System Application Models

I2ISE

UCs are important!

Application models – bridging the gap

- A lot of time has been spent on writing use cases and making domain models. Today, we cash in!

- We will use the UCs to bridge the gap between *what* the system must do (requirements) and *how* it must be done (design)

- In other words, we will use the UC's as *design drivers*

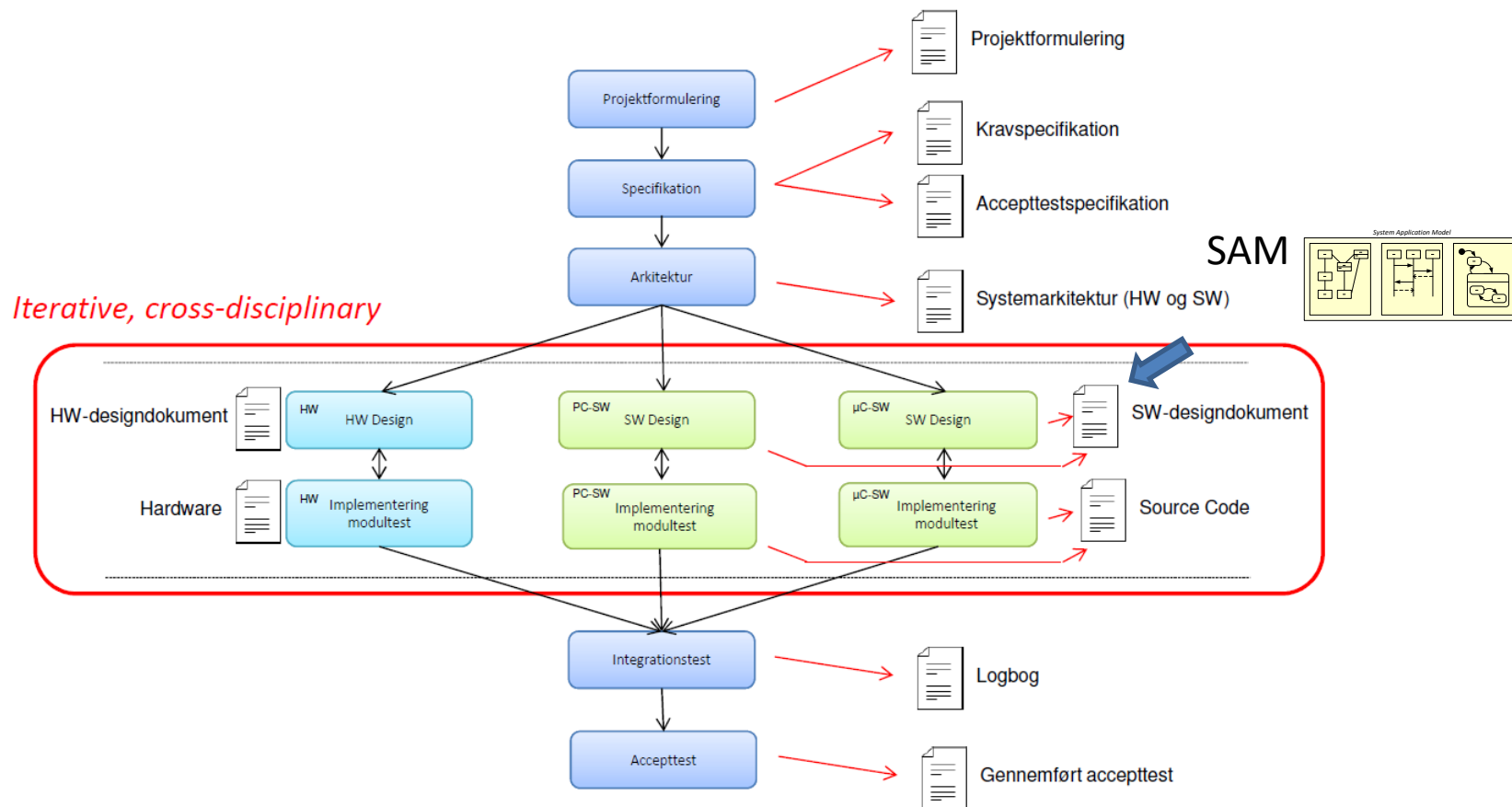- So it would seem that :

UCs are important!

What is a System Application Model?

- SAM is the first step of design!

- It will find relevant classes/modules to structure the design!

- It will describe how these interact!

