

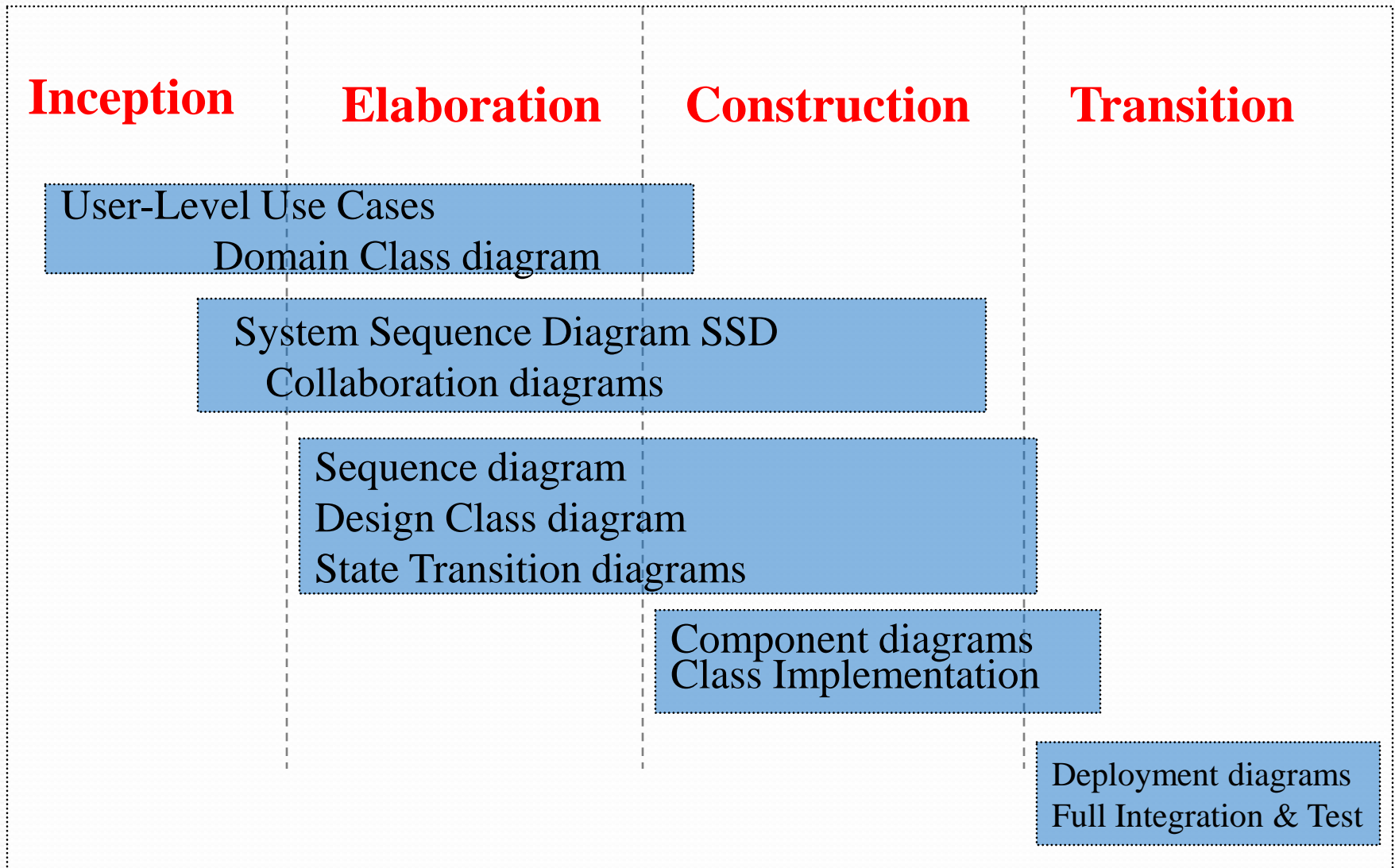
# **COMP 3711**

## **(OOA and OOD)**

### **Operation Contract**

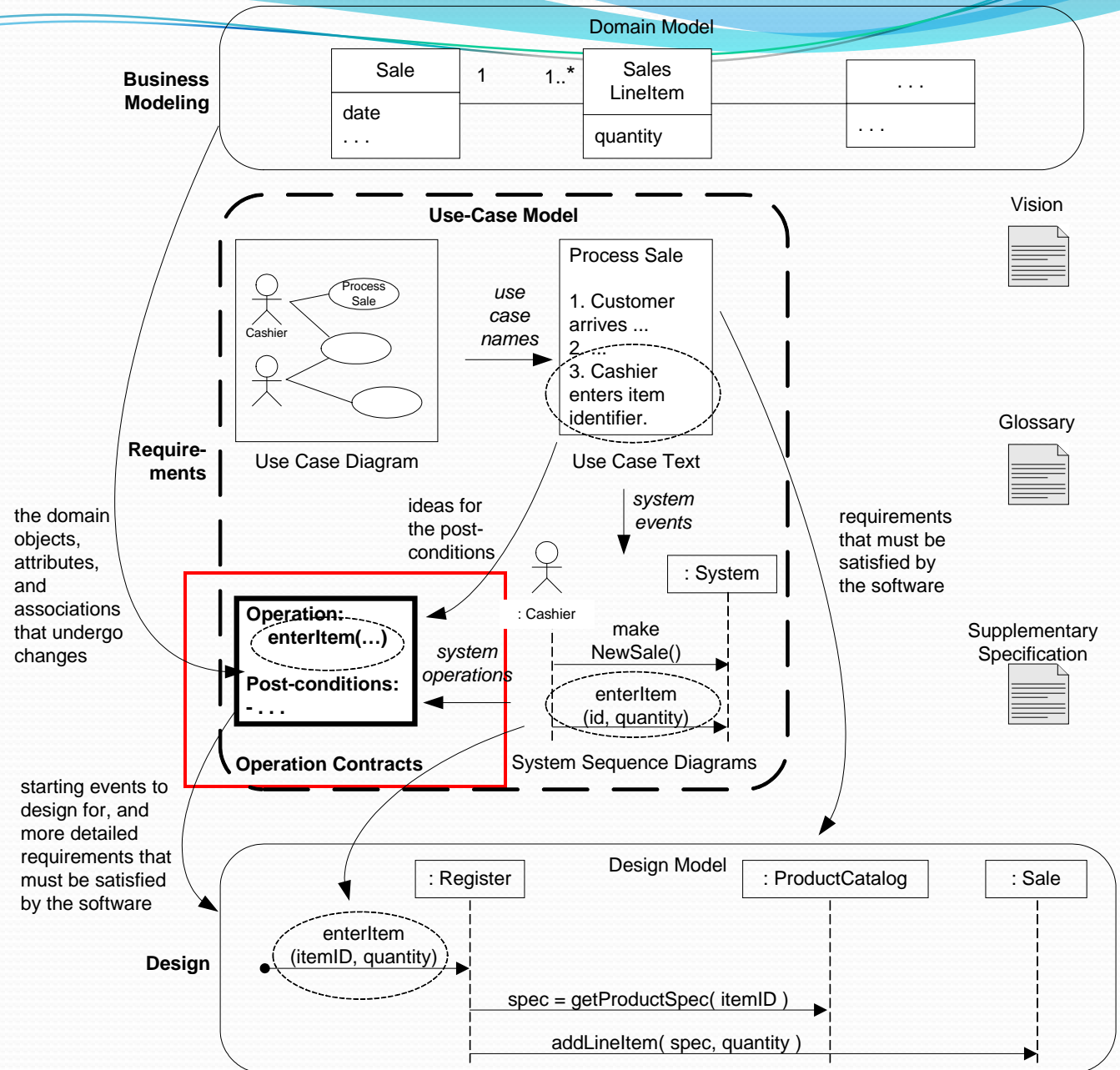
Larman Chapter 11

# UML And UP



# Operation Contracts

Part of requirements gathering with the SSD being the prime-input



# Operation Contracts History

- Link to use of OCL (Object Constraint Language) in pre and post conditions.
- Historic Contributors:
  - Tony Hoare (1960's) development of ALGOL 60 compiler, influenced by Bertrand Russell's *axiomatic theory and assertions* (pre- and post conditions) relative to the expected results.
  - IBM's PL/1 compiler (1970's), Peter Lucas' VDL (Vienna Definition Language).
  - Bertrand Meyer's Eiffel compiler (1980's) and DBC (Design By Contract approach)
  - Grady Booch's Booch Method (1990's)

# Operation Contracts - What

- The Domain Model consists of conceptual classes (real-world) objects in a domain of interest.
- Operation Contracts describe outcome of executing operations in terms of **state changes** to domain objects, after a system operation has executed.

