

# Kapitel 1:

## De datalogiske fags identitet(er) i videnskabernes landskab

Henrik Kragh Sørensen

Mikkel Willum Johansen

22. april 2022

### Indhold

5

1.1	Datalogiens sjæl . . . . .	1
1.2	Videnskabens natur . . . . .	4
1.3	Real- og formalvidenskab, induktion og deduktion . . . . .	5
1.4	Fag og genstandsområder . . . . .	8
1.5	Myten om den videnskabelige metode . . . . .	11
1.6	Den autonome videnskabs nytte . . . . .	15
1.7	Fra akademisk til post-akademisk videnskab? . . . . .	18

10

### Litteratur

20

### 1.1 Datalogiens sjæl

Selvom man har studeret datalogi i flere år, kan det være på sin plads men ikke helt let præcist at afgrænse, hvad der karakteriserer faget og adskiller det fra andre områder af den menneskelige viden. Der er nogle klassiske tilgangsvinkler til et sådant forsøg på at karakterisere feltet: Man kan forsøge at definere datalogien ud fra dens centrale egenskaber (dens natur) og de *objekter*, den behandler. Dette vil typisk være en *filosofisk* tilgangsvinkel, og den kan være både *deskriptiv* (beskrivende) og *normativ* (foreskrivende). Som et delvist alternativ kan man forsøge at indkredse datalogi ved at beskrive dens udøvelse; dvs. de processer, igennem hvilke datalogisk viden skabes, formidles og anvendes. Denne tilgangsvinkel involverer et fokus på relationer mellem mennesker, og den er således i sidste ende *sociologisk*. Endelig kan man forsøge at konkretisere indholdet i datalogi ved at gå til det med en *historisk* tilgangsvinkel, som fokuserer på, hvordan disciplinen har udviklet sig og har udsondret sig selv fra andre felter.

15

Når man skal afgrænse informationsteknologi og datalogi som felter er det også relevant at overveje, om fagene ligger nærmest ved den akademiske videnskab eller mere er at sammenligne med teknologiske vidensområder. Men for at diskutere dette er det først nødvendigt at nå en foreløbig afgrænsning af selve feltet datalogi og så siden analysere, hvad der karakteriserer videnskab og teknologi for så til sidst at kunne forsøge at positionere datalogien på det spektrum, som udspændes af akademisk og teknologisk videnskab.

20

De fleste dateringsspørgsmål er for videnskabsteoretikeren især interessante for så vidt, de kan bruges til at finde ind til de kriterier, som forskellige aktører anlægger for at definere

25

35 det daterede fænomen eller objekt. Hvis vi spørger, hvornår computeren blev opfundet, så kan vi få forskellige svar (dateringer) ud fra, hvad forskellige forskere opfatter som de centrale karakteristika ved computere (se diskussion om *computational artefacts* i Dasgupta, 2016). Tilsvarende kan vi få ganske forskellige dateringer ud fra forskellige indkredsninger af den videnskab, som vi i dag og i Danmark kalder for *datalogi*.

40 Der er dog næppe tvivl om, at der er nogle centrale begivenheder, som er med til at definere datalogiens historie som videnskab og akademisk disciplin. Institutionelt blev datalogi i alle de forskellige instanser indført på universiteterne i perioden efter Anden Verdenskrig. I Danmark skete det i første omgang i to forskellige, men forbundne, kontekster: Regnecentralen og universiteternes matematiske institutter (se Sørensen, 2006). Og på den 45 internationale scene var det især begivenheder i 1960’erne, der for alvor tegnede nogle af fagets grænser i resten af århundredet. Igennem en af afsøgning af fagets relationer og muligheder kom diskussioner om forholdet til matematik og den såkaldte *softwarekrise* til at sætte dagsordenen for en positionskamp (både indadtil og udadtil) mellem formel grundvidenskab, empirisk videnskab og ingeniørvidenskab.

50 Diskussionerne handlede især om, hvordan man skulle uddanne de flere og flere mennesker, der var nødvendige for at udnytte det stigende antal computere i industrien, forskningen og militæret: Skulle de være videnskabeligt og akademisk uddannede med en bred teoretisk viden, eller var det bedre, at de var certificerede til bestemte praktiske systemer? Problemet var blevet akut i slutningen af 1960’erne, hvor NATO sponsorerede en konference, der skulle udstikke retningslinjer for at afhjælpe den såkaldte *softwarekrise*. Og selvom konferencen mere viste forskellige veje end fælles fodslag, kom der både begrebet *software engineering* og et meget indflydelsesrigt forslag til et fælles curriculum for akademisk datalogi ud af mødet (Atchison m.fl., 1968).

55 Men en anden del af diskussionerne om datalogiens identitet i slutningen af 1960’erne handlede om navne. For skulle den nye videnskabs navn fokusere på *computeren*, på *information*, eller på *data*? Og hvad skulle videnskaben overhovedet handle om? Vi har stadig resterne af denne diskussion med os, for den akademiske disciplin, der i Danmark hedder *datalogi*, hedder i den engelsktalende verden typisk *computer science*, mens den fx i Tyskland, Holland og Frankrig typisk involverer begrebet *informatik*.

60 Selv internt i Danmark kan man godt spore forskellen af, hvordan datalogi skal forankres og bedrives. Selve *computeren* som artefakt ankom til Danmark efter Anden Verdenskrig, og den første danskproducerede computer, *DASK* var operationel i 1957. Regnecentralen, som havde stået for at bygge DASK, fortsatte med at bygge en ny, transistorbaseret computer *GIER*, som var klar i 1961. Regnecentralen var et statsligt forretagende, der også havde til formål at sikre dansk teknologisk kapacitetsopbygning, dvs. 65 det var vigtigt ikke blot at importere fungerende computere men også at opbygge kapacitet til selv at udvikle, bygge og betjene teknologien. Selve navnet GIER siger noget om de vigtigste aftagere: Det står for *Geodætisk Instituts Elektroniske Regnemaskine*, for geodæterne havde brug for omfattende regnekraft til trianguleringer og andre beregninger. Dette viser, 70 hvordan udviklingen af computere i Danmark hang nært sammen med deres akademiske aftagere, og det er ikke specielt for Danmark.

I 1966 nedsatte Københavns Universitet et fælles edb-udvalg, som anbefalede oprettelsen af et databehandlingscenter på KU, etableringen af lokale mindre datamaskiner ved enkelte institutter med forbindelse til det fælles datacenter, og etablering af undervisning i datalogi 75 og en egentlig datalogi-linje. To år senere besluttede man sig for at oprette et professor i datalogi, og allerede ved studiestart 1968 var datalogilinjen oprettet. I 1969 tiltrådte PETER NAUR (1928–2016) som den første professor ved Datalogisk Institut på Københavns

Universitet (stadig forkortet DIKU). DIKU var fra sin oprettelse altså baseret på både uddannelse af dataloger og nært samarbejde med andre faglige miljøer på Københavns Universitet.

85

I Aarhus tog udviklingen af et datalogisk miljø fart i samme periode. Et af Regnecentralens tre decentrale regnecentre (ReCAU) blev fra midten af 1960’erne det lokale centrum, og SVEND BUNDGAARD (1912–1984), som var professor i matematik og tidligere rektor, satsede stort på, at også Aarhus Universitet skulle udbyde en datalogiuddannelse. Derfor sendte han dygtige matematikstuderende med datalogisk interesse til USA for at de kunne få deres ph.d.-grad og vende tilbage til Aarhus. På denne måde blev DAIMI (Datalogisk Afdeling I Matematisk Institut) opbygget til hurtigt at blive landets største datalogiske miljø, især inden for teoretisk datalogi, hvor den nære forbindelse til Matematisk Institut satte sit præg. I 1998 blev DAIMI et selvstændigt institut, og i dag har det skiftet navn til Institut for Datalogi.

90

Denne komplekse tilblivelseshistorie — med dens mange forviklinger og lokale afhængigheder — dækker over en iboende egenskab ved datalogi: Det er på en og samme tid et afgrænset vidensdomæne og et fagområde, der rækker ind i og interagerer med rigtig mange andre former for viden. Så det er en svær opgave at indfange datalogiens identitet i en kort definition. Alligevel kan man jo godt prøve.

95

I den hierarkiske organisering af menneskelig viden, som er blevet forsøgt indenfor filosofien, har man typisk karakteriseret de forskellige vidensområder ud fra deres *genstandsfelt* (deres objekter, *ontologi*) og deres *erkendelsesmåde* (*epistemologi*). Et oplagt sted at starte en diskussion af datalogiens *ontologi* og *epistemologi* kunne være ved at slå op i et leksikon som fx *Den Store Danske Encyklopædi*. Først får man en beskrivelse af begrebets historiske ophav og etymologi:

100

Datalogi, (af *data* og *-logi*), videnskaben om data og dataprocesser; opstod efter fremkomsten af den digitale computer i midten af 1940’erne. Ordet datalogi blev indført af professor Peter Naur i 1966 som et alternativ til den amerikanske betegnelse „computer science“, der ikke i samme grad udtrykker, at faget rummer videnskabelige problemer, der rækker ud over den blotte udnyttelse af computeren. (Hansen, 2000, s. 25)

105

Dernæst beskrives datalogiens genstandsfelt og metode og relationen til computeren som redskab:

Datalogiens **forskningsobjekt** er **data**, repræsentationer af **objekter** eller **idéer**.  
[...] Datalogien handler således om datastrukturer og dataprocesser samt om karakterisering af sådanne strukturer og processers egenskaber og begrænsninger. Computeren er et centralt redskab i datalogien. Med den kan den datalogiske forsknings hypoteser afprøves i praksis, og megen datalogisk forskning omhandler principper for programmering og metoder til effektiv udnyttelse af computeren. Den vigtigste hjælpdisciplin for datalogien er matematikken; ikke den kontinuerte matematik [...], men diskret matematik. Også logik finder anvendelse i datalogien. (Hansen, 2000, s. 25)

110

120

Endelig placeres datalogien på det figurative landkort over videnskabelige discipliner, som den har mange, tværgående grænseflader med:

125

Datalogiens opgave er at afdække generelle principper for databehandling. Herved kommer datalogien til at interessere sig for mange problemstillinger, der

også kendes i andre discipliner [fx ingeniørfag (automatiske systemer), arkitektur og psykologi (brugsværdi af komplekse systemer), lingvistik (programmeringssprog), bibliotekarvidenskab (lagring og søgning), organisationsteori (store datasystemer)]. Datalogi er således en vidtfavnende tvaervidenskabelig disciplin [...]. (Hansen, 2000, s. 25)

Opslaget i *Den Store Danske Encyklopædi* angiver altså datalogiens objekter til at være *data* i form af *datastrukturer* og *dataprocesser*, det beskriver computeren som et *redskab* og hævder, at datalogien undersøger *hypoteser* og trækker på *diskret matematik* som sin vigtigste hjælpedisciplin. Og ikke mindst fremhæver opslaget datalogiens *tvaervidenskabelige* natur, idet faget trækker på og bidrager til mange andre vidensfelter.

## 1.2 Videnskabens natur

Selvom det kan synes uoverstigligt udfordrende, kan man godt forsøge at karakterisere videnskab og opstille kriterier, der kan afgøre om givne påstande er videnskabelige eller ej. Ligesom i så mange andre situationer i videnskabsteorien kan man gå til værks på forskellig vis. Man kan søge at beskrive 1. karakteristika ved *videnskabelig viden*, 2. normer for et *videnskabeligt ethos*, 3. beskrivelse af *videnskabelig praksis*, eller 4. idealer for *demokratisk videnskab*. Disse fire former vil blive kort behandlet i det følgende. For at komme nærmere en afgrænsning af begrebet videnskab, kunne man igen ty til *Den Store Danske Encyklopædi*, hvor man så ville få at vide, at videnskab er en „almen betegnelse for systematiske metoder til at frembringe, ordne og udbrede viden og kunnen samt resultaterne af denne aktivitet og de organisationsformer og administrative enheder (som fag og discipliner), hvorunder den foregår“ (Kragh, 2001, s. 139). Denne betegnelse er nødvendigvis meget bred, men man kan komme lidt tættere på ved at forsøge at afgrænse, hvad der udgør *videnskabelig viden*. Man kan spørge, hvad der er specielt ved videnskab og adskiller *videnskabelig viden* fra andre slags viden. Nogle blandt de mange forslag til en sådan definition er listet her baseret på (Kragh, 2003, s. 148–150):

1. Videnskab er en forfinelse af dagliglivets erfaringer.
2. Videnskab er teoretisk, i modsætning til praktisk viden.
3. Videnskab resulterer i udsagn, der er matematisk formulerede og hævdes at være universelt gyldige.
4. Videnskab er givet ved bestemte metoder og procedurer, fx eksperimenter.
5. Videnskab giver objektiv viden via sociale mekanismer til sikring af dens offentlige karakter.
6. Videnskab resulterer i offentlig tilgængelig viden i form af bøger og artikler.
7. Videnskab er, hvad videnskabsmænd laver.

Hver af disse definitioner indfanger væsentlige aspekter af videnskabens natur — men ingen af dem tåler at blive ophævet til almennyldige karakteriseringer af, hvad videnskab er. Nogle af dem er *tilstrækkelige* men ikke *nødvendige* betingelser, og for andre iblandt dem

forholder det sig omvendt. Nogle af dem udspringer — ligesom en stor del af videnskabsteorien — fra særlige dele af de fysiske videnskaber, og mange af dem passer overhovedet ikke på fx humaniora, samfundsvidenskab eller matematik.

I erkendelse af disse definitioners begrænsninger tilbyder videnskabshistorikeren og videnskabsteoretikeren HELGE KRAGH en definition, som fokuserer på fire egenskaber ved den viden, som videnskaben producerer:

Videnskab er en intellektuel og social proces, der stræber mod, og rent faktisk resulterer i, en form for viden, som er karakteriseret ved at være (i) *offentlig*, (ii) *fejlbart*, (iii) *korrigerbart* og (iv) *testbar*. (Kragh, 2003, s. 150)

At den videnskabelige viden skal være *offentlig* skal forstås sådan, at den videnskabelige viden ikke er af privat karakter, men skal kunne forstås og vurderes af kvalificerede medlemmer af det videnskabelige samfund. Disse kolleger skal så på grundlag af den fremstillede viden kunne nå til de samme konklusioner, hvorved der opstår *konsensus* om den videnskabelige viden. Konsensus er således et sociologisk fænomen, som ikke i sig selv implicerer hverken objektivitet eller sandhed.

Når Kragh insisterer på, at videnskabelig viden skal være *fejlbart*, så er det vigtigt at understrege, at dermed menes *ikke*, at videnskabelig viden er fejlagtig, kun at den (altsid) vil og skal være *potentielt forkert*. Videnskabelig viden er ikke på forhånd garanteret sandhed, og den videnskabelige viden vil ofte være ufuldstændig og forbundet med usikkerheder. Videnskabelig viden må altså ifølge denne definition ikke være dogmatisk. Som det skal diskuteres i et senere afsnit (1.3) møder dette punkt nogle særlige udfordringer for matematik og — dermed — dele af datalogien.

Det videnskabelige samfund skal have mulighed for — via institutionelle strukturer — at efterprøve og kritisere den frembragte viden. Dette er indholdet af Kraghs bemærkninger om, at videnskabelig viden kan *korrigeres* og *efterprøves*. Det skal være muligt *systematisk og kritisk* at afprøve videnskabelige påstande, hvilket kun er meningsfyldt, hvis påstandene kan være fejlagtige. Derfor hænger de sidste tre dele af Kraghs definition nært sammen.

### 1.3 Real- og formalvidenskab, induktion og deduktion

Når man ønsker at karakterisere videnskab og videnskabelig viden, kan man altså stille en række forskellige spørgsmål, som hverisær bidrager til besvarelsen:

1. Hvad er (karakteriserer) videnskabelig viden?
2. Hvordan opnår man videnskabelig viden?
3. Hvordan udvikler videnskabelig viden sig?

Men mere grundliggende kan man også spørge om, hvilket forhold vores viden har til den 'virkelighed', den er viden om. Her er virkeligheden sat i anførselstegn, for i sidste ende handler dette spørgsmål *også* om, hvad vi betragter som videnskabens *genstandsfelt*. I astronomi vil genstandsfeltet nok være himmellegemer, der eksisterer uafhængigt af vores viden om dem, og hvis bevægelser vi heller ikke kan påvirke med vores viden. Anderledes forholder det sig om fx socialvidenskaberne, hvor man fx studerer økonomiske udviklinger, som man selv i kraft af at være borger er del af, og hvorpå ens viden kan have indflydelse. I store områder af humanvidenskaberne kan man hævde, at viden er meget mere fortolkende

170

175

180

185

190

195

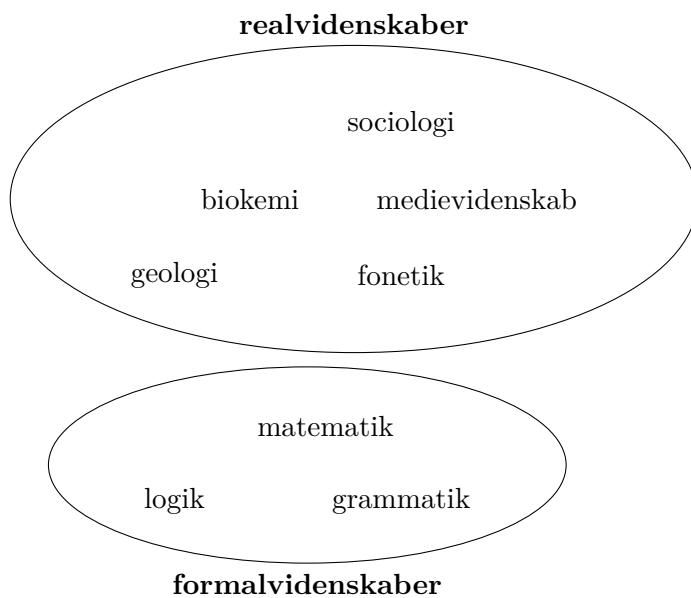
200

205

på en måde, der selvom den er underlagt metodiske normer, er afhængig af det fortolkende subjekt. I den yderste ende af spektret ligger måske matematik, hvis man opfatter det fag som studiet af konsekvenserne af arbitraert postulerede antagelser, for så er der ikke rigtig 210 noget genstandsfelt, som eksisterer uafhængigt af os, og samtidig kunne det se ud til, at alt, hvad vi kan vide, er bygget ind i antagelserne.

Denne korte opremsning antyder, at der er meget mere at sige om relationen mellem genstandsfeltet og vores viden derom: For det første kan man spørge til, hvordan man kan karakterisere vores genstandsfelt og vores adgang til at have viden om det. Og for det andet 215 kan man spørge, hvorvidt genstandsfeltet og vores viden derom kan siges at være *objektiv*.

En af måderne at få hold på genstandsfeltet kan være at skelne mellem, om det er et reelt eksisterende udsnit af en virkelighed uafhængig af vores undersøgelse, eller det er et formelt introduceret genstandsfelt, som vi har introduceret for at studere det. I den første 220 situation taler man tit om **realvidenskaber**, dvs. videnskaber som fysik, historie, økonomi, medicin etc., som handler om fænomener i en virkelighed, der foreligger uafhængigt af vores viden om den (se figur 1). Bemærk, at der ikke er tale om, at virkeligheden eksisterer strengt uafhængigt af os — den historiske virkelighed, som er defineret ved overleverede levn, ville nok ikke kunne siges at være strengt uafhængig af menneskers eksistens, og medicin ville da i hvert fald ikke. Men de er begge realvidenskaber, fordi videnskaben i disse felter 225 studerer fænomener, der er eksterne for videnskaben selv. I realvidenskaberne vil vi typisk opbygge viden ved at indsamle og systematisere informationer (data), fremsætte hypoteser om, hvordan data hænger sammen, og forsøge at opbygge evidens for vores påstande ved fx at udføre eksperimenter. Vores vidensproduktion er altså **erfaringsbaseret**, hvorfor vi siger om realvidenskab, at det er **aposteriori viden**, dvs. viden, som er afhængig af erfaring. Og 230 den er i høj grad *induktiv*, idet vi slutter fra partikulære tilfælde til generelle lovmaessigheder, og selvom vi kan stræbe efter at blive mere præcise og dækkende, kan vi af princip *aldrig* opnå absolut sikkerhed: Vores viden vil altid være foreløbig, sådan som KRAGH også understregede ovenfor.



Figur 1: Videnskaber inddelt efter det skellet mellem real- og formalvidenskaber.

Inden for andre felter, som fx matematik, kan man sige, at det genstandsfelt, vi studerer, er et, vi selv har skabt og kender fuldstændigt ud fra de (eksplicitte) antagelser, vi har opstillet om det. En sådan videnskab kan man kalde for en *formalvidenskab*, og de er karakteriseret ved, at vi i en formalvidenskab studerer et genstandsfelt, som vi har fuldstændigt og formelt kendskab til (se igen figur 1). Derved er ikke sagt, at vi på forhånd kender alle konsekvenserne af vores antagelser, men i og med at vi arbejder *deduktivt*, dvs. slutter logisk fra det generelle til det partikulære, er vores opdagelser højst overraskende som et psykologisk fænomen. I matematikken, som er indbegrebet af formalvidenskab, kan vi *bevise* påstande med *absolut sikkerhed*, dvs. vi kan — fordi vi selv har opsat spillereglerne — opnår absolut nødvendige eller tilstrækkelige betingelser, udelukkende ved logisk ræsonneren. Og fordi denne ræsonneren kan foregå inde i matematikerens hoved, siger vi, at den er *a priori*, dvs. *kommer forud for erfaringen*. Hvorvidt al matematisk viden er indeholdt i aksiomerne (antagelserne) og matematikken derfor er *tautologisk* er en stor filosofisk og matematisk diskussion (se Johansen og Sørensen, 2014), som i begyndelsen af 1900-tallet fik en direkte forbindelse til det, vi i dag kalder datalogi (se kap. 3). Men foreløbigt kan vi notere, at *hvis* matematikken (og teoretisk datalogi) virkelig var tautologiske, ville de ikke opfylde kravet om fejlbarlighed i Kraghs karakterisering ovenfor.

235

240

245

250

For de fleste naturvidenskaber findes der en rimelig klar afgrænsning mellem den studerede virkelighed og den viden, vi har om den, og som vi rettelig kan opfatte som en teori eller en model. Naturvidenskaber som fysik og biologi er baseret på, at de studerede processer er uafhængige af, at vi studerer dem. Og selv når vi udmærket ved, at et eksperiment er en kontrolleret og idealiseret manipulation af naturen, så gør vi os umage for, at eksperimentet kan genskabes og valideres af andre (således at det kan siges at være *inter-subjektivt*).

255

På tilsvarende vis kan man identificere dele af datalogien, som kan siges at handle om udsnit af virkeligheder, der eksisterer uanset, om vi studerer dem eller ej. Det kan være algoritmer til at løse abstrakte grafteoretiske problemer, eller det kan være simuleringer af proteiners foldninger. Både graferne og proteinerne kan siges at eksistere uanset, om vi datalogisk studerer dem eller ej (graferne er jo et matematisk objekt, så de kan siges at eksistere i en formel virkelighed).

260

Men når vi laver datalogiske undersøgelser og bygger datalogiske systemer til brug i samfundet, så er vi ikke længere i denne situation. Som vi skal se nærmere på i kapitel 2, så vil sådanne systemer typisk *påvirke* den virkelighed, de er sat til at modellere. På den måde flyder grænserne mellem model og virkelighed lidt væk, og vi skal være klar over, at vores modeller ikke længere har rent *beskrivende* karakter, men også er blevet *regulerende* dele af virkeligheden.

265

deskriptiv	beskrivende
normativ	foreskrivende, præskriptiv
a priori	viden, der er uafhængig af (går forud for) erfaringen
aposteriori	viden, der er afhængig af erfaringen
induktion	slutning fra det specifikke (partikulære) til det generelle
deduktion	slutning fra det generelle til det specifikke
realvidenskab	videnskab, der handler om en ekstern virkelighed, som kan være fysisk, social, ...
270 formalvidenskab	videnskab, der handler om formelle systemer postuleret af os
objektiv	gældende uafhængig af hvem, der hævder det
subjektiv	gældende betinget af visse elementer af min bevidsthed
inter-subjektiv	kommunikable mellem subjekter
epistemologi	læren om, hvordan vi opnår erkendelse og hvilken form, den erkendelse har
ontologi	læren om 'det værende', bl.a. læren om genstandsfeltet for vores erkendelse

Figur 2: Centrale videnskabsteoretiske begrebspar (modsætninger).

## 1.4 Fag og genstandsområder

Filosoffen HANS FINK har beskrevet det moderne universitet ved hjælp af fem principper, der udgør selve kernen i universitetsbegrebet (Fink, 2003a, s. 11–12). Mange af disse idealer går tilbage til romantiske ideer formuleret af WILHELM VON HUMBOLDT (1767–1835) for 200 år siden (Humboldt, 2007):

- Der består en *tæt forbindelse mellem forskning og uddannelse*, hvilket konkret har været implementeret ved, at det er forskere, der underviser, og at undervisningen følger nogle af de samme processer, som forskningen gør.
- Forskerne har metodologisk og emnemæssig *forskningsfrihed* i den forstand, at forskningen ikke bør være begrænset eller styret.
- Tilsvarende har underviserne *undervisningsfrihed* i forhold til valg af indhold og undervisningsmetode.
- Som organisation har universitetet *selvstyre* og fremstår som en autonom del af samfundet. Det har tidligere betydet, at studerende var borgere ved universitetet frem for i den by, hvor universitetet lå.
- Universitetet er netop 'universelt' både i den forstand, at det er internationalt, men især at det ligger i universitets ideal at fremme *videnskabens enhed*.

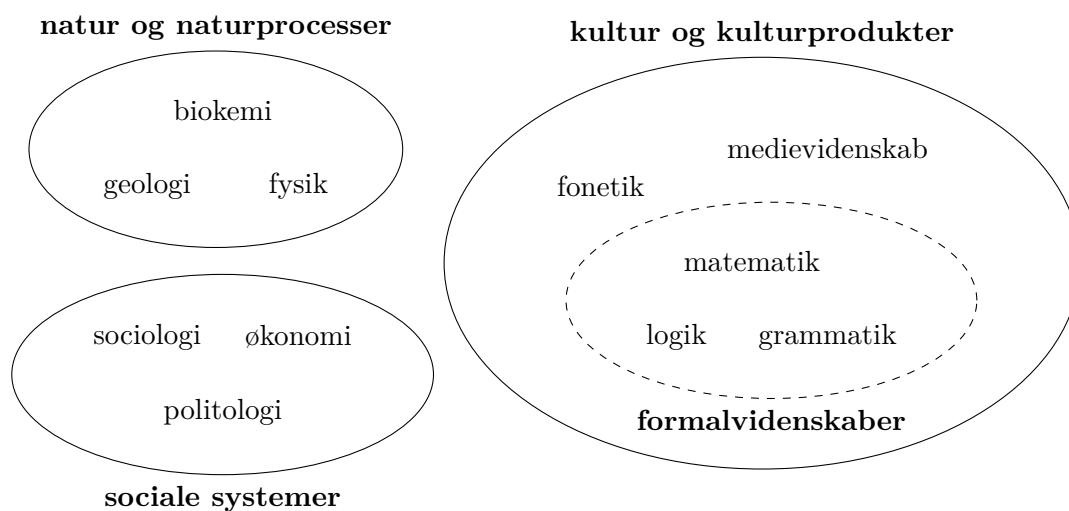
Disse idealer er ikke nemme—og måske endda ikke mulige—at kombinere i praksis. Og i forskellige konkrete situationer er de blevet vægtet og implementeret forskelligt. Og siden

FINK beskrev principperne i 2003 har flere af dem været oppe til diskussion, og nogle af dem har fundet nye former; det skal vi se yderligere på i kommende afsnit. 290

Men inden vi kommer til at diskutere det, er der et element af videnskabernes organiserings ved et universitet. Som FINK så fint beskriver det, er *fag* noget, der står ’ved siden af hinanden’ men er ’ forbundet’ — som fagene i et bindingsværkshus (Fink, 2003b, s. 194–195). Og på universiteterne er fagene typisk organiseret i *fakulteter* for fx naturvidenskab, medicin, jura, humaniora, samfundsvidenskab eller teologi. Nogle universiteter har fakulteter, der dækker alle disse traditionelle områder, og nogle har endda også fx teknologiske fakulteter, mens andre er mindre og mere specialiserede. 295

Der er gjort mange forsøg på at beskrive fagenes metoder ud fra de fakultater, de traditionelt hører under, således at man har forsøgt at indkredse en ’naturvidenskabelig’ 300 metode, som skulle karakterisere fx fysik og biologi. Men dels er der meget store metodeforskelle internt i fakulteterne, dels er der fag, der ikke let kan indplaceres i en enkelt metode. Så derfor er de fleste karakteriseringer af metodologiske og videnskabsteoretiske forskelle i videnskaberne oftest ikke godt tjent med at blive ført på niveau af fakulteter.

Omvendt er der jo også forskelle mellem de metoder, fagene på et universitet benytter. 305 Men i stedet for at føre en for generel diskussion om metoder, er det mere frugtbart først at skelne mellem de forskellige *genstandsfelter*, som forskellige fag og discipliner beskæftiger sig med. Dertil giver det overordnet mening at skelne mellem videnskaber, der handler om 1. *natur* og *naturprocesser*, dvs. fænomener som eksisterer helt uafhængigt af mennesker, 310 2. *kultur* og *kulturprodukter*, dvs. artefakter som mennesker har frembragt, og 3. *sociale systemer*, dvs. systemer og emergente fænomener som opstår, når (mange) mennesker organiserer sig (se figur 3). I følge den definition af formalvidenskab, som vi har diskuteret ovenfor, vil det være naturligt at opfatte fx matematik og grammatik som kulturprodukter, og denne inddeling efter genstandsfelt løber på mange måder på tværs af opdelingen i real- og formalvidenskab. Hvor skellet mellem real- og formalvidenskaber var godt til at analysere principielle forskelle sikkerheden af den form for viden, de hverisær producerer, så kan dette skel efter gentandsfelter sætte mere fokus på de forskellige metoder, der kan bringes i anvendelse. 315



Figur 3: Videnskaber inddelt efter genstandsfelt.

320 De datalogiske fag er ikke helt lette at indplacere entydigt efter deres genstandsfelt: selvfolgelig er computeren et kulturprodukt, og information er helt generelt betinget af sociale systemer, og datalogi bliver ofte brugt i samspil med andre videnskaber til at udforske naturen. Men nu kan vi benytte denne opdeling efter genstandsfelt til at analysere forskelle  
325 i de metoder, vi benytter inden for de datalogiske fag. Vi har ofte brug for matematiske beviser eller noget, der minder om naturvidenskabelige eksperimenter, men vi har også brug for at kunne få adgang til viden om fx brugeradfærd, som i høj grad er betinget af sociologiske metoder som interviews eller spørgeskemaundersøgelser.

Hele det felt, der undersøger interaktioner mellem mennesker og computere (*HCI* for *human-computer-interaction*) benytter sig i høj grad af sociologiske og antropologiske metoder. Hvis vi fx er interesseret i at udvikle software til bestemte arbejdsgange i en virksomhed, vil det være naturligt at undersøge disse arbejdsgange igennem observationer eller interviews med centrale medarbejdere. Men vi kan også forsøge at anstille forskellige eksperimenter, som kan give os kvantitativ viden om fx brugergrænseflader ved at sætte sensorer på nogle forsøgspersoner og måle deres øjenbevægelser eller reaktionstider.

335 Inden for sociologien og tilgrænsende videnskaber skelner man ofte mellem 1. *kvalitative metoder* som interviews og observationer og 2. *kvantitative metoder* som fx spørgeskemaundersøgelser. Dette skel peger både på et metodologisk valg og en tilhørende videnskabsteoretisk forskel.

340 *Kvantitative metoder* er, som navnet antyder, karakteriseret ved, at vi tæller og sætter tal på vores undersøgelsesobjekter med henblik på at kunne underkaste vores data en systematisk, statistisk analyse. Den slags metoder er afhængige af, at man kan få en tilstrækkelig stor og repræsentativ dækning af det felt, man ønsker at undersøge, for at de statistiske analyser kan give pålidelig viden. Til gengæld er undersøgelserne tit forholdsvis nemme at analysere (statistisk), og de kan forholdsvis nemt gentages under lignende forhold, selvom  
345 der er dybe epistemologiske udfordringer forbundet med denne påstand. For eksempel kan spørgeskemaundersøgelser jo påvirke de respondenter, man har spurgt, således at det ikke giver mening at spørge dem eller deres bekendte igen (man taler om, at man kan forurene en population). En endnu mere fundamental videnskabsteoretisk udfordring ved *kvantitative metoder* er, at man ikke som forsker har nogen indbygget mulighed for at vurdere,  
350 hvorvidt ens forventningshorisont, som jo har været brugt til at formulere spørgsmålene, er nogenlunde passende i forhold til respondenternes forståelser. Derfor kan man risikere at få svar på andre forståelser af spørgsmålene end dem, man havde til hensigt.

For at få adgang til de verdensbilleder, som personer i undersøgelsesfeltet har, er *kvalitative metoder* typisk en bedre vej at gå. Man kan fx afholde strukturerede interviews, observere praksis eller benytte fokusgrupper, og for alle disse metoder gælder, at data ikke  
355 er kvantitative, men har form af fx videooptagelser, feltnoter eller transkriberede interviews. Derfor er analysen af disse data også anderledes og ganske tidskrævende. Men den omfatter også videnskabsteoretiske udfordringer, som igen blandt andet handler om mødet mellem forskerens og subjektets verdenssyn: Enhver kvalitativ undersøgelse involverer også forskerens egne forståelser, og det er vigtigt at være sig dem bevidst, så man kan skelne mellem ens egen og subjektets forståelse og følge op på områder, hvor der kan være nuanceforskelle eller direkte modsætninger. Hvor *kvantitative metoder* er rettet mod store populationer, der kan opnå repræsentativitet igennem størrelse og diversitet, er *kvalitative metoder* betinget af små populationer, og de skal derfor oftest bruges enten eksplorativt eller eksemplarisk.  
360 Denne sidste observation bringer os til at se, at sociologiske metoder både er forskellige fra traditionelle metoder inden for fx naturvidenskab, men også at de nogle gange stræber efter at opnå de samme værdier som generalitet og reproducerbarhed. Men måske er det forfejet

at stile efter sådanne normer og mere passende at stræbe efter validitet i form af opfyldelse af KRAGHS fire kriterier for videnskabelig viden.

## 1.5 Myten om den videnskabelige metode

370

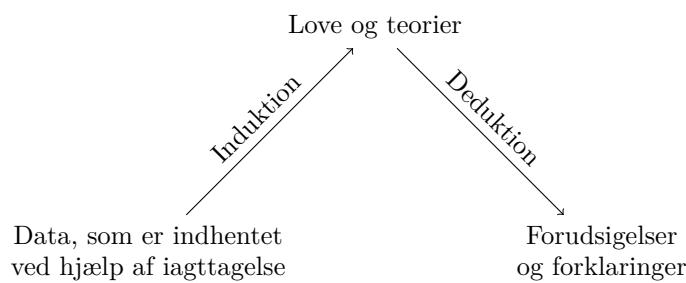
I løbet af 1900-tallet blev der gjort forskellige forsøg på at beskrive *den videnskabelige metode* og benytte denne metodoliske afgrænsning til at karakterisere videnskabelig viden. Denne tilgang er idag blevet ret udskældt, ikke mindst fordi den tog udgangspunkt i et ret snævert afgrænset syn på videnskabelig metode, som i bedste fald fandt anvendelse inden for visse dele af de fysiske naturvidenskaber. Men alligevel indeholder denne form for tilgang 375 nogle vigtige ideer omkring, hvordan vi kan nå til videnskabelig erkendelse, og hvor denne erkendelses begrænsninger ligger.

375

En af de mest tillokkende videnskabsteoretiske positioner i begyndelsen af 1900-tallet var den såkaldte *logiske positivisme*, der især blev formuleret af en gruppe filosoffer og videnskabsmænd med tilhørssforhold til Wien omkring 1920—deraf er gruppen kendt som *Wienerkredsen*, og en af dens fremmeste fortalere var filosoffen RUDOLF CARNAP (1891–1970). Ligesom så mange andre filosofiske overvejelser over videnkaberne siden 1600-tallet, 380 var udgangspunktet for de logiske positivister at forklare, hvordan mennesker kan opnå generel viden (teorier) ud fra partikulære observationer. Vi har jo kun set solen stå op et endeligt antal gange, men vi ønsker alligevel at kunne formulere og forsvare som en lovmaessighed, at solen også står op i morgen tidlig. Deres løsning var at insistere på, at de eneste udsagn om verden, der kan have en egentlig sandhedsværdi er så helt atomare observationelle udsagn af typen: „person NN observerede søndag 19. april 2020, at solen 385 stod op over København kl. 05.52“. Denne slags viden er ikke i sig selv ret interessant, men de logiske positivister hævdede, at den var *interesseløs*, dvs. en ren iagttagelse uden nogen indgriben eller indblanden fra observatøren. Ud fra indsamling af store mængder af sådanne iagttagelser ville de så foretage en *induktiv slutning* til nogle lovmaessigheder og teorier, der omfattede de gjorte iagttagelser (se figur 4). Denne induktive proces er selvfølgelig kritisk i vores erkendelse, og de logiske positivister mente, at den kunne foretages entydigt ud fra givne data—eller i hvert fald at den kunne underkastes krav om simplicitet, der sikrede 390 dens gyldighed. På den måde ville vi have opnået teorier om, hvordan verden *faktisk er*, og derudfra kunne man *deduktivt* slutte sig frem til forudsigelser og forklaringer af fremtidige og kendte fænomener ud fra, at de faldt under en given teori.

390

395



Figur 4: Skematiske oversigt over erkendelsesprocessen ifølge logisk positivisme; baseret på Chalmers (1995, s. 34).

Den logiske positivisme løb igennem 1900-tallet ind i en række uoverstigelige problemer, 400 hvoraf de to vigtigste i denne sammenhæng er: 1. *induktionsproblemets*, som ikke gik væk,

og 2. umuligheden af at have rå, teoriløse data. Disse to problemer skal vi behandle lidt mere indgående, fordi de vender tilbage, når vi i kapitel 6 skal diskutere *machine learning* som et potentielt nyt videnskabsteoretisk paradigme.

405 *Induktionsproblemer* ligger faktisk allerede til grund for selve udfordringen om at ville generalisere fra et endeligt antal observationer til en generel lovmaessighed. Problemet, som også kaldes for *Humes problem* efter den skotske oplysningsfilosof DAVID HUME (1711–1776), påpeger netop, at den form for induktiv slutning fra fortiden til fremtiden er betinget af en antagelse om universalitet, således at de samme lovmaessigheder gælder altid. Og det 410 lyder jo meget plausibelt, men vores eneste begrundelse for denne antagelse er selv, at den har virket i fortiden, så hvis vi vil slutte, at det også gælder i fremtiden er det netop en induktion af den type, princippet netop skulle være løsningen på. På spørgsmålet „Hvordan ved du, at solen står op i morgen tidlig?“ kunne du have lyst til at svare, at det siger teorien at den gør ud fra, at det har den gjort hver morgen indtil nu. Når jeg så spørger dig, hvorfor 415 du har lov til at slutte fra fortiden til fremtiden, kan du ikke svare noget meget bedre, end at det har hidtil vist sig at være en acceptabel og pålidelig praksis. Derfor har vi at gøre med en såkaldt *regres*, dvs. en uendelig gentagen afhængighed, som vi ikke kan finde uafhængig grund under. Og selvom de logiske positivistre forsøgte, så er denne indvending ikke blevet løst af slutionsformen i figur 4. For hvis vi vil sikre os, at vores induktioner fra tidlige 420 observationer også har relevans i morgen, så kan vi jo netop ikke referere til lovmaessigheder, der selv er grundet i denne form for permanens, uden at ende i en regress.

Den anden vigtige udfordring for de logiske positivistre handler om muligheden af at have ’rå, teori- og interesseløse data’. Ideen om, at jeg stiller mig op og foretager en observation af fx solopgangen over København, og at denne observation derefter er ubetvivlelig 425 er meget tillokkende. Jeg kunne selvfølgelig lyve om min observation, men det ville så blive opdaget, når der indløb mange andre, afgivende observationer fra andre, mere sanddruelige mennesker. Men selve ideen er, som senere filosoffer påpegede, problematisk, fordi vi slet ikke kan lave iagttagelser uden at have foretaget valg. Forestil dig, at jeg giver dig en blyant i hånden og bare kommanderer: „Iagttag!“ Hvad ville du begynde at skrive ned? Hvad ville 430 være vigtigt? Hvad ville du filtrere fra? På samme måde, blev det påpeget, kan man ikke foretage observationer uden at have en forventningshorisont, og sådan en er i sig selv givet af ens teorier og hypoteser om, hvad der vil komme til at ske. På den vis findes der slet ikke „rå“ data, og den generalitet, som de logiske positivistre kunne uddrage af deres objektive iagttagelsesdata viste sig at være en illusion.

435 En af de vigtigste reaktioner på den logiske positivisme blev formulert af filosoffen KARL POPPER (1902–1994), der også stod bag en del af kritikken af den tidlige position. Centralt for POPPER stod, at det ligesom var ’for nemt’ at producere sande videnskabelige udsagn, hvis de kun skulle omfatte de kendte iagttagelser (Popper, 1996). Hele den videnskabelige proces burde, mente han, være meget mere modig og ekspansiv: Det galdt om at forstå 440 mere, og derfor måtte man tage mere fejl, for vi lærer igennem vores fejl, ifølge POPPER.

POPPER formulerede derfor en videnskabelig metode baseret på *falsifikation* i stedet for de logiske positivisters *verifikationisme*: Pointen skulle være at formulere interessante udsagn og ihærdigt forsøge at modbevise (gendrive, falsificere) dem.

For at denne proces skal kunne virke, må man først igennem en heuristisk fase, hvor 445 forskellige muligheder åbent afsøges, inden man kan få en god ide om, hvad man ønsker at undersøge (se figur 5). Derefter følger en foreløbig undersøgelse, hvor der udføres nogle forsøg med henblik på at opstille en teori eller *hypotese*, hvis gyldighed rækker ud over de allerede foretagne forsøg. Denne heuristiske del, frem til formuleringen af den første hypotese, er ikke noget, POPPER beskrev i stor detaljer, og man har senere diskuteret, hvori denne del

af videnskaben egentlig består.

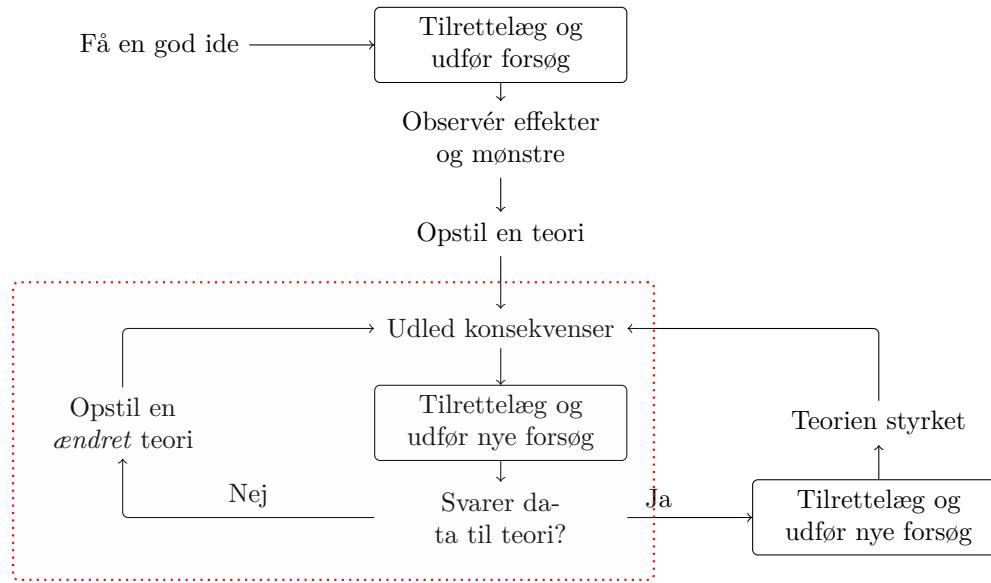
450

Den centrale del af POPPERS metode, som kaldes den *hypotetisk deduktive metode*, foregår i den nederste halvdel af figur 5 og starter, når vi har opstillet en hypotese eller teori. Ud fra vores formodning, som skal være *modig* (eng: „bold conjecture“) i den forstand, at det skal være *muligt* og måske endda *sandsynligt* at gendrive den, skal vi udlede *testbare* konsekvenser. Det vil sige, at vores formodning ikke i sig selv skal kunne undersøges empirisk, men den skal have nogle konsekvenser, som vi kan afprøve om holder eller ej. Når vi så tilrettelægger og udfører forsøg for at teste de udledte konsekvenser, kan der jo ske en af to ting: Enten svarer vores nye data til vores teori/hypotese, eller også gør de ikke. Det mest interessante for POPPER er, når data *ikke* stemmer med hypotesen, for så har vi *falsificeret* de udledte konsekvenser og dermed også selve hypotesen, og dermed har vi lært noget nyt om, hvordan virkeligheden *ikke* er indrettet. Skulle det ske, at vores data ikke gendriver hypotesen, så bliver vores tillid til hypotesen *bestyrket*, men det er afgørende for POPPER, at vi *aldrig* kan komme til at *bevise* vores hypotese: Vores viden er altid foreløbig, og den bedste viden, vi har, er den, der er blevet udsat for de mest ihærdige forsøg på gendrivelse uden held.

460

Det er afgørende for POPPERS metode, at der findes en fast ’virkelighed’, som vi kan spørge til råds igennem test, når vi forsøger at gendrive vores hypotese. Denne eksterne virkelighed kan være svær at indfange i fx socialvidenskaberne, men POPPERS hypotetisk-deduktive metode er alligevel ofte og fejlagtigt fremhævet som *den* (natur)videnskabelige metode. Ofte, fordi den er relativt let at forstå og tilsyneladende en god forklaring af vores videnskabelige videns validitet og videnskabens idealiserede metode, og fejlagtigt, fordi den ikke svarer ret godt til moderne videnskabelig praksis, hvor adskillelsen af hypotese og test ikke er så simpel, og hvor andre metoder som simulering og eksplorativ eksperimentation spiller stadig større roller (Andersen og Hepburn, 2016).

470



475

Figur 5: Skematisk fremstilling af POPPERS hypotetisk-deduktive metode.

Men også internt har POPPERS metode nogle ret grumme filosofiske udfordringer, hvoraf særligt to er vigtige i vores sammenhæng, og de handler begge om den afgørende afprøvning af en hypotese i en test, der kan give et af to mulige svar, og rammer altså den røde

firkant i figur 5: 1. Hvis vi er i færd med helt ny erkendelse, er det ikke nemt at afgøre, hvornår et forsøg er udført korrekt uden at kende det korrekte svar — denne udfordring kaldes *experimenters' regress*. 2. Hvis vi ønsker at teste en hypotese, så gør vi det aldrig i isolation, så hvilken hypotese er det faktisk, vi har falsificeret — denne udfordring kaldes *Duhem-Quine-tesen* efter fysikeren PIERRE DUHEM (1861–1916) og filosoffen WILLARD QUINE (1908–2000).

*Experimenters' regress* er endnu et eksempel på et filosofisk argument, som kræver en form for prævilegeret standpunkt for ikke at spiralere ud af kontrol i en uendelig, ond cirkel. For hvis vi nu befinner os på virkelig ny grund og skal afprøve en hypotese ved at teste en af dens konsekvenser, så skal vi jo vide, at vores apparatur og eksperiment er korrekt for at vide, om vi kan stole på vores test, eller der har sneget sig en fejl ind i apparauret eller udførelsen. Men hvordan ved vi, at et nyt apparatur er korrekt, hvis det, det skal måle, er noget, vi ikke ved noget om i forvejen? Så på den måde bliver vores viden om apparaurets korrekthed og vores viden om det observerede fænomen vævet godt og grundigt sammen. Denne regres blev påpeget af videnskabssociologerne HARRY COLLINS og TREVOR PINCH (Collins og Pinch, 2003) baseret på en analyse af bestræbelserne på at detektere tyngdebølger. Deres analyse mundede ud i, at regressen ikke kunne opnås uden inddragelse af ekstra-videnskabelige, særligt sociale, argumenter om fx anerkendelse og prestige.

Udfordringen fra DUHEM og QUINE stammer fra, at de påpeger, at man jo aldrig tester en enkelt hypotese i isolation, men at enhver teori gør brug af et helt netværk af andre hypoteser. Så det, vi faktisk udsætter for den centrale afprøvning i POPPERS model er aldrig en enkelt, isoleret hypotese  $H_0$ , men en holistisk teori bestående af mange baggrundsantagelser  $A_1, \dots, A_n$ . Og hvis den så bliver falsificeret af vores forsøg, så har vi altså i matematisk notation

$$\neg(H_0 \wedge A_1 \wedge \dots \wedge A_n),$$

hvor  $\neg$  betegner negation og  $\wedge$  betegner konjunktion (logisk 'and'). Og ud fra dette følger det logisk, at

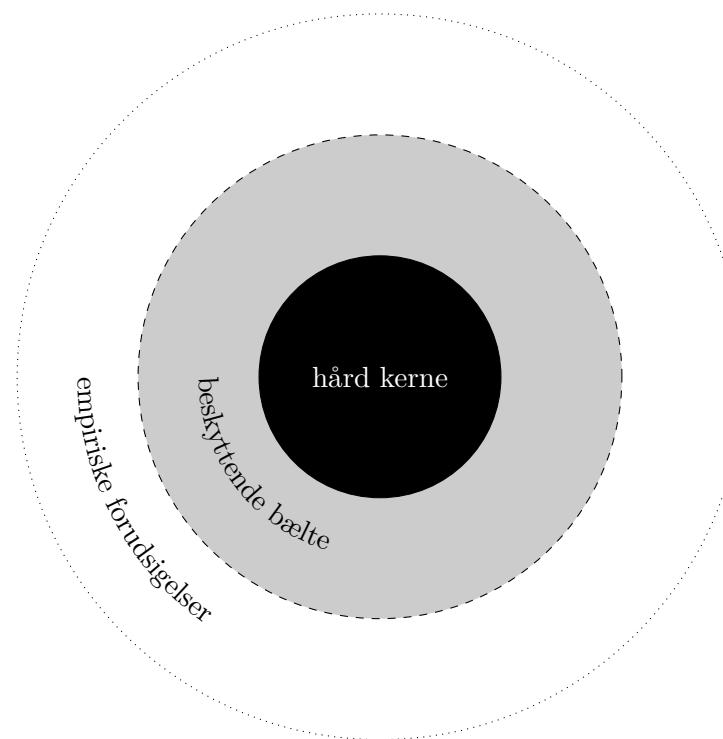
$$(\neg H_0) \vee (\neg A_1) \vee \dots \vee (\neg A_k),$$

hvor  $\vee$  betegner disjunktion (logisk 'or'). Så rent logisk kan vi ikke sige, at det er  $H_0$ , vi har gendrevet, og påstanden i *Duhem-Quine-tesen* er faktisk, at det er muligt at tilføje ad-hoc-antagelser, sådan at gendriven ligger i en vilkårlig antagelse i netværket. Med dette holistiske syn på teorier og deres afprøvning, punkterede DUHEM og QUINE selve den centrale hypotese-test i POPPERS metode, og samtidig pegede de på en af de vigtige mekanismer, som POPPER manglede at gøre rede for, nemlig hvad man stiller op, når ens teori er blevet gendrevet. I sin yderste konsekvens ville POPPER skulle starte forfra, men det virker jo hverken særligt effektivt eller intuitivt.

Et af det første gode forsøg på at løse denne udfordring blev givet af POPPERS elev IMRE LAKATOS (1922–1974), som formulerede den i termer af *videnskabelige forskningsprogrammer*. Ifølge LAKATOS skal man ikke se hypoteserne og teorierne i isolation, men som dele af forskningsprogrammer, der indeholder en *hård kerne* af de forudsætninger, vi ikke er villige til lade gendrive, et beskyttende bælte omkring, hvor vi kan lave mindre justeringer, og et bælte af empiriske forudsigelser udenom igen (se figur 6). På den måde isolerede LAKATOS den hårde kerne fra kritikken fra DUHEM og QUINE, og han foreskrev, at formålet med et forskningsprogram er at gøre stadig flere (interessante) empiriske forudsigelser. Derfor inddelte han forskningsprogrammer i *progressive*, hvis de over tid voksede og tilføjede nyt

empirisk indhold, og *degenerative*, hvis de stagnerede. LAKATOS' metode om videnskabelige forskningsprogrammer havde en vis medvind, da den stadig hævdede en rent intern, videnskabelig og rationel dynamik i videnskabernes udvikling og erkendelse. Men den blev i nogen grad overhalet af THOMAS KUHNS (1922–1996) teori om videnskabelige paradigmer, som vi skal diskutere i kapitel 2 og 3.

515



Figur 6: LAKATOS' revision af POPPERS program delte en teori ind i en uimodsigelig 'hård kerne' med et 'beskyttende bælte' omkring.

## 1.6 Den autonome videnskabs nytte

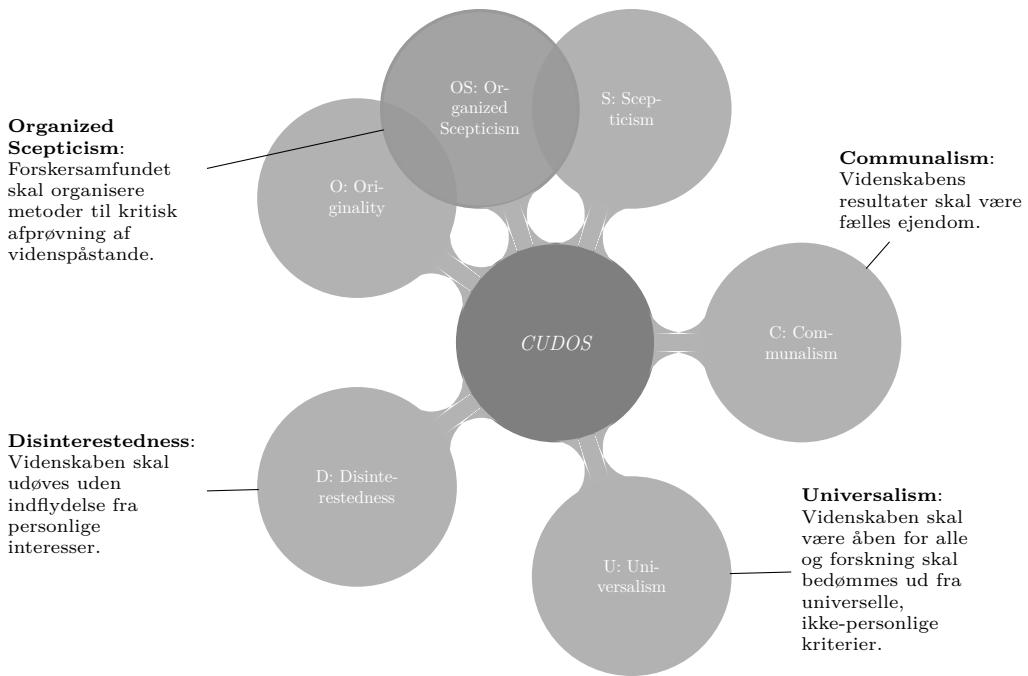
520

KRAGHS definition af videnskabelig viden som offentlig, fejbarlig, korrigerbar og testbar ligner på flere punkter den *sociologiske karakteristik* af videnskab, som videnskabssociologen ROBERT K. MERTON (1910–2003) udformede som en *normativ videnskabens etos* (Merton, 1973). MERTON lavede sin analyse i 1942 under indtryk af totaliære regimers korruption af den frie videnskab, ikke mindst i form af den såkaldte „tyske (ariske) fysik“. Som modvægt opstillede MERTON et normsæt for god og fri videnskabelighed, som siden kom til at stå som et ideal for den akademiske videnskab i den vestlige verden. MERTONS normsæt bærer akronymet *CUDOS*, hvilket står for *Communalism*, *Universality*, *Disinterestedness*, og *Organized Scepticism*. Dermed fanger MERTON ligesom KRAGH, at den videnskabelige viden skal være fælles og offentlig (C). Videnskaben skal have en grad af universel gyldighed, der ikke afhænger af den enkelte videnskabsmand, og ingen skal udelukkes fra det videnskabelige samfund (U). Omvendt må videnskabsmanden ikke lade egne interesser påvirke den videnskabelige proces eller det videnskabelige udkomme (D). I stedet skal han åbent overlade sine resultater til det videnskabelige samfunds organiserede (dvs. institutionaliserede og systematiske) kritik (OS).

525

530

535



Figur 7: MERTONS normer for akademisk videnskab (CUDOS).

