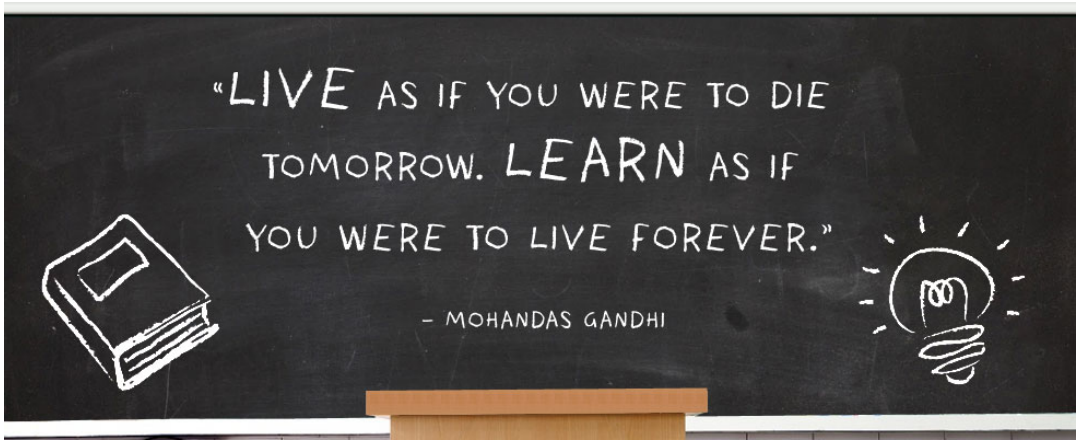
Vorlesung Softwaretechnik
SS 2019 / Teil II

Prof. Dr.-Ing. Samuel Kounev
M.Sc. Johannes Grohmann



4. System Analysis

with System Sequence Diagrams & Operation Contracts

Prof. Dr.-Ing. Samuel Kounev

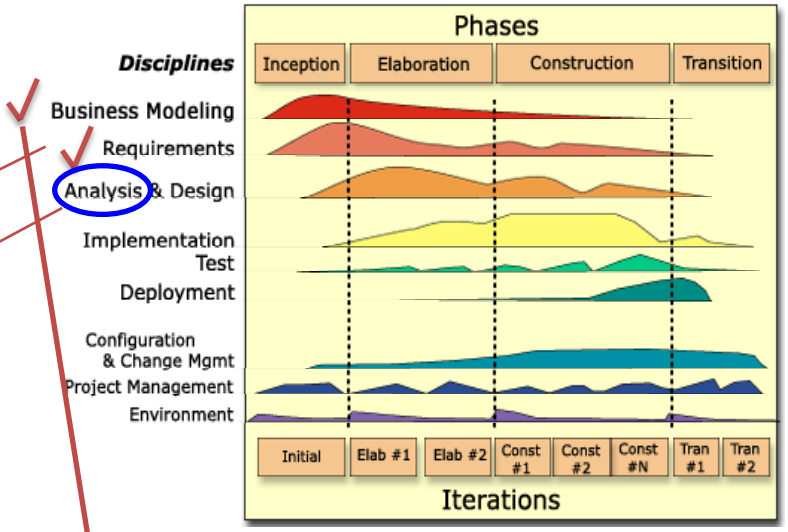
Acknowledgements

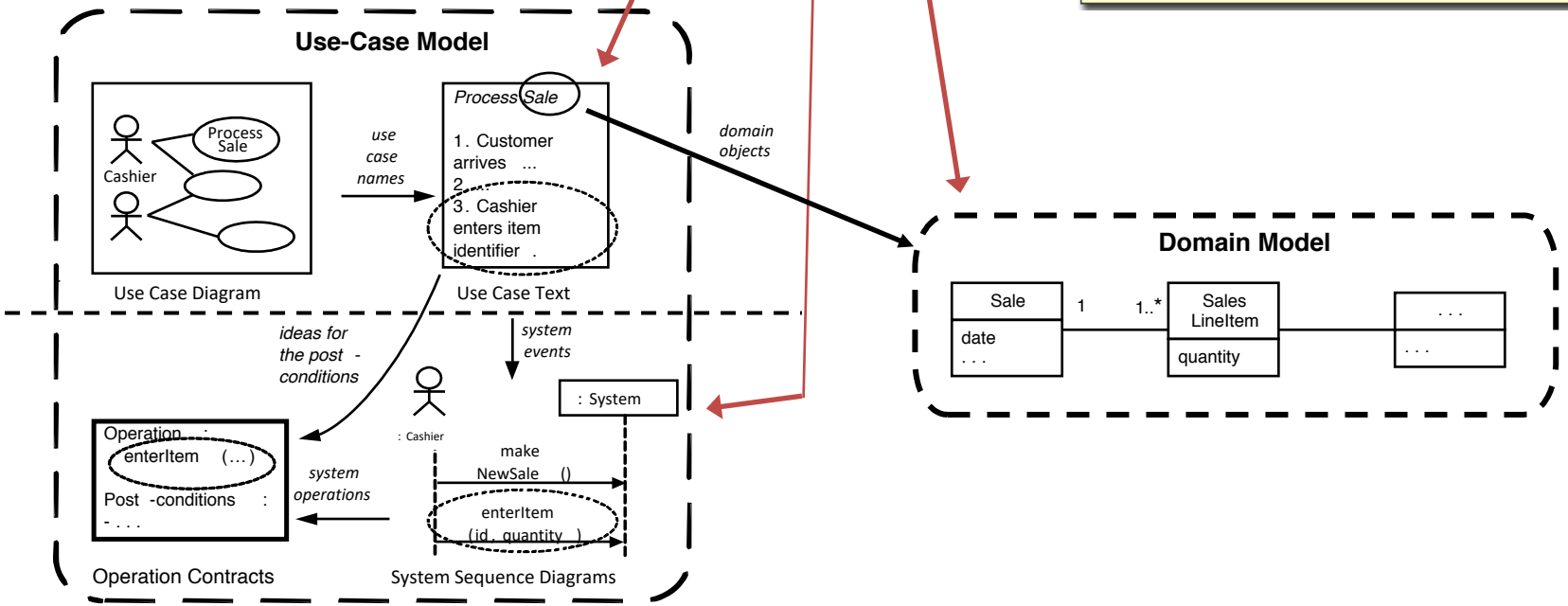
- Lecture materials from Prof. Dr. Oliver Hummel (University of Mannheim) have been used as a basis

