Applikationsmodeller

I2ISE

UCs er vigtige!

Applikationsmodellen – slår bro over kløften

- Vi har brugt masser af tid til at skrive UCs og lave Domænemodel(ler)
- I dag får vi nytte af dem!
- Vi vil bruge dem til at slå bro over kløften mellem **Hvad** systemet skal gøre (krav/specifikationer) og **Hvordan** det skal gøres (design)
- Vi vil bruge UCs som *design drivers* – input til designprocessen
- Så:

UCs er vigtige!

Hvad er Applikationsmodellen?

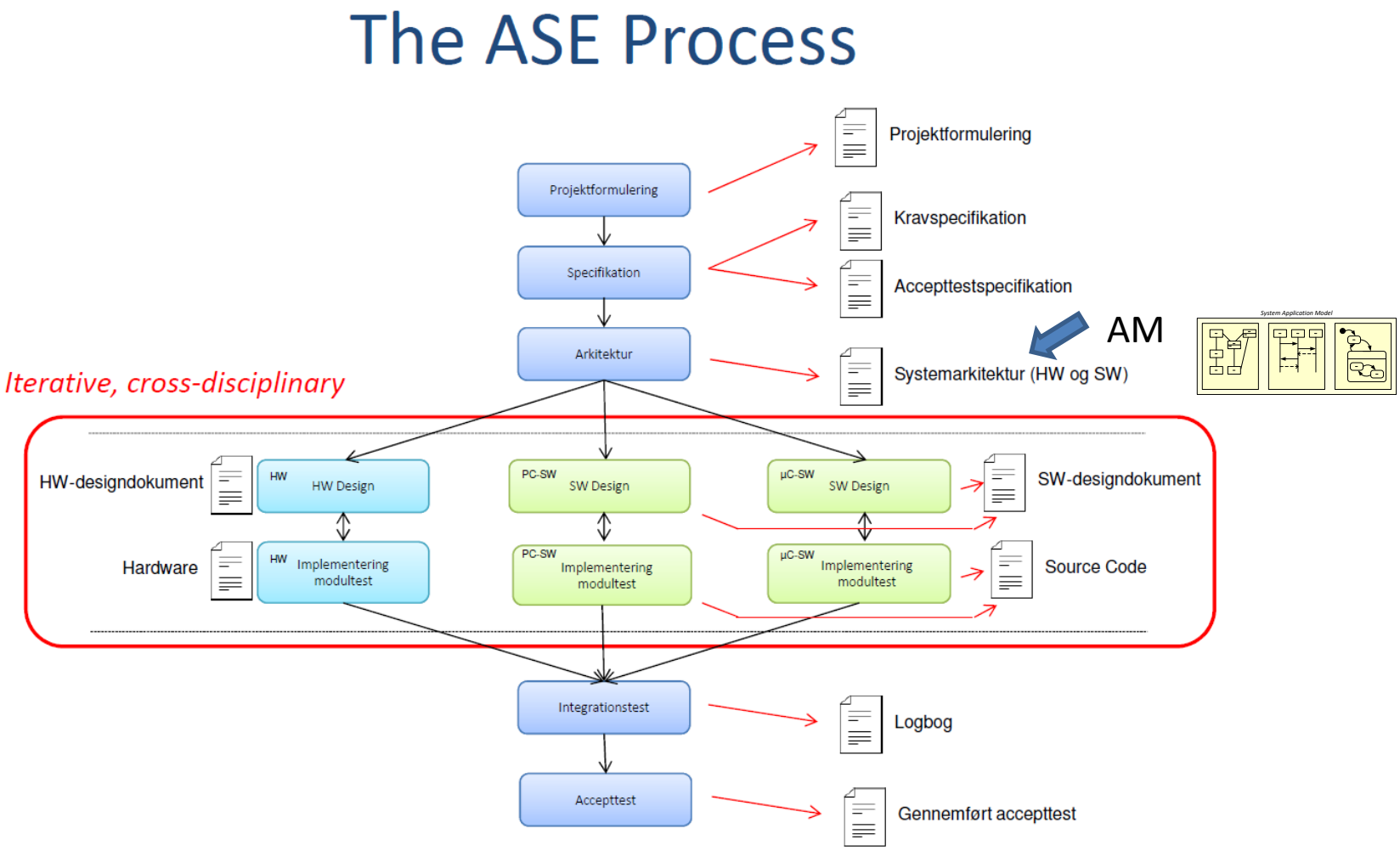
- *Applikationsmodellen* – AM – er første skridt i designprocessen!

- Den vil pege på relevante klasser/moduler som designet bygges op på!

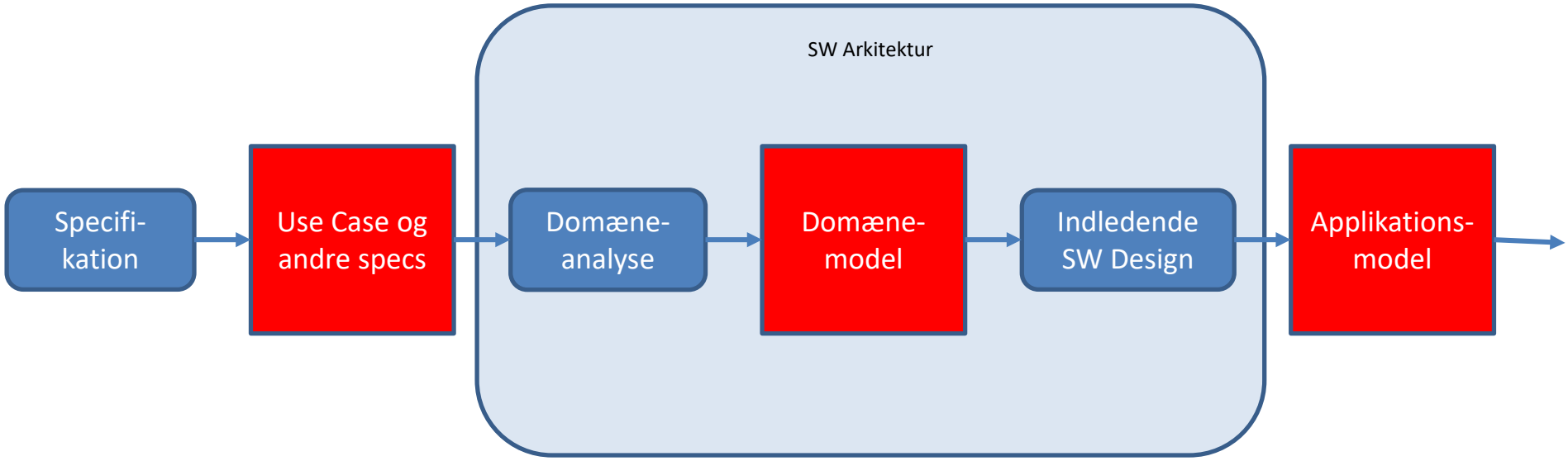
- Den vil beskrive hvordan disse interagerer

- Applikationsmodellen er en del af SW Design afsnittet i Systemarkitektur-dokumentet.

