Transition to design

- **System sequence diagram**
  - An analysis model for identifying the messages (events) actors send to the system

- **System behaviors and contracts**

- **Sequence diagram**
  - A design model. One sequence diagram for each contract (or use case).

- **Collaboration diagram**
  - Semantically equivalent to sequence diagram
System sequence diagram (I)

• Draw system sequence diagram.
  – Concentrate in the use case on events that external actor generates. These will be the external trigger for system actions.

• Translate the expanded use case into a real use case
  – UI decision
  – Database
System sequence diagram (II)

Figure 13.1 System sequence diagram for the Buy Items use case.
System events and operations

Figure 13.2 System events initiate system operations.

Figure 13.4 System operations recorded in a type named System.
Another SSD example

Take A Test

Student

login(ID, passwd)

selectATest()

doTest()

endTest()

System as a block box

System

+login( ID: , passwd: )
+selectATest()  
+doTest( )
+endTest( )

System event

lifeline
How to make a SSD

To make system sequence diagrams for the typical course of events for a use case:

1. Draw a line representing the system as a black box.
2. Identify each actor that directly operates on the system. Draw a line for each such actor.
3. From the use case typical course of events text, identify the system (external) events that each actor generates. Illustrate them on the diagram.
4. Optionally, include the use case text to the left of the diagram.
SSD with use case (I)

**USE CASE: BUY ITEMS**

Typical Course Of Events

1. This use case begins when a Customer arrives at the POST checkout with items to purchase.

2. The Cashier records the universal product code (UPC) from each item. If there is more than one of the same item, the Cashier can enter the quantity as well.

3. System determines the item price and adds the item information to the running sales transaction. The description and price of the current item are displayed.

4. and so on.

Figure 13.5 System sequence diagrams are derived from use cases.
SSD with use case (II)

For all items, the Cashier records the UPC and quantity.

On completion of item entry, the Cashier indicates to the POST that the sale is complete.

The Cashier tells the Customer the total, and the Customer gives a payment to the Cashier.

The Cashier records the cash received amount.

Figure 13.8 System sequence diagram with use case text.
Naming system events and operations

• “strive for the highest level or ultimate goal in naming the operation”

Figure 13.7 Choose event and operation names at an abstract level.
Contracts

- A contract is a document that describes what an operation commits to achieve.
- Declarative in style; *what* rather than *how*.
- Expressed in pre- and post-conditions and invariants.
- Contracts should be written for each system operation.
### Contract

<table>
<thead>
<tr>
<th>Name:</th>
<th>Name of operation, and parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsibilities:</td>
<td>An informal description of the responsibilities this operation must fulfill.</td>
</tr>
<tr>
<td>Type:</td>
<td>Name of type (concept, software class, interface)</td>
</tr>
<tr>
<td>Cross References:</td>
<td>System function reference numbers, use cases, etc.</td>
</tr>
<tr>
<td>Notes:</td>
<td>Design notes, algorithms, and so on.</td>
</tr>
<tr>
<td>Exceptions:</td>
<td>Exceptional cases.</td>
</tr>
<tr>
<td>Output:</td>
<td>Non-UI outputs, such as messages or records that are sent outside of the system.</td>
</tr>
<tr>
<td>Pre-conditions:</td>
<td>Assumptions about the state of the system before execution of the operation.</td>
</tr>
<tr>
<td>Post-conditions:</td>
<td>The state of the system after completion of the operation. Discussed in detail in a following section.</td>
</tr>
</tbody>
</table>
How to make a contract

Apply the following advice to create contracts.

To make contracts for each use case:
1. Identify the system operations from the system sequence diagrams.
2. For each system operation, construct an contract.
3. Start by writing the Responsibilities section, informally describing the purpose of the operation.
4. Then complete the Post-conditions section, declaratively describing the state changes that occur to objects in the conceptual model.
5. To describe the post-conditions, use the following categories:
   - Instance creation and deletion.
   - Attribute modification.
   - Associations formed and broken.
From use case to contract
Post-conditions

• Related to conceptual model
  – Instances created, association formed, and attributes modified are from the conceptual model

• Post-conditions are better phrased as passive past tense sentences
  – A SalesLineItem was created vs. Create a SalesLineItem
Pre-conditions

• State of the system at the beginning of the operation
  – Thing that are important to test in software at some point during execution of the operation
  – Things that will not be tested, nut upon which the success of the operation hinges.
Contract style

• Start with operation name, then responsibilities, next post-conditions, and pre-conditions last.

• Followed by notes and exceptions
Contract examples (I)

Contract

Name: enterItem
   (upc : number, quantity : integer)

Responsibilities: Enter (record) sale of an item and add it to the sale.
Display the item description and price.

