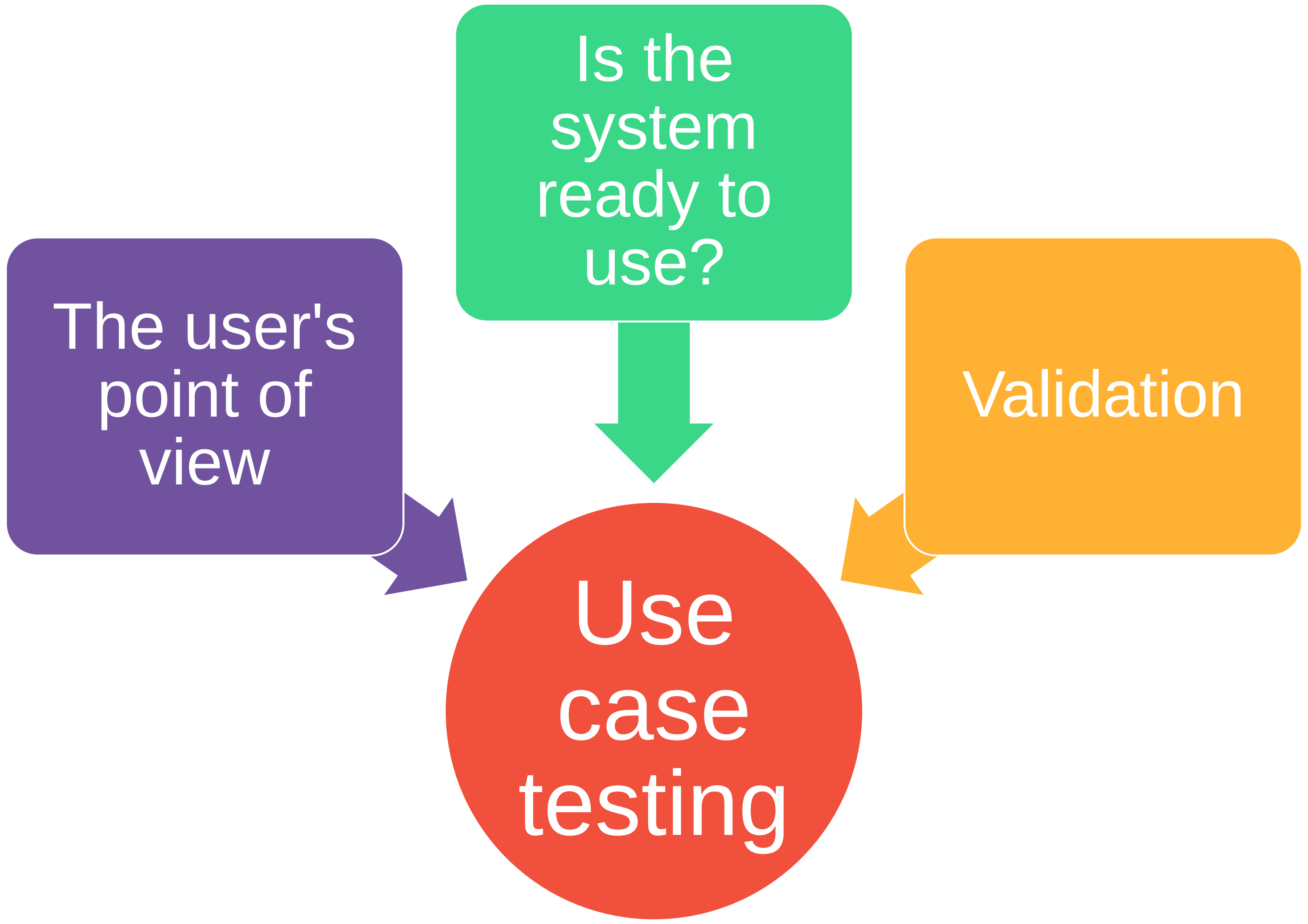
Pro and cons of state transition testing

- Allow to check the feature as a whole in a short period of time
- Provide a pictorial or tabular representation of system behavior
- Formal alternative to use case testing
- Good for smoke testing
- Requirements testing

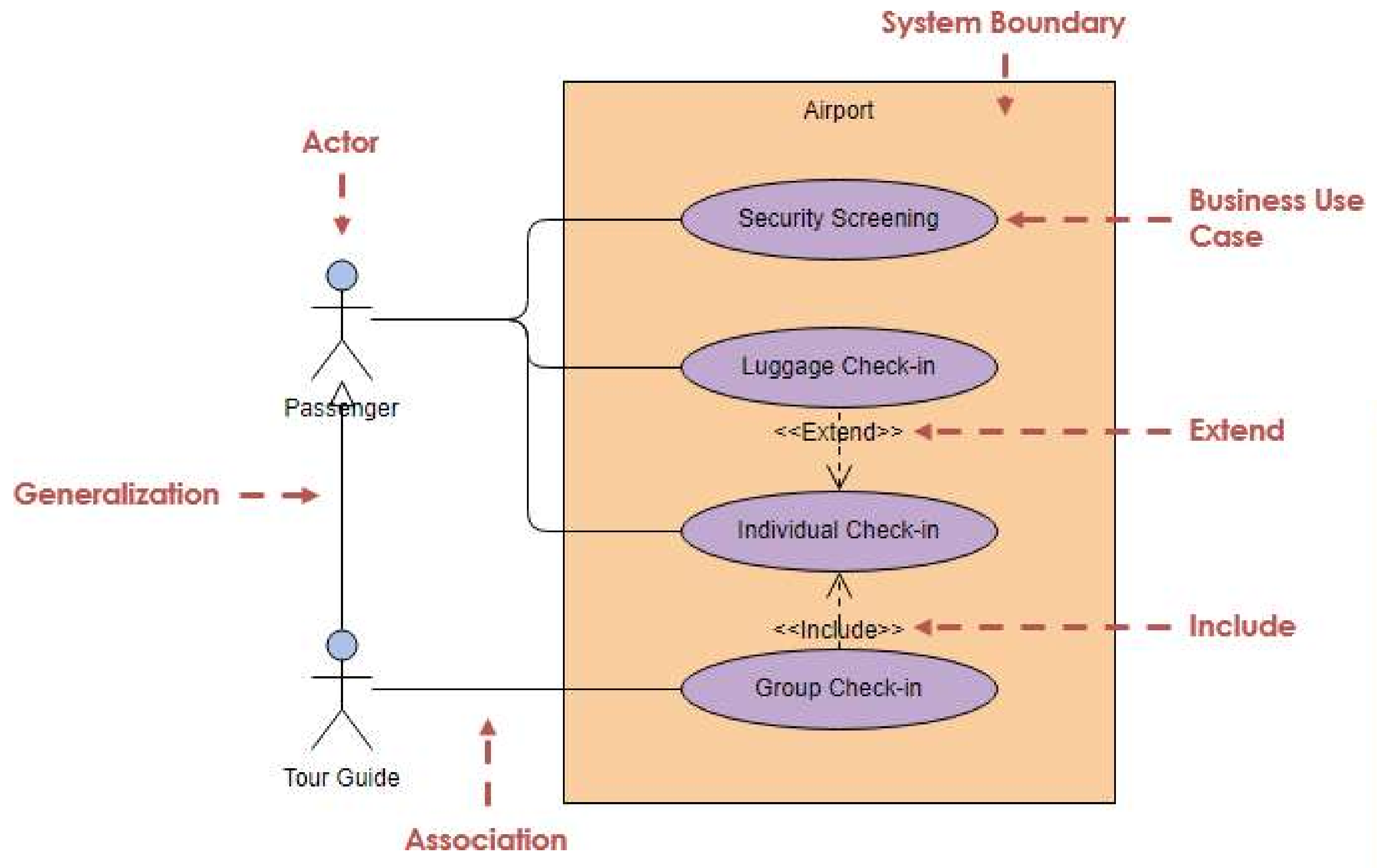
- Suitable only for some objects / systems
- Dependent on correct identification of states, events and actions
- Allows to test only one aspect of the behavior
- Provides not too deep testing

