

# Multiparadigme Programmering

Repetition: De grundliggende paradigmer  
Systematisk paradigme karakteristik.  
Paradigmesupplering.  
Symmetrisk multiparadigme programmering  
Leda eksempler på logikprogrammering  
kombineret med imperativ og  
funktionsorienteret programmering

## PS3 - Multiparadigme programmering

© Kurt Nørmark, Aalborg Universitet

12/12/96 s. 1

## Noter

Denne lektion er - delvis - baseret på

*Multiparadigm Programming in Leda* af Timothy A. Budd fra Addison  
Wesley.

## Repetition: Grundliggende programmeringsparadigmer.

- Et *grundliggende programmeringsparadigme* udspringer af en *idé*, som er inspireret af en *basal disciplin*, der er relevant i forhold til 'det at foretage beregninger'.
- Hovedparadigmer
  - Det imperative programmerings-paradigme:
    - Inspirerende disciplin: Digital hardware teknologi.
  - Det funktionsorienterede programmerings-paradigme.
    - Inspirerende disciplin : Matematisk funktionsteori.
  - Det logiske programmerings-paradigme.
    - Inspirerende disciplin : "Automatisk", logisk bevisførelse.
  - Det objekt-orienterede programmerings-paradigme.
    - Inspirerende disciplin : Model for interaktion mellem mennesker og ting.
  - Andre grundliggende paradigmer
    - De parallelle programmerings-paradigmer.
    - Det visuelle programmeringsparadigme.
    - Det constraint-baserede programmeringsparadigme.

### PS3 - Multiparadigme programmering

## Noter

Denne slide er i store træk en repetition af en slide fra første lektion, hvor vi introducerede termene *grundliggende programmeringsparadigmer* og de fire *grundliggende hovedparadigmer*.

I denne forelæsning vil vi diskutere og vurdere de fire grundliggende hovedparadigmer i forhold til hinanden, og vi vil i særdeleshed se på mulighederne for at kombinere de grundliggende hovedparadigmer.

## Grundlaget for en systematisk paradigme karakteristik.

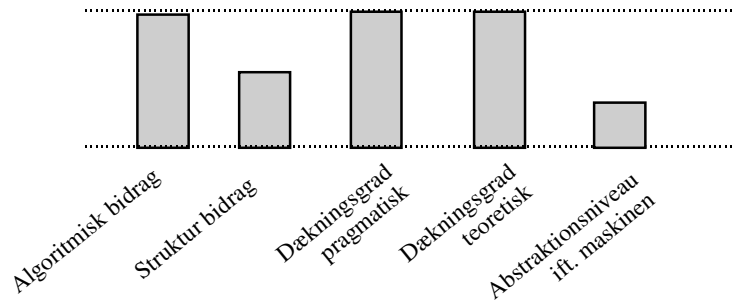
- **Algoritmisk bidrag**
  - Bidrager et paradigme med algoritmisk substans?
- **Struktur bidrag**
  - Bidrager et paradigme til strukturering af programmer?
- **Dækningsgrad**
  - Pragmatisk:
    - Hvor bred en klasse af problemer kan *på en naturlig måde* løses inden for paradigmet?
  - Teoretisk:
    - Hvor bred en klasse af problemer kan *teoretisk set* løses inden for paradigmet?
- **Abstraktionsniveau**
  - i forhold til den underliggende, fysiske maskine
  - Hvor tæt er paradigmet knyttet til idéen i den underliggende maskine?
  - Tæt tilknytning: lavt abstraktionsniveau.

### PS3 - Multiparadigme programmering

## Noter

## Karakteristik af det imperativ paradigme.

- Nøgleord:
  - Kommandoer med inkrementel tilstandsforandring som funktion af tiden.
  - Abstraktion med procedurer.
  - Effektivt, men sjældent “elegant”.



### PS3 - Multiparadigme programmering

© Kurt Nørmark, Aalborg Universitet

12/12/96 s. 4

## Noter

Det er oplagt at ovenstående karakteristik, lige som de efterfølgende, er ganske subjektive. Det betyder at der givetvis vil kunne opstå uenigheder om karakteristikken.

Det turde være klart, at dette paradigme bidrager med algorithmisk substans. Vi er vant til at udtrykke algoritmer “trin for trin”, og det er netop hvad det imperativ paradigme er egnet til.