Dette sæt normer for videnskaben var allerede eksisterende inden MERTON formulerede det, men efterfølgende blev det implementeret i endnu højere grad i et argument for at sikre den frie grundvidenskab. Under 2. verdenskrig havde især amerikanske videnskabsfolk leveret videnskabeligt baseret viden, som hjalp med til at løse konkrete logistiske og teknologiske problemer. Henimod krigens afslutning fremsatte den amerikanske forsker og forskningspolitiker VANNEVAR BUSH (1890–1974) et notat til præsidenten, hvori han argumenterede dels for den grundvidenskabelige forsknings autonomi og dels for dennes uforudsigelige anvendelighed. BUSH hævdede blandt andet:

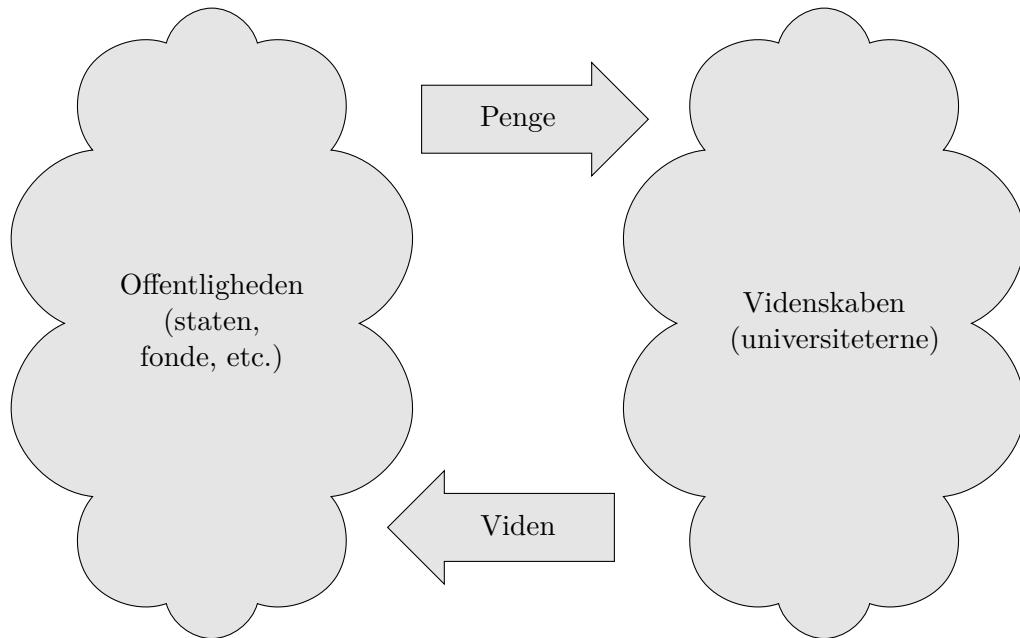
Videnskabeligt fremskridt over en bred front stammer fra frie intellektuelles frie leg, hvor de arbejder med problemstillinger, som de selv har valgt, kun diktteret af deres nysgerrighed efter at udforske det ukendte. Forskningsfriheden bør bevares i enhver regerings forskningspolitiske reformer. (Bush, 1945)

I løbet af de første år af Anden Verdenskrig mobiliserede USA landets akademiske ressourcer for krigsindsatsen. En række institutioner blev oprettet med det formål at styrke den krigsrelevante forskning, herunder Office of Scientific Research and Development (OSRD) og Office of Naval Research (ONR). Efter krigen videreførtes nogle af de udviklede ideer i nationale, civile forskningsråd som National Science Foundation (NSF) og National Research Council (NRC). Den centrale ide var at forsøge at styre forskningen i anvendelsesretning ved at finansiere forskningsprojekter med anvendelser for øje. Dermed indgik militæreret, industrien og universiteterne i strategiske samarbejder. Under krigen rekrutterede OSRD akademiske medarbejdere til at arbejde på støttede projekter, men medarbejderne forblev på universiteterne for at binde miljøerne sammen. Forskerne var civile og uden for militærrets direkte indflydelse, hvilket betød at de også var i stand til at forfølge mål, der ikke umiddelbart kunne forudsiges at give anvendelser. Efter krigen videreførtes meget af denne

tænkning i koldkrigsforskningen, som i hele den vestlige verden var influeret af BUSH' rapport „Science: The Endless Frontier“ (Bush, 1945). Argumentet deri var, at grundforskning fører til videnskabelig kapital, som kan veksles til praktiske anvendelser. Grundforskningen er — ifølge BUSH, som var rapportens forfatter — hjerteslaget i det teknologiske fremskridt, og en svag grundforskning vil give langsommere industrielle fremskridt og svag økonomi (og militær). Derfor er det i statens interesse at støtte grundforskningen. Ifølge BUSH skulle grundforskningen endvidere være *frei*, fordi anvendelser ville følge næsten umiddelbart. Denne tænkning er blevet kaldt *den lineære model* fra grundforskning til anvendelser, og den er siden blevet kritiseret meget, ikke mindst fordi anvendelser kræver betydeligt mere end simpel applicering af grundvidenskabelig viden (se kap. 2).

565

570



Figur 8: Model for samspillet mellem videnskab og samfund under *mode-1-videnskab*.

Den model, som BUSH og MERTON argumenterede for tildeler videnskaben — og herunder grundvidenskaben — en meget høj grad af autonomi. Staten finansierer videnskaben (dvs. universiteterne) med det formål at få produceret ny viden til rådighed, uden at der stilles krav til fx denne videns anvendelighed eller nytte (se figur 8). Denne model for samspillet mellem stat og videnskab kaldes nogle gange for *mode-1-videnskab*, og den hævdede videnskabens *autonomi* og beroede på, at det akademiske samfund bedst regulerede sig selv. Samfundet og dets politikere burde overlade det til videnskabsmændene og deres institutioner selv at afgøre forskningspolitiske prioriteringer. Og til gengæld skulle videnskaben så være selvregulerende, i høj grad efter de normer, som MERTON havde præsenteret. Som videnskabshistorikeren DEREK J. DE SOLLA PRICE (1922–1983) beskrev:

575

580

Da videnskaben er international og konkurrencepræget, er dens måde at vokse på næsten udelukkende diktteret af dens aktuelle kognitive stade og næsten slet ikke af nationernes ønsker eller samfundets behov. Man har derfor ikke noget virkelig forskningspolitisk valg, uddover at støtte al eksisterende frontforskning med det størst mulige beløb og de største antal talenter, der kan presses ud af befolkningen. (Solla Price, 1965, s. 237)

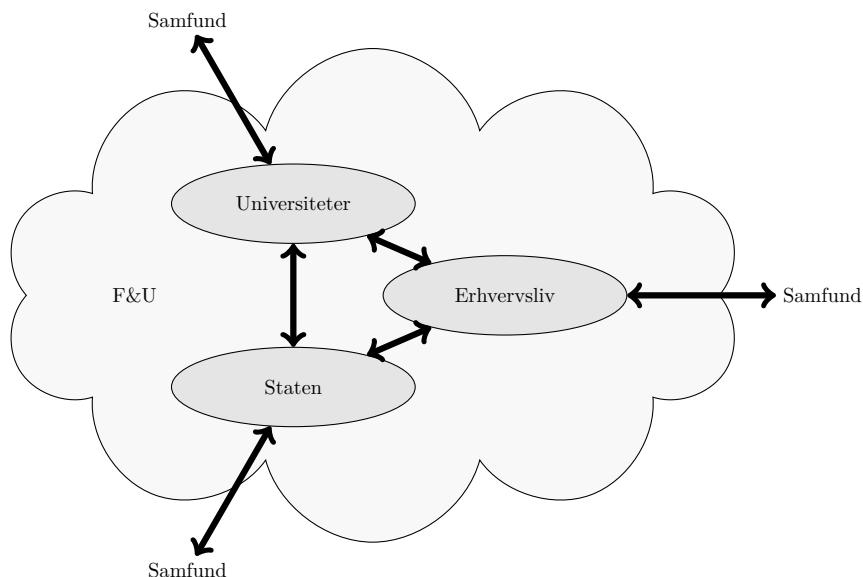
585

Denne autonomi skulle imidlertid i stigende grad blive udfordret siden DE SOLLA PRICES analyse, og i det følgende skal vi se på nogle af de udfordringer, der er blevet rejst mod den.

## 590 1.7 Fra akademisk til post-akademisk videnskab?

MERTONS *CUDOS* var ment som en norm, der skulle beskytte videnskaben — og især den rene, nysgerrighedsdrevne videnskab — i efterkrigsperioden. Men dens relevans som en faktisk beskrivelse af videnskabelig praksis viste sig i løbet af især siden 1980’erne at være aftagende: Universiteterne var ikke længere kun isolerede forskningsinstitutioner bestemt for de få og præstigeberedte, og samtidig opstod der forskningsmiljøer i mange andre sammenhænge, som ikke naturligt var underlagt et akademisk formål og normsæt.

595 For bedre at indfange den nye forskningsdagsorden har man indført begrebet *mode-2-videnskab*, som altså er en ny tilstand efter det rent akademiske *mode-1-videnskab* beskrevet i figur 8. I mode-2-videnskab er autonomien mellem videnskab og stat erstattet med et nært sammenspundet netværk af forskning og udvikling, som involverer staten, universiteterne 600 og erhvervslivet. Dette netværk står i en tovejs-relation til det omgivende samfund, som både definerer rammer og behov for forskning og udvikling. Og selve denne forskning kan udføres inden for de flydende grænser af forsknings- og udviklingsnetværket (F&U, eng: *Research and development, R&D*), således at fx erhvervsliv og universiteter kan samarbejde 605 om forskningsprojekter og endda dele infrastruktur og kommercialisering af forskningen.



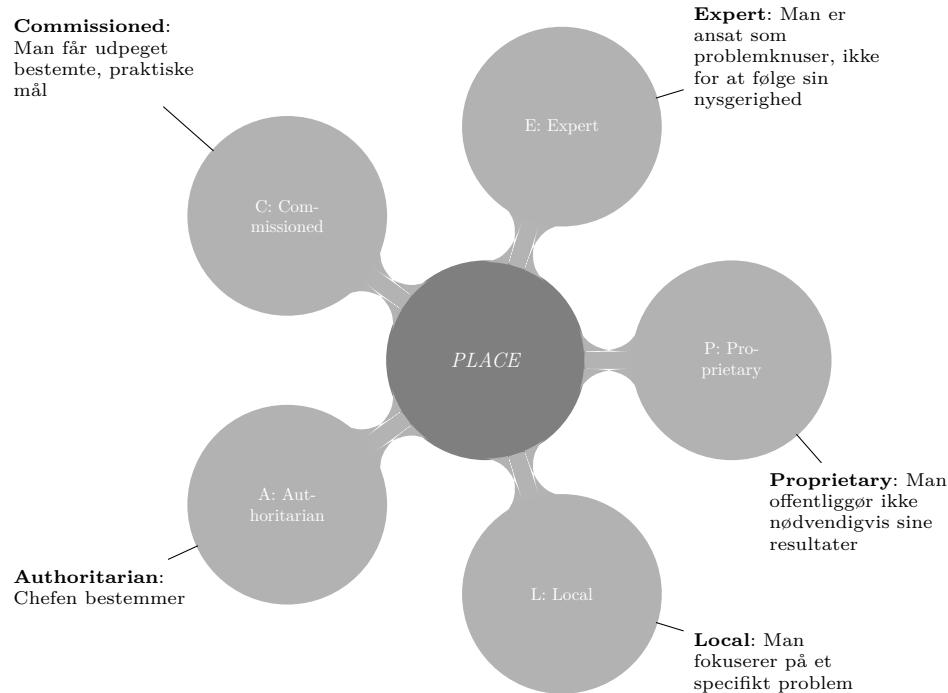
Figur 9: Model for relationerne mellem forsknings- og udviklingsnetværk (F&U) og det omgivende samfund under *mode-2-videnskab*.

Som en beskrivelse af denne nye, post-akademiske videnskab har JOHN ZIMAN (1925–2005) formuleret et deskriptivt akronym som kontrast til MERTONS normative *CUDOS* (se figur 10). ZIMAN kalder sit akronym for *PLACE*, hvilket dækker over (Ziman, 2000, s. 67–82):

P: At den producerede viden er *proprietær*, dvs. er ejet af nogen, enten en virksomhed eller et universitet eller en blanding deraf, og at den kan kommercialiseres.

- L: At den producerede viden er *lokal* i den forstand, at den er rettet mod at løse et bestemt problem og ikke en blind nysgerrighed over for verdens indretning.
- A: At forskningen er *autoritær*, dvs. at der er nogen højere oppe end den enkelte forsker, der bestemmer, hvilken retning, forskningen skal tage. 615
- C: At forskningsprocessen er *styret* (eng: *commissioned*) efter bestemte, udpegede, praktiske målsætninger.
- E: At forskeren er ansat som *ekspert* til at hjælpe med at løse de stillede problemer, ikke til at følge sin egen nysgerrighed. 620

Som det fremgår er det både den enkelte forskers autonomi og det videnskabelige samfunds selvregulering, der adskiller *CUDOS* og *PLACE*. Og det er også forbundet med meget store diskussioner af forskningens betydning og forskerens identitet at analysere, om der er tale om en udvikling eller overgang fra *CUDOS*-baseret akademiske institutioner til mere *PLACE*-baseret forskning (se også Hansen og Johansen, 2007). Men man kan i hvert 625 fald konstatere, at samarbejder mellem erhvervsliv og universiteter i dag er en helt naturlig (og økonomisk helt afgørende) ting for universiteterne, at stadig flere af statens midler til forskning gives i form af strategiske bevillinger til forskning inden for områder af stor samfundsrelevans (for tiden især klima og grøn omstilling), og at universitetsloven fra 2003 indførte et ledelsesstyret universitet med mulighed for pålægge forskere specifikke opgaver. 630 Det kan lyde meget godt, og det har da også bidraget til en stor vækst i forskning i det danske samfund, men det har også nogle potentielle slagsider for mindre vidensområder, der ikke kan påvise kortsigtet nytte. Og for den enkelte forsker var autonomien til valg af forskningsområde en rettighed stadfæstet ved det tyske begreb „Forschungs- und Lehrfreiheit“, som er blevet hyldet siden 1800-tallet (Humboldt, 2007). 635



Figur 10: ZIMANS beskrivelse af post-akademisk videnskab (*PLACE*).

## Litteratur

- Andersen, Hanne og Brian Hepburn (2016). „Scientific Method“. I: *The Stanford Encyclopedia of Philosophy*. Red. af Edward N. Zalta. Summer 2016. Metaphysics Research Lab, Stanford University.
- Atchison, William F. m.fl. (mar. 1968). „Curriculum 68. Recommendations for academic programs in computer science“. A report of the ACM curriculum committee on computer science. *Communications of the ACM*, bd. 11, nr. 3, s. 151–197. DOI: 10.1145/362929.362976.
- Bush, Vannevar (1945). *Science: The Endless Frontier*. A Report to the President by Vannevar Bush, Director of the Office of Scientific Research and Development, July 1945. Washington: United States Government Printing Office.
- Chalmers, A. F. (1995). „Induktivisme: Videnskabelig erkendelse er udledt af erfaringen“. I: *Hvad er videnskab? En indføring i moderne videnskabsteori*. Overs. af G. Lyngs. Filosofi. Gyldendal. Kap. 1, s. 27–41.
- Collins, Harry og Trevor Pinch (2003). *The Golem. What You Should Know about Science*. 2. udg. Canto. Cambridge: Cambridge University Press.
- Dasgupta, Subrata (2016). *Computer Science. A very short introduction* 466. Oxford m.m.: Oxford University Press.
- Den Store Danske Encyklopædi* (1994). 23 bd. København: Gyldendal.
- Fink, Hans (2003a). „Hvad er et universitet? Om universitetets idé og realitet i det 21. århundrede“. I: Fink, Hans m.fl. *Universitet og Videnskab. Universitetets idéhistorie, videnskabsteori og etik*. København: Hans Reitzels Forlag. Kap. 1, s. 9–29.
- (2003b). „Universitetsfagenes etik“. I: Fink, Hans m.fl. *Universitet og Videnskab. Universitetets idéhistorie, videnskabsteori og etik*. København: Hans Reitzels Forlag. Kap. 4, s. 193–221.
- Hansen, Henning Bernhard (2000). „Datalogi“. I: *Den Store Danske Encyklopædi*. Bd. 5. 23 bd. København: Gyldendal, s. 25–26.
- Hansen, Tom Børksen og Mikkel Willum Johansen (maj 2007). „Post-akademisk videnskab“. *Aktuel Naturvidenskab*, nr. 2, s. 30–33.
- Humboldt, Wilhelm von (2007). „Om den indre og ydre organisation af de højere videnskabelige læreanstalter i Berlin“. I: *Ideer om et Universitet: Det moderne universitetets idehistorie fra 1800 til i dag*. Red. af J. E. Kristensen m.fl. Overs. af M. Pedersen og J. V. Nielsen. Aarhus: Aarhus Universitetsforlag, s. 89–96.
- Johansen, Mikkel Willum og Henrik Kragh Sørensen (2014). *Invitation til matematikkens videnskabsteori*. København: Forlaget Samfundslitteratur.
- Kragh, Helge (2001). „Videnskab“. I: *Den Store Danske Encyklopædi*. Bd. 20. 23 bd. København: Gyldendal, s. 139–145.
- (2003). „Hvad er videnskab?“ I: Fink, Hans m.fl. *Universitet og Videnskab. Universitetets idéhistorie, videnskabsteori og etik*. København: Hans Reitzels Forlag. Kap. 3, s. 145–192.
- Merton, Robert K. (1973). „The Normative Structure of Science“. I: *The Sociology of Science: Theoretical and Empirical Investigations*. Chicago og London: University of Chicago Press. Kap. 13, s. 267–278.
- Popper, Karl R. (1996). „Videnskaben“. I: *Kritisk rationalisme. Udvalgte essays om videnskab og samfund*. Overs. af Knud Haakonssen og Niels Chr. Stefansen. København: Nyt Nordisk Forlag, s. 40–53.

- Solla Price, Derek de (apr. 1965). „The Scientific Foundation of Science Policy“. *Nature*, bd. 206, nr. 4981, s. 233–238.
- Sørensen, Henrik Kragh (2006). „Matematik, statistik og datalogi“. I: *Viden uden grænser, 1920–1970*. Red. af Henry Nielsen og Kristian Hvidtfelt Nielsen. Dansk Naturvidenskabs Historie 4. Aarhus: Aarhus Universitetsforlag. Kap. 6, s. 105–124. 685
- Ziman, John (2000). *Real Science: What it is, and what it means*. Cambridge etc.: Cambridge University Press.

## 690 Navneliste

- Bundgaard, Svend (1912–1984), 3  
Bush, Vannevar (1890–1974), 16, 17  
  
Carnap, Rudolf (1891–1970), 11  
Collins, Harry, 14  
  
695 de Solla Price, Derek J. (1922–1983), 17, 18  
Duhem, Pierre Maurice Marie [Pierre] (1861–1916), 14  
  
Fink, Hans, 8, 9  
  
700 Hume, David (1711–1776), 12  
  
Kragh, Helge, 5–7, 11, 15  
Kuhn, Thomas Samuel [Thomas] (1922–1996), 15  
  
705 Lakatos, Imre (1922–1974), 14, 15  
Merton, Robert K. (1910–2003), 15–18  
  
Naur, Peter (1928–2016), 2  
  
Pinch, Trevor J. [Trevor], 14  
Popper, Karl Raimund [Karl] (1902–1994), 12–15  
  
710 Quine, Willard Van Orman [Willard] (1908–2000), 14  
  
von Humboldt, Friedrich Wilhelm Christian Karl Ferdinand [Wilhelm] (1767–1835), 8  
  
715 Ziman, John Michael [John] (1925–2005), 18, 19

## Indeks

- arisk fysik, 15  
  
CUDOS-normerne, 15, 16, 18, 19  
  
720 Duhem-Quine-tesen, 14  
  
epistemologi, 3  
experimenters' regress, 14  
  
falsifikation, 12  
fejlbarlig viden, 5  
  
725 forskningsbaseret undervisning, 8  
forskningsfrihed, 8, 19  
  
HCI, 10  
human-computer-interaction, 10  
Humes problem, 12  
  
730 hypotetisk deduktive metode, 13  
  
induktionsproblem, 11, 12  
  
korrigérbar viden, 5  
kulturprodukter, 9  
kvalitative metoder, 10  
  
735 kvantitative metoder, 10  
  
National Research Council (NRC), 16  
  
740 National Science Foundation (NSF), 16  
naturprocesser, 9  
NRC, *Se* National Research Council  
NSF, *Se* National Science Foundation  
  
offentlig viden, 5  
Office of Naval Research (ONR), 16  
Office of Scientific Research and  
Development (OSRD), 16  
  
745 ONR, *Se* Office of Naval Research  
ontologi, 3  
OSRD, *Se* Office of Scientific Research  
and Development  
  
PLACE-karakteriseringen, 18, 19  
  
750 sociale systemer, 9  
  
testbar viden, 5  
totalitært regime, 15  
  
undervisningsfrihed, 8, 19  
universitetets selvstyre, 8  
  
755 verifikationisme, 12  
viden

fejlbart, 5	testbar, 5	760
korrigérbart, 5	videnskabeligt, 4	
offentligt, 5	videnskabens enhed, 8	

# Kapitel 2:

## Datalogi og teknologi

Henrik Kragh Sørensen

Mikkel Willum Johansen

22. april 2022

### Indhold

5

2.1	Fra grundforskning til innovation . . . . .	1
2.2	Teknologiske systemer . . . . .	6
2.3	Verdenssyn og datalogiens teknologiske paradigme . . . . .	9
2.4	Vidensproduktionen i datalogi og informationsteknologi . . . . .	10

### Litteratur

12 10

## 2.1 Fra grundforskning til innovation

Den 'lineære model' betegner en ide om, at videnskabelig og teknologisk udvikling hører sammen, således at viden mere eller mindre automatisk omsættes til teknologiske fremskridt. Modellen udgør (eller udgjorde i hvert fald) et argument for grundvidenskabelig forskning: Hvis man som samfund ønsker teknologisk, militær og økonomisk innovation, var svaret i anden halvdel af 1900-tallet i høj grad baseret på omfattende investeringer i grundforskning. Og som beskrevet i kapitel 1 går denne argumentation tilbage til VANNEVAR BUSH (1890–1974) rapport fra slutningen af Anden Verdenskrig:

15

Nye produkter, nye industrier og flere jobs kræver vedvarende tilføjelser til vores viden om naturens love og anvendelsen af denne viden til praktiske formål.  
[...] Denne essentielle nye viden kan kun opnås gennem videnskabelig grundforskning. (Bush, 1945)

20

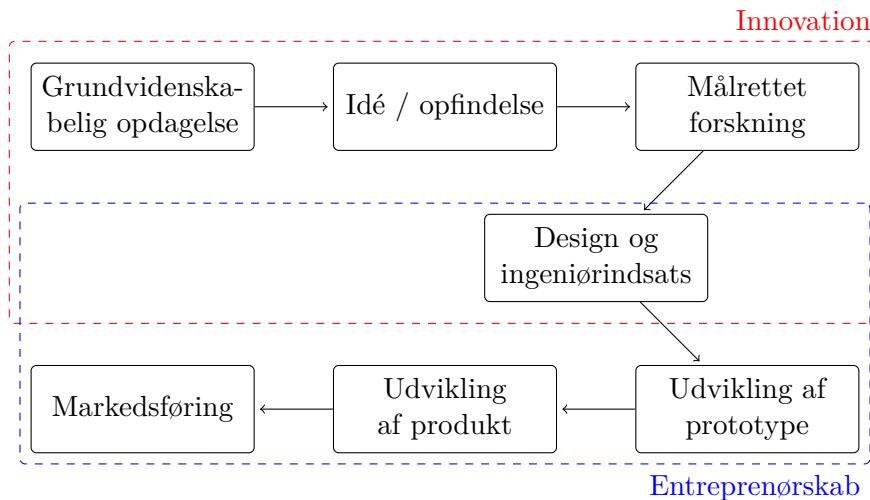
Argumentet i anden halvdel af citatet udtrykker altså en klar årsagssammenhæng, som hævder, at nye produkter kræver grundforskning. Men argumentet blev også anvendt i den anden retning, som et argument for en lineær proces fra grundforskning til produkt, som man kan anskueliggøre som en slange af veldefinerede og adskilte skridt, se figur 1 (Kline og Rosenberg, 2009). Processen begynder med en grundvidenskabelig erkendelse, måske erhvervet under ren nysgerrighedsdrevet forskning. Denne erkendelse omsættes til en opfindelse, idet nogen får en ide til at udnytte den til et praktisk formål. Derefter følger en fase med målrettet forskning, hvor der igen tilvejebringes ny viden, men denne gang med det formål at få ideen til at fungere. Derefter skal denne viden omsættes til et produkt, og heri indgår både ingeniørvidenskab, som afdækker de tekniske muligheder

25

30

og begrænsninger, og design af æstetisk og praktisk karakter. Derefter er man i stand  
 35 til at lave en prototype, som kan afprøves, inden den omsættes til et produkt, der kan (masse)produceres og markedsføres. Modellen, som den er beskrevet her, kan betragtes som omfattende to dele: en „innovationsdel“, som er karakteriseret ved at producere viden, som skaber værdi og nytte, og en „entreprenørskabsdel“, der omsætter denne viden til faktiske produkter i markedet. Og fordi denne model antyder en automatisk og fremadskridende proces, kan man også kalde den for *samlebåndsmodellen for teknologisk innovation*.

40



Figur 1: Skematisk fremstilling af 'samlebåndsmodellen' for teknologisk innovation.

Med andre ord er der i den simplificerede model i figur 1 indbygget to antagelser om videnskabelig og teknologisk videns- og værdiskabelse: 1. Processen er (i høj grad) lineær i den forstand, at den er drevet af videnskabelige erkendelser, som nærmest automatisk kan omsættes til produkter. 2. Processen kan inddeltes i to faser, styret af forskellige logikker: innovation (vidensproduktion) og entreprenørskab (værdiproduktion). Begge disse antagelser er, som vi skal se, naive og ofte direkte misledende. For eksempel har en klar adskillelse af processerne konsekvenser for, hvordan etisk ansvar fordeles i innovation eller hvordan man politisk bedst understøtter ny teknologi, og det skal vi vende tilbage til i det følgende. Men måske endnu mere kritisk for modellen er det, at den ikke er passende som beskrivelse af *faktiske innovationer*, fordi den dels har en forsimplet opdeling af vidensproduktionens motivationer og sociale indlejring og dels fundamentalt bygger på et forsimplet billede af teknologi, som den betragter mere som artefakter end som systemer. Så for at diskutere og nuancere denne model er vi nødt til at inddrage faktiske innovationer og et mere nuanceret billede på ingeniørfaget.

For det første er teknologisk innovation og grundforskning slet ikke altid så klart adskilte og modsatte processer. Det kan man illustrere ved en nuancering af samlebåndsmodellen ved hjælp af kvadranter (figur 2), sådan som politologen DONALD E. STOKES (1927–1997) præsenterede det i sin bog *Pasteur's Quadrant*. Deri forestillede han sig, at man supplerede det lineære spektrum udspændt af 'rendyrket grundvidenskab' og 'rendyrket anvendelsesvidenskab' med en yderligere dimension, således at der blev plads til blandede former for vidensproduktion (Stokes, 1997). Ved hjælp af et historisk eksempel førte dette ham frem til en model som i figur 2, hvor det særlige er den øverste højre kvadrant, opkaldt efter

LOUIS PASTEUR (1822–1895). For at forstå modellen, sådan som STOKES brugte den, kan man først skelne mellem de forskellige *formål*, som vidensproduktionen har:

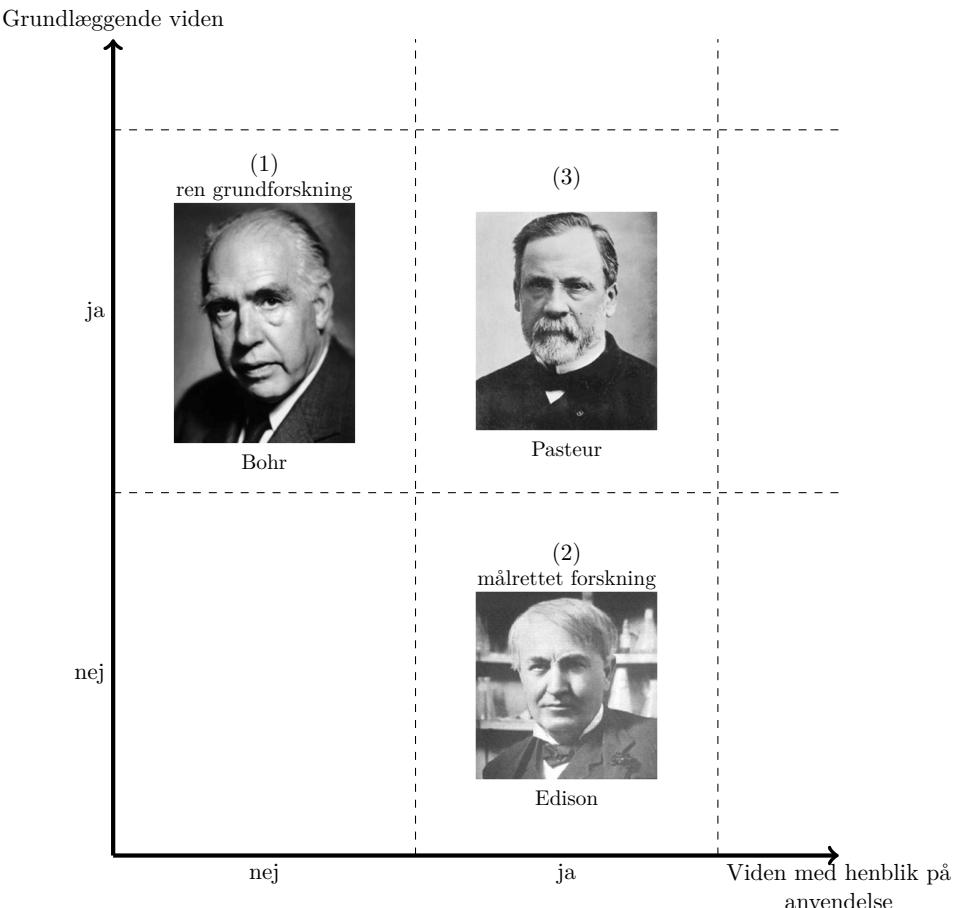
1. Den danske professor NIELS BOHR (1885–1962) er berømt for sin forskning i teoretisk fysik i begyndelsen af 1900-tallet, som bl.a. gav en ny forståelse af atomets indre struktur og særligt af kvantemekanikken. Disse teoretiske opdagelser tæller som nogle af de største videnskabelige gennembrud i 1900-tallet. BOHR publicerede mange videnskabelige artikler, og hans institut i København (grundlagt 1920) blev — og er — et internationalt centrum for teoretisk fysik. 65
2. Den amerikanske opfinder THOMAS EDISON (1847–1931) er berømt for sin entreprenante udnyttelse af nye videnskabelige emner som elektricitet, radio og lydoptagelse til kommercielle opfindelse som fx glødepæren, fonografen (apparat til lydoptagelse) og filmkameraet i anden halvdel af 1800-tallet. Disse opfindelser er centrale for den moderne verden og især for ekspansionen over det amerikanske kontinent. EDISON har udtaget mere end 1000 patenter i USA, og han byggede 'laboratorier', fx i Menlo Park (NJ), hvor hans ansatte arbejdede på nye opfindelser. 70  
75

Denne korte introduktion skelner altså mellem den rene grundforskning, som leder til opdagelser, og den rene anvendelsesforskning, som leder til opfindelser. Men fælles for de to yderpositioner er, at de begge er *forskning*, dvs. er centreret omkring vidensproduktion. Så de første forskelle skal nærmere findes i *formål* og *motivation*, og derefter kan vi begynde at udsonde forskelle i selve vidensproduktionen. For at se forskelle og overgange klarere, er det, at STOKES inddrager den franske biolog PASTEUR, hvis fascinerende karriere har gjort ham til et hyppigt brugt eksempel til at udfordre forsimplede forestillinger om videnskab. 80

3. PASTEUR var en fransk mikrobiolog og mediciner, som er særligt berømt for sine opdagelser af, at visse sygdomme skyldtes bakterieinfektioner, hvilket lagde grunden for vaccinationer, mikrobiel gæring og den proces, som vi stadig kalder pasteurisering. Disse opfindelser har enorm betydning for både det grundliggende sundhedsniveau og vores muligheder for sygdomsbehandling. PASTEURS opdagelser skaffede ham stor international berømmelse — og bragte ham også i en del kontroverser — og det institut i Paris, som han blev direktør for i 1888, blev et af de absolut førende steder at studere bakterier og sygdomme. 85  
90

Selvom PASTEURS forskning ledte til store medicinske gennembrud, var det ikke det, der fra starten motiverede ham. PASTEUR var landmandssøn, og hans forskning startede med studier af infektioner i silkeorme, miltbrand og hundegalskab. Fælles for disse studier fandt han, at en lang række smitsomme sygdomme skyldtes mikroorganismér. Denne viden var selvfølgelig grundvidenskabelig indsigt i sig selv, men PASTEURS motivation var også handlingsorienteret. Og han arbejdede derfor også innovativt med at omsætte sin grundvidenskabelige forskning til praktiske anvendelser både hos landmænd og hos læger igennem et program baseret på *Pasteur Instituttet* for at udbrede viden til de relevante brugere med henblik på at forebygge og behandle sygdomme. 95  
100

En af PASTEURS særligt vigtige videnskabelige erkendelser var, at hans eksperimenter, da de blev udført under helt sterile forhold, kunne modbevise antagelsen om, at liv (i form af bakterier) kunne opstå spontant. Denne kontrovers er også blevet anvendt til at illustrere udfordringerne indbygget i *experimenters' regress* (se kapitel 1), idet PASTEUR påviste, at det liv, som andre forskere havde observeret i deres prøver, stammede fra bakteriel forurening af prøven og ikke fra spontant opstået liv. 105



Figur 2: Kvadrantmodel over forskellige former for vidensproduktion, baseret på Stokes (1997, s. 73).

Den foregående historiske diskussion har nogle konsekvenser for, hvordan vi videnskabsteoretisk kan tænke om forholdet mellem grundvidenskabelig forskning og mere målrettet eller anvendelsesorienteret forskning. Man kunne forestille sig at adskille de to vidensformer langs en række dimensioner, hvoraf vi vil diskutere:

1. En formålsdimension, hvor man ser, at selvom der kan være forskellige motivationer for den enkelte forsker, så udgør det ikke et skarpt og konsistent skel mellem de to former for vidensproduktion.
2. En **temporal** (tidslig) dimension, som er indbygget i **den lineære model**, hvor den siger, at grundvidenskab kommer først og leder til anvendt videnskab.
3. En **kausal** dimension, som til dels forudsætter den foregående, og både siger, at anvendt videnskab er afhængig af grundvidenskab, og at grundvidenskab fører til anvendelser.
4. En **epistemologisk** dimension, hvor vi undersøger, hvilke erkendelsesmæssige forskelle, der måtte være mellem de to former for vidensproduktion og kommer frem til nogle vigtige indsigtter om teknologiske systemer.

Disse indsigt er nogle, som især teknologihistorikere og -filosoffer har uddraget af denne og lignende cases (se fx Nielsen m.fl., 2015), og vi behandler dem først en ad gangen, inden vi når til en fælles diskussion. 125

**Formål og motivation.** For det første viser eksemplerne, at det kan være nok så svært at skelne mellem de forskellige måder at producere viden på — især hvis vi ikke udelukkende forholder os til mode-1-videnskab. Mange af de processer, der indgår i forskningen, vil være de samme: teoretiske overvejelser, formulering af hypoteser, opstilling af forsøg, afprøvning af hypoteser, etc. Så vi kommer hurtigt til at en første skelnen måske vil fokusere på forskerens *formål* og *motivation*, og her er der nok en forskel at spore i eksemplerne: BOHRS forskning var situeret i en grundvidenskabelig diskussion blandt fysikere om atomare processer, og noget af teorien inden for kvantemekanik stred lodret imod almindelige sanseerfaringer og måtte søges fortolket og forklaret. På den anden side var EDISONS formål primært som forretningsmand og entreprenør, og hans opfindelser skulle kommercialiseres for at gøre gavn. Midt imellem står så PASTEURS målrettede forskning på at udvikle medicinsk behandling og forebygge smitte, som var rodfæstet i en grundvidenskabelig uddannelse og baggrund. På den måde var PASTEURS motivation nok netop dobbelt: Han ønskede både at opdage ny viden og sætte den i spil for samfundets bedste. Man kan spekulere en del i, hvad der mon var vigtigst for PASTEUR, og man kan notere sig, at han — ligesom fx opfinderne af insulinet — ikke drev direkte personlig økonomisk gevinst ud af det, men ønskede at stille det til rådighed for 'alle' som en form for filantropi (godgørenhed). 130  
135  
140

**Rækkefølge.** For det andet viser eksemplerne, at selvom der overordnet er tale om forskning, der munder i produkter, så er den tidslige adskillelse mellem grundvidenskabelig og anvendt vidensproduktion ikke så skarp, som vi kunne blive ledt til at formode. Selv EDISONS opfindelser krævede også produktion af ny viden for at kunne omsættes til kommersielle successer, så det er forkert at hævde, at de to opdagelser og opfindelser kan separeres tidsligt. Denne indsigt har også nogle konsekvenser for vores opfattelse af de institutioner, der producerer viden, som er nært sammenfaldende med fremkomsten af mode-2-videnskab: For selvom universiteter og private firmaer kunne se ud til hver for sig at være indbegrebet af hhv. grundforskning og målrettet forskning, så er grænserne mere flydende, både fordi viden flytter sig nemt over disse grænser, og fordi selve grænserne i mode-2-videnskab er blevet erstattet af et integreret netværk. Så hvis man fastholder en klar tidslig adskillelse af processerne, kan man nemt komme til at overse de innovative og adaptive side af teknologisk forskning. 145  
150  
155

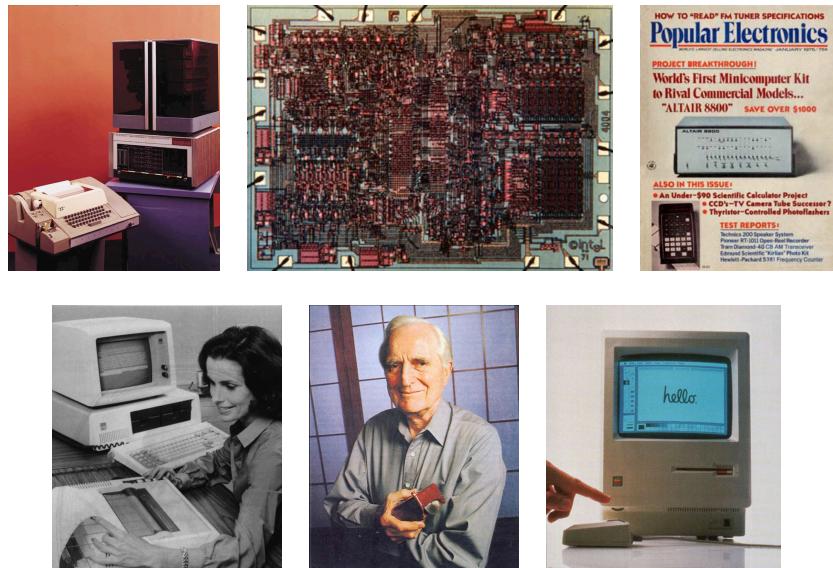
**Afhængighed.** Den tidslige rækkefølge er blot en af de måder, hvorpå man som del af den lineære model antog, at grundforskningen *kom før* den anvendelsesorienterede forskning. Og der er jo ingen tvivl om, at megen højteknologisk innovation bygger ovenpå naturvidenskabelig grundforskning, som er en uundværlig forudsætning, både hvad angår resultaterne men også hvad angår metoder og procedurer. Man kan sige, at naturvidenskab er en ressource for teknologisk forskning. Men omvendt leder teknologisk innovation også til nyt, billigere og mere præcist apparatur og nye teknikker, som bringer naturvidenskabelig forskning fremad. Og disse teknologiske forbedringer er *afgørende* for videnskabelige gennembrud, hvilket viser, at der ikke kan være tale om en en-vejs afhængighed mellem grundvidenskab og teknologi, men at der nærmere er tale om en *symbiotisk* relation, hvor begge dele er afhængige af og bidrager til hinanden. Og teknologiske gennembrud giver ofte anledning til ny grundforskning både for at forstå selve teknologien bedre og for at udnytte dens nye muligheder. 160  
165

**Epistemologi.** Disse observationer fører alle sammen til den konklusion, at teknologisk forskning, som er mere mål- og handlingsrettet end grundvidenskabelig forskning, 170

har *sin egen vidensform*, som på nogle punkter ligner, men ikke er sammenfaldende med, den grundvidenskabelige vidensform. Noget firkantet kan man sige, at grundvidenskaberne beskæftiger sig med at svare på spørgsmål om 'hvad' der er tilfældet, og 'hvorfor' det er tilfældet ved at bygge systematiske teorier. Modsat dette står så de teknologiske videnskaber, der beskæftiger sig med, 'hvordan' man kan få bestemte ønskelige ting til at ske. Og dertil hører en forskel i de kriterier, vi lægger til grund for den opnåede viden: I grundvidenskaberne er vores viden altid foreløbig, men den skal kunne holdes op imod en virkelighed (være *testbar*) og forbedres, når vi finder fejl i vores teorier (*korrigerbar*). På den anden side er teknologisk viden også foreløbig, men den vurderes ikke ved at holde den op mod en virkelighed og sammenligne *sandhed* men derimod er den formet af forskellige interesser og muligheder, og dens ultimative kriterium er anvendelighed og plads i markedet. Så hvor grundvidenskab kan siges at være *sandhedssøgende*, så er teknologisk videnskab langt mere *pragmatisk*. Men igen må man ikke tro, at pragmatik er forbeholdt teknologisk videnskab — også grundforskning er i sagens natur underlagt det muliges kunst, men det er ikke selve det *erkendelsesmæssige formål* med den.

## 2.2 Teknologiske systemer

Blandt de allervigtigste teknologihistoriske indsigt er, at teknologi bedst forstås *systemisk* og ikke som en streng af konkrete teknologiske artefakter. Denne kompakte sætning indeholder mange lag, så den skal udfoldes, og det vil vi gøre ved at betragte den personlige computer, PC'en, som en teknologisk innovation, vi ønsker at forstå. Den overordnede pointe er, at vi ikke kan forstå fx PC'ens udvikling alene ved at betragte de teknologiske 'dimser' (artefakter), der gik forud for den, men at vi er nødt til at betragte et større system eller netværk af teknologier og andre faktorer, som PC'en indgår i. Dette er den ene betydning af 'systemisk', og selvom den ikke synes synderligt kontroversiel, er den i sig selv et argument *imod teknologisk determinisme*, dvs. det synspunkt, at teknologien har en retning og følger noget, der ligner lovmæssigheder. En af de mest udbredte sådanne lovmæssigheder er den såkaldte *Moores lov*, der siger at processorkraften stiger eksponentielt over tid (se Ceruzzi, 2005). En ny teknologi som PC'en var selvfølgelig afhængig af andre nærtbeslægtede teknologier som integrerede chips til processoren og CRT-skærme til monitoren men også af mere generelle teknologier som pengeøkonomi og masseproduktion til at fremstille dem og el og (senere) telefoni til at drive maskinen og koble dem sammen. Alle disse teknologier indgik i at forme PC'en, ikke som lovmæssigheder og determinisme, men som valg og ressourcer. Den anden betydning af at forstå teknologi *systemisk* ligger i at påpege, at netværket kan indoptage lokale justeringer, og at alle forbundne hjørner i netværket i principippet bliver berørt af alle påvirkninger. Altså vil man se, at de fleste teknologier er *konservative opfindelser*, dvs. tager udgangspunkt i kendte teknologier, foretager inkrementelle forbedringer og tilpasser dem til nye formål. Og selv, når der en gang imellem er tale om *radikale opfindelser*, dvs. virkelig nye opfindelser, så vil deres ibrugtagen være betinget af, at der findes eller skabes teknologiske systemer, som de kan indpasses i.



Figur 3: Illustration af udvalgte aspekter af PC’ens tilblivelse og formning.

Teknologihistorikeren THOMAS HUGHES (1923–2014) har formuleret ideen om teknologiske systemer meget kortfattet:

Technological systems contain messy, complex, problem-solving components.  
They are both socially constructed and society shaping. (Hughes, 2012, s. 45)

215

Og han forklarer selve systemets opbygning:

Among the components in technological systems are physical artifacts, such as the turbogenerators, transformers, and transmission lines in electric light and power systems. Technological systems also include organizations, such as manufacturing firms, utility companies, and investment banks, and they incorporate components usually labeled scientific, such as books, articles, and university teaching and research programs. Legislative artifacts, such as regulatory laws, can also be part of technological systems. Because they are socially constructed and adapted in order to function in systems, natural resources, such as coal mines, also qualify as system artifacts. (Hughes, 2012, s. 45)

220

225

Så et teknologisk system er altså heterogent i den forstand, at elementerne kan have meget forskellig struktur og størrelse, og de komplekse komponenter er problemløsere af forskellig slags:

- fysiske artefakter som apparater og installationer,
- organisationer som firmaer, forsyningssystemer og finansieringskilder,
- videnskabelige elementer som artikler med viden og universiteter med forskere og studerende,
- juridiske og regulerende komponenter, som love og traktater, og
- naturressourcer kan også udgøre komponenter af teknologiske systemer.

230

235 For at udfolde og konkretisere dette, kan vi betragte PC'ens historie, som er et komplekst, men også meget vigtigt, stykke moderne teknologihistorie (se fx Freiberger og Swaine, 2000). Den moderne PCs udseende og funktion er selvfølgelig indlejret i en tradition fra tidligere computere, men også i en kontekst af store amerikanske virksomheder, automatisering af kontorarbejde, forestillinger om den virkelig *personlige* hjemmecomputer og meget andet. Figur 3 repræsenterer 6 forskellige dele af det netværk, som gjorde PC'en mulig.

- 240
1. Det første billede er af en såkaldt 'minicomputer', PDP-8, produceret af *Digital Equipment Corporation (DEC)* fra 1965. DEC var en konkurrent til den ellers totalt markedsdominerende computerfabrikant IBM, og PDP-8 var en kommercielt succesfuld minicomputer, som fik stor udbredelse i virksomheder og på universiteter.
  2. Det næste billede viser det integrerede kredsløb (eng: *integrated circuit, IC*) Intel 4004, som det dengang beskedne firma Intel udviklede i 1971. IC'en repræsenterede et stort skridt i at gøre computere mindre, ligesom dens forgænger, transistoren, der i sin tid havde erstattet radiorøret.

245 Disse to første elementer lægger den væsentligste *rent elektroniske* grund for PC'en, og man kunne formode, at udviklingen af en veldefineret og næsten pre-determineret PC nu fulgte. Men der hører flere aspekter til før PC'en kommer til at ligne sig selv og indtage sine funktioner (man noterer fx at billedet af PDP-8 ikke viser et keyboard).

- 255
3. Det tredje billede viser forsiden af magasinet *Popular Electronics*, der i 1975 solgte computeren Altair 8800 som samlesæt til entusiastiske læsere. Dermed kom computeren ud i de første amerikanske hjem, og igennem hobbyforeninger blev der skabt både et behov og en mængde software til hjemmebrug.
  4. Men det fjerde billede viser PC'ens første egentlige marked som understøttelse af kontorfunktioner og typisk betjent af sekretærer. Billedet viser også den oprindelige IBM PC XT, som blev introduceret af IBM i 1983 med disk-drev, monitor, keyboard og printer.

260 IBMs PC XT blev lanceret som en ny arkitektur og konfiguration, som skulle kunne stå på (eller under) skrivebordet, efter at IBM op igennem 1970'erne havde haft stor succes med deres IBM/360 serie, der dominerede markedet. Kort efter annonceringen blev IBM udfordret af Compaq, som lykkedes med at reverse-engineer IBMs BIOS og samtidig introducerede en bærbar version af PC'en, der var kompatibel med IBMs. Men for at vi for alvor kan genkende den moderne personlige computer, mangler både noget i hardware og i software.

- 265
5. Det femte billede viser ingenieren DOUGLAS ENGELBART (1925–2013), som er ansvarlig for en række vigtige opfindelse, heriblandt computermusen, som han udviklede mens han arbejdede på forskningslaboratoriet *SRI International*. SRI International var en selvejende institution, men fik store bevillinger både fra Stanford University og fra DARPA, som hører under det amerikanske forsvar.
  6. Det sidste billede viser en Apple Macintosh, som blev lanceret i 1984 som den første personlige computer med et grafisk interface og en mus. Apples maskiner var baseret på en familie af processorer fra Motorola, mens IBM PC'en var baseret på Intel chips. Igennem 1990'erne var de to brands store konkurrenter om markedet for personlige computere, og deres maskiner var ikke kompatible.

Denne ekskurs om PC’ens teknologi viser, at den moderne PC er resultat af et samspil mellem bl.a. internt teknologiske (elektroniske) gennembrud, udviklinger i markedet, hvor nye opgaver skal formuleres og opdyrkes, og standarder for grænseflader og interfaces, hardware og software. Nogle af interessenterne er private firmaer som DEC, IBM, Compaq og Apple, men også akademiske forskningsinstitutioner og militære sponsorater har påvirket historien. 280

Dermed er PC’ens tilblivelse og udformning et eksempel på den systemiske teknologi-forståelse, hvor en lang række valg og kontingente (ikke-nødvendige) udviklinger førte til en teknologi, der i dag er definerende for store dele af det moderne liv. Sådanne teknologier, der dårligt kan forestilles anderledes, kaldes nogle gang *definerende teknologier*, og de indfanger ofte store dele af vores verdenssyn. Og dette er ikke et særtilfælde for PC’en; det samme gør sig gældende for alle relevante teknologier fra elnetværk, cykler, medier, og mobiltelefoner (se fx Nielsen m.fl., 2015). 285

## 2.3 Verdenssyn og datalogiens teknologiske paradigme

290

I dagligsproget taler vi nogle gange om et ’paradigme’, som er en betegnelse, der omfatter et slags *verdenssyn* og er den horisont, som man ikke kan se bort fra. Dette begreb blev indført i videnskabsteorien med stor effekt af videnskabshistorikeren og -filosoffen THOMAS KUHN (1922–1996), som i sin bog *The Structure of Scientific Revolutions* benyttede det til at indfange videnskabelige gruppens fælles verdenssyn. KUHNS teori er mere omfattende, og vi skal vende tilbage til den i kapitel 3, hvor vi skal se nærmere på hans begreb om videnskabelige revolutioner. 295

Man kan godt betragte KUHNS videnskabsteori som en tilføjelse af sociologiske og historiske betragtninger til KARL POPPERS (1902–1994) falsifikationisme. Som vi så i kapitel 1, førte POPPERS hypotetisk-deduktive metode til et problem, kaldet *Duhem-Quine-tesen*, 300 når man ønskede at udpege den falsificerede hypoteze, for ’skylden’ for en falsifikation kunne overføres til en vilkårlig hypoteze i det netværk, der udgjorde ens teori. IMRE LAKATOS (1922–1974) forsøgte at svare på dette problem ved at udnævne visse hypoteser til et forskningsprograms ’hårde kerne’, men LAKATOS gav ikke nogen klar forklaring på, hvordan denne udvælgelse skulle finde sted. Og det er her, man kan se KUHNS baggrund i videnskabshistorien sætte ind: KUHN foreslog, at noget, vi også kunne kalde ’tradition’ og ’sammenhæng’ inden for et videnskabeligt felt ville spille en vigtig rolle i videnskabens udvikling. Og hans argument for at gå i denne retning var, at han ønskede en videnskabsteori, der som minimum var inspireret af faktisk videnskabshistorie og passede overordnet med videnskabelig praksis. 310

For KUHN indfanger et videnskabeligt paradigme et *fælles verdenssyn*, som deles af *en gruppe videnskabsfolk*. Det er altså noget, som konstitueres af en social gruppe, som på en eller anden måde udgør en identificerbar enhed. Sådan et videnskabeligt paradigme er, når KUHN skulle forklare det (Kuhn, 1962; Kuhn, 1970), instantieret i en *disciplinær matrix*, som bl.a. omfatter og udmøntes i 315

- en række „symboliske generaliseringer“, dvs. basale udsagn, som ligner naturlove, men egentlig (også) er *definitioner*,
- visse „metafysiske dele af paradigmet“, som er delte overbevisninger, som gruppen er forpligtet på, herunder *metaforer* og *analogier*,
- et sæt af „værdier“, som er det, der binder større videnskabelige samfund sammen, fx værdier om forudsigelsers vigtighed, og 320

- en række (paradigmatiske) „eksemplarer“ i form af problemer, løsninger, klassiske værker, lærebøger etc., som er den måde, hvorved nye forskere socialiserer ind i paradigmet.

325 På den måde opstiller paradigmet standarden for den videnskab, der udøves under det, herunder de fundamentale lovmæssigheder og teoretiske sammenhænge, som udøverne er forpligtede på, nogle typiske måder at anvende disse love og sammenhænge på, nogle overordnede, metafysiske principper, som styrer arbejdet, og nogle meget generelle metodologiske anvisninger.

330 Med til KUHNS teori hører helt centralt også en kontroversiel model for videnskabens udvikling (se kapitel 3), men i denne første omgang er det tilstrækkeligt at notere den deskriptive del af KUHNS sociale karakterisering af videnskaben.

## 2.4 Vidensproduktionen i datalogi og informationsteknologi

Den foregående diskussion har vist, at videnskabsbegrebet har visse oplagte og interessante overlap med datalogi-feltet: Der er faktisk store dele af datalogien, der falder ind under videnskabsbegrebet, som det er defineret af HELGE KRAGH. Datalogisk viden er ofte, og især når den er produceret på akademiske institutioner, faktisk offentlig, fejlbart, korrigerbart og testbar, sådan som KRAGH opstætter som betingelser. Dette er ikke noget tilfælde, for den akademiske videnskab er traditionelt udformet efter CUDOS, og KRAGHS definition ligger tæt op ad disse.

340 Det er imidlertid ikke let og generelt at afgøre, om informationsteknologien og datalogien falder ind under *formalvidenskab* eller *realvidenskab*. Store dele af datalogien er tæt forbundet med matematik og er ligesom matematikken primært *formalvidenskabelige*, men der findes også andre dele som fx HCI, som benytter *realvidenskabelige* metoder, og disse er også af central vigtighed for informationsteknologien som fag.

350 Det er oplagt, at PLACE passer bedre på hele den proprietære forskning og udvikling, som indgår i datalogien. Store softwarefirmaer som Microsoft og Google afsætter betragtelige midler til forskning, hvorfra en stor del er drevet af hensyn, der lever fint op til de i PLACE beskrevne. En stor del af denne viden vil have form af kildekode eller algoritmer, og firmaerne vil ofte være tilbageholdende med at dele disse forspring. Der er imidlertid også andre af deres forskningsprojekter, som søger at udnytte elementer fra CUDOS og bidrage til den offentlige viden, så man kan ikke sætte et helt skarpt skel mellem *privat* og *offentligt* finansieret forskning på denne måde, ligesom *mode-2-videnskab* også har fundet indpas på offentlige universiteter. Men der produceres også både grundvidenskabelig og teknologisk viden helt uden for disse strukturer. Det foregår af idealistiske grunde og i små græsrodsbevægelser, hvis motivation er at forandre verden til det bedre, enten for sig selv eller for større dele af menneskeheden. Et oplagt eksempel fra datalogiens verden vil være *Open Source Software*-miljøet, så det lader sig ikke gøre at indplacere datalogi og informationsteknologi entydigt i nogen af de klasser, vi har præsenteret. I stedet kan vi bruge de forskellige karakteriseringer til at opnå et reflekteret billede af faget, som kan belyse de komplekse videnskabsteoretiske problemstillinger, vi skal diskutere i de følgende kapitler.

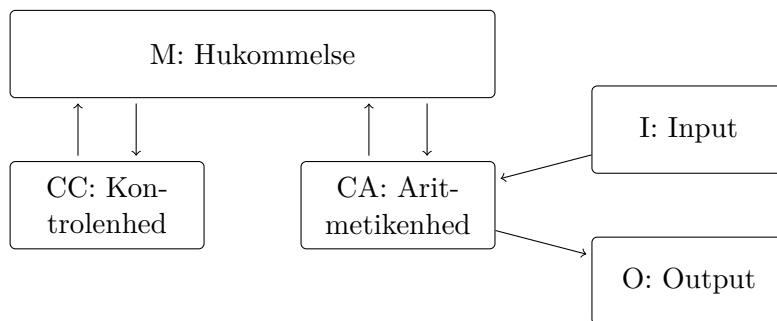
365 Ligesom andre grene af videnskaben omfatter datalogien en række centrale antagelser, som definerer discipliner og underdiscipliner. Blandt de centrale grundantagelser, som evt. kunne komme på tale som kandidater til datalogiske paradigmer, bemærker vi fire, hvoraf vi her skal gå lidt dybere ind i den første:

1. Centrale antagelser om computerens arkitektur
2. Centrale antagelser om beregningers kompleksitet
3. Teorier om metoder for softwareudvikling
4. Teorier om programmeringssprog

370

Og en af de bedste kandidater til et datalogisk paradigme er den såkaldte **von Neumann-arkitektur**, som kan siges at udgøre et teknologisk paradigme for (de allerfleste) moderne computere. Denne arkitektur er opkaldt efter JOHN VON NEUMANN (1903–1957), som beskrev den i rapporten *First Draft of a Report on the EDVAC*, som lagde grunden for den første såkaldte „stored program computer“, hvor hukommelsen ikke blot indeholdt data men også det program, der skulle afvikles. Denne nye ide adskilte EDVAC fra den tidligere ENIAC og varslede den moderne computerarkitektur.

375



Figur 4: Skematisk fremstilling af VON NEUMANNS arkitektur for EDVAC, baseret på Neumann (1945).

På den måde var VON NEUMANNS arkitektur en generel model for realiseringen af en computer i termer af en programmerbar maskine, hvor program og data er sidestillet, som omfatter seks centrale enheder, nemlig en central aritmetisk enhed (CA), en central kontrolenhed (CC), hukommelse (M), to såkaldte 'organer' for input (I) og output (O) og en langsom ekstern hukommelse (R). Sammen med interne registre vil CC og CA i dag udgøre det, vi kalder en CPU (eng: *central processing unit*). De tre centrale enheder CA, CC og M skulle kommunikere langs en 'bus', og det var afgørende, at både programinstruktioner og data blev opbevaret i hukommelsen (M). Dermed blev det også teknisk muligt at skrive software, der kunne oversætte instruktioner til maskinsprog, hvilket er essentielt for både operativsystemer og programmeringssprog. Og VON NEUMANN foreslog også et sæt instruktioner, som senere blev udvidet med flere, ligesom han senere opdyrkede selve programmeringen af maskinen i form af flow-diagrammer og pseudokode.