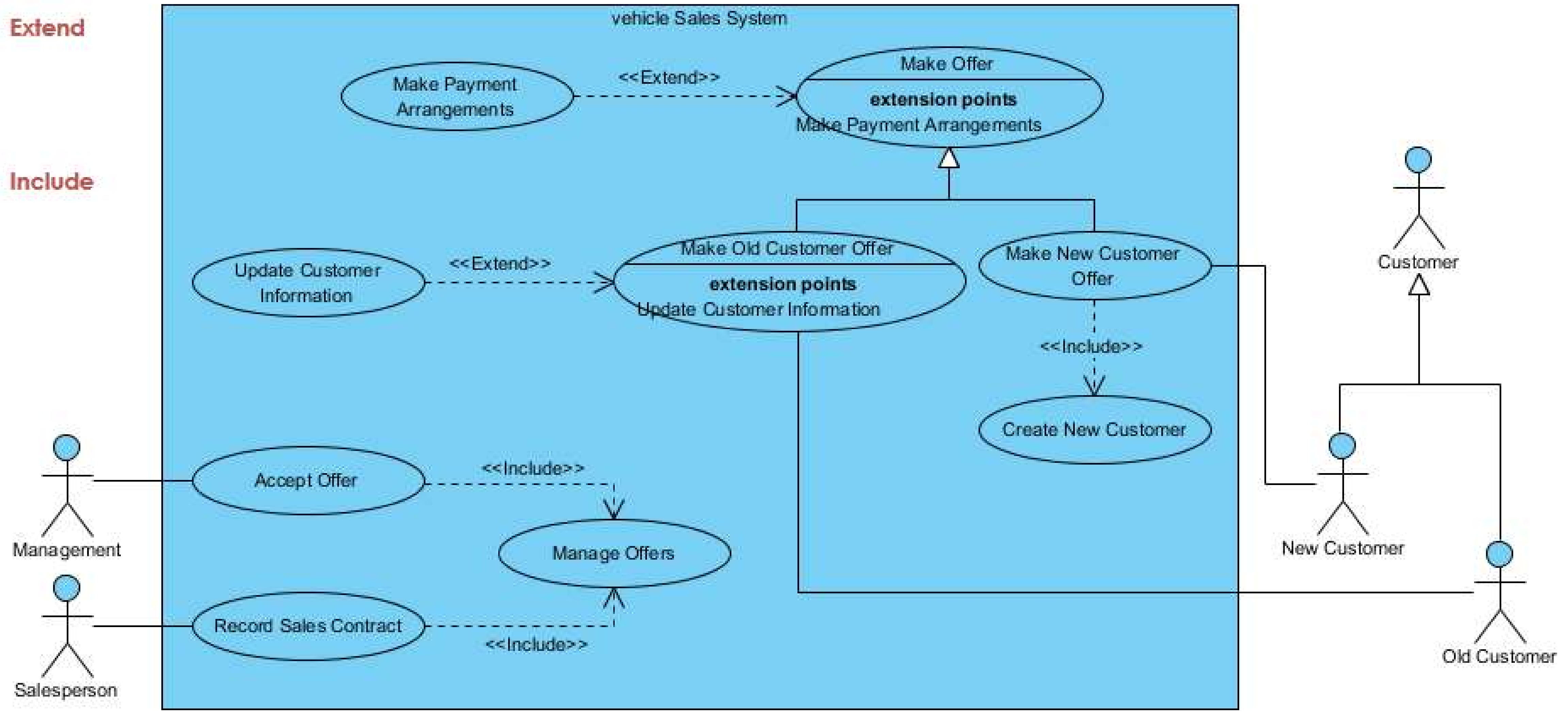
Use case testing

Base ideas


Is the system ready to use?
The user's point of view
Validation
Use case testing

What is a use case?

