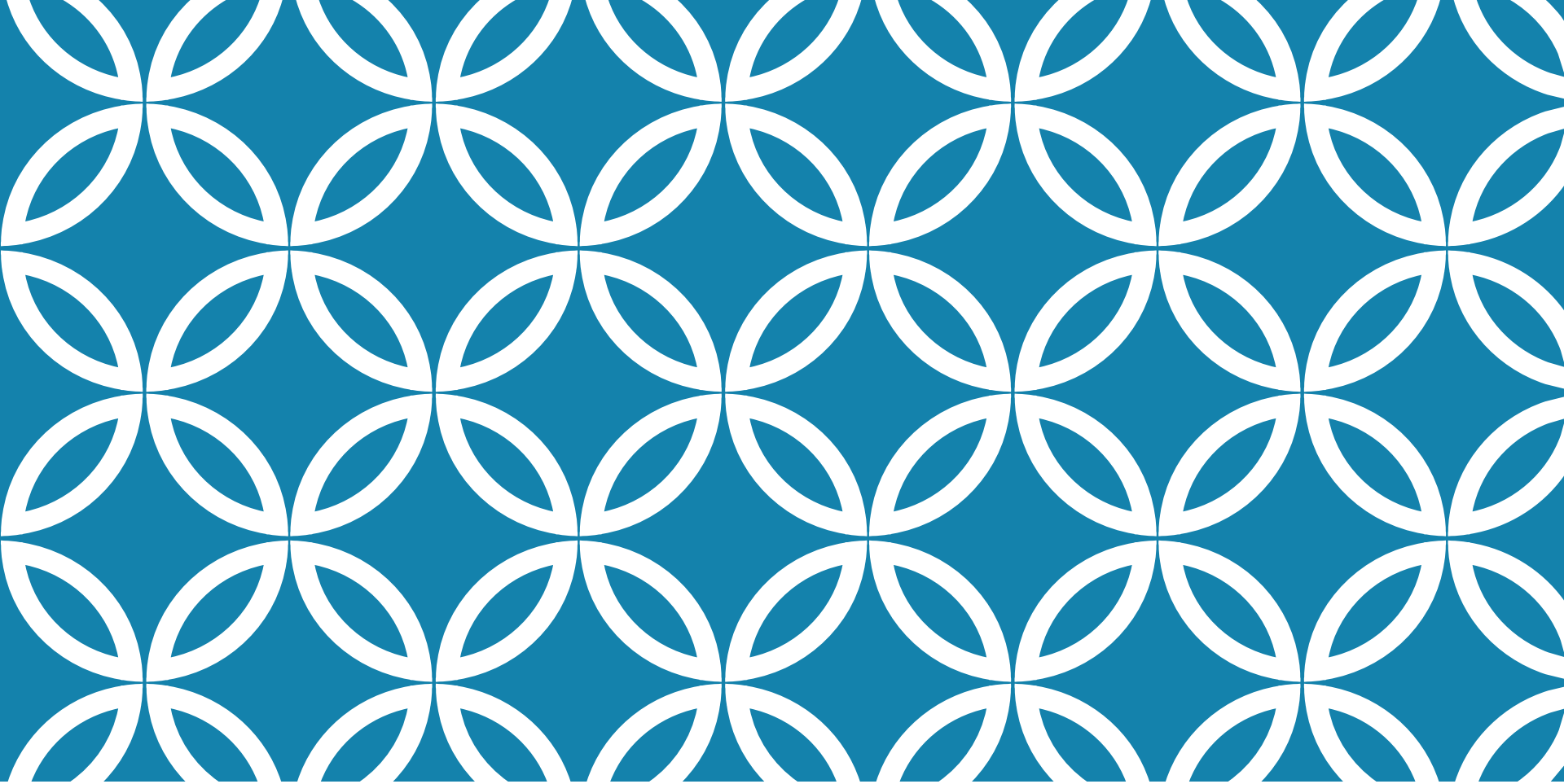


STATE TRANSITION TESTING TECHNIQUE

STATE ORIENTED AND STATELESS SYSTEMS

Software systems can be broadly classified into two groups, namely, *stateless and state-oriented systems. The actions of a stateless system do not depend on the* previous inputs to the system.

A compiler is an example of a stateless system because the result of compiling a program does not depend on the programs that had been previously compiled. The response of the system to the present input depends on the past inputs to the system in a state-oriented system.