380

385

390

Man kan anse von Neumann-arkitekturen for datalogiens *teknologiske* paradigme, idet denne arkitektur siden umiddelbart efter 1945 er kommet til at dominere alle succesfulde computerkonstruktioner. Der er intet uomgængeligt ved denne arkitektur, men den er ved sin effektivitet og fleksibilitet kommet til at danne grundlag både for teoretiske udviklinger og for teknologiske bestræbelser på fx at gøre CPU'en kraftigere, udvikle det mest optimale instruktionssæt, eller finde måder at lagde hukommelsen på, så nogle dele af den kan tilgås endnu hurtigere, fx igennem **caching**.

395

## Litteratur

- Bush, Vannevar (1945). *Science: The Endless Frontier*. A Report to the President by Vannevar Bush, Director of the Office of Scientific Research and Development, July 1945. Washington: United States Government Printing Office.
- Ceruzzi, Paul E. (2005). „Moore’s Law and Technological Determinism: Reflections on the History of Technology“. *Technology and Culture*, bd. 46, nr. 3, s. 584–593. DOI: 10.1353/tech.2005.0116.
- Freiberger, Paul og Michael Swaine (2000). *Fire in the Valley. The Making of the Personal Computer*. 2. udg. New York etc.: McGraw-Hill.
- Hughes, Thomas P. (2012). „The Evolution of Large Technological Systems“. I: *The Social Construction of Technological Systems. New Directions in the Sociology and History of Technology*. Red. af Wiebe E. Bijker, Thomas P. Hughes og Trevor Pinch. The MIT Press, s. 45–76.
- Kline, Stephen J. og Nathan Rosenberg (2009). „An Overview of Innovation“. I: *Studies on Science and the Innovation Process. Selected Works by Nathan Rosenberg*. Red. af Nathan Rosenberg. World Scientific Publishing. Kap. 9, s. 173–203.
- Kuhn, Thomas S. (1962). *The Structure of Scientific Revolutions*. Chicago og London: The University of Chicago Press.
- (1970). „Postscript: 1969“. I: *The Structure of Scientific Revolutions*. 2. udg. Chicago & London: The University of Chicago Press, s. 174–210.
- (1996). *The Structure of Scientific Revolutions*. 3. udg. Chicago og London: The University of Chicago Press.
- Neumann, John von (jun. 1945). *First Draft of a Report on the EDVAC. Contract No. W-670-ORD-4926 Between the United States Army Ordnance Department and the University of Pennsylvania*. Moore School of Electrical Engineering, University of Pennsylvania.
- Nielsen, Henry m.fl. (2015). *Forandringens vinde. Nye teknologihistorier*. København: Praxis: Nyt Teknisk Forlag.
- Stokes, Donald E. (1997). *Pasteur’s Quadrant. Basic Science and Technological Innovation*. Washington, D.C.: Brookings Institution Press.

# Kapitel 3:

## Fundamentale modeller og datalogiens teoretiske paradigme

Henrik Kragh Sørensen

Mikkel Willum Johansen

22. april 2022

### Indhold

5

3.1 Datalogiens matematiske rødder . . . . .	1
3.2 Turings maskiner og Hilberts program . . . . .	3
3.3 Algoritmebegreb(er) . . . . .	8
3.4 Computere og matematiske beviser . . . . .	13
3.5 Videnskabelige revolutioner . . . . .	17

10

### Litteratur

20

### 3.1 Datalogiens matematiske rødder

Selvom datalogi og matematik i dag er forskellige, specialiserede fagligheder med separate akademiske institutioner, er matematikken på mange måder datalogiens nærmeste nabo i det akademiske landskab. Det gælder historisk, biografisk og institutionelt, og det gælder også for en lang række af de metoder, som udgør centrale dele af datalogiens videnskabelige værktøjskasse. I dette kapitel vil vi undersøge den nære forbindelse mellem datalogi og matematik ved at analysere nogle *symbiotiske* forhold, hvor de to fag lapper over og gensidigt bidrager til hinandens videnskabelige undersøgelser. I dette kapitel skal vi se på matematikkens rolle i at udvikle og formulere det, vi vil kalde *datalogiens teoretiske paradigm* igennem at diskutere, hvordan datalogiens fundamentale model for beregnelighed har et matematisk og matematikfilosofisk ophav. I kapitel ?? kommer vi til at diskutere, hvordan formelle metoder spiller en vigtig rolle i programmeringssprog og sikkerhedsspørgsmål omkring software.

15

Datalogiens og matematikkens historie udviser et bemærkelsesværdigt dobbelt parløb: På den ene side kan man se udviklingen inden for hardware og software som (i hvert fald delvist) motiveret af behov for beregningskraft til matematiske og matematisk fysiske problemer inden for fx tabulering, logistik, kodebrydning og simulering. På den måde kan man sige, at computeren — og læren om den — er en forlængelse af anvendt matematik. På den anden side observerer man, at store dele af datalogien (i modsætning udvikling af den fysiske computer) handler om *abstrakte* beregninger, som har taget deres definition og formulering i matematiske termer. På den måde kan man sige, at datalogien — i hvert fald store teoretiske dele af den — har et matematisk grundlag og behandler problemer, der kan siges at ligge inden for matematisk logik og diskret matematik.

20

30

### Faktabox 3.1 (Historiske eksempler på symbiosen)

**Tabulering:** Allerede i perioden omkring Den franske Revolution — længe før de teknologiske gennembrud, vi associerer med computere — iværksattes store tabuleringsprojekter af fx matematiske funktioner, hvori to hold af *menneskelige computere* fulgte nøje beskrevne procedurer i en strengt arbejdsdelt lineær proces (se Campbell-Kelly m.fl., 2003). Siden sammenholdtes resultaterne af de to beregninger som kvalitetscheck. Undervejs benyttes andre tidlige tabuleringer.

**Logistik:** Under Den anden Verdenskrig investerede den amerikanske flåde betydelige midler i udviklingen af effektive logistiske løsninger. Via et bemærkelsesværdigt møde mellem matematikerne HAROLD KUHN (1925–2014) og JOHN VON NEUMANN (1903–1957) førte dette både til grundlæggelsen af et nyt teoretisk felt, *lineær programmering*, og til praktiske løsninger og hardware-udvikling. (Hoff Kjeldsen, 2000)

**Kodebrydning:** På Bletchley Park nord for London lykkedes det fra 1942 for britiske kodebrydere at knække aksemagternes kryptering, særligt Enigma- og Lorentz-koderne. Der indgik en masse logisk tænkning og automatiseret afprøvning, men man kan med rette diskutere, hvorvidt maskinen Colossus var en virkelig *computer* (i moderne forstand). Og projekt Ultra var et tidligt operationsanalytisk gennembrud, idet man måtte behandle de vundne oplysninger operationelt hen-sigtsmæssigt. (Kahn, 1996)

**Simulering:** I *Manhattan Projektet* blev der gjort banebrydende brug af ENIAC-computeren, der med en tilsvarende vis ret kan siges at være en af de allerførste (moderne) computere. Den blev brugt til omfattende simuleringer af udløsningen af atomare kædereaktioner til brug for atombomben. Denne simulering var nødvendig for at udskifte en ellers umulig (eller i hvert fald uønskelig) trial-and-error med et præcist estimat, som imidlertid ikke kunne beregnes analytisk. (Haigh, Priestley og Rope, 2014)

De fire listede symbioser har mange historiske fortilfælde, men det er ikke tilfældigt, at symbiosen bliver så meget stærkere omkring Den anden Verdenskrig, hvor teknologiske udviklinger, et akut behov og finansiering, og et bemærkelsesværdigt personligt sammenfald i VON NEUMANN var med til at krystallisere symbiosen, så den efterfølgende blev et vigtigt videnskabeligt og politisk agenda.

<sup>35</sup> Denne dobbelthed møder nogle udfordringer, som har med forholdet mellem real- og formalvidenskab som allerede behandlet i kapitel 1: For så vidt matematik handler om den åbne virkelighed, er dens deduktive slutningsformer ikke gyldige, og for så vidt dens slutninger er absolut sikre, handler de om *abstrakte entiteter* og højest approksimativt om virkelige fænomener. Derfor er et videnskabsteoretisk vigtigt og interessant at dykke længere ind i forholdet mellem datalogien og en af dens allernærmeste nabodiscipliner, nemlig matematikken.

## 3.2 Turings maskiner og Hilberts program

I datalogiens historie er engelske ALAN TURING (1912–1954) en central skikkelse: Han formulerede en fundamental model for beregnelighed i 1930’erne, han var involveret i kodebrydningen på Bletchley Park, og efter krigen udarbejdede han et forslag til en ny computer kaldet *ACE*, i 1950’erne tænkte han dybt over menneskelig og mekanisk intelligens og formulerede en test deraf, og datalogiens fornemste pris er i dag opkaldt efter ham.

45

### Hilberts program

For at forstå TURINGS centrale teoretiske bidrag er det nødvendigt at forstå hans udgangspunkt, som handlede om tekniske og filosofiske diskussioner omkring matematikkens grundlag, der udspandt sig i begyndelsen af 1900-tallet (se også Johansen og Sørensen, 2014, kap. 5). Et af de fremherskende synspunkter var, at matematik skulle og kunne opnå absolut sikkerhed som et formaliseret sprog med klare, logiske deduktioner. Denne position, som byggede videre på et egentligt *logicistisk program* udviklet af bl.a. GOTTLÖB FREGE (1848–1925) og BERTRAND RUSSELL (1872–1970), blev kraftfuldt fremført af den ledende tyske matematiker DAVID HILBERT (1862–1943). HILBERTS *Program* tog udgangspunkt i sikkerheden af mentale operationer i *den endelige aritmetik*. Derved kombinerede HILBERT dels IMMANUEL KANTS (1724–1804) anskuelsesform om tid med en opfattelse af, at atomare operationer i den endelige aritmetik var sikre, nærmest mekaniske. Ovenpå dette metafysiske grundlag byggede HILBERT en programmatisk vision for matematikkens sikkerhed, der var blevet draget i tvivl i slutningen af 1800-tallet ved opdagelsen af forskellige ikke-intuititive objekter og ikke-stringente slutionsformer:

50

1. Matematikkens grundlag skulle sikres gennem *formalisering* og *aksiomatisering*, dvs. at HILBERTS ide var at opstille syntaktiske aksiomssystemer, hvori al mening var givet implicit af aksiomerne.
2. Pointen var at opstille formelle systemer og bevise deres *uafhængighed*, *konsistens* og *fuldstændighed* igennem *endelige metoder*. Dermed ville man have bevist, at disse centrale aksiomssystemer ikke kan indeholde modstrid.
3. Derefter kunne man tilføje *ideale elementer* igennem *konservative udvidelser*, om hvilke man kunne bevise, at de ikke *tilføjede* modstrid (se også Hilbert, 1983). Dermed ville man have sikret (kunne sikre) hele matematikken, særligt den uendelige del fra mængdelæreren og differentialregningen.

65

70

I en berømt forelæsning i 1931 formulerede og beviste KURT GÖDEL (1906–1978) sine *ufuldständighedssætninger*, hvoraf den mest specifikke beviste at i ethvert aksiomssystem, som er rigt nok til at indeholde de naturlige tal, kan man bevise, at hvis det er konsistent, så er det ufuldstændigt. Dermed led HILBERTS generelle program skibbrud, og matematikerne har siden måttet ty til andre måder at legitimere deres grundlag på (fx gennem såkaldt *pragmatisk formalisme*).

75

analytisk	udsagn, hvor udsigelsen er indeholdt i det, der udsiges noget om
syntetisk	udsagn, der ikke er analytisk
syntaktisk	udsagn i kraft af deres form
semantisk	udsagn i kraft af deres mening

Figur 1: Centrale videnskabsteoretiske begrebspar (modsætninger).

## 80 Turings maskiner og mekanisk beregnelighed

Til HILBERTS program hørte også spørgsmålet om *afgørbarhed*: Kan man effektivt (evt. mekanisk) afgøre om en sætning er sand eller ej? Og det var dette problem, TURING løste i sin vigtige artikel „On Computable Numbers, with an Application to the Entscheidungsproblem“ fra 1936. I artiklen formulerede TURING en *model* for, hvad det vil sige at beregne et tal. Dette kan synes som en indskrænkning i forhold til HILBERTS oprindelige spørgsmål om at afgøre *bevisbarheden* af en proposition, men ved at udnytte, at de formelle sprog er endeligt frembragte og at aksiomssystemet skal indeholde aritmetikken (samme trick, som GÖDEL havde benyttet), kan man se, at det at afklare beregneligheden af tal er det samme som at afklare bevisbarheden af propositioner.

90 Den model af en maskine, som TURING formulerede, var karakteriseret ved følgende egenskaber:

1. Den havde endeligt mange tilstande  $q_1, \dots, q_R$ , som han kaldte „*m*-konfigurationer“.
2. Den havde et „bånd“ inddelt i „kvadrater“, som hver kan indeholde et „symbol“ (fra et endeligt alfabet).
- 95 3. Til ethvert tidspunkt er der kun et kvadrat „i maskinen“ (det „scannede kvadrat“), fx det  $r$ ’te kvadrat med symbolen  $\mathfrak{S}(r)$ .
4. Maskinen er kun „direkte opmærksom“ på det scannede symbol, men ved at ændre *m*-konfigurationen kan maskinen faktisk huske nogle af de symboler, den har „set“.
5. Parret  $(q_n, \mathfrak{S}(r))$  kaldte han for „konfigurationen“, og det bestemmer maskinens opførsel:
  - Maskinen kan skrive et symbol i et tomt kvadrat,
  - Maskinen kan slette et symbol i et kvadrat,
  - Maskinen kan ændre det kvadrat, der scannes, men kun ved at flytte en plads til højre eller venstre,
  - Og maskinen kan desuden ændre sin *m*-konfiguration.

Ved at starte sådan en ’maskine’ ville det *tal*, som den skrev på sit bånd siges at være beregnet. Det var muligt, at maskinen gik i et loop og måske blev ved med at skrive tegn på båndet; så betegnede TURING den (lidt kontraintuitivt, se nedenfor) „*cirkelfri*“.

Det var TURINGS formening, at modellen ovenfor indfangede præcist, hvad det vil sige at *beregne* et tal igennem en proces, der kunne siges at være mekanisk eller automatisk, men uden at han på nogen måde havde en fysisk realisering af maskinen i tankerne:

It is my contention that these operations include all those which are used in the computation of a number. (Turing, 1936, p. 232)

Der ligge en grundantagelse, som også er blevet formuleret af andre nogenlunde samtidig, og som i dag går under betegnelsen *Church-Turing-tesen*: Alt hvad der kan siges at være beregneligt, kan beregnes af en maskine, som den TURING beskrev i 1936, hvilket TURING gjorde eksplisit i en artikel fra 1948 (se også Copeland, 2008). På det tidspunkt kaldte han

sine teoretiske maskiner, dvs. det vi i dag kalder Turing-maskiner, for „logical computing machines“:

120

It is found in practice that L.C.M.s [i.e. logical computing machines] can do anything that could be described as 'rule of thumb' or 'purely mechanical'.  
(Turing, 2004, p. 414)

På den måde hævdede TURING, at han havde indfanget indholdet af 'tankeløs handlen' i sine maskiner.

## Den universelle Turingsmaskine og Halting-problemet

125

Udover at have indfanget det centrale i *beregnelighed* tillod TURINGS model ham også at ræsonnere omkring beregninger ved at tilbyde et meta-niveau, ligesom HILBERTS formalistiske program også udgjorde en *metamatematik*. Særligt var TURING interesseret i, om han kunne afgøre, om en given maskine var *cirkelfri* eller ej. Og dertil udviklede og udnyttede han en anden central ide, nemlig at betragte maskiner (programmer) som strenge (data), 130 hvilket ledte ham til det, vi i dag kalder *den universelle Turing-maskine*.

TURING havde udviklet en systematisk måde at angive maskinerne på ved at liste konfigurationerne i tabeller, der ud fra tilstanden og symbolet på båndet angav handlingen i form af hvad der skulle skrives på båndet, hvordan hovedet skulle flytte sig, og hvilken tilstand, der skulle skiftes til. Dette tillod ham at hævde, at

135

A computable sequence  $\gamma$  is determined by a description of a machine which computes  $\gamma$ . (Turing, 1936, p. 239)

Og idet han kunne systematisere tabellerne, der beskriver maskiner, (til *standard descriptions*) og oversætte dem til tal (*description numbers*) fik han en afbildung fra beskrivelses-tallet  $n$  til maskinen  $\mathcal{M}(n)$ . Nu gælder det, at til enhver beregnelig sekvens hører mindst 140 et beskrivelsestal, men et beskrivelsestal kan højst svare til en sekvens. Derfor er mængden af beregnelige sekvenser tællelig.

Nu fik TURING den revolutionerende ide, at ved at indkode instruktionstabellen i tilstande, kunne man konstruere en *universel* maskine:

145

It is possible to invent a single machine which can be used to compute any computable sequence. (Turing, 1936, p. 241)

Hvis maskinen  $\mathcal{U}$  forsynes med et bånd med standardbeskrivelsen af en maskine  $\mathcal{M}$ , så vil  $\mathcal{U}$  beregne den samme sekvens som  $\mathcal{M}$ . Man kan altså sige, at TURINGS maskine  $\mathcal{U}$  emulerer opførslen af maskinen  $\mathcal{M}$ , som er dens input. Programmeringssprog, instruktionssæt og automater siges at være *Turing-komplette*, hvis de kan simulere en Turingmaskine. Deres 150 (beregningsmæssige) 'udtrykskraft' er altså den samme.

Denne ide satte TURING i stand til at ræsonnere omkring udtrykskraften af Turing-maskiner og i særdeleshed at løse det såkaldte *Haltingproblem*, der er forbundet med afgørbarhedstesen. TURING antog, at der findes en maskine  $\mathcal{Q}$  således, at hvis den gives en standardbeskrivelse af en vilkårlig anden maskine  $\mathcal{M}$ , så vil  $\mathcal{Q}$  markere „u“, hvis  $\mathcal{M}$  er „cirkulær“, og markere „s“, hvis  $\mathcal{M}$  er „cirkelfri“. Nu kombinerede han så  $\mathcal{Q}$  og  $\mathcal{U}$  og fik derved en maskine  $\mathcal{H}$ , der beregner  $\beta'$ , dvs. tallet hvis  $n$ 'te ciffer er  $\phi_n(n)$ , hvor  $\phi_n(m)$  er det  $m$ 'te ciffer i den  $n$ 'te beregnelige sekvens  $\alpha_n$ . Udførelsen af maskinen  $\mathcal{H}$  er opdelt i

160 sektioner. I den  $N$ ’te sektion tester  $\mathcal{Q}$  tallet  $N$ . Og hver sektion af beregningen er endelig, så  $\mathcal{H}$  er cirkelfri.

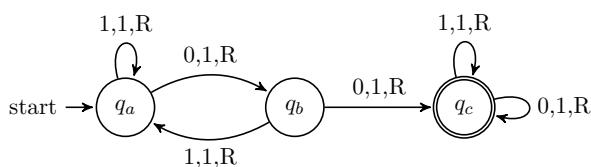
165 Lad så  $K$  være beskrivelsestallet for  $\mathcal{H}$ . Hvad skal  $\mathcal{H}$  stille op i den  $K$ ’te sektion af sin beregning? Eftersom  $K$  er beskrivelsestallet for  $\mathcal{H}$ , og  $\mathcal{H}$  er cirkelfri, så kan svaret ikke være „u“. Men svaret kan heller ikke være „s“: Hvis svaret skal være „s“, så skal maskinen beregne de første  $R(K - 1) + 1 = R(K)$  cifre beregnet af maskinen hørende til  $K$  og skrive det  $R(K)$ ’te ciffer som en del af den sekvens, som  $\mathcal{H}$  beregner. Derved opstår en regress, og det  $R(K)$ ’te ciffer vil aldrig blive fundet. Derfor er  $\mathcal{H}$  cirkulær. Altså har vi opnået en modstrid, og maskinen  $\mathcal{Q}$  er umulig.

170 Dette tekniske argument viser altså, at det ikke er muligt at have en generel Turingmaskine  $\mathcal{Q}$ , der afgør om en vilkårlig anden Turingmaskine er cirkelfri eller ej. Andre teoretiske 175 datalogger har arbejdet videre med TURINGs ideer og for eksempel indført en standardiseret notation for *endelige automater* (se figur 2). Og i 1952 lykkedes det datalogen STEPHEN COLE KLEENE (1909–1994) at formulere TURINGs resultat i termer af *terminerende* maskiner. Derved beviste KLEENE, at der ikke kan findes en generel maskine, der afgør om en vilkårlig maskine terminerer. Antag nemlig, at der findes en procedure (en Turingmaskine) **HALTS**( $P, I$ ) som giver **true** hvis  $P$  terminerer på input  $I$  og **false** ellers. Betragt så proceduren **Z**, som også er en Turingmaskine:

```
180    Z(string x) {
181        if (HALTS(x,x)) {
182            loop forever
183        } else {
184            halt
185        }
186    }
```

Nu må **Z**( $Z$ ) enten terminere eller ej. Men begge dele leder hurtigt til modstrid.

190 TURINGs resultat afgør *Entscheidungsproblemets* afkræftende: Der kan ikke gives en generel, effektiv (beregnelig) algoritme, der afgør, om en streng er en sætning i et givet formelt system. Ligesom GÖDELS ufuldstændighedssætninger, var TURINGs bevis bygget op omkring selvreference, og de deler også det forhold, at de begge er *forpligtende* for matematikere 195 og datalogger: Beviserne er interne og baseret på grundlæggende antagelser, så man kan ikke tvivle på konklusionerne uden at drage antagelserne i tvivl. Samtidig viser TURINGs (og KLEENES) bevis også vejen til mange andre uafgørbare problemer ved *reduktion*: Hvis  $X$  var afgørbart, så kan vi konstruere en afgørbar procedure til **HALTS**; det kan ikke lade sig 195 gøre, så  $X$  er også uafgørbart (se også Davis, 1982). Og det fører til problemkomplekser og kompleksitetsteori (se næste afsnit), herunder til en udviklet teori om såkaldte ’automater’ (se figur 2) og spørgsmålet om  $P = NP$ .



Figur 2: Moderne repræsentation af en *endelig automat* med alfabetet  $\{0, 1\}$ . Den er givet et bånd med en streng af tegn, og hvis den når til tilstanden  $q_c$  (markeret som en *halting-tilstand*), siger vi, at den har *genkendt* (eller accepteret) strengen. Hvis man fx står

i tilstanden  $q_a$  og læser 0 på båndet, så skriver man 1 på båndet, flytter hovedet til højre ( $R$ ) og skifter til tilstanden  $q_b$ .

## Hartmanis' paradigme

En af de dataloger, der medvirkede centralt til at udfolde TURINGs indsigt om beregne-  
lighed var den litauiske datalog JURIS HARTMANIS. Hans egen karriere er interessant, for  
etter Den anden Verdenskrig bragte den politiske situation ham ud på en omtumlet uddan-  
nelse, som han selv beskriver som en „random walk“ (Shustek, 2015). Men den endte med  
at føre ham til USA, hvor han fik en karriere i industrien, hvor han udviklede de ideer, der i  
1993 gav ham Turingprisen for hans (grund)forskning i kompleksitetsteori. I den forbindelse  
reflekterede HARTMANIS over sine egne bidrag og deres udspring (Hartmanis, 1994). Han  
beskriver, hvordan han var inspireret af TURINGs 1936-artikel og af CLAUDE SHANNONS  
(1916–2001) kommunikationsteori fra 1948. Sammen med RICHARD E. STEARNS ledte det  
ham til i 1965 at formulere det centrale spørgsmål:

Could there be quantitative laws that determine for each problem how much  
computing effort (work) is required for its solution and how to measure and  
determine it? (Hartmanis, 1994, p. 38)

200  
205

210

Og svaret på deres spørgsmål fandt de i TURINGs maskiner, der derved, efter at de havde  
modificeret dem til at tillade flere bånd, blev til deres *fundamentale modeller*:

To capture the quantitative behavior of the computing effort and to clas-  
sify computations by their intrinsic computational complexity, which we were  
seeking, we needed a robust computing model and an intuitively satisfying clas-  
sification of the complexity of problems. The Turing machine was ideally suited  
for the computer model, and we modified it to the multi-tape version. (Hart-  
manis, 1994, p. 38)

215  
220

225

Nu kunne problemer så klassificeres, og fx studerede de klassen  $C_{n^2} = \text{TIME}[n^2]$ , som er  
alle de problemer, hvis instanser af længde  $n$  kan løses i  $n^2$  skridt på en Turingmaskine med  
flere bånd. Derved fik man et abstrakt mål på kompleksiteten (målt på forskellige parametre  
som fx tid og plads) af beregninger, og man kunne efterhånden vise, at visse problemer var  
hårdere at løse end andre.

Foruden denne retro- og delvist introspektive forklaring bevægede HARTMANIS sig i  
sin pristale også ud i filosofiske refleksioner over datalogiens universalitet og relation til  
matematik:

The digital computer is a universal device and can perform in principle any  
computation (assuming the Church-Turing thesis, it captures all computations)  
and, in particular, any mathematical procedure in an axiomatized formal sys-  
tem. Thus in principle the full power of mathematical reasoning, which has been  
civilization's primary scientific tool, can be embodied in our computers from  
numerical computations and simulation of physical processes to symbolic com-  
putations and logical reasoning to theorem proving. (Hartmanis, 1994, p. 40)

230  
235

Man fornemmer, at HARTMANIS er, hvad Bolter (1986) har kaldt en „Turing's Man“: Han  
anskuer verden i termer af beregninger, og Turingmaskinen udgør den centrale indfangning

af, hvad det vil sige at beregne noget. Man ser nærmest, at HARTMANIS ophøjer Turingmaskinen til det centrale definerende paradigme, og denne fornemmelse bestyrkes af en endnu 240 bredere videnskabsfilosofisk betragtning fra samme tale:

Thinking about the previously mentioned (and other) theoretical work in computer science, one is led to the very clear conclusion that theories do not compete with each other for which better explains the fundamental nature of information. Nor are new theories developed to reconcile theory with experimental results that reveal unexplained anomalies or new, unexpected phenomena as 245 in physics. In computer science there is no history of critical experiments that decide between the validity of various theories, as there are in physical sciences.

The basic, underlying mathematical model of digital computing is not seriously challenged by theory or experiments. The ultimate limits of effective computing, imposed by the theory of computing, are well understood and accepted. There is a strong effort to define and prove the feasible limits of computation, 250 but even here the basic model of computation is not questioned. (Hartmanis, 1994, p. 40)

HARTMANIS står altså som eksponent for en grundlæggende position, at datalogi handler 255 om beregninger, og beregninger er, hvad der kan indfanges af Den universelle Turingmaskine. Denne position er bundet til den teoretiske datalogi, idet den udelukkende behandler abstrakte Turingmaskiner (med fx ubegrænsede bånd) og ikke tager hensyn til de konkrete kørselstider, men betragter problemers og algoritmers kompleksitet.

### 3.3 Algoritmebegreb(er)

260 En stor del af datalogi handler om *algoritmer*, men hvad dækker dette begreb egentlig over, og hvilke typer viden kan man have om dem? Det er spørgsmålene i dette afsnit.

I moderne sprogbrug fylder *algoritmer* sommetider meget i den offentlige debat: Facebooks timeline-algoritme eller Netflix's anbefalelsesalgortime kritiseres for eksempel for at føre til en identitets-bobble og et *confirmation bias*, hvor vi risikerer kun at blive præsenteret 265 for nyheder og informationer, som vores venner og bekendte allerede har syntes godt om (Pariser, 2011; Zuboff, 2019). Dette er et eksempel på en algoritme betragtet som et socio-teknisk object, hvor teknologien indgår i og påvirker sociale relationer. 'Algoritmen' er et objekt i verden, som nogen ejer og nogen bruger, og som påvirker sociale relationer og i sidste ende kan siges at forme virkeligheden (Seaver, 2017). Hvis man går skridtet videre 270 og tilføjer, at man også kan *kommunikere med* en sådan algoritme, træder den næsten ud og får karakter af et socialt objekt. Hvis man fx klikker systematisk (og meget!), vil man kunne påvirke 'algoritmens' fremtidige opførelse, og det er jo ikke så anderledes end at man kommunikerer med fx børn ved at belønne ønsket adfærd. Så i den forstand kan man faktisk godt tillægge algoritmen *agens* i den forstand, at det ikke bare er metaforisk, når vi siger, at 275 algoritmer *ser*, *vælger* eller *siger*, selvom algoritmer (forstået som teknologiske systemer) jo teknisk set ikke besidder sanser, bevidsthed eller sprog (se fx Thomas, Nafus og Sherman, 2018).

Der findes også et snævrere algoritme-begreb, som typisk er det, dataloger møder og arbejder med. Det har sidst udspring i mekaniserede udførselse af fx regning og matematisk 280 problemløsning, og dets historie kan spores tilbage til meget længe før computerens opfindelse. Fra den mesopotamiske kulturmødre (ca. 1800 f.v.t.) har vi overlevet en lang række

små lertavler, hvori er indprentet løsninger på matematiske opgaver, hvoraf vi vil fortolke nogle som løsninger af andengradsligninger (for mere om matematikhistorisk fortolkning, se Sørensen, 2016a). Et eksempel er lertavlen YBC 6967, som i tekstnær dansk oversættelse indeholder følgende tekst:

285

*Igibûm* overstiger *igûm* med 7. *Igûm* og *igibûm* hvad? Du: 7 hvormed *igibûm* overstiger *igûm*, til to bræk: 3,5. 3,5 sammen med 3,5 lad holde: 12,25. Til 12,25, som fremkommer for dig, 60 fladen tilføj: 72,25. Ligesiden af 72,25 hvad? 8,5. 8,5 og 8,5, dets modstykke, indtegn. 3,5, det som holder, fra den ene udriv, til den anden tilføj. Den ene bliver 12, den anden bliver 5. 12 er *igibûm*, 5 er *igûm*.

290

(Høyrup, 1998, s. 49–50)<sup>1</sup>

Dette ligner en 'opskrift' på at løse et bestemt problem, men der er nogle observationer, som vi hurtigt gør os: For det første ser det ud til, at der er to efterspurgte størrelser (*igibûm* og *igûm*), men kun en relation givet imellem dem. Imidlertid ved vi af andre grunde, som har med positionstalsystemet at gøre, at de to begreber er hinandens *reciprokke*, så *igibûm* ganget med *igûm* giver 60. For det andet noterer vi os ved at følge opskriften, at det ser ud til at den udregner størrelsen, der i algebraisk notation kan skrives som

$$\sqrt{\frac{7}{2} \times \frac{7}{2} + 60 \pm 3\frac{1}{2}}, \quad (1)$$

hvilket netop svarer til at løse ligningssystemet  $x - y = 7, xy = 60$ . Men den hjælpsomme algebraiske notation er ikke at finde i den mesopotamiske kilde og blev først opfundet 3000 år senere. Men at skrive en formel som (1) er også at angive en opskrift, og vi lærer i skolen at afkode og udføre den. For det tredje noterer vi også, at opskriften arbejder med helt konkrete tal; både forskellen mellem *igibûm* og *igûm* og alle andre afledte størrelser er givet i konkrete talværdier. Der er altså ikke et *variabelbegreb*, men der er spor af en teknik til præcist at henvise til tidlige skridt som „hvormed *igibûm* overstiger *igûm*“ eller „det som holder“. Opszriften er altså *konkret*, men har en mulighed for at kunne bringes i anvendelse også i andre situationer, selvom det er svært at forestille sig, hvordan den mesopotamiske skriver har lært sig, hvilke tal, der henviser til andre, og hvilke der ikke gør. Endelig, for det fjerde, er der i opszriften ikke nogen erklæring af, hvad den skal opnå eller hvorfor den virker. Der er dog spor af det i det geometriske sprog, hvor vi taler om at „brække“ og „tilføje“, som vi mener henviser til en bagvedliggende geometrisk konstruktion, som vi kender fra senere kulturer som at 'fuldende kvadratet'.

295

300

305

Vi kan nu sammenfatte eksemplet og fremdrage nogle aspekter, som vi finder relevante:

1. 'Opszriften' består af en rækkefølge af handlinger, som skal udføres. Hver handling er (i hvert fald efter tillæring) klar og har et veldefineret resultat. Opszriften kan derfor også ses at give et korrekt resultat.
2. I kilden er input helt konkret, men med et passende variabelbegreb kan man gøre opszriften anvendelig på en række beslægtede problemer (andre talværdier), og den bliver mere *generel* men har stadig et anvendelsesdomæne, selvom det ikke er angivet.
3. Med et moderne, matematisk/datalogisk syn på algoritmebegrebet ville vi sige, at kilden svarer til en *afvikling* (*kørsel*) af en (bagvedliggende, skjult) mere generel opskrift,

310

<sup>1</sup>Vi har her, ligesom i Høyrup (1998), oversat selve taltegnene fra kileskrift til det moderne talsystem, og desuden har vi konverteret fra det mesopotamiske 60-tals positionssystem til moderne decimaltal.

315 som vi kunne kalde et *program* eller *kode*, og som også er indfanget i (1). Denne er igen en konkretisering af en yderligere bagvedliggende *algoritme*, som er det begreb, vi ønsker at indkredse.

320 Selve navnet „algoritme“ stammer fra den islamiske matematiker MUHAMMAD IBN MŪSĀ AL-KHWĀRIZMĪ (ca. 780–ca. 850), der i slutningen af 900-tallet beskrev det tal-system, vi i dag bruger og kalder for de hindu-arabiske tal. Da AL-KHWĀRIZMĪS værk blev oversat til latin, blev hans navn latiniseret som *Algorismus*, og deraf udledtes *algoritme* som navn for den sekvens af operationer, han beskrev for at regne med de nye cifre i 10-tals positionssystemet. Det er værd at bemærke, at simpel regning, for eksempel multiplikation af 3-cifrede tal, netop består af en ’opskrift’, som skal følges, og nogle primitiver, som skal kendes på forhånd (de små additions- og multiplikationstabeller).

325 Det er i sig selv bemærkelsesværdigt, at selvom matematikere og dataloger arbejder meget med algoritmer, er selve algoritmebegrebet svært at indkredse. Det algoritmebegreb, som vi kredser om, kan man forsøge at indfange ved hjælp af følgende definition:

330 En *algoritme* er en endelig, abstrakt, effektiv, sammensat kontrolstruktur,  
som er givet imperativt, og som opnår et givet formål under givne forhold.  
(Hill, 2016, s. 47)

Definitionen omfatter to dele, nemlig først en indkredsning af, hvad en opskrift-delen af algoritmen skal opfylde, og dernæst en (ikke helt standard) insisteren på, at algoritmer har formål og afgrænsede anvendelsesdomæner.

340 Det er centralt, at ROBIN K. HILL definerer algoritmens essens som en *kontrolstruktur*, dvs. som en struktureret beskrivelse, der foreskriver og kontrollerer den række handlinger, der skal udføres. Det er i den forstand, at den skal være imperativt givet — dvs. udtrykt i bydeform, der kræver visse handlinger udført. Men begrebet omfatter også, at der er tale om en kontrolleret række af handlinger, hvor rækkefølgen bestemmes (kontrolleres) af den aktuelle tilstand, kørslen befinner sig i. Man kan med andre ord beskrive, at visse handlinger skal gentages et vist antal gange (som bestemmes undervejs), eller at den næste handling skal være afhængig af informationer, vi aktuelt har tilgængelige.

At denne kontrolstruktur skal være endelig, betyder ikke, at den ikke kan kontrollere en uendelig proces, men at selve beskrivelsen skal være endelig. Så der ligger altså ikke i kravet om endelighed noget om, at algoritmen skal terminere. Den skal også være abstrakt og dermed dække forskellige instanser i modsætning til konkrete udførelser (som fx YBC 6967). Endvidere skal den være effektiv (eng.: *effective*) i den forstand, at der ikke må skulle indgå udefrakommende valg, fx truffet ud fra menneskelig forståelse eller lignende. Algoritmen skal altså være fuldt ud præciseret, og skal kunne udføres ’tankeløst’ eller ’mekanisk’, hvilket dog ikke kræver nogen egentlig mekanisme, hvorfor HILL ikke ønsker at bruge dette begreb.

345 Ud fra HILLS definition og hentydning til ’forudbestemthed’ (eng.: *predetermined*) er fx randomiserede og andre ikke-deterministiske beskrivelser faktisk udelukket fra begrebet *algoritme*. Tilsvarende overraskende er det, at HILL påpeger, at *rekursive definitioner* faktisk ikke er at betragte som algoritmer, fordi de ikke er givet imperativt, selvom de kan formuleres i en kontrolstruktur. Alle disse afgrænsninger kan — og er blevet — diskuteret indgående, når dataloger og filosoffer har forsøgt at indkredse det ellers så intuitivt plausible algoritmebegreb.

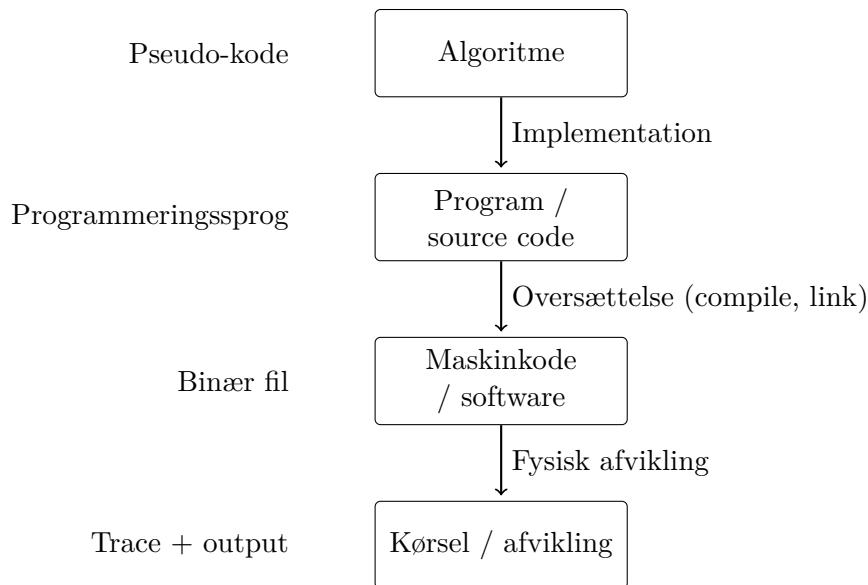
355 Men det måske allermest bemærkelsesværdige ved HILLS definition er faktisk en udvidelse af andre klassiske algoritmebegreber, idet hun kræver, at algoritmen opnår givne formål under givne forhold. Det er jo måske ikke så underligt (det betyder, at algoritmen

virker) — men det interessante er, at hun inddrager det som en del af selve definitionen af en algoritme. Dermed er en beskrivelse af en kontrolstruktur uden en beskrivelse af dens formål og antagelser altså slet ikke nogen algoritme i denne definition.

Man kan godt tænke om fysisk afvikling, maskinkode, program, og algoritme som gradvise lag af abstraktioner: Algoritmen er mere abstrakt end programmet, idet den fx ikke bundet til et bestemt programmeringssprog med fx bestemt syntaks og en bestemt måde at tilgå hukommelsen på. Og programmet er mere abstrakt end maskinkoden, hvortil det er blevet oversat og linket. Og endelig er maskinkoden mere abstrakt end kørslen, da programmet er en tekstuelt repræsentation, hvorimod kørslen er en fysisk og tidslig proces, der godt nok resulterer i noget, der igen kan indfanges tekstuelt (fx et ’trace’ og et ’output’).

365

370



Figur 3: Relationen mellem algoritme, program, maskinkode og kørsel som en kæde af abstraktioner og instanser.

Denne måde at opfatte programmering af en computer på åbner for, at vi kan betragte algoritmer, programmer og maskinkode som ’immaterielle objekter’, dvs. som ting, hvis egentlige eksistens ikke er som fysiske ting, selvom de kan repræsenteres fysisk som fx printouts af kildekode eller en USB-key med en binær fil. Endvidere kan vi opfatte processen fra algoritme til program til maskinkode som *oversættelser* der tager en given intention (formål) beskrevet på et abstraktionsniveau og omformer det til et andet abstraktionsniveau.

375

Hvis vi ønsker at sikre os, at en computer eller et program udfører *det rigtige*, dvs. at vi står foran en *korrekt* oversættelse, så har vi basalt set to muligheder: 1. Vi kan gå empirisk til værks og opfatte hele processen i figur 3 som en fysisk afvikling. Det er det, vi gør, når vi *tester* programmer (se mere i kapitel ??). 2. Men hvis vi afgrænser os til de tre øverste niveauer og oversættelserne, så kan vi faktisk betragte dem alle som matematiske objekter, og derfor kan vi også gå formalvidenskabeligt til værks og *bevise*, at et underliggende niveau er en *formelt korrekt* instans af det overliggende niveaus specifikation.

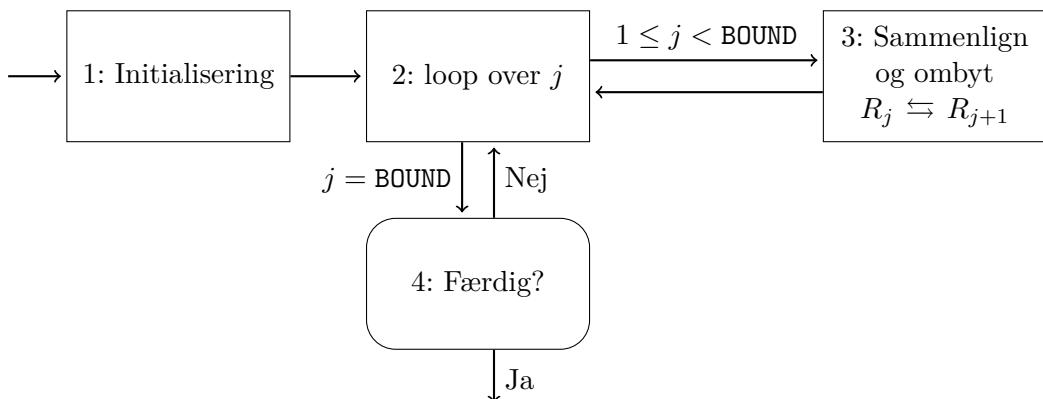
380

## 385 Bubble sort

Datalogen og Turing Award-vinderen DONALD KNUTH, som er blevet beskrevet som Silicon Valleys Yoda (Roberts, 2018), har igennem mere end et halvt århundrede arbejdet med sit store og stadigt ufuldendte opus *The Art of Computer Programming*. Deri beskriver og analyserer KNUTH en lang række algoritmer, og hele bind 3 er dedikeret til søgning og 390 sorteringsalgoritmer. Blandt de mange sorteringsalgoritmer, som han behandler, og som man også tit møder i lærebøger er den såkaldte *bubble sort*, som er en ‘in-place’ sorteringsalgoritme af en liste af ‘records’  $R_1, \dots, R_N$ , som hver indeholder en ‘key’  $K_1, \dots, K_N$  (Knuth, 1998, s. 107). Algoritmen består af fire trin:

1. Initialisering: Sæt  $\text{BOUND} := N$
- 395 2. Loop over  $j$ : Sæt  $t := 0$ . Udfør trin 3 for  $j = 1, 2, \dots, \text{BOUND} - 1$  og gå så til trin 4.
3. Sammenlign og byt: Hvis  $K_j > K_{j+1}$ , så ombyt  $R_j$  og  $R_{j+1}$  og sæt  $t := j$ .
4. Færdig? Hvis  $t = 0$  så terminerer. Ellers sæt  $\text{BOUND} := t$  og gå tilbage til trin 2.

KNUTH illustrerede denne tekstuelle algoritme med et flow chart, som svarer til figur 4.



Figur 4: KNUTHS diagrammatiske repræsentation af *bubble sort* som et *flow chart* (Knuth, 1998, s. 107). De rettangulære kasser er handlinger, mens den rundede kasse er en afgørelse.

400 Tallene i flow chartet (figur 4) henviser til trin i algoritmen, og det var typisk den måde, man beskrev algoritmer på som en blanding af flow chart og instruktioner for hver kasse. Men KNUTH angav også et program, skrevet i maskinkode med instruktionssættet **MIX**, som han arbejdede med samtidig.

Efter at have angivet algoritmen, analyserede KNUTH dens kompleksitet i termér af antal gennemløb, antal ombytninger og antal sammenligninger, og han fandt, at de sidste to var kvadratiske i  $N$  og værre end *insertion sort*, som han også havde analyseret. Endvidere foreslog han forskellige forbedringer til *bubble sort*, inden han dog nåede den lakkoniske konklusion:

In short, the bubble sort seems to have nothing to recommend it, except a 410 catchy name and the fact that it leads to some interesting theoretical problems.  
(Knuth, 1998, s. 110)

For dog at trække en vigtig teoretisk pointe skal vi vende tilbage til *bubble sort* i det følgende. Og trods KNUTHS patos-fulde avisning af *bubble sort*, vedbliver den at være meget udbredt, måske fordi den er nem at huske og nem at implementere (Astrachan, 2003). Den er på samme tid en lidt kluntet schweizerkniv i programmørens lomme og en datalogiens *drosophila* for den teoretiske datalog, der kender den som et udbredt og gennemstuderet modeldyr.

415

### 3.4 Computere og matematiske beviser

Den skotske videnskabssociolog DONALD MACKENZIE har interesseret sig meget for forholdet mellem det, han kalder forskellige „beviskulturer“. Ved at opfatte beviser i termen af *kulturer*, flytter han fokus fra mere traditionel matematikfilosofi, der har opfattet beviser som tidsløse sandhedsgaranter, over i en mere praksis-orienteret videnskabsteori, hvor beviser er noget, *mennesker* producerer i bestemte sociale sammenhænge. En af de mest spændende analyser, som kommer ud af denne tilgang, er at betragte forskellige opfattelser af, hvad det vil sige at en computer-chip er *bevist* korrekt, når man spørger henholdsvis programmører, matematikere og jurister. Sådan en analyse var MACKENZIE i stand til at gennemføre for den såkaldte „VIPER chip“, som var den første commercielle mikroprocessor, som var blevet formelt bevist korrekt (MacKenzie, 1991). Og mere generelt har MACKENZIE interesseret sig for forholdet mellem matematik og computere, hvor han har undersøgt forskellige former for overlap (se figur 5). Han betragter især tre former for brug af computere i matematik: 1. Til såkaldt ’automated theorem proving (ATP)’, som består i, at et stykke software med eller uden hjælp fra et menneske søger efter en formel udledning af en matematisk sætning ud fra givne præmisser (se også Johansen, Breman og Sørensen, 2019). 2. Til AI-understøttelse af matematisk kreativitet (ikke kun i form af ATP men også fx mere heuristisk); inden for de seneste tiår har man endda udviklet en *eksperimentel* gren i matematikken, som bruger computeren interaktivt og udforskende (se Sørensen, 2016b; Sørensen, 2013; Sørensen, 2010). 3. Og til understøttelse af meget store matematiske beviser med mange cases at behandle, som det for eksempel er tilfældet med Firefarvesætningen og Keplers formodning.

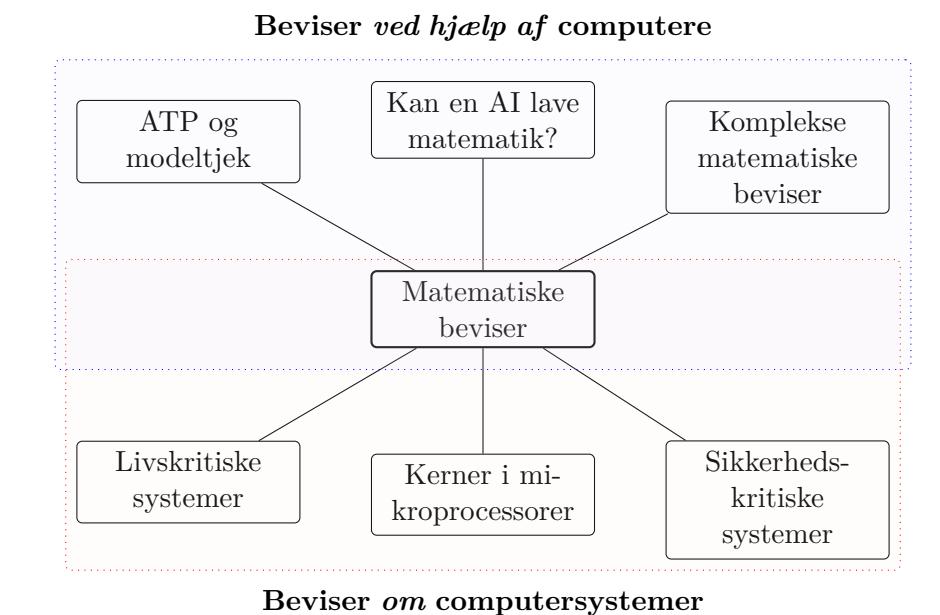
420

425

430

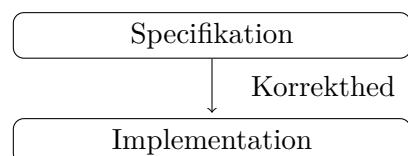
435

440



Figur 5: Matematiske beviser og computere, baseret på MacKenzie (2005).

Denne allerede for 20 år siden omfattende *brug* af computere og software til formal-videnskabelig vidensproduktion er vigtig i sig selv, men her er det særligt vigtigt, at den også finder anvendelse *inden for* datalogien. MACKENZIE opnår tre særlede områder, hvor formelle beviser *om* computersystemer har spillet en betydelig rolle: 1. Det gælder omkring kerner og mikroprocessorer, for hvis man kan bevise, at processoren på det laveste niveau, abstraktioner oven på den, og oversættelserne imellem dem er korrekte, så er man tæt på at bevise korrektheden af computersystemet. 2. Det gælder inden for livs- og 3. sikkerhedskritiske systemer til både civil og militær anvendelse, for det vil være oplagt at søge at *bevise*, at et kritisk computersystem fx ikke kan hackes, ikke kan komme til at foretage brud på fortrolighed, eller ikke kan bringes til at crashe eller malfunktionere på anden vis. Da MACKENZIE skrev for 15 år siden, var udfordringerne ved brug af formelle metoder imidlertid meget store, og primært af to slags: Dels en praktisk udfordring, som bestod i, at det var enormt tidskrævende at gennemføre formelle beviser, og dels en mere principiel udfordring, som består i, at formelle beviser aldrig kan bevise udsagn om realvidenskabelige forhold (se fx diskussionen mellem De Millo, Lipton og Perllis, 1979; Fetzer, 1988). Men siden har feltet udviklet sig enormt hurtigt, ikke mindst fordi datalוגer og matematikere har fælles interesser i at udvikle platforme, der kan hjælpe med til formel verifikation. Så de oprindelige praktiske udfordringer er blevet mindre med udviklingen af værktøjer som Isabelle/HOL, og i dag findes der formelt verificerede C-compilere, operativsystemer, mikroprocessorer, konferencesystemer og meget mere. Men om alle disse systemer skal man erindre, at et formelt bevis 'blot' er et bevis for, at et niveau af software-stakken er en korrekt implementation af en specifikation på et højere niveau (se figur 6).



Figur 6: En formel verifikation er et (typisk computer-assisteret) bevis for, at et niveau af software-stakken er en korrekt implementaton af en specifikation på et højere niveau.

## Beviser om algoritmer og modeller

En af de mest bemærkelsesværdige egenskaber ved algoritmer er, at vi kan komme til at betragte dem som matematiske objekter. Det er ikke helt så ligetil at skrive en algoritme ned i ord eller tegn, men dele af HILLS definition er netop med for at sikre, at det kan lade sig gøre. Algoritmen er forskellige fra en konkret implementation af den, men man har udviklet forskellige mere eller mindre formaliserede måder at repræsentere *algoritmer* på, uafhængigt af specifikke aspekter ved givne programmeringssprog. Blandt de mest udbredte repræsentationsformer var *flow-diagrammer* og *pseudo-kode*, som typisk er baseret på en simplificeret form af udbredte programmeringssprog som fx *FORTRAN* (se også kap. ??).

Formuleringen af fx *bubble sort*-algoritmen gennemgået ovenfor tillader os at *bevise* fx

1. Korrekthed af algoritmen: Vi kan *bevise*, at hvis algoritmen terminerer, så opfylder  $K_i \leq K_j$  for  $i \leq j$ .
2. Terminering af algoritmen: Algoritmen terminerer faktisk, dvs. vi når den endelige tilstand, hvor der ikke er flere ombytninger at lave.
3. Kompleksiteten af algoritmen kan udtrykkes i  $\mathcal{O}$ -notation, dvs. vi kan angive asymptotiske grænser for, hvor mange hhv. gennemløb, ombytninger og sammenligninger, der foretages. Disse siger kun noget om, algoritmens effektivitet i *grænsen*, dvs. for store værdier af  $N$ , og kun hvis vores valg mellem gennemløb, ombytninger og sammenligninger faktisk indfanger den mest omkostningsfulde operation på den givne implementation.

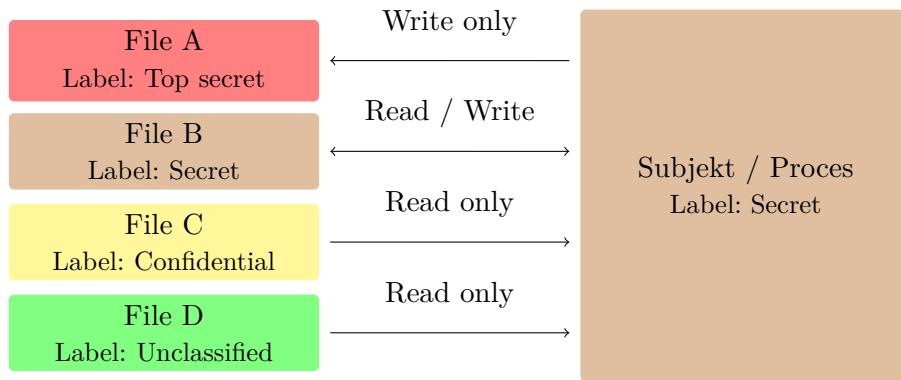
Så sammenfattende kan vi *bevise* noget om *algoritmens* korrekthed og effektivitet, og dette overfører til udsagn om de programmer, der er korrekte implementationer af algoritmen. Men det overfører ikke til viden i den fysiske verden om udkommet af en konkret beregning (som kan blive ramt af alle mulige uforudsete hændelser) eller beregningshastigheden, som sagtens kan være domineret af startomkostninger (i dette tilfælde konstantled og lineære led).

Det er måske lidt fjallet at diskutere *bubble sort* så meget, når nu KNUTH og mange andre allerede har advaret os imod den algoritme, fordi den er „dårlig“. Men pointerne er principielle, vedvarende, vigtige og omkostningsfulde, sådan som vi skal se ved at betragte den såkaldte *Bell-LaPadula-model*.

Som et af resultaterne af at man under den såkaldte *softwarekrise* i slutningen af 1960’erne erkendte, at megen software var upålidelig, indførte det amerikanske forsvar retningslinjer for, hvordan den software, de skulle købe eller bygge, skulle overholde standarder for sikkerhed. Problemets var særligt udtalt, da computerne på dette tidspunkt ikke var isolerede enkelmaskiner, men mange bruger kørte jobs på den samme mainframe ved hjælp af *time sharing*. Derfor var det en udpræget risiko, at en proces kunne komme til at læse eller skrive i hukommelse, som (også) tilhørte en anden proces. Og selvom dette ville være en generel *bug* ved et time sharing system, kunne det have kritiske konsekvenser, hvis det betød læk af fortrolig militær viden. Derfor satte militæret sig for at specificere en model, der skulle sikre, at informationsflow overholdt fortrolighedsprincipper — denne model blev til *Bell-LaPadula-modellen* (se Bell, 2011). Og endvidere indførte militæret den betingelse på

505 deres kontrakter, at software *skulle* være bevist at overholde *Bell-LaPadula-modellen*, før de kunne købe det — denne specifikation i den såkaldt 'Orange bog' måtte dog løsnes igen, da det viste sig alt for omkostningsfuldt.

510 Den model, som DAVID ELLIOTT BELL og LEONARD J. LAPADULA producerede og dokumenterede for det amerikanske militær blev kendtegnet ved to regler: *no write-down* og *no read-up*. Det skal forstås sådan, at alle elementer af computersystemet har en label, som angiver fortroligheden (top secret, secret, confidential og unclassified), og alle subjekter og processer, der tilgår systemet har ligeledes en af disse labels. En proces kan så ikke skrive på et niveau under sit eget og ikke læse fra et niveau over sit eget (se figur 7).



Figur 7: Bell-LaPadula-modellen, som blev udviklet for det amerikanske forsvar. Der er tale om en sikkerhedsmode for multi-level-systemer, som sikrer *no write-down, no read-up*. Den brune proces (subjekt) kan læse filer på samme eller lavere niveau men kan kun skrive på sit eget eller højere sikkerhedsniveau.

515 Med denne specifikation kunne militæret nu kræve, at leverandører beviste, at deres systemer levede op til *Bell-LaPadula-modellen*. Men der skulle vise sig at være et problem, hvis systemet havde delte ressourcer som fx et filsystem eller en printer. For så viste det sig, at man kunne løbe ind i det, der i dag kende som en 'covert channel' — en hemmelig og utilsigtet kanal.

520 Antag, at vi har følgende fire operationer, som alle overholder *Bell-LaPadula-modellen*:

**READ objname:** Hvis objektet `objname` eksisterer, og subjektet har læseadgang, returner værdien til subjektet; ellers returner 0.

**WRITE objname, value:** Hvis `objname` eksisterer, og subjektet har skriveadgang, så erstat værdien med `value`; ellers gør ingenting.

525 **CREATE objname:** Hvis `objname` ikke eksisterer, så opret det på subjektets niveau; ellers gør ingenting.