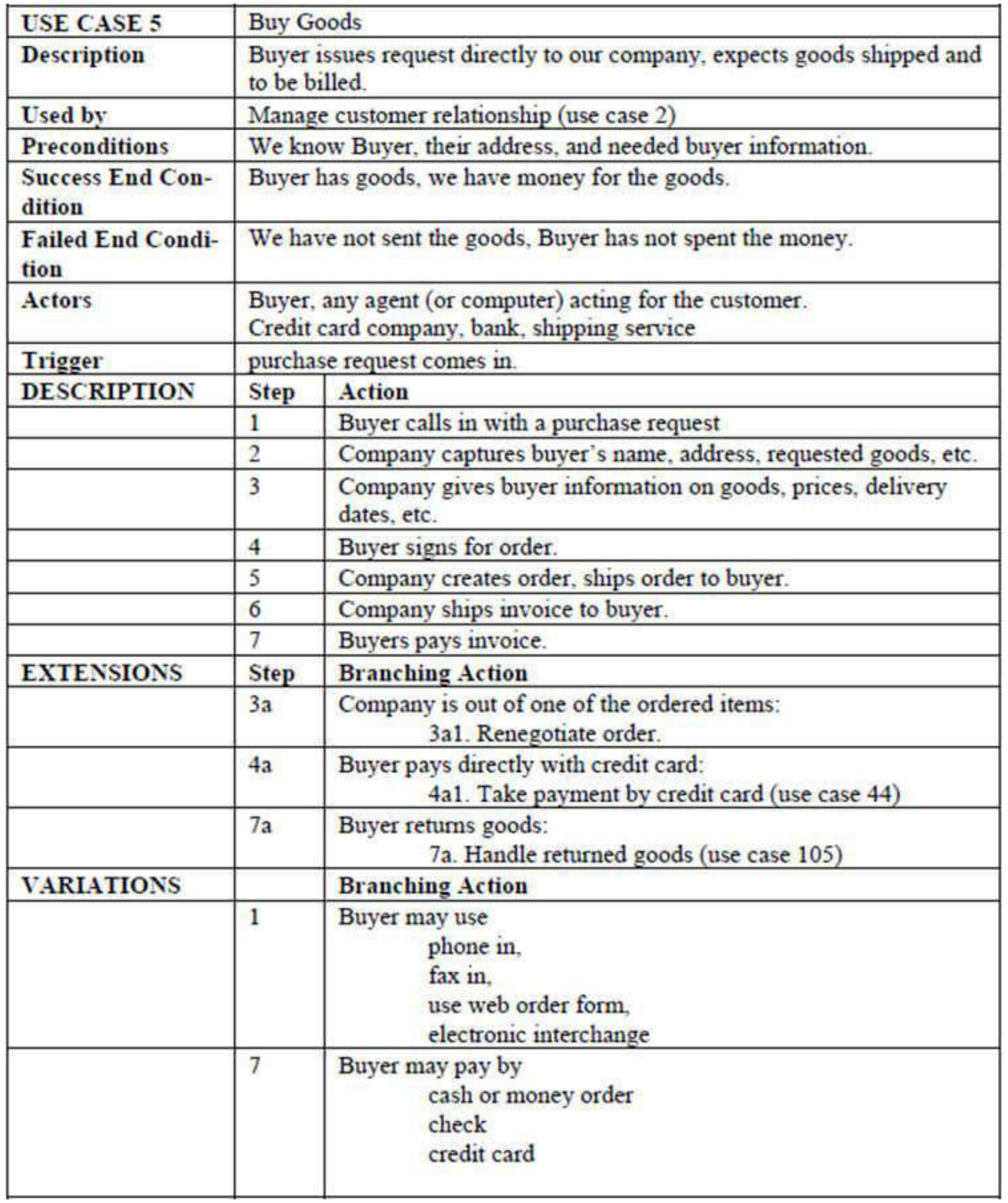




What is a use case?

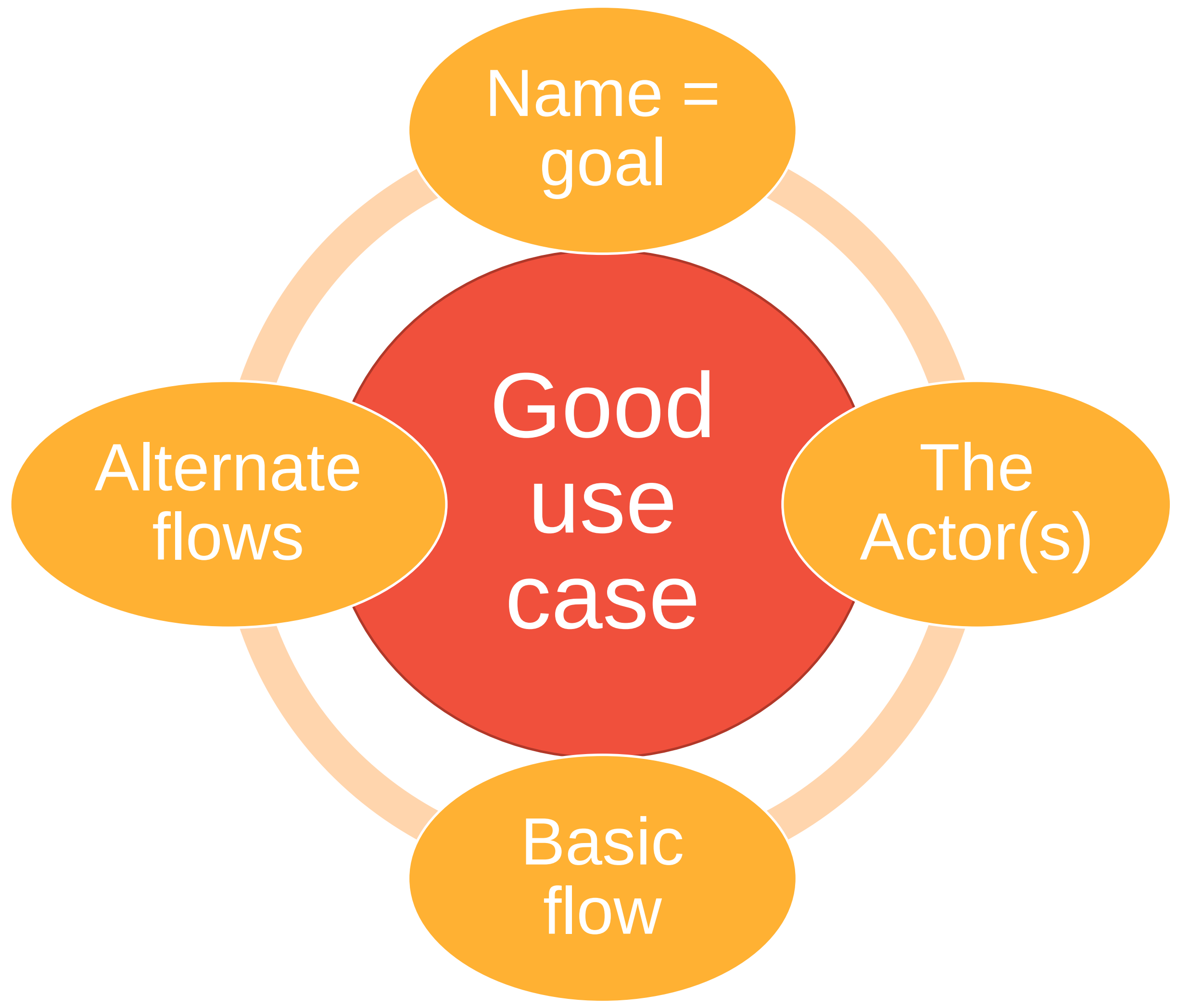
	Step	Description
Main Success Scenario A: Actor S: System	1	A: Inserts card
	2	S: Validates card and asks for PIN
	3	A: Enters PIN
	4	S: Validates PIN
	5	S: Allows access to account
Extensions	2a	Card not valid S: Display message and reject card
	4a	PIN not valid S: Display message and ask for re-try (twice)
	4b	PIN invalid 3 times S: Eat card and Exit