- Further lecture materials that have been used
  - Craig Larman

Roadmap

- What's next?
  - Analyze the requirements
    - identify system events
    - specify system operations





System Sequence Diagrams (1/2)

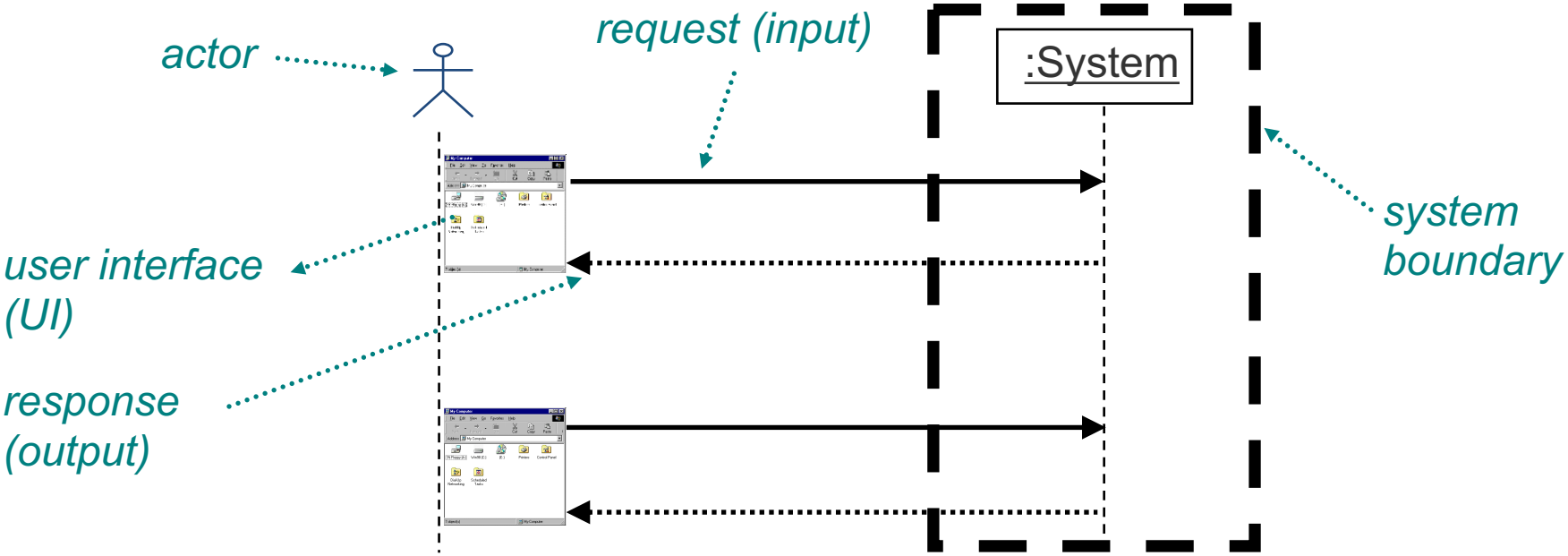
- Use cases describe how actors interact with the system
  - users generate **events** that request **operations** to be performed (system operations)
- As a starting point for system design it is important to identify the individual system operations
- **System Sequence Diagrams (SSD)** can be used for this
  - Based on the notation of UML sequence diagrams
  - For a particular scenario of a use case, an SSD shows
    - the events that external actors generate
    - the order of these events
    - necessary parameters and their types as well as return values
    - inter-system events

System Sequence Diagrams (2/2)

- The emphasis is on interactions that cross the sys. boundary
  - from actors to the system
- SSDs are typically created for the main success scenario and frequent or complex alternative scenarios
  - in practice typically not all SSDs are drawn
- SSDs are created in an iterative manner
  - as soon as a use case is going to be analysed
  - in the UP most SSDs are thus created during the elaboration phase

