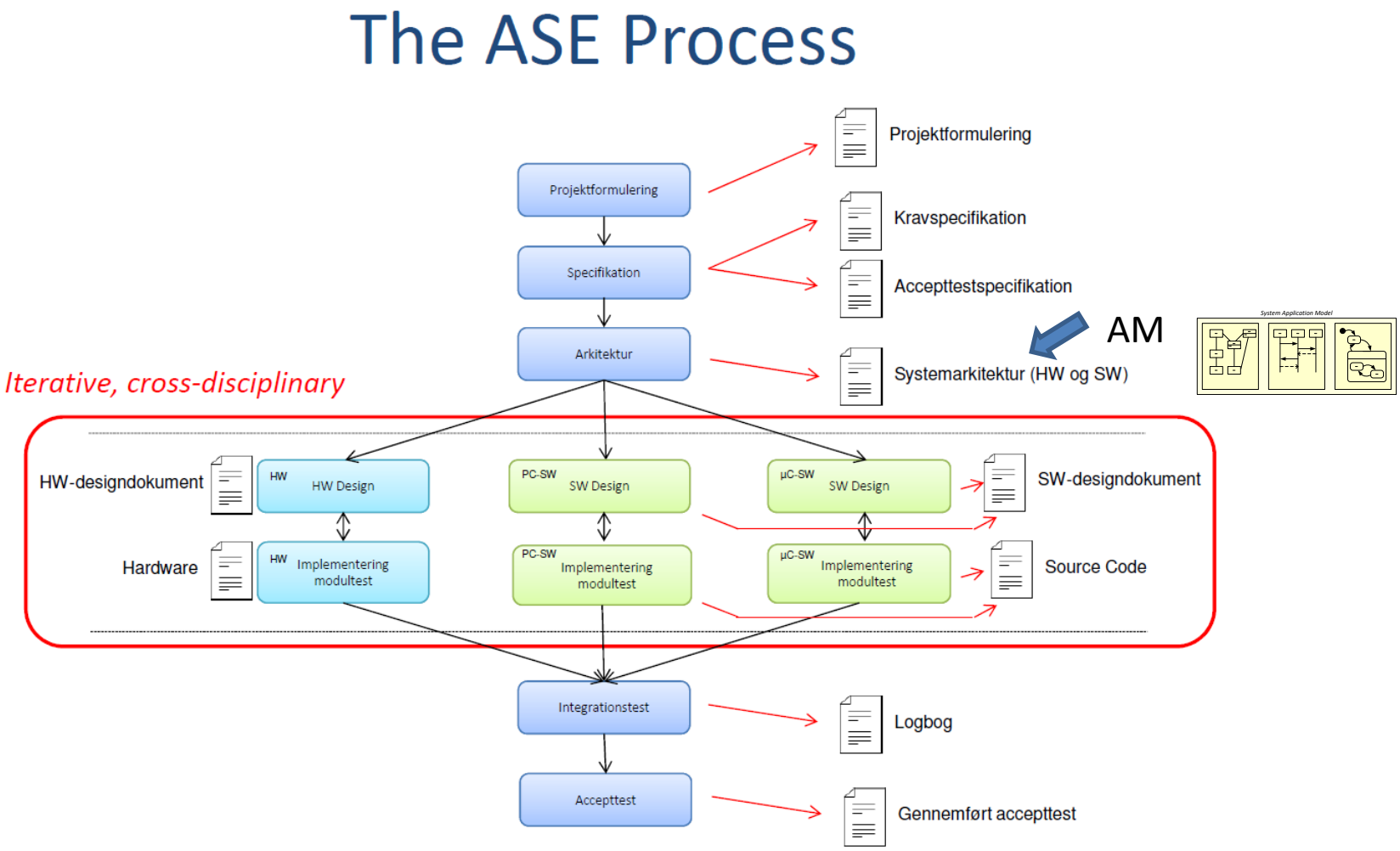
Applikationsmodeller
Del 2
Find de **rigtige** *boundary* klasser