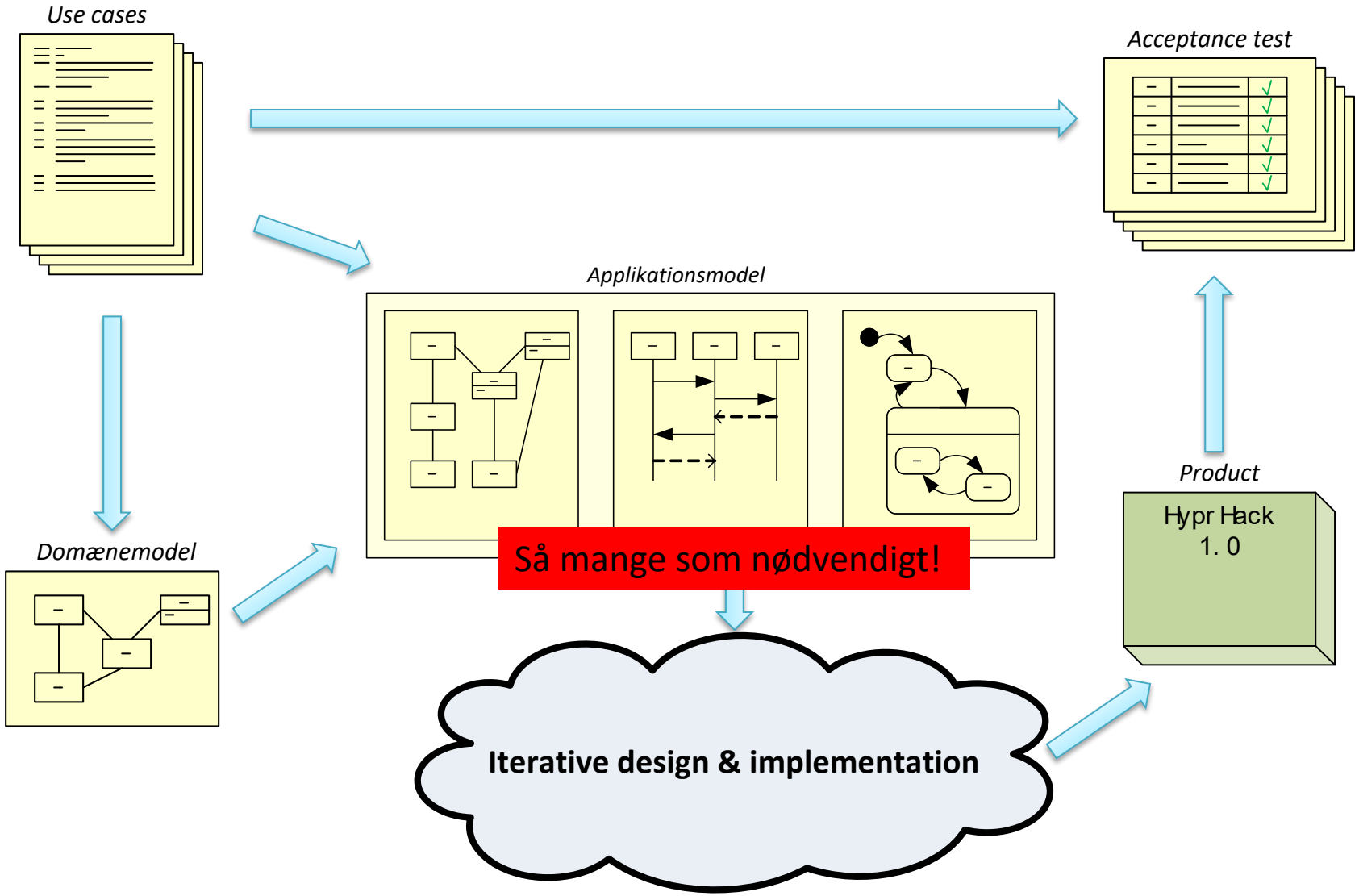
AM's plads i dokumenterne



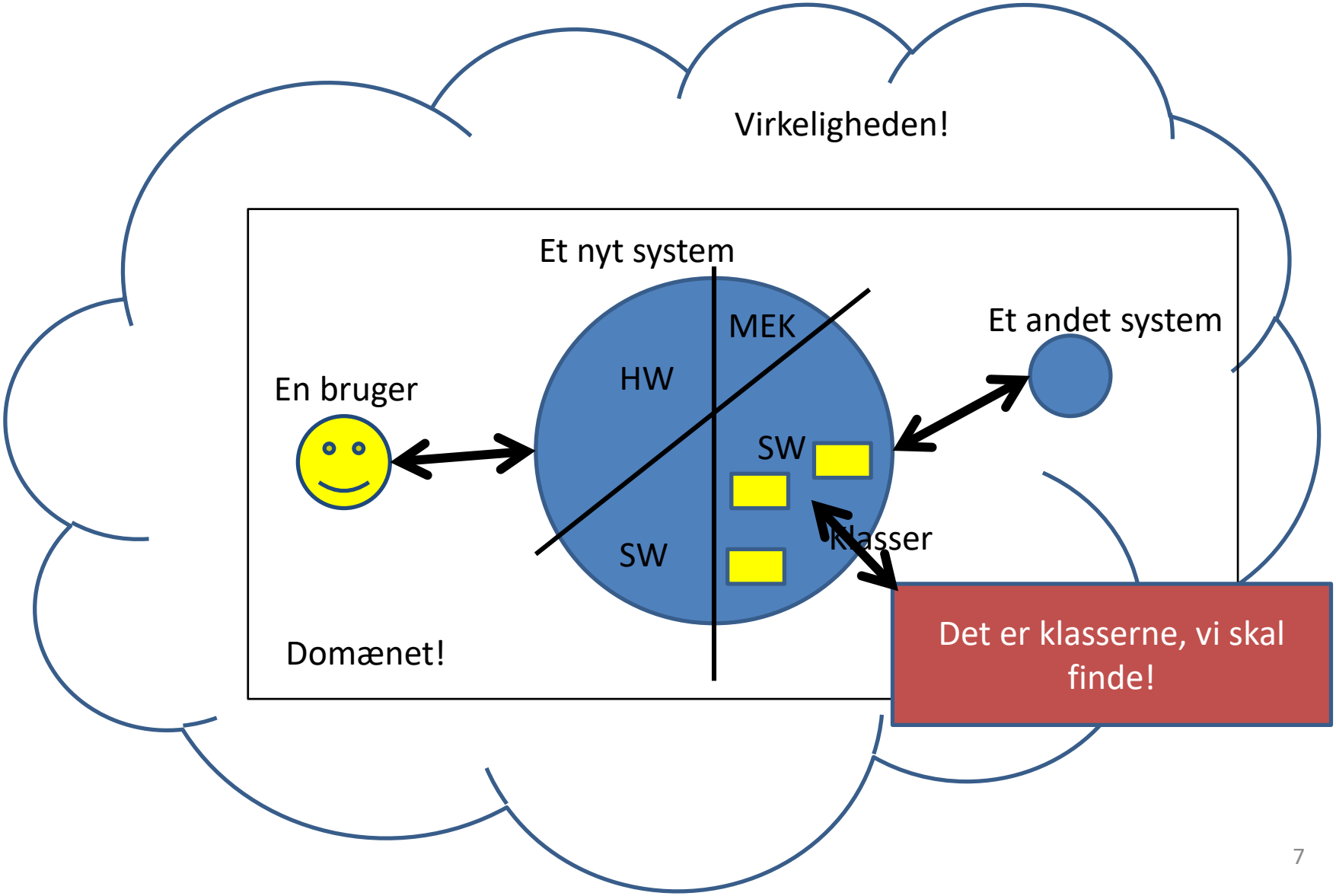
SW Arkitektur – fra UC til Design



Applikationsmodellens plads i det store billede



Virkeligheden og systemet



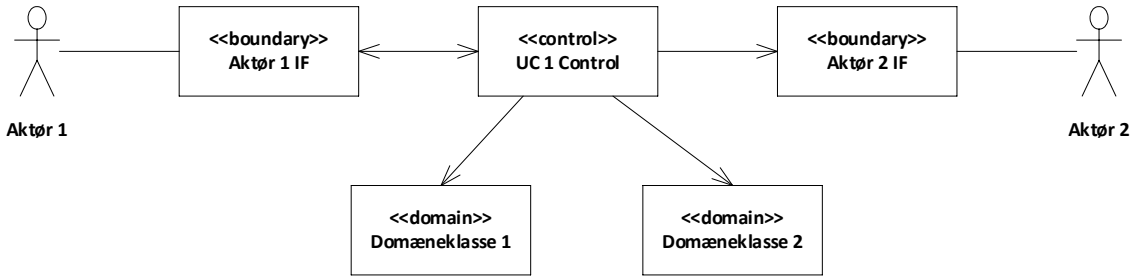
Arkitektur – hvad er det?

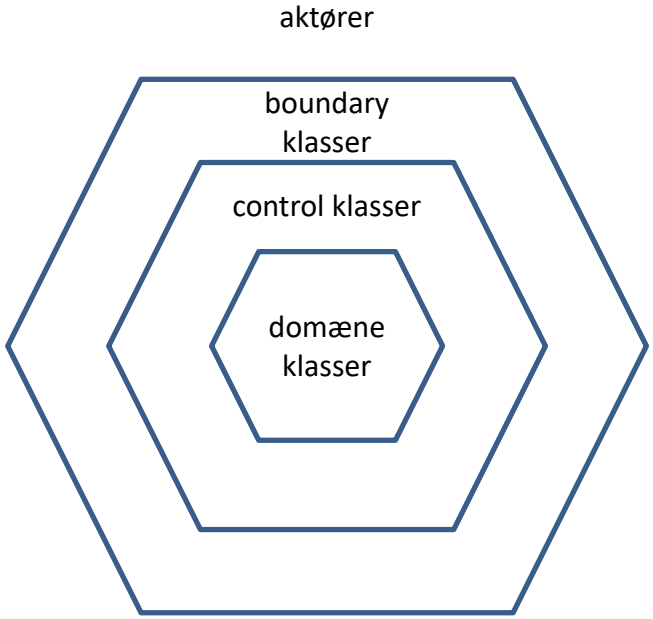
- Vi mangler en god ide til at organisere disse klasser!
- Det kaldes Arkitektur
- Vi har ganske vist Domæneklasserne
- Men hvordan får vi styret afviklingen af UC?
- Det bør Domæneklasserne ikke vide noget om!
- Og Domæneklasserne bør ikke vide noget om Hardware og andre interfaces til omverdenen

Vores arkitektur

- … hedder *Entity-Control-Boundary* pattern
- Dette er et velkendt og velafprøvet *Architectural Pattern*
- Et *pattern* er et mønster, som man kan genkende i mange godt designede og godt fungerende applikationer
- Vi vil her kalde *Entity* klasser for *Domain* klasser

Domain – Control - Boundary





Applikationsmodellen

- Applikationsmodellen er sammensat af følgende 3 typer af diagrammer:
  - *Klassediagram* (cd) for strukturen (statisk)
  - *Sekvensdiagrammer* (SEQ) og
  - *Tilstandsdiagrammer (STM)* for aktiviteter (dynamisk)

- Der laves et tilstrækkeligt antal **sæt** af disse til at beskrive **alle UCs**