A Partial Use Case for PIN Entry

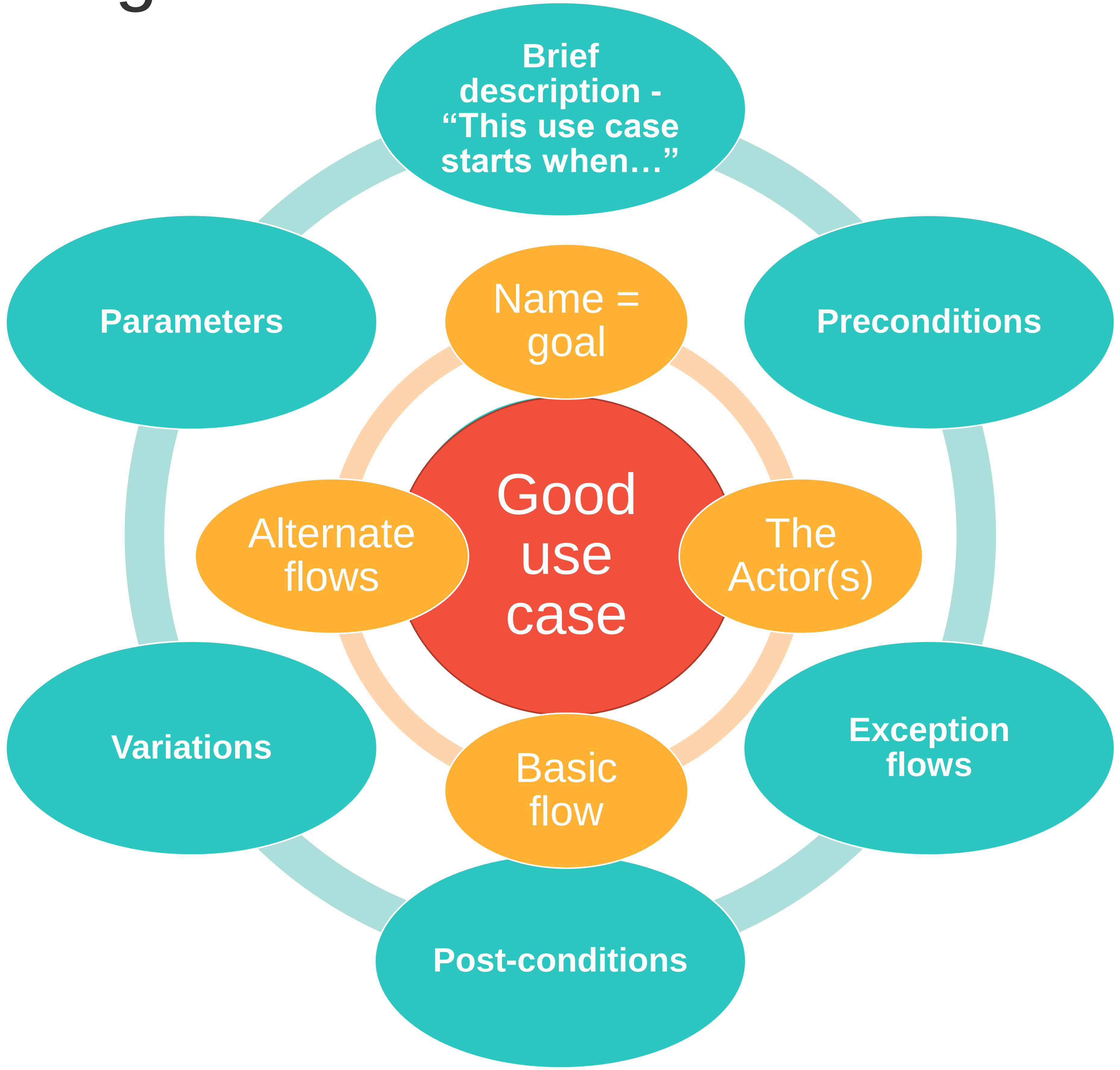


USE CASE 5	Buy Goods	
Description	Buyer issues request directly to our company, expects goods shipped and to be billed.	
Used by	Manage customer relationship (use case 2)	
Preconditions	We know Buyer, their address, and needed buyer information.	
Success End Condition	Buyer has goods, we have money for the goods.	
Failed End Condition	We have not sent the goods, Buyer has not spent the money.	
Actors	Buyer, any agent (or computer) acting for the customer. Credit card company, bank, shipping service	
Trigger	purchase request comes in.	
DESCRIPTION	**Step**	**Action**
	1	Buyer calls in with a purchase request
	2	Company captures buyer's name, address, requested goods, etc.
	3	Company gives buyer information on goods, prices, delivery dates, etc.
	4	Buyer signs for order.
	5	Company creates order, ships order to buyer.
	6	Company ships invoice to buyer.
	7	Buyers pays invoice.
EXTENSIONS	**Step**	**Branching Action**
	3a	Company is out of one of the ordered items: 3a1. Renegotiate order.
	4a	Buyer pays directly with credit card: 4a1. Take payment by credit card (use case 44)
	7a	Buyer returns goods: 7a. Handle returned goods (use case 105)
VARIATIONS		**Branching Action**
	1	Buyer may use phone in, fax in, use web order form, electronic interchange
	7	Buyer may pay by cash or money order check credit card

How should a good use case be cooked?

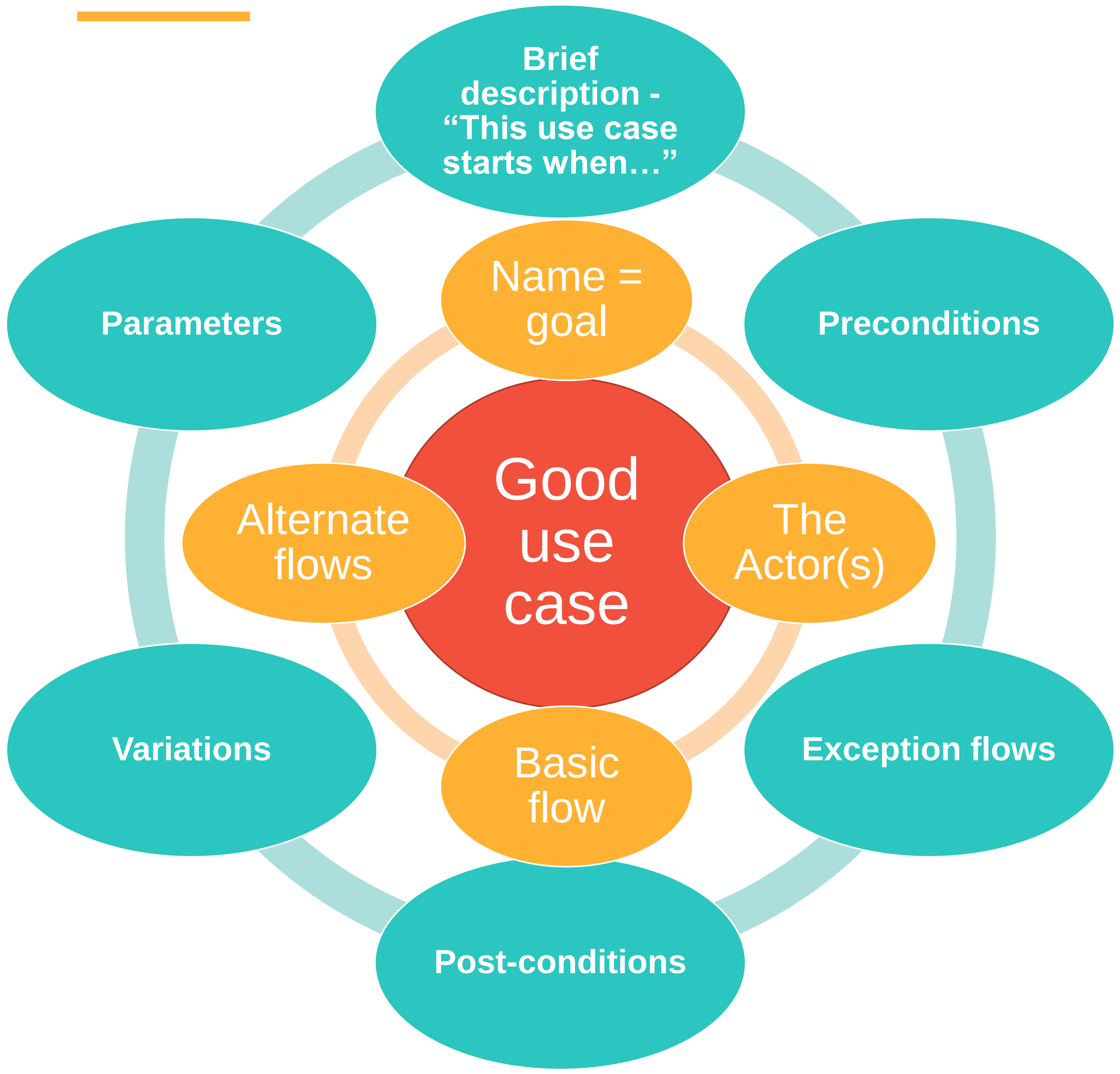


How should a good use case be cooked?