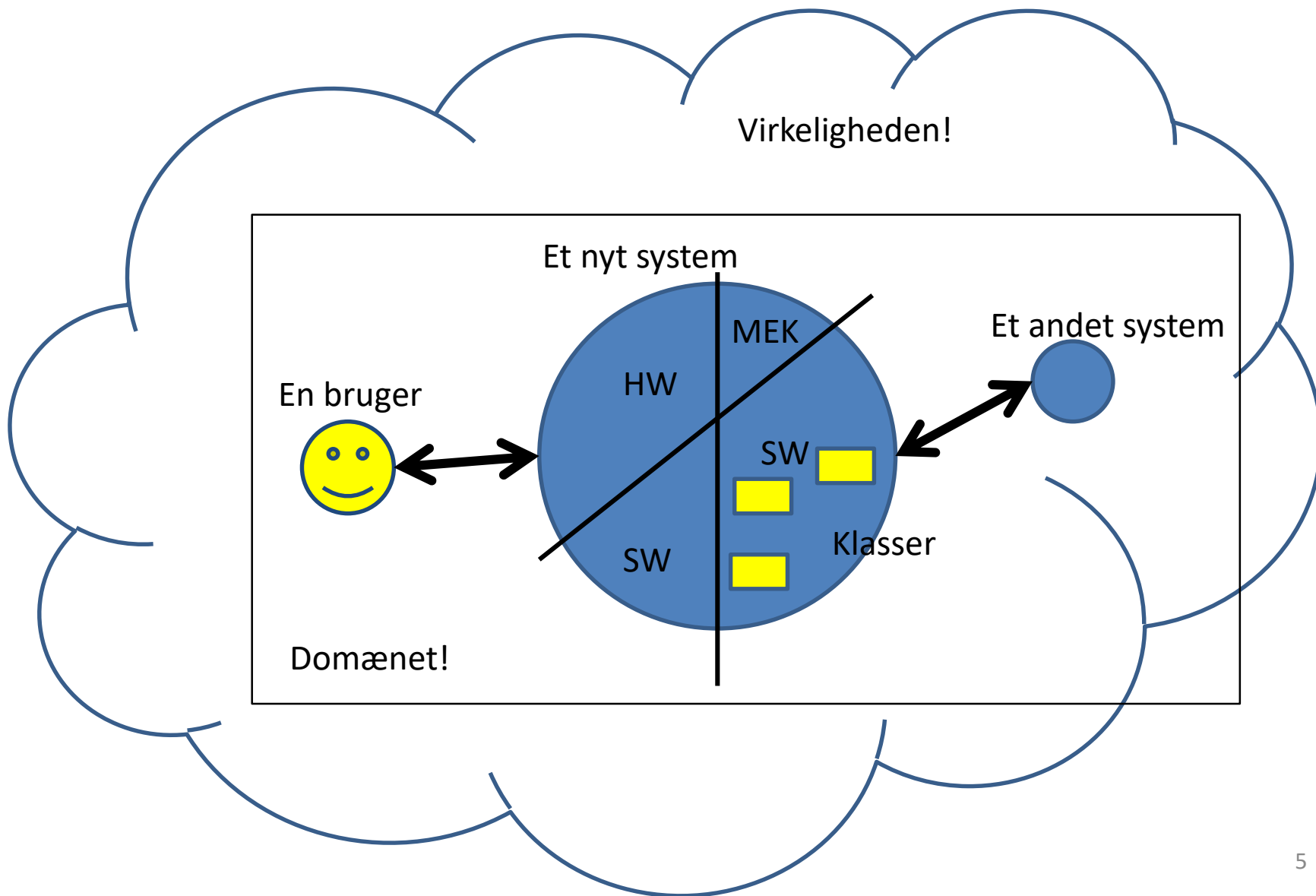- The *System Application model* is an artifact of design
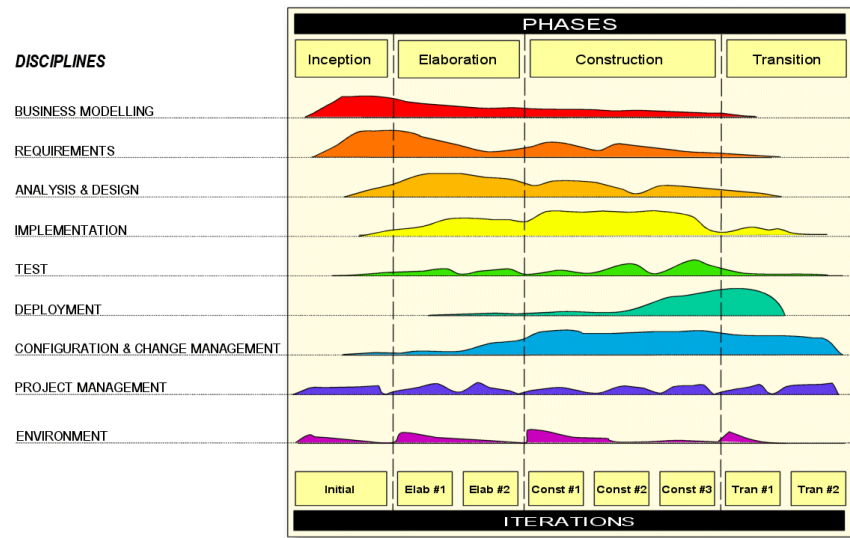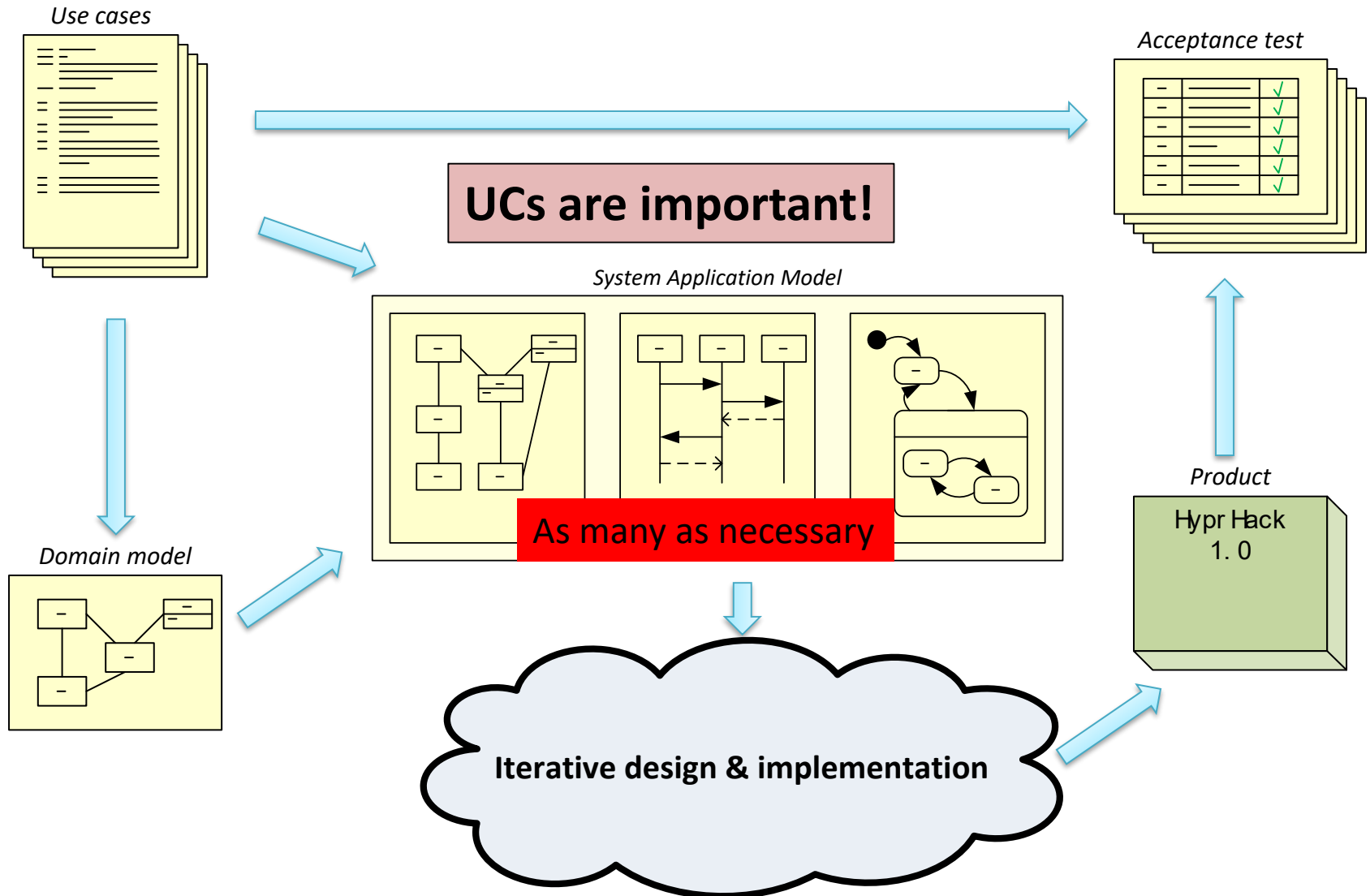
The SAM's place in the artefacts