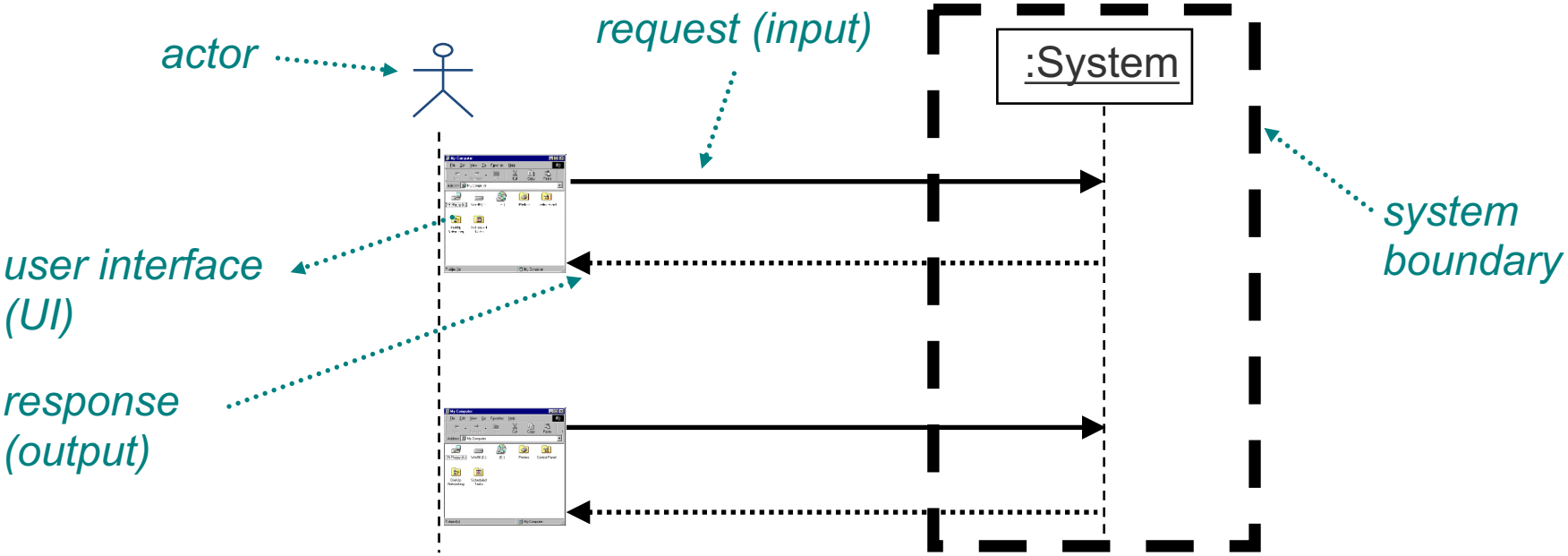
System Boundary

- To identify system events one must be clear on the system boundary
- System events are external events that directly stimulate the system



System Boundary

- To identify system events one must be clear on the system boundary
- System events are external events that directly stimulate the system
  - i.e., since the customer interacts with the cashier (and not directly with the system) the customer is not a generator of system events
  - typically human users interact with the system via a **User Interface (UI)**, which, however, is ignored in SSDs