**DESTROY objname:** Hvis `objname` eksisterer og subjektet har skriveadgang til det, så slet det; ellers gør ingenting.

530 Hvis nu  $H$  er en agent på højt niveau, som vil kommunikere til  $L$  på et lavt niveau, så kan de indgå i en protokol (hvilket selvfølgelig kræver, at de samarbejder). Husk:  $H$  kan ikke skrive til objekter på  $L$ 's niveau, og  $L$  kan ikke læse objekter på  $H$ 's niveau.

1. Hvis  $H$  vil sende 0: CREATE F0; hvis  $H$  vil sende 1: gør ingenting
2.  $L$ : CREATE F0; WRITE F0, 1; READ F0; DESTROY F0

Nu ser  $L$  det samme, som  $H$  ønskede at sende: For hvis  $H$  vil sende 1 kan  $L$  godt oprette og skrive til F0, hvorfor  $L$  læser 1; omvendt hvis  $H$  vil sende 0 kan  $L$  ikke komme til at oprette F0, og derfor ikke læse fra F0, hvorfor  $L$  ser 0. Altså er information blevet sendt fra højt niveau til lavt niveau, hvilket netop strider mod *no write down*. 535

Covert channels er et fundamentalt problem, som peger på, at modellen er utilstrækkelig i sine antagelser. For formalvidenskabens deduktive beviser kan *kun* benyttes om *lukkede systemer*, hvor vi kender og tager højde for alle faktorer — og der er de ufejlbare. Men i åbne systemer, hvor agenter kan opføre sig på måder, vi ikke havde taget højde for, nyttet vores matematiske beviser ikke. Sagt på en anden måde, så kan vi (forsøge at) bevise, at en *implementation* er en korrekt konkretisering af en *specifikation*. Men vi kan ikke bevise, at specifikationen i figur 6 ikke har utilsigtede *åbninger* eller sideeffekter. Dertil kræves andre tilgange — hvoraf den mest oplagte er, at vi lærer af vores tidligere fejl, sådan at vi nu indbygger potentialet af covert channels i vores antagelser. På den måde lever *Bell-LaPadula-modellen* videre — ikke længere som en uomgængelig standard, men som grundlag for læring og videreudvikling. 540  
545

### 3.5 Videnskabelige revolutioner

Nu vender vi tilbage til TURINGS beskrivelse af beregninger, som HARTMANIS gav udtryk for, var et grundlag for den måde, vi i datalogi måler arbejdet ved en beregning (kompleksitetsteori). 550

I kapitel 2 beskrev vi THOMAS KUHNS (1922–1996) videnskabelige paradigmer som det fælles verdenssyn, som en gruppe videnskabsfolk er fælles om at dele. Paradigmet var konstitueret af en *disciplinær matrix*, som omfattede symbolske generaliseringer, metafysiske antagelser, værdier og eksemplarer, som gruppen er forpligtede på. Men vi forklarede ikke, hvordan KUHN benyttede denne indfagning af den fælles sociologiske ramme til at forklare videnskabens udvikling. Nu er det på tide at vende tilbage og beskrive denne dynamik for at kunne bringe den i anvendelse til at karakterisere og perspektivere det, vi kunne kalde 'datalogiens teoretiske paradigm'. 560

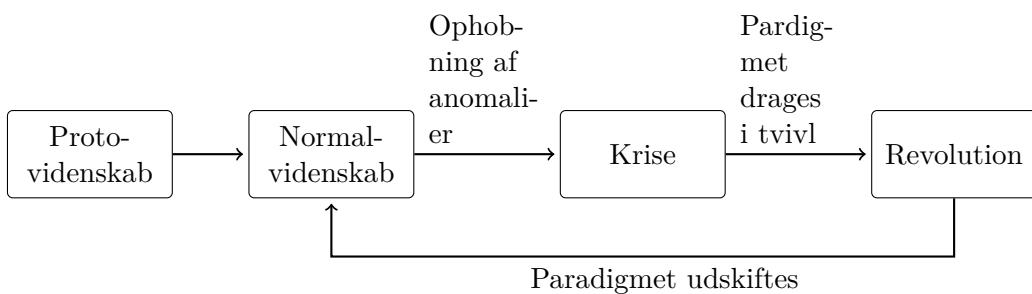
For det første bemærker man, at en enkelt videnskabsmand (m/k) i streng forstand, ifølge KUHNS teori, kun kan tilhøre et enkelt paradigme, idet et paradigme skal omfatte et *helt* verdenssyn. Så paradigmer er ikke noget, man let skifter ud, og den enkelte udøver vil være meget tilbøjelig til at forsøre og søge at opretholde sit paradigme, fordi man har 'investeret' så meget i at blive socialiseret ind i det og blive dygtig til at udøve det. 565

Den videnskab, som man udøver *inden for* et paradigme, kalder KUHN for 'normalvidenskab', og den er karakteriseret ved, at udøverne så at sige *overholder* og *udforsker* paradigmet. Man søger at udvide den indsigt, som kan formuleres som svar på spørgsmål inden for paradigmet. KUHNS egen baggrund er i videnskabshistorien, nærmere bestemt fysikhistorien, og hans gennemgående eksempel er verdensbilledets udvikling, og særligt overgangen fra et geocentrisk verdensbillede, som associeres med den helenistiske astronom KLAUDIOS PTOLEMAIOS (ca. 100–ca. 170), til et heliocentrisk verdensbillede, som vi kalder moderne og især associerer med NIKOLAUS KOPERNIKUS (1473–1543). I det geocentriske paradigme blev jorden betragtet som universets centrum, og man mente, at planeter og stjerne bevægede sig i cirkelbaner rundt om jorden. Det heliocentriske verdensbillede satte 570  
575

derimod solen i centrum for universet, og forandre jordens status til en planet om solen. I første omgang beskrev man bevægelserne som cirkulære omkring solen, men med JOHANNES KEPLER (1571–1630) blev planetbanerne raffineret til at være elliptiske. Omkring verdensbilledet forklaerer KUHN normalvidenskabelig praksis inden for fx det geocentriske verdensbillede som yderligere observationer af planeternes baner og mere og mere forfinede beregninger af fx fremtidige solformørkelser.

580 Imidlertid kan der i løbet af den normalvidenskabelige praksis blive afdækket såkaldte „anomalier“, dvs. uoverensstemmelser mellem paradigmets forudsigelser og empiriske fænomener. I modsætning til en radikal fortolkning af KARL POPPERS (1902–1994) metode, mente KUHN, at sådanne anomalier enten ville blive forsøgt bortforklaret inden for paradigmets begrebsverden eller simpelthen gent et væk og efterladt til senere undersøgelse. Man ville altså netop *ikke*, ifølge KUHN, smide paradigmatet (hypotesen i POPPERS terminologi) overbord ved første tegn på modstand, men i stedet henlægge problemer og arbejde videre. 585 I astronomihistorien er det mest berømte eksempel på anomalier, at man opdagede, at visse himmellegemer nogle gange bevægede sig 'baglæns' (retrograd bevægelse), hvilket ikke fik astronomer i PTOLEMAIOS' paradigme til at opgive den geocentriske antagelse, men i stedet til at tilføje yderligere teoretiske antagelser om, at himmellegemerne bevægede sig på såkaldte „epicykler“, dvs. på cirkler omkring punkter på cirkelbevægelser omkring jorden. Andre anomalier blev simpelthen noteret men ikke forfulgt.

590 På et tidspunkt kan ophobningen af anomalier blive så omfattende, at nogle af udøverne begynder at drage paradigmatet i tvivl, og så siger KUHN, at paradigmatet kommer i 'krise' og en vis del af den videnskabelige praksis går over i en ekstraordinær tilstand, hvor man afsøger mulige alternative formuleringer af de helt centrale antagelser. Denne krisetilstand kan (kun) løses igennem en „revolution“, hvorved det gamle paradigme forkastes og et nyt 600 sættes i stedet. Det vil typisk ikke være en omvæltning, den enkelte udøver er i stand til at gøre, da man har investeret for meget i det gamle paradigme, og KUHN beskriver det som, at gamle paradigmer dør ud med de sidste af deres tilhængere. Men for gruppen sættes et nyt paradigme i stedet, og krisen er blevet løst, så man nu vender tilbage til tilstanden af normalvidenskab (mono-paradigmatisk videnskab), men under et nyt paradigme. KUHNS dynamik er forklaret skematisk i figur 8.



Figur 8: Skematisk repræsentation af KUHNS dynamik for videnskabens udvikling.

Hvor POPPER og mange andre videnskabsteoretikere havde søgt at forklare, hvordan videnskaben *gør fremskridt*, satte KUHN altså en cirkulær proces i stedet, hvor man kun kan tale om fremskridt inden for den normalvidenskabelige praksis. For på tværs mellem to 610 efterfølgende paradigmer eksisterer der, ifølge KUHN, hvad han kalder „inkommensurabilitet“. Dette begreb er en meget omdiskuteret del af KUHNS teori, og ordet betyder strengt taget 'usammenmåelighed', og det dækker over to ting: 1. Dels henviser det til, at KUHN

mente, at det ikke var muligt at *oversætte* mellem to paradigmer, for deres grundantagelser var simpelthen inkompatible, og samme ord kunne fx betyde to helt forskellige ting, ligesom samme objekt kunne have ganske forskellig status i de to paradigmer. For eksempel var jorden jo universets centrum for PTOLEMAIOS men blot en blandt mange planeter hos KOPERNIKUS. 2. Mere radikalt henviser inkommensurabilitet også til, at der ikke findes nogen måde rationelt at *sammenligne* to paradigmer på: Man kan ikke stå uden for dem og veje dem op i mod hinanden på at udefrakommende, rationelt grundlag, for hvert paradigme vil have sine egne værdier og grunde til at hævde sine egne kvaliteter. KUHN argumenterer således for, at der ikke findes objektive kriterier at vurdere paradigmer (teorier) ud fra (Kuhn, 1995). Denne uoversættelighed og usammenlignelighed er en af de helt karakteristiske egenskaber ved en „videnskabelig revolution“, sådan som KUHN fremlægger det. Den har nogle særlige konsekvenser, hvoraf vi kun kort skal berøre to, der handler om teoriladethed og interdisciplinaritet.

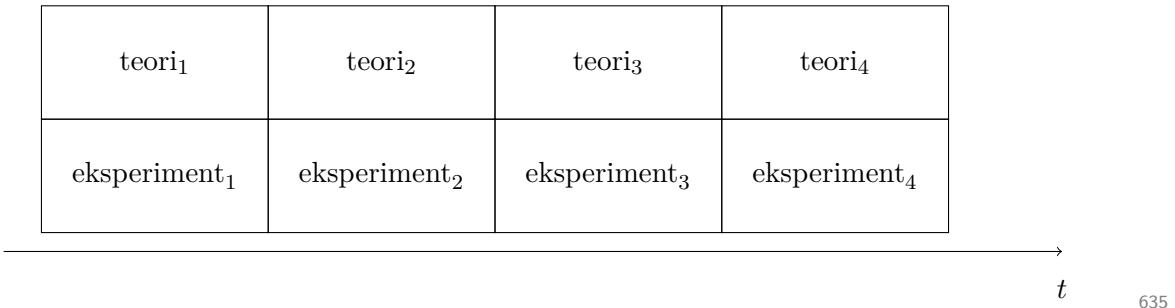
615

I KUHNS opfattelse er det ikke kun de centrale begreber og antagelser, der forandres hen over et paradigmeskift — det er så at sige også selve verden omkring os, eller i hvert fald hele vores adgang til verden omkring os. For paradigmet omfatter hele vores verdenssyn, og hvis to paradigmer ikke er oversættelige, så er al vores viden heller ikke oversættelig. Det gælder således ikke kun vores teorier om verden, men også vores empiriske adgang til den (vores eksperimenter). Dette har videnskabsteoretikeren PETER GALISON beskrevet som KUHNS „centrale metafor“: At i en revolution forkastes både teorier og eksperimenter, for eksperimenterne er *teoriladede* og kan ikke tænkes adskilt fra den teoretiske ramme, de er udført under (se figur 9).

620

630

635



Figur 9: KUHNS såkaldt „centrale metafor“ som introduceret i Galison (1988).

Men KUHN står over for en væsentlig udfordring, som består i, at forklare, hvor mange samtidige paradigmer, der kan findes. For eksempel kan kemikere og fysikere jo i principippet godt begge udforske naturen, men tilhører de det samme paradigme? KUHN ville nok sige nej — paradigmer er mere specialiserede end blot 'naturvidenskab' eller 'humaniora', og man kan godt tænke sig dem opdelt efter det, vi typisk kalder *discipliner*. Men hvad så med områder som fx nano-videnskab, der bevidst går på tværs mellem paradigmer som biologi, kemi og fysik? Og her har videnskabsteoretikere med udgangspunkt i KUHNS teori analyseret, hvordan sådanne „interdisciplinære“ områder kan være ramt af konfliktede værdier og uoversætteligheder mellem deres indgående paradigmer.

640

Lad os nu vende tilbage til 'datalogiens teoretiske paradigme', ved hvilket vi vil forstå den grundantagelse (verdenssyn), som TURING og HARTMANIS tilsammen kan ses at give: Datalogien handler om studiet af beregninger (data processer) og deres kompeksitet, og alle typer beregninger kan (ultimativt) hidføres til den fundationale teoretiske model, som vi kalder den universelle Turing-maskine.

645

650 Vi kan nu dels overveje, om denne indfangning faktisk er dækkende for vores opfattelse af datalogien, og den stemmer i hvert fald overens med mange aspekter af den karakterisering, vi i kapitel 1 uddrog fra forskellige positioner og leksika. Dernæst kan vi overveje, om den giver mening som et kuhnsk paradigme, altså som et delt verdenssyn for en gruppe af udøvere med symbolske generaliseringer, metafysiske antagelser, værdier og eksemplarer, 655 og det er præcist det, at HARTMANIS udtrykker i citaterne, og som er blevet beskrevet som at dataloger er „Turing’s Men“ (Bolter, 1986) fordi de går til verden med et filter, der ser verden udelukkende i termer af beregninger.

Nu kan vi så også overveje, om karakteriseringen af datatalogi som en „vidtfavnende tværvidenskabelig disciplin“ giver mening, for hvis dette er datalogiens teoretiske paradigm 660 kan vi forudse, at der kan opstå uoversætteligheder i mødet med andre paradigm, fx inden for naturvidenskabelige discipliner. Og netop her ligger kimen til en af datalogiens store indsigter de sidste 50 år: Hvis datalogi skal være en hjælp til andre videnskaber (og sikkert også til andre opgaver), så kræver det nok netop ’tværvidenskabelighed’. Og så kunne karakteriseringen måske udvikles til, at den form for datalogi nærmere er en *hybrid* mellem 665 det teoretiske paradigme og andre naturvidenskabelige paradigm, ligesom nano-videnskab er en hybrid, fordi det udvikler *sit eget* paradigme på tværs af fysik, biologi og kemi.

Og endelig kan vi overveje, om der er nogen tegn på, at det teoretiske paradigme kunne 670 være på vej i krise og måske ikke er fuldt dækkende for det, vi vil opfatte som ’beregning’. Her er kvantecomputeren måske den mest oplagte udfordring, for hvis det lykkes at gøre den praktisk anvendelig i større stil, kunne den udfordre antagelsen om, at alle beregninger lader sig reducere til den universelle Turing-maskine. Disse bestræbelser på at opnå „quantum supremacy“, dvs. bygge en kvantecomputer, der kan løse problemer, en almindelig computer ikke kan (i samme kompeksitet), er helt aktuel forskning i mange forsknings- og udviklingsnetværk, der omfatter private virksomheder som Google og Microsoft og førende universiteter verden over.

## Litteratur

- Astrachan, Owen (jan. 2003). „Bubble sort. An Archaeological Algorithmic Analysis“. *ACM SIGCSE Bulletin*, bd. 35, nr. 1. DOI: 10.1145/792548.611918.
- Bell, David Elliott (2011). „Bell–LaPadula Model“. I: *Encyclopedia of Cryptography and Security*. Red. af Henk C. A. van Tilborg og Sushil Jajodia. Springer US, s. 74–79. DOI: 10.1007/978-1-4419-5906-5\_811.
- Bolter, J. David (1986). *Turing’s Man: Western Culture in the Computer Age*. London etc.: Penguin Books.
- Campbell-Kelly, Martin m.fl., red. (okt. 2003). *The History of Mathematical Tables*. Oxford University Press. DOI: 10.1093/acprof:oso/9780198508410.001.0001.
- Copeland, B. Jack (2008). „The Church-Turing Thesis“. I: *The Stanford Encyclopedia of Philosophy*. Red. af Edward N. Zalta.
- Davis, Martin (1982). *Computability and Unsolvability*. Mineola: Dover.
- De Millo, Richard A., Richard J. Lipton og Alan J. Perllis (maj 1979). „Social Processes and Proofs of Theorems and Programs“. *Communications of the ACM*, bd. 22, nr. 5, s. 271–280.
- Fetzer, James H. (sep. 1988). „Program Verification. The Very Idea“. *Communications of the ACM*, bd. 37, nr. 9, s. 1048–1063. DOI: 10.1145/48529.48530.

- Galison, Peter (1988). „History, Philosophy, and the Central Metaphor“. *Science in Context*, bd. 2, nr. 1, s. 197–212. 695
- Haigh, Thomas, Mark Priestley og Crispin Rope (2014). „Los Alamos Bets on ENIAC. Nuclear Monte Carlo Simulations, 1947–1948“. *IEEE Annals of the History of Computing*, bd. 36, nr. 3, s. 42–63. DOI: 10.1109/MAHC.2014.40.
- Hartmanis, Juris (okt. 1994). „Turing Award Lecture: On Computational Complexity and the Nature of Computer Science“. *Communications of the ACM*, bd. 37, nr. 10, s. 37–43. 700
- Hilbert, David (1983). „On the infinite“. I: *Philosophy of mathematics. Selected readings*. Red. af Paul Benacerraf og Hilary Putnam. 2. udg. Cambridge: Cambridge University Press, s. 183–201.
- Hill, Robin K. (2016). „What an Algorithm Is“. *Philosophy & Technology*, bd. 29, nr. 1, s. 35–59. DOI: 10.1007/s13347-014-0184-5. 705
- Hoff Kjeldsen, Tinne (2000). „A Contextualized Historical Analysis of the Kuhn-Tucker Theorem in Nonlinear Programming: The Impact of World War II“. *Historia Mathematica*, bd. 27, s. 331–361.
- Høyrup, Jens (1998). *Algebra på lertavler*. Matematiklærerforeningen.
- Johansen, Mikkel Willum, Hester Breman og Henrik Kragh Sørensen (sep. 2019). „Human and Computerized Mathematical Practice. Experimental Mathematics and Interactive Theorem Provers“. Under udarbejdelse. 710
- Johansen, Mikkel Willum og Henrik Kragh Sørensen (2014). *Invitation til matematikkens videnskabsteori*. København: Forlaget Samfunds litteratur.
- Kahn, David (1996). *The Codebreakers. The Comprehensive History of Secret Communication from Ancient Times to the Internet*. Revideret og opdateret. New York: Scribner. 715
- Knuth, Donald E. (1998). *The Art of Computer Programming*. Bd. 3: *Searching and Sorting*. 2. udg. Reading (Mass) m.m.: Addison-Wesley.
- Kuhn, Thomas S. (1995). „Objektivitet, værdidom og valg af teori“. I: *Videnskabens revolutioner*. København: Forlaget Fremad, s. 262–285. 720
- MacKenzie, Donald (aug. 1991). „The fangs of the VIPER“. *Nature*, bd. 352, nr. 6335, s. 467–468. DOI: 10.1038/352467a0.
- (2005). „Computing and the cultures of proving“. *Philosophical Transactions of The Royal Society. A*, bd. 363, s. 2335–2350. DOI: 10.1098/rsta.2005.1649.
- Pariser, Eli (2011). *The filter bubble. What the Internet is hiding from you*. New York: Penguin Press. 725
- Roberts, Siobhan (dec. 2018). „The Yoda of Silicon Valley“. *New York Times*.
- Seaver, Nick (2017). „Algorithms as culture. Some tactics for the ethnography of algorithmic systems“. *Big Data & Society*, s. 1–12. DOI: 10.1177/2053951717738104.
- Shustek, Len (2015). „An Interview with Juris Hartmanis“. *Communications of the ACM*, bd. 58, nr. 4, s. 33–37. DOI: 10.1145/2736346. 730
- Sørensen, Henrik Kragh (2010). „Exploratory experimentation in experimental mathematics. A glimpse at the PSLQ algorithm“. I: *PhiMSAMP. Philosophy of Mathematics: Sociological Aspects and Mathematical Practice*. Red. af Benedikt Löwe og Thomas Müller. Texts in Philosophy 11. London: College Publications, s. 341–360. 735
- (2013). „Er det matematiske bevis ved at dø ud?“ I: *Fremtiden*. Red. af Ole Høiris. Aarhus: Aarhus Universitetsforlag, s. 99–132.
- (apr. 2016a). *Kildecentreret matematikhistorie i praksis: En kort indføring i matematikhistorisk metode*. RePoSS: Research Publications on Science Studies 38. Aarhus: Centre for Science Studies, University of Aarhus. 740

- Sørensen, Henrik Kragh (2016b). „'The End of Proof'? The integration of different mathematical cultures as experimental mathematics comes of age“. I: *Mathematical Cultures. The London Meetings 2012–2014*. Red. af Brendan Larvor. Trends in the history of science. Birkhäuser, s. 139–160. DOI: 10.1007/978-3-319-28582-5\_9.
- 745 Thomas, Suzanne L., Dawn Nafus og Jamie Sherman (2018). „Algorithms as fetish. Faith and possibility in algorithmic work“. *Big Data & Society*, s. 1–11. DOI: 10.1177 / 2053951717751552.
- Turing, A. M. (1936). „On Computable Numbers, with an Application to the Entscheidungsproblem“. *Proceedings of the London Mathematical Society (series 2)*, bd. 42, s. 230–265. DOI: 10.1112/plms/s2-42.1.230.
- 750 — (2004). „Intelligent Machinery“. National Physical Laboratory Report. I: *The Essential Turing. Seminal Writings in Computing, Logic, Philosophy, Artificial Intelligence, and Artificial Life plus The Secrets of Enigma*. Red. af B. Jack Copeland. Oxford: Clarendon Press, s. 410–432.
- 755 Zuboff, Shoshana (2019). *The age of surveillance capitalism. The fight for a human future at the new frontier of power*. London: Profile Books.

# Kapitel 4:

## Kunsten at få en computer til at adlyde (I): Programmering

Henrik Kragh Sørensen

Mikkel Willum Johansen

22. april 2022

### Indhold

5

4.1 Modularisering, abstraktion og grænseflader . . . . .	1
4.2 Programmeringssprog . . . . .	3
4.3 Programmeringsparadigmer . . . . .	11
4.4 Fra algoritme til afvikling . . . . .	13
4.5 Fuld fart frem . . . . .	16
	10

### Litteratur

18

#### 4.1 Modularisering, abstraktion og grænseflader

På det tekniske niveau består en afvikling af et program på en moderne computer principielt i, at processoren henter instruktionerne en ad gangen, afkoder instruktionen og udfører den. Betragtet på den vis, er et computerprogram en streng af instruksionskoder — det, vi i kapitel 3 har kaldt *software* eller *maskinkode*. I nogle sammenhænge, hvor plads eller hastighed er voldsomt begrænsende faktorer, kan det være nødvendigt foringeniøren at programmere computeren direkte i maskinkode, kun hjulpet af en såkaldt *assembler*, der ikke stiller ret megen understøttelse til rådighed for programmøren. Men i de fleste andre tilfælde foregår programmeringen af den fysiske computer igennem en lang række lag af abstraktioner, modulariseringer og oversættelser. I dette afsnit skal vi først betragte abstraktioner og modulariseringer, inden vi i afsnit 4.2 fokuserer på programmeringssprog.

15

Når man skal designe software er der mange hensyn at afbalancere, og vægtningen kan variere mellem forskellige projekter: Nogle gange er pålidelighed afgørende, nogle gange er hastighed vigtigst, nogle gange skal man prioritere omkostninger ved at vedligeholde koden over en lang årrække. Og som regel spiller alle disse hensyn nært sammen uden at programmøren har en eksplisit *videnskabelig* teori at basere sine valg på. Derfor er softwareudvikling ofte blevet betegnet som en mere ingeniøragtig praksis, hvor erfaringer og 'best-practices' spiller en stor rolle, eller endda som en artistisk udfoldelse, sådan som DONALD KNUTH har argumenteret for (se Bond, 2005).

20

Men også softwareudvikling — og især under termen *software engineering* — har stræbt efter at finde et kollektivt grundlag at bygge på i form af noget, der kunne ligne et *paradigme* (Northover m.fl., 2008). Selve begrebet *software engineering* blev første gang brugt under en meget omtalt og betydningsfuld konference arrangeret af NATO i Garmisch i Vesttyskland

25

30

35 i 1968 (se fx MacKenzie, 2001). Konferencen var blevet indkaldt, fordi man i løbet af  
1960’erne havde oplevet mange software-projekter gå over tid og over budget eller ikke leve  
op til forventningerne: Man stod i en *softwarekrise*. Udvikling af software havde, med andre  
ord, vist sig svær at kontrollere, og det havde stor bevågenhed både fra militær side og fra  
40 de store civile producenter. Og vejen ud af krisen syntes at være at gøre udviklingen af  
software til en *ingeniørvidenskab*, som kunne bygge bro mellem den akademiske, teoretiske  
viden og praktikerens behov. Men fornemmelsen af krise i 1960’erne førte også til, at man  
søgte at udvikle og forfine processer og især programmeringssprog, der kunne understøtte  
effektiv og pålidelig software.

45 Blandt de mange nye ideer, som *softwarekrisen* førte med sig, var ideen om *struktureret  
programmering*, som særligt den hollandske datalog EDSGER DIJKSTRA (1930–2002) havde  
fremført fra 1966 og som han forsvarede indgående og polemisk i et opgør med den ellers  
så anvendte *goto*-kommando (Dijkstra, 1968). Alle instruktionssæt havde en primitiv kom-  
mando *JUMP*, der flyttede programtælleren til en given adresse i hukommelsen, og denne  
egenskab var naturligt også blevet medtaget i de tidligere programmeringssprog, herunder  
50 også det populære sprog ALGOL (se Rutishauser, 1967).

The **go to** statement as it stands is just too primitive; it is too much an  
invitation to make a mess of one’s program. (Dijkstra, 1968, s. 147)

DIJKSTRAS indvendinger gik blandt andet på, at *udførelsen* af et program, der indeholder  
mange *goto*-kommandoer, blev uoverskuelig, og det betød blandt andet, at det blev meget  
55 sværere at ræsonnere omkring programmet, herunder at finde fejl i det.

Alternativet blev en ny generation af *strukturelle sprog*, der understøttede DIJKSTRAS  
ide om *struktureret programmering* ved at bryde kildekoden op i strukturelle portioner  
med subrutiner, blokke og kontrolstrukturer, hvorved det blev muligt bedre at bevare  
overblikket over kodens udførelse.

60 En anden væsentlig indsigt, som kom ud af *softwarekrisen* i 1960’erne, var gevinsten  
ved *modularisering*: Ved at bryde store systemer op i mindre og selvindeholdte dele, kunne  
man dels genbruge samme løsning i forskellige delproblemer og dels arbejde på at gøre de  
enkelte moduler så robuste og fejlfrie som muligt. Denne tænkning havde sådan set været  
med helt fra computerens fødsel, hvor nogle af de første programmer, der blev udviklet, var  
65 moduler, der kunne udregne matematiske funktioner og konstanter, og som kunne bruges  
ved indsættelse bestemte steder i koden. Oprindeligt betød indsættelse her, at man rent  
fysisk skiftede båndet med maskinkoden ud, men senere blev det en væsentlig del af den  
proces, vi stadig kalder *linking*, hvor det nye program forbindes til de biblioteker af allerede  
udviklede redskaber, som i dag er en stor del af programmeringssprogene.

70 Samlet set kan vi betragte både struktureret programmering og modularisering som  
eksempler på en abstraktionsproces, hvor de mere hardware-nære elementer af program-  
meringen skjules for programmøren igennem en *grænseflade*. Og den ide—om lag af ab-  
straktioner adskilt af eksplíciterede grænseflader—er en af de mest gennemgående ideer i  
75 datalogien. Vi har allerede set den i forbindelse med formel verifikation af software, og vi  
skal møde den igen i opfattelsen af den stak af oversættelser, der omdanner en algoritme til  
en fysisk eksekvering (se afsnit 4.4). Men også i tænkningen om programmeringssprog kan  
denne ide om abstraktion og grænseflade vise sig nyttig, hvilket vi skal behandle i næste  
afsnit.

80 Særligt vigtigt er det, at abstraktion væk fra hardware i retning mod programmørens  
behov og begrebsverden har gjort det muligt at tænke om programmeringssprog og biblioteker  
(API’er) som *abstrakte computere*, der kan tilgås og programmeres ved hjælp af et sæt

af højereordens begreber (interfaces, grænseflader). Og nogle af disse *abstrakte* computere bliver faktisk instantieret i form af *virtuelle maskiner*, hvor en (måske fysisk) computer *emulerer* udførslen af en anden, *virtuel* maskine. Mest udbredt er dette sikkert i form af virtuelle maskiner, der tillader at emulere et operativsystem som et vindue under et andet operativsystem, men den samme teknologi bliver brugt i mange kritiske systemer til at *indeslutte* (encapsulate) potentielt usikre elementer, så de ikke kan komme til at påvirke de kritiske funktioner (se fx arkitekturen beskrevet i Klein m.fl., 2018).

85

## 4.2 Programmeringssprog

Den østrigske datalog NIKLAUS WIRTH er en af de store specialister i design af programmeringssprog, og han beskriver den rolle, som programmeringssprog bør spille således:

90

A language represents an abstract computer whose objects and constructs lie closer to, and reflect more directly, the problem to be represented than the concrete machine. (Wirth, 2008, s. 34)

Denne måde at betragte et sprog som en abstrakt computer, der ligger nærmere programmørens formål end den fysiske maskine, er et meget velvalgt perspektiv, som WIRTH kan anlægge retrospektivt. Men i hele datalogiens historie har problemet med på effektiv vis at give computeren instruktioner til udførelse været en stor udfordring.

95

### Sprog som abstraktioner

I slutningen af 1950’erne — inden for bare få år — blev denne udfordring mødt på fire ganske forskellige måder, betinget af forskellige hensyn (se også figur 2):

100

1. International Business Machines (IBM) var den markedsdominerende producent af hardware, og dengang betragtede man software som en del af den hardware, man betalte penge for se også Haigh, 2002: En IBM computer var med andre ord ikke så meget værd, hvis der fandtes brugbar software til den, og opgaven med at producere denne software tilfaldt hardwareproducenten, *in casu* IBM (Haigh, 2002). Derfor udviklede IBM i 1957, under ledelse af JOHN BACKUS (1924–2007), programmerings-sproget FORTRAN til især deres kunder ved forskningsinstitutioner, hvor der var stort behov for numeriske beregninger og simulationer. FORTRAN stod for ‘FORmu-la TRANslation’, og sproget var udviklet med dette formål for øje: At gøre det ’nemt’ for forskere og andre at programmere deres beregninger og sikre, at de blev udført korrekt og hurtigt.

105

2. Som et delvist modsvar til IBM’s dominans samledes en gruppe akademikere og andre sig om at beskrive det sprog, der i 1958 skulle blive beskrevet som ALGOL (ALGO-rithmic Language), og med ALGOL-60 skulle blive en stor og varig succes. Gruppen bag ALGOL ønskede, at sproget skulle udgøre et akademisk og europæisk alternativ til IBM’s FORTRAN, hvilket blandt andet havde betydning for, at ALGOL var *maskinagnostisk* og blev oversat til mange forskellige arkitekturen, herunder de danske computere DASK og GIER (Vilstrup, 1963). Udviklingen af ALGOL var et fælles anliggende (drevet af *CUDOS*-lignende normer), men blandt de ledende drivkræfter var amerikanerne BACKUS og ALAN PERLIS (1922–1990) og — særligt — danske PETER NAUR (1928–2016), der fik det koordinerende hovedkvarter (og nyhedsbrevet)

110

115

120

flyttet til København (Nofre, 2010). Selve sproget var designet til tunge akademiske beregninger, men også som en forsøgsplads for algoritmiske eksperimenter.

- 125 3. Allerede det følgende år, i 1959, blev et tredje vigtigt programmeringssprog klar, som satte en helt ny retning på programmeringssprogenes formål. Ved det prestigiose Stanford University i Californien var forskningen i kunstig intelligens (AI) inde i en rivende udvikling (se også kapitel 6), og JOHN McCARTHY (1927–2011) var sammen med MARVIN MINSKY (1927–2016) en af de forskere, der spændte over både datalogi og kognitionsvidenskab. Til brug for forskningen i AI udviklede McCARTHY 130 programmeringssproget LISP, som var det første eksempel på et *funktionelt programmeringssprog*, hvor abstraktionen fra den underliggende hardware var gennemført for at stille en abstrakt maskinmodel til rådighed for forskere, der var interesserede i at modellere menneskelig tænkning og gjorde det baseret på den såkaldte  $\lambda$ -calculus, som 135 blev introduceret af logikeren ALONZO CHURCH (1903–1995) i 1930’erne for at besvare nogle af de samme spørgsmål, som vi har diskuteret i relation til ALAN TURING (1912–1954) i kapitel 3.
4. Det fjerde af disse klassiske programmeringssprog tog sit udgangspunkt et helt andet sted. COBOL blev udviklet af det amerikanske forsvar i samarbejde med industrien 140 som en bestræbelse på at gøre programmering til en aktivitet, der ikke krævede *særligt* trænede specialister, men kunne udføres og især forstås af det etablerede arbejds- og kommandohierarki (Sammet, 1978). COBOL er et akronym, som står for CCommon Business-Oriented Language, og det var designet, så det lignede naturligt engelsk langt mere end de meget tekniske notationer i mange andre programmeringssprog. COBOL 145 fandt stor udbredelse i bl.a. bank- og finansverdenen, og der er stadig mange vigtige systemer, der er skrevet i COBOL og kører på bagudkompatible installationer, og en ny specifikation af sproget udkom så sent som 2002, selvom sproget ikke længere indgår i de store IT-uddannelser.

Ved starten af 1960’erne havde man altså mindst fire store og blivende aktører på ’markedet’ for programmeringssprog: FORTRAN, ALGOL, LISP og COBOL. Hvert af disse 150 sprog havde vægtet forskellige kriterier i deres design, og hvert af dem fandt stor udbredelse i sin egen niche (Nofre, Priestley og Alberts, 2014). Fremvæksten af forskellige sprog var så bemærkelsesværdig, at forsiden af tidsskriftet *Communications of the ACM* i 1961 viste programmeringssprogenes babelstårn. Med henvisning til den bibelske fortælling tegnede 155 tårnet et billede af forfængelighed (hurtig udvikling), som gik ud over forståelsen sprogene imellem. Og—måske—lå der også under illustrationen en normativ påstand om, at et (eller i hvert fald en lille håndfuld) programmeringssprog burde være tilstrækkeligt.

## Babelstårnet

Men sådan skulle det ikke gå. I 1972 kunne datalogen JEAN E. SAMMET (1928–2017) se 160 tilbage på et årti, hvor mængden og variationen af programmeringssprog var eksploderet. Mange nye sprog var kommet til, men af det oprindelige babelstårn fra 1961 stod kun 10 tilbage som en stige til at understøtte det væltende tårn (Sammet, 1972, s. 42), heriblandt de fire, vi har beskrevet ovenfor (se også Ensmenger, 2010). Men også nye vigtige ideer var kommet til at yde indflydelse på design af programmeringssprog (se fortsat figur 2):

- 165 5. Abstraktionen fra hardwarens opbygning i retning af programmørens behov tog et stort skridt med programmeringssproget SIMULA, som blev udviklet af OLE-JOHAN

DAHL (1931–2002) og KRISTEN NYGAARD (1926–2002) ved *Norsk Regnesentral* (Norwegian Computing Centre), som var under (delvis) kontrakt fra det norske forsvar (Holmevik, 1994). SIMULA var — som navnet antyder — beregnet til at simulere processer, hvor forskellige *agenter* interagerer med hinanden. Det kan være ventende passagerer ved et busstop, men det kunne også være elementarparklærer i en nuklear reaktion. Der til designede DAHL et *objekt-orienteert* programmeringssprog, hvor hver agent kunne tænkes som et stykke selvstændig kode, der indkapslede både data og kommandoer (Krogdahl, 2005). Og med SIMULA-67 blev sproget forbundet til ALGOL og kom til at udgøre en både teoretisk og praktisk vigtig brik blandt programmeringssprog. 170

6. Hvor mange af de udviklede nye programmeringssprog havde bestemte anvendelser i sigte, var der også nogle sprogudviklere, der fokuserede på undervisningsbehov. Et af de sprog, der kom ud af den tradition var BASIC, som blev udviklet på det meget progressive Dartmouth College af JOHN G. KEMENY (1926–1992) og THOMAS E. KURTZ i 1964. Formålet var at gøre programmering tilgængeligt for de universitetsstuderende, og derfor havde BASIC, som står for 'Beginners' All-purpose Symbolic Instruction Code' et let tilgængeligt instruktionssæt, som nøje svarede til de simpleste algoritmiske kontrolstrukturer med loops og branches. Rækkefølgen var bygget op omkring linjenumre, så **go to** var allestedsværende, og omfaktorisering var en væsentlig forhindring, da der typisk ikke var indbygget mulighed for at justere linjenumre og skaffe mere 'plads' et sted i koden. Til gengæld var sproget let at implementere, og det vandt bl.a. bred udbredelse som det indbyggede programmeringssprog i den uhyre populære *Commodore 64*. 180

Denne blok af programmeringssprog var altså blevet tilføjet 'babelstårnet', da NATO-konferencen i 1968 identificerede nogle af de presserende behov inden for softwareudvikling i de kommende år. Blandt andet pegede man på, at der skulle uddannes mange flere programører, og at deres uddannelse skulle sikre, at de bidrog til at skrive pålideligt software. Samtidig pegede mange af de nævnte dataloger på konferencen også på, at *programmerings-sprog* kunne være en understøttelse af udviklingen af god software, og særligt, at *struktureret programmering* kunne være vejen frem imod klarere og mere modulariseret software, der var nemmere at teste og ikke blev så kompliceret af et flow, der mest af alt mindede om en portion kogt spaghetti. 190

7. Et af de andre sprog, som blev udviklet med både *struktureret programmering* og undervisningsbehov for øje var PASCAL, som blev designet af WIRTH, som efterfølgende også udviklede andre programmeringssprog fx MODULA-2, og udkom i 1970. WIRTH havde deltaget i gruppearbejdet med at udvikle en ny version af ALGOL, men han var blevet frustreret over de mange plenumsdiskussioner og teoretiske uenigheder. I stedet udviklede han PASCAL, som vandt stor udbredelse blandt „computer novices“, der i modsætning til „the high-priests of computing centres“ ikke var bundet af opbygget erfaring og præference for forudgående programmeringssprog (Wirth, 1993, s. 11). PASCAL blev hurtigt udbredt til mange platforme, bl.a. fordi man oversatte kildekoden til en slags halvfabrikata, *p-code*, som enten kunne oversættes videre eller fortolkes. 200
8. Programmeringssproget PROLOG er det mest udbredte eksempel på et *logisk programmeringssprog*. Selve navnet er igen et akronym, denne gang for 'PROgrammatiON 210

215

en LOGique', og som dette antyder, er ophavet til PROLOG fransk-canadisk (Colmerauer og Roussel, 1993, s. 2). PROLOG var resultatet af arbejde med at bygge en såkaldt *interaktiv prover* til naturlige sprog. Formålet var at få computeren til at 'forstå' og 'ræsonere logisk' omkring fx *syllogismen* (se figur 1 og kapitel 6). Og i 1972 blev PROLOG implementeret i varianten ALGOL-W, som WIRTH havde udviklet. Hovedmændene bag designet af PROLOG var ALAIN COLMERAUER (1941–2017) og ROBERT KOWALSKI, og styrken af PROLOG skyldes både det interaktive interface og den effektive *heuristiske pruning*, som siden også er blevet udviklet kraftigt.

<b>Input:</b>
Every psychiatrist is a person. Every person he analyses is sick. Jacques is a psychiatrist in Marseilles. Is Jacques a person? Where is Jacques? Is Jacques sick?
<b>Output:</b>
Yes. In Marseilles. I don't know.

Figur 1: Eksempel på en original session i PROLOG (Colmerauer og Roussel, 1993, s. 7).

220

Selvom der gik en udvikling i retning af at gøre sprogene mere tilgængelige for brugerne, så var der også et modsatrettet behov for programmeringssprog, der kunne levere effektiv og sikker kode til især såkaldt *system software*. Dette omfattede de mest centrale del af et computersystem som fx et *operativsystem*, der står for håndteringen af hukommelse, filer og andre eksterne enheder. Og selvom fx PASCAL også blev brugt til *system software*, så blev denne rolle i høj grad overtaget af andre sprog og især af C.

230

9. DENNIS RITCHIE (1941–2011) designede programmeringssproget C i begyndelsen af 1970’erne, mens han arbejdede for *Bell Laboratories*, som er en amerikansk forskningsinstitution, oprindeligt grundlagt af opfinderen ALEXANDER GRAHAM BELL (1847–1922). RITCHIE var sammen med KEN THOMPSON centralt involveret i at designe og implementere UNIX, og C blev det fortrukne programmeringssprog til implementationen, hvilket gjorde UNIX portabelt til mange platforme i takt med, at C-*compileren* blev porteret. C var en videreudvikling af et tidligere programmeringssprog kaldet *B*, og et af de fascinerende elementer i tilblivelsen af disse sprog var, at THOMPSON var i stand til at implementere en *B-compiler* i sproget *B*. Dermed blev det muligt at 'bootstrappe' compilere ved at implementere en (lille) kerne af sproget i sig selv (Ritchie, 1993, s. 4). Og kombineret med en *preprocessor* og et stadigt voksende antal standardbiblioteker med udvidelser blev C hurtigt til et potent og udbredt programmeringssprog. Det førte også til, at C blev *standardiseret* i 1982, og siden er der kommet forskellige udviklinger i sproget.

235

10. En af de væsentligste udvidelser af C er det *objekt-orienterede programmeringssprog* 240 C++, som den danske datalog Bjarne STROUSTRUP designede fra slutningen af 1970’erne. STROUSTRUP beskriver selv sine designkriterier således:

C++ was designed to provide SIMULA’s facilities for program organization together with C’s efficiency and flexibility for systems programming. It was intended to deliver that to real projects within half a year of the idea. 245  
It succeeded. (Stroustrup, 1993, s. 1)

STROUSTRUP havde kendskab til SIMULA fra sin uddannelse i Aarhus, og han tilføjede i første omgang klasser til C ved at programmere en *preprocessor*, inden at det egentlige C++ så dagens lys i 1985 og hurtigt blev et meget anvendt værktøj til *objekt-orienteret programmering*. 250

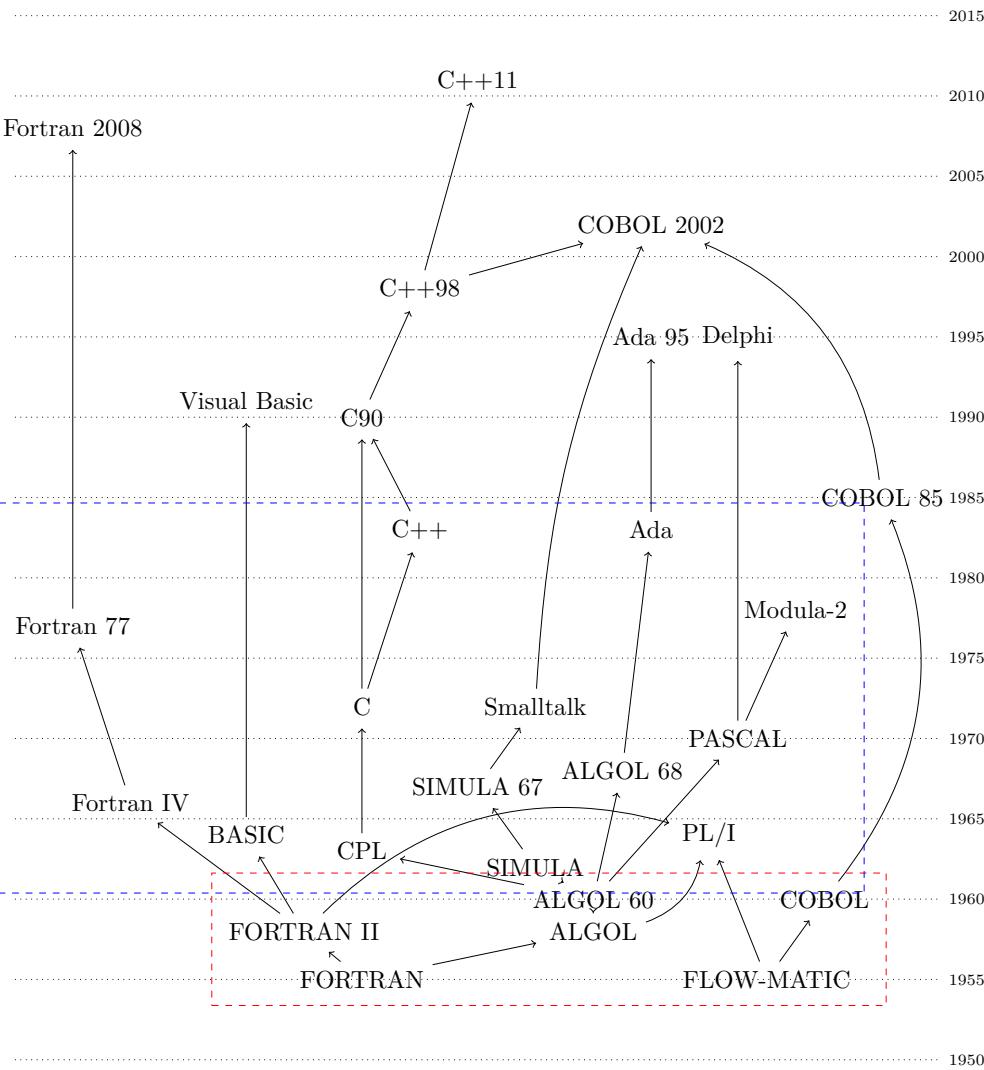
Navn	Årstal	Design	Paradigme
ALGOL	1958	BACKUS, NAUR, PERLIS et al	I
BASIC	1964	KEMENY og KURTZ	I
C	1972	RITCHIE v/ Bell Labs	P/S
C++	1985	STROUSTRUP	M/P/S/F/OO
COBOL	1959	US DoD	I/P/(OO)
LISP	1958	McCARTHY	M/F/P
FORTRAN	1957	BACKUS v/ IBM	I
PASCAL	1970	WIRTH	I/S
PROLOG	1972	COLMERAUER og KOWALSKI	L
SIMULA	1962	DAHL	M/P/S/OO

Figur 2: Oversigt over nogle af de vigtigste programmeringssprog. Årstal henviser til første design, mange af sprogene er udviklet løbende og er stadig i anvendelse. Forkortelserne er M: multi-paradigmatisk, I: imperativt, P: procedurelt ( $P \subset I$ ), S: struktureret, F: funktionelt, L: logisk, OO: objekt-orienteret.

## Hvad gør et godt sprog?

Ud fra denne korte gennemgang af 10 vigtige programmeringssprog (se også figur 2) kan man gøre sig en række observationer.

For det første kan man se, at den allerførste generation af programmeringssprog som især FORTRAN og ALGOL kom til at udgøre, hvad man kan kalde en *sprogstamme*: Rigttig mange af de efterfølgende programmeringssprog er blevet til som videreførelser af disse tidlige programmeringssprog. Man taler endda om FORTRAN-ALGOL-C-familien af programmeringssprog (se figur 3), hvortil mange af de sprog, vi bruger i dag, stadig hører. 255



260

Figur 3: Relationerne mellem programmeringssprog i FORTRAN-ALGOL-C-familien. Der er mange flere sprog og sprogvarianter, men de fleste af de vigtigste er med her. Den røde firkant indrammer de store sprog i 1. generation (1950'erne), og den blå firkant indrammer store sprog i 2. generation.

Hvis man følger sprogenes udvikling over tid kan man også se, hvordan de gensidigt påvirker hinanden, så meget, at flere af de nyere sprog er opstået som fusioner mellem tidligere sproggrene. Det er mest tydeligt tilfældet med C++, der eksplicit blev skabt for at kombinere egenskaber fra SIMULA (programorganisering) og C (effektivitet).

265

Endvidere kan man se, at der har været forskellige forstillinger om sprogenes formål og foretaget forskellige vægtninger imellem hensyn, når nye sprog er blevet designet. Behov i uddannelse var en vigtig motivation for BASIC og PASCAL (Goosen, 2008). Og relationen mellem naturligt sprog og programmeringssprog var en tilbagevendende overvejelse og forskningsinteresse bag såvel COBOL som PROLOG (Good og Howland, 2017).

270

Et af de vigtige skel i forhold til programmeringssprogs formål handler om vægtningen mellem kontrol og understøttelse for programmøren. Og deri ligger også en god del af diskussionen om programmørens status og relation til brugeren (se også kapitel 7). Man

kan indføre et skel imellem *lavniveausprog*, som er designet til at give programmøren en høj grad af kontrol over, hvordan programmet oversættes til maskinkode i et trade-off mod større besvær med programmeringen. Denne type sprog vil kunne give meget effektiv softwareudførelse, men med større investeringsbehov i udviklingen. Nogle typiske eksempler på *lavniveausprog* er C og FORTRAN. Modsætningen hertil vil være *højniveausprog*, som skjuler mange implementationsdetaljer i et trade-off mod ydelse af koden. Eksempler på denne type sprog er BASIC, LISP og PROLOG. Disse sprog er typisk *fortolkede sprog* og tilbyder fx hukommelseshåndtering i form af *garbage collection*. Jo højere niveauet af programmeringssproget, des sværere bliver det at forudsige ydelsen af den producerede software, hvilket vil være et stort problem for realtids- eller ydelseskritiske systemer. 275

Afvejningen af hensyn hænger også nært sammen med en afvejning af anvendelsesområder. I en direkte sammenligning af PASCAL og C identificerede Feuer og Gehani (1982) fire typiske anvendelser: 1. „business data processing“, 2. „scientific programming“, 3. „programming of operating systems“, og 4. „programming for system utilities“ (Feuer og Gehani, 1982). På de fleste parametre fandt denne sammenligning, at sprogene var lige (u)egnede til 1, hvor COBOL dominerede, 2, hvor FORTRAN stadig dominerede, og 4. Men på programmering af *operativsystemer*, var C klart det mest effektive, hvilket jo også netop var RITCHIES motivation bag sproget. 280

Hvis man ønsker at karakterisere et succesfuldt sprog kan nogle af de parametre, som eksemplerne ovenfor har antydet, være 285

- Hvor let det er at *lære* sproget. Det har været et designkriterium for fx BASIC og PASCAL.
- Hvor let det er at *udtrykke* ting i sproget, når man først har lært det—vi taler sommetider om, at sproget er *kraftfuldt*. Det har været et designkriterium for fx C og LISP.
- Hvor let det er at *implementere* sproget, hvilket også kan sige noget om, hvor hurtigt det kan nå udbredelse på forskellige platforme. Det var en del af designsuccessen for BASIC. 295
- Hvor let det er at *genererer god maskinkode* (målt på fx hurtighed eller størrelse). Det var særligt en motivation for fx FORTRAN.
- Hvor let det er at finde *opbakning* fra vigtige spillere i markedet til sit sprog. En stor del af successen af COBOL kan henføres til dette kriterium, og siden hen har andre sprog som VISUAL BASIC eller JAVA fundet opbakning fra nogle af de helt store software- og hardwareproducenter. 300
- Hvor let det er at nå nem og billig *udbredelse* til et stort publikum. Her er både PASCAL og JAVA gode eksempler på, at det kan lykkes, hvis man stiller programmeringssproget frit til rådighed.

Nogle af de egenskaber, som sprog værdsætter meget forskelligt er fx *rekursion*, *stak-håndtering*, *statiske typer* og *garbage collection*: Nogle sprog kan ikke fungere uden en af disse, mens andre sprog slet ikke indeholder dem. Og mange diskussioner om programmeringssprogs relative meritter handler om præferencer blandt disse egenskaber, sprogenes syntaks og deres primære anvendelsesområde. To af de væsentligste balancegange, som 310

315 sprogdesignere skulle gå mellem hensyn til programmøren og effektivitet i oversættelse og afvikling, var *syntaks* og *typehåndtering*.

For *syntaks* var balancen typisk mellem formel elegance og effektiv evaluering, og mange programmeringssprog opfandt ideo-synkratiske syntaktiske konstruktioner. Men alligevel blev noget nedarvet, især igennem *sprogstammen*, så man kan typisk lettere læse og forstå 320 nært beslægtede programmeringssprog — ligesom tilfældet er for naturlige sprog. At syntaks også afspejler en abstraktion fra hardware i retning af programmøren kan man fx se ved at betragte forskellige sprogs syntaks for *increment*: operationen, der lægger 1 til en variabel. FORTRAN, som netop var designet omkring elegant omgang med formler, vil forvente, at man skriver følgende:

325 `i = i+1`

Det er jo super klart, hvad der skal foregå, men det er ikke sikkert, at det giver maksimalt effektiv kode. I C, derimod, opfandt THOMPSON følgende syntaks:

330 `i++;`

Indholdet er det samme, men i C-koden ved *compileren* på forhånd, at der er tale om et 335 *increment*, og den kan oversætte det til den tilsvarende maskininstruktion, der altid findes i instruktionssættet. Det kunne FORTRAN-*compileren* også *udlede*, men her var det givet, og C-syntaksen fjerner jo også mulighed for visse skrivefejl, omend den forvirrer mange, første gang, de ser den.

Dette er et eksempel på, hvordan *syntaks* kan bruges til at understøtte robust og hensigtsmæssig kode. Et eksempel kunne også være følgende lille programstump, der viser en 340 lille, lidt ikke-intuitiv justering af koden, der gør den lidt mere robust.

345 `unsigned long factorial(unsigned char n) {  
 if (1==n) {  
 return 1;  
 } else {  
 return n*factorial(n-1);  
 }  
}`

350 For hvor lighedstegn jo i C bruges til tildeling, er syntaksen for lighedstest et dobbelt ligheds-tegn. Dette er måske en uhensigtsmæssig konvention for den uindviede, men kodestumpen gør, at, hvis man ellers skriver sine tests konsekvent på den måde, vil en skrivefejl af `=` for `==` blive fanget af *compileren*.

Den anden væsentlige design-betræftning, som vi vil behandle kort her er *typehåndtering*. 355 Typer er jo ikke noget, der findes på hardware-niveau, så de hører til i den abstraktion, som programmeringssproget stiller til rådighed. Og samtidig viste (statiske) typer sig at være både en velsignelse og en forbundelse for programmøren: Typetjek er gode til at fange mange fejl i programmeringen, men statiske typer er træls at arbejde med, når man fx skriver generiske rutiner til sortering. Og her valgte nogle sprog, som C, så den løsning at 360 stille både typetjek og muligheden for at omgå det til rådighed for programmøren. Deraf kommer meget af C's højt værdsatte effektivitet og fleksibilitet, men også en meget stor del af bugs i C skyldes fejlagtig *recasting* og dårlig håndtering af stakke.

## 4.3 Programmeringsparadigmer

En måde at organisere de mange programmeringssprog, som er blevet udviklet, på, er at gruppere dem efter deres basale abstraktioner. Det fører til en inddeling i det, man typisk 365 kalder *programmeringsparadigmer*, som vi her kort skal introducere og diskutere.

Man skelner overordnet mellem to typer af programmeringssprog, nemlig *imperative* og *deklarative* sprog (se figur 4). De *imperative* sprog er karakteriseret ved, at man som 370 programmør giver kommandoer, mens programmører, der bruger *deklarative* sprog, i stedet deklarerer antagelser og mål for afviklingen af programmet. Denne skelnen fortjener at blive udfoldet og forklaret, og det vil vi gøre ved yderligere at underinddele hver kategori.

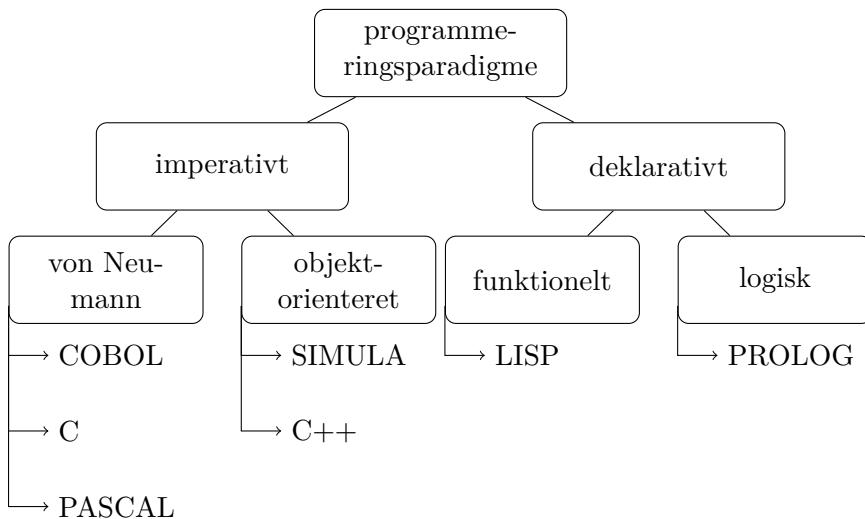
Blandt de *imperative* sprog skelner man typisk imellem *procedurelle* sprog — også kaldet 375 *von Neumann-sprog* — som er den klasse af sprog, der er tættest baseret på hardware-paradigmet (se også kapitel 2). De arbejder altså med en program-tilstand, som ændres over tid i takt med, at kommandoer styrer program-flow og manipulerer hukommelsen. Disse kommandoer udgør kontrolstrukturer og omfatter bl.a. assignment, I/O-operationer 380 og procedurekald. Dermed ligner denne type sprog ganske meget karakteriseringen af *algoritmer* (se kap. 3), og de har *procedurer* som deres naturlige abstraktioner. Mange af de klassiske programmeringssprog, som vi har gennemgået ovenfor, falder i denne kategori, fx COBOL, PASCAL og C. Nogle af de andre sprog er også *imperative* uden at have en så god understøttelse af *modularisering* igennem procedurekald, fx ALGOL, FORTRAN og BASIC.