Virkeligheden og systemet

The System Application Model

- The application model is a first, *incomplete* shot of a design – the "bridge"

- The application model is based on the system's *use cases* and the *domain model*.
  - So, again:

    UCs are important!

- The application model is built using three different types of diagrams
  - *Class diagrams* for structure
  - *Sequence diagrams* and *state machine diagrams* for behaviour

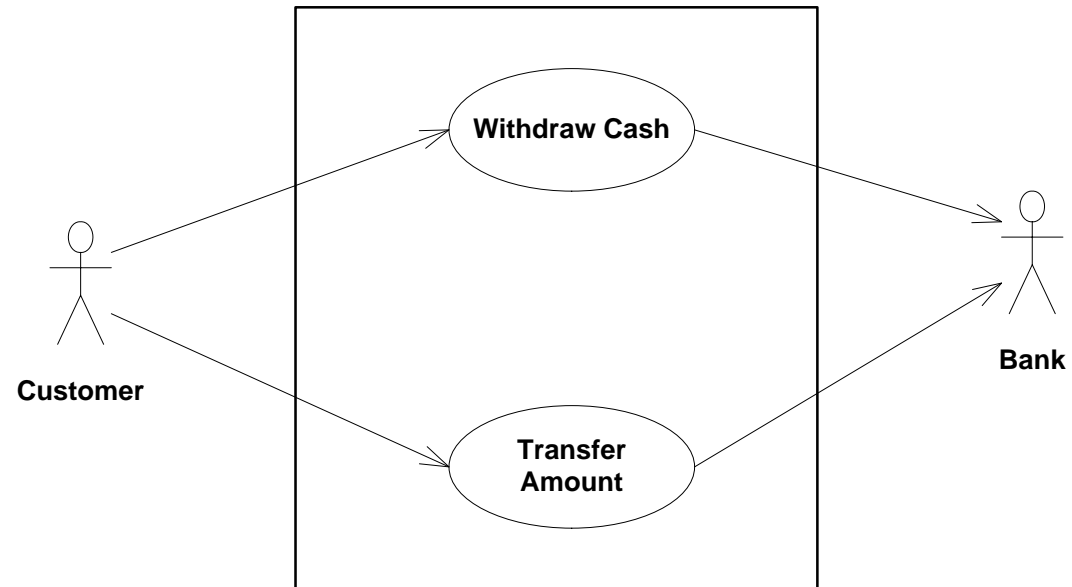# System Application Model
in the big picture

Use cases

Acceptance test

UCs are important!

Domain model

System Application Model

As many as necessary

Iterative design & implementation

Product

Hypr Hack
1. 0

Today's example: The ATM

ATM use cases

Withdraw Cash

Transfer Amount

Customer

Bank

ATM domain model
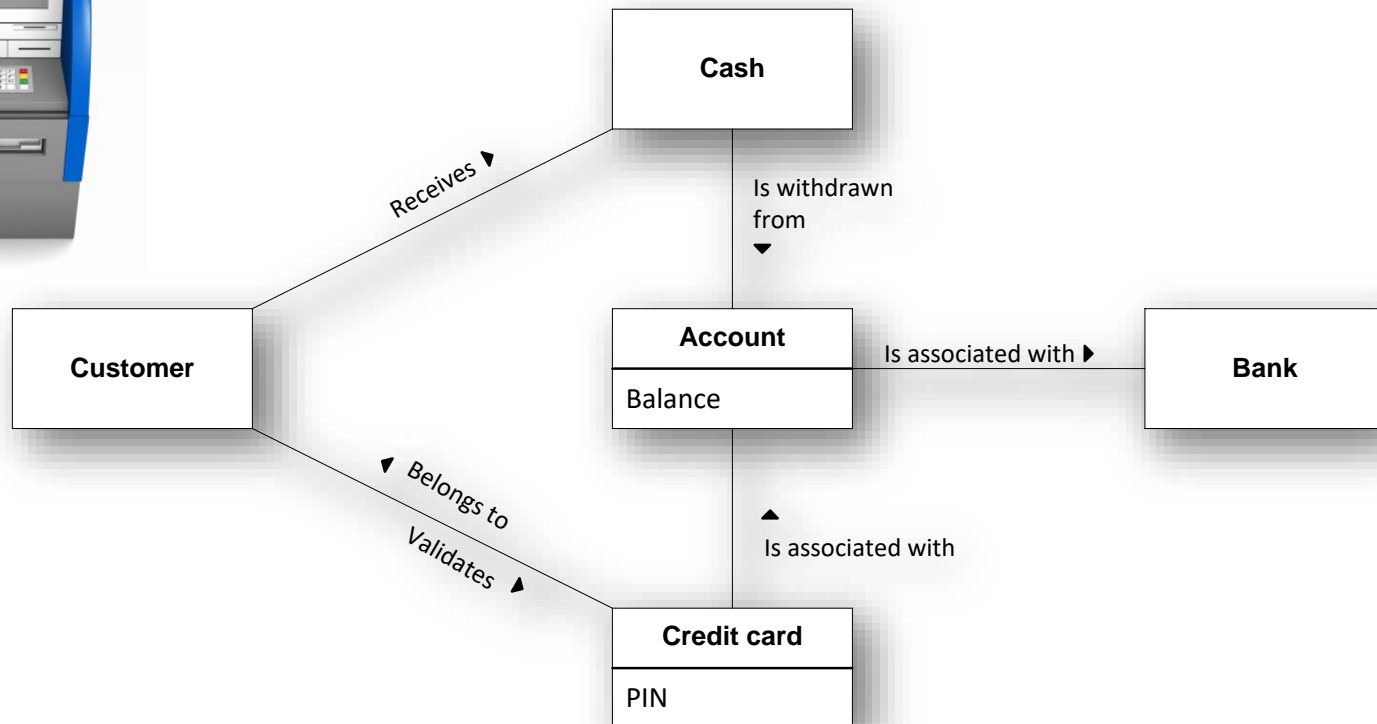
The System Application Model – Step 1