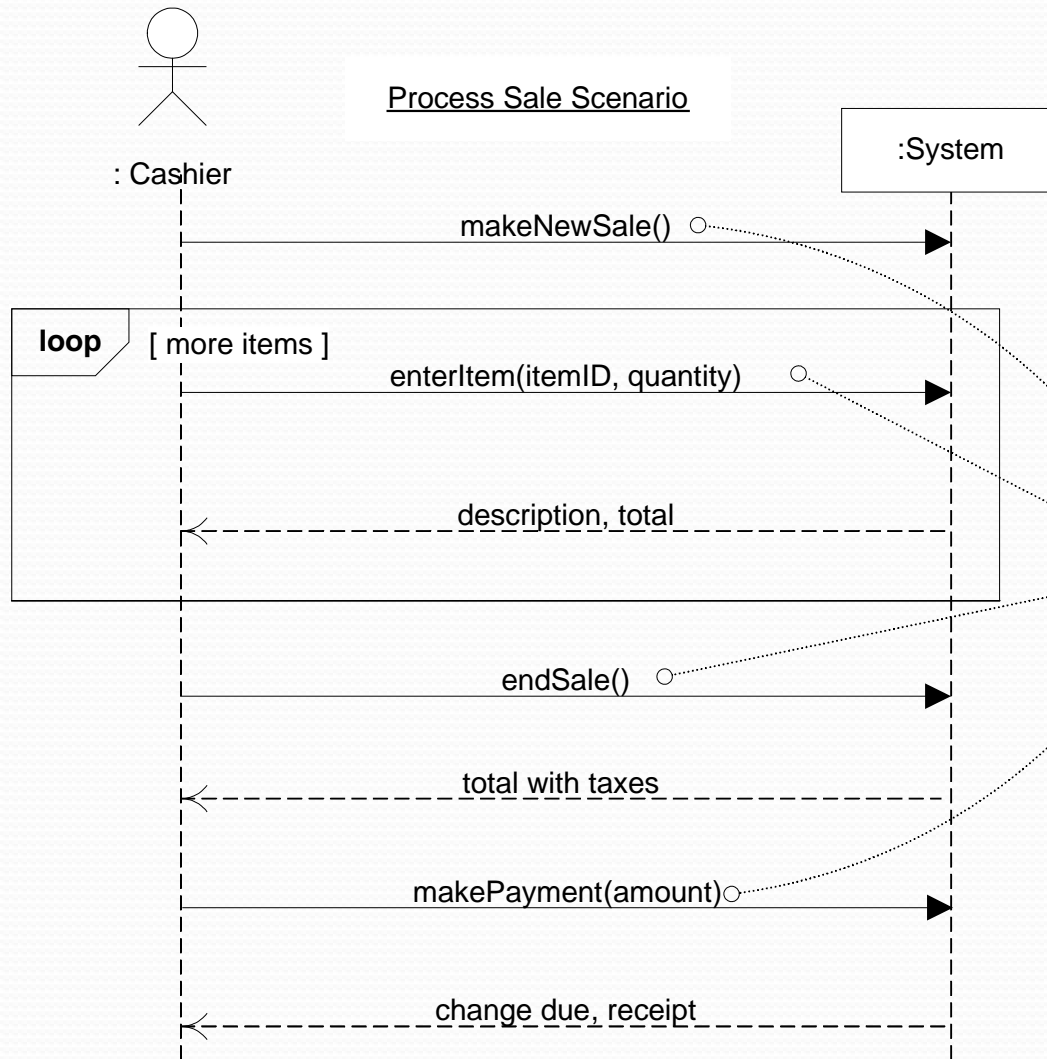
# Operation Contracts - Why

- Operation Contracts become necessary when Use Cases are insufficient for describing system behavior.
- In most instances, Operation Contracts may not be necessary.

# Operation Contracts - How

- Help define detailed system behavior on *system operations*
- Describe operations that the system, as a black box , offers in its public interface to handle incoming system events
- Implied by system (input) events which are described by Use Case Description

# Example: POS Input Events



**SSD shows  
system events**

these input system events  
invoke system operations

the system event *enterItem*  
invokes a system operation  
called *enterItem* and so forth

this is the same as in object-  
oriented programming when  
we say the message *foo*  
invokes the method (handling  
operation) *foo*

Larman Fig 11.2



# Operation Contract Sections

Operation:	Name of <b>operation and parameters</b>
Cross References:	(optional) <b>Use cases</b> this operation can occur within
Preconditions:	Noteworthy (non-trivial) <b>assumptions</b> about the state of the system or objects in the Domain Model before execution of the operation.
Postconditions:	The state of objects in the Domain Model after completion of the operation.

# Postconditions

- Changes in the state of objects in the Domain Model, which include:
  1. Instances created and deleted
  2. Attribute modification
  3. Associations (UML links) formed and broken
- Declarations about Domain Model objects that are true when the operation has finished – *after the smoke has cleared.*
- Expressed as declarative in past tense.

# Postconditions

- Describes **state changes** required of a system and **not how** they are going to be achieved.
- Focus analytically on **what must happen**, and **not how** it is to be accomplished.