- UC bruges til at konstruere dem

Applikationsmodellen – Step 1

- Applikationsmodellen opbygges skridt for skridt, hvor hvert skridt styres af én UC

Step 1.1:	Vælg den næste fully-dressed UC til at designe for (hvordan?)	
Step 1.2:	Identificer alle involverede **aktører** i UC → ***Boundary klasser***	Er der tvivl, er en klasse boundary klasse før den er domæneklasse
Step 1.3:	Identificer **relevante klasser i Domænemodellen** som er involveret i UC → ***Domain* klasser**	
Step 1.4:	Tilføj én UC *control* → ***Control* klasse**	

Nogle hvad for nogle klasser?

- Applikationsmodellen består af 3 forskellige klassetyper: *Boundary*, *domain*, og *control* klasser

- *Boundary* klasser repræsenterer UC *aktører*
  - De er aktørernes interface **til systemet** (UI, protokol, …)
  - De gør systemet **synligt for aktørerne**
  - Indeholder ingen "business logic" – dvs. ingen styring af UC
  - Mindst 1 per aktør, deles mellem de UCs der har samme aktører
  - Bør forsynes med stereotypen «boundary»

- *Domain* klasser repræsenterer systemets *domæne*
  - Data, domæne-specifik viden, konfigurationer, etc.
  - 0, 1 eller flere, deles mellem de UCs der bruger samme begreber
  - Kan forsynes med stereotypen <<domain>>

Nogle hvad for nogle klasser?