- The application model is constructed incrementally in units of use cases. So, apparently, **UCs are important!**

Step 1.1:   Select the next fully-dressed UC's to design for (how?)

Step 1.2:   Identify all actors involved in the UC → *Boundary* classes

Step 1.3:   Identify relevant classes in the domain model involved in the UC → *Domain* classes

So are DM classes!

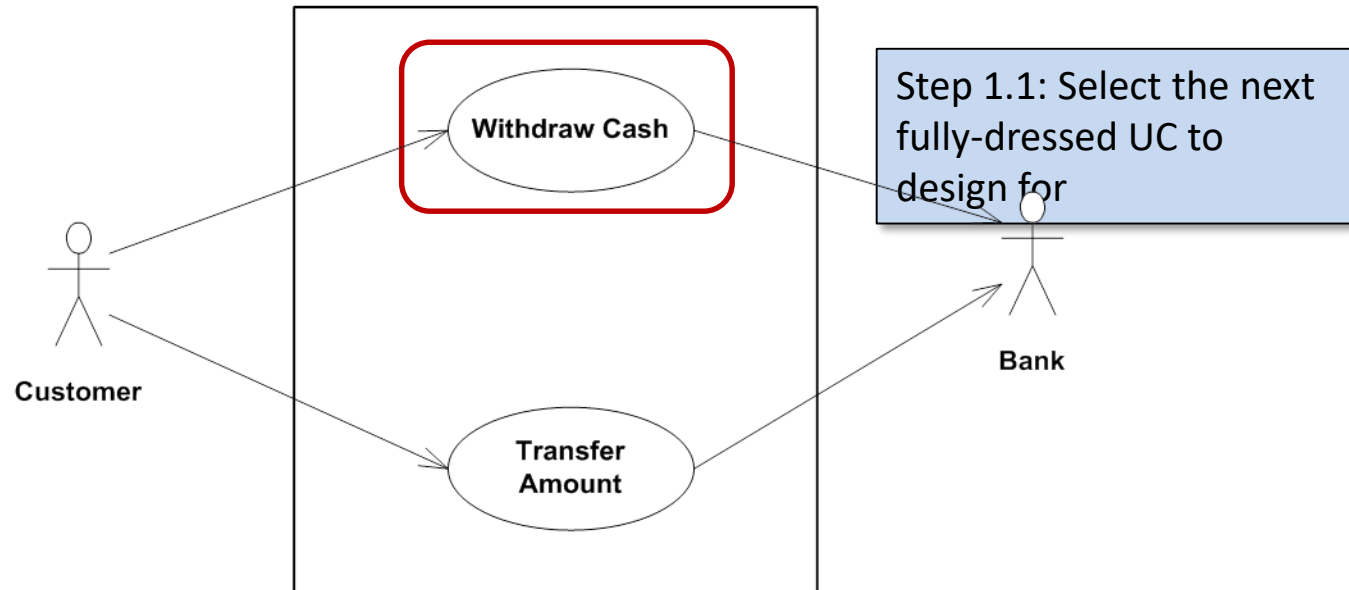Step 1.4:   Identify the UC *controller* → *Controller* class

Identify the what, the what and the what?!?

- Our application model consists of three different types of classes: *Boundary*, *domain*, and *controller* classes

- *Boundary* classes represent UC *actors*
  - They are the actors' interface to the system (UI, protocol, …)
  - They *present* the system but contain no business logic.
  - 1 per actor, shared between UCs
  - Optionally stereotyped «boundary»