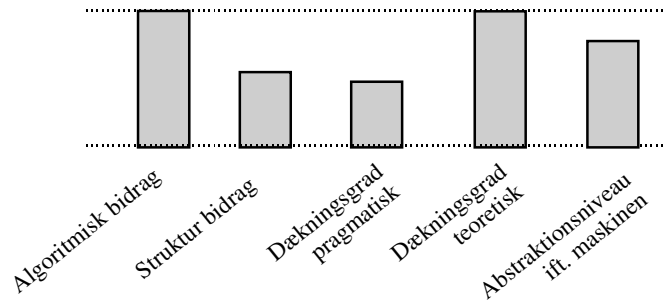
Strukturbidraget er middelmådigt. Vi kan dyrke abstraktion over kontrollerede kommandoer, hvilket giver procedurer. Der er naturligvis muligheder for at overlejlre endnu mere struktur på paradigmet (moduler mv.), men i bund og grund er det paradigmet uvedkommende i forhold til vores definition af imperativ programmering.

Den pragmatiske såvel som den teoretiske dækningsgrad i paradigmet er højt. Det betyder at så at sige alle problemer vil kunne løses på en rimelig naturlig måde i paradigmet.

Abstraktionsniveauet i paradigmet i forhold til maskinens niveau er relativt lavt. Dermed mener vi at der er en tæt kobling mellem den måde maskinen virker på og den måde som imperativ programmering virker på.

## Karakteristik af det funktionsorienterede paradigme.

- Nøgleord:
  - Udtryk der beregnes til ikke-muterbare værdier.
  - Værdier kan transformeres til andre værdier, som er uafhængige af de oprindelige værdier
  - Abstraktion med funktioner.
  - Matematisk elegant, men undertiden ineffektivt.



### PS3 - Multiparadigme programmering

## Noter

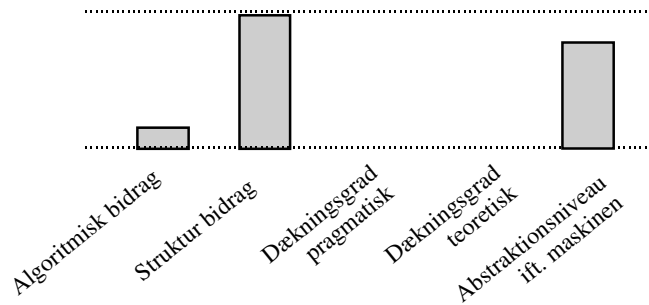
Algorithmisk bidrag og strukturbidrag er ganske som for imperativ programmering.

Den pragmatisk dækningsgrad er dog klart lavere af funktionsorienteret programmering end af imperativ programmering. Der er en mindre mængde af programmer der kan udtrykkes naturligt i funktionsorienteret programmering end i imperativ programmering. (Dette er formodentligt kontroversielt og problematisk i forhold til fortalere for funktionsorienteret programmering). Fra et teoretisk synspunkt kan man dog givetvis dække enhver problem's løsning i paradigmet.

Der er en betydelig afstand mellem funktionsbegrebet og den underliggende maskine. Vi siger altså, at abstraktionsniveauet af det funktionsorienterede paradigme er højere end abstraktionsniveauet af det imperative paradigme. Altså, med det funktionsorienterede paradigme har vi yderligere abstraheret i forhold til de imperative paradigme (i forhold til den basale, konventionelle maskine).

## Karakteristik af det objekt-orienterede paradigme.

- Nøgleord:
  - Indkapsling, information hiding, nedarvning
  - Abstraktion over datadefinitioner: dataabstraktion.



### PS3 - Multiparadigme programmering

© Kurt Nørmark, Aalborg Universitet

12/12/96 s. 6

## Noter

Udgangspunktet for ovenstående karakteristik er måske lidt usædvanligt: vi tager snævert udgangspunkt i en karakteristik af objekt-orienteret programmering som understøttende indkapsling, information hiding og nedarvning.

I mange definitioner af objekt-orienteret programmering tager man afsæt i imperativ programmering, og udvider der fra. Dermed besidder objekterne i objekt-orienteret programmering tilstand, som forandres inkrementelt via kommandoer.

I forbindelse med en diskussion af multi-paradigme programmering er det mere end fristende at fokusere på de absolutte karakteristika ved objekt-orienteret programmering uden a priori at binde disse til imperativ programmering.

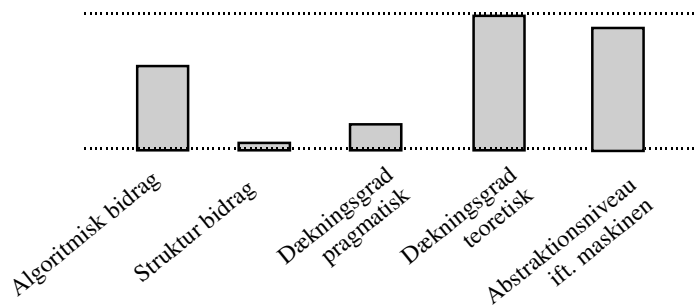
Man kan måske sige, at vi her ser på objekt-orienteret programmering som et paradigme mix-in snarere end et fuldt paradigme i sin egen ret.