I2ISE

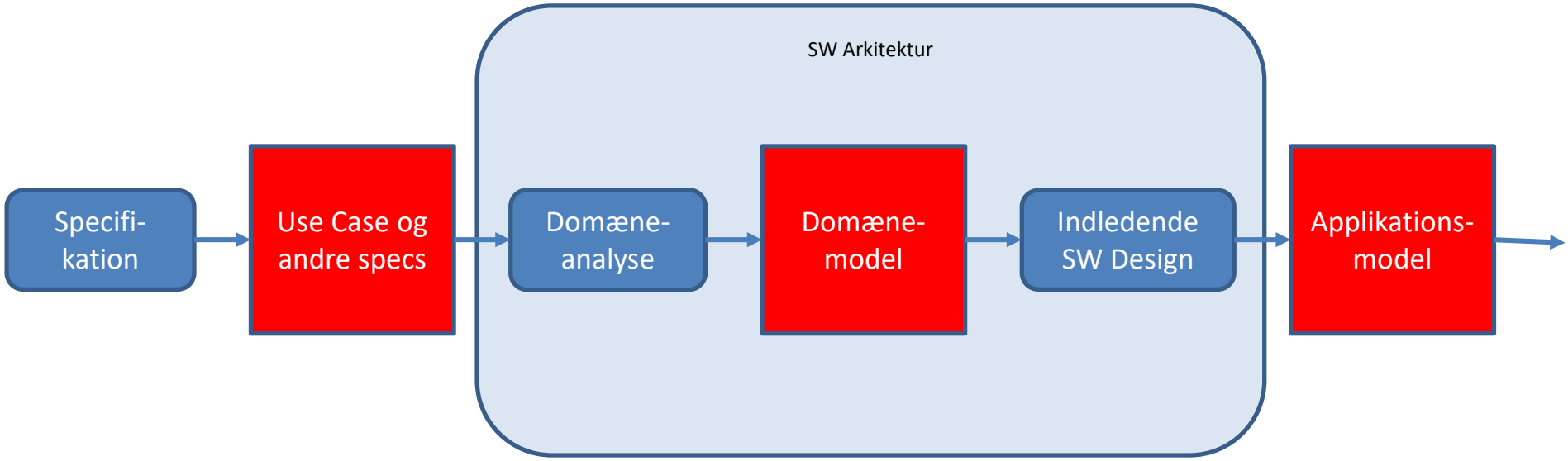
Dagens program

- Resumé
- Find nogle gode boundary klasser
- Regler og guide lines for applikationsmodeller
- Øvelse

AM's plads i dokumenterne



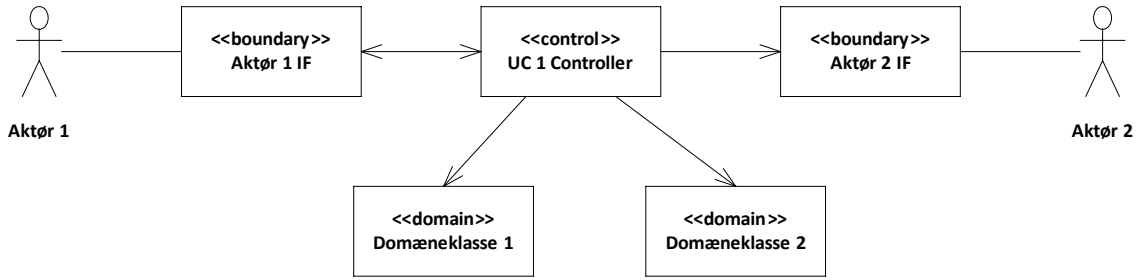
SW Arkitektur – fra UC til Design

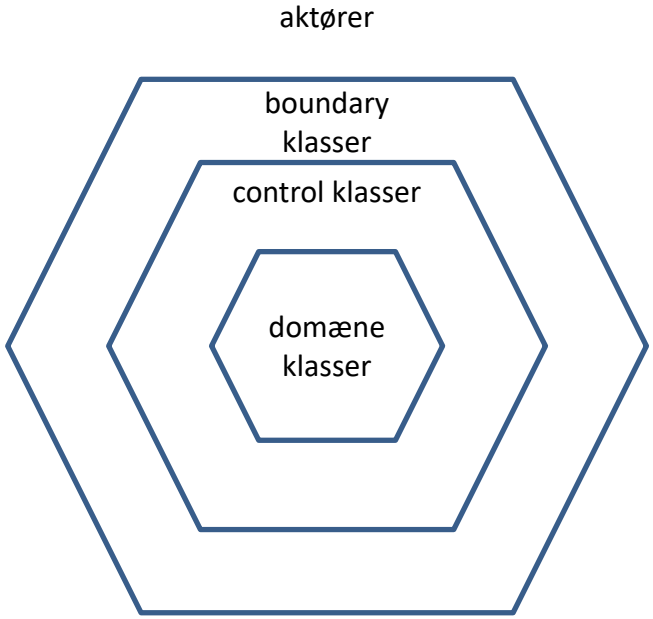


Vores arkitektur

- … 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

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`» or «`controller`»