- *Domain* classes represent the system's *domain*
  - Memory, domain-specific knowledge, configuration, etc.
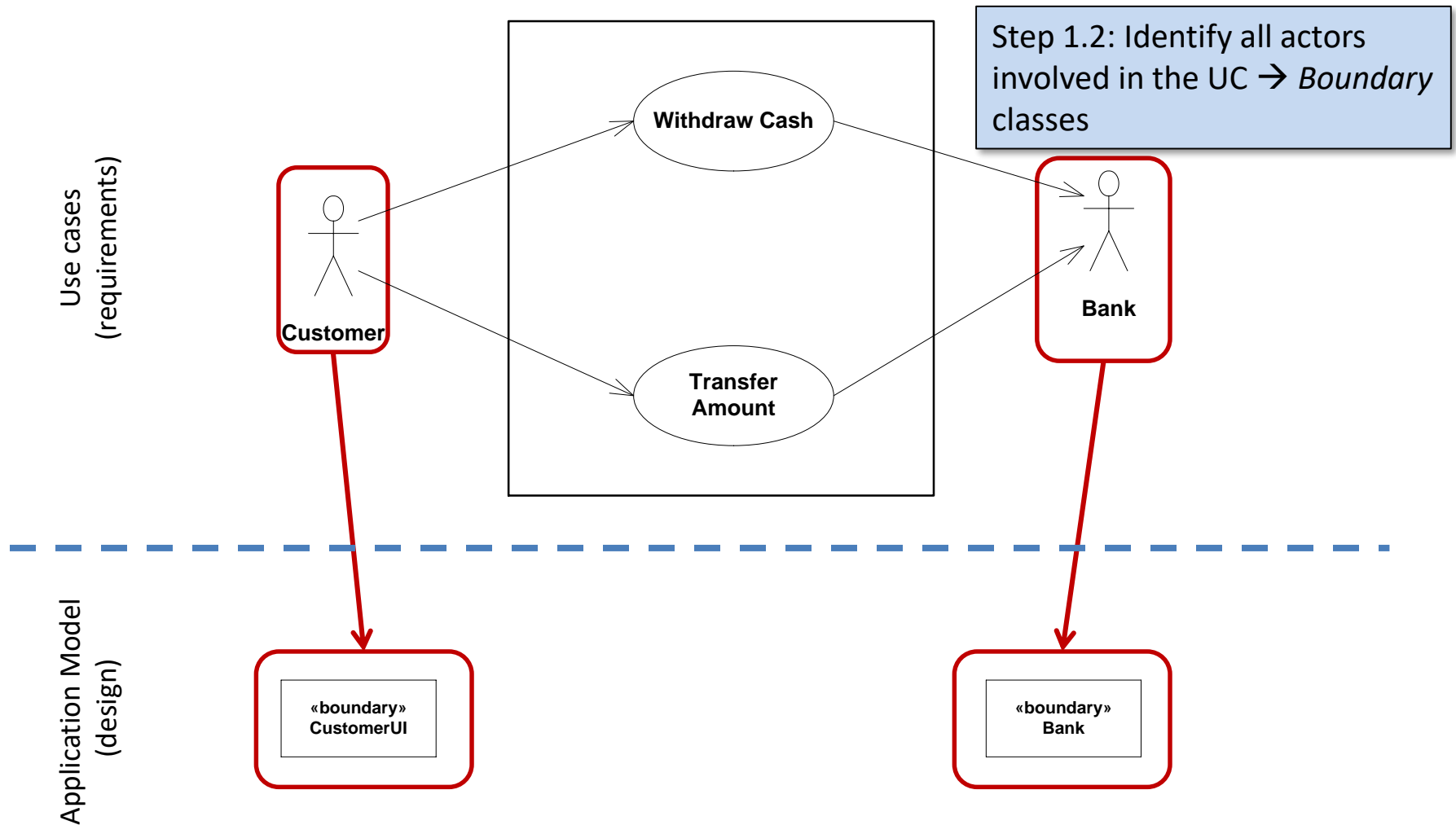 - 1 or more, shared between several UCs

Identify the what, the what and the what?!?

- Our application model consists of three different types of classes: *Boundary*, *domain*, and *controller* classes

- The *Controller* class holds the UC business logic
  - It "executes" the use case by interacting with the boundary and domain classes.
  - Named after the UC
  - Typically 1 per UC or 1 shared among a couple of UCs
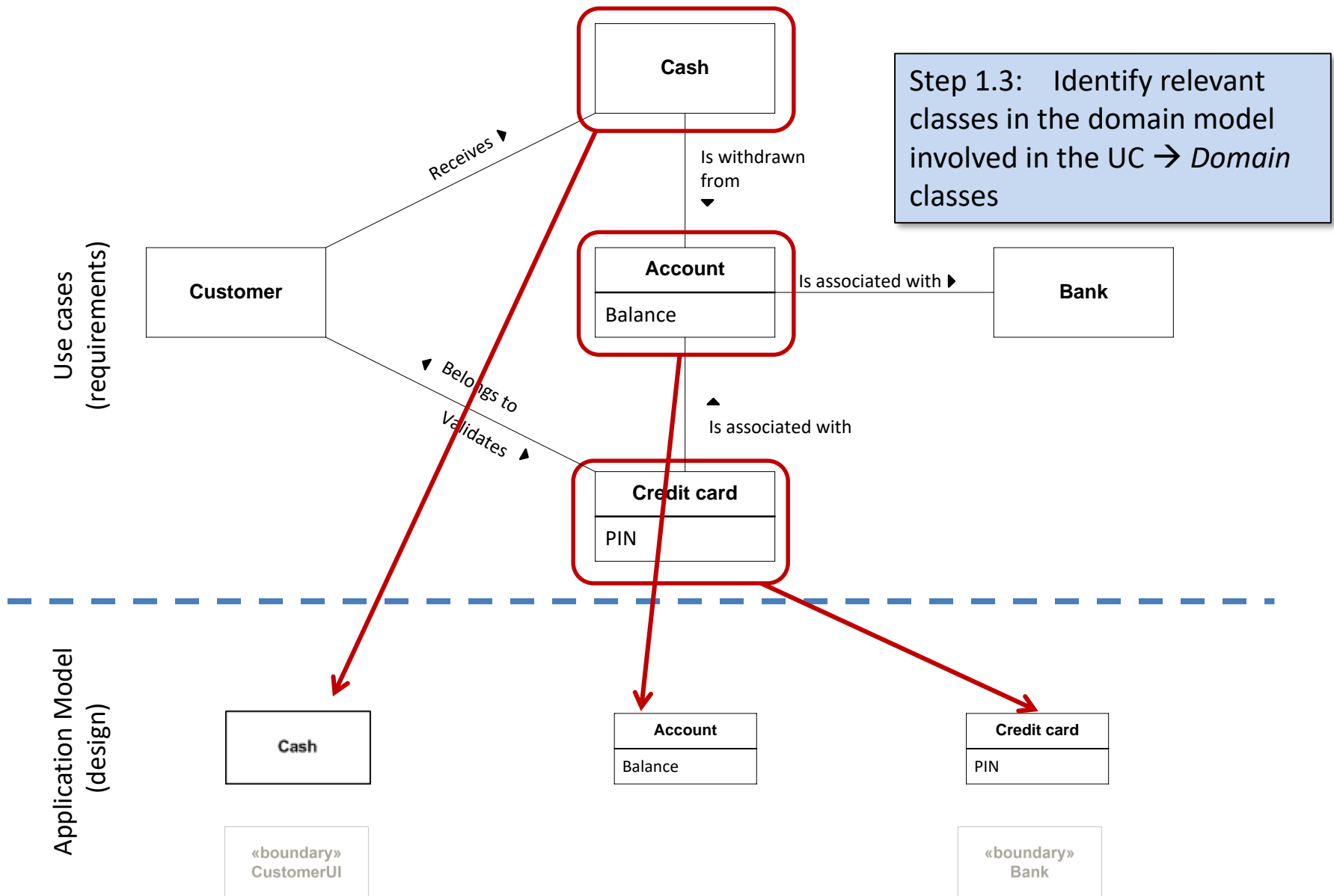 - Optionally stereotyped «control» or «controller»