Den anden store klasse af *imperative* sprog udgøres af de *objekt-orienterede* sprog. Disse 385 sprog er baseret på de måder, relationer består mellem begreber, og på abstraktioner fra den måde, menneskelig kommunikation og interaktion med omverdenen foregår på. De er karakteriseret ved en bestemt form for *modularisering*, hvor både data og operationer er *indkapslet* i *objekter*, hvorved intern information i form af attributter og operationer i form af håndtag er skjult for resten af programmet. I stedet foregår en afvikling af programmet ved, at objekter interagerer igennem beskeder. I *objekt-orienterede sprog* er objekterne typisk beskrevet som instanser af mere generelle klasser (begreber) med mulighed for ned- 390 arvning og specialisering. Blandt de klassiske programmeringssprog er især SIMULA og C++ eksempler på *objekt-orienterede programmeringssprog*.

Også iblandt de *deklarative* sprog skelner man typisk mellem to store underklasser. Den ene og mest righoldige underklasse udgøres af *funktionelle* programmeringssprog, som er baseret på funktioner som den naturlige abstraktion. Det betyder, at funktioner er de primære 395 objekter, ligesom tal og lister typisk er det. I modsætning til de *imperative* sprog er *funktionelle* sprog i principippet atemporelle, og afvikling foregår igennem anvendelse (*applicering*) af funktioner. Det oprindelige 1.-generationssprog, hvor det *funktionelle* programmerings- 400 paradigm blev etableret, er LISP, men siden er der kommet mange forskellige varianter til, ligesom flere af de *imperative* sprog også muliggør *funktionel* programmering.

Den sidste vigtige underklasse udgøres af *logiske* programmeringssprog, der er en meget 405 ren form for *deklarative* sprog, baseret på erfaringer fra ATPautomatiserede bevisgeneratører (se kap. 3). Disse sprog har, ligesom de *funktionelle* sprog, deres oprindelse i AI, hvor de blev brugt både til modellere logisk ræsonnement og processering af naturlige sprog. De basale entiteter er her aksiomer, inferensregler og forespørgsler fra brugerne, og en afvikling svarer til en systematisk afsøgning fra en mængde facts under brug af aksiomerne og inferensreglerne og en søgeheuristik med effektiv pruning af søgeræsetræet (se også figur 1). Eksemplet på et *logisk* programmeringssprog iblandt de 10 klassiske sprog er PROLOG, men der er inden for de seneste tiår kommet mange andre og meget stærke *automatiske*

<sup>410</sup> bevisgeneratorer på markedet, som også falder i denne kategori.



Figur 4: Skematisk oversigt over de centrale programmeringsparadigmer.

<sup>415</sup> Foruden de 'rene' eksemplarer i hvert programmeringsparadigme, findes der også forskellige hybrider, der kombinerer på tværs af kategorierne. For eksempel er C++ et *multiparadigmatisk sprog*, idet det omfatter både det *imperative* sprog C og *objekt-orientering*. Det samme gør sig gældende for en række af de såkaldte *script-sprog* som fx PERL og PYTHON.

<sup>420</sup> Vi omtaler disse forskellige programmeringsparadigmer netop som *paradigmer*, fordi de ligesom *kuhnske paradigmer* udgør en hel tilgang — en verdensanskuelse (se også kapitel 2). Det er ikke kun et spørgsmål om forskellige syntakser og mindre styrker og svagheder ved det enkelte programmeringssprog, der skelner programmeringsparadigmerne. Det er derimod spørgsmål om hele problem-opfattelsen og problem-løsningen, der ser forskellig ud fra forskellige paradigmers verdenssyn. Hvis man fx *tænker objekt-orienteret*, så *består* den verden, vi skal modellere med computersystemer, af objekter, som er instanser af begreber (klasser). Programmeringsparadigmet er med andre ord det filter, som vi anskuer og modellerer verden igennem. På den måde svarer programmeringsparadigmer jo udmærket til THOMAS KUHNS (1922–1996) beskrivelse, og man kan også argumentere for, at dataloger netop igennem deres uddannelse socialiseres ind i en disciplinær matrix for hvert programmeringsparadigme: Der er nogle klassiske og typisk problemer, som de forskellige sproglærere kan løse, ligesom man igennem lærebøger lærer at anvende typiske teknikker til problemløsningen. Og mange af lærebøgerne og sprogbeskrivelserne har eksemplarisk status og er blevet til små ikoner, som fx BRIAN KERNIGHAN og RITCHIES beskrivelse af C (Kernighan og Ritchie, 1978). Og endelig vil mange programmører måske også opleve noget i retning af en *omvendelse*, når de første gang møder et programmeringsparadigme, der rækker ud over det *procedurelle*. Dermed er der mange lighedspunkter, både i det altomfattende verdenssyn, den sociale disciplinering og den psykologiske oplevelse, som berettiger og måske fordrer, at vi sammenligner programmeringsparadigmer med *kuhnskuhnske paradigmer*.

<sup>430</sup>

Men der er også nogle vigtige forskelle, som har med programmeringens særlige status som vidensfelt at gøre. For alle disse programmeringsparadigmer er *sameksisterende*. Og selvom det kan være svært at vænne sig til et andet programmeringsparadigme, er der fuld

oversættelighed imellem dem, både teoretisk idet alle sprogene har præcist samme udtrykskraft (de er alle *Turing-komplette sprog*) og fordi de alle afvikles på sammenlignelige og emulerbare arkitekturer. Så hvor KUHN forestillede sig, at paradigmmer (i *realvidenskaberne*) er *inkommensurable*, så er programmeringssprog — i kraft af, at de er formelle systemer, som mennesker har bestemt — anderledes stedt. For inden for *formalvidenskaberne* er der nærmere tale om, at paradigmmet definerer *vores* måde at anskue verden på, når vi foretager *modellering* og er klar over, at vores perspektiv er ufuldstændigt. 440

Alligevel kan der være god grund til at reflektere over programmeringsparadigmer som verdensanskuelser (Wernick og Hall, 2004). For det første sætter det tydeligt fokus på, at forskellige paradigmmer har forskellige styrke — og at disse styrke skal ses i forhold til programmeringen og ikke i forhold til virkeligheden. For det andet viser parallelten os, at der altid er visse elementer, vi ikke kan medtage i vores *modellering* (se også kapitel 5), hvorfor vi gør klogt i at reflektere over vores valg af paradigmme og dets relative blinde vinkler. 450

## 4.4 Fra algoritme til afvikling

Det betaler sig — både fra et datalogisk og et videnskabsteoretisk standpunkt — at have et billede af den proces, hvormed algoritmer kan omsættes til afvikling på en computer (se også Bryant og O'Hallaron, 2011). 455

Lad os forestille os, at vi har følgende opskrift:

**Algoritme 4.1**

For at beregne  $\frac{c(c+1)}{2}$  skal du lægge tallene fra 1 til  $c$  sammen.

Denne opskrift ligner det, som ROBIN K. HILL karakteriserede som en algoritme (se kapitel 3), ved at den opfylder at være 1. *endelig* (den er skrevet op på mindre end to linjer her), 2. *abstrakt* (den gælder ikke kun for fx  $c = 4$ , men for generelle  $c$ ), 3. *effektiv* (den kan udføres mekanisk, i hvert fald, hvis addition ikke kræver mere end 'tankeløs handlen'), 4. *sammensat kontrolstruktur* (den giver instruktioner om sekvensen af operationer, der hver især er sammensatte) og 5. givet *imperativt* (hver af operationerne er givet som ordrer til udførelse). Desuden har opskriften et 6. *formål* (nemlig at udregne  $\frac{c(c+1)}{2}$ ), men 7. den har ikke eksplisit angivne *betingelser*. Dermed opfylder opskriften alle kriterier i HILLS første definition af en algoritme, men den mangler at opfylde et enkelt i hendes anden, mere raffinerede definition. 460

Opskriften er endvidere en *korrekt* algoritme, hvilket man kan bevise ved et simpelt matematisk induktionsbevis over  $c$ , der beviser ligheden

$$VS(c) := \frac{c(c+1)}{2} = \sum_{x=1}^c x =: HS(c). \quad (1)$$

Som induktionsstart indser vi, at for  $c = 1$  er  $\frac{1(1+1)}{2} = 1$ . Og som induktionsskridt kan vi

opskrive

$$\begin{aligned} VS(c+1) &= \frac{(c+1)((c+1)+1)}{2} = \frac{c(c+1) + 2(c+1)}{2} = \frac{c(c+1)}{2} + (c+1) \\ &= \sum_{x=1}^c x + (c+1) = \sum_{x=1}^{c+1} x = HS(c+1), \end{aligned}$$

hvor linjeskiftet svarer til brug af induktionshypotesen.

Denne opskrift kan implementeres på mange forskellige måder og i mange forskellige formalismere og programmeringssprog. Lad os betragte følgende lille stump C-kode til beregning af højresiden i (1):

Kildekode 4.1: En implementation af algoritme 4.1

```
475   x=c; y=0;
      while (x>0) {
        y+=x; x--;
    }
```

Denne kode kunne umiddelbart se ud til at være en korrekt implementation af algoritme 4.1: For hvert gennemløb af while-loopet forøges  $y$  med værdien af  $x$ , hvorefter  $x$  tælles ned fra  $c$  til 0, så når vi når slutningen, ser  $y$  ud til at være summen af  $x$  for  $x$  mellem 1 og  $c$ , og altså  $y=c(c+1)/2$ .

Men hvad nu, hvis  $c=-1$  ved starten af programstumpen? Så bliver loop-betingelsen aldrig opfyldt, og vi får  $y=0$  ved slutningen af koden. Dette kunne sådan set godt anses som korrekt opførsel, idet formlen stadig er opfyldt. Men hvis man havde sat  $c=-2$ ; ved starten af koden, havde man stadig fået  $y=0$ , og det er klart, at koden ikke altid giver korrekte resultater for negative værdier af  $c$ . Men faktisk er den indvending ikke så meget rettet mod kildekode 4.1 som imod algoritme 4.1, hvor vi netop ikke havde opstillet nogen betingelser på  $c$ . Og antagelsen ligger faktisk implicit i formel (1). Så for at algoritme 4.1 virkelig skal opfylde den raffinerede definition hos HILL, skal der netop gøres den antagelse, at  $c$  er et positivt (hel)tal.

Hvis vi vender tilbage til figur ?? (se figur 6, nedenfor), beskrev den, hvordan et program som kildekode 4.1 (en implementation i kildekode) er en specificering af den abstrakte algoritme til et konkret programmeringssprog ①. Og en af egenskaberne ved C er, at det er et typet sprog med forskellige heltalstyper. Specielt er der ikke i C indbygget understøttelse af vilkårligt store tal, så allerede med valget af programmeringssprog indtræder en begrænsning: Der vil være værdier af  $\frac{c(c+1)}{2}$ , som ikke kan repræsenteres i de indbyggede tal-typer.

Den næste specificering ② i figur 6 handler om, at oversættelsen fra kildekode til maskinkode er betinget af den arkitektur, der oversættes til. Så hvordan ville programmets betydning se ud, hvis der inden starten af stumpen var foretaget en erklæring som `int x, c=16; char y;` og jeg ønskede at compile programmet til min x86-arkitektur? Så vil der ske et overrun i  $y$ , fordi en `char` ifølge specifikationen af C er en byte-variabel, som på denne arkitektur er 8-bit og kun kan indeholde heltal i intervallet  $[-128, 127]$ . Vi vil derfor få  $y=-120$ , selvom  $c*(c+1)/2==136$ . Denne fejl skyldes altså en begrænsning, som *compile-ren* og den underliggende arkitektur lægger ned over problemet (her i form af et overrun), så maskinkoden er altså ikke en korrekt implementation af programmet og algoritmen uden tilføjelse af yderligere betingelser. Og hvis min arkitektur havde været en anden, havde problemet ligget et andet sted (men stadig eksisteret); figur 5 viser, hvordan størrelsen af

	DEC PDP-11	Honeywell 6000	IBM 370	Interdata 8/32
	ASCII	ASCII	EBCDIC	ASCII
char	8 bits	9 bits	8 bits	8 bits
int	16	36	32	32
short	16	36	16	16
long	32	36	32	32
float	32	36	32	32
double	64	72	64	64

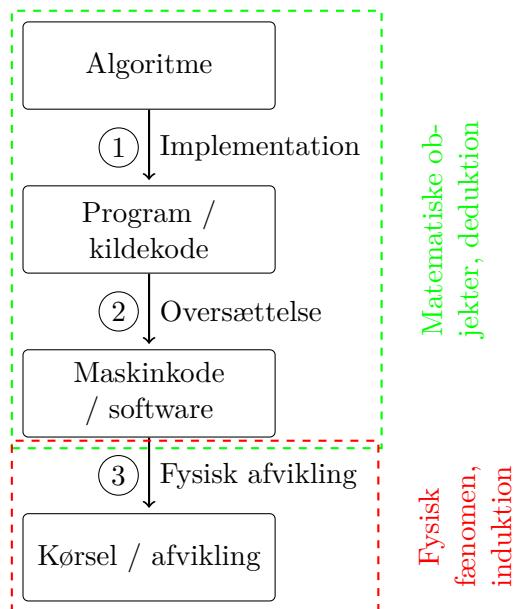
Figur 5: Størrelse af forskellige variabel-typer på forskellige fremherskende 1978-arkitekturen som beskrevet i Kernighan og Ritchie (1978, s. 42).

Endelig viser (3), at maskinkoden skal bringes til fysisk afvikling på en konkret computer. For at denne kørsel af maskinkoden skal give korrekte resultater, forstået som resultater, der er i overensstemmelse med algoritmens specifikation, er det selvfølgeligt afgørende, at de ovenfor omtalte oversættelser (specificeringer), som fører til maskinkoden er korrekte. Men derudover tilstøder der en yderligere komplikation, som hænger sammen med overgangen fra matematiske, immaterielle objekter til afvikling på en fysisk, materiel maskine. For hvor matematikken forholder sig til beviselig korrekthed inden for et *lukket system*, så er afviklingen på en fysisk computer del af et *åbent system*. Det betyder konkret, at vi ikke kan være sikre på, at computeren ikke har produktionsfejl, at tilfældige forbi passerende ikke afbryder beregningen, og at den kosmiske baggrundsstråling ikke påvirker de interne registre og skifter værdien af en bit. Alle disse — og utalligt mange flere — faktorer er ikke modelleret med i vores lukkede system, og vi kan ikke udelukke dem med formelle metoder (deduktive beviser). I stedet må vi forlade os på, at de er sjældne og bruge realvidenskabelige, empiriske argumenter til at overbevise os om det.

515

520

525



Figur 6: Figur ?? med yderligere bemærkninger.

## 4.5 Fuld fart frem

I august 2009 omkom MARK SAYLOR (-2009), hans kone, datter og svoger i en trafikulykke i San Diego. Denne ulykke var imidlertid helt særlig af en række omstændigheder: Føreren af bilen, SAYLOR, var ansat ved California Highway Patrol, der har ansvaret for sikkerheden på statens veje. Bilen, de forulykkede i, var en lånebil af mærket *Toyota Lexus 2009 ES350*. Ulykken skyldtes, at bilen pludseligt accelererede uventet og ustoppeligt, så de bragede ind i autoværnet. Og imens ulykken stod på nåede svogeren at ringe 911, så hele ulykkesforløbet blev optaget på alarmcentralen, og efterfølgende har mange kunnet høre med på SAYLOR-familiens panik og ulykke. Sagen opnåede hurtigt stor offentlig bevågenhed, dels på grund af de dramatiske kilder, dels fordi denne slags ustoppelig acceleration ikke var helt uhørt, og dels fordi føreren af bilen var en ansvarlig og billyndig myndighedsperson.

Faktisk havde det relevante bureau i US, National Highway Traffic Safety Administration (NHTSA), længe oplevet problemer med ustoppelig acceleration (Anderson, 2016). Og i 1986 havde NHTSA foretaget en undersøgelse, der slog fast, at denne slags fejl skyldtes, at føreren havde fejlbetjent bilen og i panik og forvirring havde hamret speederen i bund i stedet for at stå på bremsen. Denne forklaring var blevet brugt til at bortforklare sporadiske uheld og en række mere sammenhængende tilfælde med biler af Audi-mærket. Men i 2009 eksploderede antallet af indberetninger om problemer med Toyotaer, særligt Lexus ES350; og mange flere af tilfældene omfattede personskade og endda død som i SAYLORS sag. Det var nu en ganske alvorlig ting, som både kunne føre til omfattende tilbagetrækning af modeller fra Toyota og til omfattede søgsmål om erstatning og evt. ansvarsplacering (Finch, 2010).

Men før man kunne komme ret meget længere i sagen, var man nødt til at finde ind til, hvori fejlen bestod: kunne den reproduceres og kunne den forklares? Og i det amerikanske system var der tre forskellige aktører, der indgik i denne fejlfinding: Toyota, NHTSA og repræsentanterne for de skadeslidte, der lagde sag an mod Toyota. På grund af den pludselige stigning i antallet af tilfælde og med det udgangspunkt, at panik-hypotesen ikke virkede særligt overbevisende eller betryggende, rettede noget af fokus sig på det forhold, at alle moderne biler nu brugte et Electronic Throttle Control System (ETCS), som altså er en kombination af hardware og software, der ligger 'imellem' førerens kontrol og bilens opførsel. Alle havde de deres eget udgangspunkt: Toyota havde jo fuld kendskab til bilernes mekanik, hardware og software, men havde ikke interesse i at dele denne viden alt for åbent. NHTSA havde ansvaret for at teste bilerne, men havde allerede i 1986 lagt sig fast på en bestemt forklaring, nemlig førerfejl. Og sagsøgerne havde næsten ingen adgang til hardware og software, men havde en klar hypotese om, at fejlen skyldtes ETC-systemet.

I forbindelse med en sag om endnu en ulykke i Oklahoma i 2013 blev datalogen PHIL KOOPMAN engageret som ekspertvidne for sagsøgerne i en erstatningssag imod Toyota (Cummings, 2016). Og KOOPMANS argument for retten er en af de få kilder, vi har til sagens tekniske dimensioner, for efterfølgende er der blevet indgået fortrolighedsklausuler i de kompromisser, der blev indgået.

I forbindelse med sagerne blev National Aeronautics and Space Administration (NASA) bedt om at lave en test af biler af samme mærke, for at forsøge at finde frem til fejlen. I rapporten, som NASA udarbejdede i 2011, kan man blandt andet læse:

Due to system complexity [...] and the many possible electronic hardware and software system interactions, it is not realistic to attempt to 'prove' that the [Electronic Throttle Control System] cannot cause [Unintended Accelerations]. Today's vehicles are sufficiently complex that no reasonable amount of analysis or testing can prove electronics and software have no errors. Therefore absence of proof that the ECTS-I caused a UA does not vindicate the system. (Citeret fra Anderson, 2016, s. 1428)

570

Og endvidere:

Proof of the hypothesis that the Electronic Throttle Control System] caused the large throttle opening [Unintended Acceleration]s as described in submitted [Vehicle Owner Questionnaires] could not be found with the hardware and software testing performed. [...] Because proof that the ETCS-i caused the reported UAs was not found does not mean it could not occur. However, the testing and analysis described in this report did not find that [ETCS] electronics are a likely cause of large throttle openings as reported in the VOQs. (Citeret fra Koopman, 2014)

580

585

Men den amerikanske transportministers udlægning af NASA's ekspertvurdering er helt modsat. RAY LAHOOD sagde i den tilhørende pressemeldelse:

We enlisted the best and brightest engineers to study Toyota's electronics systems, and the verdict is in. There is no electronic-based cause for unintended high-speed acceleration in Toyotas. (*U.S. Department of Transportation Releases Results from NHTSA-NASA-Study of Unintended Acceleration in Toyota Vehicles* 2011)

590

Fra KOOPMANS analyse får man et indtryk af kompleksiteten af den software, der ligger i ETCS: Den består af omkring 256.600 linjer aktiv C-kode (excl. kommentarer) og omkring 39.500 headers. Koden afvikles dels på en primær CPU og en proprietær Monitor Chip med tilhørende proprietær software. Dertil kommer, at softwaren ikke i praksis udføres lineært, idet den er ansvarlig for handlingsbaserede forstyrrelser, fx i forbindelse med registrering af pedalernes tilstand og et fail-safe overvågningssystem, der er indbygget for at monitorere det primære system og gøre ind, hvis noget går galt. Professor KOOPMAN pegede i sit indlæg på en række potentielt farlige problemer med den software, som han altså ikke kunne komme til at studere i alle detaljer. Blandt de primære mistanker var en stribe klassiske syndere i kritiske systemer: Problemer omkring samtidighed (fx race-problemer), overskrivning af globale variable, og stack overflows.

595

600

Som både NASA og KOOPMAN sådan set var enige om, så kunne ingen mængde af prøvning fritage ETCS for mistanken om fejl. Men omvendt havde man heller ikke kunnet påvise fejl af den kritiske type i nogle af de kontrollerede forsøg, man havde anstillet med andre biler af samme mærke. Men disse eksperimenter havde jo — trods forsøg på at gøre dem så realistiske som muligt — ikke været gjort 'in the wild'. Så hvad skal man så stille op?

605

Der er nok (mindst) to perspektiver at se på dette spørgsmål ud fra. Det ene er et juridisk perspektiv: I retssagen skulle det *bevises*, at producenten var skyld i dødsfaldene, og det var der trods alt ikke nogen 'smoking gun', der beviste. Og på den måde kom adskillige bevisbegreber i spil i retssagen: et matematisk og absolut bevisbegreb, et begreb

610

om formel verifikation af software og hardware, og det juridiske begreb, der i USA er baseret  
615 på ’beyond reasonable doubt’ (se også MacKenzie, 2004). Retssagerne endte i stedet med, at der blev indgået hemmelige forlig med tavshedsklausuler, som gör, at vores viden stadig er begrænset. Men man kan godt antage, at Toyota så selv betragtelige forlig som en mere kosteffektiv løsning på sagen end omfattende og højtprofilerede tilbagetrækninger af modeller fra hele verden (Gokhale, Brooks og Tremblay, 2014).

Men der er også et datalogisk perspektiv at antage, som handler om, hvordan man sikrer sine kritiske systemer mod uforudsete fejl. Og her er der to oplagte muligheder: Vi kan enten forsøge at *bevise* korrektheden af vores kritiske systemer igennem formel verifikation. Men det er oftest meget omkostningsfuldt, og vi løber alligevel ind i de begrænsninger, som vi allerede har diskuteret i forhold til overgangen fra lukkede til åbne systemer. Eller vi kan forsøge at underkaste vores kritiske systemer omfattende, systematisk og teoridrevne afprøvning. Og her når vi så, trods transportministerens naive logiske fejlslutning, til erkendelse af, at vi ikke kan være *sikre* på, at vi har fanget alle potentielle fejl og problemer. Derfor må vi gå en mellemvej, som vi skal se nærmere på i kapitel ?? og tænke korrekthed og afprøvning ind i hele udviklingsforløbet, således at vi kan tage forbehold for den slags fejl, vi allerede kender (race-conditions, overflow, skrivbare globale variable, covert channels, ...) og have en god udviklings-, programmerings- og dokumentationskultur, således at vi kan stille os op som eksperter og forsøre, at koden er så tæt på fejlfri, som det er *menneskeligt* muligt under de givne betingelser. Det er nemlig det bedste, vi kan håbe på.

## Litteratur

- 635 Anderson, Antony F. (2016). „Case Study. NHTSA’s Denial of Dr Raghavan’s Petition to Investigate Sudden Acceleration in Toyota Vehicles Fitted With Electronic Throttles“. *IEEE Access*, bd. 4, s. 1417–1433. DOI: 10.1109/access.2016.2545713.
- Bond, Gregory W. (aug. 2005). „Software as Art“. *Communications of the ACM*, bd. 48, nr. 8, s. 118–124.
- 640 Bryant, Randal E. og David Richard O’Hallaron (2011). *Computer Systems. A Programmer’s Perspective*. 2. udg. Addison-Wesley.
- Colmerauer, Alain og Philippe Roussel (1993). „The birth of Prolog. Drawn from the [Second] ACM SIGPLAN History of Programming Languages Conference, April 20–23, 1993, HOPL-II“. I: New York: ACM Press, s. 37–52. DOI: 10.1145/154766.155362.
- 645 Cummings, David M. (dec. 2016). „Embedded Software Under the Courtroom Microscope. A Case Study of the Toyota Unintended Acceleration Trial“. *IEEE Technology and Society Magazine*, bd. 35, nr. 4, s. 76–84. DOI: 10.1109/mts.2016.2618681.
- Dijkstra, Edsger W. (1968). „Go To Statement Considered Harmful“. *Communications of the ACM*, bd. 11, nr. 3, s. 147–148. DOI: 10.1145/362929.362947.
- 650 Ensmenger, Nathan (2010). *The Computer Boys Take Over. Computers, Programmers, and the Politics of Technical Expertise*. Cambridge & London: MIT Press.
- Feuer, Alan R. og Narain H. Gehani (mar. 1982). „Comparison of the Programming Languages C and Pascal“. *ACM Computing Surveys (CSUR)*, bd. 14, nr. 1, s. 73–92. DOI: 10.1145/356869.356872.
- 655 Finch, Joel (2010). „Toyota Sudden Acceleration. A Case Study of the National Highway Traffic Safety Administration — Recalls for Change“. *Loyola Consumer Law Review*, bd. 22, nr. 4, s. 472–496.

- Gokhale, Jayendra, Raymond M. Brooks og Victor J. Tremblay (nov. 2014). „The effect on stockholder wealth of product recalls and government action. The case of Toyota’s accelerator pedal recall“. *The Quarterly Review of Economics and Finance*, bd. 54, nr. 4, s. 521–528. DOI: 10.1016/j.qref.2014.06.004. 660
- Good, Judith og Kate Howland (apr. 2017). „Programming language, natural language? Supporting the diverse computational activities of novice programmers“. *Journal of Visual Languages & Computing*, bd. 39, s. 78–92. DOI: 10.1016/j.jvlc.2016.10.008.
- Goosen, Leila (2008). „A Brief History of Choosing First Programming Languages“. I: *History of Computing and Education 3 (HCE3)*. Red. af John Impagliazzo. Springer US, s. 167–170. DOI: 10.1007/978-0-387-09657-5. 665
- Haigh, Thomas (jan. 2002). „Software in the 1960s as Concept, Service and Product“. *IEEE Annals of the History of Computing*, bd. 24, nr. 1, s. 5–13. DOI: 10.1109/85.988574.
- Holmevik, Jan Rune (1994). „Compiling SIMULA: A historical study of technological genesis“. *IEEE Annals of the History of Computing*, bd. 16, nr. 4, s. 25–37. DOI: 10.1109/85.329756. 670
- Kernighan, Brian W. og Dennis M. Ritchie (1978). *The C Programming Language*. 1. udg. Prentice Hall PTR.
- Klein, Gerwin m.fl. (2018). „Formally Verified Software in the Real World“. *Communications of the ACM*, bd. 61, nr. 10, s. 68–77. DOI: 10.1145/3230627. 675
- Koopman, Phil (sep. 2014). *A Case Study of Toyota Unintended Acceleration and Software Safety*. URL: [https://users.ece.cmu.edu/~koopman/toyota/koopman-09-18-2014\\_toyota\\_slides.pdf](https://users.ece.cmu.edu/~koopman/toyota/koopman-09-18-2014_toyota_slides.pdf) (bes. 13.07.2020).
- Krogdahl, Stein (2005). „The birth of Simula“. I: *History of Nordic Computing*. Springer, s. 261–275. DOI: 10.1007/0-387-24168-X\_24. 680
- MacKenzie, Donald (2001). „A View from the Sonnenbilch: On the Historical Sociology of Software and System Dependability“. I: *History of Computing: Software Issues*. Red. af Ulf Hashagen, Reinhard Keil-Slawik og Arthur Norberg. International Conference on the History of Computing, ICHC 2000. April 5–7, 2000, Heinz Nixdorf MuseumsForum, Paderborn, Germany. Berlin etc.: Springer, s. 97–122. 685
- (2004). „Computers and the Sociology of Mathematical Proof“. I: *New Trends in the History and Philosophy of Mathematics. New Trends in the History and Philosophy of Mathematics*. Red. af Tinne Hoff Kjeldsen, Stig Andur Pedersen og Lise Mariane Sonne-Hansen. University of Southern Denmark Studies in Philosophy 19. Odense: University Press of Southern Denmark, s. 67–86. 690
- Nofre, David (2010). „Unraveling Algol. US, Europe, and the Creation of a Programming Language“. *IEEE Annals of the History of Computing*, bd. 32, nr. 2, s. 58–68. DOI: 10.1109/MAHC.2010.4.
- Nofre, David, Mark Priestley og Gerard Alberts (2014). „When Technology Became Language. The Origins of the Linguistic Conception of Computer Programming, 1950–1960“. *Technology and Culture*, bd. 55, nr. 1, s. 40–75. DOI: 10.1353/tech.2014.0031. 695
- Northover, Mandy m.fl. (2008). „Towards a Philosophy of Software Development: 40 Years after the Birth of Software Engineering“. *Journal for General Philosophy of Science*, bd. 39, nr. 1, s. 85–113. DOI: 10.1007/s10838-008-9068-7. 700
- Ritchie, Dennis M. (1993). „The development of the C language. Drawn from the [Second] ACM SIGPLAN History of Programming Languages Conference, April 20–23, 1993, HOPL-II“. I: New York: ACM Press, s. 201–208. DOI: 10.1145/154766.155580.
- Rutishauser, Heinz (1967). *Description of ALGOL 60*. Die Grundlehren der mathematischen Wissenschaften 135. New York: Springer-Verlag. DOI: 10.1007/978-3-642-86934-1. 705

- Sammet, Jean E. (jul. 1972). „Programming languages. History and future“. *Communications of the ACM*, bd. 15, nr. 7, s. 601–610. DOI: 10.1145/361454.361485.
- Sammet, Jean E. (aug. 1978). „The early history of COBOL“. *ACM SIGPLAN Notices*, bd. 13, nr. 8, s. 121–161. DOI: 10.1145/960118.808378.
- 710 Stroustrup, Bjarne (1993). „A history of C++. Drawn from the [Second] ACM SIGPLAN History of Programming Languages Conference, April 20–23, 1993, HOPL-II“. I: New York: ACM Press, s. 271–297. DOI: 10.1145/154766.155375.
- U.S. Department of Transportation Releases Results from NHTSA-NASA-Study of Unintended Acceleration in Toyota Vehicles (feb. 2011). NHTSA. URL: <https://one.nhtsa.gov/About-NHTSA/Press-Releases/2011/ci.U.S.-Department-of-Transportation-Releases-Results-from-NHTSA%20%93NASA-Study-of-Unintended-Acceleration-in-Toyota-Vehicles.print> (bes. 13.07.2020).
- 715 Vilstrup, Helle (1963). *Lærebog i GIER-Algol*. København: Akademisk Forlag.
- Wernick, Paul og Tracy Hall (2004). „Can Thomas Kuhn’s paradigms help us understand software engineering?“ *European Journal of Information Systems*, bd. 13, s. 235–243.
- 720 Wirth, N. (1993). „Recollections about the development of Pascal. Drawn from the [Second] ACM SIGPLAN History of Programming Languages Conference, April 20–23, 1993, HOPL-II“. I: New York: ACM Press, s. 333–342. DOI: 10.1145/154766.155378.
- 725 Wirth, Niklaus (2008). „A Brief History of Software Engineering“. *IEEE Annals of the History of Computing*, bd. 30, nr. 3, s. 32–39. DOI: 10.1109/MAHC.2008.33.

## Navneliste

- |   |     |  |
|---|-----|--|
| Backus, John Warner [John]<br>(1924–2007), 3, 7     |     | McCarthy, John (1927–2011), 4, 7                                 |
| Bell, Alexander Graham (1847–1922), 6               |     | Minsky, Marvin Lee [Marvin]<br>(1927–2016), 4                    |
| Church, Alonzo (1903–1995), 4                       | 730 | Naur, Peter (1928–2016), 3, 7                                    |
| Colmerauer, Alain (1941–2017), 6, 7                 |     | Nygaard, Kristen (1926–2002), 5                                  |
| Dahl, Ole-Johan (1931–2002), 5, 7                   |     | Perlis, Alan Jay [Alan] (1922–1990), 3, 7                        |
| Dijkstra, Edsger Wybe [Edsger]<br>(1930–2002), 2    |     | Ritchie, Dennis MacAlistair [Dennis]<br>(1941–2011), 6, 7, 9, 12 |
| Hill, Robin K., 13, 14                              | 735 | Sammet, Jean E. (1928–2017), 4                                   |
| Kernighan, Brian Wilson [Brian], 12                 |     | Saylor, Mark (–2009), 16   |
| Knuth, Donald E. [Donald], 1                        |     | Stroustrup, Bjarne, 7  |
| Koopman, Phil, 16, 17                               |     | Thompson, Kenneth Lane [Ken], 6, 10                              |
| Kowalski, Robert, 6, 7                              |     | Turing, Alan Mathison [Alan]<br>(1912–1954), 4                   |
| Kuhn, Thomas Samuel [Thomas]<br>(1922–1996), 12, 13 | 740 | Wirth, Niklaus Emil [Niklaus], 3, 5–7                            |
| LaHood, Ray H. [Ray], 17                            |     |  |

Indeks

- |   |     |  |
|---|-----|--|
| AI, 11  |     |  |
| ALGOL, 2–8, 11  |     |  |
| algoritme, 11   |     |  |
| applicering, 11                                       |     |  |
| assembler, 1  |     |  |
| automatiseret bevisgenerator, 11                      |     |  |
| <br>B, 6  |     |  |
| BASIC, 5, 7–9, 11                                     |     |  |
| Bell Laboratories, 6                                  |     |  |
| <br>C, 6–12, 14, 17                                   |     |  |
| C++, 7, 8, 11, 12                                     |     |  |
| COBOL, 4, 7–9, 11                                     |     |  |
| Commodore 64, 5                                       |     |  |
| compiler, 6, 10, 14                                   |     |  |
| CUDOS-normerne, 3                                     |     |  |
| <br>DASK, 3   |     |  |
| deklarative programmeringssprog, 11                   |     |  |
| <br>Electronic Throttle Control System<br>(ETCS), 16  | 760 |  |
| ETCS, <i>Se</i> Electronic Throttle Control<br>System |     |  |
| formalvidenskab, 13                                   |     |  |
| fortolkede sprog, 9                                   |     |  |
| FORTRAN, 3, 4, 7–11                                   |     |  |
| funktionelle sprog, 11                                |     |  |
| funktionelt programmeringssprog, 4, 11                |     |  |
| <br>garbage collection, 9                             |     |  |
| GIER, 3   |     |  |
| <br>heuristisk pruning, 6                             |     |  |
| højniveausprog, 9                                     |     |  |
| <br>IBM, <i>Se</i> International Business Machines    | 770 |  |
| imperative programmeringsprog, 11, 12                 |     |  |
| indkapsling, 11                                       |     |  |
| inkommensurabel, 13                                   |     |  |
| inkommensurable paradigmer, 13                        |     |  |
| interaktiv prover, 6                                  |     |  |
| International Business Machines (IBM), 3              |     |  |
| <br>JAVA, 9   | 785 |  |
|   | 795 |  |

- kuhnsk paradigme, 12
- lavniveausprog, 9
- LISP, 4, 7, 9, 11
- logisk programmeringssprog, 5
- logiske, 11
- maskinkode, 1
- modellering, 13
- MODULA-2, 5
- modularisering, 2, 11
- multiparadigmatisk sprog, 12
- NASA, *Se* National Aeronautics and Space Administration
- National Aeronautics and Space Administration (NASA), 16
- National Highway Traffic Safety Administration (NHTSA), 16
- NHTSA, *Se* National Highway Traffic Safety Administration
- objekt-orienteret programmering, 5, 7, 12
- objekt-orienteret programmeringssprog, 11, 12
- objekter, 11
- operativsystem, 3, 6, 9
- paradigme, 1, 12
  - kuhnsk, 12
  - programmeringsparadigme, 11
- PASCAL, 5–9, 11
- PERL, 12
- preprocessor, 6, 7
- procedure, 11
- procedurelle programmeringssprog, 11, 12
- programmeringsparadigme, 11
  - deklarativt, 11
  - imperativt, 11, 12
- programmeringssprog, 5
  - ALGOL, 2–8, 11
  - BASIC, 5, 7–9, 11
  - C, 6–12, 14, 17
  - C++, 7, 8, 11, 12
  - COBOL, 4, 7–9, 11
  - deklarativt, 11
  - FORTRAN, 3, 4, 7–11
  - funktionelt, 4, 11
  - højniveau, 9
- imperativt, 11, 12
- JAVA, 9
- lavniveau, 9
- LISP, 4, 7, 9, 11
- logisk, 5
- MODULA-2, 5
- objekt-orienteret, 5, 7, 11, 12
- paradigme, 11
- PASCAL, 5–9, 11
- PERL, 12
- procedurelt, 11, 12
- PROLOG, 5–9, 11
- PYTHON, 12
- SIMULA, 4, 5, 7, 8, 11
- struktureret, 2, 5
- VISUAL BASIC, 9
- von Neumann, 11
- PROLOG, 5–9, 11
- PYTHON, 12
- realvidenskab, 13
- recasting, 10
- rekursion, 9
- script-sprog, 12
- SIMULA, 4, 5, 7, 8, 11
- software, 1
- software engineering, 1
- softwarekrise, 2
- sprogstamme, 7, 10
- stakhåndtering, 9
- standardiseret, 6
- statiske typer, 9
- struktureret programmering, 2, 5
- syllogisme, 6
- syntaks, 10
- system software, 6
- Toyota Lexus 2009 ES350, 16
- Turing-komplette sprog, 13
- typehåndtering, 10
- UNIX, 6
- videnskab
  - formalvidenskab, 13
  - realvidenskab, 13
- VISUAL BASIC, 9
- von Neumann-sprog, 11

# Kapitel 5:

## Kunsten at få en computer til at adlyde (II): Softwareudvikling

Henrik Kragh Sørensen

Mikkel Willum Johansen

22. april 2022

5

### Indhold

5.1	Softwareudvikling som teori og praksis . . . . .	1
5.2	Hvis vi ikke kender historien, gentager vi den . . . . .	2
5.3	Hvad er god software? . . . . .	5
5.4	Hvordan organiserer man udvikling af god software? . . . . .	7
5.5	Kan man prøve sig frem til god software? . . . . .	10
5.6	Hvad med brugeren? . . . . .	12
5.7	Når IT-projekter stadig fejler . . . . .	12
5.8	Therac-25: Når man ikke ved, hvor meget er nok . . . . .	13
	<b>Litteratur</b>	<b>16</b>
		15

### 5.1 Softwareudvikling som teori og praksis

Hvis vi også vender tilbage til karakteriseringen af datalogi i kapitel 1, så noterer vi os, at der blandt datalogiens „vidtfavnende“ områder er der både teoretisk viden om computeres og beregningers natur og mere teknologisk viden om, hvordan man kan bringe IT i anvendelse. I det foregående har vi beskæftiget os med de indre forbindelser mellem en algoritme og den software, som implementerer den. Men en mindst lige så stor praktisk og filosofisk udfordring i de datalogiske videnskaber er den *proces*, hvorved et problem identificeres og løses ved hjælp af software. Her er der altså tale om *softwareudvikling* som både en praksis og en række videnskabsteoretiske refleksioner over samme. Og dette emne er vigtigt både for dets indre værdi som en væsentlig del af datalogens værktøjskasse og for de filosofiske, etiske, økonomiske, juridiske og politiske aspekter, som softwareudvikling også spænder over.

20

For at få hold på softwareudvikling både som praksis og som del af videnskabsfaget datalogi, er det nyttigt at udforske nogle forbindelser, analogier og kontraster. Man kan med god ret tænke på softwareudvikling som *anvendt* og *teknologisk videnskab*, der er *handlingsorienteret* (se også Kragh, 2003, s. 177–180). Heri ligger, at teknologiske videnskaber går ud over at ønske blot at *forstå* forhold i deres domæne, men også gerne vil angive metoder og teknikker til at *kontrollere* og *manipulere* forhold i domænet med (menneske-centrerede)

30

formål (se også kapitel 2). I det omfang softwareudvikling er en teknologisk videnskab,  
35 handler den altså om at teoretisere over ’metoder og processer til udvikling af software’ og herunder at udlede ’best practices’, som kan komme udviklere til gavn. Men selvom det kunne være oplagt at understrege forskellen mellem denne form for teknologisk videnskab og en mere *grundvidenskabelig del* af datalogien, så er forskellene sværere at fastholde en som så: Både grundvidenskab og teknologisk videnskab stræber efter at producere ny viden, begge former udøves typisk i de samme kontekster (universitetsinstitutter eller private forskningslaboratorier), begge former hører til uddannelseskanonen for datalogistuderende, og ofte udføres begge former af de samme individer eller forskningsgrupper.

En anden dimension af sammenligning kunne være at fokusere på de metoder, som bruges i softwareudvikling. I den praktiske udførelse af softwareudvikling er der ofte tale  
45 om en mere induktiv metode, baseret i nogen grad på *trial-and-error*. Dette står i kontrast til den teoretiske datalogis brug af formelle metoder, og er noget, vi skal udforske i afsnit 5.5. Og i studiet af softwareudvikling inddrages kvalitative, sociologiske metoder for at kunne komme til at forstå aktørernes interesser og handlinger. I ret direkte forlængelse heraf kunne en sidste, foreløbig form for sammenligning handle om de nærmest tilgrænsende felter. Hvor 50 teoretisk datalogi og programmering måske ligger tæt på matematik og sprogteori i det akademiske landskab, så trækker softwareudvikling både på studier af menneskelig adfærd, designvidenskab og organisationsteori.

Hvis man forsøger at identificere og beskrive forskellige overordnede positioner i debatterne om software-engineering — både de faglige og de filosofiske debatter — så kan man i  
55 overensstemmelse med Gruner (2011, s. 284) pege på i hvert fald tre forskellige *partier*:

1. En *formalistisk* position, der tager udgangspunkt i, at software-udvikling handler om matematiske objekter, og derfor argumenterer for brug af matematiske metoder til at sikre god software, fx igennem formel verifikation eller model-verifikation.
2. En anden *ingeniør-centreret* position, som fremhæver stringent produktionsprocesser og workflow-modeller, som de kender fra fx mekaniske fabrikker, til at sikre god softwareudvikling.  
60
3. En tredje *humanistisk* position, der fremhæver de sociale interaktioner undervejs i udviklingen af software og de konsekvenser, software har i det omgivende samfund.

Et nuanceret, filosofisk syn på softwareudvikling bør tage alle disse perspektiver med i  
65 betragtning og kende til deres respektive fokuspunkter og begrænsninger.

I dette kapitel skal vi derfor studere softwareudvikling som videnskaben om de processer, der indgår i at fremstille og anvende software. Det skal vi gøre ud fra en *systemisk tilgang til software*, en *inkrementiel tilgang til udvikling* og en *bruger-orienteeret tilgang til kvalitet*.

## 5.2 Hvis vi ikke kender historien, gentager vi den

70 Det berømte citat fra den spanske filosof og essayist GEORGE SANTAYANA (1863–1952) — „Those who cannot remember the past are condemned to repeat it“ (Santayana, 2011, s. 172) — har en dobbelt betydning inden for softwareudvikling: På den ene side er det altid vigtigt at have et historisk blik for, hvordan vores moderne ideer om softwareudvikling er blevet udviklet igennem de sidste 50 år (Northover m.fl., 2008), siden man begyndte aktivt  
75 at tænke over, hvad *software engineering* kunne og skulle være. Men på den anden side

er indholdet af moderne softwareudvikling også på mange måder formet af gentagne *læringsmomenter*, hvor fiaskoer af forskellige former har fået dataloger og andre til at tænke over, hvordan software kan og bør udvikles. Nogle af disse episke fiaskoer handler om overskridelser af budgetter og tidsplaner, nogle af dem handler om, at man ikke kan forudse teknologiens fremtidige udvikling, og nogle af dem handler om, at softwareens begrænsninger ikke kunne indses på forhånd, men måtte opdages og læres — af historien!

80

Et af de mest indflydelsesrige *læringsmomenter* var, da International Business Machines (IBM) næsten brækkede halsen på det nu notoriske operativsystem *OS/360*. Siden umiddelbart efter Anden Verdenskrig havde IBM domineret markedet for store computere til myndigheder, banker og store virksomheder. Pointen med *OS/360* var, at IBM ønskede at udvikle deres computere og deres software i retning af kompatibilitet, således, at kunder skulle kunne opskalere deres hardware uden at være nødt til at udvikle og adaptere deres software, sådan som man ellers havde været nødt til. Man skal huske, at man i 1950'erne og 1960'erne — af grunde, som vi bl.a. skal vende tilbage til i kapitel 7 — ikke anså software som et selvstændigt produkt. Derimod opfattede man software som noget, der blev leveret og evt. vedligeholdt enten af hardwareproducenten eller af en inhouse computer-afdeling i de største virksomheder (Haigh, 2002).

85

90

95

Men det storslæde projekt med at lave en fælles platform i form af et delt *operativsystem*, som skulle sætte forskellige stykker hardware i stand til at køre *samme* stykke software, var lige ved at gå helt galt for IBM. Lederen af projektet, FRED BROOKS beskrev efterfølgende sine erfaringer i en bog, der kom til at sætte dagsordenen for en del af det, der var galt med den tidlige, før-professionaliserede form for software-udvikling (Brooks, 1995): Blandt BROOKS' konklusioner er i hvert fald en lille håndfuld i dag gået over som folklore iblandt dataloger, fx 1. 'no mythical man-month', 2. 'no silver bullet', 3. 'second system effect' og 4. 'irreducible number of errors' (se figur 1). BROOKS observerede også, at projekter 'bliver forsinkel en dag ad gangen' i takt med at planerne skrider fx ved at nye features bliver foreslået implementeret. Det fik ham til at insistere på, at software skulle ses som en opsplitning mellem arkitektur og implementation: Arkitekturen er den samlede begrebsslige beskrivelse af systemet, og der skal stræbes efter høj grad af klarhed og integritet. Hvordan arkitekturen så implementeres er en anden sag, og her kan alle mulige former for lokal kompleksitet komme til at vise sig nødvendig — men det skal ikke 'foreure' den arkitektoniske integritet. I stedet skal implementationen selv være læselig, dokumenteret og afprøvet.

100

105

No mythical man-month	BROOKS' måske mest provokerende erfaring var, at programmører ikke er udskiftelige og skalbare ressourcer i softwareudviklingen. Man kan ikke ved at ansætte dobbelt så mange programmører få produktionen af software til at gå dobbelt så hurtigt. Så fraværet af den <i>mytiske mandemåned</i> , han refererer til, gik imod gængs tænkning om effektivisering og ledelse i fx <i>Fordismen</i> .
No silver bullet	En anden provokerende erfaring var, at BROOKS ikke mente, at et enkelt greb — som i en enkelt strategi eller filosofi — omkring softwareudvikling ville kunne garantere pålidelige systemer, der overholdt budgetterne. I stedet skulle softwareudvikling selv anses som en kompleks proces med behov for styring og tilpasning af mange forskellig slags.
Second system effect	En af de konkrete udfordringer, som BROOKS var stødt på med <i>OS/360</i> var en gradvist stigende kompleksitet af opgaven i takt med, at designere og programmører fik øjnene op for endnu flere smarte muligheder, de kunne bygge ind i systemet. Så selvom BROOKS ligesom andre anbefalede, at den første udgave af et-hvert stykke software blev anset som en pilot-version, der kunne kasseres helt, så identificerede han også et modsatrettet træk i retning af, at version 2 bliver <i>for kompleks</i> — og mere generelt et teknologisk <i>pull</i> i retning af, hvad der er muligt nærmere end hvad der er ønskeligt (eller bestilt).
Irreducible number of errors	En yderligere grund til softwareens kompleksitet og fordyring kommer, mener BROOKS, af selve processen med at gøre software pålidelig: Hver gang vi retter en fejl risikerer vi nemlig at smige <i>nye</i> fejl ind i software — enten lokalt, hvor vi rettede, eller et andet sted i systemet, hvor vi ikke havde tænkt over, at rettelsen kunne have konsekvenser. Pessimistisk udtrykker han dette fænomen som, at der er en nedre men positiv grænse for, hvor få fejl, et stykke software kan have. Vi kan aldrig, siger han, opnå fejlfri software!

Figur 1: Opsummering af BROOKS' erfaringer fra *OS/360* baseret på hans bog *The Mythical Man-Month* (Brooks, 1995).

110 Erfaringerne fra *OS/360* og en række andre storlæede fiaskoer i at planlægge og styre softwareudvikling var en medvirkende årsag til, at NATO i 1968 afholdt den første af en række konferencer om softwareudvikling. Denne konference i Sonnenbichl i Garmish i Sydtyskland er siden blevet anset som fødselsstedet for begrebet *software engineering*, og nogle af de store diskussioner om programmeringssprog og deres betydning for udvikling af god software (se kapitel ??) fandt sted under og i kølvandet på dette møde.

115 At NATO havde en så stærk interesse i softwareudvikling i 1960'erne er værd at bemærke: Foruden universiteter, de allerstørste private virksomheder, banker og finansvæsenet var militæret nogle af de få, der havde eget computerudstyr at udnytte og vedligeholde. Og i regi af militæret og rumudforskningen var nogle af de største datalogiske gennembrud sket.

En af de interessante cases handler om et af de tidligste realtidssystemer med grafisk bruger-interface, som det amerikanske militær begyndte udviklingen af i årene umiddelbart efter Anden Verdenskrig i samarbejde med forskere fra Massachusetts Institute of Technology (MIT). Formålet var at bygge en computer, kaldet *Whirlwind*, der kunne hjælpe med at simulere flyvning. Til den brug udviklede man mange nye teknologiske gennembrud, blandt andet en ny måde at opbygge computerens hukommelse på og muligheden for at vise data på en CRT-skærm. Projektet var altså en teknologisk succes i den forstand, at det bidrog til mange ting, vi i dag tager for givet. Men det var voldsomt dyrt og slugte det meste af flådens forskningsbudget. Så i begyndelsen af 1950’erne *forvandlede* man *Whirlwind computeren* til brug for et andet prestige-projekt, kaldet *SAGE*, som var et tidligt semi-automatisk sporings-system til luftforsvaret (Smith, 1976). Og i den form blev grafiske interfaces til en uomgængelig del af at *semi-automatisere* kritiske processer. Men lektien om, at *Whirlwind* var blevet så dyrt hang alligevel ved i militære kredse, hvor pålideligheden af softwaren i forvejen var af høj prioritet. 120  
125  
130

### 5.3 Hvad er god software?

De foregående nedslag i softwareudviklingens historie har vist nogle af de interesser, der kan være på spil for at sikre god software. Og flere af disse hensyn spiller stadig en afgørende rolle, så det er værd at begynde overvejelserne over, hvordan man skaber *god* software med at reflektere over, hvilke kvaliteter, software egentlig kan og bør have. 135

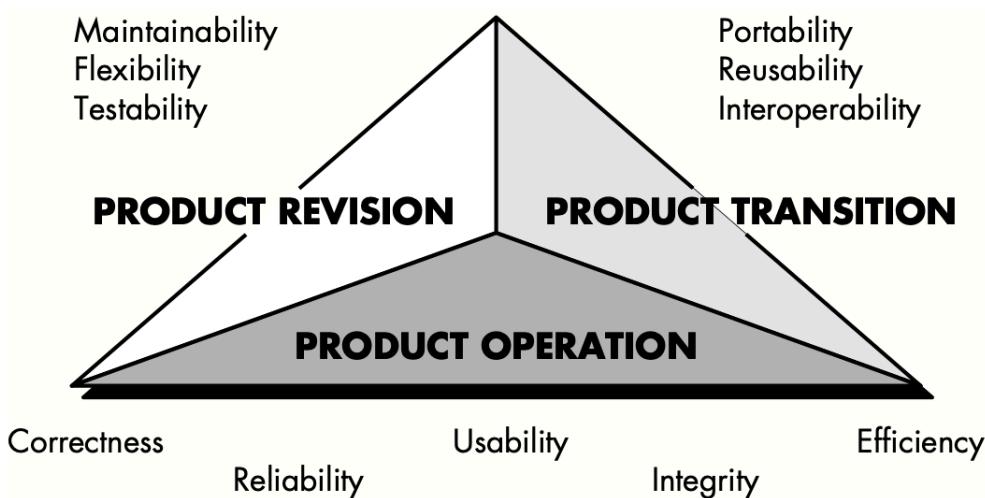
Software — i forstanden programmer og systemer, som brugere kan eftersørge, anskaffe, bruge og evt. videreudvikle — kan med nogen ret betragtes som *artefakter*: De er skabt af mennesker for at tjene et formål. Og på den måde kan man stille nogle af de samme kriterier op for kvaliteter ved software, som man kan for andre *artefakter* som biler, skruetrækere eller computere (se fx Floridi, Fresco og Primiero, 2015, s. 1200). Mange af disse kvaliteter, som vi eftersørger om fx computere vil være, at de fungerer pålideligt — for eksempel, at de tænder, når vi trykker på knappen, og altid fremviser konsistent opførsel, således at vi kan stole på, at de virker også i fremtidige situationer. Andre kvaliteter, som vi er vant til at eftersørge, er anvendelighed til at løse de opgaver, vi støder ind i, således at teknologien gør det *nemmere* for os at opnå *vores* mål (som brugere). En stor del af disse kvaliteter er *designet ind i artefakten*, således, at skruetrækkeren fx har et solidt håndgreb, men som vi så i kapitel 2, er der også mange af kvaliteterne, der i virkeligheden afhænger af det eller de *tekologiske systemer*, som den enkelte *artefakt* indgår i: Der er ingen kvalitet i en stjerneskruetrækker, hvis jeg ikke kunne købe stjerneskruer at skrue i med den. 140  
145  
150

Alligevel adskiller software sig på nogle punkter fra mere traditionelle artefakter, fx ved at software ikke *forgår* — i hvert fald ikke på samme måde, som fx kærvnen på min skruetrækker bliver slidt (Floridi, Fresco og Primiero, 2015). Og derfor er diskussioner om softwarekvalitet også på nogle områder forskellig fra til tilsvarende diskussioner om hardwarekvalitet, selvom det var et oplagt sted at hente inspiration for kvalitetssikring (Pressman, 2001, s. 212). 155

En anden forskel ligger måske i softwarens forhold til de mere ’eksterne teknologiske systemer’ og arbejdsprocesser, som den skal indgå i. Hvor der er meget lang tradition for hamre og søm og en klar kontinuitet fra hestevogne til biler, så er indførelsen af software i mange teknologiske systemer en mere brat og omdefinerende begivenhed. Det giver mening, at *datamatisering* og i dag mere udbredt *digitalisering* både er blevet set som det store rationelle fremskridt og effektivisering og som fremmedgørende og angstfremkaldende i dele 160

165 af befolkningen: For hele arbejdsgange og professioner er blevet afhængige af, at software  
er en medspiller og ikke en modspiller i den teknologiske udvikling.

Så hvis vi skal indfange kvaliteter ved god software, kan vi tage udgangspunkt i den  
omfattende liste af faktorer i god software, som blev udarbejdet af eksperter fra General  
Electric Company (GE) i slutningen 1970'erne på bestilling af det amerikanske luftvåben.  
170 Ved et stort litteratur-survey fandt forskere frem til 14 faktorer, der sammen med kvalite-  
ten af dokumentation, tidsforbrug og pris på softwaren indgår i vurderinger af softwarens  
kvalitet (McCall, Richards og Walters, 1977, bd. I, s. 2.7). Listen er siden blevet bearbejdet  
yderligere og forsimplet, således at den fx i Pressman (2001, s. 509) kan sammenfattes i tre  
overordnede komponenter: softwarens operation, transition og revision (se figur 2).



175

Figur 2: McCall's 14 kriterier anskueliggjort som hørende til softwarens operation, transition  
og revision (Pressman, 2001, s. 509).

Mange af faktorerne er ret nemt genkendelige: For eksempel omfatter *korrekthed*, at  
programmet opfylder specifikationerne og indfrier kundens formål; *pålidelighed* indfangede  
for JIM A. McCall og hans kolleger, at man kan stole på, at programmet opfylder sit  
mål med den givne præcision; *integriteten* er udtryk for, at adgangen til programmet og  
dets data kan kontrolleres, så herunder hører privacy og sikkerhed; og *brugbarhed* måler den  
indsats, en bruger skal investere for at kunne lære og bruge programmet. Men det mest  
interessante ved figuren er nok, at softwarekvalitet i denne betragtning rækker (langt) ud  
over kvaliteten af brugen af produktet og også omfatter vedligeholdelse og transision af  
produktet. Det er faktorer, som lægger sig tættere op ad en ingeniørsmæssig tænkning om  
systemer, der skal kunne vedligeholdes, opdateres og skaleres. Og selvom kataloget af faktor-  
erer er omfattende, er der også andre aspekter, som kunne tænkes med ind i softwarekvalitet,  
fx den produktionsforøgelse og relevans, som produktet medfører (Landauer, 1995)—det  
vil være en anden form for effektivitet end den, McCall anlægger, som handler om ma-  
skinressourcer—eller det stykke menneskelig og social 'engineering', der skal lægges, for  
180 at programmet bruges godt i organisationer og arbejdsgange (Glass, 1998). Disse kvalifi-  
ceringer peger på, at softwarekvalitet er en meget bredere kategori end blot korrekthed og  
pålidelighed.  
185  
190

I en interessant sammenligning mellem nogle af de allerførste refleksioner i begyndelsen af 1960’erne over softwareudvikling som proces og de praktiske erfaringer 20 år senere (Boehm, 1987) får man et indtryk af, at nogle af de udfordringer, som stadig er relevante, har en lang historie og måske faktisk har en mere fundamental karakter:

195

- Specifikationernes præcision og testbarhed: Det bør være klart, hvad softwarens specifikationer skal være — og dette bør være tænkt, beskrevet og aftalt på forhånd. Og det skal være muligt at afgøre, om et krav i specifikationen er opfyldt eller ej.
- Softwarens grænseflader: Grænsefladerne — både i forhold til andre systemer og i forhold til brugeren — er en del af specifikationen, der er særligt vigtigt at få aftalt og godt dokumenteret. Filformater var længe en stor udfordring for integration af processer og udveksling af data, og vigtigheden af gennembrud som markup-language skal ses i det lys.
- Topstyret design og „lean staffing“ i projektets tidlige faser: W. A. HOSIER beskrev det karakteristisk for den arkitektoniske, designmæssige tilgang til software: „The designers should not be saddled with the distracting burden of keeping subordinates profitably occupied“ og „Quantity is no substitute for quality; it will only make matters worse“ (Hosier, 1987, s. 321).