Dækningsgraderne er ikke angivet ovenfor. Man kan måske også sige, at objekt-orienteret programmering uden det underliggende imperative paradigme, ikke dækker noget som helst.

Abstraktionsniveauet i forhold til maskinens er relativt højt. Objekter er (ifølge Budd) maskiner inden i maskinen: rekursive beregningsenheder. Alene denne forståelse berettiger til et løft i abstraktionsniveau i forhold til det imperative paradigme. Men også nøgleordene "indkapsling, information hiding og nedarvning" berettiger klart til et løft i forhold til imperativ programmering. (Jeg sammenligner ikke på dette punkt med funktionsorienteret programmering - hvad angår abstraktionsniveau er objekt-orienteret og funktionsorienteret programmering usammenlignelige størrelser).

## Karakteristik af det logiske paradigme.

- Nøgleord:
  - Facts, inferensregler, forespørgsler.
  - Deklarativ i udtryksmåden.
  - Matematisk elegant, men undertiden ineffektivt.



### PS3 - Multiparadigme programmering

© Kurt Nørmark, Aalborg Universitet

12/12/96 s. 7

## Noter

Det algoritmiske bidrag af logik programmering er tilstede, men det er ikke så udpræget som i funktionsorienteret programmering eller imperativ programmering.

I paradigmet som så er der så at sige ingen strukturbidrag.

Den pragmatiske dækningsgrad er lav, idet paradigmet kun gør det naturligt at løse en bestemt og begrænset problemkreds. (På disse egnede problemer virker paradigme så til gengælde godt. Og man kan ofte udtrykke sig ekstremt elegant). Den teoretiske dækningsgrad er sikkert lige så høj som for alle de andre paradigmer: Man kan teoretisk set løse ethvert problem i paradigmet. Fra en praktisk synsvinkel er dette dog ikke mere interessant end det faktum, at de fleste problemer kan løses på en Turing maskine.

Det logiske programmeringsparadigme repræsenterer nok det paradigme som distancerer sig mest fra selve maskinens virkemåde. Det udtrykker vi ved et meget højt abstraktionsniveau i forhold til maskinen.

## Paradigmesupplering.

Sup- Para- dige Udgangs- punkt	Imperative paradigme	Funktions- orienterede paradigme	Objekt- orienterede paradigme	Logiske paradigme
<i>Imperative paradigme</i>		Forekommer hyppigt, dog ofte med svagt funktionsbegreb	Imp. sprog med OO overbygning: C++	(Leda)
<i>Funktions- orienterede paradigme</i>	Forekommer hyppigt: ML, Scheme, ...		Meningsfuldt.. OOP suppleret i mange fnkt-or sprog	???
<i>Objekt- orienterede paradigme</i>	Konventionel OOP	Stærkere eller svagere fnkt begreb i OOP		???
<i>Logiske paradigme</i>	(Leda)	???	???	

### PS3 - Multiparadigme programmering

© Kurt Nørmark, Aalborg Universitet

12/12/96 s. 8

## Noter

Spørgsmålet på denne slide er hvorvidt et paradigme kan anvendes som *suppleringsparadigme* inden for et programmeringsparadigme, som vi har valgt som udgangspunkt for vores programmering.

## Vurdering af paradigmesupplering.

Sup- Para- dige Udgang- punkt	Imperative paradigme	Funktions- orienterede paradigme	Objekt- orienterede paradigme	Logiske paradigme
Imperative paradigme	X	Attraktivt	Sjældent vellykket: lig i lasten.	Udfordrende
Funktions- orienterede paradigme	Urent	X	Attraktivt	Udfordrende
Objekt- orienterede paradigme	De facto = OOP	Attraktivt	X	Udfordrende
Logiske paradigme	Urent	Udfordrende. Bedre end .....	Må være afprøvet!	X

### PS3 - Multiparadigme programmering

## Noter

## Leda og symmetrisk multiparadigme programmering

Anvendelse af to eller flere paradigmer på en *symmetrisk måde* i et program.

- Intet paradigme er a priori udgangspunktet.
- Intet paradigme spiller rollen som suppleringsparadigme.

Leda understøtter de fire grundliggende hovedparadigmer.

- Det klart mest interessante og innovative aspekt er hvordan det logiske paradigme spiller sammen med de tre andre.
  - Interessant for det logiske paradigme pga. dets lave pragmatiske dækningsgrad, og pga. det begrænsede algoritmiske bidrag i paradigmet.
  - Interessant for de andre paradigmer pga. de særdeles elegante løsninger som kan formuleres i det logiske paradigme.