- Unnecessary and undesirable to be overly verbose and detailed.
- **<Better>** → A SaleLineItem was created
- **<Worse>** → Create a SalesLineItem

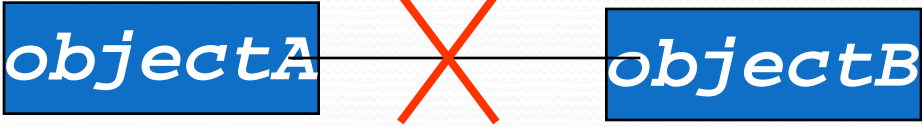
## Example Of POS Cash-only Process Scenario:

1. Customer arrives at a POS checkout with goods and/or services to purchase.
  2. Cashier starts a new sale.
  3. Cashier enters item identifier.
  4. System records sale line item and presents item description, price, and running total.
- Cashier repeats steps 3-4 until indicates done.
5. System presents total with taxes calculated.
  6. Cashier tells Customer the total, and asks for payment.
  7. Customer pays and System handles payment.

# Example Of *enterItem* Operation Contract

Operation:	enterItem(itemID : ItemID, quantity : integer)
Cross References:	Use Cases: Process Sale
Preconditions:	There is a sale underway.
Postconditions:	<ul style="list-style-type: none"><li>☀ A <i>SalesLineItem</i> instance <i>sli</i> was created (instance creation)</li><li>☀ <i>sli</i> was associated with the current <i>Sale</i> (association formed)</li><li>☀ <i>sli.quantity</i> became <i>actual quantity</i> (attribute modification)</li><li>☀ <i>sli</i> was associated with a <i>ProductSpecification</i>, based on <i>itemID</i> match (association formed)</li></ul>

# Broken Associations or Instance

- Broken Association 
  - When a loan is paid off.
  - When someone cancels their membership in a club
- Broken Instance
  - Removal of bankruptcy declaration after 7 years

# Operation Contracts Evolution

- Generating a complete detailed set of postconditions for system operation is not necessary during initial requirements work.
- It is all about iterative development:
  - discoveries during previous iteration enhances investigation and analysis of following iteration.