Boundary klasser og hardware interfaces

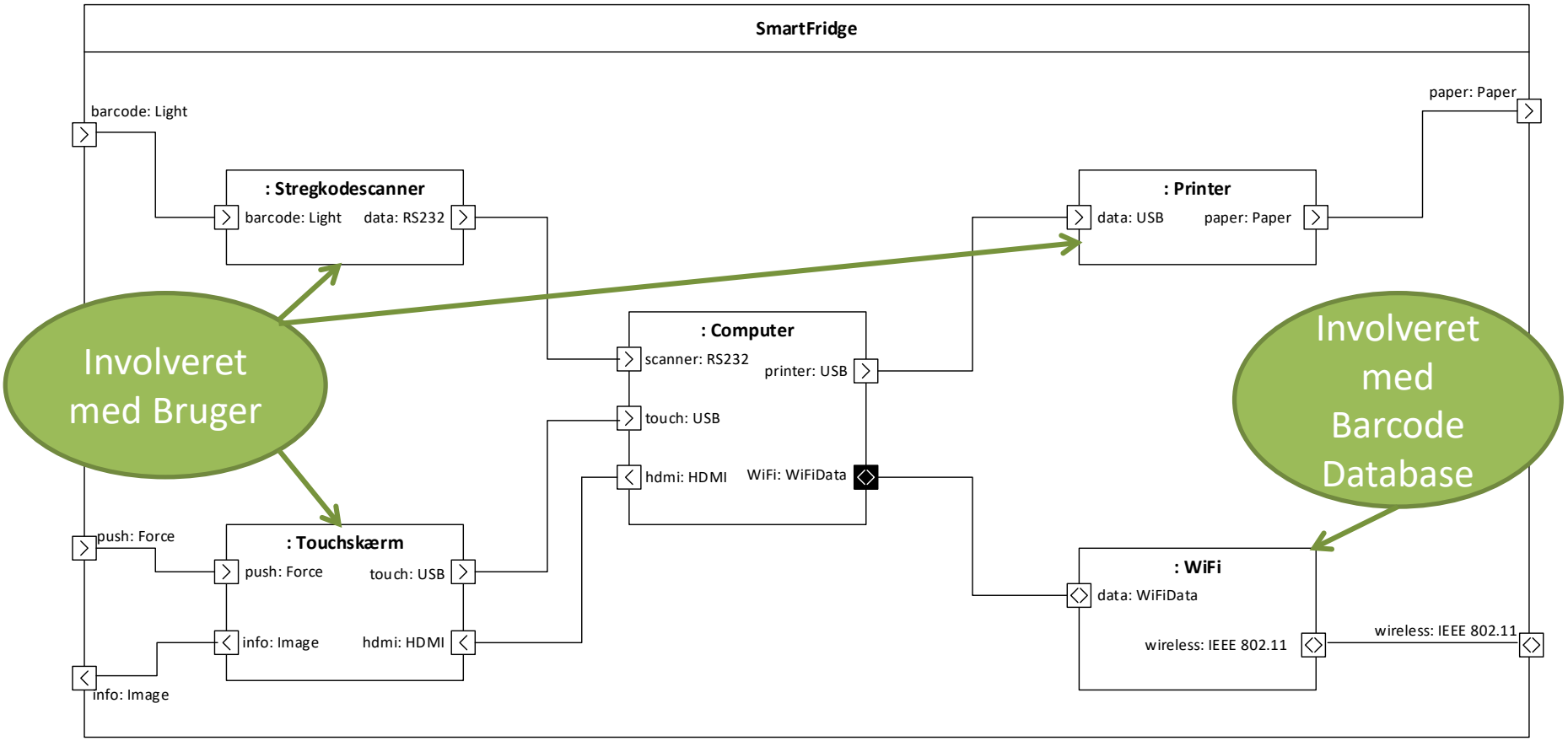
- ***Boundary* klasserne** er den del af softwaren, der tager sig af interfaces til ***Actors***
- Softwaren kører på computeren/microcontrolleren
- Så må interfaces til ***Actors*** være interfaces på computeren/microcontrolleren!
- Derfor: Led efter hardware interfaces der er involveret med ***Actors***!
- For hver af dem: Lav en ***boundary* klasse**!

Applikationsmodellen – Step 1 Version 2!

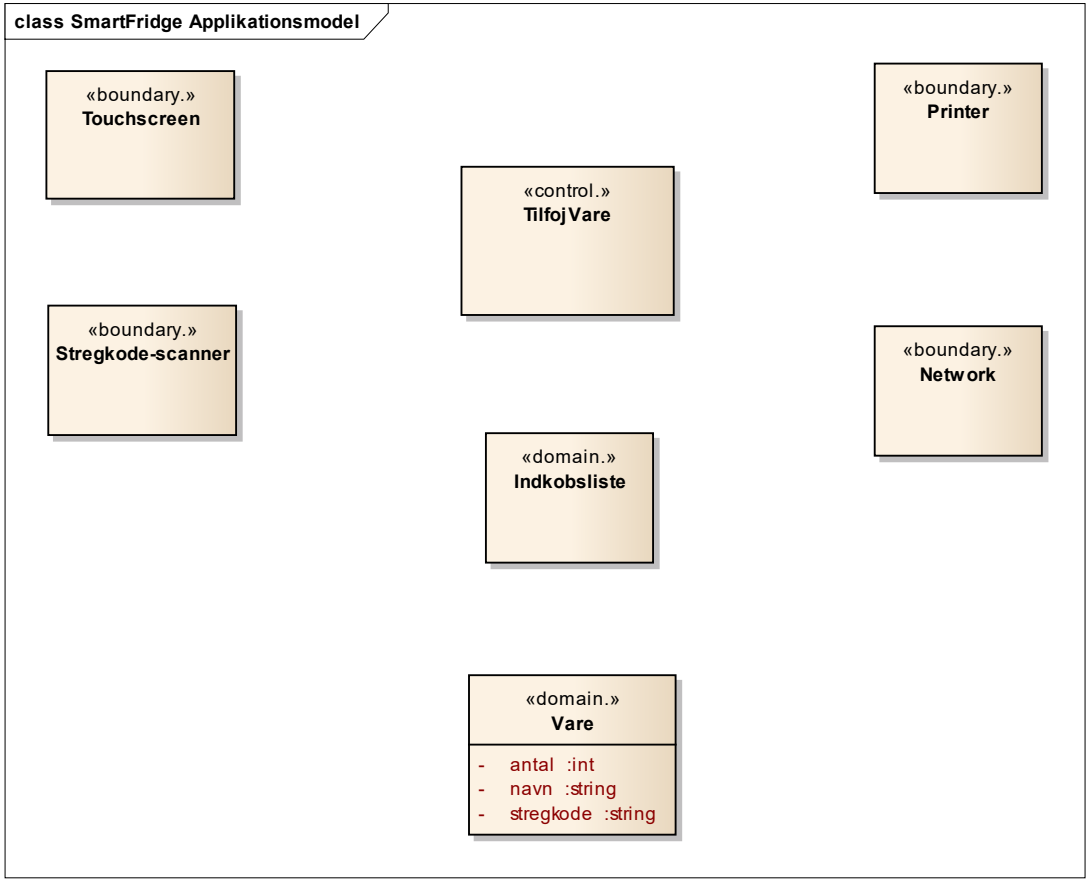
- Applikationsmodellen opbygges skridt for skridt, hvor hvert skridt styres af én UC

Step 1.1:	Vælg den næste fully-dressed UC til at designe for (hvordan?)
Step 1.2a:	Identificer alle involverede **aktører** i UC → ***Boundary* klasser**
Step 1.2b:	**Hvis man har et IBD som definerer de faktiske hardware interfaces: opsplit *Boundary* klasserne i relevante *hardware interface Boundary* klasser!**
Step 1.3:	Identificer **relevante klasser i Domænemodellen** som er involveret i UC → ***Domain* klasser**
Step 1.4:	Tilføj én UC *control* → ***Control* klasse**

Find Boundaryklasser



1. Version SAM klassediagram



Applikationsmodellen er en Softwaremodel!

