13
Objekt-orienteret Design.

Analyse i forhold til design.

Programbeskrivelse og designbeskrivelse.

Sømløs udvikling.

Design i forhold til OO Eiffel programmering.

Kategorisering af klasser i et design.

Design beskrivelsessprog.

- Statisk model.
- Dynamisk model.

PS1 -- Objekt-orienteret design

Noter

Dette kapitel udtrykker generelt nogle synspunkter og tanker om *design* fra et programmeringsperspektiv. Mere specifikt afspejler kapitlet nogle designmæssige forhold i forhold til det programmeringssprog, som vi er optaget af på dette kursus, nemlig Eiffel.

Analyse og programmering er er rimelig velforståede aktiviteter. Men det opleves af mange som lidt for brutalt at kaste sig over programmering efter en overstået analysefase. Derfor er det efterhåndet vundet frem, at efter analyser kommer der en design-fase, hvorefter man (endelig) kan programmere selve systemet. Set fra et lidt dogmatisk programmeringsperspektiv kan man godt påstå, at “design” blot er et nyt modeord for det som altid har været den store udfordring i programmering: at få struktueret sit program hensigtsmæssigt

Design-notationen, som foreslåes i sidste halvdel af dette kapitel, stammer fra artiklen *Jean-Marc Nerson*, "Extending Eiffel towards O-O analysis and Design".

Design i forhold til Analyse.

Analyse er rettet mod *problemet*.

- Forståelse af problemet.
- Løsrevet fra teknologi.

Design er rettet mod *løsningen* af problemet.

- Overordnet beskrivelse af programmet og dets udførelse.
  - Overskuelighed frem for detaljer.
- Sproguafhængig?
- Knyttet til teknologien, som løsningen baseres på.
  - Brugergrænseflade
  - Lagring af data

PS1 -- Objekt-orienteret design

Noter

Programmeringssprog er primært rettet mod beskrivelse af aspekter, som har en semantisk betydning. Undertiden er vi interesseret i -- under designbeskrivelser -- at beskrive aspekter, som ikke lader sig beskrive gennem programmeringssproget. Designsprogets opgave er således *også* at understøtte beskrivelse af væsentlige aspekter, sondringer og strukturer, som ikke kan fastholdes i programmet gennem programmeringssproget. Som eksempler kan nævnes følgende:

> Aggregering kan i Eiffel bedst beskrives ved brug af ekpanderede typer, men i visse situationer vil vi også ønske at realisere aggregering ved brug af referencer. Dette kan ikke udtrykkes i programmeringssproget.
>
> I OO programmeringssprog er klasser organiserede i klassehierarkier. Der er dog også behov for andre organiseringer, såsom gruppering. Eiffel sproget understøtter ikke dette. (Til gengæld understøtter Eiffel omgivelsen klynger af klasser).

Nogler forfattere mener, at sproguafhængighed i designbeskrivelserne er væsentlig. Som det vil fremgå af det følgende, kan der sige både "for og imod" omkring dette.

Design- og programbeskrivelser (1).

Genstandsområde:	statiske aspekter	dynamiske aspekter
Formaliseringsgrad:	formel	uformel
Udtryksform:	tekstuel	grafisk
Detaljeringsgrad:	detaljeret	overordnet

	Program		*Design*	
Genstandsområde:	x		x	x
Formaliseringsgrad:	x		x	
Udtryksform:	x		(x)	x
Detaljeringsgrad:	x			x

Noter

Det er væsentlig ikke at misforstå sondringen mellem statiske og dynamiske aspekter:

De dynamiske aspekter er knyttet til *programudførelse*. De statiske aspekter er knyttet til vores traditionelle forståelse af *programbeskrivelse*. I den statiske programbeskrivelse centreres interessen om klasser; når vi tænker konkret om et objekt-orienteret program centreres interessen om objekter. Det kan være et problem i vore beskrivelser, at objekter kun optræder indirekte i vore beskrivelser. Man kan forestille sig beskrivelsesformer, hvor objekter optræder på en langt mere direkte måde.