# Contracts & Domain Model updates

- It is common that during creation of Operation Contracts you discover need to record new conceptual classes, attributes, or associations in the Domain Model.
- Enhance your Domain Model as you make new discoveries while thinking through the Operation Contracts.



# Contracts vs. Use Cases

- Operation Contracts may not be needed if Use Cases provide most or all of the detail necessary for design.
- Operation Contracts, on the other hand, are handy where details and complexity of required state changes are awkward to capture in Use Cases:
  - Don't want to have to record a long list of creates, associations and state changes in the Use Case
  - Hotel Booking system: reserveRoom ...

# Operation Contracts Pitfalls

- Do not make Operation Contracts for every system operation of every Use Case.
- If you find yourself creating too many Operation Contracts, then the project may be suffering from the following ailments:
  1. Use cases are poorly done
  2. Not enough ongoing collaboration or access to a subject matter expert
  3. Your team is doing too much unnecessary documentation

# Contract pitfalls (continued ...)

- Remember to establish a memory between existing objects and those newly created by defining the forming of an association

## Postconditions:

- ☀ A *SalesLineItem* instance *sli* was created (instance creation)
- ☀ *sli* was associated with the current *Sale* (association formed)
- ☀ *sli.quantity* became *quantity* (attribute modification)
- ☀ *sli* was associated with a *ProductSpecification*, based on *itemID* match (association formed)

*It is necessary to define the forming of a new association*

# Operation Contracts Guidelines

How to make Operation Contracts?

1. Identify system operations from the SSDs (System Sequence Diagrams)
2. Construct Operation Contracts for operations that are complex, subtle in their results, or not clear in the use case
3. Use following categories to describe postconditions:
  - i. Instance creation and deletion
  - ii. Attribute modification
  - iii. Associations formed and broken

# POS example

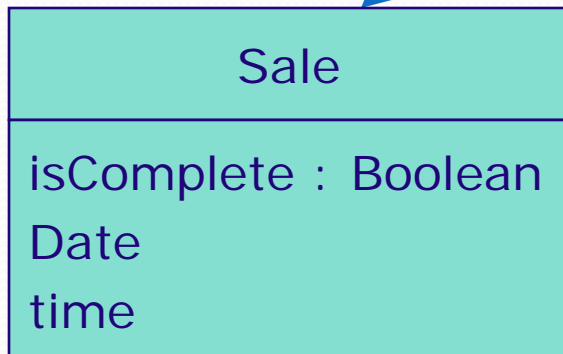
Contract C01: makeNewSale	
Operation:	makeNewSale()
Cross References:	Use Cases: Process Sale
Preconditions:	none
Postconditions:	<ul style="list-style-type: none"><li>• A <i>Sale</i> instance <i>s</i> was created (instance creation)</li><li>• <i>s</i> was associated with the <i>Register</i> (association formed)</li><li>• Attributes of <i>s</i> were initialized</li></ul>

# POS example

Contract C02: enterItem	
Operation:	enterItem(itemID : ItemID, quantity : integer)
Cross References:	Use Cases: Process Sale
Preconditions:	There is a sale underway.
Postconditions:	<ul style="list-style-type: none"><li>• A <i>SalesLineItem</i> instance <i>sli</i> was created (instance creation)</li><li>• <i>sli</i> was associated with the current <i>Sale</i> (association formed)</li><li>• <i>sli.quantity</i> became <i>quantity</i> (attribute modification)</li><li>• <i>sli</i> was associated with a <i>ProductSpecification</i>, based on <i>itemID</i> match (association formed)</li></ul>

# POS example

Contract C03: endSale	
Operation:	endSale()
Cross References:	Use Cases: Process Sale
Preconditions:	There is a sale underway
Postconditions:	☀ <i>Sale.isComplete</i> became <i>true</i> (attribute modification)



One way to represent endSale is to add an isComplete attribute in the Sale Class.