Type: System

Cross References: System Functions: RI.1, RI.3, RI.9

Notes: Use Cases: Buy Items

Exceptions: If the UPC is not valid, indicate that it was an error.

Output:
Pre-conditions: UPC is known to the system

Post-conditions:

- If a new sale, a Sale was created (instance creation).
- If a new sale, the new Sale was associated with the POST (association formed).
- A SalesLineItem was created (instance creation).
- The SalesLineItem was associated with the Sale (association formed).
- SalesLineItem.quantity was set to quantity (attribute modification).
- The SalesLineItem was associated with a ProductSpecification, based on UPC match (association formed).
## Contract examples (II)

<table>
<thead>
<tr>
<th><strong>Name:</strong></th>
<th>endSale()</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Responsibilities:</strong></td>
<td>Record that it is the end of entry of sale items, and display sale total.</td>
</tr>
<tr>
<td><strong>Type:</strong></td>
<td>System</td>
</tr>
<tr>
<td><strong>Cross References:</strong></td>
<td>System Functions: R1.2</td>
</tr>
<tr>
<td><strong>Use Cases:</strong></td>
<td>Buy Items</td>
</tr>
<tr>
<td><strong>Notes:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Exceptions:</strong></td>
<td>If a sale is not underway, indicate that it was an error.</td>
</tr>
<tr>
<td><strong>Output:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Pre-conditions:</strong></td>
<td>UPC is known to the system.</td>
</tr>
<tr>
<td><strong>Post-conditions:</strong></td>
<td></td>
</tr>
</tbody>
</table>

- `Sale.isComplete` was set to `true` (attribute modification).
Contract examples (III)

<table>
<thead>
<tr>
<th>Contract</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name:</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Responsibilities:</td>
</tr>
<tr>
<td>Type:</td>
</tr>
<tr>
<td>Cross References:</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Notes:</td>
</tr>
<tr>
<td>Exceptions:</td>
</tr>
<tr>
<td>Output:</td>
</tr>
<tr>
<td>Pre-conditions:</td>
</tr>
<tr>
<td>Post-conditions:</td>
</tr>
</tbody>
</table>
Real use case (I)

<table>
<thead>
<tr>
<th>Use case:</th>
<th>Buy Items-version 1 (Cash only)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actors:</td>
<td>Customer (initiator), Cashier</td>
</tr>
<tr>
<td>Purpose:</td>
<td>Capture a sale and its cash payment.</td>
</tr>
<tr>
<td>Overview:</td>
<td>A Customer arrives at a checkout with items to purchase. The Cashier records the purchase items and collects a cash payment. On completion, the Customer leaves with the items.</td>
</tr>
<tr>
<td>Type</td>
<td>primary and real</td>
</tr>
<tr>
<td>Cross References:</td>
<td>Functions: R1.1, R1.2, R1.3, R1.7, R1.9, R2.1,</td>
</tr>
</tbody>
</table>

Figure 16.1 Window-1.
### Typical Course of Events

<table>
<thead>
<tr>
<th>Actor Action</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. This use case begins when a Customer arrives at the POST checkout with items to purchase.</td>
<td>3. Adds the item information to the running sales transaction.</td>
</tr>
<tr>
<td>2. For each item, the Cashier types in the Universal Product Code (UPC) in A of Window-1. If there is more than one of an item, the quantity may optionally be entered in E. They press B after each item entry.</td>
<td>5. Calculates and displays the sale total in C.</td>
</tr>
<tr>
<td>4. On completion of item entry, the Cashier indicates to the POST that item entry is complete by pressing widget I.</td>
<td>6. ...</td>
</tr>
<tr>
<td></td>
<td>The description and price of the current item are displayed in B and F of Window1.</td>
</tr>
</tbody>
</table>
Collaboration diagram

- Elements
  - Objects
  - Links
  - Messages
Sequence diagram

- Elements
  - Objects with life line
  - Messages
Comparison

• Collaboration and sequence diagrams are semantically equivalent
• Collaboration is good at showing objects are statically connected.
  – It is similar to the conceptual diagram derived during analysis and class diagram derived during design.
• Sequence diagram is good at showing the order in which things occur.
• Both models quickly lose their advantage for expressing complex conditional and looping logic.
Collaboration diagram: `enterItem`

Figure 19.6 Sale creation.
Collaboration diagram: enterItem

Figure 19.7 The enterItem collaboration diagram.
Further Readings

- Larman 1st ed., Chapters 13-17
- Fowler/Distilled, Chapter 4.