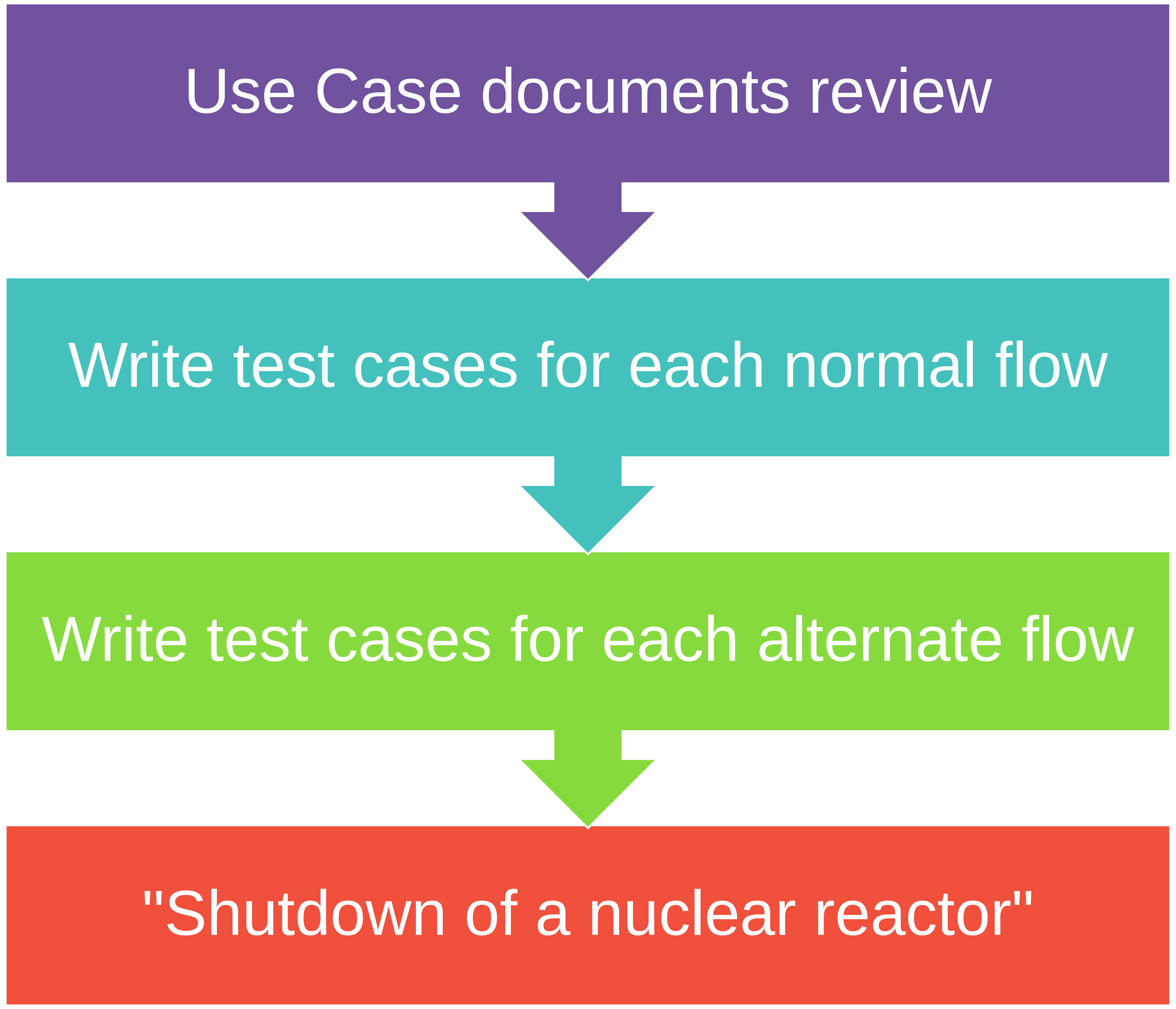
How should a good use case be cooked?

DataArt

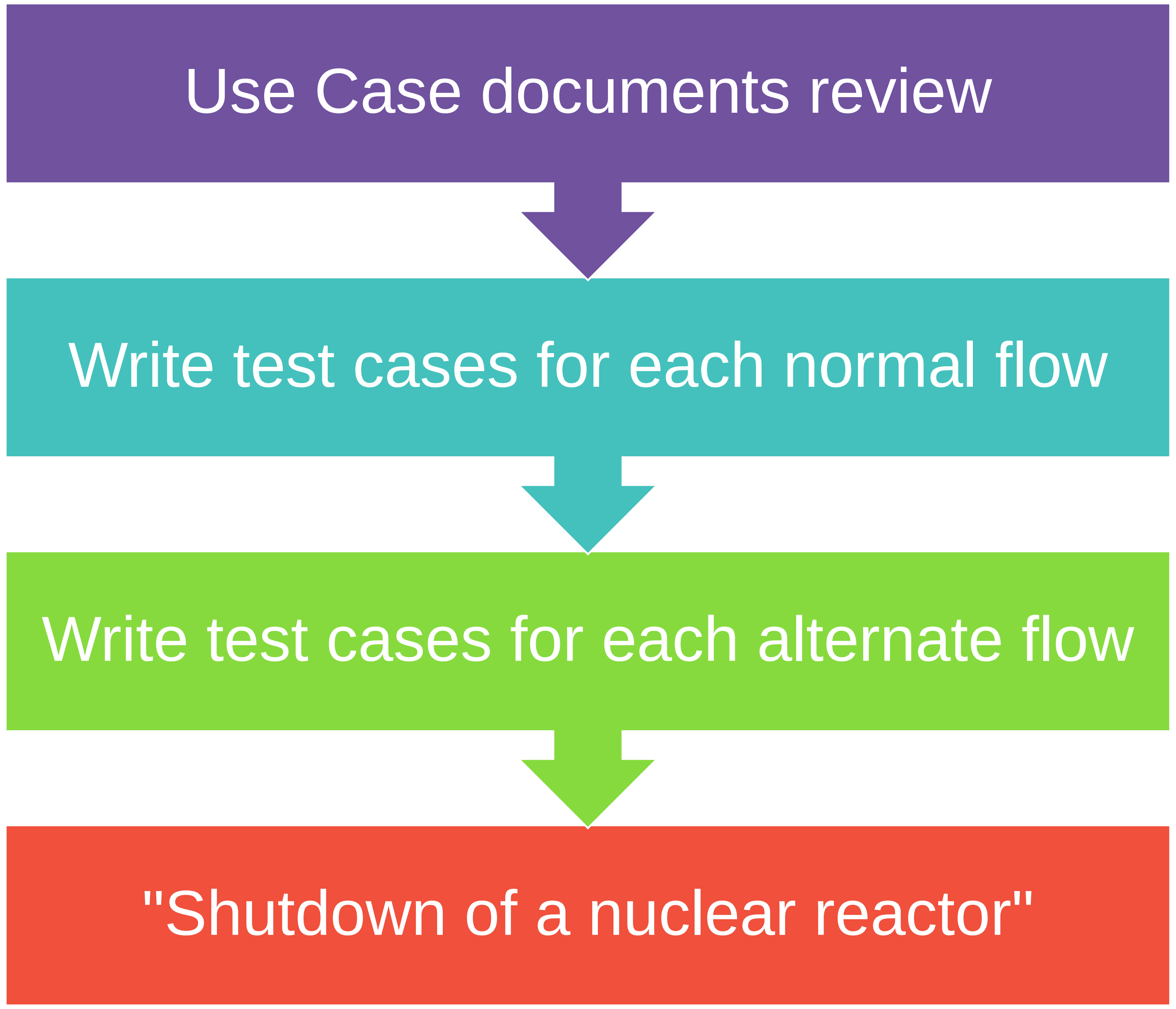


- ⛔ Use case is not based on user goal
- ⛔ System initiates use case
- ⛔ System acts as Terminator with own goals
- ⛔ Basketball instead of ping pong
- ⛔ Action to eliminate instead of the alternative
- ⛔ Too many details
- ⛔ Use Cases aren’t atomic
- ⛔ Alternative flows aren’t complete