A state-oriented system memorizes the sequence of inputs it has received so far in the form of a *state. A telephone switching system is an example of a state-oriented system.*

STEPS IN STATE DIAGRAM

1. understand State Transition diagrams and State tables
2. derive test cases from the State Transition diagrams and State Tables
3. Sequences – shortest, longest sequences
4. Null or Invalid transitions

State-transition diagrams are very useful for describing the behavior of a system and are part of the Software Design Document.

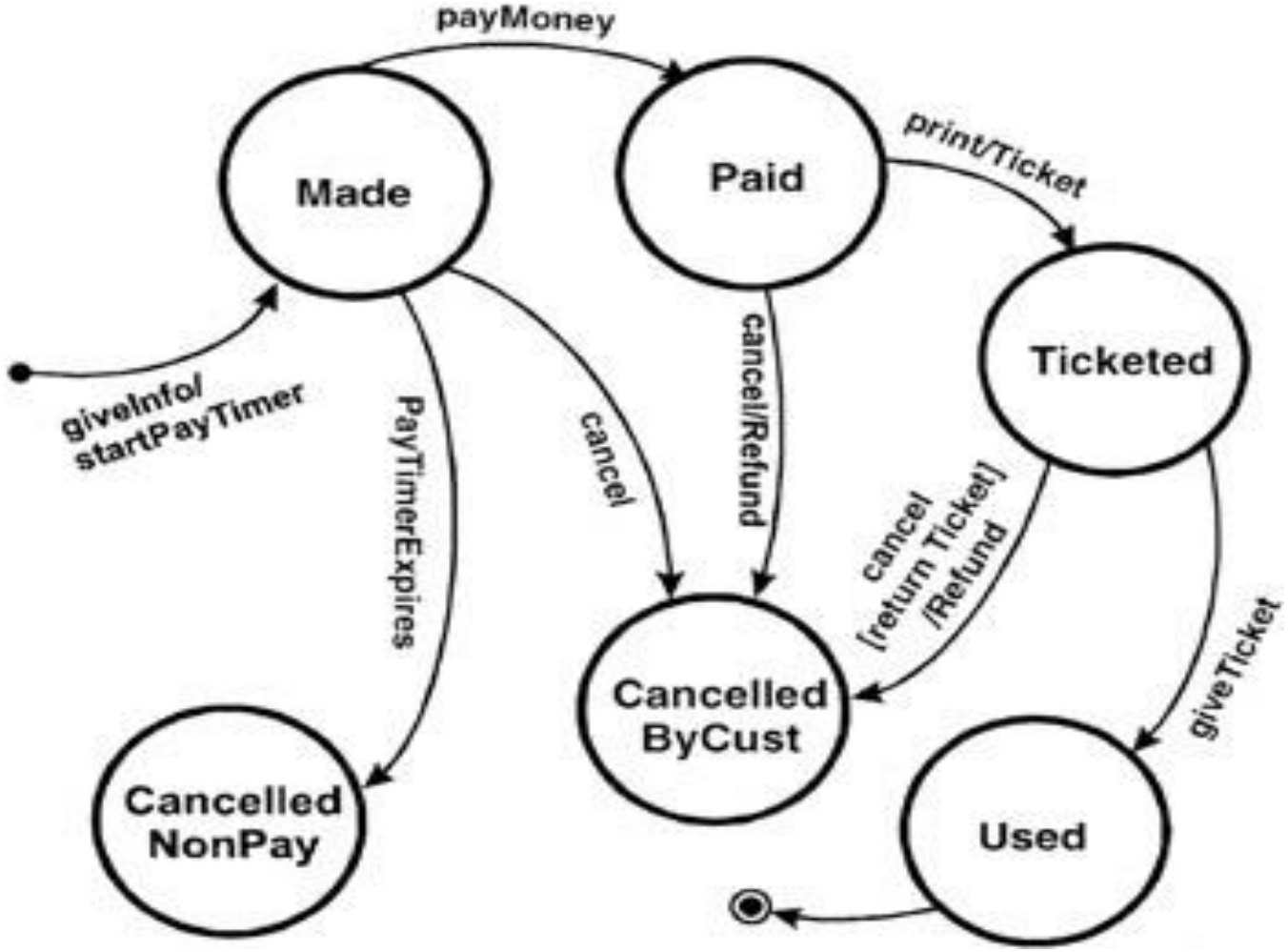
State (represented by a circle)—A state is a condition in which a system is waiting for one or more events. States "remember" inputs the system has received in the past and define how the system should respond to subsequent events when they occur.

Transition (represented by an arrow)—A transition represents a change from one state to another caused by an event.