- Applikationsmodellen består af 3 forskellige klassetyper: *Boundary*, *domain*, og *control* klasser

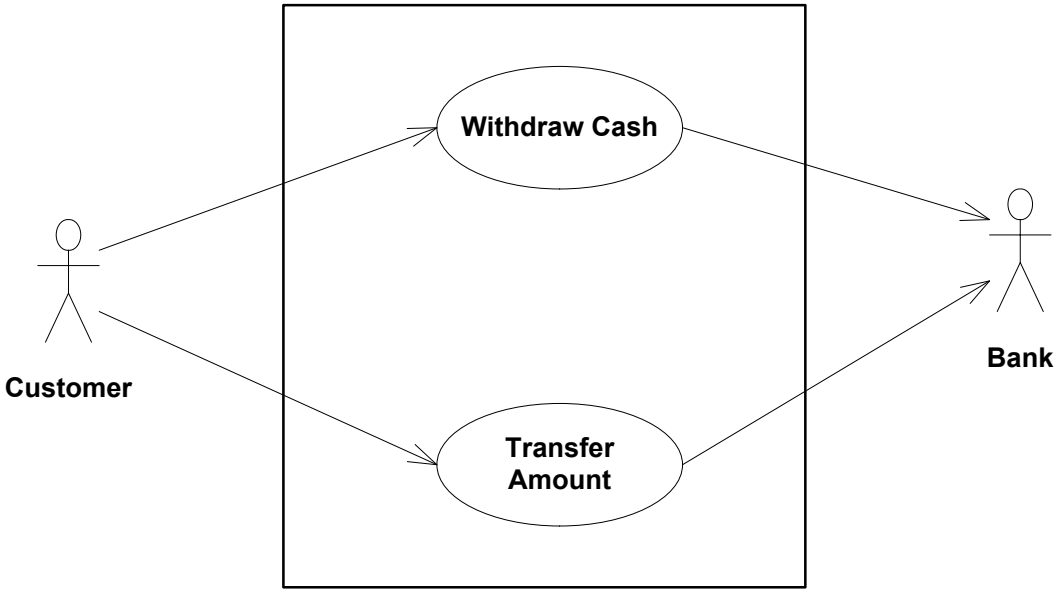
- *Control* klass**en** indeholder UC'ens *business logic*
  - Den styrer ("*executes*") UC'en ved at interagere med *boundary* og *domain* klasserne
  - Den skal have navn efter UC'en
  - Typisk er der 1 per UC eller 1 som deles mellem nogle få UCs
  - Bør forsynes med stereotypen «`control`»

Q&A Domænemodeller

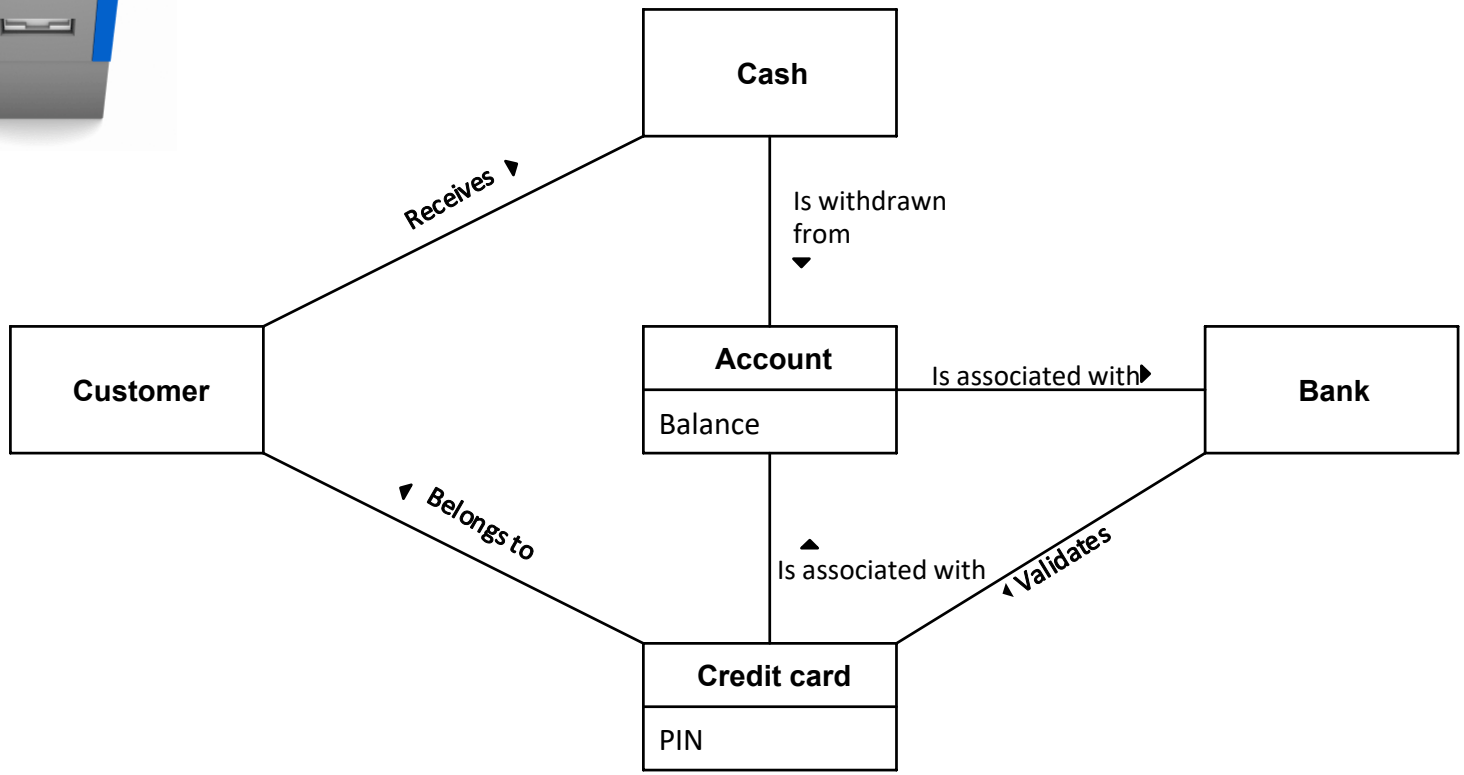
- Gå til padlet, link på BB

Kontant-/Bankautomaten
(ATM – Automatic Teller Machine)