### PS3 - Multiparadigme programmering

## Noter

## Datatypen Set i Leda (1).

```
class set [X: equality];
var
  value: X
  next: set[X];

function includes(val: X) -> Boolean;
begin ... end

function remove (val: X) -> Set[X];
begin ... end

function items(byref val: X) -> relation;
begin
  return unify[X](val, value)
    | defined(next) & next.items(val);
end;
end;
```

### PS3 - Multiparadigme programmering

© Kurt Nørmark, Aalborg Universitet

12/12/96 s. 11

## Noter

Ovenstående viser en skitse af en klasse fra Kapitel 3 i Leda bogen.

Vi ser definitionen af en rekursiv datatype Set.

Vi ser også anvendt funktionsorienteret programmering i remove: Det at fjerne et element val fra en mængde returnerer en ny mængde, hvor elementet var ikke er med.

Det interessante er imidlertid funktionen items, som involverer elementer fra det logiske paradigme. Vi diskuterer og beskriver denne på næste slide.

## Datatypen Set i Leda (2).

```
function items(byref val: X) -> relation;
begin
  return unify[X](val, value)
    | defined(next) & next.items(val);
end;
```

- `items(element)` returnerer (boolsk) hvorvidt *element* er element af mængden.
- `items(variable)` binder variabel til det første element i mængden.
  - Dette er bidraget af `unify` i kroppen af `items`.
  - Unify forsøger på at gøre sine to parametre ens ved at ændre på den første (som er variabelen overført “by reference” til `items`).
  - Variabelbinding fra Unify kan omgøres hvis næste element er defineret.
  - I en for kontrolstruktur “backtracks” gennem alle mulige bindinger af `val`.

```
if aset.items(var) then      for aset.items(var) then
{gør noget med den         {gør noget med den
  nu bundne variabel}       nu bundne variabel}
else ...
```

### PS3 - Multiparadigme programmering

## Noter

## Datatypen Graph i Leda (1).

```
Class Graph;
var vertexSet: Set[Integer];
    edgeSet: Set[Edge];

    function addVertex(newVal: integer) -> Graph; ...
    function addEdge(newStart, newEnd: Integer) -> Graph; ...
    function includesVertex(val: integer): -> Boolean; ...
    function includesEdge(t: integer;
        h: integer) -> Boolean;...

    ...

    function vertex(byRef val: integer) -> Relation;
    function edge(byRef tail: integer,
        byref head: integer) -> Relation; ...
    function path(byRef i: integer;
        byRef j: integer) -> Relation ...
    ...
    function inDegree(aVertex: integer) -> Integer;

    ...

end;
```

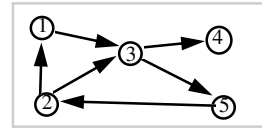
### PS3 - Multiparadigme programmering

## Noter

## Datatypen Graph i Leda (2).

```
function edge(byRef tail: integer,  
             byRef head: integer) -> relation;  
var anEdge: Edge;  
return  
  defined(edgeSet) &  
  edgeset.items(anEdge)  
    & unify[integer](tail, anEdge.tail)  
    & unify[integer](head, anEdge.head)  
end;
```

- `edge(1,3)` returnerer true idet der er en kant fra knude 1 til knude 3.
- Det er muligt at iterere over kanter som knuder hvortil der er en kant til en bestemt anden knude:



```
x := NIL;  
for aGraph.edge(x, 3) do  
  print(x);
```



Udskriver knude 1 og 2

### PS3 - Multiparadigme programmering

## Noter

### Datotypen Graph i Leda (3).

```
function inDegree(aVertex: integer) -> integer;  
var head: integer;  
    count: integer;  
begin  
    count := 0;  
    head := aVertex;  
    for edge(NIL,head) do count := count + 1;  
    return count;  
end;
```

### Noter

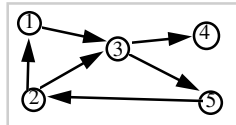
## Datotypen Graph i Leda (4).

```
function path(byRef i: integer,  
             byRef j: integer)  
             -> Relation;  
var k: integer;  
begin  
  return vertex(i) & vertex(j) &  
    ((i = j)  
     | edge(i, j)  
     | edge(i, k) & removeVertex(i).path(k, j));  
end;
```

```
i := NIL;  
for aGraph.path(i, 5) do begin  
  print("vertex"); print (i)  
  print("can reach vertex 5\n");  
end;
```



vertex 1 can reach vertex 5  
vertex 2 can reach vertex 5  
vertex 3 can reach vertex 5  
vertex 5 can reach vertex 5



### PS3 - Multiparadigme programmering

## Noter