En formel beskrivelse kan gives en præcis betydning (en præcis semantik). Dette er langt vanskeligere for en uformel beskrivelse, der som oftest er baseret på formuleringer i naturlig sprog.

Nederst på denne side er vist, hvordan programbeskrivelser og designbeskrivelser *typisk* vil fordele sig på de fire beskrivelseskarakteristika.

Design- og programbeskrivelser (2).

Sproget, som anvendes til udtrykkelse af designet er

1. *uafhængigt af programmeringssproget.*
   a. designet kan programmeres i mange forskellige sprog.
   b. risiko for usammenligneligheder mellem design- og programmeringssprog.
2. *afhængig, men separat fra programmeringssproget.*
   a. sikrer god sammenhæng mellem programmeringssprog og designsprog.
   b. giver god mulighed for "forward-" såvel som "reverse enginering".
3. *en del af programmeringssproget*
   a. i en simple programmeringsomgivelse bliver designsproget tekstuelt, og en delmængde af programmeringssproget.
   b. i en avanceret programmeringsomgivelse kan program- og designbeskrivelserne være forskellige *syn* af den samme interne repræsentation.

Noter

På denne slide diskuterer vi tre mulige forhold mellem designbeskrivelser og programbeskrivelser.

Ad 2. Der er en sammenhæng mellem designsprog og programmeringssprog, som udmønter sig i, at ethvert design kan transformeres til et programskelet, og ethvert program kan transformeres til et design.

Man skal bemærke, at i og med, at der er tale om to forskellige sprog i hver deres repræsentation, vil et design og et program kunne blive inkonsistente.

"Forward engineering" betyder i denne sammenhæng, at en designbeskrivelse transformeres til et skelet af en programbeskrivelse. "Reverse engineering" betyder, at en designbeskrivelse kan udtrækkes af et program. Sidstnævnte er f.eks. relevant, i det tilfældet at et program er fremkommet "på gammeldags vis", uden udafbejdelse af egentlig designbeskrivelse. Reverse engineering er også nyttig, hvis et design er transformeret til programbeskrivelse, og derefter videreudviklet med hensyn til aspekter, som påvirker designet. Det er da nyttigt, at kunne udtrække designet fra programmet.

Ad 3. Idet design og program her udtrykkes i det samme sprog (eller i det avancerede tilfælde deler intern repræsentation) er der ikke som oven for risiko for inkonsistens mellem programmet og designet.

Sømløs udviklingsproces.

En udviklingsproces kaldes sømløs (“Seamless”) hvis der ikke er kantede og ubehagelige overgange mellem processens faser.

Betegnelsen er en metafor, som nok stammer fra tekstilbranchen snarere end fra tømrer/snedkerbranchen.

Sømløshed i udviklingsprocessen fra analyse til programmering:

- Udnyttelse af objekt-orienterede ideer hele vejen igennem giver færre søm.
- Tæt tilknytning mellem design- og programmeringssprog giver færre søm.
- Befordrer genbrugelighed mellem processens faser.
- Øger produktiviteten.

Noter

“Sømløshed” er et forsøg på oversættelse af den mere og mere hyppigt anvendte engelske betegnelse “seamlessness”.

Man kan observere sammenhængen mellem sømløshed og den tredie af punkterne på den forrige slide om integration af programmeringssprog og designsprog. Jo tættere de to formalismer er på hinhanden, jo færre søm bliver der mellem disse faser i udviklingsprocessen.

Hvilke dele af OO Programmering fra PS1 er Design?

- Klasser som dataabstraktion
- Klasseinvariant samt pre- og postbetingelser af eksporterede operationer
- Klassens klientinterface
- Organisering af klasser i et nedarvningshierarki
- Abstrakte klasser
- Generiske klasser

Grov formulering:

En samling Eiffel klasser uden kroppe af operationer udgør en designbeskrivelse.

Abstrakte præsentationer, som er resultatet af ***short*** , er designbeskrivelser.