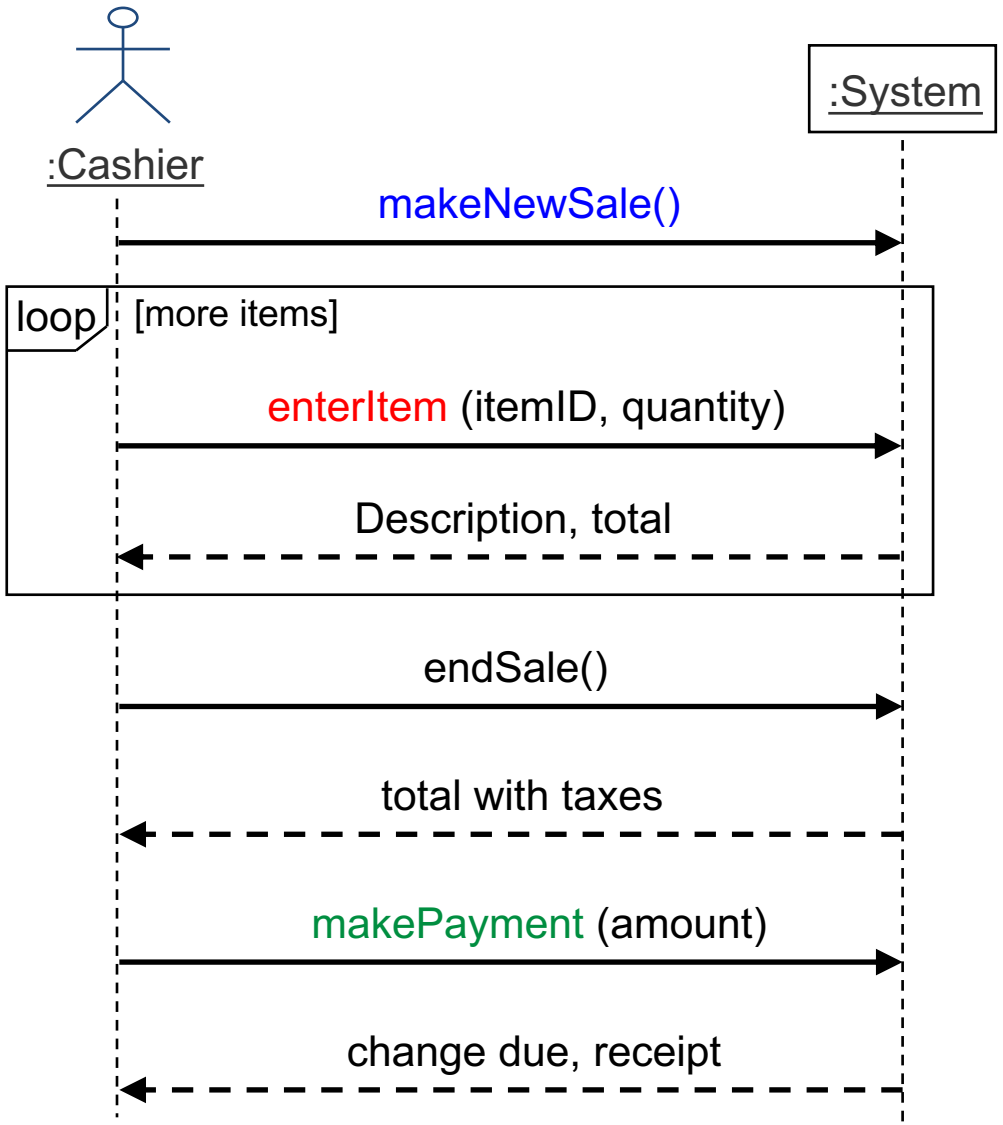
Deriving SSDs from Use Cases

Simple cash-only Process Sale operation

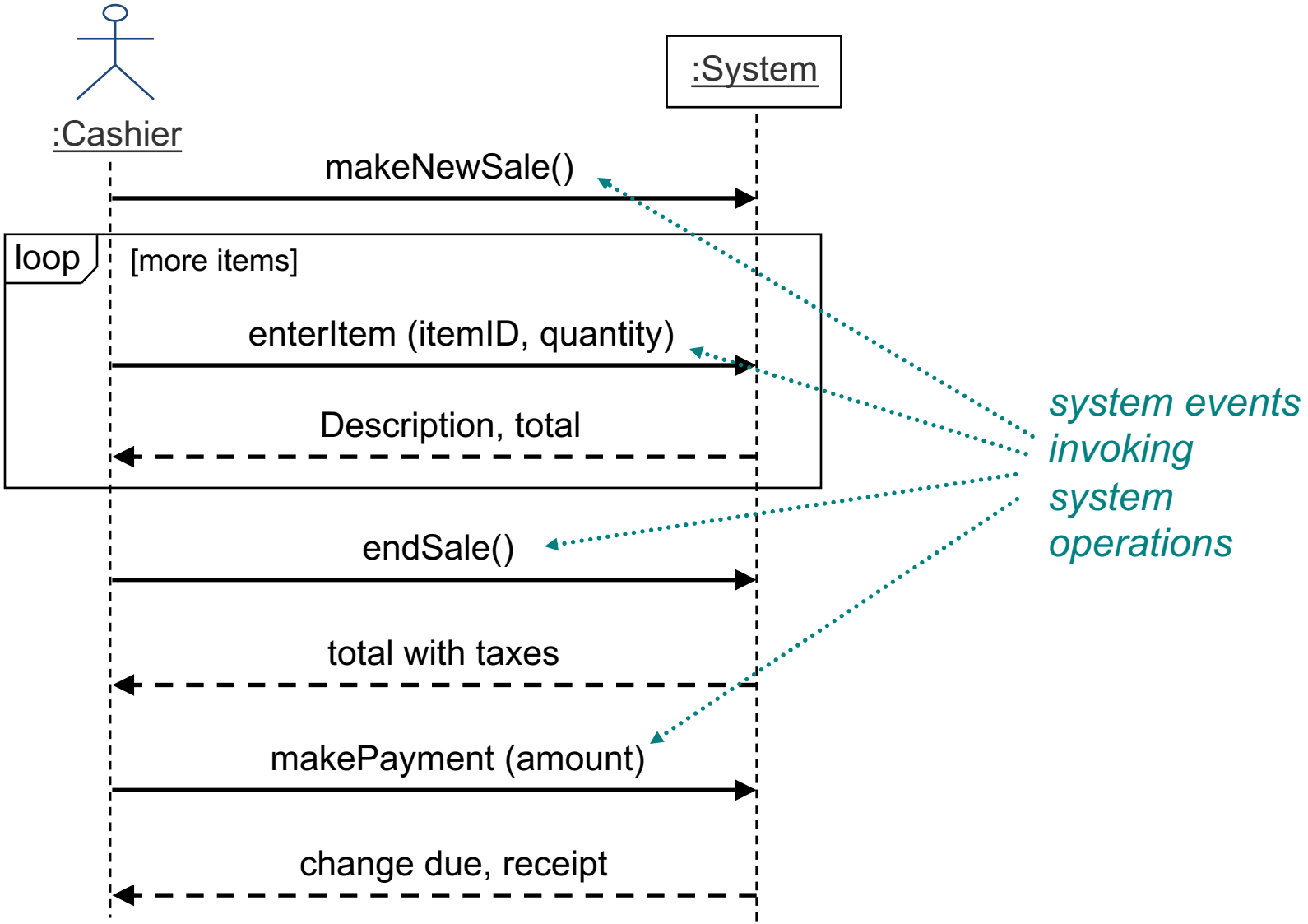
1. Customer arrives at POS checkout with goods and/or services to purchase
2. Cashier starts a new sale
3. Cashier enters an item
4. System records sale line item and presents item description, price and running total