200

205

210

## 5.4 Hvordan organiserer man udvikling af god software?

210

En af de interessante videnskabsteoretiske vinkler på softwareudvikling kommer af at betragte software som arbejdsprocesser, både i produktionen af software og i de sociale og erhvervsmæssige sammenhænge, hvor software skal bruges (Ensmenger og Aspray, 2001). Dermed stiller der sig både interessante spørgsmål til ledelse og inddragelse og til den professionalisering af software, som også er et fremtrædende karakteristikum ved de sidste 50 år (se fx Ensmenger, 2010).

215

Som allerede beskrevet har organiseringen af god softwareudvikling været et særdeles dybtfølt behov både i civile og militære kredse, men det har ikke været uden problemer at finde frem til gode organisationsmodeller. For softwareudvikling er en krævende branche, som historisk alligevel var præget af lav social indtrædelsestærskel og en række idealer om autonomi, som var svært forenelige med sædvanlig amerikansk erhvervkultur (Ensmenger, 2003). Og hvor enkeltpersoner eller små, sammentømrede hold i starten kunne udvikle brugbare stykker software, er software blevet ’big business’, både målt i omsætning og i antallet af beskæftigede på de enkelte projekter. Og både anekdotisk viden og forsøg på systematiske undersøgelser viser, at det er langt billigere at træffe de rette designvalg tidligt end at skulle ud og ændre efter release (Pressman, 2001, s. 14 taler om en vækst i forandringsomkostninger fra 1 i designfasen til mellem 1,5 og 6 i udviklingsfasen og helt op til mellem 60 og 100 efter release). En klassisk kritik af de meget topstyrede organisationsformer, kaldet ’katedralerne’, i modsætning til mere demokratiske og flade strukturer, kaldet ’basarer’, kan findes i ERIC STEVEN RAYMONDS forsvar for Open Source Software (Raymond, 2001), men de relevante videnskabsteoretiske diskussioner er sådan set ikke betinget af forretningsmodeller.

220

225

230

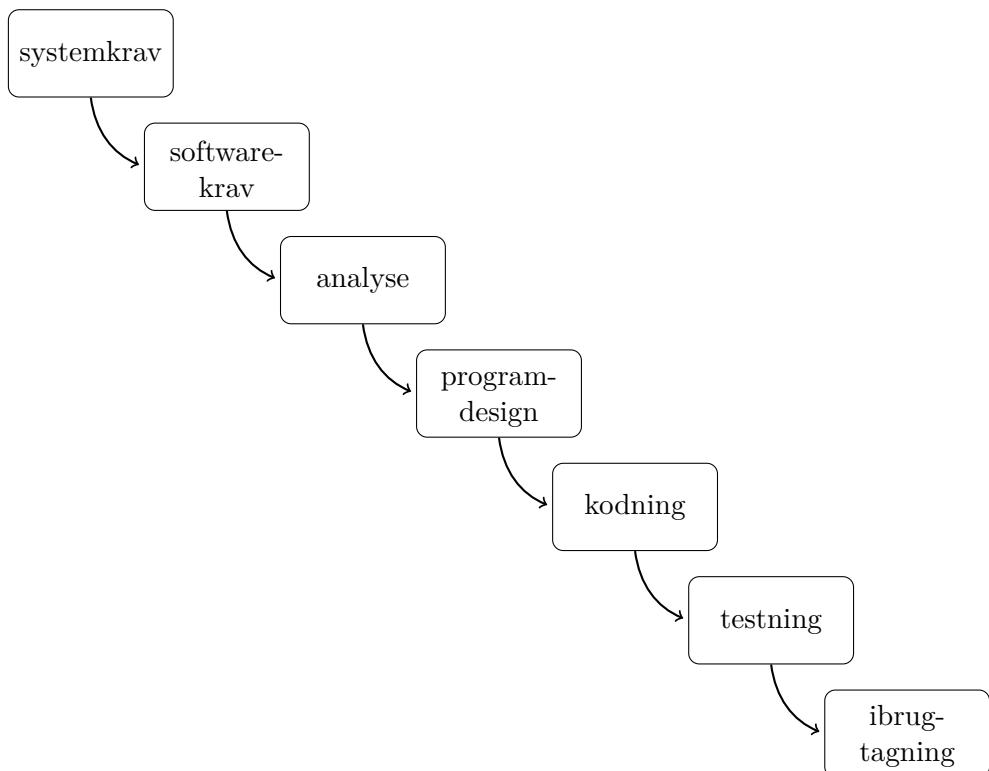
For at komme til nogle af de videnskabsteoretiske refleksioner over udviklingsprocesser og organisering deraf vil vi først meget kort præsentere tre karakteristiske modeller for softwareudvikling: vandfaltsmodellen, model-drevet udvikling og agil udvikling. De tre tilgange repræsenterer en klassisk ingeniørmæssig og korporativ udviklingsmodel, en meget

235

formalistisk-orienteret model og en mere bruger-inddragende og i den forstand humanistisk tilgang. Der findes mange andre modeller og tilgange (se fx Northover m.fl., 2008, s. 87),  
240 særligt for objekt-orienteret analyse, design, implementation og test, men disse er udvalgt fordi de repræsentere væsentlige videnskabsteoretiske forskelle, og de kan i virkeligheden bringes i anvendelse i tænkning om (næsten) al slags software.

## Vandfaldsmodellen

Den såkaldte *vandfaldsmodel*, som i sin kanoniske version blev formuleret fx af WINSTON W. ROYCE (1929–1995) fra Lockheed, som har store kontrakter med det amerikanske militær,  
245 bliver ofte trukket frem og kritisert for dens rigide adskillelse af de forskellige processer, som indgår i softwareudvikling. Men historisk er det lidt ufortjent, for ROYCES version har (mindst) to vigtige kvaliteter: 1. Den sætter ord på (nogle af) de forskellige processer, som traditionelt har indgået i softwareudvikling (se figur 3): specifikation, analyse, implementering, test og ibrugtagning, og 2. ROYCES eget formål med artiklen var faktisk at diskuer  
250 og nuancere modellens opbygning — men det går sommetider glemt. I sin karikerede version er *vandfaldsmodellen* en *topstyret* (top-down) udviklingsproces, hvor de mest abstrakte trin skal foretages først og er forudsætninger for, at 'vandet kan spilde over' til næste trin i processen. Den er derfor blevet skældt ud for at argumentere for, at der ikke må skrives kode, før hele specifikationen er på plads. Men selvom denne designproces kunne virke tillokkende for mere traditionelle produktionsvirksomheder, viste allerede BROOKS' erfaringer fra IBM, at softwareudvikling i praksis synes at være en meget mere flydende — næsten kaotisk — proces.  
255



Figur 3: ROYCES simpleste vandfaldsmodel (Royce, 1987, s. 329).

*Vandfaldsmodellen* taler til en vision af, at software kan specificeres, designes, programmeres og testes i den rækkefølge og som adskilte skridt. Men allerede ROYCE talte for tidligt design for at afprøve muligheder, behovet for dokumentation også undervejs og imellem trinene og en filosofi om 'to-it-twice', fordi de første forsøg på fx implementation sjældent var optimale og lappeløsninger var (og blev anset for at være) en af kilderne til den enorme kompleksitet og fordyrelse, som software har været plaget af siden 1960'erne.

260

En del af kritikken og videreudviklingen af *vandfaldsmodellen* har, ligesom ROYCES formål, sigtet på at tage højde for, at information og beslutninger fra underliggende trin kan påvirke tidlige trin, uden at det betyder, at hele processen skal starte forfra. Et eksempel kunne være den rolle, som test spiller i processen (se også afsnit 5.5, nedenfor), hvor den karikerede model vil sige mod at bruge tests til at opdage fejl ved systematisk afprøvning af ellers givne og opfyldte specifikationer. Men mødet med virkeligheden — og især med brugeren — har vist sig (også) at være en produktiv kilde til at finde mangler i de specifikationer, som vandfaldsmodellen vil forudsætte, at man har opstillet til at begynde med (se fx kapitel 3).

270

I sine kommentarer argumenterer ROYCE også for at inddrage kunden undervejs i processen, men hans arguemnt er alligevel sigende for den topstyrede, kontraktfikserede tilgang, for kundeinddragelse skulle primært være med til at forebygge konflikter under overdragelsen af det færdige produkt. Så der er ikke så meget tænkt på samskabelse eller co-design som på informationsflow fra producent til kunde og milestones for ændringer i bestillingen.

275

## Modelbaseret udvikling

Der findes i dag adskillige forskellige videreudviklinger af den simple vandfaldsmodel, som forsøger at tage hensyn til kompleksiteterne i softwareudvikling. En af dem er såkaldt *modelbaseret udvikling* (MBD), som forsøger at kombinere mere formelle metoder med systematiske tests. Fortalere for denne tilgang arbejder ofte ud fra en V-formet udviklingsmodel som gengivet i figur 4.

280

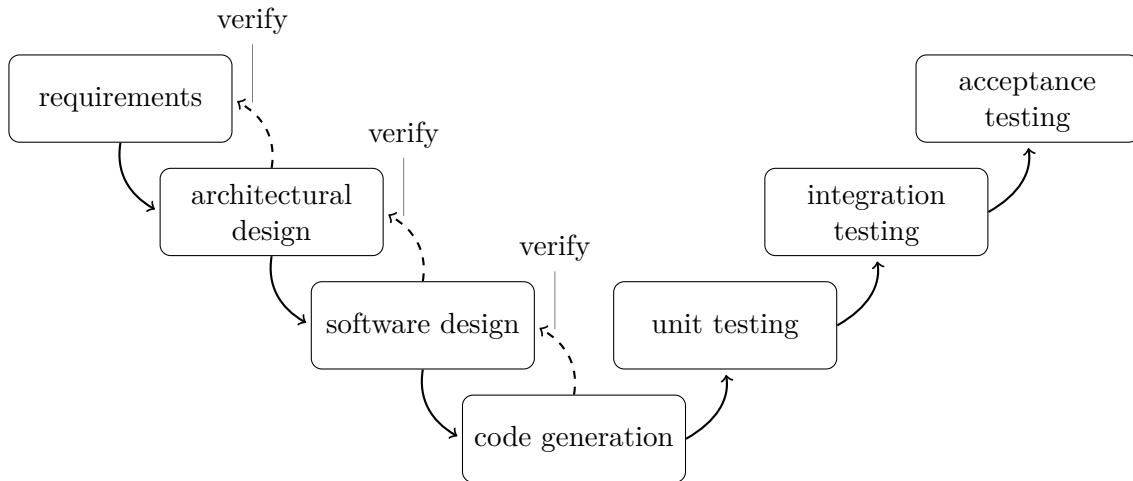
Ideen med MBD er, som navnet mere end antyder, at man ønsker at udnytte nogle af de fordele, som modeller mere generelt giver, fx for at kunne tale om samme ting på forskellige abstraktionsniveauer afhængig af konteksten. Nogle fortalere (Brambilla, Cabot og Wimmer, 2017, s. 8) argumenterer for at erstatte NIKLAUS WIRTHS ellers berømte symbolske ligning

$$\text{algoritmer} + \text{datastrukturer} = \text{software} \quad (1)$$

med

$$\text{modeller} + \text{transformationer} = \text{software}. \quad (2)$$

285



Figur 4: Model-baseret udvikling (Bialy m.fl., 2017, s. 42).

290

295

På overfladen kunne det ligne, at modeller *blot* er specifikationer, men i MBD er mekanismerne noget forskellige: Man taler om modelniveauet, hvor selve modellen er defineret, og realisationsniveauet, hvor løsninger er implementeret. Men det særlige ved denne tilgang er fokus på de afbildninger, der forbinder disse to niveauer, og som udgør et *automationsniveau* (Brambilla, Cabot og Wimmer, 2017, s. 10). Det afgørende er nu, at man kan tilføje et metaniveau til disse transformationer og dermed have en sammenhæng at argumentere om deres korrekthed inden for. Derved kombinerer MBD kaskadelementet af vandfaltsmodellen med bevisstyrken af formel verifikation. Og vel at bemærke sker dette *i løbet af* udviklingsprocessen, således at den formelle verifikation ikke er en påklistret og voldsomt fordyrende tilføjelse til det endelige produkt.

300

Der findes adskillige forskellige måder at implementere denne modelbaserede tilgang på, herunder flere forskellige sprog og notationer til at udvikle og tænke om modeller i, hvorfaf *UML* nok er det mest kendte. Og visionen — at man kan automatisere og formalisere store dele af softwareudviklingen — virker tilløkkende i mange sammenhænge, ikke mindst i forhold til kritiske systemer.

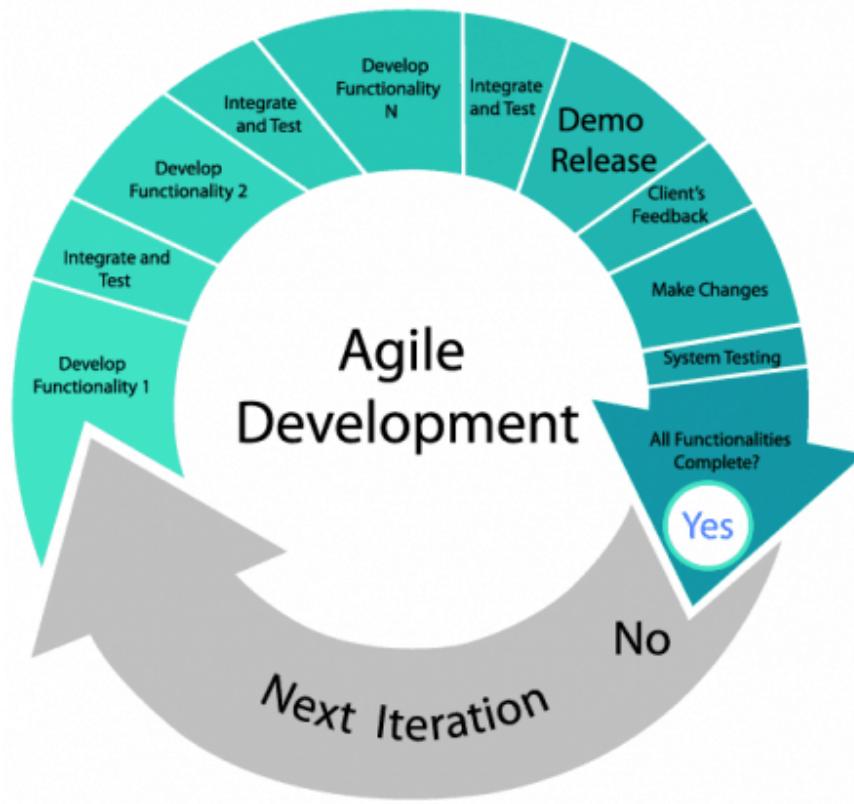
305

Man kan anskue MBD som et forsøg på igennem automatisering at omgå nogle af de processer, som man har erkendt er fordyrende og tilføjer kompleksitet til softwareudvikling. På den måde bliver det til et (blandt flere) eksempler på *metaprogrammering*, der er den yderste konsekvens af ideen om, at programmeringssprog skal understøtte menneskelig tænkning og skabning og ikke være bundet til den underliggende arkitektur (kapitel ??).

## Agile metoder

310

En radikalt anden måde at forsøge at omgå nogle af de samme omkostningstunge og forsinkende dele af vandfaltsmodellen har vundet stor indpas under navne som *agile metoder*. Hvor MBD holdt fast i en (i hvert fald i principippet) lineær udviklingsproces, så argumenterer man inden for agil udvikling for en cyklistisk og iterativ proces, der fokuserer mere på at levere en strøm af synlige og brugbare prototyper end på at specificere og designe systemet til mindste detalje inden implementationen går igang (se figur 5).



Figur 5: Agile metoder illustreret som cyklisk, iterativ proces. [Figuren skal tilpasses]

Agile metoder er med andre ord baseret på et helt andet sæt værdier om softwareudvikling (citeret fra Northover m.fl., 2008, s. 91):

315

1. individuals and interactions over processes and tools,
2. working software over comprehensive documentation,
3. customer collaboration over contract negotiation, and
4. responding to change over following a plan.

Det agile ved denne filosofi består altså både i en accept af, at software er og vedbliver at være en proces (og altså i udvikling og flux), og at omstillingssparathed i mødet med virkeligheden og kunden kan være vigtigere end en rigid udviklingsproces. På den måde står agil udvikling også vinkelret på mange af de bestrebelser, der er gjort på at kvantificere softwareudvikling med henblik på dokumentation og måloptimering. 320

Blandt de kendtegnende faktorer ved agil udvikling er hyppig brugerinddragelse og justering af både mål og prioriteter i projektet. Dette lyder meget fristende i en tid karakteriseret af hurtig udvikling og hård konkurrence inden for software, men det efterlader nogle af de klassiske kvalitetsmål som korrekthed og pålidelighed med meget mindre fokus end mere klassiske tilgange. 325

## 330 5.5 Kan man prøve sig frem til god software?

### 5.6 Hvad med bruger?

Et af de områder, hvor de forskellige udviklingsmodeller hidtil har stillet sig meget forskelligt og haft forskellige ting at bidrage er brugerens rolle, særligt i forhold til design af interfacet mellem maskine og bruger. Dette område er et selvstændigt forskningsfelt, ofte kaldet *human-computer interaction* (HCI), men også en kreativ designproces i at skabe gode *brugerinterfaces* (UI) og gode *brugeroplevelser* (UX). Og såvel i forskningsfeltet som i de kreative processer indgår der en lang række metoder, der rækker ud over de tekniske og matematiske tilgange, som vi tidligere har diskuteret, og i stedet trækker på både psykologisk og sociologisk viden. Og dertil kommer en lang række mere dogmatiske tommelfingerregler for designbeslutninger, der handler om brugerens interaktioner med maskiner og andre brugere.

340 Nogle af de mest succesfulde brugerdesigns har med rette fået ikonisk status i det moderne samfund — tænk på mus og cursor, drag-and-drop, swipe-right, skriveborde og skraldespande på skærmen. Og det er interessant at overveje, hvordan relationen mellem disse designede, virtuelle artefakter og den omgivende virkelighed er. For skriveborde og skraldespande er der en klar *metaforsik relation*: At lægge et dokument på skrivebordet vil være en handling, der giver fysisk mening i 1800-tallet og giver virtuel mening i dag. Noget andetledes ser det ud med nogle af de andre elementer af brugergrænsefladen, som har en meget mere normativ, definerende og systemisk natur. Der er ikke nogen metaforisk relation, der siger, at jeg udtrykker noget som helst ved at flytte min pegefinger til højre på min telefon eller trække musen henover bordfladen. Her er i stedet tale om, at brugergrænsefladen definerer ny adfærd hos mennesker, og at denne adfærd kan vise sig at række langt ud over den oprindelige motivation for den. Når det sidste sker, har vi fået tilføjet en ny mening til en handling, som så siden kan (og bør) udnyttes i andre tilsvarende situationer, hvor det giver mening.

Gennem disse refleksioner har vi nærmet os de tre 'gyldne regler', som brugergrænsefladedesigneren THEO MANDEL har opstillet (Mandel, 1997, kap. 5), og som ofte refereres:

1. Place Users in Control,
2. Reduce Users' Memory Load,
3. Make the Interface Consistent.

360 Hvert af disse dogmer udmøntes så i en række konkrete anbefalinger, som fx (under 1) at brugerens behov for at ombestemme sig ikke skal være en uønsket undtagelse for designeren, men noget, der aktivt understøttes, (under 2) at det er bedre at forlade sig på genkendelse end på genkaldelse, så brugergrænsefladen give et kognitivt off-load, og (under 3) at brugergrænsefladen, som vi jo lige sagde var en kreativ proces, alligevel bør holdes konform og konsistent mellem situationer og produkter.

En brugerflade og en brugeroplevelse er med andre ord noget, som udvikleren designer, og MANDELS regler er opstillet for at minde om, at det ikke er designerens men brugerens kognitive og sociale behov, der skal tilgodeses ved gode designs.

370 Dertil rejser sig så selvfølgelig spørgsmålet om, hvordan man som udvikler får adgang til brugerens behov — det være sig den enkelte slutbruger såvel som den mulige kunde. Og her kommer den systemiske vinkel igen i spil, for man skal også som udvikler have øje

for, at man leverer et stykke software, hvis funktion kan være med til radikalt at forandre menneskers liv og hverdag (se mere i kapitel ??).

Der findes selvfølgelig en række sociologiske metoder som observationsstudier, interviews og fokusgrupper, som kan kvalificere såvel den generelle analyse af softwareens formål som mere specifikt brugerens interaktionsbehov og -muligheder. Og nogle af disse metoder og anbefalinger kan tænkes ind i selve ingenørprocessen (Seffah og Metzker, 2004), således som det fx hyppigt er tilfældet med agile metoder.

Sammenfattende kan man sige, at selvom UI er en design-aktivitet, er den teleologisk i den forstand, at den har et klart formål, nemlig brugerens målopfyldelse. Derved adskiller UI sig alligevel fra kunstneriske aktiviteter, og UI kan ses som en artefakt, der både er del af og på linje med software.

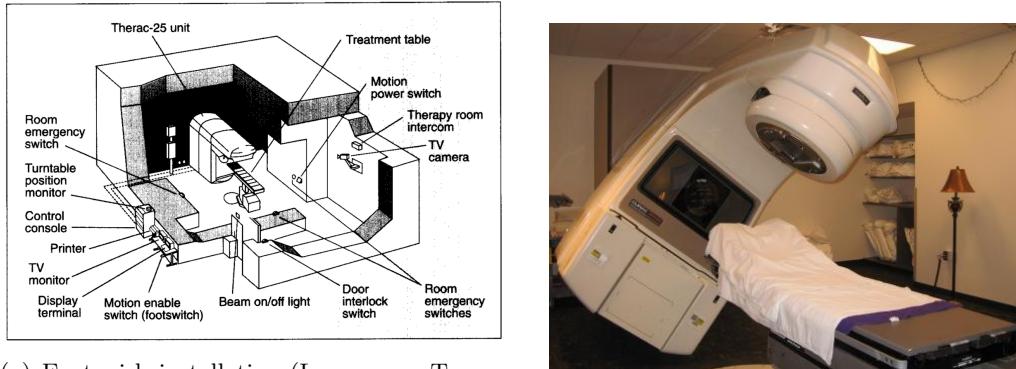
## 5.7 Når IT-projekter stadig fejler

Trods mere end 50 års fokus på udvikling af god software sker det stadig, at softwareprojekter fejler ved enten at blive voldsomt forsinkede eller fordyrede eller ved at munde ud i produkter og services, som ikke bliver indoptaget af de kunder og brugere, for hvem de var udviklet. Hvis vi kort vender tilbage til *vandfaldsmodellen* kan man sige, at enhver fase deri og enhver overgang er en potentiel katastrofe, der venter på at ske.

Den danske datalog ERIK FRØKJÆR har igennem årtier forsøgt at give sin akademiske viden om softwareudvikling videre til især offentlige software- og digitaliseringsprojekter. Blandt de grunde, som FRØKJÆR peger på, er der flere systemiske forhold, som han peger på, at man i principippet kunne undgå (se fx Frøkjær, 1987; Frøkjær, 2017). En af faktorerne kan kaldes utidig indblanding fra kundens side, fx ved at fremskynde ibrugtagning af systemer. Dette forhold peger jo i nogen grad på en udviklingsmodel, der reserverer tid og ressourcer til de enkelte trin i en autonom udviklingsproces. Og FRØKJÆRS observation peger dermed nok på, at agile udviklingsmål er dårligt forenelige med udrulning af store, definerende offentlige systemer. En anden faktor, som også kan identificeres har med effektivisering at gøre. For selvom digitalisering og software bærer løfter om automatisering og dermed effektivisering af menneskelige arbejdsgange, så er rækkefølgen bestemt ikke uvæsentlig: Udvikling, indførelse og indkøring af digitaliseringsprojekter er på den korte bane nok oftere dyrere end normal drift, men stiller så en effekt i udsigt på længere sigt. En tredje faktor har med det skel mellem styring/ledelse af projektet og udførelsen, som vi allerede berørte under professionaliseringen af softwareudvikling. For at kunne styre softwareprojekter er detailkendskab til processen nok værdifuldt, end man måske oplever i andre produktionsbrancher. Hvis det er en empirisk observation, så peger den i retning af, at softwareudvikling faktisk har sin egen disciplinære identitet, som ledere af udviklingsprojekter — private som offentlige — skal have kendskab til og forståelse for (se også Limoncelli, 2019).

## 5.8 Therac-25: Når man ikke ved, hvor meget er nok

Et af de mest omdiskuterede eksempler på et fejlende computersystem opstod i USA og Canada i midten af 1980'erne. Det drejer sig om en strålekanon til kræftbehandling, som gik under navnet Therac-25. Ialt var der installeret 11 maskiner af denne type, ligeligt fordelt nord og syd for grænsen, og de var i hyppig brug i kræftbehandling (en typisk installation er gengivet i figur 6).



(a) En typisk installation (Leveson og Turner, 1993, s. 19).

415

(b) Selve strålekanonen (wikipedia).

Figur 6: Strålekanonen Therac-25.

Therac-25 blev produceret af firmaet Atomic Energy of Canada Limited (AECL), som dengang var et statsdrevet selskab (se Leveson og Turner, 1993). Selve strålekanonen var en videreudvikling af tidligere, mindre kraftige modeller, som AECL havde udviklet i samarbejde med en fransk partner. Både de tidligere modeller (*Therac-6* og *Therac-20*) og Therac-25 blev styret af en DEC PDP-11 computer, som var en af de mest udbredte 'små' computere. Men hvor computeren havde spillet en mindre rolle i de tidligere versioners brug, blev Therac-25 fra starten designet til at udnytte computerstyring. Hvor man i tidligere modeller havde haft fysiske sikringsmekanismer til at overvåge strålingen, blev mange af disse på Therac-25 i stedet sikret gennem software. Denne software var i sig selv en videreudvikling af software fra *Therac-6*, og den var udviklet over en længere årrække af en enkelt programør og skrevet i PDP-11 assemblér.

Den slags hospitalsudstyr skal igennem ganske rigide sikkerhedsanalyser, før maskinen kan tages i brug. Og da Therac-25 var klar i 1982/83 blev den testet af AECL, hvor de fokuserede særligt på omfattende test af softwaren på en hardwaresimulator under brugslignende forhold. De konkluderede, at øvrige softwarefejl ikke var omfattet og analysen, at software ikke slides op, og at kommende problemer derfor måtte forudsese at stamme fra fejlagtig hardware eller tilfældige hændelser forårsaget af  $\alpha$ -stråling eller elektromagnetisk støj (Leveson og Turner, 1993, s. 21). Denne analyse ledte ingeniørerne til et fejl-træ, som angav de forskellige sikkerhedsproblemer og tildelte dem sandsynligheder. Således blev sandsynligheden for at computeren angav en forkert dosis stråling, uden videre forklaring, vurderet til  $10^{-11}$ .

Men fra 1985 begyndte brugerne af Therac-25 at opleve problematiske behandlinger. Og indtil maskinen blev tilbagekaldt i 1987 førte den til seks tilfælde af voldsomme, livstruende overdoseringer af stråling. Patienterne fik stråleskader, som betød at en måtte gennemgå en mastektomi, flere mistede førlighed i arme, skuldre og hofter, afhængig af hvor behandlingen havde sigtet. Selvom hospitalerne var i kontakt med AECL, og der blev udført ad-hoc justeringer af maskinen, blev årsagen ikke fundet og fjernet. Derfor blev de amerikanske myndigheder U.S. Food and Drug Administration (FDA) involveret i sommeren 1986, og en stor del af vores viden om casen stammer fra udredning og sagsanlæg. Man mistænkte elektriske fejl i fx justeringen af drejebordet, men fejlen skulle vise sig at ligge i softwaren.

Faktisk var der ikke tale om blot 1 softwarefejl, men om flere—og nogle af dem delte Therac-25 med maskinen *Therac-20*, hvor de bare ikke var blevet så kritiske, fordi fysiske

forholdsregler havde stoppet maskinen i at skade patienterne. Men den mest direkte fejl i Therac-25 var forbundet til en særligt mystisk omstændighed: Det så ud som om maskinen fungerede helt fint, når den blev installeret på hospitalerne—det var først efter nogen tid, og ganske uforudsigeligt, at problemerne opstod. Alligevel var problemerne jo gentagne, så det virkede usandsynligt, at de skulle være rene tilfældigheder. Ved en nærmere analyse af den software, der kontrollerer inddateringen af dosis, fandt man imidlertid frem til en årsag, der kunne forklare både mørnstræt af overdoseringer og det faktum, at overdoseringerne var så voldsomme. På det tidspunkt var programmøren ikke længere ansat i firmaet, og han viste sig umulig at opspore til sagsanlægene. 455

Problemet lå i kommunikationen mellem tastaturet og computeren. Når der blev tastet på keyboardet, blev der sendt et *interrupt* til PDP-11, som så skulle suspendere sin øvrige aktivitet, aflæse keyboardet, gemme information om den anslæde tast og vende tilbage til sin oprindelige udførsel, som var langsomt at dreje strålekanonen. Denne interrupt-mekanisme er i dag standard i de fleste hardware- og operativsystemer, men på PDP-11 var den implementeret direkte i hovedprogrammet. Og problemet med overdoseringerne blev nu sporet tilbage til, at hvis man trykkede på tasterne tilstrækkeligt hurtigt efter hinanden—som man som bruger af apparatet ville kunne lære at gøre, når man blev vant til det—blev det ene interrupt ikke afsluttet, før det næste løb ind. Derfor fik man en forkert afvikling af aflæsningen, og den indlæste og gemte dosering var ikke længere det, lægen havde ordineret. 465

Denne episode fandt sted på et tidspunkt, hvor understøttelse af interrupts i operativsystemer ikke var udbredt—og jo i hvert fald på en maskine, der ikke understøttede en interrupt-stack. Men at dette skulle være et problem for softwaren var ikke noget, man havde forudset, før man opdagede, at det kunne være et problem. Efterfølgende er det blevet et særligt opmærksomhedspunkt i al software, der skal kunne håndtere samtidige hændelser. 470

I 1987 og i forbindelse med sagen beskrev ED MILLER, der var direktør i Center for Devices and Radiological Health, som er en del af FDA, hvad man kunne lære af denne ulykkelige case:

FDA has performed extensive review of the Therac-25 software and hardware safety systems. We cannot say with absolute certainty that all software problems that might result in improper dose have been found and eliminated. However, we are confident that the hardware and software safety features recently added will prevent future catastrophic consequences of failure. (MILLER 1987 citeret fra Leveson og Turner, 1993, s. 38) 475  
480

Her påpegede MILLER således, at *absolut sikkerhed* for fejlfrie softwaresystemer ikke er en mulighed, men samlet kan man se, at selv omfattende testning af kritisk software på hardwaresimulatorer i brugslignende situationer ikke var tilstrækkeligt til at finde fejlen i Therac-25. Fejlen lod sig kun trigge i *særlige* brugssituationer, nemlig når brugeren var blevet tilstrækkelig hurtig til at trykke på tasterne. Endvidere er man nødt til at huske, at hvis interrupt-overflow ikke er en del af testerens forventningshorisont, så vil det ikke blive samlet op og testet. På den måde er Therac-25-fejlen og opdagelsen af covert channels i *Bell-LaPadula-modellen* sammenlignelige: Ingen af dem tager højde for uforudset brug. Men hvor Therac-25 var testet empirisk, så var *Bell-LaPadula-modellen* en matematisk model, og dette siger, at ingen af metoderne er tilstrækkelige til at garantere, at man fanger uhensigtsmæssig brug før i brugtagen. Casen var dermed også med til at fremskynde udviklingen af *system engineering* og *software engineering* som vigtige del af professionel programmering, og til at indføre rigide kvalitetstjek af software til kritiske systemer. For som FRANK HOUSTON fra det amerikanske FDA skrev: 485  
490

495 A significant amount of software for life-critical systems comes from small firms, especially in the medical device industry; firms that fit the profile of those resistant to or uninformed of the principles of either system safety or software engineering. (HOUSTON i 1985 citeret fra Leveson og Turner, 1993, s. 18)

500 Therac-25-casen er yderligere interessant, fordi den involverer så mange aktører med forskellige roller: programmøren, firmaet, hospitalerne, brugerne, patienterne, myndighederne og retssystemet. For hver af dem kan man lave en analyse af deres involvering og eventuelle ansvar — og i kapitel ?? vil vi tage den op igen i forbindelse med programmørernes professionelle ansvar.

## Litteratur

- 505 Benington, Herbert D. (1983). „Production of Large Computer Programs“. *IEEE Annals of the History of Computing*, bd. 5, nr. 4, s. 350–361.
- Bialy, M. m.fl. (2017). „Software Engineering for Model-Based Development by Domain Experts“. I: *Handbook of System Safety and Security*. Elsevier. Kap. 3, s. 39–64. DOI: 10.1016/b978-0-12-803773-7.00003-6.
- 510 Boehm, Barry W. (1987). „Software Process Management. Lessons Learned From History“. I: *ICSE '87. Proceedings of the 9th International Conference on Software Engineering*. Washington, DC: IEEE Computer Society Press, s. 296–298. DOI: 10.5555/41765.41798.
- Brambilla, Marco, Jordi Cabot og Manuel Wimmer (2017). *Model-Driven Software Engineering in Practice*. 2. udg. Bd. 3. 4. Morgan & Claypool. DOI: 10.2200/s00751ed2v01y201701swe004.
- 515 Brooks Jr., Frederick P. (apr. 1987). „No Silver Bullet. Essence and Accidents of Software Engineering“. *IEEE Computer*, bd. 20, nr. 4, s. 10–19. DOI: 10.1109/MC.1987.1663532.
- (1995). *The Mythical Man-Month. Essays on Software Engineering*. Anniversary edition. Boston etc.: Addison-Wesley.
- 520 Ensmenger, Nathan (2003). „Letting the “Computer Boys” Take Over: Technology and the Politics of Organizational Transformation“. *International Review of Social History (IRSH)*, bd. 48, nr. Supplement 11, s. 153–180. DOI: 10.1017/S0020859003001305.
- (2010). *The Computer Boys Take Over. Computers, Programmers, and the Politics of Technical Expertise*. Cambridge & London: MIT Press.
- 525 Ensmenger, Nathan og William Aspray (2001). „Software as Labor Process“. I: *History of Computing: Software Issues*. Red. af Ulf Hashagen, Reinhard Keil-Slawik og Arthur Norberg. International Conference on the History of Computing, ICHC 2000. April 5–7, 2000, Heinz Nixdorf MuseumsForum, Paderborn, Germany. Berlin etc.: Springer, s. 139–165.
- 530 Floridi, Luciano, Nir Fresco og Giuseppe Primiero (2015). „On malfunctioning software“.  
*Synthese*, bd. 192, s. 1199–1220. DOI: 10.1007/s11229-014-0610-3.
- Fraser, Steven og Dennis Mancl (jan. 2008). „No Silver Bullet. Software Engineering Relo-  
aded“. *IEEE Software*, bd. 25, nr. 1, s. 91–94. DOI: 10.1109/ms.2008.14.
- 535 Frøkjær, Erik (1987). „Styringsproblemer i det offentliges edb-anvendelse“. *Politica: Tids-  
skrift for Politisk Videnskab*, bd. 19, nr. 1, s. 31–56.
- (jun. 2017). „Finansministeriet“. *Weekendavisen*, nr. 23.
- Glass, R. L. (1998). „Defining quality intuitively“. *IEEE Software*, bd. 15, nr. 3, s. 103–104,  
107. DOI: 10.1109/52.676973.

- Gruner, Stefan (feb. 2011). „Problems for a Philosophy of Software Engineering“. *Minds and Machines*, bd. 21, nr. 2, s. 275–299. DOI: 10.1007/s11023-011-9234-2. 540
- Haigh, Thomas (jan. 2002). „Software in the 1960s as Concept, Service and Product“. *IEEE Annals of the History of Computing*, bd. 24, nr. 1, s. 5–13. DOI: 10.1109/85.988574.
- Hosier, W. A. (1987). „Pitfalls and Safeguards in Real-Time Digital Systems with Emphasis on Programming“. I: *Proceedings of the 9th International Conference on Software Engineering*. ICSE '87. Monterey, California, USA: IEEE Computer Society Press, s. 311–327. 545
- Kragh, Helge (2003). „Hvad er videnskab?“ I: Fink, Hans m.fl. *Universitet og Videnskab. Universitetets idéhistorie, videnskabsteori og etik*. København: Hans Reitzels Forlag. Kap. 3, s. 145–192.
- Landauer, Thomas K. (1995). *The trouble with computers. Usefulness, usability, and productivity*. Cambridge og London: MIT Press. 550
- Leveson, Nancy G. og Clark S. Turner (jul. 1993). „An Investigation of the Therac-25 Accidents“. *IEEE Computer*, bd. 26, nr. 7, s. 18–41.
- Limoncelli, Thomas A. (jun. 2019). „The top 10 things executives should know about software“. *Communications of the ACM*, bd. 62, nr. 7, s. 34–40. DOI: 10.1145/3316776. 555
- Mandel, Theo (1997). *The elements of user interface design*. New York: John Wiley & Sons.
- McCall, Jim A., Paul K. Richards og Gene F. Walters (nov. 1977). *Factors in Software Quality. Concept and Definitions of Software Quality*. 3 bd. Final technical report RADC-TR-77-369. New York: Rome Air Development Centre.
- Northover, Mandy m.fl. (2008). „Towards a Philosophy of Software Development: 40 Years after the Birth of Software Engineering“. *Journal for General Philosophy of Science*, bd. 39, nr. 1, s. 85–113. DOI: 10.1007/s10838-008-9068-7. 560
- Pressman, Roger S. (2001). *Software Engineering. A Practitioner's Approach*. 5. udg. Boston m.m.: McGraw-Hill.
- Raymond, Eric (2001). *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. Beijing etc.: O'Reilly. 565
- Royce, Winston W. (1987). „Managing the Development of Large Software Systems“. I: *ICSE '87. Proceedings of the 9th International Conference on Software Engineering*. Washington, DC: IEEE Computer Society Press, s. 328–338. DOI: 10.5555/41765.41801. 570
- Santayana, George (2011). *The Life Of Reason. Introduction and Reason In Common Sense*. Red. af Marianne S. Wokeck og Martin A. Coleman. Critical edition. Works of George Santayana 7. MIT Press.
- Seffah, Ahmed og Eduard Metzker (dec. 2004). „The obstacles and myths of usability and software engineering“. *Communications of the ACM*, bd. 47, nr. 12, s. 71–76. DOI: 10.1145/1035134.1035136. 575
- Smith, Thomas M. (jul. 1976). „Project Whirlwind: An Unorthodox Development Project“. *Technology and Culture*, bd. 17, nr. 3, s. 447–464.

## Navneliste

- 580 Brooks, Frederick Phillips [Fred], 3, 4, 8  
Frøkjær, Erik, 13  
Hosier, W. A., 7  
Houston, Frank, 15, 16  
Mandel, Theo, 12  
585 McCall, Jim A., 6  
Miller, Ed, 15  
Raymond, Eric Steven, 7  
Royce, Winston Walker [Winston W.]  
(1929–1995), 8, 9  
590 Santayana, George (1863–1952), 2  
Wirth, Niklaus Emil [Niklaus], 9

## Indeks

- AECL, *Se* Atomic Energy of Canada Limited  
595 agile metoder, 10  
artefakt, 5  
Atomic Energy of Canada Limited (AECL), 14  
automationsniveau, 10  
600 Bell-LaPadula-modellen, 15  
brugbarhed, 6  
bruger-orienteret tilgang til kvalitet, 2  
brugerinterface, 12  
brugeroplevelse, 12  
605 datamatrisering, 5  
digitalisering, 5  
FDA, *Se* U.S. Food and Drug Administration  
Fordismen, 4  
610 GE, *Se* General Electric Company  
General Electric Company (GE), 6  
grundvidenskab, 2  
handlingsorienteret videnskab, 1  
human-computer interaction, 12  
615 IBM, *Se* International Business Machines  
IBM System/360 Operating System, 3, 4  
inkrementiel tilgang til udvikling, 2  
integriteten, 6  
International Business Machines (IBM), 3  
620 korrekthed, 6  
Massachusetts Institute of Technology (MIT), 5  
metaprogrammering, 10  
MIT, *Se* Massachusetts Institute of Technology  
model-baseret udvikling, 9  
mytiske mandemåned, 4  
NATO, 4  
operativsystem, 3  
630 OS/360, 3, 4  
pålidelighed, 6  
SAGE, 5  
software engineering, 2, 4  
systemisk tilgang til software, 2  
635 teknologisk system, 5  
teknologisk videnskab, 1, 2  
Therac-20, 14  
Therac-25, 13–16  
Therac-6, 14  
640 trial-and-error, 2  
U.S. Food and Drug Administration (FDA), 14  
UML, 10  
vandfallsmodellen, 8, 9, 13  
645 videnskab  
grundvidenskab, 2  
handlingsorienteret, 1  
teknologisk, 1, 2  
Whirlwind, 5

# At få computeren til at hjælpe os

Henrik Kragh Sørensen

Mikkel Willum Johansen

20. februar 2023

## Indhold

1	Matematiske modeller og modellering . . . . .	1	5
2	Et legetøjseksempel . . . . .	2	
3	Modellingsprocessen . . . . .	3	
4	Modeller med stor kompleksitet . . . . .	5	

Litteratur	10
------------	----

## 1 Matematiske modeller og modellering

10

Vi er omgivet af og benytter os konstant af forskellige former for modeller for at navigere i verden. Men for at forstå, hvad modeller er, og hvad de kan, er det nødvendigt at kigge lidt mere filosofisk på selve modelbegrebet og på den slags viden, som modeller muliggør.

En model er generelt en form for repræsentation af et genstandsfelt (domæne, det modellen er en model *af*), hvor visse aspekter er udvalgt frem for andre til repræsentationen. Tænk på et kort over metroen i København: Sådan et metrokort er lavet for at det er let at orientere sig mellem stationerne og se, hvor man kan skifte mellem linje M1 og M3. Men metrokortet er fx ikke målfast, så der kan sagtens være længere at gå mellem Lufthavnen og Kastrup end mellem Rådhuspladsen og Hovedbanegården, selvom begge par er nabostationer på kortet. Altså ser vi, at metrokortet er fokuseret på et formål og derfor har prioriteret visse informationer frem for andre. Og de to forhold er generelt gældende for modeller.

15

Modeller er kendtegnet ved deres *epistemiske funktion*. Derved adskiller de sig fra andre repræsentationer: En modeljernbane er ikke en model — med mindre vi kan lære noget ud fra den. Solsystemet er ikke en model af atomkernen — men kan tjene som en effektiv analogi. Men en skaleret arkitekt-model af et højhus kan godt være en model, hvis man fx kan undersøge lysindfald ved hjælp af den — og den er samtidig en visualisering, som man kan danne æstetiske indtryk fra.

20

I videnskaben taler man om forskellige modeller: Mus kan være modeller i kræftforskning, fx hvis musenes organer på vigtige punkter ligner menneskers. Eller rotter kan være modeller for menneskelig adfærd, hvis man kan tillægge rotterne egenskaber som stedsans. Når vi taler om modeller i videnskaben handler det altså om, at **modellen på en eller anden måde** repræsenterer det, som modellen skal være en model af.

25

En anden måde at forsøge at indskærpe, hvad en (videnskabelig) model er, er ved at forsøge at dele det komplekse begreb op i underbegreber eller eksempler. Vi har allerede omtalt eksempler på skaledede, fysiske modeller og på analogier, der ikke har model-karakter.

30

35

I det følgende skal vi behandle flere typer modeller: matematiske modeller, beregningsmodeller og modeller til simulering. Nogle af dem er ret simple, men har alligevel interessante erkendelsesmæssige udfordringer. Andre er meget komplekse, og det rejser blot endnu flere spørgsmål at stille — og forsøge at besvare.

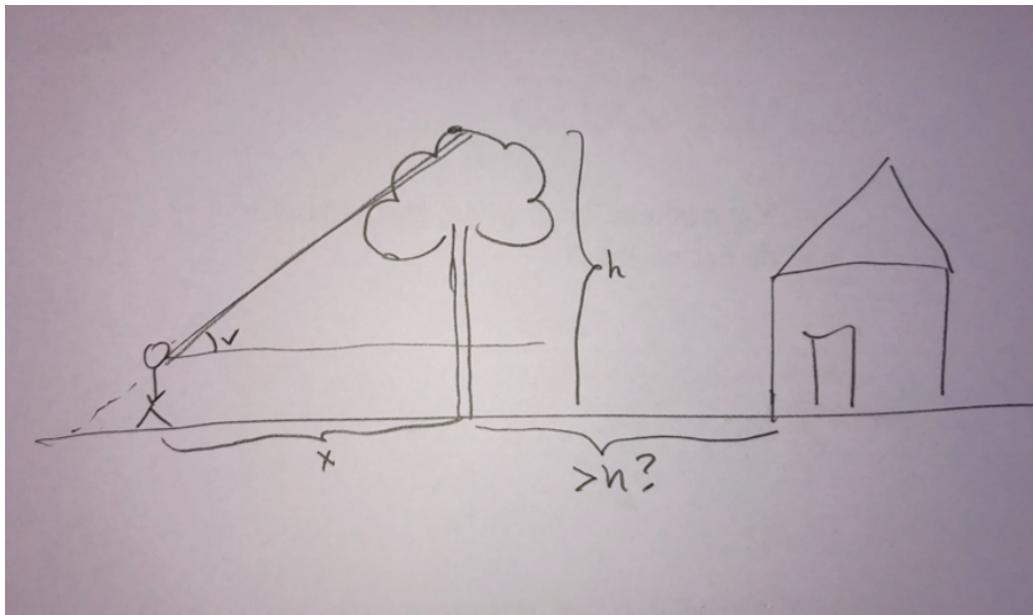
Overordnet kan man inddelte modeller i forskellige typer alt efter, hvordan modellens forbindelse til dens domæne er: Vi taler således om 1. *ikoniske modeller*, hvis modellen *ligner* dens domæne, men fx er nedskaleret i størrelse, 2. *analogi-modeller*, hvis model og domæne ikke ligner hinanden, men alligevel deler en form for metaforisk analogi, og 3. *abstrakte* eller *matematiske modeller*, hvis modellen er en abstrakt og formel repræsentation af sit domæne. Ikoniske modeller kan være enormt brugbare i konstruktioner, hvor man fx kan bygge en model af et hus for at kunne visualisere lysforhold, inden man faktisk begynder at lægge mursten oven på hinanden. Og modellen virker, fordi der er *lighed* mellem modellens dele og domænets dele: modellen *står i stedet for* sit domæne i alle relevante sammenhænge. I forlængelse af denne definition kan man også betragte *virtuelle* modeller (VR) som ikoniske modeller, selvom der ikke foreligger nogen materiel model men derimod en computer-medieret simulation af at bevæge sig rundt i modellen.

Anderledes ser det ud for analogi-modeller, som fx NIELS BOHRS (1885–1962) atommodel, som er baseret på en analogi med vores solsystem: atomkernen er i midten, ligesom solen, og elektronerne kredser i baner om kernen, ligesom planeterne går om solen. Og analogien rækker videre, således at der ud af modellen næsten følger, at der må være en kraft, der holder atomet sammen, ligesom tyngdekraften holder solsystemet sammen. Denne slags modeller kan være heuristiske 'pumper', der kan give ideer, men det er sværere — uden yderligere antagelser — at argumentere for, at de altid virker og fx kan bruges som årsagsforklaringer.

Endelig er der de matematiske modeller, som vi skal beskæftige os mest med her. De matematiske modeller repræsenterer et 'virkelig' fænomen (i den bredeste betydning) ved en formel beskrivelse, og logiske relationer i det formelle system skal så svare til faktiske relationer i virkeligheden, hvis modellen ellers er konstrueret ordentligt. På den måde kan man bruge matematiske modeller både til forklaringer og til at kontrollere og forudsige udsnit af virkeligheden.

## 2 Et legetøjseksempel

Lad os betragte et simpelt eksempel. Lad os forestille os, at jeg skal fønde et træ i min indkørsel, og det naturligt nok er vigtigt for mig, at træet ikke vælter ind i mit hus. Vi identificerer hurtigt, at dette problem kan analyseres ved hjælp af en matematisk model. Og som noget af det første foretager vi os nogle abstraktioner: Vi ser bort fra træets grene, vi er ligeglade med, at huset er et træhus, og garagen på den anden side af huset betyder ikke noget for vores model. Nogle af de ting, vi abstraherer væk, kunne godt være relevante for vores problem, men vi vurderer alligevel, at det er nemmest at se bort fra dem — i hvert fald lige nu. Dermed er vi i stand til at opstille en foreløbig skitse af vores model: træet er skitseret, mit betragningspunkt og husets placering er angivet, men der er endnu ikke sat matematiske begreber og størrelser på modellen. Men hvis vi betegner afstanden mellem mig og træet med  $x$ , højden af træet med  $h$  og sigtevinklen til træets top med  $v$ , så kan vi udtrykke en relation mellem størrelserne som fx  $h = x \tan(v)$ , og det gør det muligt for mig at beregne  $h$  ud fra  $x$  og  $v$ , som jeg kan måle (se figur 1).



80

Figur 1: En skitse af den matematiske model, som skal hjælpe mig til at fælde mit træ.

Dermed kunne vi se ud til at være færdige— jeg kan beregne træets højde og sikre mig, at det ikke rammer huset, når jeg fælder det.

Men det kan være, at det kommer meget tæt på, og jeg bliver bekymret. En grund til bekymring kunne være, at jeg ikke er sikker på, hvor præcise mine målinger er. Så jeg bliver nødt til at sige noget om usikkerheden på min beregning af  $h$ . Det kan jeg fx gøre ved at se, hvad der sker, hvis jeg lægger lidt til eller trækker lidt fra  $x$ . På den måde kan vi bruge den matematiske model til at regne forskellige scenarier igennem (se nedenfor), hvilket kan være meget effektivt som en slags eksperimenter i situationer, hvor vi ikke bare kan foretage et fysisk eksperiment.

85

Men der er også nogle idealiseringer, som vi har foretaget i selve modellen, som kan have endog stor indflydelse på modellens forudsigelser. I vores konkrete skematiske form er der taget hensyn til, at jeg er to meter høj, hvorfor vinklen  $v$  faktisk skulle måles et andet sted. Og vi har antaget, at træet faktisk vokser vinkelret op, ligesom vi slet ikke har taget hensyn til, at jordens overflade jo er krum. Dette er idealiseringer fordi vi faktisk godt ved, at de er kontrafaktiske antagelser—men de har hjulpet os til at bygge modellen. Hvis det nu viser sig, at modellen er utilstrækkelig af en eller anden grund, så må vi jo genbesøge vores idealiseringer og se, om vi kan forbedre dem.

95

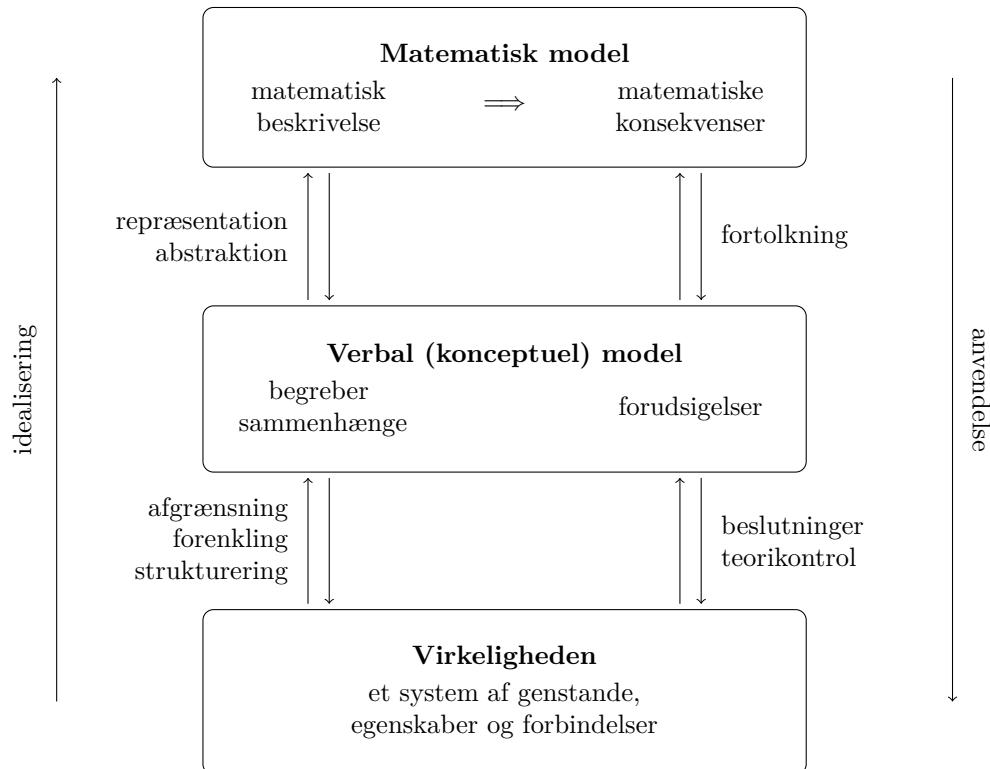
Selvom dette er et legetøjseksempel, viser det nogle af de principielle udfordringer, som er nødvendige at reflektere over for at lave en faglig-kritisk vurdering af matematiske modeller: Hvilke antagelser er bygget ind i modellen? Hvilke faktiske forhold er der set bort fra? Hvilke matematiske og evt. statistiske begreber og beregninger er involveret? Hvad betyder det for modellens usikkerheder? Hvordan fortolkes og formidles modellens resultater?

100

### 3 Modelleringsprocessen

Man kan illustrere de væsentligste dele af den matematiske modelleringsproces som i figur 2, hvor et udsnit af virkeligheden afgrænses til en verbal model, der *matematiseres* til

105



Figur 2: Matematisk modellering er en iterativ og dialektisk proces (Johansen og Sørensen, 2014, s. 181).

Det, vi netop har skitseret, vil være et enkelt gennemløb af processen i figur 2, men det  
115 er nogle af de vigtigste indsigter i matematiske modeller inden for de sidste årtier, at denne  
proces er **dialektisk** og **iterativ**. At den er **dialektisk** betyder, at hvert skridt i modellen er en  
afvejning af muligheder og behov: Der er ikke noget vundet ved at udlede en matematisk  
beskrivelse, som man ikke har redskaberne til at undersøge — så er det bedre at forenkle  
den begrebslige model yderligere ved at se bort fra nogle faktorer, selvom man ved, at de  
120 nok øver indflydelse på den modellerede opførsel. Men der kan også være mange andre  
hensyn end denne slags **pragmatik** på spil: Hvis man fx er ude efter at bruge modellen til  
at forstå et fænomen, vil man ofte vægte simplicitet frem for stor forudsigelsespræcision,  
og den slags værdier (cf. THOMAS KUHNS (1922–1996) paradigmer) spiller også centralt  
ind i modelleringsprocessen. At modelleringsprocessen er iterativ skyldes, at selv efter nok  
125 så mange gennemløb af processen, er den opnåede model kun 'foreløbig', men forhåbentlig  
anvendelig viden. Hvis modellens (fortsatte) brug forudsætter større præcision eller en mere

nuanceret inddragelse af faktorer, må man udvide eller forfine modellen, og dette vil føre til gentagne gennemløb af processen i figur 2.