- Derfor skal vi finde metoderne på klasserne
- Alle messages mellem objekterne på sekvensdiagrammet er metodekald! Derfor har de "()"
- Udtænk
  - et godt navn
  - parametre og parametertyper
  - returværdi (for synkrone kald som ikke er void)
  - synkron/asynkron

Interrupts vs. polling

Applikationsmodellen – Step 2

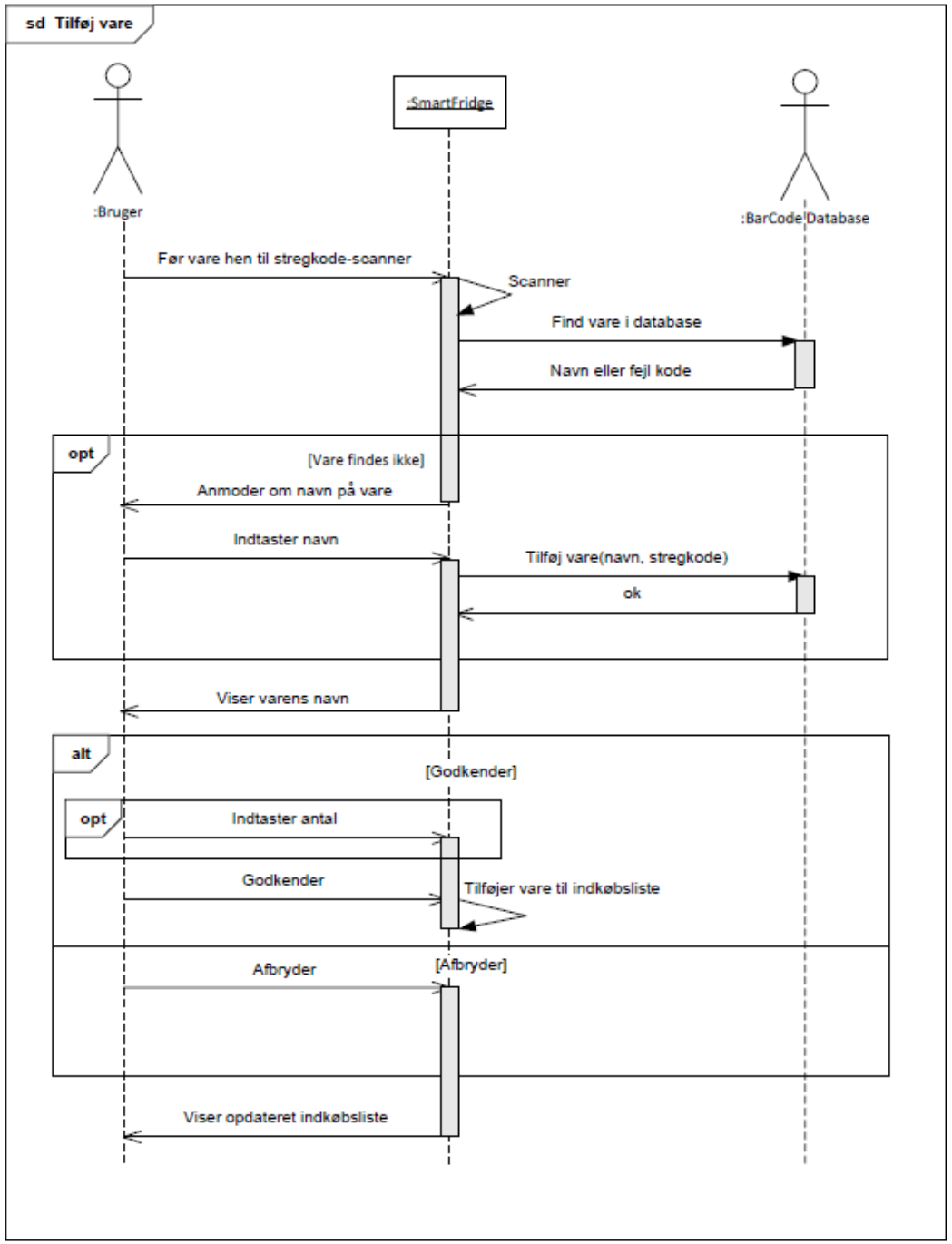
- Samarbejdet mellem klasserne udledes nu fra UC

Step 2.1:	Gennemgå UC's hovedscenarie skridt-for-skridt og udtænk hvordan klasserne kan samarbejde for at udføre skridtet
	Hvis man er heldig, har man et System SEQ, der viser UC – brug det!
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

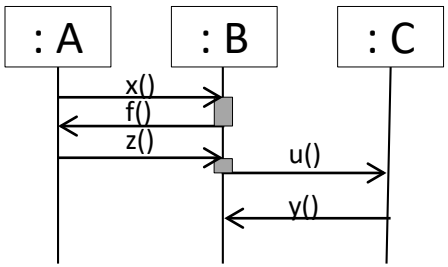
Hovedscenarie

Hovedscenarie	1. Bruger fører en vare hen til systemets stregkode-scanner 2. Systemet scanner varens stregkode 3. Systemet sender varens stregkode til BCDB 4. BCDB returnerer varens navn til Systemet *[Extension 1: Stregkoden findes ikke i BCDB]* 5. Systemet viser varens navn 6. Bruger redigerer og godkender antallet af varer *[Extension 2: Bruger afbryder tilføjelsen af en vare]* 1. Systemet tilføjer varens navn og antal til indkøbslisten 2. Systemet viser en opdateret indkøbsliste