Noter

På denne side diskuterer vi situation 3 fra forrige slide. Altså situationen, hvor Eiffel anvendes som et design sprog. Listen af sproglige egenskaber, som er opremset øverst i billedet, angiver hvilke dele af Eiffel, der bidrager til et designsprog.

Klasserne i Objekt-orienteret Design.

Problem-orienterede klasser

- Stammer fra analysen.
- Skal bearbejdes, udvides og reorganiseres i designfasen.

Basisklasser

- Klasser som afspejler fundamentale datastrukturer.
- Klasser som understøtter ekstern lagring.

Applikationsklasser

- Ikke dataabstraktioner, men derimod indkapslinger af applikationens procedurelle aspekter.

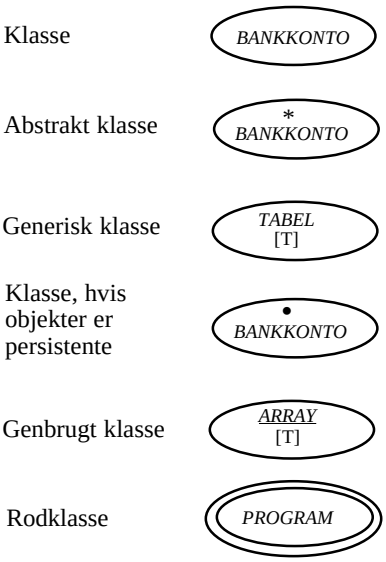
Brugergrænseflade klasser

PS1 -- Objekt-orienteret design

Noter

Beskrivelsessprog til OO design.

Statisk model: Klasse symboler.



PS1 -- Objekt-orienteret design

Noter

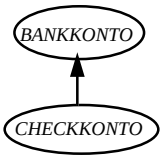
Det grafiske sprog, hvis elementer bliver beskrevet på denne og følgende sider, er knyttet til Eiffel. Sproget er beskrevet i artiklen "Extending Eiffel Towards O-O Analysis and Design".

En komplicerende faktor i designbeskrivelsessproget er håndteringen af generiske klasser. Som det vil fremgå, kan disse indgå i en del forskellige relationer med ikke generiske såvel som generiske klasser.

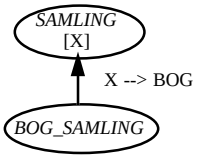
Vi gør ikke noget forsøg på i disse noter at komme op med speciel notation til beskrivelse af operations interfacet af klasser og de tilknyttede assertions . Vi foreslår, at vi bruger Eiffel og det tilknyttede dokumentationsværktøj (**short**) til dette formål.

Beskrivelsessprog til OO design.
Statisk model: Nedarvningsrelationer.

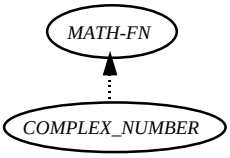
Generalisering/specialisering:



Nedarvning fra generisk klasse:



Egenskabstilgængelighed:



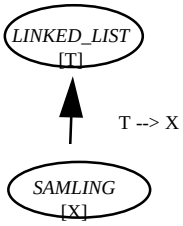
PS1 -- Objekt-orienteret design

Noter

Relationen "egenskabstilgængelighed" er vist grafisk som en pil, tegnet med en stiplet linie. Egenskabstilgængelighed afspejler en nedarvningsrelation mellem to klasser A og B (såsom math_fn og complex_number), hvor B ikke er en specialisering af A. Vi har tidligere i disse noter talt om at stille egenskaber direkte tilrådighed ved brug af nedarvning.