   Cashier repeats steps 3-4 until indicates done

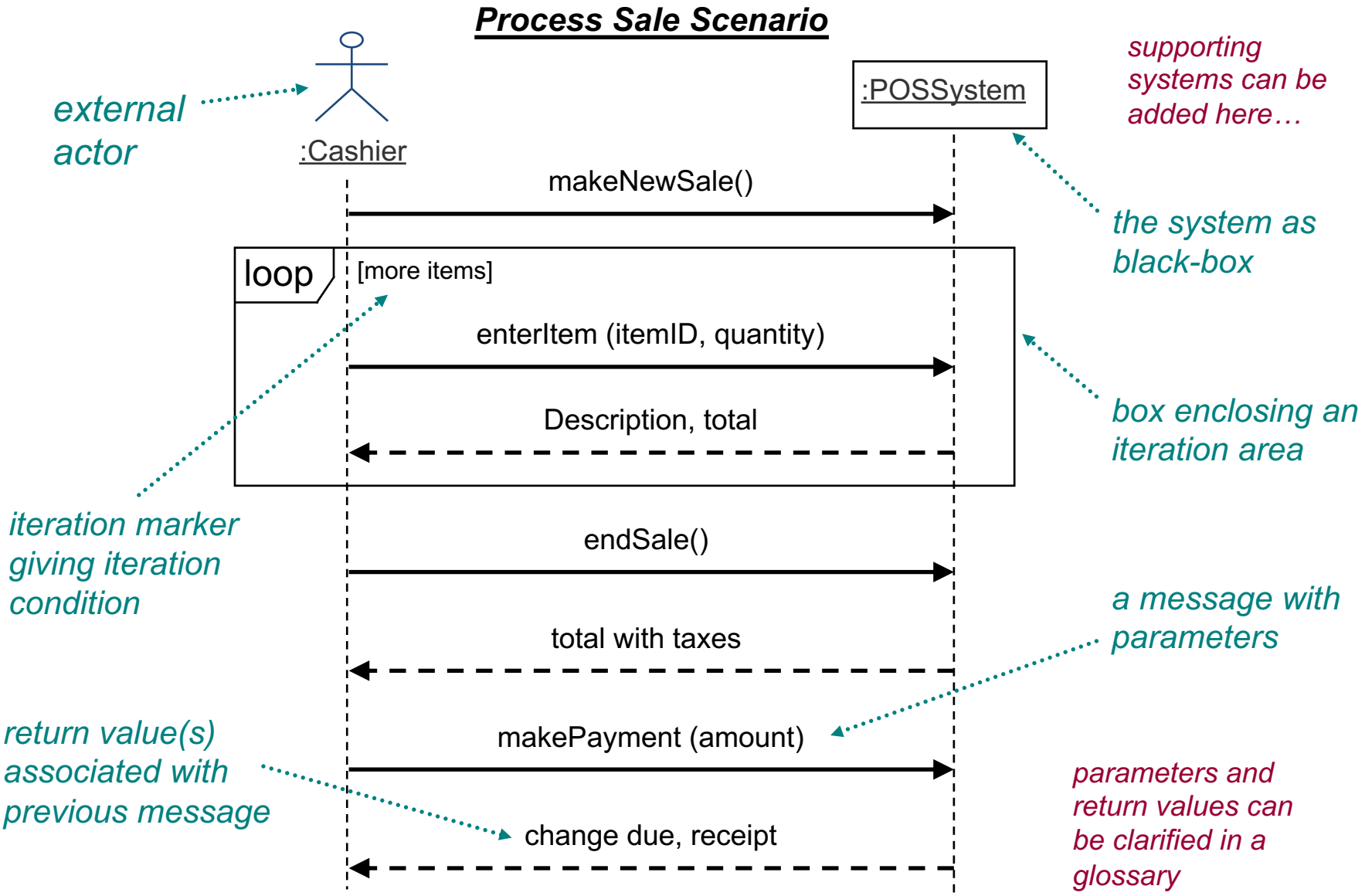
5. System presents total with taxes calculated
6. Cashier tells Customer the total, and asks for payment
7. Customer pays and System handles payment