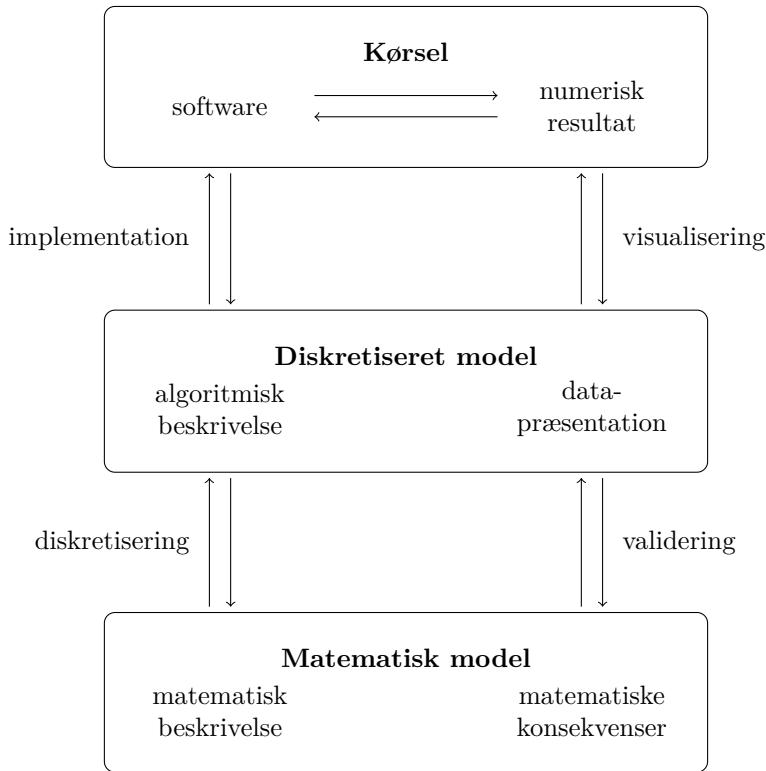
## 4 Modeller med stor kompleksitet

I nogle af de områder, hvor man særligt bruger modeller, og som samtidig rejser vigtige filosofiske spørgsmål, modellerer man meget komplekse systemer. Et særligt interessant problem opstår, når den komplekse model opstår som en *aggregering* af mindre størrelser: enten aktører eller andre modeller. Sådan nogle komplekse modeller opstår fx i økonomi eller klimamodellering. 130

I økonomiske modeller er der en stående diskussion omkring relationen mellem de enkelte agenter (fx borgere), som foretager handlinger på *mikroniveau*, og de effekter, som kan aflæses på det overordnede niveau (fx BNP), som kan ses på *makroniveau*. I denne slags modeller består aggregeringen altså i at samle de mange individuelle aktørers handlinger, som er styret af antagelser (og dermed modeller) på mikroniveau, til nogle af de oplysninger på makroniveau, som økonomen er interesseret i. 135 140

I klimamodellering kan man sjældent holde den enkelte model op imod en objektiv virkelighed for validering, og i stedet bruger man nogle gange *andre modeller* til at bedømme og korrigere sin model. Hvis man således har en samling af modeller, måske udviklet i forskellige laboratorier og helst ud fra forskellige grundantagelser, så kan man håbe på, at en *aggregeret* model kan være bedre end nogen af de enkelte modeller, dels fordi den måske tager højde for flere faktorer i alt, og dels fordi den typisk vil være mindre sensitiv, hvis den fx er dannet som et gennemsnit af andre modeller. Men hvordan man foretager denne aggregering, og om alle de indgående modeller vægtes ligeligt, er beslutninger med store konsekvenser, som man kan have svært ved at forsvarer ud fra generelle principper. 145

For sådanne komplekse modeller vil man ofte benytte computer-simulationer, hvor man så kan regne på forskellige scenarier for at følge konsekvenserne. På den vis kan man i nogle henseender se computersimulationer som en afart af klassiske eksperimenter, og nogle gange som tankeeksperimenter. Hvis vi opfatter komplekse modeller som alternativer til traditionelle eksperimenter, så udmaærker simulationerne og især scenarietænkningen sig jo ved, at man kan foretage 'eksperimenter' *in silico*, som man ikke ville kunne foretage i virkeligheden, enten fordi tidsperspektivet er for langt (fx klimamodeller) eller fordi eksperimentet er dyrt, etisk uforsvarligt eller farligt (fx simulationer af nukleare processer). 150 155



Figur 3: Udvidelse af skemaet om matematisk modellering for at tage hensyn til beregningsmodeller. Denne del af skemaet kan udgøre udforskning af forskellige *scenarier* ved at variere nogle af de indgående parametre, som ellers er konstante i hver gennemregning af modellen.

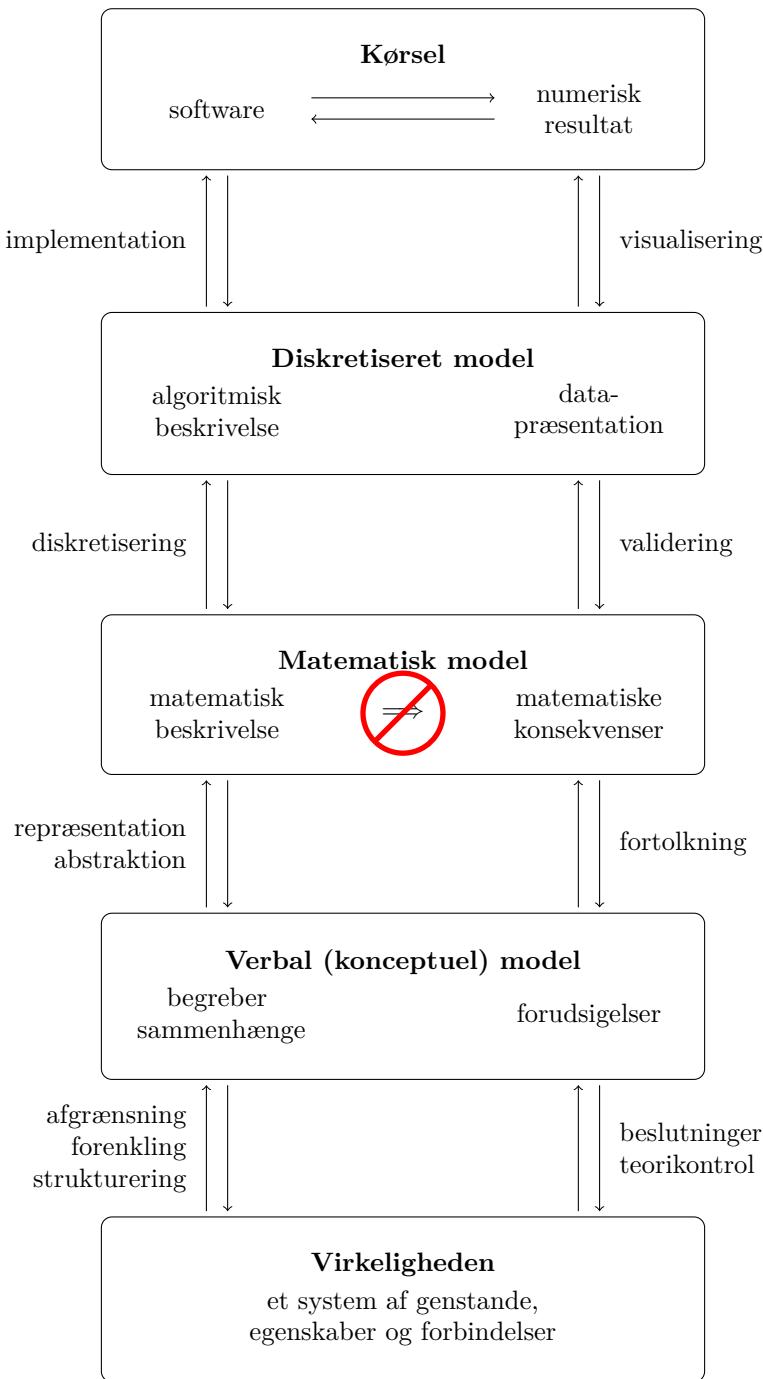
Ud fra disse repræsentationer af resultaterne er det ofte muligt at foretage en foreløbig 160 og delvis *validering* af beregningsmodellen ved at sammenholde data med betingelser, man har kunnet udlede af den matematiske model. Det kan være helt simple observationer som fortegn etc., men det kan være nok så relevant til at fange *numeriske artefakter*, dvs. opførsel, som ikke hidrører fra modellens teoretiske antagelser og matematiske formulering, men er opstået under implementationen og kørslen af beregningsdelen. Endelig kan resultaterne så 165 lede til forudsigelser og beslutninger omkring det udsnit af virkeligheden, som modellen er bygget til at handle om.

Ligesom det er tilfældet for matematiske modeller mere generelt, er denne modelleringsproces tænkt som både iterativ og dialektisk. Den er først og fremmest *dialektisk* i den forstand, at valg og beslutninger påvirker modelleringsprocessen både 'fremad-' og 'bagudrettet' (alle pile i det samlede skema i figur 4 peger i begge retninger): Kendskab til hvilke 170 diskretiseringer, der kan lede til effektive implementitioner, har oftest en indflydelse på, hvilke former for afgrænsning og abstraktion, der giver mening i de forudgående led. Og omvendt vil valget af implementation såsom sprog og skalering af arkitektur selvfølgelig afhænge af egenskaber ved den algoritme, som man ønsker at implementere. Og modellen skal også forstås som *iterativ* i den forstand, at modelleringsprocessen gennemgås flere gange i takt med at resultaterne peger på muligheder for forfinelser og udvidelser. Dette kan være drevet både af utilstrækkeligheder i den tidlige model og af nye muligheder i form 175 af større beregningskraft, flere data, eller nye faktorer, som skal integreres.

Men der er også en yderligere form for gentagelse, som gør beregningsmodeller meget brugbare, og som typisk ikke består ved almindelige matematiske modeller. Man kan nemlig variere nogle af de modellerede parametre og køre beregningen igen. Derved kan man få et indblik i disse parametres indvirken på resultaterne, og dette bruges til såkaldte *scenarier*. Dette er en anden måde at bruge beregningsmodeller på, som man kan sammenligne med en slags *udforskende eksperimenteren* (eng: *exploratory experimentation*, se Steinle, 2016), hvorved beregningen bliver mere til et eksperiment end en test af en hypotese.

180

185



Figur 4: Samlet skematisk oversigt over processerne i beregningsmodellering.

## Modellering af smittespredning: en simpel beregningsmodel

Når man skal modellere sygdomsspredning, er der rigtigt mange faktorer, man kunne forestille sig have indflydelse: sygdommens alvorlighed og spredningsform, befolkningens sammensætning og spredning, tiltag for at begrænse spredningen etc. Smittespredning er derfor et eksempel på et komplekst problem, som man ikke kan gøre sig forhåbninger om at finde en eksakt matematisk beskrivelse af. I stedet benytter man sig af beregningsmodeller, som kan udvikles og tilpasses de specifikke omstændigheder ved en given sygdom.

I kernen af rigtig mange smittemodeller ligger en forholdsvis simpel afgrænsning af problemet, idet man som udgangspunkt antager, at befolkningen er stabil inden for den horisont, hvor modellen skal virke: Der sker ikke ændringer i befolkningens størrelse eller sammensætning, og man antager fx at befolkningen er konstant og homogen. Yderligere antagelser, som man typisk gør sig i denne slags første modeller på befolkningsniveau er, at befolkningen med hensyn til denne sygdom kan inddeltes i tre disjunkte kategorier:  $S$  som er modtagelige (eng: *susceptible*) for sygdommen,  $I$  som er inficerede med sygdommen, og  $R$  som enten er døde eller er blevet immune over for sygdommen, og derfor er fjernet fra den relevante population (eng: *removed*). Hvert individ kan altså følge en progression  $S \rightarrow I \rightarrow R$ , og modellen kaldes derfor *SIR-modellen*. Dermed er vi på vej til at indskrænke et udsnit af virkeligheden (smittespredning af en given sygdom) til en *verbal model* (befolkningen er konstant og homogen, og sygdommen spredes på en bestemt måde).

For at kunne opstille en matematisk model er vi nødt til at gøre nogle yderligere antagelser og indføre noget notation. For det første kan vi omsætte vores antagelse om konstant befolkning til  $N = S(t) + I(t) + R(t)$ , hvor  $t$  angiver vores tidsparameter. Og hvis vi antager, 1. en modtagelig har risikoen  $\frac{I(t)}{N}$  for at møde en smittet og en konstant risiko,  $\beta$ , ved hvert sådan møde for selv at blive smittet, og 2. at smittede dør eller opnår immunitet med en fast sandsynlighed  $\gamma$ , så kan vi begynde at formulere vores matematiske model i termer af ændringshastighederne i de tre kategorier:

$$\begin{aligned}\frac{dS}{dt} &= -\frac{\beta}{N} \cdot I(t) \cdot S(t), \\ \frac{dR}{dt} &= \gamma \cdot I(t),\end{aligned}$$

som betegner tilføjede og fjernede smittede, hvorfor

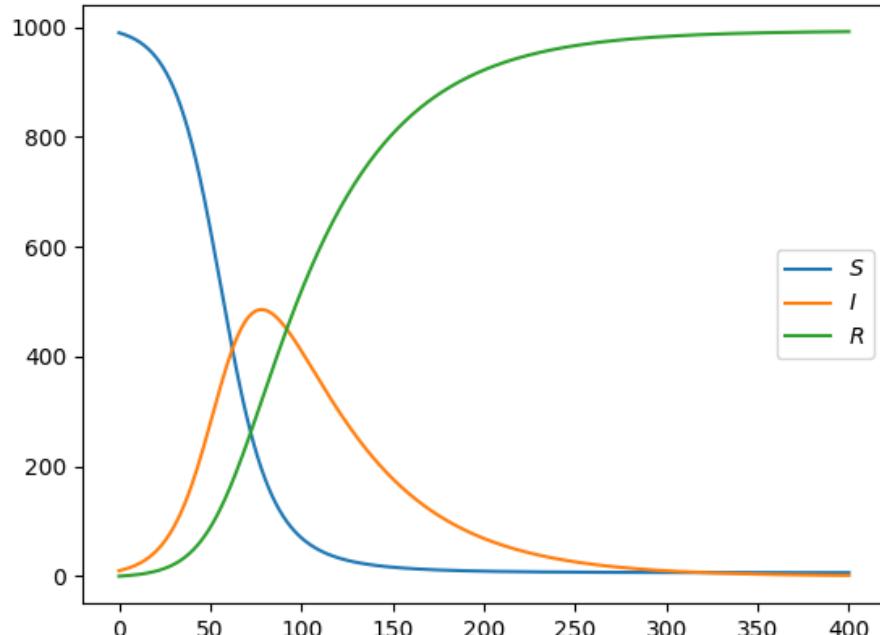
$$\frac{dI}{dt} = \frac{\beta}{N} \cdot I(t) \cdot S(t) - \gamma \cdot I(t).$$

Derved har omsat vores begrebslige model til en *matematisk beskrivelse* i form af tre koblede differentialligninger. Man bemærker her, at der også er en hel del faktorer, som nok konkret er vigtige, som vi ikke har medtaget i vores verbale model; fx problemet omkring latenstid, idet vi antager, at alle inficerede selv smitter videre med konstant sandsynlighed fra første øjeblik.

Nu kunne man tænke sig, at dette sæt koblede differentialligninger tillod en eksakt løsning — men det er generelt ikke tilfældet, og fordi modellen er så forsimplet har man lyst til i næste iteration at tilføje komplexitet, fx ved at gøre parametrene  $\beta, \gamma$  variable. Derfor tyr man i stedet til beregningsmodellering. Men før man kan oversætte systemet af

differentialligninger til noget, en computer kan regne effektivt på, skal den matematiske formulering diskretiseres. Afhængig af problemfeltet findes der forskellige etablerede metoder til at diskretisere, men i det konkrete tilfælde vil det sikkert være så simpelt som at betragte tiden som en diskret følge med konstante spring. Så i stedet for at betragte  $t$  som en kontinuert variabel og fx  $\frac{dS}{dt}$  som instantan hastighed, betragter vi i stedet tiden som en diskret størrelse og undersøger udviklingen  $\Delta S(t) = S(t + \Delta t) - S(t)$ , hvor  $\Delta t$  er en fast (og ’lille’) konstant. Så er  $\frac{\Delta S}{\Delta t}$  en diskret approximation til  $\frac{dS}{dt}$ , og den matematiske model kan laves om til en *algoritmisk beskrivelse*, hvor vi lader  $t$  gennemløbe værdierne  $t_0, t_0 + \Delta t, t_0 + 2\Delta t, \dots, t_1$  og hele tiden opdaterer, fx  $S(t + \Delta t) = S(t) + \Delta t \cdot \Delta S(t)$ . 215 220

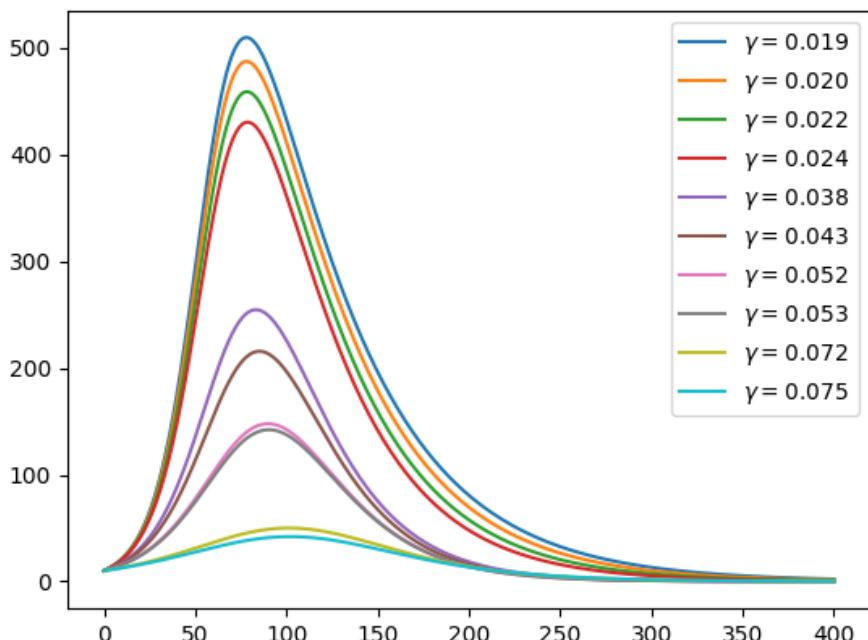
Denne algoritme skal nu implementeres i et programmeringssprog med henblik på afvikling på en given arkitektur, dvs. som et stykke *software*. Og som omtalt i kapitel ?? giver det anledning til en række overvejelser, for både sprog og arkitektur kan være mere eller mindre egnede til den konkrete opgave. Til denne slags simuleringsopgaver vil det generelt være en god ide at benytte fx Python eller Fortran, som er udviklet specifikt til videnskabelige beregninger, og som har masser af biblioteker, der understøtter det. Men hvis man fx valgte at implementere det i Excel eller C eller et andet sprog med en fast, endelig opløsning i repræsentationen af reelle tal, kan man risikere at visse numeriske beregninger går galt på grund af afrundingsfejl. En implementation i Python kan ses i appendix ??, og når den bliver afviklet på en understøttet arkitektur, giver den anledning til en række talværdier for udviklingen af  $S, I, R$  over tid. Dette er det *numeriske resultat* af kørslen. Meget ofte vil det være meget mere ønskeligt at bruge computeren til også at fremstille andre *repræsentationer af resultaterne*, fx gode visuelle repræsentationer af forløbet over tid (en sådan graf er gengivet i figur 5). 225 230 235



Figur 5: Grafisk repræsentation af forløbet af  $S, I, R$  for givne parametre.

Modellens resultater kan så holdes sammen med viden om den matematiske model.  
 240 For eksempel udviser kurverne i figur 5 det forventede overordnede forløb, så vi har ikke umiddelbart grund til at betvivle den numeriske beregning. Og selvfølgelig skal modellens resultater også bringes i spil til at lave de forudsigelser eller beslutninger, som var formålet med at bygge den i første omgang.

I figur 6 er gengivet, hvordan antallet af smittede  $I$  afhænger af parametren  $\gamma$  for et  
 245 antal forskellige (her normal-fordelte) værdier. Dermed kan man fx begynde at estimere denne parameter ud fra fx kendte data eller ud fra kapacitetsbetragtninger, hvilket måske ikke kan gøres eksplisit. Hvis fx kapaciteten af en eller anden grund er på 200 inficerede, kan man ud af scenarierne aflæse, at man skal stræbe efter en værdi  $\gamma \geq 0.043$ .



Figur 6: Grafisk repræsentation af forløbet af  $I$  under forskellige scenarier.

## 250 Litteratur

- Johansen, Mikkel Willum og Henrik Kragh Sørensen (2014). *Invitation til matematikkens videnskabsteori*. København: Forlaget Samfundslitteratur.
- Steinle, Friedrich (2016). *Exploratory Experiments. Ampere, Faraday, and the Origins of Electrodynamics*. Overs. af Alex Levine. Pittsburgh: University of Pittsburgh Press.

# Kapitel 6:

## At få computeren til at hjælpe os: Del II — datadrevne modeller

Henrik Kragh Sørensen

Mikkel Willum Johansen

20. maj 2023

### Indhold

6.1	Modeller med meget store mængder data	1
6.2	Proxies: Hvad vil vi — og hvad kan vi — måle?	2
6.3	Data er altid beskidt: Indsamling og kuratering af data	3
6.4	Teori og model er underbestemt af data	4
6.5	Machine-learning-modeller	4
6.6	Skilsmisser og margarine: Korrelationer og årsager	8
6.7	FFF: forklaring, fortolkning og forsvar af modeller	12

### Litteratur

17

Der vil givetvis være en række slåfejl, uklarheder og (forhåbentlig få) fejl. Informationen om slåfejl og forbedringsforslag modtages meget gerne.

### 6.1 Modeller med meget store mængder data

En anden type modeller, som særligt lægger op til brug af computere og datalogi, handler om at modellere fænomener ud fra meget store mængder data. Hvor der indtil midten af 1900-tallet var mangel på data, står vi i dag overfor en situation, hvor der er alt for meget data, til at vi kan behandle hvert stykke data udelukkende på dets egne præmisser. Derfor har vi brug for at *aggregere* data og behandle data med statistiske redskaber. Nogle sådanne statistiske metoder er klassiske i den forstand, at både metoderne og deres begrænsninger velkendte. Mange af disse begrænsninger handler dybest set om at have det rette kendskab til metoderne, så man kan anvende den bedst egnede metode i en given sammenhæng. Men der er også nogle mere principielle, videnskabsteoretiske vinkler, hvoraf vi vil behandle nogle udvalgte i det følgende.

## 6.2 Proxies: Hvad vil vi — og hvad kan vi — måle?

To af de allerførste overvejelser, man typisk gør sig, når man skal bygge en model, er: 'hvad skal modellen måle?' og 'hvordan vi måle det?'. Det er nemlig langt fra alle interessante oplysninger, vi kan måle direkte. I stedet anvender vi andre størrelser, som vi kalder for *proxies*, som det er muligt og evt. nemmere at måle, og som vi med mere eller mindre begrundelse anser for at være forbundet med det, vi i virkeligheden gerne ville have målt.

For eksempel kan det være, at vi gerne vil modellere klimaets udvikling igennem de sidste 10,000 år (dvs. siden starten af den nuværende mellemistid), og dertil har vi brug for at vide noget om atmosfærens temperatur. Men det er lettere sagt end gjort. For det første er der jo nogle uklarheder i spørgsmålet: Skal det være temperaturen hvert sekund de over disse mange år, temperaturen på denne dato igennem 10,000 år, eller fx middeltemperaturen over et års forløb.<sup>1</sup> Det sidste er nok både det mest realistiske og også teoretisk velbegrundet: Vi ved, at temperaturen varierer over årets gang, så det giver mening at aggregere på det niveau.

Men selv efter, at vi har afgrænset denne art af pragmatiske, men teoretiske valg, er vi endnu ikke i mål. Vi kan jo ikke rejse tilbage med et termometer, så selve temperaturmålingerne er nødt til at foregå *via proxy*. Dermed menes, at vi observerer en anden størrelse end den, vi faktisk er interesseret i. I eksemplet med tusinde år gamle temperaturer måler man fx forholdet mellem koncentrationer af to forskellige stabile iltisotoper i de dybe isboringer, man har foretaget i Grønland (Dahl-Jensen, 2009). For at koble iltisotoper og middeltemperaturer sammen kræver det ret omfattende teoretisk og empirisk begrundelser. Men det er nærmere et vilkår for empirisk og modelbaseret videnskab.

Nogle af disse oversættelser mellem *teoretiske* data og de *observerbare* data, som vi sætter i stedet, er både gamle, anerkendte og velbegrundede. Optimalt ville vi jo gerne vide, at vores proxy data er stærkt *korreleret* med vores teoretiske data; så ville observation af proxy jo tilsyneladende blot være en *indirekte* observation af teoretisk data. Men ofte kan vi ikke uden videre være sikre på, at de to størrelser faktisk er stærkt korrelerede, og selvom de måtte være det, er der indbyggede farer ved at lægge for meget vægt på korrelationer i store datamængder (se 6.6 nedenfor om korrelation).

Nogle *proxies* har en anden fordel ud over, at de er forbundne med det, vi ønsker at måle: De er blevet *kanoniske* i den forstand, at de har vist sig så bredt anvendelige og rimeligt målbare, at de er tæt på at blive opfattet som sel vindlysende og uproblematiske. Faktisk kan de være medbestemmende for hele teorier og centrale elementer i *disciplinære matricer* for visse felter, fx inden for medicin. Når læger ønsker at *diagnosticere* sygdomme (fx udbredte folkesygdomme), er deres første *proxies* fx *BMI* (body mass index), blodtryk og oplysninger om arvelige sygdomme. De to første af disse *proxies* indgår i en kompleks vekselvirkning mellem lægelige ønsker og teknologiske muligheder: Jo mere *BMI* er blevet brugt diagnostisk, jo mere relevant er det, at kunne måle BMI og vurdere de resulterende data. Der er altså ikke blot tale om, at fx *BMI* er en kraftigt forsimplet *proxy*, idet den ser bort fra mange dimensioner i data, som vi har grund til at antager, har betydning. Men udviklingen af denne slags proxies er altså også resultat af en social og teknologisk udvikling, der *oversætter* mellem de teoretiske data og deres *proxies* (se nedenfor om modellers *performativitet*). Andre *proxies*, som her fx arvelige sygdomme, er ikke til at *måle* fra det enkelte individ, så i stedet forsøger læger og andre at opbygge databaser med genetisk

<sup>1</sup>Tilsvarende spørgsmål gælder naturligvis også den rumlige dimension: Taler vi om Danmark, om Grønland, om den nordlige halvkugle, eller en helt anden afgrænsning?

information eller skaffe elektronisk adgang til patientjournaler, således at også denne slags information kan indsamles, gemmes og bruges *diagnostisk*.

Som det præsenteres her, er proxies på flere måder sammenlignelige med de idealiseringer, der foregår i modelleringsprocessen: Vi sætter noget andet i stedet for det, vi egentlig er interesserede i, for overhovedet at muliggøre processen. I yderste tilfælde er en proxy en kontrafaktisk idealisering. Og ligesom andre idealiseringer, er valg og udvikling af proxies ikke værdi-neutrale. Faktorer som sædvane, tilgængelighed, og effektivitet er *pragmatiske* hensyn og værdier, som vi med rette kan anlægge. Deres effekt vil hyppigt have betydning for, *hvorvidt* modellen virker og er altså af epistemisk karakter. Men der kan også indgå mere udtalte værdier, hvis vi fx vælger proxies, der ser bort fra individers særinteresser (fx ved aggregering). I så fald kan valget af proxies siges at have indflydelse på, *hvordan* og *hvornår* modellen virker, og de kan derfor siges desforuden at have en etisk karakter. Som vi skal se nedenfor kan proxies endda have en *performativ* karakter, idet de kan påvirke den virkelighed, som de egentlig er tænkt som idealiseringer af.

### 6.3 Data er altid beskidt: Indsamling og kuratering af data

Som forklaret i kapitel 6a kan man inddøle modeller i teori- og datadrevne modeller. Hvor teoridrevne modeller (herunder særligt matematiske modeller) er beskrevet i kapitel 6a, opstår der nogle særlige udfordringer for data-drevne modeller — og den mest fundationale udfordring er netop *data*. For hvor kommer *data* fra, hvordan er det indsamlet og behandlet, og hvor repræsentativt er det? Det er problemer, der altid kan stilles til data, men når datamængderne bliver så store, at det ikke er menneskeligt muligt at undersøge kvaliteten af data *i hånden*, bliver problemet kun endnu større og mere akut.

Det kan være fristende at anlægge det synspunkt, at de data, som vi indsamler til vores modeller, bør være så *rå* som muligt for at give størst mulig *objektivitet*. Hvad der præcist menes med disse begreber kan være svært at forklare, men man kan måske forestille sig, at *rå data* er karakteriseret ved at være uafhængige af subjektive, menneskelige faktorer — *rene faktuelle oplysninger* om verden.

Men som vi allerede stiftede bekendtskab med i kritikken af logisk positivisme (se kapitel 1), er der principelle indvendinger imod *fordomsfrie* (teorfrie) observationer: Hvis vi bare skal *observere*, hvad skal vi så observere, hvordan skal vi gøre det, og hvilke delelementer er vigtige end andre.

I stedet for at opfatte og tilgå data som *rå*, giver det et mere passende nuanceret billede, hvis vi går til data som en indsamlingsprocess, hvor menneskelig interaktion er uundgåelig i form af data-design og (især) *kuratering*.

Ved begrebet *kuratering* forstår vi her den (oftest hermeneutiske) proces, hvorved *genstandsfeltet afgrænses*, data bliver målt (igenem proxies), data bliver annoteret ved *metadata*, data bliver filtreret (ikke kun for outliers, men ofte også med henblik på *stratificering* og andre systematiske forskelle i indsamlingsprocessen), data bliver holdt opdateret (hvad det end betyder i sammenhængen). Det er her vigtigt at pointere, at under opfattelsen af data som en *kurateringsproces*, er ingen af disse skridt i sig selv problematiske, så længe der opretholdes en form for *transparens* om dataprocessen (se nedenfor), og modellen vurderes under hensyntagen til denne *kuratering*.

## 6.4 Teori og model er underbestemt af data

Selv når vi har anerkendt, at vi ikke måler det, vi direkte ønsker men i stedet oftest (altid) mäter via *proxies*, og at vi ikke mäter på så data alligevel, så udestår stadig udfordringer i form af implicitte eller (forhåbentlig) eksplikt valg (Johansen og Sørensen, 2018): Hvilken model er egnet til at behandle hvilke data — og hvad stiller modelvalget af krav til data? Igen er disse udfordringer ikke udelukkende relevante for datadrevne modeller, men de antager en særlig form, når datamængderne er så store, at vi ikke kan være sikre på at kunne overskue dem på nogen direkte vis.

Alle disse valgprocesser er at sammenligne med de *dialektiske*, *pragmatiske* og kreative skridt i den venstre side af modelleringsprocessen illustreret i figur ???. Den venstre side i modelleringsprocessen for teoridrevne modeller handler om *konceptualisering* og *idealisering* af et udsnit af virkeligheden til en teori, hvis kvalitet (skal) vurderes ud fra værdiladede kriterier som *forudsigelsespræcision*, *forklaringskraft* eller endnu mere subjektive værdier som *simplicitet* og *skønhed*.

### Logisk positivisme 2.0

I begyndelsen af det 21. århundredes første årti, mens *big data* stadig var en helt ny og potentiel revolutionerende mulighed for videnskabelig forskning, blev der fremsat adskillige optimistiske, i dag at se som ret utopiske, bud på, hvordan den videnskabelige erkendelsesproces stod foran en stor transformation. En af de mest udtalte fortalere for den nye vision var CHRIS ANDERSON, som i en kort artikel i magasinet *Wired* annoncerede „The End of Theory“, hvori han sparkede liv i en ny form for logisk positivisme (Anderson, 2008). Han hævdede eksplikt heri, at ikke blot ville der ikke længere være behov for teorier i videnskaben, men også at hele ’den videnskabelige metode’ var blevet overflødig.

ANDERSON var ikke den første, til at annoncere den eksisterende, klassiske videnskabelige metode for død. Allerede tiår forinden havde STEPHEN WOLFRAM foreslået noget, der på visse punkter lignede. Men hans pointe var alligevel en anden: I stedet for empirisk verifikation foreslog han en *beregningsbaseret* videnskab, hvor computerberegninger kunne erstatte mange empiriske forsøg (Wolfram, 2002). Men WOLFRAM s argument forudsatte, at videnskabens teorier allerede var formulerede: Den var ikke en *heuristisk* men en *verifikationistisk* metode.

I forhold til WOLFRAM gik ANDERSON altså hele linen ud: De nye enorme datamængder kunne udgøre det for positivisternes *objektive* og *fordomsfrie observationer*, og anvendelsen af statistiske metoder i stor skala kunne udtømme mulighederne for teorier og dermed nå frem til *lovmaessigheder* ved en *induktiv* proces. Dermed var cyklussen komplet: Den logiske positivismus var blevet etableret igen — i en *in silico* version. Men som vi skal se nedenfor i afsnittet om korrelationer, møder denne nye form for automatiseret teoridannelse nogle uoverstigelige statistiske barrierer.

## 6.5 Machine-learning-modeller

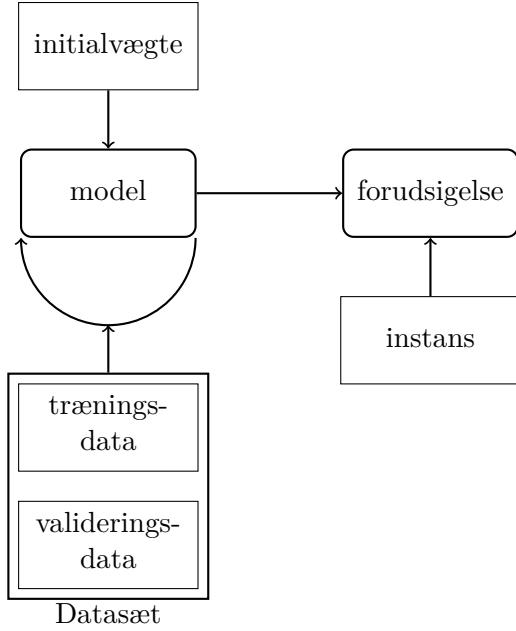
I det følgende vil vi beskrive machine-learning-modeller som *algoritmer*, selvom det er en diskussion i sig selv, hvorvidt og hvordan de opfylder sædvanlige definitioner af algoritmer, som fx den givet af ROBIN K. HILL og beskrevet i kapitel 3.

*Machine-learning-modeller* kommer i forskellige *familier* og til forskellige formål. Formålene kan være *klassifikation*, *rangordning*, *oversættelse* eller *genkendelse* og den præcise

anvendelse kan finde sted inden for forskellige områder, fx behandling af *naturligt sprog* (*NLP, natural language processing*) eller billedanalyse (*CV, computer vision*). Begge disse formål er for øvrigt gamle klassikere inden for forskning i og finansiering af kunstig intelligens: NLP i forsøget på at automatisere oversættelse, konkret oversættelser fra russisk til militære formål og behandling af visuel information i (autonome) robotter, igen med oplagt militær bevågenhed.

Hvis vi skal indkredse typerne eller familerne af *ML-modeller*, så er der forkellige niveauer, man kan anlægge til at skelne. Machine-*learning* handler jo, som navnet siger, om, at algoritmen *lærer* og derved udvikler sig. Man kan forsøge at anlægge dette perspektiv til en beskrivelse af forskellige typer læring: *ikke-supervisoreret læring* er former for statistiske analyser, der forsøger at uddrage information fra data *uden* et niveau af menneskeligt verificeret basisviden. For eksempel kan man lave *clustering algoritmer*, der inddeler et datasæt bestående af  $n$ -dimensionale vektorer i grupper, hvis elementer ligger tæt på hinanden og langt fra de andre grupper. Vi siger, at *datasættets størrelse* er *antallet* af vektorer i datasættet, mens vi bruger betegnelsen *datasættets dimension* til at angive *længden* af hver vektor. Hvad den underliggende mening af disse vektorer måtte være er ikke vigtigt for kliedannelsen, så derfor er der — efter at datasættet er dannet — ikke yderligere brug for menneskelig overvågning af algoritmens læring. Men for at kunne anvende denne slags algoritmer i praksis kan man være nødt til at lægge forskellige betingelser ned over deres proces for at de kan bruges i anvendelser og fortolkning. For eksempel er antallet af kliker en størrelse, som er vigtig for cluster-algoritmer, men hvor det er umuligt at give en *meningsfuld* værdi uden at der er mennesker involveret på dette trin i processen.

Der findes også en meget stor og vigtig klasse af algoritmer baseret på menneskelig involvering i træningsprocessen, mere specifikt at algoritmen er trænet ud fra data, der er beriget med en form for mening. Denne familie af algoritmer betegnes *supervisoreret læring*. Pointen med denne type modeller er, at de ud fra de tidlige *labels* givet til data i træningssættet kan forudsige labels for data, som modellen ikke hidtil har mødt. En skematisk illustration af, hvordan en ML-model af denne type kan trænes og bruges til forudsigelser er givet i Figur 1. Disse modeller består af et meget stort antal *vægte* (tal mellem 0 og 1), som *trænes* ud fra data i træningsdata, således at modellens forudsigelser af labels for data fra valideringsdata bliver bedst muligt. Denne forudsigelse består i at beregne virkningen af vægtene på en given instans, og den resulterer typisk i et tal mellem 0 og 1, som ofte kan fortolkes som en form for sandsynlighed. Træningsprocessen kan eventuelt tage udgangspunkt i et sæt af *fortrænede vægte*, som er beregnet på forhånd til en bestemt anvendelse, for eksempel billedklassifikation. Modellerne kan også trænes videre ved at tilføje nye data til trænings- og valideringssæt og dermed øge *størrelsen af datasættet*.



Figur 1: Skematisk fremstilling af træning og fortolkning ved superviserede ML-modeller (se også en yderligere udfoldet version i Figur 7).

### Eksempel: En binær classifier

For at illustrere nogle epistemologiske aspekter ved superviserede ML-modeller, kan vi tage udgangspunkt i en af de simpleste typer modeller overhovedet, nemlig *binær classifier*. Dette er en type *supervised* model, som skal klassificere data i *to* disjunkte klasser *A* og *B* — det kan fx være, at data er træfrugter, og modellens opgave er at adskille æbler (*A*) og pærer (*B*). Ingen frugt kan være både et æble og en pære, så klasserne er disjunkte (ikke-overlappende,  $A \cap B = \emptyset$ ), og alle data, som meningsfuldt kan gives til modellen er elementer i  $A \cup B$ .

Hvis denne klassifier er baseret på *CV* modeller, vil den altså tage udgangspunkt i et billede af en frugt og forsøge at placere billedet i den klasse, der passer bedst. Som træningsdata vil modellen have en række billeder af frugter, som alle er annoteret med en label, der angiver, om billedet viser et æble eller en pære. Træningen består i, at justere modellens vægte således, at modellens forudsigelser på data fra valideringssættet er mest muligt korrekte forstået som at den label, som modellen forudsiger stemmer overens med den label, som billedet er annoteret med på forhånd.

Der er nu fire tilfælde at betragte (se Figur 2), nemlig følgende, hvis vi betragte æbler som positive og pærer som negative (tænk på det, som at modellen skal fjerne de uønskede pærer fra et samlebånd med æbler og pærer).

modellens forudsigelse	annoteret label	tilfældets navn
æble	æble	Korrekt positiv (TP, true positive)
æble	pære	Forkert positiv (FP, false positive)
pære	æble	Forkert negativ (FN, false negative)
pære	pære	Korrekt negativ (TN, false negative)

Figur 2: Mulige resultater af en binær classifier. Man kalder sommetider falske positiver for *Type-I fejl* og falske negativer for *Type-II fejl*.

		prediction	
		TP	FN
fact	TP	FP	TN
	FN		

Figur 3: Udkommet af en binær opdeling kan illustreres som sande og falske positiver og negativer i form af en *confusion matrix*.

Korrekte positive og korrekte negative forudsigelser dækker altså over, at modellens forudsigelse stemmer overens med den angivne label. Hvis modellen fx skal forudsige om en patient har en dødelig sygdom, som kan behandles, er både sande positiver (modellen siger korrekt at patienten har sygdommen, og vi kan behandle den) eller falske positiver (modellen siger korrekt, at patienten er rask, og vi kan lade være med at bekymre patienten yderligere) selvagt langt at foretrække. Men modellen kan også tage fejl på to forskellige måder: Den kan klassificere et øble som en pære eller omvendt. Disse to tilfælde måske fremstå som symmetriske, men er det sjældent i anvendelser. Hvis modellen fra før for eksempel forkert forudsiger, at patienten er rask kommer vi til at undlade at behandle patienten, som så dør af sygdommen. Omvendt hvis modellen forkert forudsiger, at patienten har sygdommen, så går vi i gang med at behandle patienten, men finder ud af, at der ikke er nogen sygdom at behandle. På den måde fører den første type fejl (FN) til patientens død, hvorimod den anden type fejl (FP) kun fører til en del bekymring og en unødig operation. Selv hvis vi optimerer træningen af modellen til at minimere risikoen for falske negativer, vil beslutninger på grundlag af machine-learning-modeller *altid* foregå på et *ufuldstændigt grundlag*, hvilket vil blive behandlet nedenfor i forbindelse med *modellers forudsigelser* og *ekspertroller*.

Når vi skal udtales om modellens kvalitet, gør vi det ofte ved at anvende en metrik, der mäter, hvor godt modellens forudsigelser på valideringsdata svarer til de kendte labels. Der er flere forskellige metrikker, som man kan anvende, og de indfanger alle forskellige betydninger af, hvad der vægtes som modellens kvalitet (se Figur 4).

$$\begin{aligned}
 \text{Acc} &= \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}} && (\text{Accuracy}) \\
 \text{Pre} &= \frac{\text{TP}}{\text{TP} + \text{FP}} && (\text{Precision}) \\
 \text{Spec} &= \frac{\text{TN}}{\text{TN} + \text{FP}} && (\text{Specificity}) \\
 \text{Rec} &= \frac{\text{TP}}{\text{TP} + \text{FN}} && (\text{Recall/sensitiviy}) \\
 \text{F1} &= 2 \times \frac{\text{Pre} \times \text{Rec}}{\text{Pre} + \text{Rec}} && (1) \\
 &= \frac{\text{TP}}{\text{TP} + \frac{1}{2}(\text{FP} + \text{FN})} && (\text{F1})
 \end{aligned}$$

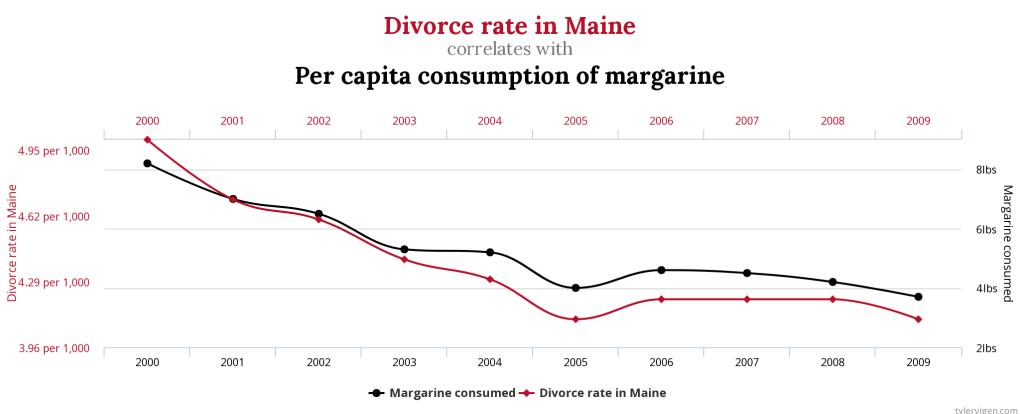
Figur 4: Forskellige metrikker som kan finde anvendelse for at måle kvaliteten af en binær classifier.

## 6.6 Skilsisser og margarine: Korrelationer og årsager

En vigtig viden om ML-modeller er, at de virker ved *korrelationer*. To hændelser siges uformelt at være korrelerede, hvis de *følges ad*, således at når den ene af dem for eksempel vokser, så gør den anden det også. Nogle gange er der faktisk tale om en *kausal sammenhæng*, idet man kan finde ud af, at den ene hændelse er en direkte *årsag* til den anden, som har form af årsagens *virkning*. Effekten er her direkte, hvis den har form af, at årsagen *trigger* virkningen uden variation. Man kan måske tale om, at relationen nærmest er *deterministisk*: virkningen er forudbestemt af årsagen, hvorfor årsagen tidsligt må ligge forud for virkningen.

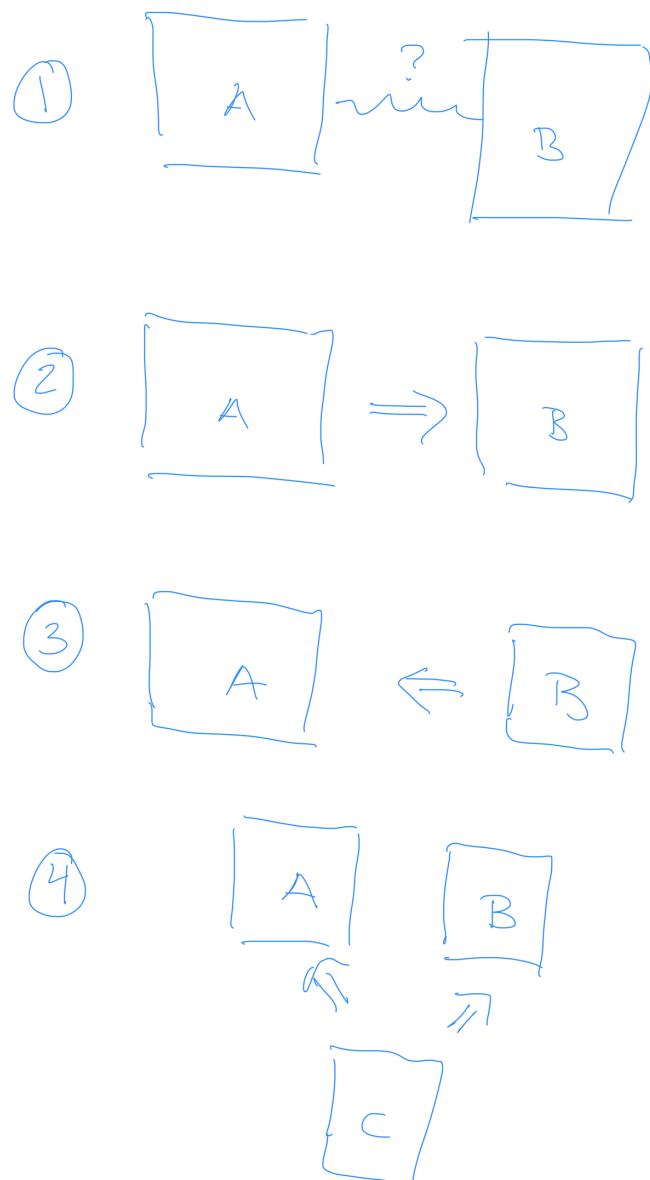
Men når ML-modeller foretager forudsigelser, så er der netop tale om korrelationer og ikke årsag-virkning-sammenhænge: Modellen er trænet på datasæt, hvor en label er tilknyttet hvert datapunkt, så visse variationer i datapunktet vil medføre en forandring i den tilhørende label. Men der er ikke tale om en kausal virkning, idet vi (typisk) ikke kan på en række direkte virkninger, men i stedet ser sammenhængende opførsel.

For at illustrere forskellen mellem korrelation og kausalitet, kan man betragte nogle af de såkaldte *spurious correlations* (se Figur 5).



Figur 5: En korrelation, der næppe er en kausal sammenhæng — en *tilfældig korrelation* (*spurious correlation*), se <https://www.tylervigen.com/spurious-correlations>. Se også Ringgaard (2017) for yderligere forklaring og danske eksempler på tilfældige korrelationer.

Selvom to variable fx følges ad over en periode, behøver der langtfra at bestå en kausal sammenhæng imellem dem. Der kan fx være tale om, at begge variable er relaterede til en tredje variabel (*fælles faktor* eller *confounding factor*) eller at det tilsyneladende parallelle forløb er en ren tilfældighed (se Figur 6).



Figur 6: Figuren viser forskellige måder, hvorpå to størrelser kan være korreleerde: 1. Tilfældig korrelation, 2. A er årsag til B, 3. B er årsag til A, 4. A og B har en fælles årsag. Der er mange flere muligheder for relationerne, men disse fire alene viser, at man ikke fra korrelation mellem A og B kan udlede en kausal relation mellem A og B. Punkterne 2 og 3 alene viser, at eftersom kausalitet har en retning, og korrelationer ikke har, kan vi ikke engang afgøre, *hvor* der evt. er en årsag, og *hvad* der er en virkning.

## Sorte kasser for enhver

Vi bruger betegnelsen *black box* til at beskrive dele af et system, som *for nogle* relevante aktører forbliver lukket, uigennemskueligt og typisk af en størrelse, der overstiger, hvad et individ kan overskue. Denne definition spejler beskrivelser af, hvad der sker, når den individuelle autonome erkendelse bliver umuliggjort af forskellige slags kompleksitet (se fx Sørensen og Andersen, 2018). Man kan altså skelne imellem forskellige gruppens behov for indsigt (dannelse) i *machine-learning-modellers* virkemåde og epistemiske status, og vi kan i hvert fald identificere tre forskellige grupper: forbrugere, anvendere og skabere af *machine-learning-modeller*.

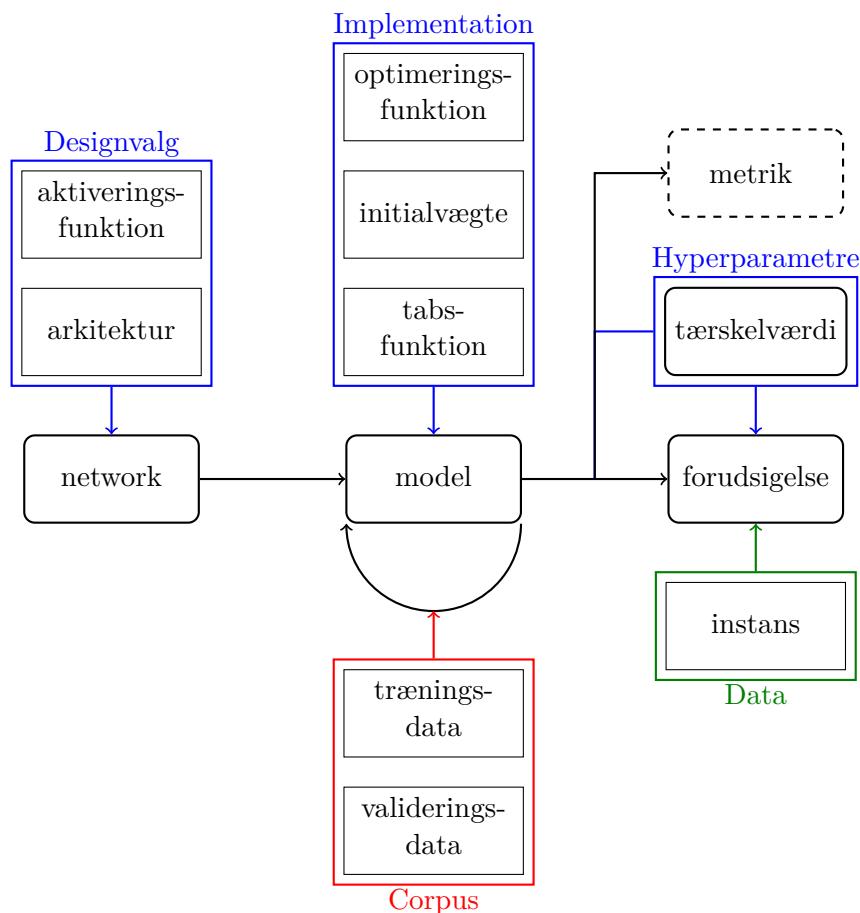
Fra *forbrugerens perspektiv* vil det for fx forslagsalgoritmer fra Netflix eller Facebook være relevant at vide, at anbefalingerne er baseret på, at det enkelte individ placeres i en *referencegruppe* med andre individer i den nærmeste klike baseret på modellens forudsigelser. Man får altså de samme anbefalinger som *andre ligesom en selv* også får, men hvor den nærmere beskrivelse af referencegruppen ikke er til at give, da den typisk er baseret på meget store og højdimensionale datasæt, som typisk forbliver en *black box* for forbrugeren. Resten af modellens opbygning, træning og tekniske funktion forbliver *black-boxet* for forbrugeren.

Anlægger man derimod et perspektiv, som tager udgangspunkt i dem, der skal *anvende machine-learning-modeller* fx i beslutningsprocesser eller til rådgivning, hører der yderligere elementer til en dannet forståelse. Foruden at vide, at forudsigelser er baseret på *referencegrupper*, vil anvenderen også skulle vide, at disse referencegrupper er baseret på fortidig data, og at *machine-learning-modeller* derfor har sit eget *induktionsproblem*: Den kan, ligesom en papegøje, kun gentage det, den allerede er trænet på. For anvenderen har dette særlig betydning, når biases i datasættet, som er baseret på fortidige forhold, propageres ind i fremtiden. Men hvordan datasæt er *kurateret* vil ofte stadig forblive en *black box* for anvenderen, ligesom selve modellens interne funktion også vil være det.

Endelig, og måske mest relevant i denne kontekst, skal vi behandle spørgsmålet om nødvendig indsigt ud fra perspektivet, som tilhører en, der *skaber machine-learning-modeller*. Den overordnede skematiske beskrivelse af en superviseret *machine-learning-model* i Figur 1 dækker over en række elementer, der er af central betydning for udfordringerne ved at forklare *machine-learning-modeller* ud fra et *skaber-perspektiv*. I Figur 7 er denne figur udvidet med yderligere grupper af *designvalg*, *implementationsvalg* og *hyperparametre*, alle indrammet i blåt. Figurens element om *datasæt* er indrammet i grønt, hvilket angiver, at indsigt i dette område fra et *skaber-perspektiv* vil blive taget op igen i forbindelse med diskussioner om etisk og professionelt ansvar.

Når man bygger og udvikler *machine-learning-modeller*, er der først en række *designvalg*, der skal træffes. Det drejer sig blandt andet om valg af det underliggende netværks arkitektur og antallet af neuroner, og det drejer sig om at fastsætte, hvornår neuronerne aktiveres. Disse valg er *pragmatiske* i den forstand, at de ikke behøver at være *absolut optimale*, men blot skal være *gode nok* til, at modellen kan udføre den funktion, den er bygget til, på en tilfredsstillende vis. Men i praksis er de også hyppigt formet af hensyn til *sædvane* og tidlige *erfaring*. På den måde ligner disse valg de valg, som *ingeniører* træffer, når de skal udvikle nye teknologier, idet dog en omfattende inddragelse af *sædvane* og *tavs viden* vil placere konstruktionen af *machine-learning-modeller* som skridtet *før* egentlig *ingeniørvidenskab*.

Noget tilsvarende gør sig delvist gældende for de valg, som træffes under selve implementationen. Men implementationsvalgene er typisk noget, man lettere kan eksperimentere med at justere for bedre performance. På den måde synes disse valg mere at ligne empirisk



Figur 7: Skematisk fremstilling af *machine-learning-model* fra et skaber-perspektiv.

## 6.7 FFF: forklaring, fortolkning og forsvar af modeller

Det står i den internationale *GDPR-forordning*, at enhver der er blevet utsat for en beslutning truffet helt eller delvist på grundlag af en automatiseret procedure (en algoritme) har krav på en *forklaring*. Dette kan synes at være et rimeligt demokratisk krav, men det viser sig i både teori og praksis at være virkelig svært at realisere.

For at komme nærmere på problemerne med at forklare *machine-learning-modeller*, vil vi tage udgangspunkt i begrebet *forståelse*, som i filosofisk forstand er meget nært knyttet til begrebet *forklaring*. Men ved at fokusere på individers eller gruppers *forståelse* bliver det klarere, hvilke roller *sorte kasser* spiller, idet de netop er forhindringer for *forståelsen*.

## Hvad er en forklaring overhovedet?

Som vi allerede er stødt på i kapitel 6a, er et af de mulige formål med modeller, at vi ønsker at bruge dem til at *forklare* egenskaber ved det genstandsfelt, som modellen er bygget over. Men hvad skal vi forstå ved en *forklaring*? Der findes mange forskellige filosofiske tilgange til dette ret centrale spørgsmål (se fx Woodward, 2011), men her er udvalgt to af de klassiske forklaringsmodeller, som er særligt vigtige for diskussionerne af forklaringer af modeller med store datamængder: *forklaringer fra lovmaessigheder* og *forklaringer fra (mekanisk) kausalitet*.

Som et første forsøg på indkredse betydningen af en forklaring kan vi skelne mellem videnskabelige udsagn og *forklaringer* af selv samme videnskabelige udsagn. En simpel første begrebsafgrænsning kunne være, at selve udsagnet hævder, *at* noget er tilfældet, hvorimod forklaringen hævder, *hvorfor* noget er tilfældet. Denne tilgang skelner mellem *hvad-spørgsmål* ('what questions') og *hvorfor-spørgsmål* ('why questions').

Hvor videnskabelige udsagn henter belæg i form af evidens, som for eksempel kan være empirisk, må forklaringer hente deres belæg et andet stedet fra. Men præcist *hvor* vi skal finde dette belæg afhænger i høj grad af hvilke andre filosofiske antagelser, vi har gjort os.

Hvis man fx går ud fra et synspunkt som *logisk positivisme*, så vil førende logiske positivist typisk hævde, at forklaring opstår som svar på et *hvorfor-spørgsmål*, og det som kan forklares er fænomener, der falder under en af de generelle lovmaessigheder, som er udledt ved induktion ud fra de fordomsfrie observationer (se Figur ?? i kapitel 1). En forklaring på, hvorfor Saturns bane er ellipseformet ville således være, at ud fra Keplers love er *alle* planeters baner elliptiske, og Saturn er en planet. På den måde bliver forklaringer ud fra dette synspunkt til deduktioner.

Denne ide om hvad en forklaring består i, blev fremført klarest af CARL GUSTAV HEMPEL (1905–1997) i 1940’erne, og den har siden udgjort den position, som de fleste andre forslag på at beskrive, hvad det vil sige at forklare en videnskabelig sammenhæng, er gået ud fra at forfine. Man kalder ofte denne form for forklaringsbegreber for *deduktiv-nomologiske* (DN) forklaringsteorier, idet forklaringen består af en udledning (*deduktion*) ud fra lovmaessigheder (som filosoffer betegner *nomologiske*). Vi kan sammenfatte positionen som at forklaringer har form af svar på hvorfor-spørgsmål af formen, at påstanden følger deduktivt fra lovmaessigheder.