ATM step 1.1: Select next Use Case

Step 1.1: Select the next fully-dressed UC to design for

ATM step 1.2: Actors -> boundary classes

Use cases (requirements)

Application Model (design)

Withdraw Cash

Transfer Amount

Customer

Bank

Step 1.2: Identify all actors involved in the UC → *Boundary* classes

«boundary»
CustomerUI

«boundary»
Bank

ATM step 1.3: Domain classes

Step 1.3: Identify relevant classes in the domain model involved in the UC → *Domain* classes

Use cases (requirements)

Cash

Receives ▶

Is withdrawn from ▼

Customer

Account

Balance

Is associated with ▶

Bank

◀ Belongs to

Validates ▶

▲ Is associated with

Credit card

PIN

Application Model (design)

Cash

Account

Balance

Credit card

PIN

«boundary» CustomerUI

«boundary» Bank

ATM step 1.4
Identify UC controller -> Controller class

Use cases (requirements)

Withdraw Cash

Transfer Amount

Customer

Bank

Application Model (design)

Cash

«controller» WithdrawCash

Credit card

PIN

«boundary» CustomerUI

Account

Balance

«boundary» Bank
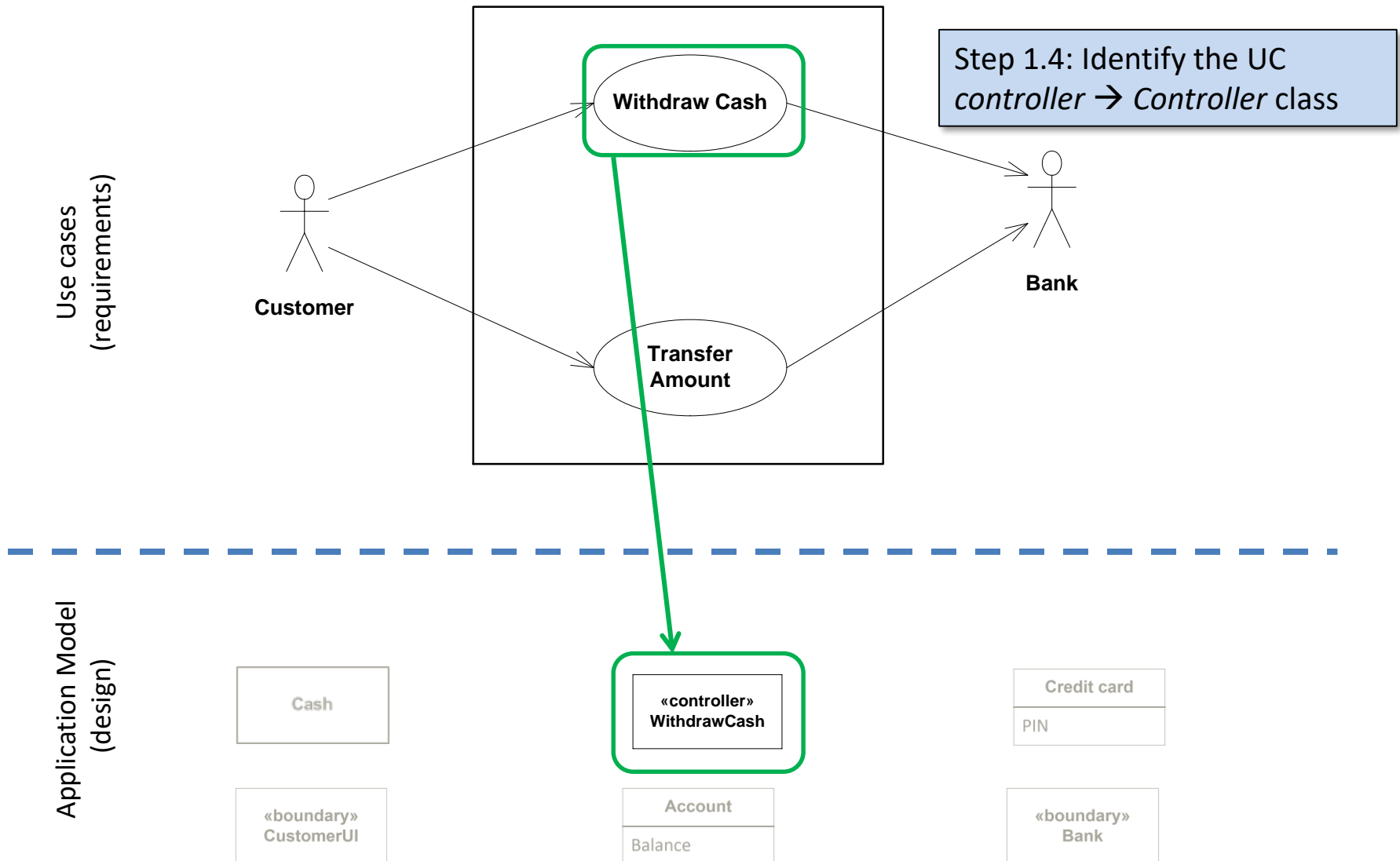
Step 1 complete – so far, so good

- We have now completed Step 1 and identified 6 candidate SW classes for our initial design
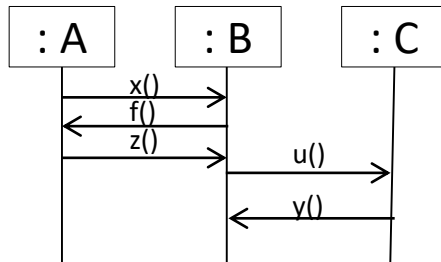- To do this, we used our *use cases* and our *domain model*