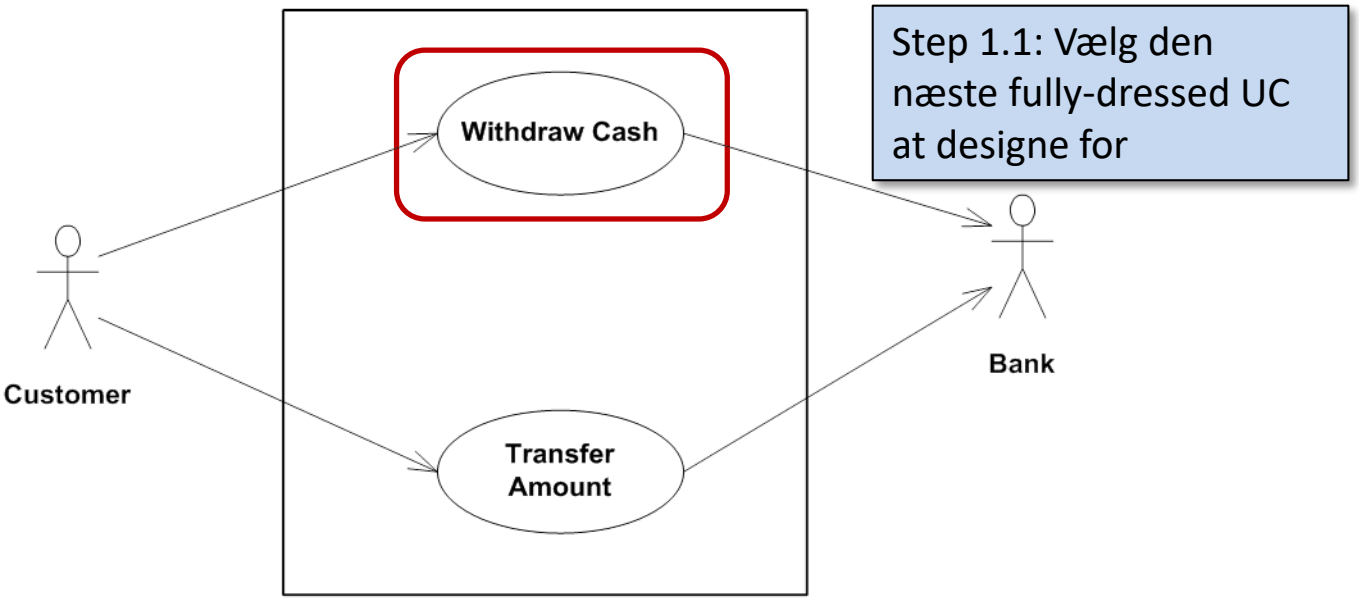
ATM Use Cases



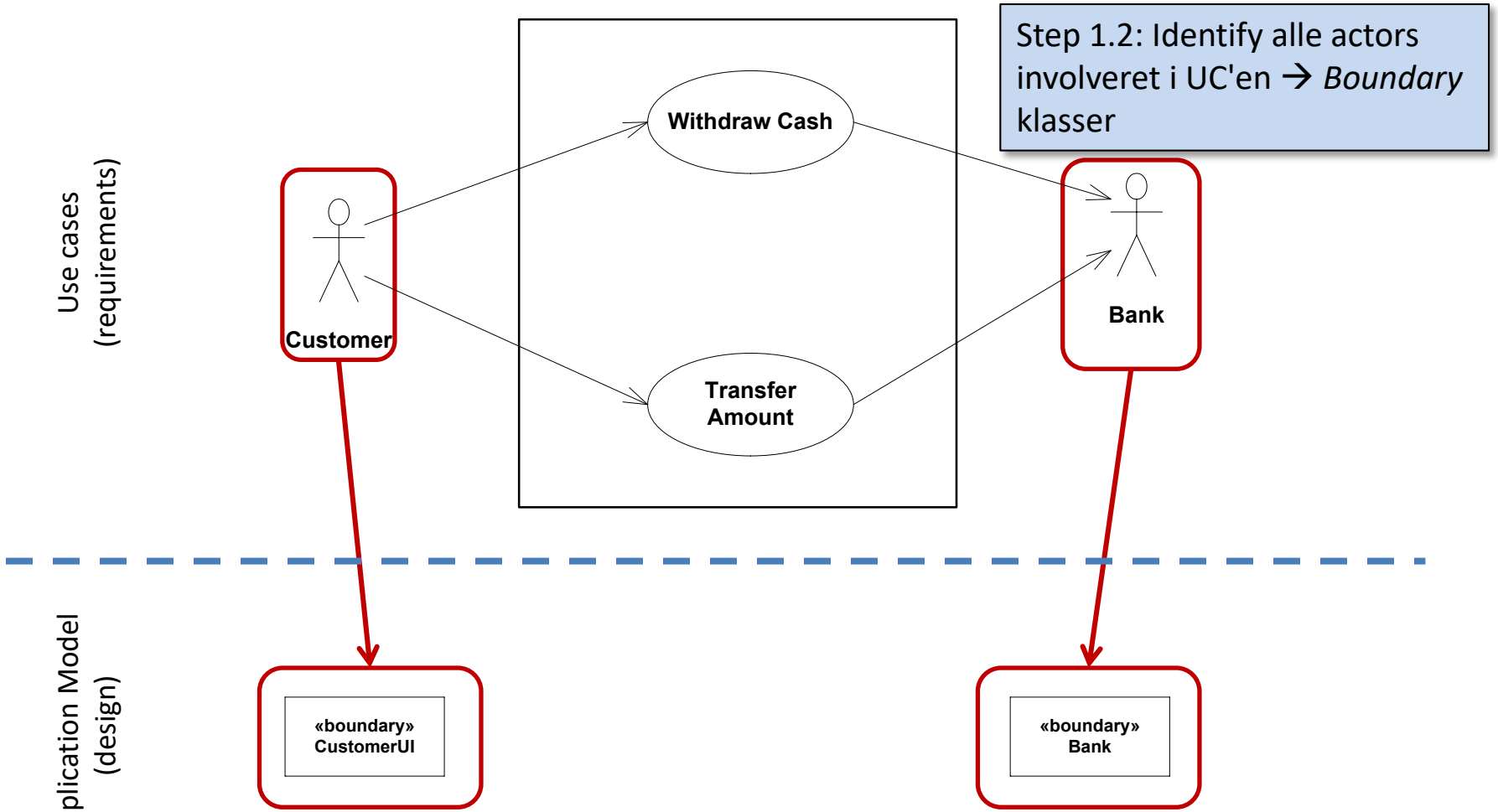
ATM Domænemodel



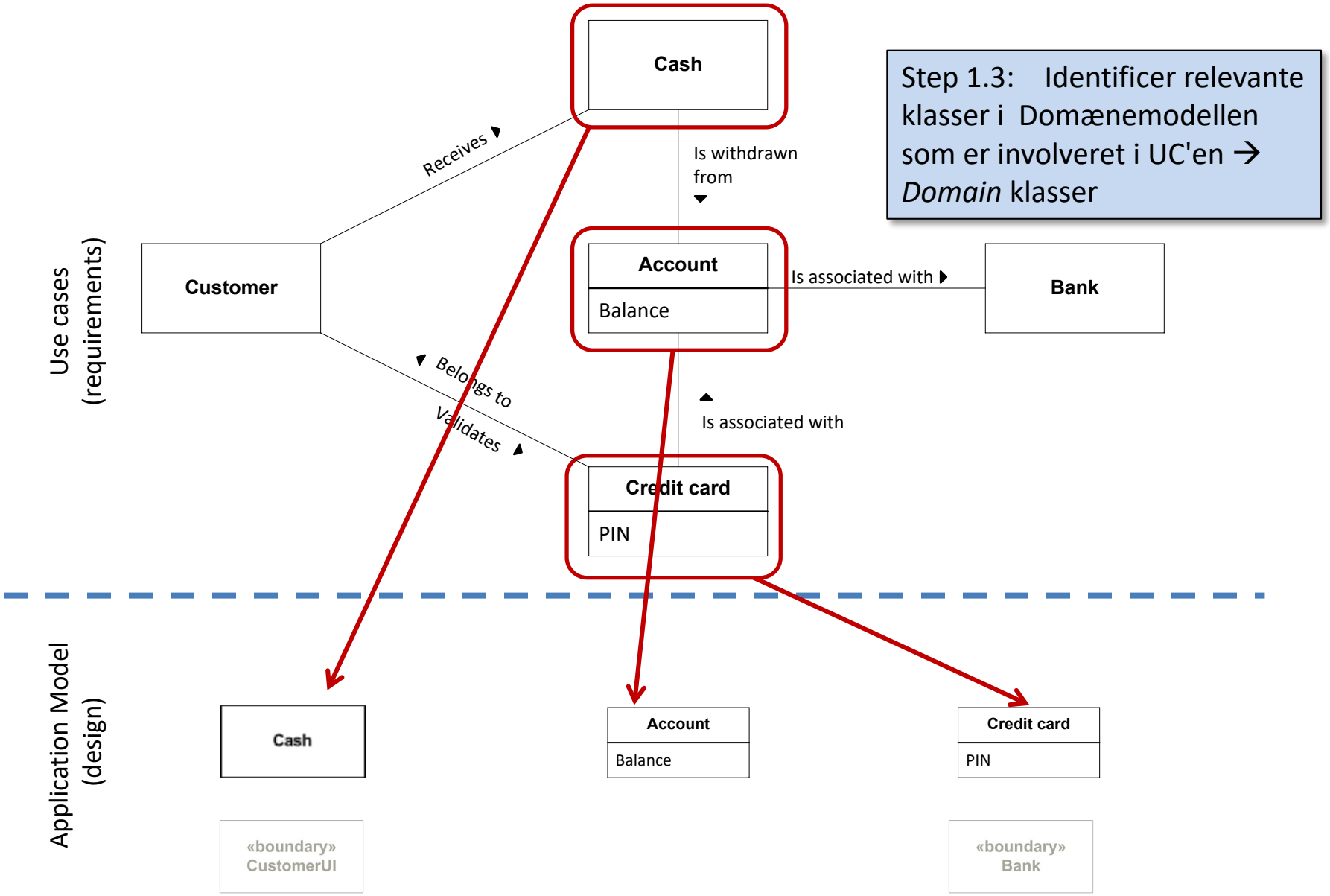
ATM step 1.1: Vælg den næste UC



ATM step 1.2: Actors -> *boundary* klasser

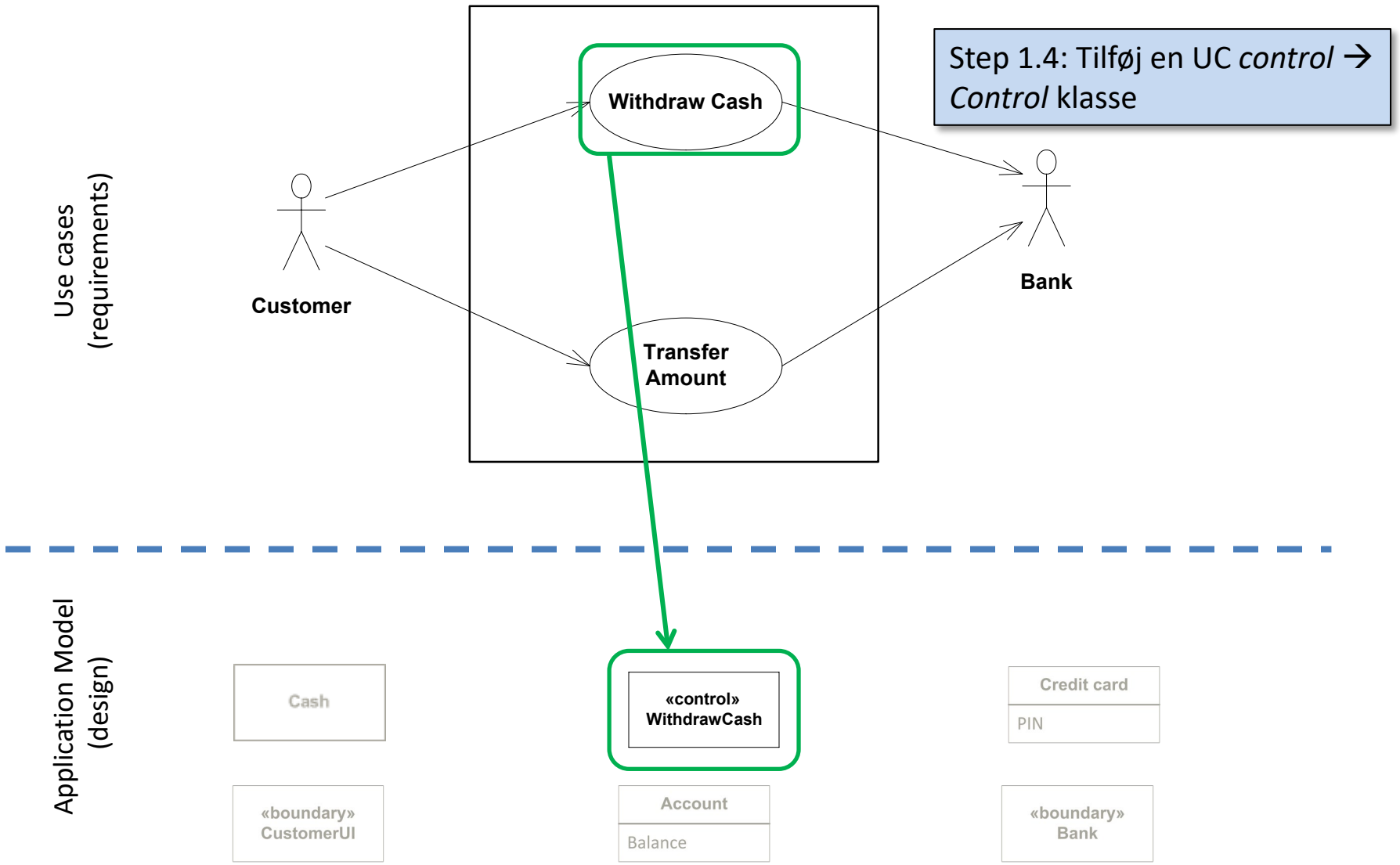


ATM step 1.3: *Domain* klasser



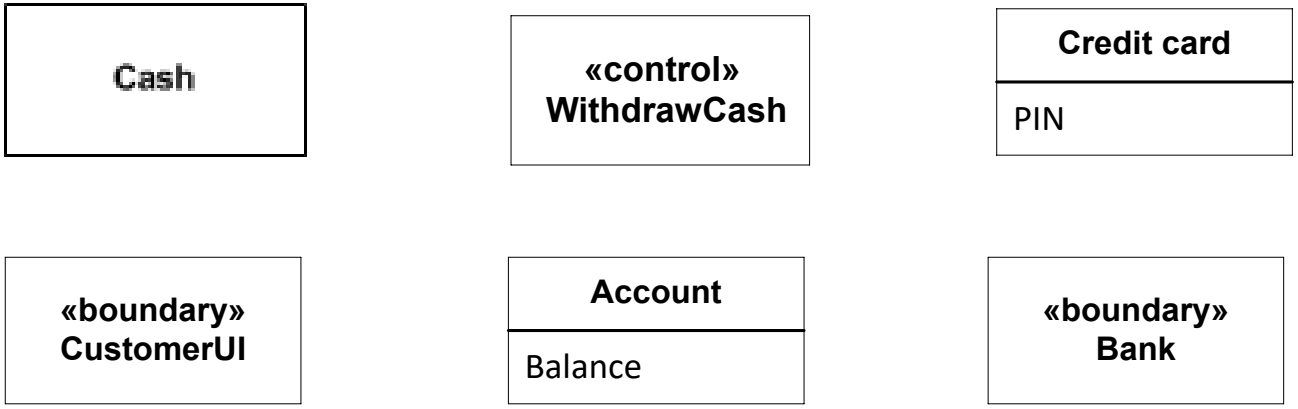
Step 1.3: Identificer relevante klasser i Domænemodellen som er involveret i UC'en → *Domain* klasser

ATM step 1.4
UC control -> *Control* class



Step 1 færdigt – så langt, så godt

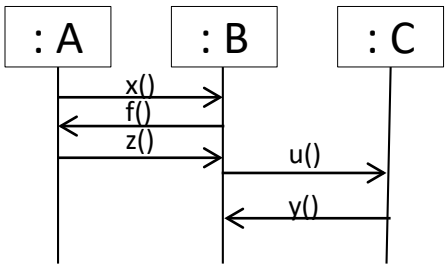
- Vi er nu færdige med Step 1 og har identificeret 6 kandidater som SW klasser for vores indledende design