System Operations



Elements in an SSD



Naming System Operations

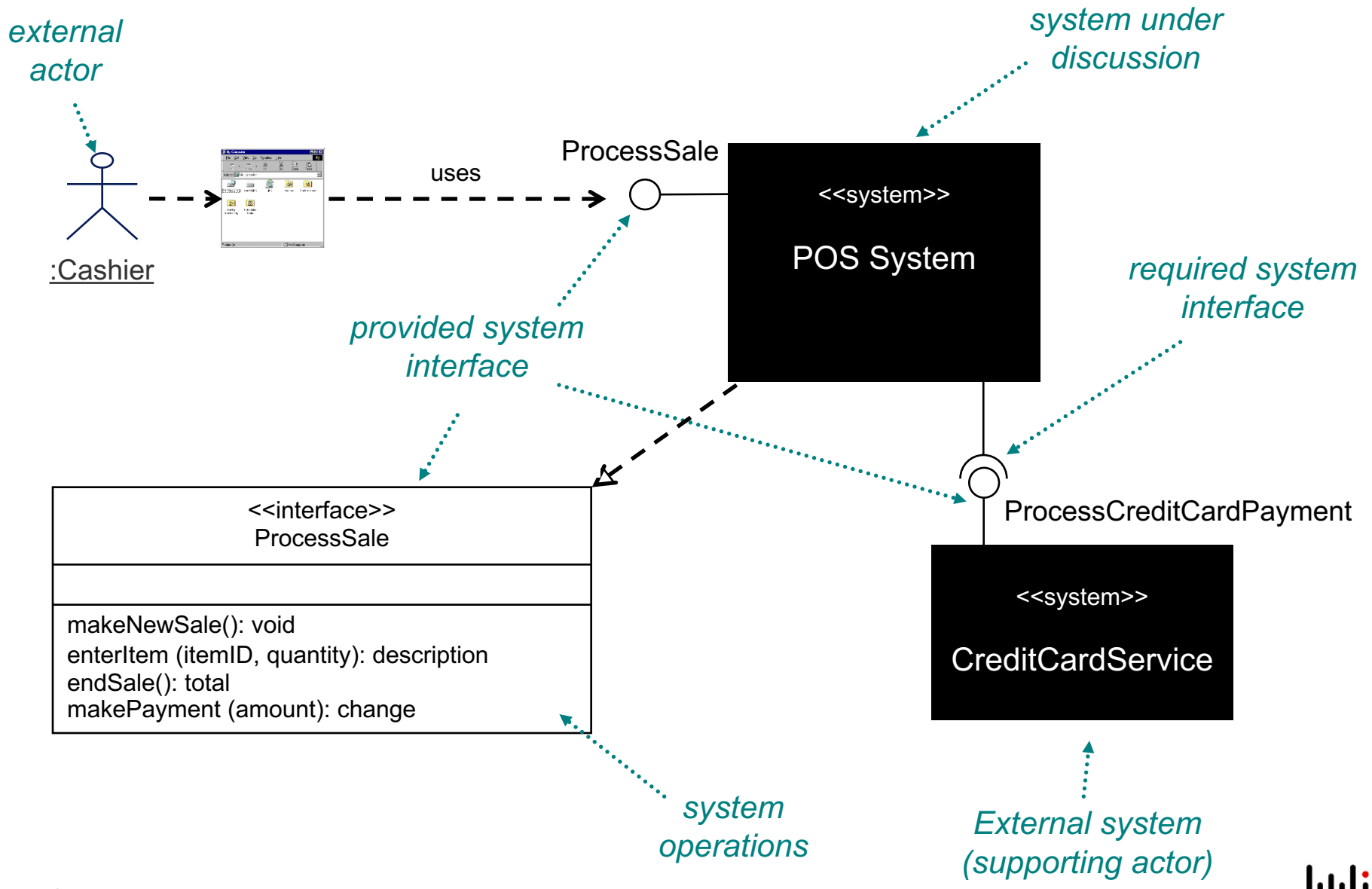
- System operations are operations the system offers to answer resp. satisfy incoming events (resp. requests)
- System operations should be expressed at the level of intent
  - rather than at the level of physical devices or interface widgets
- It adds clarity to start the name with a verb
  - add…, enter…, end…, make…, get…, is…, set…, retrieve…, delete…

  → Thus, *enterItem* is better than just *scan*
    - captures intent, but is non-committal on realization
- The entire set of system operations, across all use cases, defines the **public system interface**

Example UC 6.1 Watchlist erstellen

1. *Der Bankkunde stößt die Erstellung einer neuen Watchlist an.*
2. **Das DM-System fordert den Kunden zur Eingabe eines Namens für die Watchlist auf.**
3. *Der Bankkunde gibt einen Namen für die gewünschte Watchlist ein.*
4. **Das DM-System legt die gewünschte Watchlist an.**
5. **Das DM-System bietet dem Bankkunden die Möglichkeit, nach zu beobachtenden Aktien für die neu angelegte Watchlist zu suchen.**
6. *Der Bankkunde gibt Suchparameter für eine gewünschte Aktie an und löst eine entsprechende Suche danach aus.*
7. **Das DM-System sucht nach passenden Aktien und bietet die Ergebnisse dem Bankkunden zur Übernahme in die Watchlist an.**
8. *Der Bankkunde wählt die gewünschte Aktie aus und legt jeweils eine obere und/oder untere Kursgrenze fest, bei deren Über- bzw. Unterschreiten er vom System informiert werden möchte.*
9. **Das DM-System fügt die gewünschten Aktien zur Watchlist hinzu und kehrt in den Ausgangszustand zurück.**