Cash

«controller» WithdrawCash

Credit card
PIN

«boundary» CustomerUI

Account
Balance

«boundary» Bank
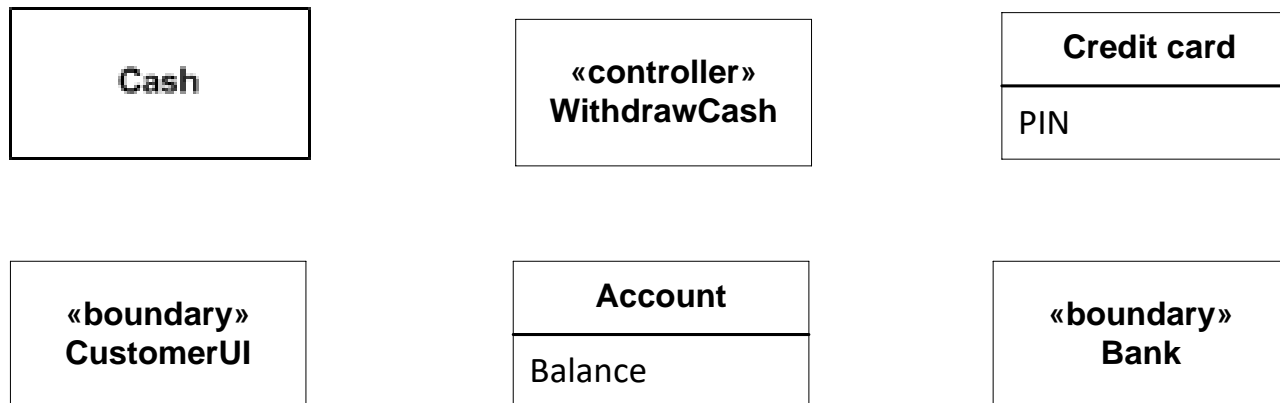
- We must now add *behaviour* to these classes – that's Step 2

Principle for step 2:
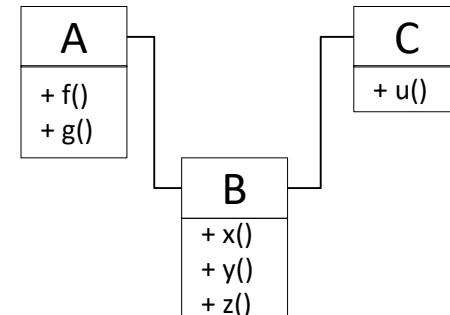# Go through main scenario, update collaborations
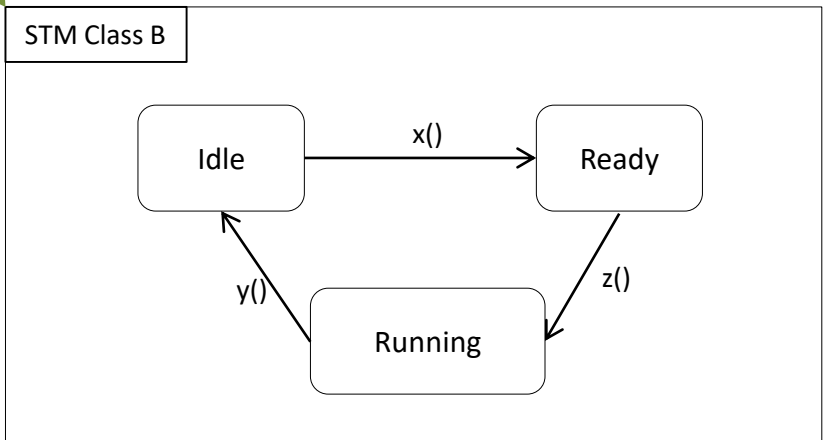
The System Application Model – Step 2

- The collaboration between the classes is now explored from the UC description – so, still,

UCs are important!

Step 2.1:	Go through the UC main scenario step-by-step and identify collaborations (**actor- or system-initiated**)
Step 2.2:	Update the application model's sequence and class diagrams to reflect the collaboration (relations, operations, attributes)
Step 2.3:	Identify any classes with state-based behavior and update STMs for the classes (states, events, transitions) .
	(Step 2.3 is skipped if none classes with state-based behavior)
Step 2.4:	Verify that the diagrams adhere to the UC (descriptions, test)
Step 2.5:	Repeat 2.1 – 2.4 for all UC exceptions. Refine model.

- Note: All 3 diagrams (class, SEQ, STM) are updated at the *same time* in this process.

# Steps 2.1-2.4 for
UC *Withdraw Money*

Main scenario:
1. Customer inserts credit card in System
2. System requests Customer's PIN code
3. Customer enters PIN code
4. System validates card info and PIN code with Bank
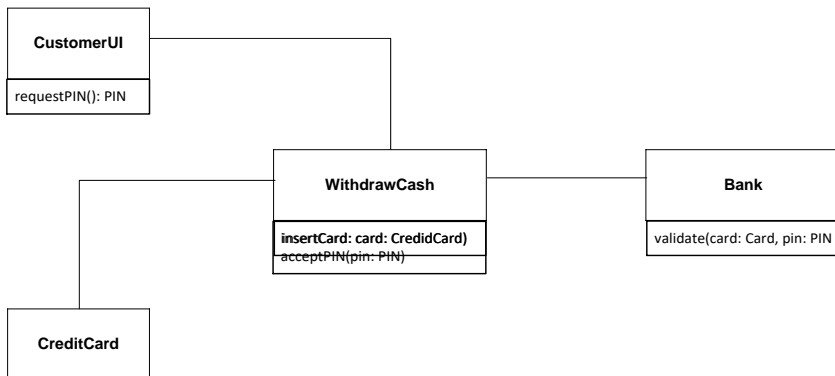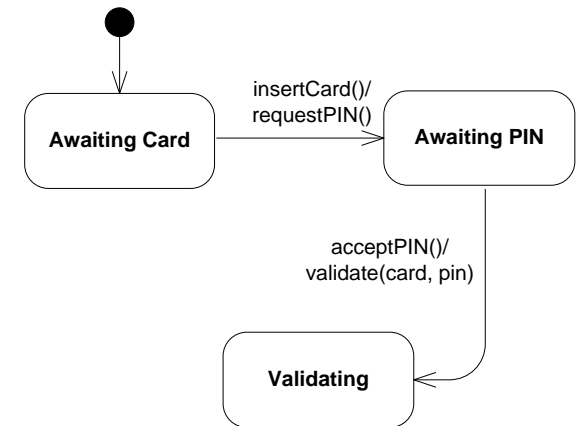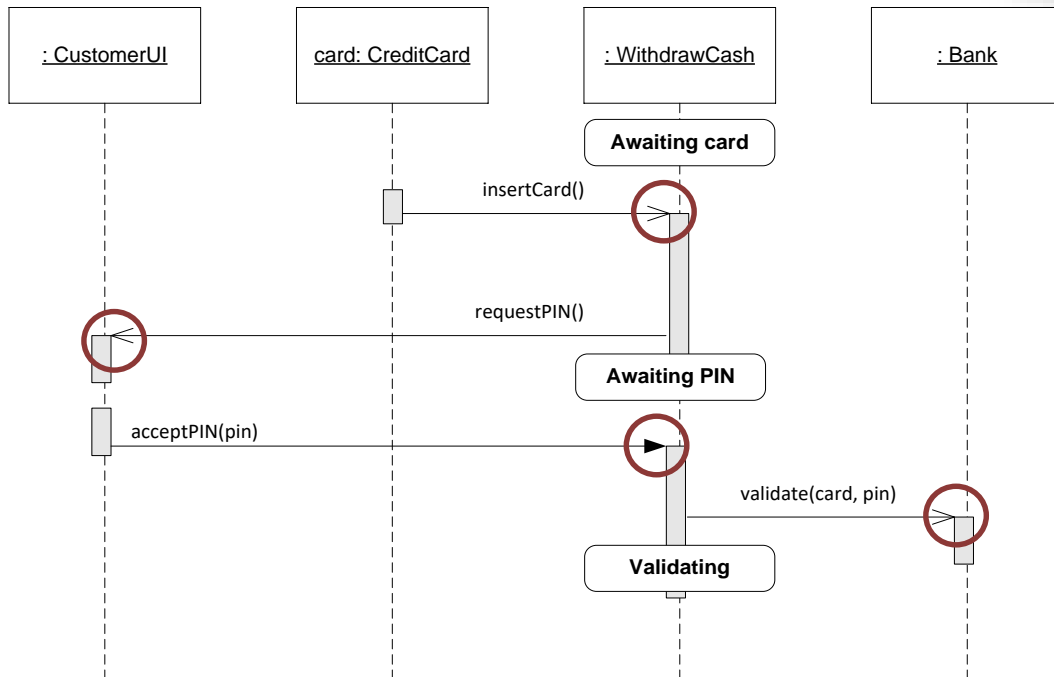
Find the STM from the Scenario w/Extensions