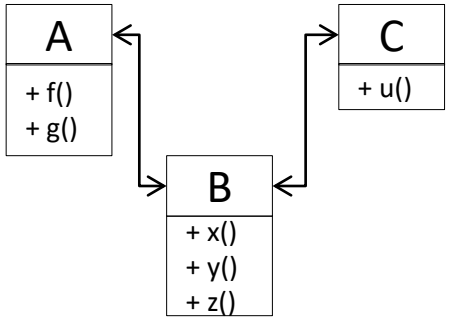

sd Tilføj vare
:Bruger
:SmartFridge
:BarCode Database
Før vare hen til stregkode-scanner
Scanner
Find vare i database
Navn eller fejl kode
opt
[Vare findes ikke]
Anmoder om navn på vare
Indtaster navn
Tilføj vare(navn, stregkode)
ok
Viser varens navn
alt
[Godkender]
opt
Indtaster antal
Godkender
Tilføjer vare til indkøbsliste
Afbryder
[Afbryder]
Viser opdateret indkøbsliste

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

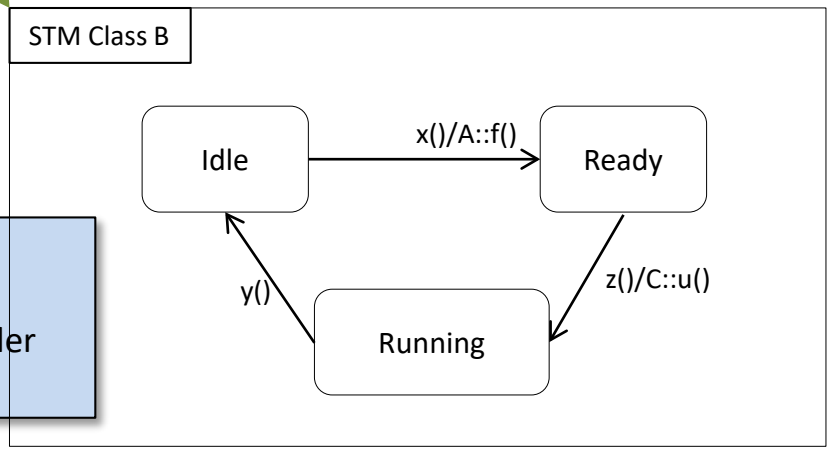
Sekvensdiagram:
- Tilføj kald af objekternes metoder



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

Kommunikationsregler

- Det er *control* klassen der tager alle logiske beslutninger – derfor gælder:
  - *Boundary* klasser kalder **KUN** til *control* klassen/r!
    - (og evt. nødvendige *domain* klasser der bruges som parametre (*Data Transfer Objects - DTO*))!
  - *Boundary* klasser kalder **IKKE direkte** til andre *boundary* klasser!
    - (medmindre man har en lagdelt *boundary* struktur)!
  - *Domain* klasser kalder **IKKE** til *boundary* klasser!
  - *Domain* klasser starter **IKKE** noget på eget initiativ!

Kommunikationsregler

Principielle kommunikationsregler i arkitekturen

	Til Boundary	Til Domain	Til Control
Fra Boundary	Nej!	Nej!	Ja!
Fra Domain	Nej!	Nej!	Nej!
Fra Control	Ja!	Ja!	Ja!

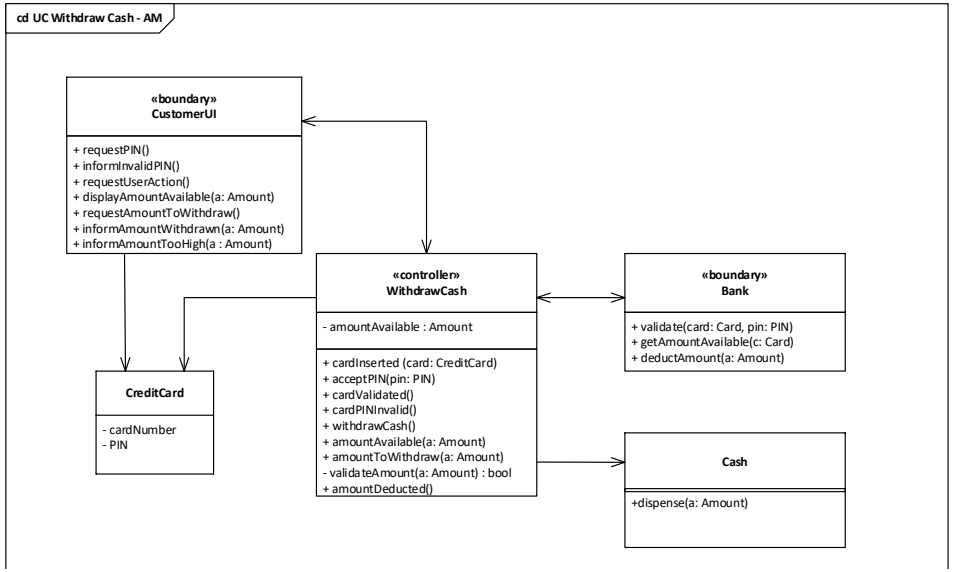
Pragmatiske kommunikationsregler i arkitekturen

	Til Boundary	Til Domain	Til Control
Fra Boundary	(Lagdelt I->I og O->O)	(Data Transfer Object)	Ja!
Fra Domain	Nej!	(Komposition)	Nej!
Fra Control	Ja!	Ja!	Ja!