Event (represented by a label on a transition)—An event is something that causes the system to change state. Generally, it is an event in the outside world that enters the system through its interface. Sometimes it is generated within the system such as **Timer expires** or **Quantity on Hand goes below Reorder Point.**

Action (represented by a command following a "/")—An action is an operation initiated because of a state change. It could be **print a Ticket, display a Screen, turn on a Motor,** etc. Often these actions cause something to be created that are outputs of the system. Note that actions occur on transitions between states. The states themselves are passive.

EXAMPLE


payMoney
Made
Paid
print/Ticket
Ticketed
giveInfo/
startPayTimer
PayTimerExpires
cancel
cancel/Refund
cancel
[return Ticket]
/Refund
giveTicket
Cancelled
ByCust
Cancelled
NonPay
Used

STATE-TRANSITION TABLE.

Current State	Event	Action	Next State
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	--	Ticketed
Ticketed	print	--	Ticketed
Ticketed	giveTicket	--	Used
Ticketed	cancel	Refund	Can-Cust
Ticketed	PayTimerExpires	--	Ticketed

STATE-TRANSITION TABLE.

Using a state-transition table can help detect defects in implementation that enable invalid paths from one state to another.

The disadvantage of such tables is that they become very large very quickly as the number of states and events increases.

In addition, the tables are generally sparse; that is, most of the cells are empty.

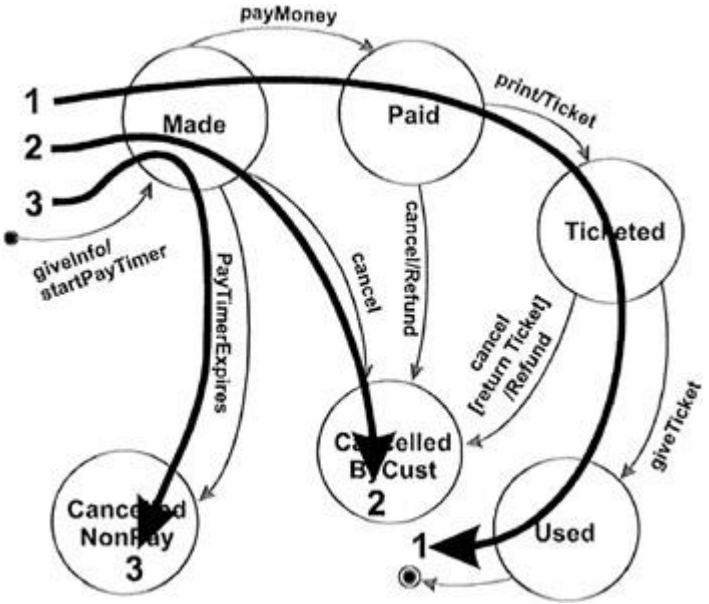
CREATING TEST CASES

Information in the state-transition diagrams can easily be used to create test cases. Four different levels of coverage can be defined:

1. Create a set of test cases such that **all states are "visited" at** least once under test.

Generally this is a weak level of test coverage.