System Interface Example



Operation Contracts (1/2)

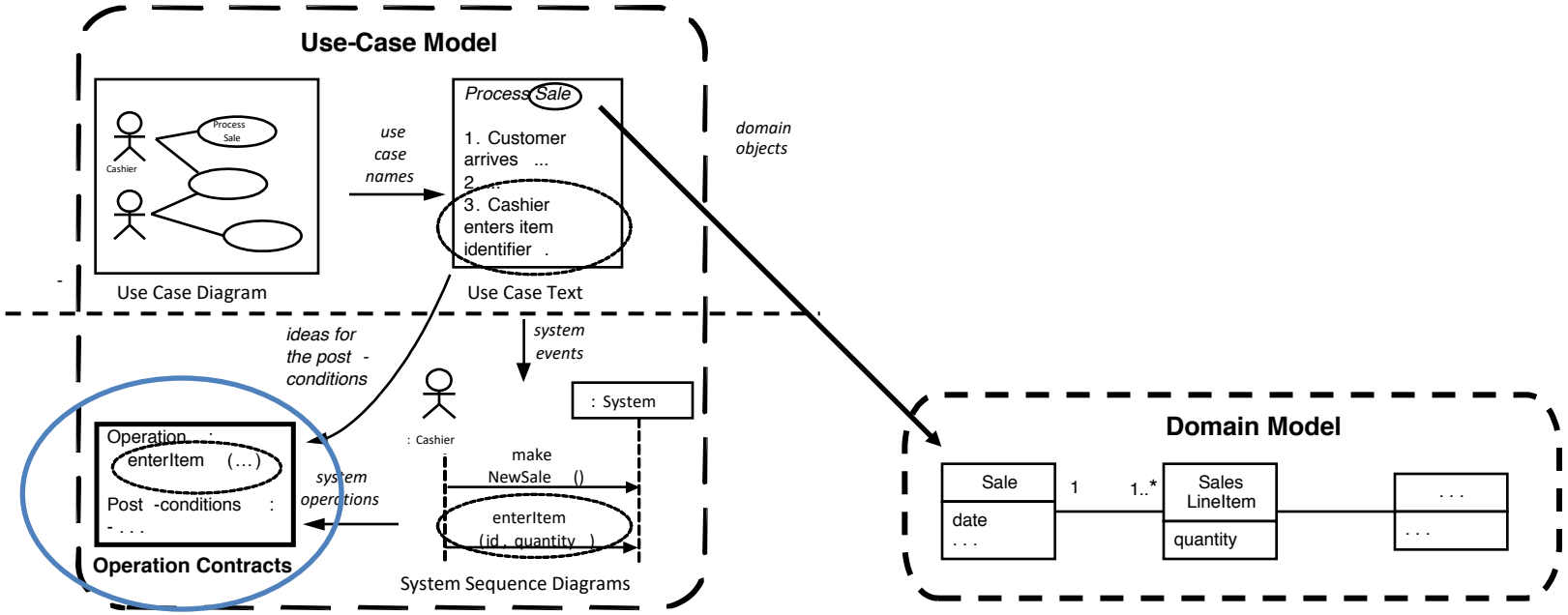
- **Use cases** are the primary way of describing system behavior
  - however, a more detailed description of system behavior has value as a starting point for system design
    - based on the “Design by Contract” approach popularized by Bertrand Meyer

- **Contracts**
  - are expressed in a declarative state change fashion
  - focus on ***what*** must happen rather than ***how*** it has to be accomplished

Party	Obligations	Benefits
Client	Provide letter or package of no more than 5 kgs, each dimension no more than 2 meters. Pay 100 francs.	Get package delivered to recipient in four hours or less.
Supplier	Deliver package to recipient in four hours or less.	No need to deal with deliveries too big, too heavy, or unpaid.

Operation Contracts (2/2)

- Operation contracts specify **detailed system behavior**
  - in terms of state changes to objects in the domain model
  - before vs. after a system operation has executed
    - i.e., they describe the outcome of the processing in the system



Operation Contract Schema

Operation:	name of operation and parameters
Cross References:	(optional) use case(s) this operation can occur within
Preconditions:	Noteworthy assumptions about the state of the system or objects in the Domain Model before execution of the operation. These are non-trivial assumptions the reader should know were made and will not be tested within the logic of the this operation. They are assumed to be true.
Postconditions:	The states of objects in the Domain Model after completion of the operation.

Contracts can be helpful in other contexts as well…

for use cases, business processes, methods in objects, etc.