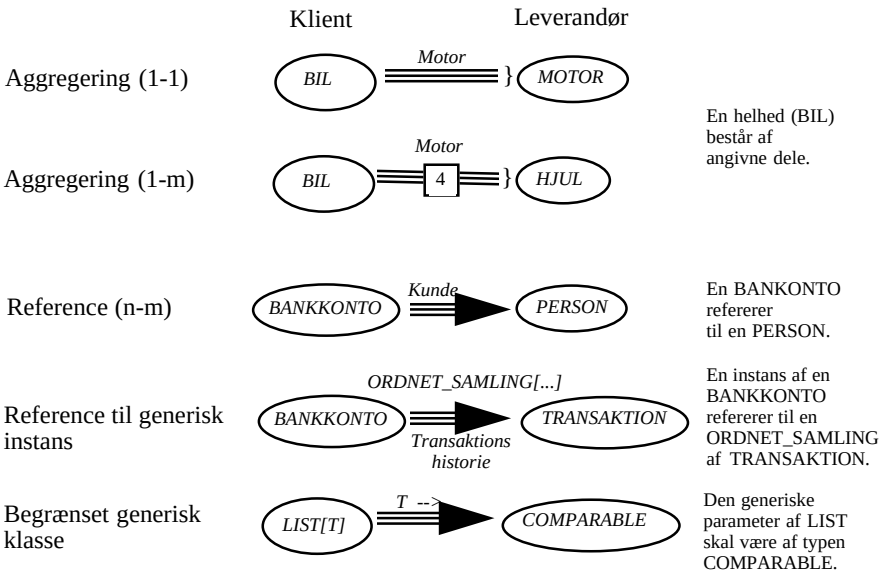
Nedarvning fra generisk klasse kan også angives med stiplet pil.

Der er ikke noget til hinder for, at både sub- og superklasser kan være generiske:



"Egenskabstilgængelighed" og "nedarvning fra generisk klasse" er nye i forhold til designsproget beskrevet i artiklen "Extending Eiffel towards O-O analysis and Design".

Beskrivelsessprog til OO design.
Statisk model: Klient-leverandør relationer.



Noter

Følgende er et alternativ til ovenstående "reference til generisk instans":

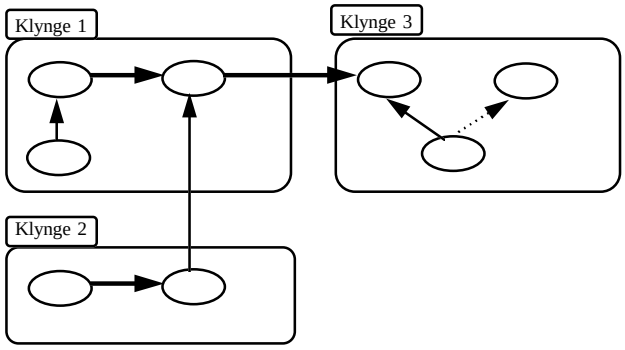
Bemærk endvidere, at vi tilsvarende kan angive "aggregering af generisk instans" ved blot at erstatte pilen med "tuborg".

Beskrivelsessprog til OO design.

Gruppering af klasser.

Gruppering af klasser anvendes til angivelse af

- subsystemer
- en mængde af klasser, som har en nær sammenhæng



PS1 -- Objekt-orienteret design

Noter

På denne slide kalder vi en gruppe af klasser for en klynge. Husk, at klynger er blevet introduceret i forbindelse med Eiffel programmeringsomgivelsen, med det formål at beskrive den samling af klasser, der skal tages i betragtning under oversættelse af et Eiffelprogram.

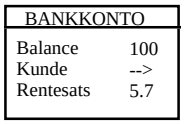
Beskrivelsessprog til OO design.

Dynamisk model.

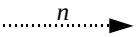
I den dynamiske model vises objekter og deres kommunikation med andre objekter.

Kan anvendes til beskrivelse af kommunikations-scenarier mellem objekter.

Instans af klasse (objekt) evt. med felter



Nummereret kommunikation mellem to objekter



n: forklarende tekst.

Noter

Det er mindre præcist formuleret på denne side, hvori en beskrivelse af den dynamiske model egentlig består. Vi vil tænke forholdsvist frit om en sådan. Det er målsætningen at sættes os i stand til at beskrive et scenario, hvori der indgår konkret objekter samt nærmere beskrevne kommunikationer mellem disse.