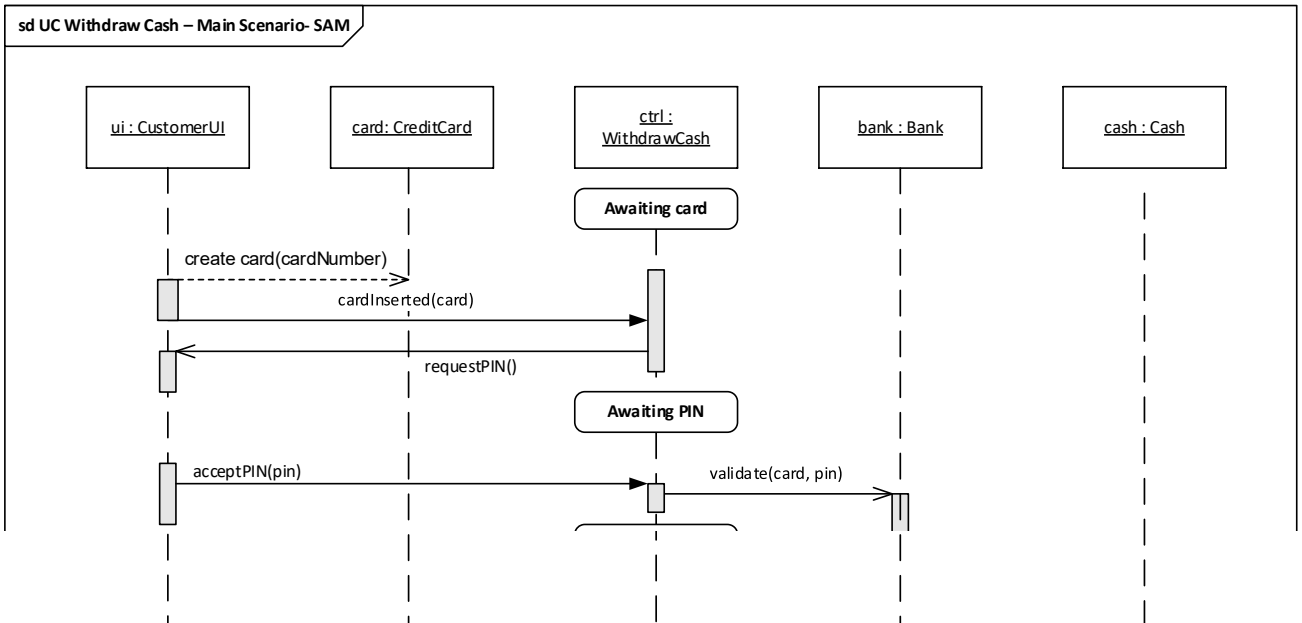
Designregler

- **Alle klasser** på **sekvensdiagrammet**, skal også være **på klassediagrammet** (men ikke nødvendigvis omvendt)
- **Alle metoder** skal tilknyttes **den klasse på klassediagramet**, der står for **enden af pilen på sekvensdiagrammet**
  - Det er et **metodekald** fra den ene klasse til den anden
- **Alle associationer** skal **pege samme vej** på **klassediagrammet**, som de gør på **sekvensdiagrammet**
  - For at den **ene klasse kan kalde den anden**, skal den have **en association til den**
- **Associationer kan være tovejs**, hvis begge klasser kalder den anden i løbet af UC