Contract Example C02

Contract C02: enterItem

Operation:	**enterItem** (itemID : ItemID, quantity : integer): description, total
Cross References:	Use Cases: Process Sale
Preconditions:	There is a sale underway
Postconditions:	- A SaleLineItem 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 (association formed)

Various Uses in Practice

1. Business Processes
   - normally called Service Level Agreement (SLA) here
   - *example: operation of a web server*
2. Use Cases
   - *example: process DVD rental*
3. System Operations
   - *example: withdraw money with an ATM*
4. (Java) Methods
   - *example: assertions for binary search method*
5. Roommate Agreements ☺
   - *"If one of the roommates ever invents time travel, the first stop has to aim exactly five seconds after this clause of the Roommate Agreement was signed"* *[TBBT]*
   - *http://www.youtube.com/watch?v=N0qDy0T5WXM*

Pre- and Postconditions

- **Preconditions**
  - are usually simple and mostly require the previous execution of another system operation
- **Postconditions**
  - describe **changes in the state** of domain model objects –
    - instance creation and deletion
    - links (i.e., association instances) formed and broken
    - attribute modifications
  - are **declarative** rather than imperative
    - i.e., they are not actions to be performed
      - but declarations about the domain model objects that must be true when the operation has finished

The Spirit of Postconditions

- Postconditions should be expressed in the **past tense**
  - to emphasize that they **declare** changes that must have occurred
  - " a SalesLineItem was created" instead of "Creates a SalesLineItem"

- A helpful metaphor is to think of a system operation in terms of the system and its objects being presented on a theatre stage

1. before the operation take a picture of the stage
2. close the curtains on the stage and apply the system operation
3. open the curtains and take a second picture
4. compare the before and after pictures and express the changes in the state of the stage as postconditions

How complete should postconditions be?

- Contracts may not be needed, but if desired, generating a detailed set of postconditions is usually not necessary
  - their creation should be treated as an **initial best guess** and regarded as something that will not be complete
  - their early creation (even if incomplete) is better than deferring this investigation until design work
    - there developers should be concerned with the design of a solution, not with sorting out what needs to happen
  - however, some of the fine details will be discovered during design

- When creating contracts it is common to discover the need to record new elements in the Domain Model

  → enhance the Domain Model as you gain understanding

Contracts versus Use Cases

- Use cases are the main repository for project requirements
  - they may provide most or all of the necessary detail necessary to know what to do in design
  - however, there are situations where the details and complexity of required state changes are awkward to capture in use cases
    - e.g., addNewReservation in an airline reservation system

- The postcondition format of contracts encourages a more precise analytical language that supports thoroughness

Contract Creation Guidelines

1. Identify system operations from the SSDs
2. For system operations that are complex and perhaps subtle in their results, or which are not clear in the use case, construct a contract
3. To describe the postconditions, use the following categories
   - instance creation and deletion
   - attribute modification
   - associations formed and broken

- State the postconditions in a declarative, passive tense form
  - e.g., was…
- the most common mistake in creating contracts is forgetting to include the forming of associations
  - e.g., it is not enough that a new SalesLineItem instance was created by the enterItem system operation
  - it also needs to have been associated with Sale

Contract Example C01

Contract C01: makeNewSale

Operation:	makeNewSale()
Cross References:	Use Cases: Process Sale
Preconditions:	none
Postconditions:	- A Sale instance s was created (instance creation)
	- s was associated with a register (association formed)
	- attributes of s were initialized

Contract Example C03

Contract C03: endSale

Operation:	endSale()
Cross References:	Use Cases: Process Sale
Preconditions:	There is a sale underway
Postconditions:	- sale.isComplete became true (attribute modification)

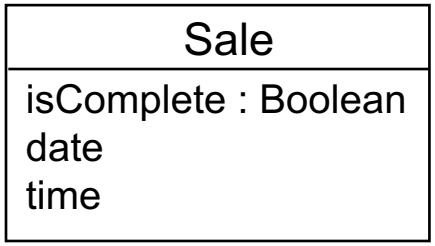
Contract Example C04

Contract C04: makePayment

Operation:	makePayment (amount: Money)
Cross References:	Use Cases: Process Sale
Preconditions:	There is a sale underway
Postconditions:	- A Payment instance p was created
	- p.amountTendered became amount
	- p was associated with the current Sale
	- the current Sale was associated with the Store (to add it to the historical log of completed sales)

Example Update to the Domain Model

- There is an item of information suggested in the previous contracts that is not yet represented in the domain model
- The completion of item entry into a sale
  - the endSale contract modifies it
  - and makePayment will probably need to test it
- One way to represent this information is with an isComplete attribute in Sale
  - hence, the Sale class in the domain model might be updated as follows



Operations, their Contracts and the UML

- The UML specifies operations
  - having a signature and pre- and post-conditions
- Natural language contracts are perfectly acceptable in UML
  - however, the so-called **Object Constraint Language (OCL)** offers a more formal way to express contracts
  - beyond the scope of this lecture

Concluding Remarks

- **Important!** Don't get dragged into a waterfall thinking and try to model the whole system until perfection
  - you will not be able to achieve this
  - try to model it just good enough
  - requirements analysis and modeling is rather a matter of hours or days per iteration in the UP

Outlook: Design