- For at komme så langt, brugte vi vores *use case* og vores *Domænemodel*



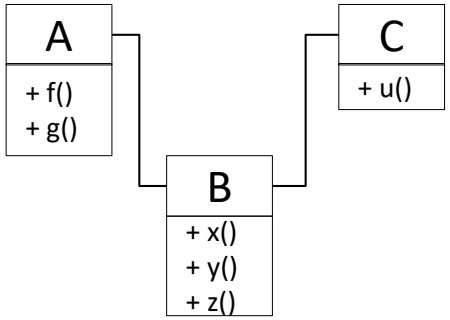
- Nu skal vi tilføje aktiviteter – det er Step 2

Principperne for step 2.1-4: Gå igennem hovedscenariet for UC og opdater løbende

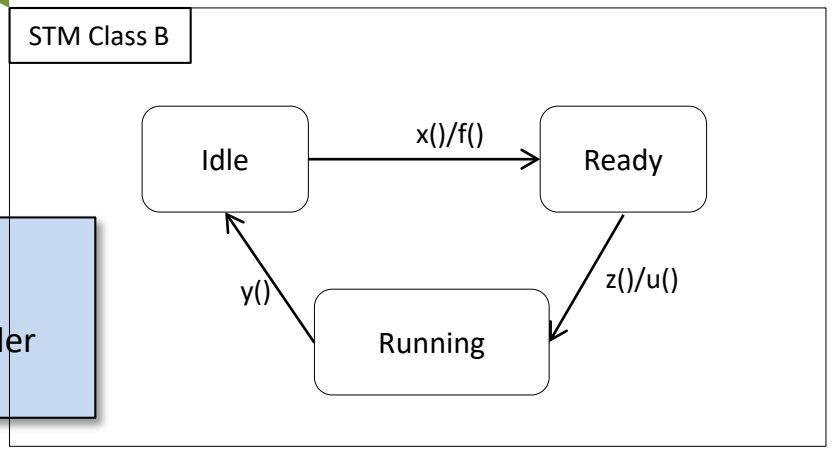
Sekvensdiagram:
- Tilføj kald af objekternes metoder



Klassediagram:
- Tilføj metoder til klasserne
- Opdater associationer



Opdateres parallelt!



Tilstands/aktionsdiagram:
- Lav det for de klasser der har tilstande
- Triggerne til transitionerne er kald til klassens metoder
- Aktionerne er kald til andre klassers metoder

Applikationsmodellen – Step 2