Use case testing


Use Case documents review
Write test cases for each normal flow
Write test cases for each alternate flow
"Shutdown of a nuclear reactor"

Use case testing


Use Case documents review
Write test cases for each normal flow
Write test cases for each alternate flow
"Shutdown of a nuclear reactor"

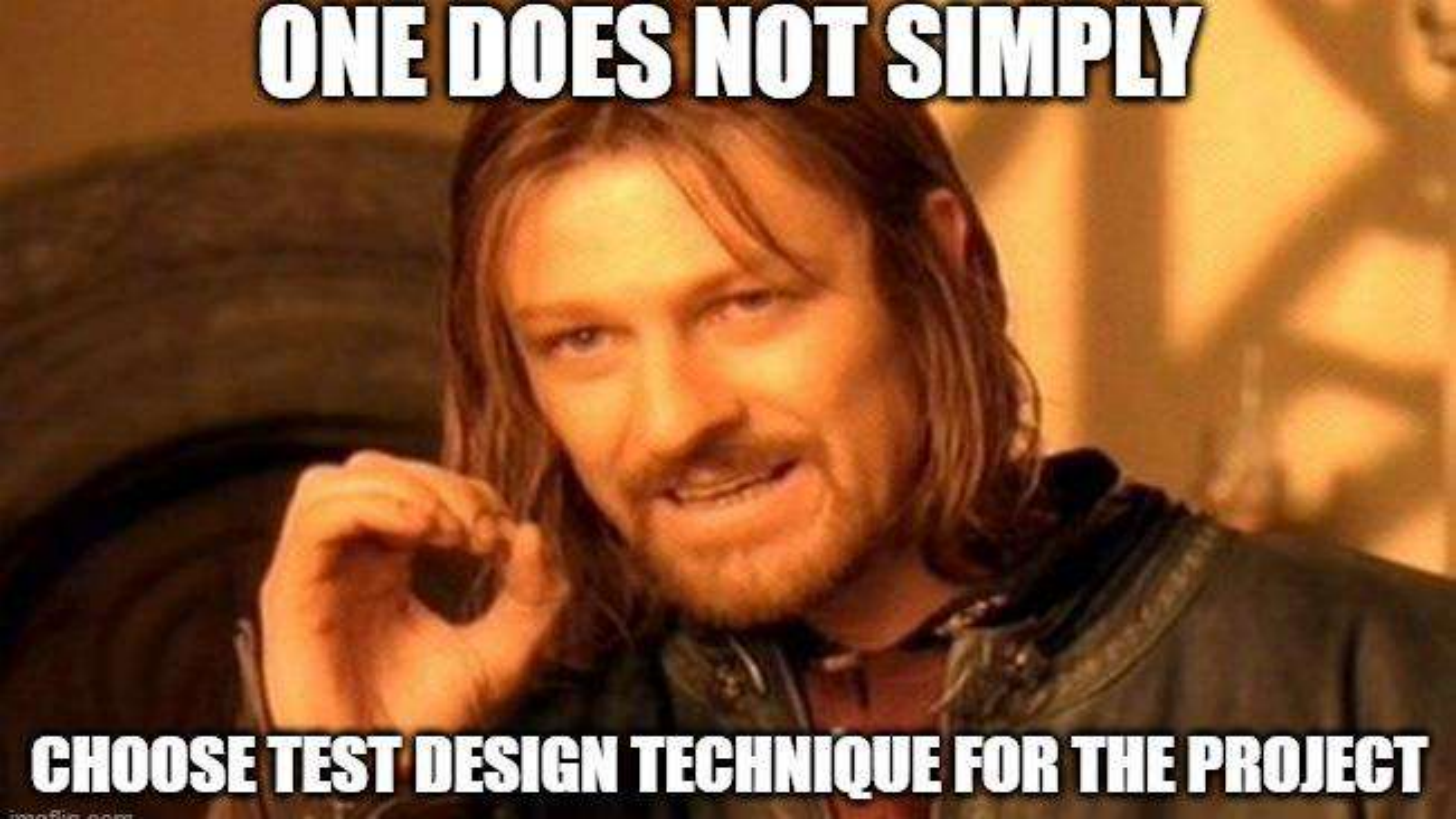
- Check boundary values and invalid values
- Try to perform the operation in an unusual order
- Transform precondition if it can happen
- If the flow has cycles, run it in a loop
- Reverse steps ordering