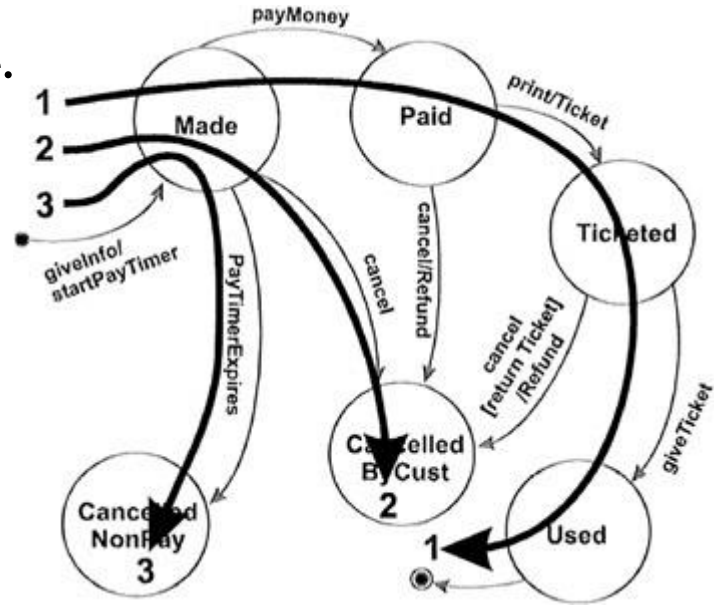
1.ALL STATES VISITED

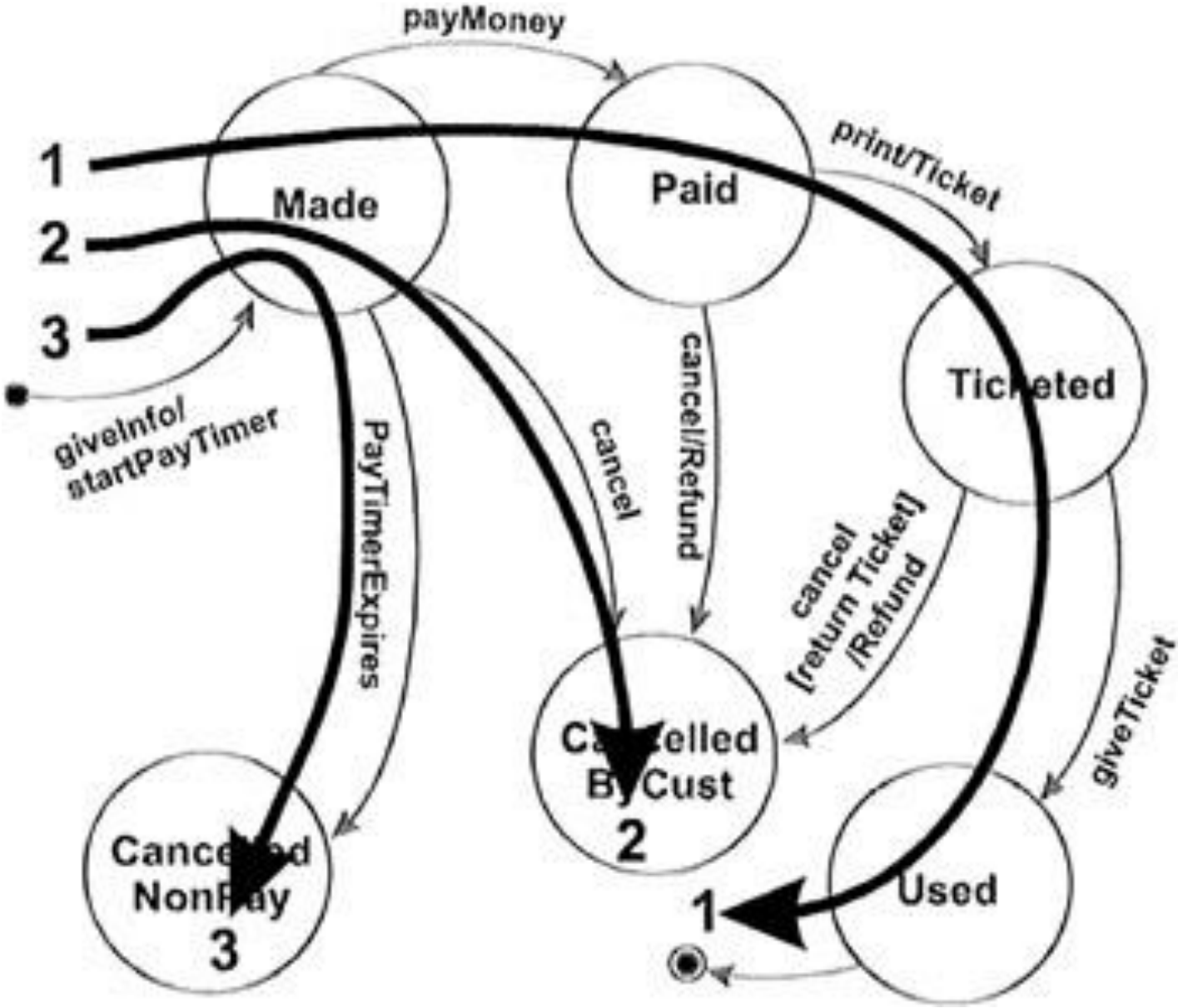


2. ALL EVENTS

2. Create a set of test cases such that **all events are triggered** at least once under test. Note that the test cases that cover each event can be the same as those that cover each state.

Again, this is a weak level of coverage.




payMoney
1
2
3
Made
Paid
print/Ticket
giveInfo/
startPayTimer
PayTimerExpires
cancel
cancel/Refund
cancel
[return Ticket]
/Refund
Ticketed
giveTicket
Cancelled
By Cust
2
Cancelled
NonPay
3
Used
1

3. ALL PATHS

Create a set of test cases such that **all paths** are executed at least once under test.

While this level is the most preferred because of its level of coverage, it may not be feasible

If the state-transition diagram has loops, then the number of possible paths may be infinite.