- Samarbejdet mellem klasserne udledes nu fra UC

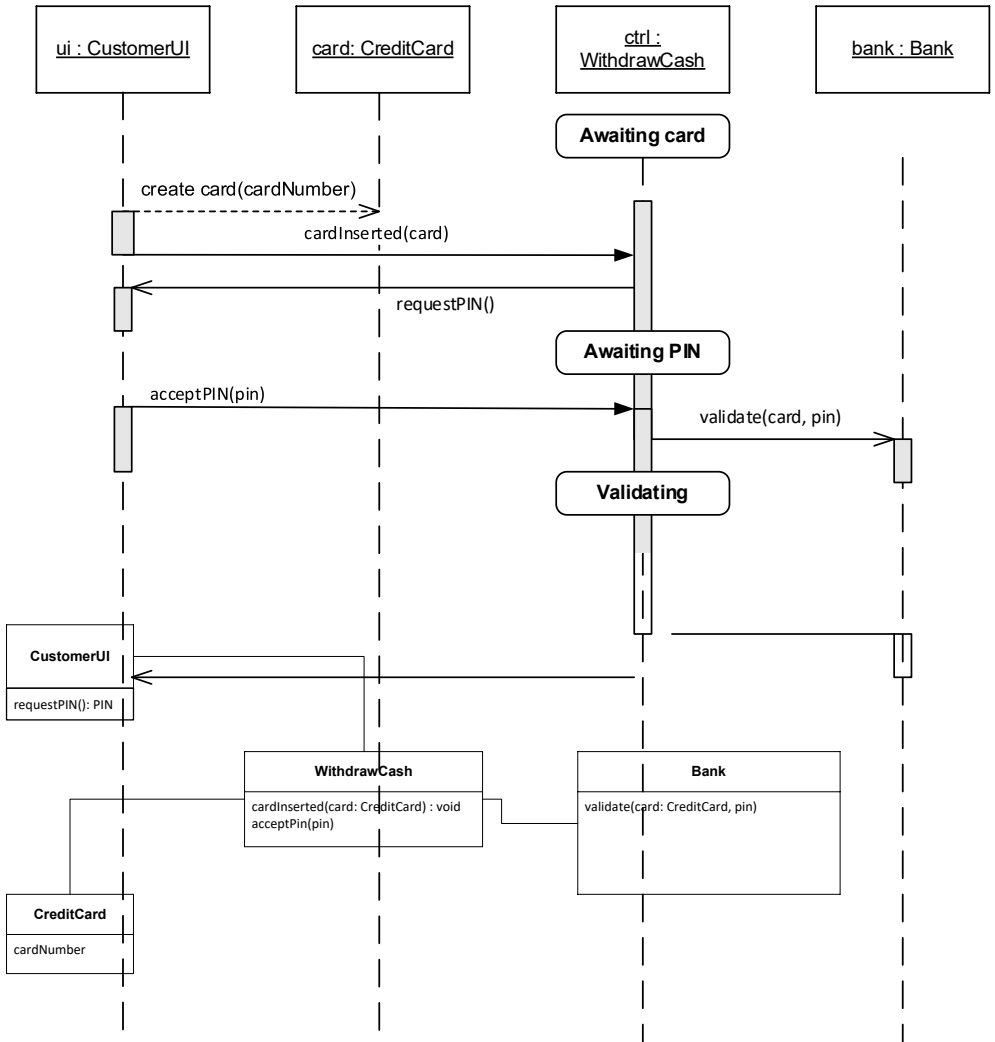
Step 2.1:	Gennemgå UC's hovedscenarie skridt-for-skridt og udtænk hvordan klasserne kan samarbejde for at udføre skridtet
Step 2.2:	Opdater sekvens- og klassediagrammet for at beskrive samarbejdet (metoder, associationer, attributter)
Step 2.3:	Hold øje med, om der er state-baserede aktiviteter og opdater STMs for disse klasser (tilstande, triggere, overgange, aktioner) *(Step 2.3 springes over hvis der ikke er nogen tilstandsbaserede klasser)*
Step 2.4:	Verificer at diagrammerne passer med UC (postconditions, test)
Step 2.5:	Gentag 2.1 – 2.4 for alle UC extentions. Finpuds modellen.