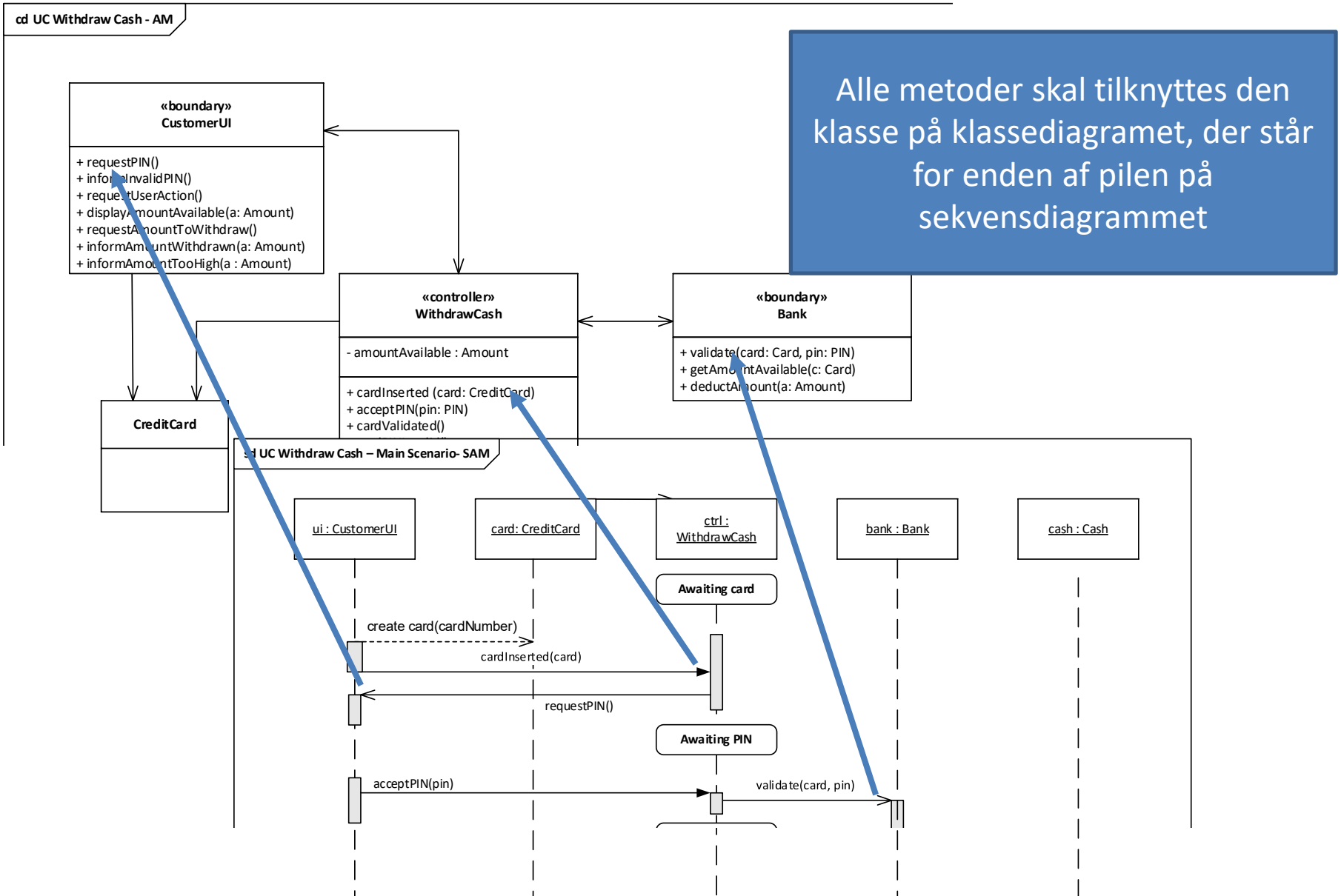
Designregler



Alle klasser på sekvensdiagrammet skal også være på klassediagrammet!

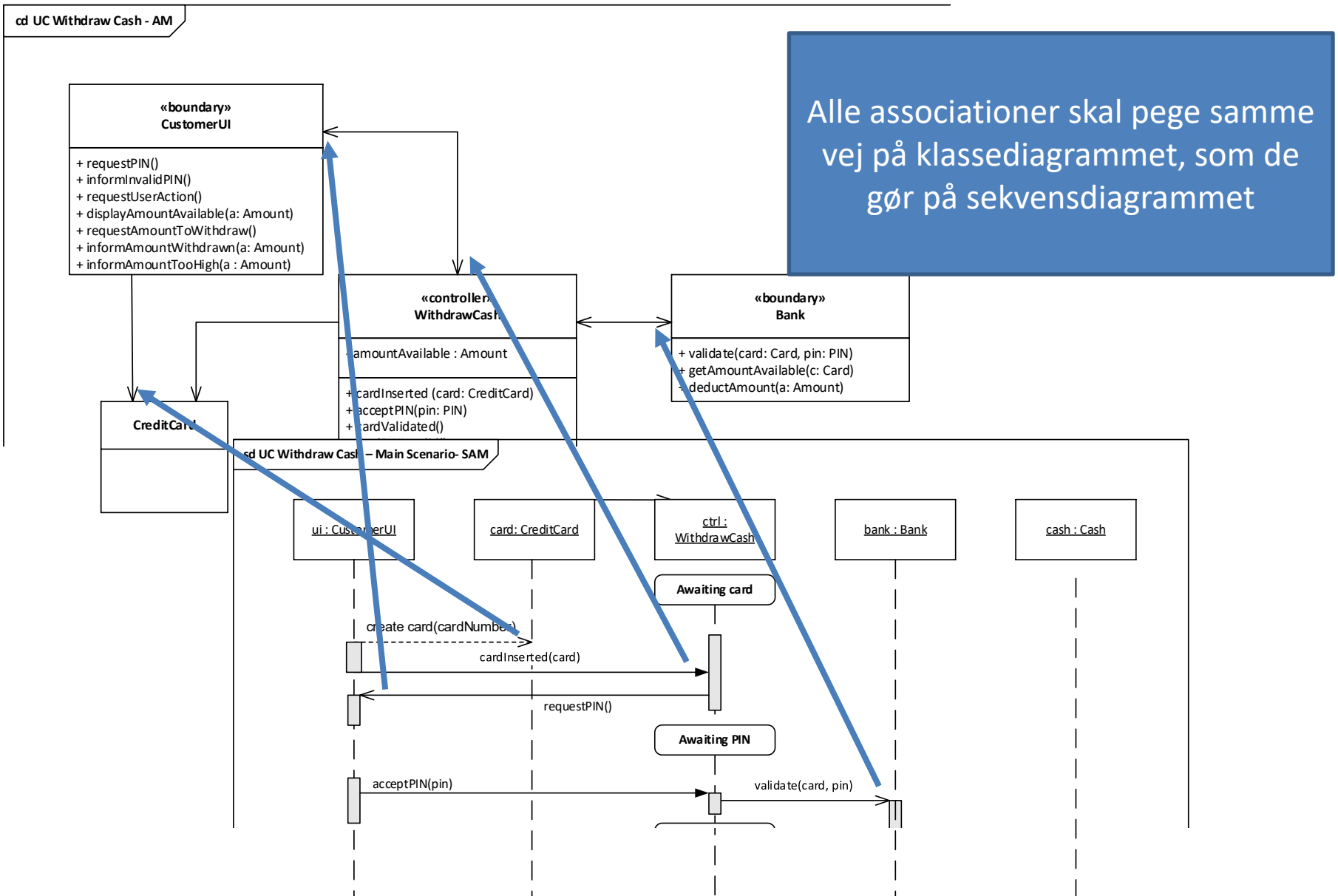


Designregler



Alle metoder skal tilknyttes den klasse på klassediagramet, der står for enden af pilen på sekvensdiagrammet