1. Customer inserts credit card in System

2. System requests Customer's PIN code

3. Customer enters PIN code

4. System validates card info and PIN code with Bank

5. Bank validates card
   [*Ext. 5.1: Invalid PIN entered*]

6. System requests desired action from customer

7. Customer selects "Withdraw Cash"

8. ...

Awaiting card

Awaiting PIN

Validating

cardInvalid()/

acceptPin()/...

Awaiting Action
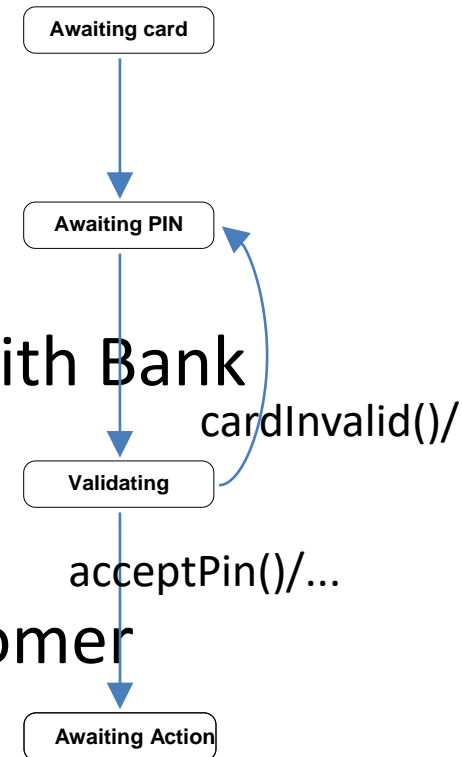
The System Application Model – Step 3 and beyond

- As you add more UCs to the application model you will begin to discover *reuse* of the previous classes
  - Domain and boundary classes often repeat
  - Different domain classes may be so closely related that they might as well be "collapsed" into one
  - Sometimes, even controllers "collapse"

- At this time, experience must ensure the correct cut between reuse and new classes

Then what??
The first version of the Application Model is used for the first code design

Your turn: Complete system application model for UC *Withdraw Cash*

- The complete text for UC Withdraw Cash is on the BlackBoard. You have the following tasks:
  - Complete the System Application Model for the main scenario for the UC
  - Complete the System Application Model for all extensions for the UC

- Starting points for 3 diagrams
- Continue work with this in next lecture (L19)

Start the exercise

- The start of the Sequence Diagram
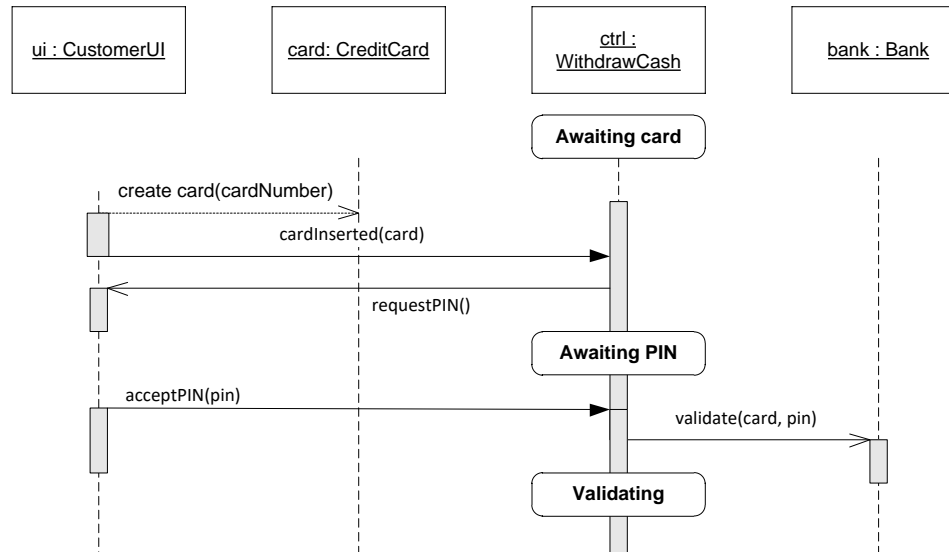
Start the exercise

- Start of class diagram and state machine diagram