Pro and cons of use case testing

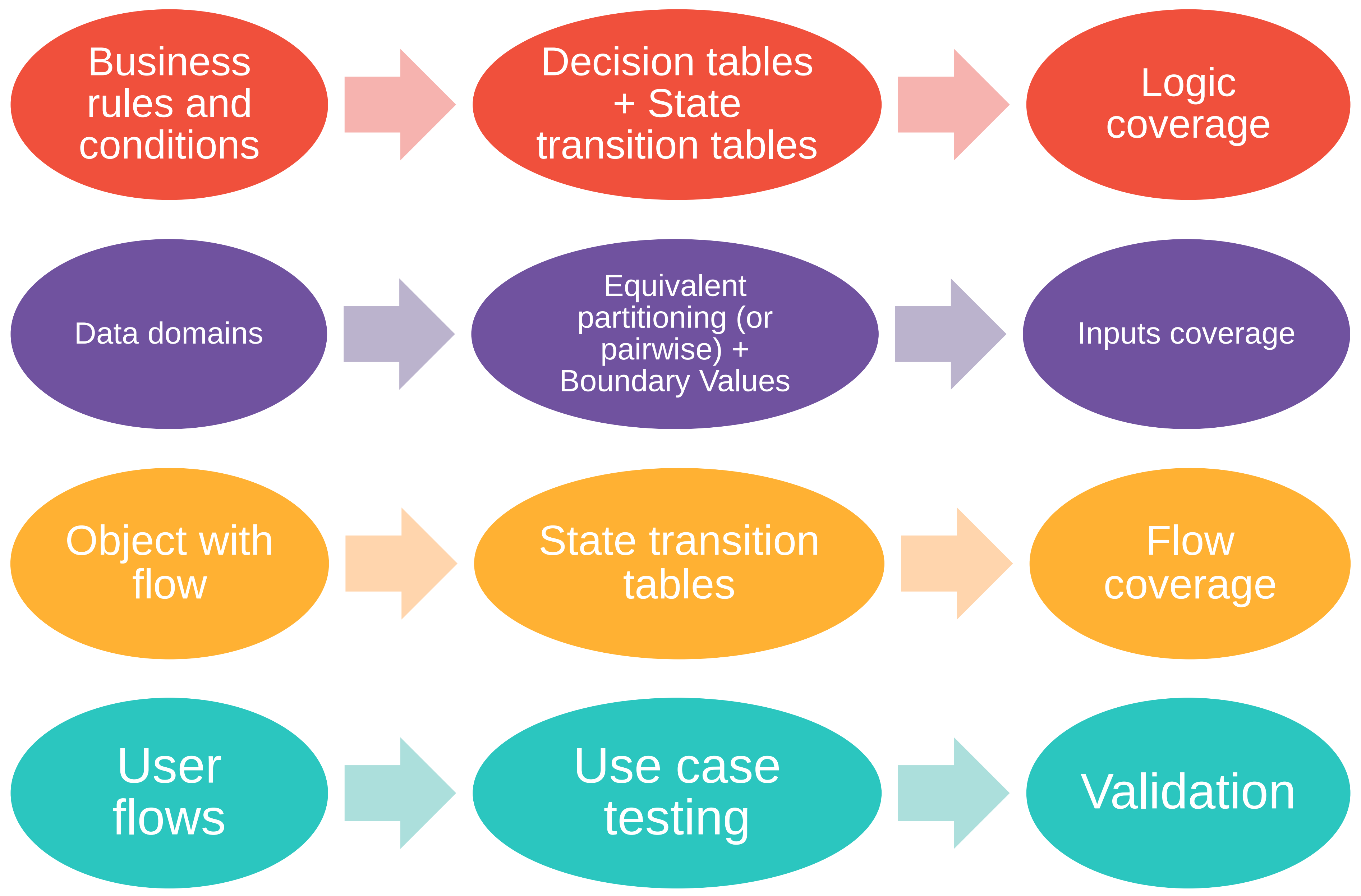
- Validation
- Simple usability testing
- Very effective in defining the scope of acceptance tests
- Can uncover integration defects

- Dependent on use case existence
- Dependent on use case quality
- Can’t measure the quality of the software
- No entire coverage of the user application

How to choose the best technique?


ONE DOES NOT SIMPLY
CHOOSE TEST DESIGN TECHNIQUE FOR THE PROJECT

Try to keep it simple



Try to keep it simple

Business rules and conditions → Decision tables + State transition tables → Logic coverage

Data domains → Equivalent partitioning (or pairwise) + Boundary Values → Inputs coverage

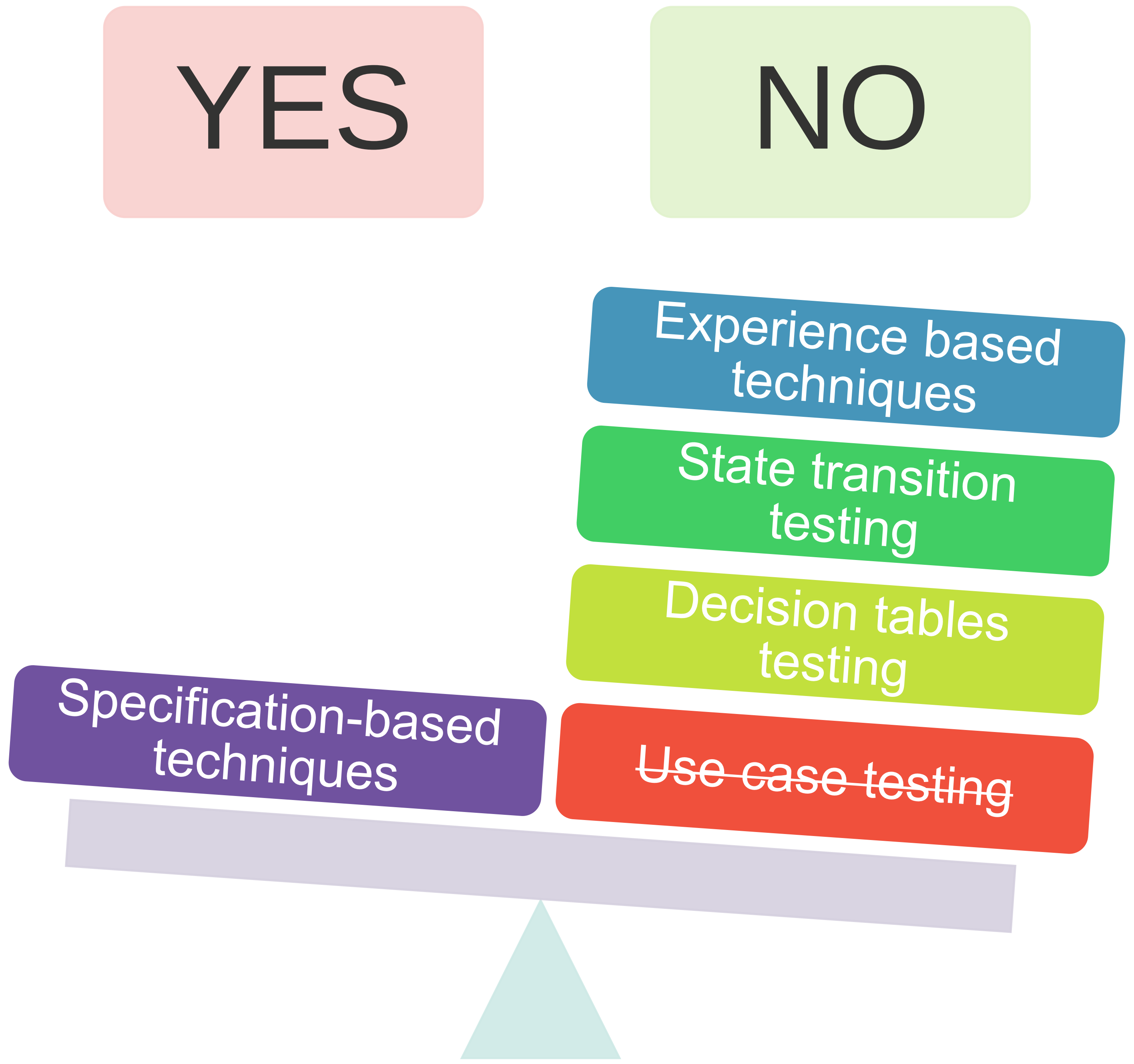
Object with flow → State transition tables → Flow coverage

User flows → Use case testing → Validation

What to consider?

- **Risk assessment**
- Test objectives
- Regulatory standards
- Time and budget
- Level of testing
- **Defects**
- **Documentation**
- Tester’s skill and knowledge
- Previous experience in types of defects tracked
- Type of system or software application
- Application development life cycle

Do you have good basis of testing?