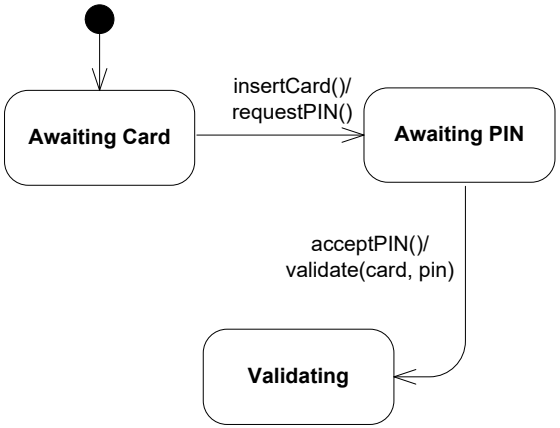
- Alle 3 diagrammer (cd, SEQ, STM) opdateres *parallelt/samtidigt* under dette arbejde

Steps 2.1-2.4 for UC *Withdraw Money*

Main scenario:

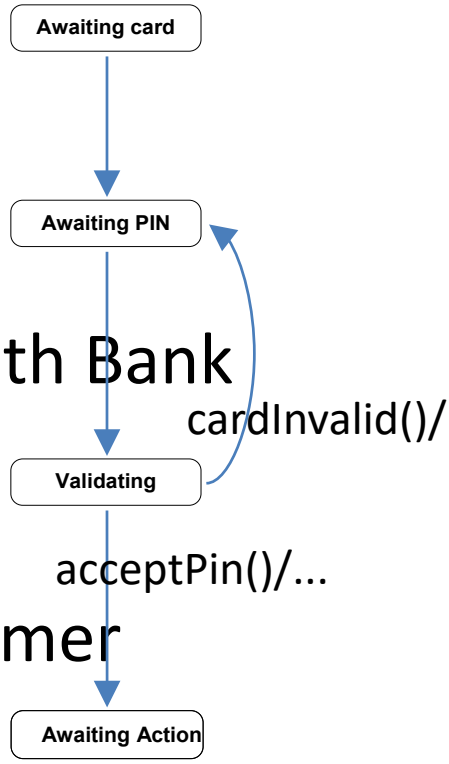
1. Customer inserts credit card in System
2. System requests Customer's PIN code
3. Customer enters PIN code
4. System validates card info and PIN code with Bank





Find STM fra hovedscenariet med 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. ...



Applikationsmodellen – hvad nu?

- Fortsæt med næste UC
- Efterhånden som man tilføjer flere UC'er vil man opdage at man kan genbruge nogle af de allerede fundne klasser
  - *Domain* og *boundary* klasser dukker ofte op igen
  - Forskellige *domain* klasser er måske så nært beslægtede, at de kan slås sammen til en
  - Nogle gange kan også *control* klasser slås sammen
- At tage det rigtige valg mellem genbrug, slå sammen eller introducere nye klasser kommer med erfaringen

Hvad så NU?
Applikationsmodellen bruges til det første kodedesign
Mere om dette i lektion 21 og 22

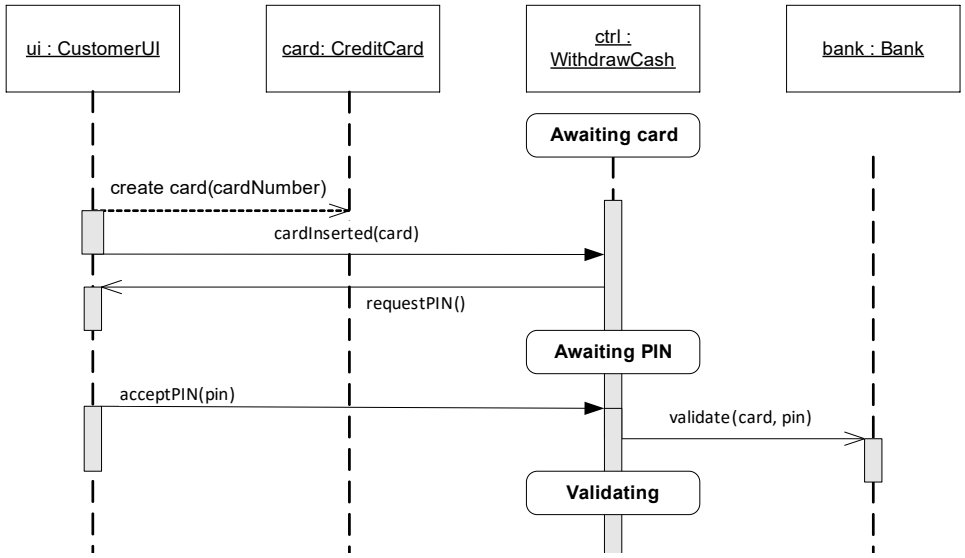
Så er det jeres tur: Gør Applikationsmodellen for UC *Withdraw Cash* færdig!

- Den fulde tekst for UC Withdraw Cash findes på BlackBoard. I skal:
  - Gøre Applikationsmodellen færdig for hovedscenariet for UC'en
  - Udbygge Applikationsmodellen med alle extensions for UC'en

- Udgangspunktet er de 3 diagrammer vi har lavet her ved gennemgangen – ses nedenfor
- Arbejd videre nu og hjemmearbejde til næste gang

Start øvelsen

- Starten på Sekvensdiagrammet



Start øvelsen

- Starten på klassediagrammet og tilstandsdiagrammet