A→B

A→B→A

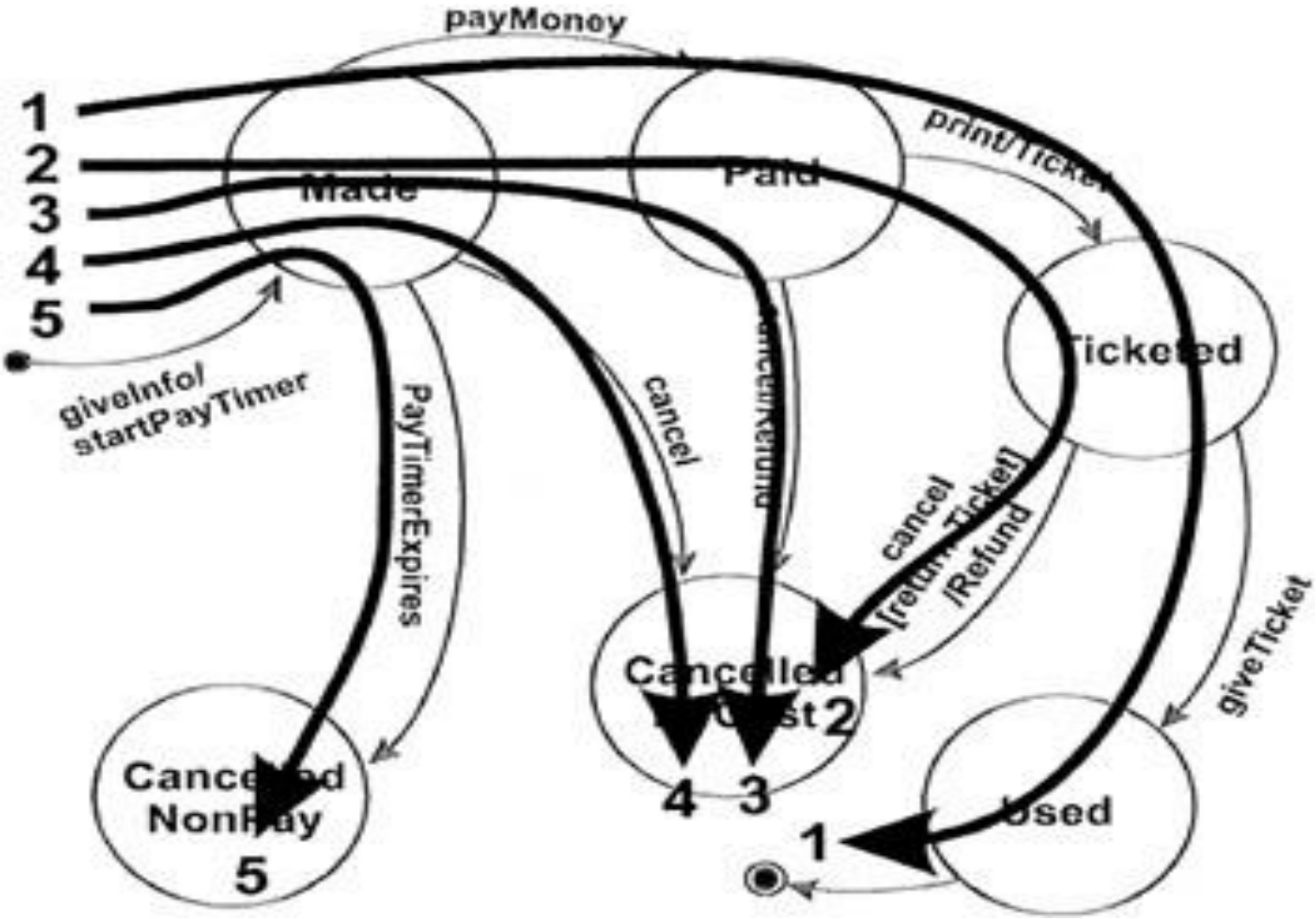
A→B→A→B→A→B

A→B→A→B→A→B→A

A→B→A→B→A→B→A→B→A→B

4. ALL TRANSITIONS

Create a set of test cases such that **all transitions are** exercised at least once under test. This level of testing provides a good level of coverage without generating large numbers of tests. This level is generally the one recommended.


payMoney
1
2
3
4
5
Made
Paid
printTicket
Ticketed
giveInfo/
startPayTimer
PayTimerExpires
cancel
cancel
/Refund
giveTicket
Cancelled
Cancelled
NonPay
5
4
3
2
1
Used

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.

SOURCE

https://flylib.com/books/en/2.156.1/state_transition_testing.html