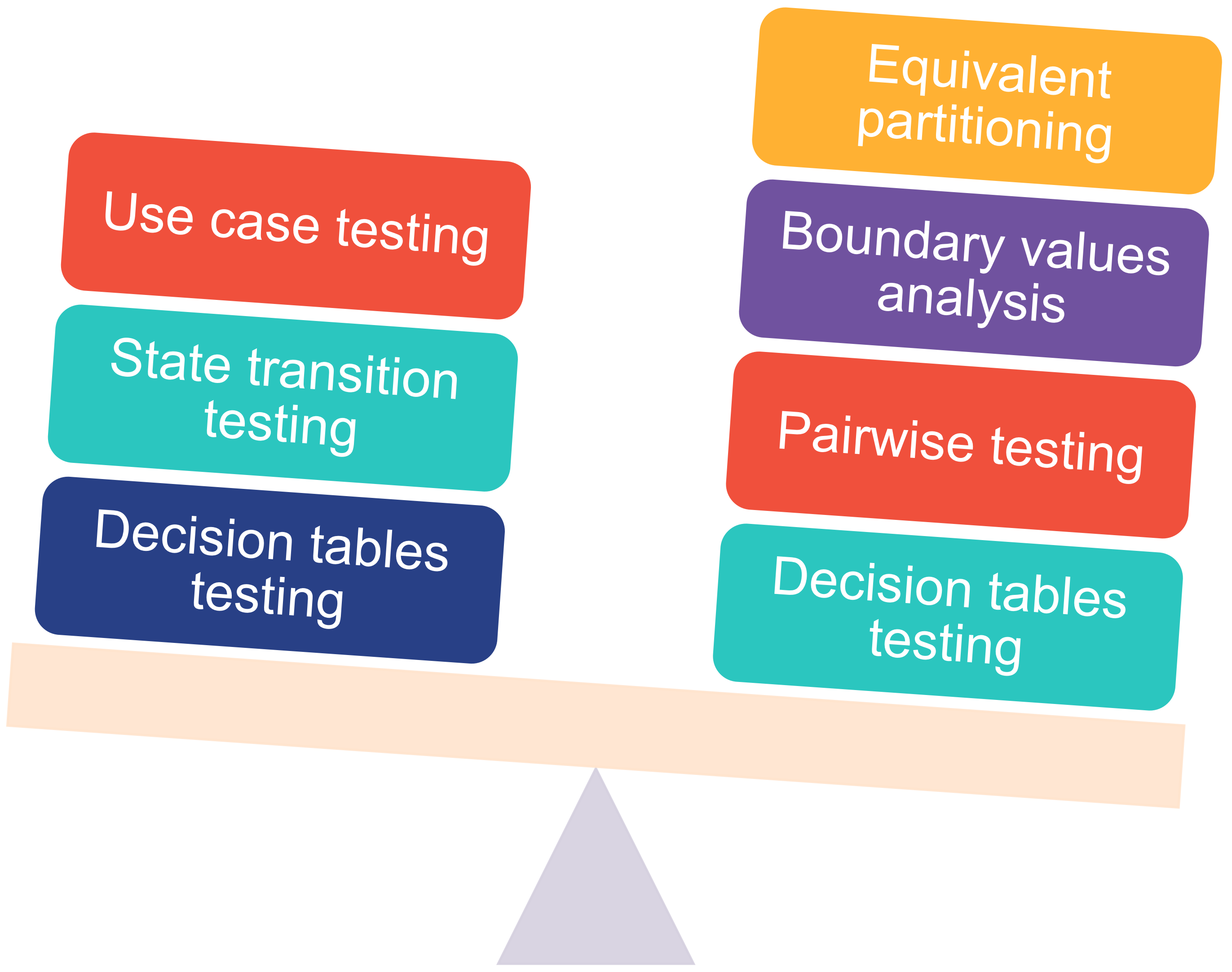


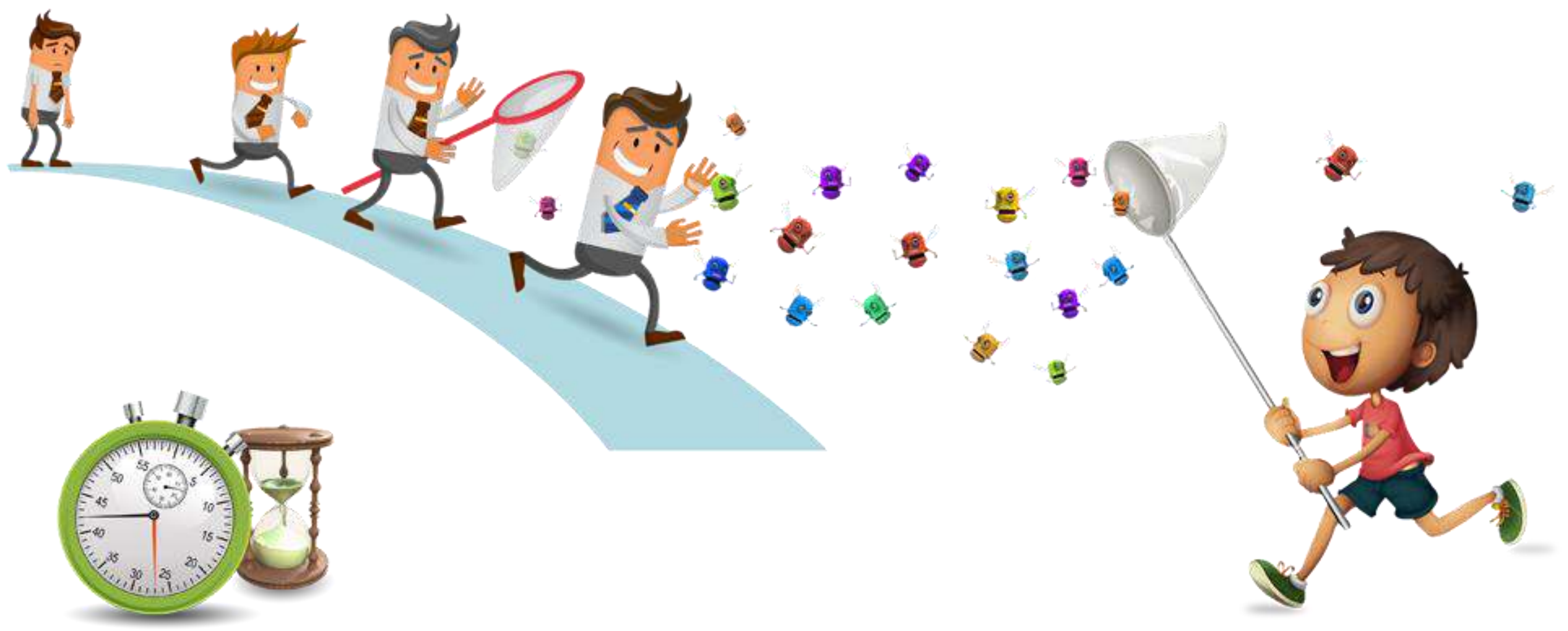
How detailed testing is needed?

DataArt

Surface testing

Detailed testing





Some good combinations

DataArt

- Decision tables + Boundary values analysis
- State transition tables + Boundary values analysis
- Equivalent partitioning / pairwise + Use case testing

- High risks → Equivalent partitioning or Decision tables
- Acceptance testing → Use cases testing + checking other criteria
- Complex business logic → Decision tables + State transition tables
- Complex test data → Equivalent partitioning + Boundary Values

Develop a testing approach

http://testingchallenges.thetestingmap.org/challenge6.php

Boundary Values Analysis training

Questions