The new data item suggested in this Operation Contract is not yet represented in Domain Model.

# POS example

Contract C04: makePayment	
Operation:	makePayment ( amount:Money)
Cross References:	Use Cases: Process Sale
Preconditions:	There is a sale underway.
Postconditions:	<ul style="list-style-type: none"><li>• A <i>Payment</i> instance <i>p</i> was created (instance creation)</li><li>• <i>p.amount</i> tendered became <i>amount</i> (attribute modification)</li><li>• <i>p</i> was associated with the current Sale (association formed)</li><li>• The current Sale was associated with the Store (association formed); to add it to the historical log of completed sales)</li></ul>



# UP Phases - Iterative

- Inception:
  - Operation Contracts are typically not motivated during Inception, perhaps they are too detailed.
- Elaboration:
  - If used at all, most Operation Contracts are written during Elaboration, when most Use Cases are written.

# Summary

- Writing Operation Contracts leads to Domain Model updates.
- In UML, an operation is a specification of a transformation or query that an object may be called to execute.
- Most Operation Contracts will be written during elaboration, when most use cases are written. Only write Operation Contracts for the most difficult to understand or complicated system operations.

# Additional Exercise:

## New Systems Operations

- Let us create new system operations for a NExGen Point of Sale system to handle different forms of payment:
  - makeCreditPayment
  - makeCheckPayment

# New System Operations (2)

- The system event and operation for cash payment in an earlier iteration was simply `makePayment`.
- As the payments are of different types the operation is renamed as `makeCashPayment`.

# New System Operation Contracts

- The Operation Contracts bring value, on occasion, by their precise and detailed approach to identifying what happens when a complex operation is invoked on the system, in terms of state changes to objects defined in the Domain Model.
- System operation contracts are an optional requirements artifact that adds fine detail regarding the results of a system operation.

# Contract: makeCreditPayment

- Operation: makeCreditPayment  
(creditAccountNumber, expiryDate)
- Cross References: Use Cases: Process Sale
- Preconditions: An underway sale exists and all items have been entered.

# Contract: makeCreditPayment (2)

- Postconditions:
  - A credit payment pmt was created
  - pmt was associated with the current Sale  
sale a CreditCard cc was created.  
cc.number = creditAccountNumber,  
cc.expiryDate = expiryDate
  - cc was associated with pmt

# Contract: makeCreditPayment (3)

- a CreditPaymentRequest cpr was created
- pmt was associated with cpr
- a ReceivableEntry re was created
- re was associated with the external AccountsReceivable
- Sale was associated with the Store as a complete sale



# makeCreditPayment

- There exists a postcondition indicating the association of a new receivable entry in accounts receivable.
- Although, this responsibility is outside the boundaries of the NextGen system, the accounts receivable system is within the control of the business.

# makeCreditPayment (2)

- The statement has thus been added as a correctness check.
- During testing, it is clear from this post-condition that the accounts receivable system should be tested for the presence of a new receivable entry.

# Contract:makeCheckPayment

- Operation: makeCheckPayment  
(driversLicenseNumber)
- Cross References: Use Cases: Process Sale
- Preconditions: An underway sale exists and all items have been entered

# Contract: makeCheckPayment (2)

- Postconditions:
  - a CheckPayment pmt was created
  - pmt was associated with the current Sale sale
  - a DriversLicense dl was created,  
dl.number = driversLicenseNumber
  - dl was associated with pmt

# Contract:makeCheckPayment (3)

- a CheckPaymentRequest cpr was created
- pmt was associated with cpr
- sale was associated with the Store as a completed sale

# Summary

- Contracts describe detailed system behavior in terms of state changes to objects in the Domain Model after a system operation.
- Contracts have sections of Operations, Cross references, Preconditions and Postconditions. **Postconditions are the most important section.**
- Postconditions describe changes in the state of objects in the Domain Model.
- Domain Model state changes include instances created, associations formed or broken, and attributes changed.