Designregler



Alle associationer skal pege samme vej på klassediagrammet, som de gør på sekvensdiagrammet

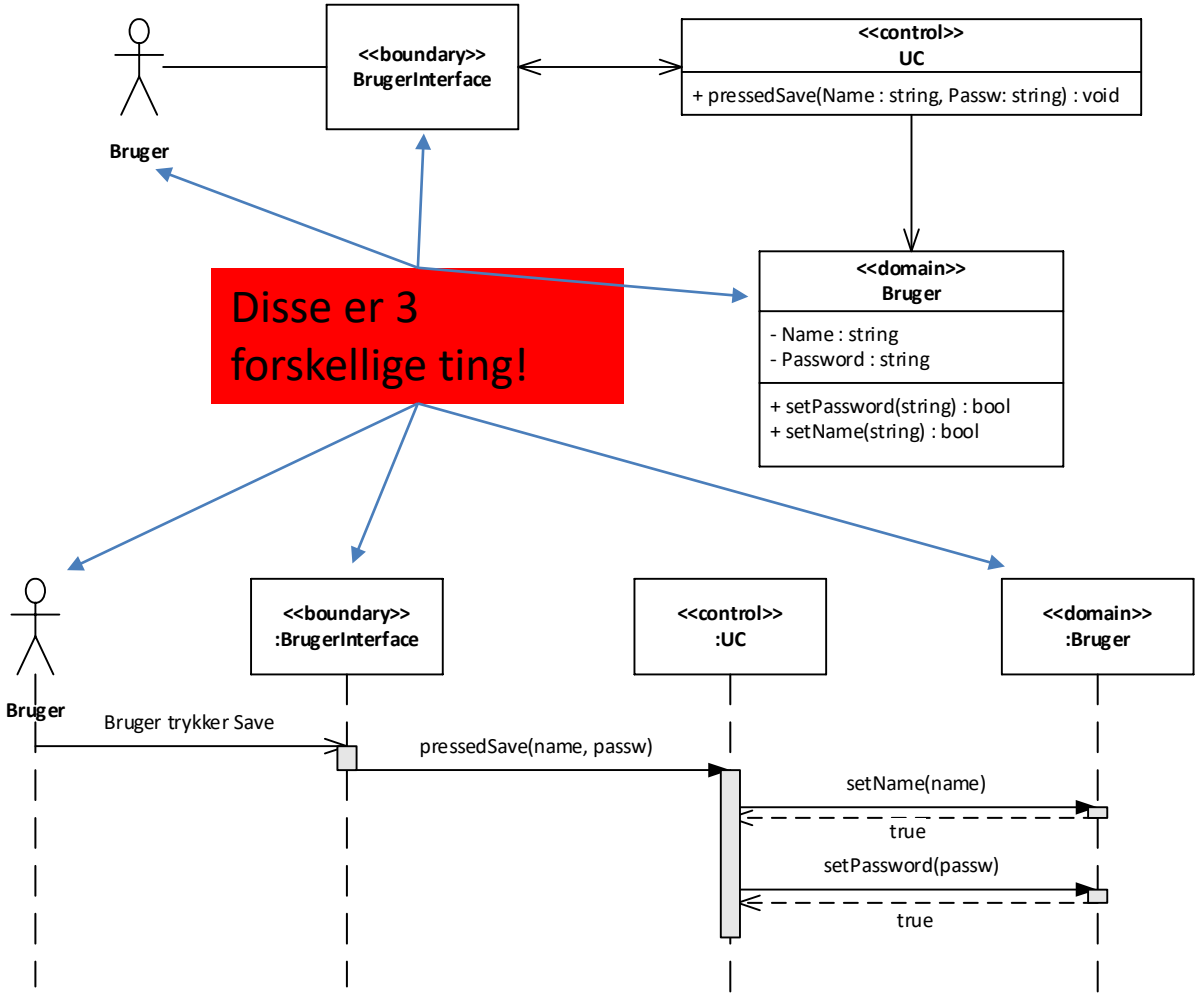
Guidelines "control"

- ***Control* klassen** er **IKKE** det samme som **hardware controlleren** eller **µ-controlleren** eller **”control unit” på BDD/IBD**!
- ***Control* klassen** er en del af **softwaren** som kører **på CPUen** i disse hardware controllers!
- ***Control* klassen** har navn efter den UC, den udfører!

Guideline Actor

- Man må gerne bruge en UC **Actor** (tændstiksmand) på Applikationsmodellens sekvensdiagram – **MEN:**
  - Den faktiske **Actor** og ***boundary* klassen/r** for aktøren og den ***domain* klasse** som bruges til at gemme attributter for aktøren – er 3 forskellige ting!
  - En faktisk **Actor har IKKE metoder** og **kan IKKE kalde metoder** – det kan kun ***boundary* klassen/r for** aktøren!
    - Ikke alle tegneværktøjer kan tegne messages uden () på sekvensdiagrammer :-(

Guideline Actor



Opgave:

- Udfør Step 2.1-2.5 for SmartFridge, mens I overholder designregler og andre guidelines