Siden HEMPELS forklaringsmodel er der formuleret en del andre, og særligt forklaringsmodellen baseret på såkaldt *mekanistisk kausalitet* udgør et interessant alternativ (se evt. Ross og Woodward, 2023). Den indfanger en anden intuition omkring forklaringer, denne gang baseret på intuitioner om fysiske hændelser, nemlig at det, der skal forklares, er en kausal virkning af en årsag. Denne tilgang indfører dermed et tidsligt element i forklaringen, idet årsagen må forekomme før virkningen og være en direkte (mekanisk) *trigger* af denne, ligesom en billardkugle, der rammer en anden kugle, er årsag til, at den anden kugle bevæger sig.

En tredje forklaringsmodel forsøger at indfange *forklaring* ved hjælp af intuitioner om *statistisk relevans* — denne slags forklaringer kaldes derfor *SR-forklaringer*. Som det formuleres i Woodward (2011) er en egenskab *C* for en population *A* statistisk relevant for en

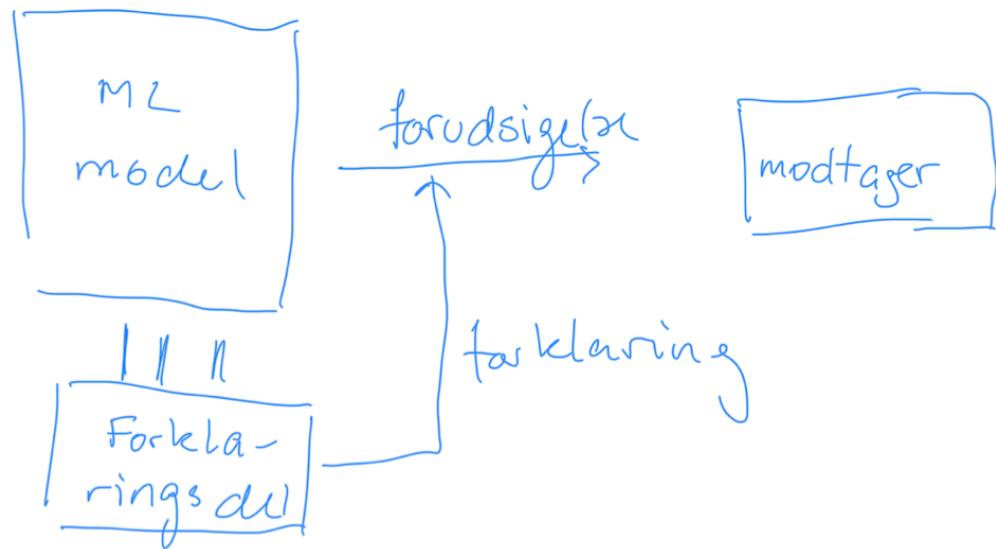
anden egenskab  $B$ , hvis  $P(B|A.C) \neq P(B|A)$ , dvs. hvis den betingede sandsynlighed for  $B$  givet  $A$  forandres af antagelsen af, at  $C$  også indtræffer. *SR-forklaringsmodellen*, fremført af WESLEY SALMON (1925–2001), siger da, at statistisk relevante forhold (som  $C$ ) er *forklarende* for de forhold, som de er statistisk relevante for (i vores tilfælde  $B$ ).

## Forklaring af modeller

Hvis vi søger at anlægge HEMPELS model om forklaringer ud fra deduktioner fra lovmaessigheder til at forklare *black boxes* i anvendelsen af machine learning, så løber vi ind i en variant et klassisk **problem** ved den **logiske positivisme** (se også Kapitel 1): Hvordan kan vi opstille og tilgå de lovmaessigheder, hvis de er opstillet af en stor og kompliceret *machine-learning-model* ud fra korrelationer i vores eksisterende datasæt. Vi kan sagtens bruge modellen til at lave forudsigelser (svarende til deduktioner hos HEMPEL), men at formulere en *forklaring* af en forudsigelse som „det følger af modellen“ uden at være i stand til at åbne den sorte kasse, som omgiver modellen, er ikke nogen god forklaring, selv ud fra det positivistiske synspunkt, da det næppe kvalificerer som et svar på et hvorfor-spørgsmål, men i stedet henviser til modellen som *blind autoritet*.

Men hvorfor ikke lade en computer forsyne forklaringer af *machine-learning-modeller*, der er uigennemskuelige og uigennemtrængelige *for mennesker*? Man har forsøgt sig med forskellige moduler, som enten kan bygges ind i selve den oprindelige *machine-learning-model* eller kan kobles efter den (se Figur 8). Men alle disse forsøg er (hidtil) stødt ind i de samme principielle problemer, som er forbundet med at give en *forklaring*: De er typisk baseret på *visualiseringer*, som giver gode måder at *fortolke* modellens forudsigelser (se næste afsnit), men ikke er formulerede i hverken lovmaessigheder eller årsagssammenhænge. Der findes dog også lovende forsøg på at bygge *selvforskarende modeller*, fx såkaldte *bayesiske machine-learning-modeller*, der er en slags generalisering af *beslutningstræer*, som de blev brugt i de første ekspertsystemer (se også Kapitel 7). Forskellen fra de overskuelige, klassiske *beslutningstræer* er imidlertid at også i denne form for *machine-learning-modeller* bliver selve *mængden* af skridt i en *forklaring* uoverskueligt stort.

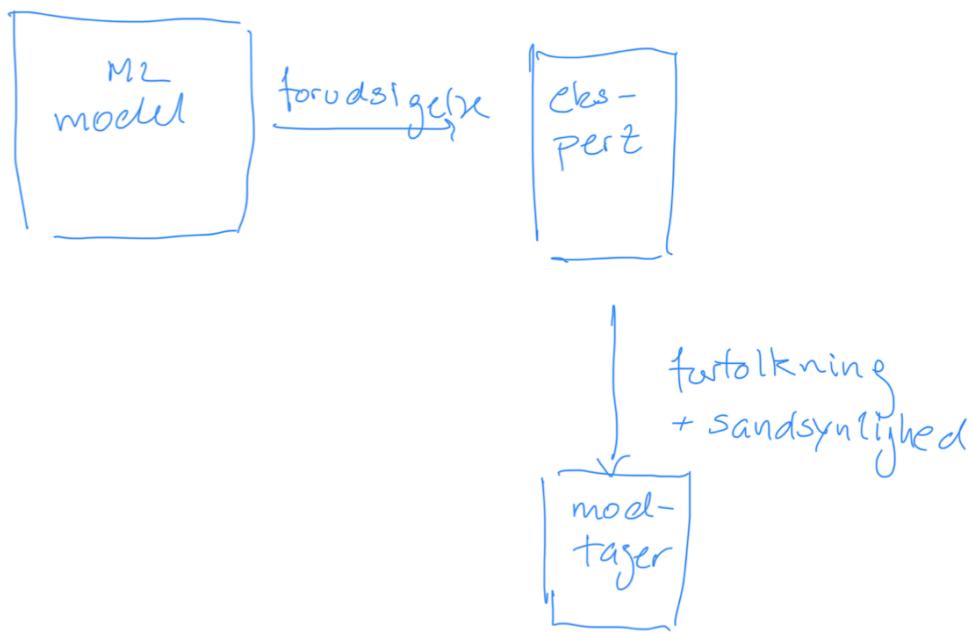
Når en *machine-learning-model* bliver uoverskuelig, fx fordi antallet af vægte bliver enormt stort, kan man forsøge at bygge andre typer forklaringsmodeller ind i tekniske løsninger (se Figur 8). Hidtil har man dog ikke opnået forklaringer af *machine-learning-modeller*, der levet op til klassiske intuitioner om forklaringsmodeller, hverken i form af lovmaessigheder, statistisk relevans eller mekanistisk kausalitet.



Figur 8: Skitse af en *machine-learning-model*, hvortil er tilføjet en *forklaringsenhed*, enten internt i modellen eller som en teknisk tilføjelse.

### Fortolkning af modeller

I stedet for at søge forklaringer *inden i* videnskabelige teorier eller metoder, kan man også anlægge et *kommunikativt* perspektiv på forklaringer: *Nogen* forklarer *noget* til *nogen*. Med denne tilgang vil *forklaringen* næppe leve op til de strenge forhåbninger diskuteret ovenfor, så for at adskille denne proces fra HEMPEL s forklaringsmodel og kausale forklaringer, betegner vi den her som *formidling af modeller* og deres forudsigelser.



Figur 9: En fortolkning af en models forudsigelser kan anskues som en kommunikationsproces mellem modellen, en ekspert-fortolker og en modtager.

Dette kommunikationsbaserede syn på *fortolkning* af *machine-learning-modeller* involverer tre *aktør*rakter: selve modellen, som producerer forudsigelser, en ekspert, som har tilstrækkelig viden om og forståelse af modellen til at kunne omsætte forudsigelserne til fortolkninger, og en modtager. I forhold til det syn på *forklaringer* af *machine-learning-modeller*, som blev præsenteret ovenfor, er hele processen altså tænkt som en kommunikation og der indskudt en (menneskelig) *ekspert*. At processen er en kommunikation betyder altså blandt andet, at information og spørgsmål kan flyde begge veje.

I praksis vil en af ekspertens vigtigste (og sværreste) opgaver hyppigt være at udlægge sandsynligheder på en tilgængelig vis for en almindelig modtager. Den slags udfordringer er ikke nye, og der er fx i lægeverdenen udviklet forskellige redskaber til at inddrage patienten i vigtige beslutninger, der er baseret på sandsynligheder. En af de mest brugte metoder er at inddrage *visualisering* i form af kurver eller kort, der fx angiver 'høj', 'middel' eller 'lav' risiko for at oversætte procentsatser og tekniske formuleringer til et mindre formelt medium.

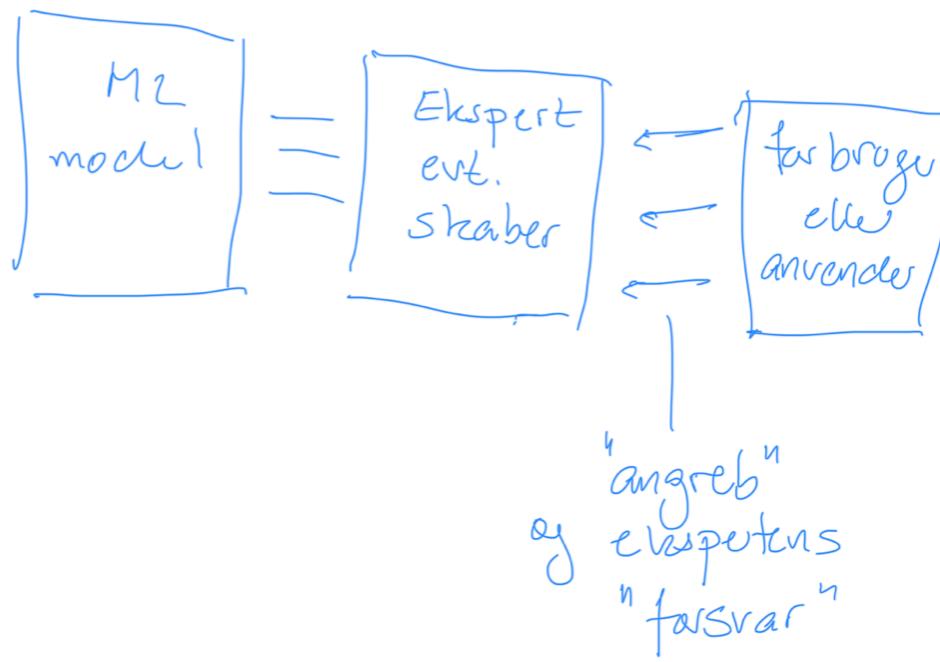
Tidligere var disse former for scenarier typisk bygget op omkring *proxies*, der er lettere at måle og for hvilke, der ofte er en traditionel forståelse også hos modtageren. En vigtig del af fortolkning af sandsynligheder og inddragelse i valgprocesser for modtageren er muligheden for at påvirke, hvorvidt modellens forudsigelser også kommer til at holde stik. Derfor kan nye, nærmest interaktive *scenarie-beregninger* næsten virke som en form for personlig coach. I stedet for at få at vide, hvordan forudsigelserne ville se ud for andre i nogle af mine nærmeste referenceklasser (typisk udvalgt efter traditionelle *proxies*), kan jeg nu interaktivt eksperimentere med forskellige parametre i modellen. Men for at disse eksperimenter giver faglig mening og for at udlægge modellens forudsigelser og begrænsninger, er ekspertens rolle vigtig. Eksperten kommer dermed til at kombinere *domæneviden* (fx inden for lægevidenskab) med tilstrækkelig indsigt i modellen til at bygge en form for bro mellem modtageren og den for vedkommende uigenmæsigtige model.

## Forsvar af modeller

Som et tredje og sidste forsøg på at antyde, hvordan vi kan komme til bedre forståelse af *machine-learning-modeller*, kan man betragte en anden form for kommunikation mellem en model, en ekspert og en eller flere modtagere. Hvor dialogen i fortolkningen af modeller fandt sted mellem modellen og modtageren med eksperten som mediator, er ideen i dette tilfælde, at eksperten skal kunne *forsvare* modellen imod angreb fra modtagerne (se Figur 10). Modtagernes angreb kan være kritiske spørgsmål eller påpegning af uhensigtsmæssige forhold omkring modellens forudsigelser. Hvis eksperten på troværdig vis kan *forsvare* modellen, kan vi være tilbøjelige til at tilskrive eksperten *forståelse* af modellen.

Men hvori består så sådanne forsvar? Det er ikke tilstrækkeligt, at eksperten kan give en teknisk forklaring ved at opremse faktiske forhold omkring modellens konstruktion. Der må også skulle finde et vist stykke tilpasning til modtagernes angreb sted, men vigtigere er det nok, at ekspertens forsvar har en tilstrækkelig *holistisk* tilgang til at kunne forklare modellens *opførsel* og *funktion*. Herunder vil det også være relevant at kunne forsvara modellens kvalitet og kvalificere dette på individniveau. Det vil være oplagt, at eksperten også skal kunne forsvere de valg, som er truffet i modellens konstruktion (se Figur 7).

Forsvaret af en model ligner altså på flere måder det, som vi ovenfor har kaldt fortolkning af modellens forudsigelser. Men alligevel adskiller de to tilgange sig: Det at kunne forsvare en model kræver en del mere indsigt fra eksperten — tilgengæld er hun 'kun' forpligtet på at kunne forsvare modellen mod de angreb, der faktisk forekommer.



Figur 10: Som forsvar af en model skal eksperten kunne svare på kritiske spørgsmål og indvendinger mod modellens opbygning og funktion.

## Litteratur

- Anderson, Chris (23. jun. 2008). „The End of Theory. The Data Deluge Makes the Scientific Method Obsolete“. *Wired*, bd. 16, nr. 7, s. 108–109.
- Dahl-Jensen, Dorthe (okt. 2009). „Istiden sluttede ekstremt hurtigt“. *Kvant*, s. 3–8.
- Johansen, Mikkel Willum og Henrik Kragh Sørensen (2018). „Modelvalg og ansvar“. *Aktuel Naturvidenskab*, nr. 1, s. 36–39.
- Ringgaard, Anne (31. maj 2017). „Korrelation eller kausalitet. Hvornår er der en årsags-sammenhæng?“ *Videnskab.dk*.
- Ross, Lauren og James Woodward (2023). „Causal Approaches to Scientific Explanation“. I: *The Stanford Encyclopedia of Philosophy*. Red. af Edward N. Zalta og Uri Nodelman. Spring 2023. Metaphysics Research Lab, Stanford University.
- Sørensen, Henrik Kragh og Line Edslev Andersen (mar. 2018). *Mapping Social Aspects of Mathematical Knowledge Production*. RePoSS: Research Publications on Science Studies 43. Aarhus: Centre for Science Studies, University of Aarhus.
- Wolfram, Stephen (2002). *A New Kind of Science*. Champagne (IL): Wolfram Media.
- Woodward, James (2011). „Scientific Explanation“. I: *The Stanford Encyclopedia of Philosophy*. Red. af Edward N. Zalta. Winter 2011.

# Kapitel 10:

## Etik, redelighed og privacy

Henrik Kragh Sørensen

Mikkel Willum Johansen

22. april 2022

### Indhold

5

10.1 Deskriktiv etik, normativ etik og jura . . . . .	1
10.2 Første normative teori: Utilitarisme . . . . .	4
10.3 Anden normative teori: Kants pligtetik . . . . .	6
10.4 Nagels syntese . . . . .	8
10.5 Analyser i praksis . . . . .	10
10.6 Big data-etik og privacy . . . . .	12
10.7 Bias og algoritmisk transparens . . . . .	14
10.8 Systemisk etik . . . . .	17
10.9 IT-professionelles etiske ansvar . . . . .	18
10.10 Redelighed . . . . .	19
	15

### Litteratur

23

### 10.1 Deskriktiv etik, normativ etik og jura

På platformen Moral Machines (<http://moralmachine.mit.edu/>) har en gruppe forskere fra MIT i en årrække indsamlet information om, hvordan folk mener, at en selvkørende bil skal reagere i en række såkaldt no-win-scenarier, hvor bilen vil blive tvunget til at dræbe enten én gruppe personer eller en anden gruppe personer. Forskerne har løbende indsamlet og analyseret brugernes svar, hvilket der er kommet en række interessante artikler ud af. Fx har det vist sig, at der er store regionale forskelle i de valg, folk tager (Maxmen, 2018). Hvor man i Sydamerika har en præference for at redde kvinder, børn og folk af høj status, er man i den sydlige del af Asien mere optaget af at redde fodgængere og lovlydige, og foretrække desuden at undgå at handle. Undersøgelsen er et eksempel på det, man kan kalde deskriktiv etik, dvs. en undersøgelse af de etiske valg folk faktisk foretager (eller siger de vil foretage) i praksis. Som undersøgelsen antyder, kan der være kulturelle forskelle mellem de etiske præferencer, folk i praksis har.

20

25

30

Det, at folk handler på en bestemt måde, betyder naturligvis ikke, at det er den rigtige måde at handle på. Det ville være en klar fejlslutning (der i etikken kaldes *den naturalistiske fejlslutning*). Undersøgelsen af, hvordan folk bør handle for at opføre sig etiske korrekt kaldes normativ etik, og består til dels af teorier, der giver generelle bud på, hvordan man bør opføre sig i forskellige situationer. Som eksemplet med Moral Machines viser, er der ikke

35 enighed om, hvad man skal gøre i en given situation, og den uenighed søger de normative teorier at råde bod på.

Nu er der heller ikke enighed om, hvilken normative etik, der er den rigtige, og vi vil her se på to forskellige teorier, der repræsenterer meget forskellige etiske intuitioner.

40 Hvis man bliver overbevist af en af de normative teorier, vil man have (mere eller mindre) klare handlingsanvisninger i alle de etiske valg, man fremover vil stå i. Som vi skal se, rummer de normative teorier dog selv en række problemer, men selv om man ikke accepterer teorierne i deres helhed er det stadig værdifuldt at overveje rækkevidden af de overordnede principper, teorierne repræsenterer. Fra det mere pragmatiske synspunkt på 45 etikken, som vi vil tillade os at anlægge i denne bog kommer den mest grundlæggende etiske fordring hverken fra den ene eller den anden etiske teori, men består i at man i det hele taget tager den etiske udfordring op og forsøger at nå frem til en form for reflekteret ligevægt (jf Rawls, 1972), hvor man er konsistent i den forstand, at der er sammenhæng mellem ens overordnede principper og de konkrete domme, man foretager, og at de argumenter man accepterer i en kontekst også bliver accepteret i andre. Vi vil senere i kapitlet give en 50 struktur for, hvordan den slags analyser kan foregå.

Udover deskriptiv og normativ etik er det i denne sammenhæng også værd at inddrage det juridiske område, dvs. spørgsmålet om, hvad der er lovligt og hvad der ikke er lovligt. Ideelt set burde der være et perfekt sammenfald mellem det etiske og det juridiske, sådan at lovlige handlinger altid er etisk korrekte og omvendt. I praksis er det dog ikke så simpelt, 55 og der opstår jævnligt situationer, hvor folk bliver nødt til at bryde love for at gøre, hvad de anser for etisk rigtigt. Et godt eksempel er *whistleblowere* som EDWARD SNOWDEN, (visse) 'white-hat' hackere, der bryder loven for at gøre opmærksom på sårbarheder i kritiske IT-systemer, eller spørgsmålet om ophavsret og deling af vigtig information. Det sidste blev fx sat på spidsen af aktivisten AARON SWARTZ (1986–2013), der i det såkaldte *Guerilla Open Access Manifesto* fra 2008 skrev:

60 It's called stealing or piracy, as if sharing a wealth of knowledge were the moral equivalent of plundering a ship and murdering its crew. But sharing isn't immoral—it's a moral imperative. Only those blinded by greed would refuse to let a friend make a copy. [...] There is no justice in following unjust laws. It's time to come into the light and, in the grand tradition of civil disobedience, declare our opposition to this private theft of public culture. (Swartz, 2020)

SWARTZ begik i 2013 selvmord mens han var under anklage for ulovligt at have downloaded et meget stort antal videnskabelige artikler til sin computer.

70 Det er et åbent spørgsmål, om SWARTZ havde ret i, at man har en moralsk forpligtelse til at dele information, men der er her en klar brudflade mellem etik og jura, og diskussionen har været med til at flytte både juraen, vores praksis og hvad vi anser for etisk korrekt fx gennem opfindelsen af ny rettighedsformer som *creative commons* (som SWARTZ også var med til at udforme).

75 Brudfladen mellem etik og jura kan også udspille sig på den måde, at ting, vi anser for etiske forkerte, faktisk er lovlige. Specielt på områder, hvor der er en hastig teknologiudvikling, vil der ofte være en form for juridisk vakuum, hvor teknologien muliggør nye praksiser, der forekommer åbenlyst uetiske, men som alligevel er lovlige i en årrække af den simple grund, at lovgiverne først skal opdage problemet og bagefter blive enige om en passende lov-givning for at ulovliggøre den uetiske praksis, og det kan tage meget lang tid. Kritikere som 80 SOSHANA ZUBOFF hævder, at det er præcis dette vakuum firmaer som Google og Facebook

bevidst har udnyttet til i en årrække at profitere af en forretningsmodel, der grundlæggende er uetisk (Zuboff, 2019). Det kan naturligvis diskuteres, om ZUBOFF og ligesindede har ret i deres kritik (vi vil vende tilbage til spørgsmålet senere), men der er til gengæld meget lidt tvivl om, at det juridiske vakuum hastig teknologiudvikling skaber, er en realitet.

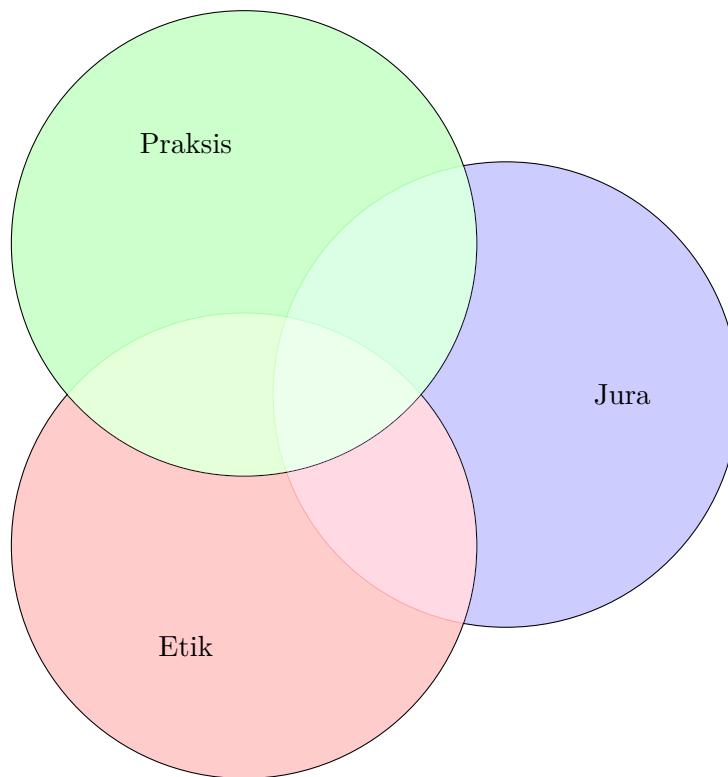
Det tre områder, der udgøres af vores faktiske *praksis*, den *normative etik* og *juraen* er altså tæt forbundne, men ikke altid sammenfaldende. Af den grund er undskyldninger som „alle gør det, så er det etisk korrekt“, eller „det er ikke ulovligt, altså er det etisk korrekt“ åbenlyst fejlslutninger: Undskyldningerne ville kun virke, hvis der var perfekt overensstemmelse mellem hvad vi faktisk gør, hvad vi burde gøre og hvad der er lovligt; men som vi har set er dette ikke tilfældet, specielt i områder med hastig teknologiudvikling. De tre områder er dog tæt sammenflettede, og udviklingen på et område kan meget let påvirke udviklingen på de to andre. I 1987 fik vi således i Danmark *Det Etiske Råd*, der skulle rådgive politikere om udviklingen inden for gen- og bioteknologi. Siden da har både praksis, lovgivningen og den normative vurdering af fx en teknologi som kunstig befrugtning udviklet sig dramatisk, men hele tiden som et tæt sammenspil mellem de tre områder, hvor fx ændrede etiske vurderinger har ført til ny lovgivning, der banede vejen til nye praksiser, der har ændret den normative vurdering osv. (se fx Adrian, 2016). Nu har vi fået et *Dataetisk Råd*, der bl.a. skal rådgive i spørgsmål om privatliv og brugen af kunstig intelligens. Der er derfor al mulig grund til, at man som datalog er både parat og klædt på til at gå ind i de etiske og juridiske debatter, der rejses af den teknologiudvikling, man er en del af.

85

90

95

100



Figur 1: De tre domæner.

Hvis man har en mere instrumentel tilgang til etikken, kan man sige følgende med reference til figur 1: Det er bedst at befinde sig, hvor der er overensstemmelse mellem etik, jura og praksis, men hvis man kommer uden for det område, gør man klogt i at være på vagt.

105 Hvis man befinder sig et sted, hvor den lokale praksis er lovlig, men uetisk risikerer man etisk fordommelse (og samvittighedskvaler). Befinder man sig et sted, hvor praksis både er ulovlig og uetisk, gør man klogt meget hurtigt at ændre praksis, og er ens praksis etisk men ulovlig er det man gør nok etiske set prisværdigt, men det kan have store personlige omkostninger (som SNOWDEN og andre whistleblowere illustrerer).

110 **10.2 Første normative teori: Utilitarisme**

Den første normative etiske teori, vi skal se på, kaldes *nytteetik* eller *utilitarisme*. Utilitarismen er et eksempel på det, man lidt bredere kalder en *konsekvensetik*, dvs. en etik, hvor den etiske vurdering af en handling udelukkende inddrager konsekvenser af handlingen, og ikke det princip, handlingen repræsenterer. I sin reneste form siger utilitarismen ganske simpelt, at den etiske set rigtige handling i enhver given situation er den handling, der maksimerer den samlede nytte (eller lykke) for samtlige involverede parter.

115 Utilitarismen har rødder tilbage til antikken, men i den moderne udgave blev den primært formuleret af den engelske filosof JEREMY BENTHAM (1748–1832) samt JAMES MILL (1773–1836) og hans søn JOHN STUART MILL (1806–1873) i den første halvdel af 1800-tallet. Utilitarismen var (navnlig for J. S. MILL) et led i en bevægelse for social reform; utilitarismen tager ikke hensyn til rang eller stand, men stiller alles nytte lige. J. S. MILL var således en tidlig og stærk fortaler for for kvinders stemmeret. Utilitarismen kan desuden ses som et forsøg på at videnskabeliggøre etikken. I stedet for at tage udgangspunkt i religiøse dogmer eller mere eller mindre obskure fornuftsprincipper tager utilitarismen udgangspunkt i en enkelt kvantitet — mængden af nytte — og søger at maksimere denne.

120 Når nytteetikken søger at maksimere netop nytte skyldes det, at nytte for dem er et såkaldt *intrinsisk gode*, dvs. noget, der er godt i sig selv. Andre goder som penge, mad eller et sted at bo er instrumentelle goder, dvs. goder, vi gerne vil have, for at opnå noget andet. Man vil have en uddannelse, så man kan få et job, man vil have et job, så man kan tjene penge, man vil have penge, så man kan købe ting, og man vil have de ting, man køber, fordi de gør en lykkelig. Så for alle andre goder kan man spørge: Hvorfor vil du have det gode? Men det giver ikke nogen mening at spørge: Hvorfor vil du være lykkelig? Lykke (eller nytte) er et gode i sig selv.

125 I udgangspunktet virker utilitarismen som en simpel og næste selvindlysende teori, men når man kigger nærmere efter, er den forbavsende nok ingen af delene. Hvis vi ser på kravet om at maksimere nytten for alle involverede parter må man for det første spørger, hvad det egentlig er, vi skal maksimere. Vi har i teksten her skiftet mellem termerne 'lykke' og 'nytte', og det er ikke en oversættelsesfejl; BENTHAM og J. S. MILL skifter selv mellem at bruge termerne 'happiness' og 'utility', så allerede her er der en vis uklarhed, og diverse udgaver af utilitarismen har sidenhen på forskellig vis forsøgt at præcisere begrebet, uden at der er opnået enighed.

130 Et andet godt spørgsmål er, hvem vi skal maksimere lykken for. Den oprindelige utilitarisme blev brugt til at argumenter for social reform, da alles lykke tæller lige, men den blev også brugt til at argumentere for dyrevelfærd. I et berømt citat siger BENTHAM: „The question is not, Can they reason? nor, Can they talk? but, Can they suffer?“ (Bentham, 2000, kap. XVII §4). Med det mener han, at nyttemaksimeringen ikke skal begrænse sig til intelligente væsener (mennesker), men til alle dyr, der er i stan til at føle smerte og lykke. Det gør selvfølgelig ikke regnestykket simplere — hvor meget lykke kan en bændelorm føle, og har vi en moralsk forpligtelse til at maksimere den? Og hvordan skal vi forholde os

til robotter med syntetiske følelser? Tilsvarende kan man spørge hvilken vægt fremtidige generationer har i ligningen. I klimadebatten vægter hensynet til fremtidige generationers lykke højt, men hvor meget skal de vægte og hvor langt skal man se frem? 150

Alt i alt viser det sig, at den simple idé om at maksimere lykken for alle involverede parter er så vag, at den ikke udgør en egentlig normativ teori, men snarere en skitse til en teori. Den bliver først en operationaliserbar teori, når man har præciseret, hvad man mener med de forskellige begreber, og den præcisering kan foregå på flere forskellige måder. Så når man fremfører utilitaristiske argumenter eller indgår i en debat med en utilitarist, skal man være opmærksom på, hvordan teorien (eller rettere: teoriskitsen) konkret er blevet implementeret, og det er klart, at man ikke i en sammenhæng kan definere 'alle involverede parter' på en måde, hvis man i en anden sammenhænge har defineret det anderledes. 160

Utilitarismen har også andre og mere grundlæggende problemer. For det første er det ikke altid klart, hvilke konsekvenser en konkret handling har. Specielt i teknologiudvikling er dette et virkligt og konkret problem; TIM BERNERS-LEE, som var hovedmanden bag World Wide Web, kunne således næppe have forudset de enorme konsekvenser internettet har haft for vores demokratiske debat og vores måde at knytte sociale relationer på. Utilitarismen kan (i et vist omfang) håndtere den indvending ved udelukkende at inddrage sandsynligheden for forudsebare konsekvenser. Så hvis man arbejder på en forbedret algoritme til ansigtsgenkendelse, må man med andre ord overveje, hvor sandsynligt det er, at den bliver brugt til forskellige ting (at låse smartphones op, at undertrykke opositionen i diktaturstater mv.) og derefter lave kalkulen. Det er et godt spørgsmål, hvor grundig af-dækningen af de mulige konsekvenser skal være. Er det nok, at jeg ser på de umiddelbart intenderede anvendelser af min teknologi, eller har jeg, som teknologifilosoffen HANS JONAS (1903–1993) hævdede, et mere grundlæggende ansvar for at være futerolog, dvs. for systematisk og grundigt at gennemtænke de mulige konsekvenser af den teknologi, jeg udvikler. I en datalogisammenhæng har navnlig udviklingen af autonome våbensystemer og kunstig intelligens fået forskere til at tage et aktivt ansvar for de langsigtede, mulige konsekvenser af deres forskning. 165 170 175

Det mest alvorlige problem for utilitarismen er imidlertid, at det entydige fokus på den samlede lykke betyder, at teorien ikke tager højde for, at vi som enkeltindivider kan have rettigheder, der beskytter os mod at blive udnyttet af flertallets behov. Hvis det fx var nyttemaksimerende at inficere guatemalanske børnehjemsbørn med könssygdomme for at teste en ny medicin — som JOHN CHARLES CUTLER (1915–2003) rent faktisk gjorde det med støtte fra de amerikanske sundhedsmyndigheder i 1946–48 og uden børnenes samtykke — ville det fra et utilitaristisk synspunkt være en etisk set korrekt handling (Reverby, 2011). Tilsvarende kan utilitarismen blive brugt til at retfærdiggøre handlinger, der overskrider vores ret til privatliv, vores ret til autonomi og hensynet til retfærdighed. 180 185

Utilitarismen har to svar på den udfordring. Det ene er at sige „so what?“. Hvis vores intuition fortæller os, at det er forkert at bruge børnehjemsbørn til medicinske forsøg eller at give køb på vores privatliv for at fremme det fælles bedste, så er det ikke et problem for utilitarismen, men et problem for vores etiske intuition. Det viser os blot, at vores intuition ind i mellem er forkert, præcis som vores forestillinger på så mange andre områder var fejlagtige, før de blev guides af rationelle, systematiske undersøgelser. 190

Det andet mulige svar er at introducere den såkaldte **regelutilitarisme**. I den oprindelige utilitarisme tager man udgangspunkt i og søger at lykkemaksimere konkrete enkeltehandlinger. Den version af utilitarismen kaldes **handlingsutilitarisme**: Hvis det i denne konkrete situation vil maksimere lykken i verden, at jeg bruger børn til medicinske forsøg uden deres tilladelse, så er det utilitaristiske set det rigtige at gøre. **Regelutilitarismen** ser derimod på 195

generelle handlingekategorier, og søger at opstille regler, der generelt maksimerer lykken, hvis de bliver fulgt. Her kan man opstille det argument, at hvis man generelt kan bruge børn til medicinske forsøg uden deres tilladelse, vil det formentlig føre til et fald i lykke, da folk forståeligt nok vil være nervøse for, at deres børn vil blive brugt som forsøgskaniner. Regelutilitarismen bringer dermed til en vis grad den utilitaristiske tankegang bedre i overensstemmelse med vores moralske intuition, men på den løser næppe alle problemerne. Specielt er det ikke klart, hvad det er for en generel handlingskategori, jeg følger. Det er måske rigtigt, at en regel om at bruge børn generelt til medicinske forsøg næppe vil være lykkemaksimerende, men det kan jo godt være, at en regel, der tillader brugen af en meget specifik gruppe børn — fx guatemalanske børnehjemsbørn — til medicinske forsøg faktisk vil være lykkemaksimerende. Og dermed er vi lige vidt. Det er dermed klart, at der skal en anden form for etisk teori i spil, hvis vi ønsker at redegøre for den intuitive fornemmelse af, at individet på en eller anden måde skal beskyttes mod kravene fra det fælles bedste. Og netop et sådant synspunkt finder vi hos IMMANUEL KANT.

### 10.3 Anden normative teori: Kants pligtetik

Den anden normative etiske teori vi skal betragte her er en *pligtetik*, der blev udformet af den tyske filosof IMMANUEL KANT (1724–1804). I pligtetikken (eller *deontologien*, som pligtetikken også kaldes, efter det græske ord for pligt ’deon’) bliver en handlings moralske værdi bedømt på handlinge i sig selv og ikke på dens konsekvenser. Typisk vil man se på, om handlinge er i overensstemmelse med et sæt givne regler. Pligtetikken er dermed en direkte modsætning til konsekvensetikken.

Præcis som der findes mange forskellige former for konsekvensetik (hvor utilitarismen kun er én udgave) findes der også mange forskellige former for pligtetik, og KANTS pligtetik er blot en af disse. Det er dog ikke helt tilfældigt at vi netop inddrager KANTS pligtetik, da den på mange måder har været med til at forme det pligtetiske modsvar til utilitarismens konsekvensetiske betoning af det fælles bedste.

KANT udviklede sin etik i slutningen af 1700-tallet primært i de to værker *Grundlegung zur Metaphysik der Sitten* (1785) og *Kritik der praktischen Vernunft* (1788). Vi vil ikke gå i dybden med KANTS argumentation, men i korte træk påpeger KANT, at det er uholdbart at bruge en handlings konsekvenser til at vurdere dens moralske kvalitet, da konsekvenserne altid er usikre. Derfor er vi nødt til at se på handlingsprincippet, når vi afgøre om en handling er etisk korrekt eller ej. KANT hævder desuden, at menneskets status som moralske subjekter primært skyldes, at vi er fornuftsvæsner, der er i stand til at træffe autonome, rationelle beslutninger. Det centrale spørgsmål for etikken er, hvordan vi forvalter denne frihed, dvs: hvad er den gode vilje? KANT afviser eksplisit, at mennesker skal stræbe efter lykke eller velfærd; hvis vi handler efter vanе, lyst eller begær, gør vi os til slaver af vores dyriske natur, og opträder dermed ikke som frie fornuftsvæsner. Hvis vi vil hævde vores frihed må vi lade fornuften styre og sætte sig sine egne mål uden hensyn til hverken konsekvenserne eller vores lyster og følelser. Og det, fornuften først og fremmest kræver, er konsistens. Vi skal derfor sørge for, at de handlingsprincipper vi følger, ikke kan føre til en interne modstrid.

På den (lidt snørklede) måde når KANT frem til et grundlæggende moralsk princip, kaldet det *kategoriske imperativ*. Som navnet antyder, skal principippet forstås som et absolut påbud, man til enhver tid og uanset konsekvenserne skal følge, hvis man vil opføre sig etiske korrekt. I en af flere formuleringer, lyder principippet: „Handl kun ifølge den maksime ved

hvilken du samtidig kan ville, at den bliver en almennyldig lov“ (Kant, 1999, s. 78). Det kategoriske imperativ skal forstås som en form for laksprøve, man kan bruge til at teste sine handlinger med. Hvis man vil udføre en handling, skal man først finde ud af hvilket handlingsprincip (eller ’maksime’) man handler efter, og derefter skal man overveje, om handlingsprincippet overholder det kategoriske princip. Det kategoriske imperativ fungerer med andre ord på den måde, at det udstikker en række forbud mod bestemte handlingstyper.

245

Som et eksempel, ser KANT på følgende handlinge: Du låner penge og lover at betale dem tilbage, selv om du ikke har tænkt sig at gøre det. Maksimen for din handling er her: At afgive et løfte, man ikke har tænkt sig at holde. Den maksime kan ikke klare at blive testet med det kategoriske imperativ, for hvis alle afgav løfter, selv om de ikke havde tænkt sig at holde dem, ville selve løfteinstitutionen bryde sammen, fordi ingen længere ville tro på folk, når de afgav løfter – løfter ville blive meningsløse. Af den grund kan man ikke ville, at maksimen bliver til almen lov. Bemærk, at det ikke er fordi, man ikke bryder sig om en verden, hvor der ikke findes løfter (hvilket vil være en konsekvensetisk tænkning), at handlingen er uetisk ifølge KANT. At afgive et falsk løfte er uetisk fordi det er selvmodsigende og dermed fornuftsstridigt at udføre en handling, der ville ophæve sig selv, hvis det blev almen lov at udføre den. Tilsvarende vil handlinger som at stjæle, at lyve og at slå folk ihjel være uetiske ifølge det kategoriske imperativ idet ingen af dem kan ophæves til almen lov uden at gøre sig selv meningsløse eller umulige.

250

255

260

Den version af det kategoriske imperativ, vi har set på her, er nok den, der forklarer KANTS tankegang bedst, men samtidig er den noget vanskelig at anvende. KANT formulerede dog også imperativet i en anden (og ifølge KANT ækvivalent) version, der er lettere at anvende i praksis, og derfor nogle gange går under navnet det *praktiske imperativ*. Dette lyder: „Handl således, at du altid tillige behandler menneskeheden, såvel i din egen person som i enhver andens person, som mål og aldrig blot som middel“ (Kant, 1999, s. 88). Denne version af det kategoriske imperativ har haft en kolossal indflydelse på den etiske debat, siden KANT formulerede den. Det praktiske imperativ giver individet en klar og tydeligt beskyttelse mod at blive udnyttet af andre, uanset om udnyttelsen er til fællesskabets bedste. Det fortæller os præcist, hvad der er galt i at inficere børnehjemsbørn med kønssygdomme for at teste en ny medicin; børnene bliver brugt udelukkende som et middel til at gøre andre mennesker raske, og det går ikke ifølge KANT.

265

270

Man skal selvfølgelig ikke forstå det praktiske imperativ på den måde, at man aldrig må bruge andre som et middel til et eller andet. Så længe man husker, at de også skal være et mål i sig selv, er det ok. Det kan man i praksis sikre på forskellig vis. I forbindelse med medicinske forsøg kan man fx sikre, at forsøgspersoner afgiver et informeret samtykke om at være med i forsøget. På den måde behandler man forsøgspersonerne som mål i sig selv idet man respekterer deres autonomi og ret til selvbestemmelse. Det er dog også klart, at et informeret samtykke let kan misbruges, hvis det ikke afgives frit og informeret. *Cookie-consents* og overdrevent lange og komplicerede slutbrugeraftaler på software er gode eksempler, hvor den oprindelige intention om at et samtykke skulle udtrykke respekt for brugeren ikke altid er opfyldt. Så selv om den type samtykke fra et juridisk synspunkt er gyldige, er det fra et (kantiansk) etisk synspunkt tvivlsom, hvilke reel værdi de har. Så her er nok et eksempel på, at jura og etik ikke altid er det samme.

275

280

285

Selv om KANTS etik har være meget indflydelsesrig, har den også sine problemer. Specielt betyder det ensidige fokus på principippet bag en handling (fremfor konsekvenserne), at teorien kommer i modstrid med almindelig moralsk intuition. I en diskussion af sin teori behandler KANT følgende eksempel (Kant, 1996): En ven, der er på flugt fra en morder, har søgt tilflugt i dit hus. Morderen kommer nu forbi og afkræver dig et svar på, om vennen

290

er i huset. Er det da etisk set korrekt at lyve for morderen? KANTS svar er et klart: Nej! Hvis du fortæller sandheden til morderen, har du opfyldt din moralske pligt, og hvad der derefter sker, er ikke dit ansvar. Man er kun forpligtet i forhold til sine egen frie handlinger, mens man, som KANT ser det, ikke har nogen etiske forpligtelser i forhold til hvordan andre vælger at udøve deres autonomi. Hvis du derimod lyver for morderen, vil du, hævder KANT, paradoksalt nok have pådraget dig et ansvar, hvis din løgn fører til, at morderen finde din ven (fx fordi vennen har sneget sig ud af dit hus, uden at du vidste det). Det kategoriske imperativ udstikker med andre ord i KANTS øjne en radikal og undtagelsesløs pligt, men til gengæld har du udelukkende ansvar for dine egne handlinger.

Eksplilet med den udspørgende morder illustrerer også et andet og mere grundlæggende problem i KANTS etik. Specielt den første udgave af det kategoriske imperativ lader til at forudsætte, at man entydigt kan finde ud af, hvilket handlingsprincip (eller maksime) en konkret handling tilhører. Det er dog ikke indlysende, at det forholder sig sådan i praksis, for hvilket princip følger man, hvis man lyver for den udspørgende morder? Lyver man, eller forsøger man at redde et menneske, der er i fare? Og hvis det ikke er klart, hvilket handlingsprincip man følger i en konkret situation, forsvinder den absolute sikkerhed, KANT ønskede i sin etiske tænkning.

## 10.4 Nagels syntese

Modsatningen mellem utilitarismens hensyn til almenvellets interesser og den kantianske deontologis beskyttelse af individet udgør en helt central og grundlæggende konflikt i den moderne etiske debat. Konflikten er konstant tilstede i den politiske og juridiske debat og udgør et væsentligt tema i populærkulturen.

Konflikten kan selvfølgelig løses ved at man tager et klart valg: Man kan vælge at være enten utilitarist eller deontolog, og så vil den normative teori, man har valgt, give (mere eller mindre) klare handlingsanvisninger i de etiske dilemmaer, man står i. Der har dog også været forsøg på at samtænke de to teorier. Vi vil her kort præsentere den amerikanske filosof THOMAS NAGELS forsøg på at gøre dette, baseret på Nagel (1979) og Nagel (1986).

Hvor utilitarismen og deontologien kan beskrives som en form for 'nede fra og op'-teorier, der tager udgangspunkt i hvert sit grundlæggende aksiom (hhv. lykkemaksimering og det kategoriske imperativ) og derfra udleder, hvad der er rigtigt og forkert i alle konkrete situationer, tager NAGEL udgangspunkt i det enkelte individets etiske virkelighed og søger at give vejledning i konkrete dilemmaer og etiske valgsituationer. NAGELS primære budskab er (i vores fortolkning), at vi ikke kan reducere etikken til et simpelt valg mellem utilitarismen og pligtetik, men at etiske valg typisk vil rumme både utilitaristiske og pligtetiske aspekter. Det etiske arbejde vil så bestå i at analysere og vægte disse aspekter i den konkrete situation.

I sin analyse af den etiske situation identifierer NAGEL to kvalitativt forskellige former for etiske værdier, nemlig *agentneutrale* og *agentrelative* værdier, og han undersøger de forskellige former for etisk relevante begrundelser, vi kan give for en handling, med udgangspunkt i disse værdier.

De agentneutrale værdier er de mest oplagte, så vi vil starte med dem. De *agentneutrale* værdier er værdier, der ikke er knyttet til et bestemt, personligt perspektiv. Folk deler dem generelt og kan derfor umiddelbart forstå, at de bør fremmes. Fravær af smerte er et eksempel på en agentneutral værdi: Hvis vi ser en person i smerte, kan vi alle identificere os med smerten, og vi kan forstå, at det vil være godt, hvis smerten hørte op. De *agentneutrale* værdier er dermed objektive i den forstand at de er universelle og søger at fremme almenvellet,

frem for enkeltindividers specifikke interesser. Det er klart, at det agentneutrale perspektiv er mere eller mindre sammenfaldende med utilitarismen, men hvor utilitarismen hævder, at de agentneutrale værdier er de eneste, der er etisk relevante, påpeger NAGEL, at der findes (mindst) tre andre former for etisk relevante begrundelser, der ikke kan reduceres til agentneutrale begrundelser, nemlig:

340

**Autonome begrundelser.** Dvs. begrundelser, hvor vi slet og ret tager udgangspunkt i vores egne autonome livsprojekter og de værdier, der følger af dem.

**Deontologiske begrundelser.** Dvs. begrundelser, der udspringer af andres ret til ikke blive behandlet dårligt af mig.

**Særlige forpligtelser.** Dvs. forpligtelser, man kan have fx over for familiemedlemmer eller i form af loyalitet rettet mod en arbejdsplads.

345

Vi vil her nøjes se nærmere på de to første af de tre typer agentrelative begrundelser.

Disse begrundelser knytter alle an til et bestemt individts unikke perspektiv og er derfor hvad NAGEL kalder *agentrelative*. De *autonome begrundelser* tager udgangspunkt i det forhold, at vi mennesker har en fri vilje og er i stand til at formulere vores egne livsprojekter og mål. Disse projekter kan spille en etisk relevant rolle, også selvom de muligvis kun giver mening fra vores eget individuelle perspektiv, og ikke nødvendigvis udspringer af eller fremmer agentneutrale værdier. Hvis du fx ønsker at løbe et maratonløb, er det næppe et projekt, der vil fremme det gode i verden, og andre mennesker vil ikke nødvendigvis kunne identificere sig med dit mål, på samme måde som de kan identificere sig med dit ønske om at blive fri for en hovedpine. Men det er et mål, du kan sætte dig, og det kan give mening set ud fra dit specifikke perspektiv. Autonome værdier og begrundelser er derfor agentrelative værdier, i den forstand at de er uløseligt knyttet til en bestemt agent, dvs. et bestemt autonoomt handlende og rationelt væsen. De har ikke værdi i en objektiv forstand, men har kun værdi i forhold til det konkrete individ.

355

Man kan hævde (og det gør NAGEL indirekte), at det har en iboende etisk værdi, at mennesker kan forfølge deres egne autonome mål og livsprojekter, men de agentrelative, autonome værdier kan ikke på nogen simpel måde reduceres til agentneutrale utilitaristiske værdier (selvom forskellige versioner af utilitarismen med diverse krumspring har forsøgt). Så hvis man ønsker at anerkende autonome begrundelser og værdier som et gyldigt, etisk perspektiv, må man acceptere dem som en selvstændig klasse af etiske begrundelser og værdier. Det er også klart, at de autonome værdier og begrundelser indimellem vil være i modstrid med agentneutrale begrundelser og værdier; Du har lyst til at løbe en maraton, men verden ville være et bedre sted, hvis du i stedet brugte din energi på fx at hjælpe folk i nød. I det omfang man giver agentrelative, autonome begrundelser forrang fremfor agentneutrale begrundelser, kommer de autonome begrundelser til at sætte en grænse for, hvor meget hensynet til fællesskabets bedste kan kræve af dig som individ. Det er dog uklart, præcis hvor denne grænse går, og en væsentlig del af det etiske arbejde i en konkret situation vil bestå i at forhandle grænsen.

365

Skellet mellem agentneutrale og agentrelative værdier gør det dog klart, at der er en tydelig asymmetri: En agentneutral værdi kan bruges til at gøre et krav på andre individer, men det kan en agentrelativ ikke. Hvis du sulter eller er i smerte, kan du have en agentneutralt begrundet forventning om, at andre vil hjælpe dig, også selv om det måske vil gå ud over deres egne agentrelative livsprojekter, men du kan ikke på samme måde gøre krav på, at andre skal hjælpe dig med at løbe en maraton; Jeg kan etisk set være forpligtet på

375

380

at give dig mad, hvis du er sulten, men jeg er ikke på samme måde etiske set forpligtet til at købe løbesko til dig, så du kan træne din maraton — det er dit eget projekt, der kun har værdi relativt til dit perspektiv. Agentneutrale værdier kan derfor (i visse tilfælde) trumfe et individts autonome værdier, men det kan aldrig gå den anden vej. Så selv om vi stadig mangler at finde ud af, præcis hvornår agentneutrale værdier har forrang for de autonome, er vi i det mindste blevet lidt klogere.  
385

Hvor de autonome værdier giver individets egne projekter en vis beskyttelse mod krav rejst ud fra hensynet til det fælles bedste, fungerer de *deontologiske værdier og begrundelser* som begrænsninger og forbud i forhold til både autonome og agentneutrale værdier. Hvis  
390 jeg skal respektere andres autonomi, er der grænser for, hvad jeg kan gøre imod dem (som KANT påpegede). Jeg kan ikke tillade mig at lyve for folk eller på anden vis behandle dem på måder, der går ud over deres integritet. De deontologiske værdier tager udgangspunkt i en konkret relation mellem mig og et anden menneske (eller en gruppe af andre mennesker), og ikke i et ønske om at gøre verden som helhed til et bedre eller lykkeligere sted. Derfor  
395 de deontologiske værdier ogsås agentrelative, og de adskiller sig fra de agentneutrale (dvs. utilitaristiske) værdier og begrundelser. Som vi har set i diskussionen mellem utilitarisme og deontologi, er deontologiske værdier og begrundelser langt fra altid i overensstemmelse med agentneutrale, utilitaristiske værdier; der kan opstilles scenarier, hvor hensynet til det fælles bedste kræver, at vi på den ene eller anden måde overskrider andre menneskers autonomi.  
400 Deontologiske værdier kan altså ikke reduceres til agentneutrale, og de deontologiske værdier udgør dermed et selvstændigt tredje niveau af etiske værdier og begrundelser, der giver en begrænsning både i forhold til mine egne autonome værdier og i forhold til agentneutrale værdier, der søger at fremme almenvellet. Spørgsmålet er naturligvis, i hvilken grad den deontologiske begrænsning skal honoreres. NAGEL mener tilsyneladende, at begrænsningen skal forstås som absolut (dvs. at man aldrig må overskride deontologiske forpligtelser), men  
405 det kan naturligvis diskuteres.

Opsamlende kan man sige, at agentneutrale (utilitaristiske) værdier udspringer af et tredjepersonsperspektiv, hvor man ser på verden udefra som en neutral tilskuer. De agentrelative, deontologiske værdier udspringer af et andenpersonsperspektiv, der tager udgangspunkt i de relationer, mennesker kan have med hinanden, mens de agentrelative, autonome værdier tager udgangspunkt i et førstepersonsperspektiv, der er bundet til den enkeltes individuelle projekter. Alle de tre perspektiver er væsentlige i en nuanceret etisk vurdering af en situation, og ingen af dem kan reduceres til de andre. Første- og andenpersonsperspektiv gør det ikke muligt se almenvellets interesser tydeligt, men et rent tredjepersonsperspektiv, som det, utilitarismen indfører, overser også noget væsentligt: Der er forskel på, om noget sker for nogen, eller om det sker for mig, og der er forskel på, om noget bare sker, eller om det er mig, der får det til at ske. Alle tre perspektiver er dermed nødvendige og (hvis man accepterer tankegangen ovenfor) legitime.  
410  
415

## 10.5 Analyser i praksis

Både utilitarismen og KANTS deontologi kan give nogenlunde klare svar på, hvordan man (ifølge dem) skal handle i en given situation. NAGELS beskrivelse af de forskellige etiske værdier giver ikke den slags klare svar, og det er heller ikke pointen. Pointen er snarere, at man med udgangspunkt i en forståelse af de forskelligartede værdier, der er i spil i en given situation, kan give en mere nuanceret etisk analyse og vurdering. Der findes ikke nogen simple algoritme for, hvordan de etiske valg skal træffes, men man kan, med baggrund i  
420  
425

NAGEL, give en skabelon for, hvordan et etisk problem kan addresseres mere nuanceret. Vi vil her gengive en skabelon, der er bearbejdet og forkortet på grundlag af Christiansen (2015, s. 41):

- Identificér hvori det etiske dilemma består: Mellem hvilke grundlæggende etiske intuitioner er der konflikt? 430
- Hvilke etiske grundværdier er de etiske intuitioner, der er i konflikt med hinanden, baseret på (autonome værdier, deontologiske værdier, utilitaristiske værdier)? Siger det noget om, hvordan de forskellige intuitioner skal vægtes i forhold til hinanden?
- Hvilke interesser er involveret i konflikten, og hvilke interesser har de hver især (1., 2. og 3. personperspektiver)? 435
- Er der ikke-etiske perspektiver, fx konkret empirisk viden, der kan kaste lys over situationen og styrke den etiske analyse?
- Er der løsninger, hvor de etiske intuitioner, der er i konflikt med hinanden, kan respekteres? Og hvis ikke: Hvordan er det nødvendigt at modifcere de etiske intuitioner du startede med? Kan du leve med de modifikationer i andre situationer?
- Beskriv dit løsningsforslag så konkret som muligt. 440

For at illustrere, hvordan en etisk analyse kan se ud i praksis, kan vi kort se på en sag fra 2015, hvor en datalogistuderende skrev speciale i samarbejde med Macom, firmaet bag læringsplatformen *Lectio*, som de fleste danske gymnasier bruger (Behrendt, 2014). Den studerende brugte data fra *Lectio* om godt 70.000 gymnasielever og byggede ved hjælp 445 af machine learning en model, der med 93% sikkerhed kunne forudsige, om en given elev ville droppe ud af gymnasiet inden for de næste tre måneder. Modellen blev kortvarigt implementeret som en funktion i *Lectio*, men blev fjernet efter en uge efter krav fra Danske Gymnasier (Møllerhøj, 2015).

Vi vil ikke lave en egentlig analyse af sagen, men blot illustrere, hvordan NAGELS teori 450 kan bringes i spil i den etiske undersøgelse af en konkret sag. En etisk analyse af situationen kunne som deontologiske perspektiver fx inddrage det forhold, at de elever, hvis data modellen blev bygget på, ikke havde givet samtykke til, at deres data blev brugt til at lave en model af den slags. Man kunne også inddrage det forhold, at eleverne ikke fik del i den økonomiske gevinst, implementeringen af modellen kunne have givet Macom (hvis modellen ikke var blevet fjernet) og det, at eleverne ikke havde noget reel valg i forhold til at benytte *Lectio*. Som et utilitaristisk perspektiv kunne en etisk analyse overveje modellens troværdighed og rækkevidde (ikke mindst i lyset af induktionsproblemet og de forskellige andre metodologiske problemer machine learning-modeller generelt står over for, se kapitel 5). Fra et utilitaristisk perspektiv ville det dog også være relevant at gøre sig nogle grundige overvejelser om de forudsebare konsekvenser indførelsen af en model vil have på elever og læreres adfærd. Her vil det muligvis være relevant at indhente ny viden fx gennem pilotstudier eller interviews med lærere og elever, så man ikke er nødt til apriori at antage, at en implementering af modellen vil fremme nytten for de involverede parter. Udover de 460 2.- og 3.-personperspektiver, som deontologien og utilitarismen leverer, har de involverede parter desuden haft deres egne 1.-personperspektiv på sagen, hvor udviklingen af modellen på forskellige vis har spillet en rolle i deres egne, individuelle livsprojekter. Det overordnede spørgsmål er nu, hvordan de forskellige perspektiver skal vægtes i forhold til hinanden,

og om det med passende justeringer (fx brug af informeret samtykke, bedre afdækning af modellens konsekvenser) vil være muligt i højere grad at tilgodehæNSE flere af de konkurerende perspektiver.

## 10.6 Big data-etik og privacy

Som et andet eksempel på hvordan flere etiske perspektiver kan bringes i spil, kan vi se mere principielt på den etiske diskussion af brugen af big data. I et review identificerede de 475 to etikere BRENT MITTELSTADT og LUCIANO FLORIDI følgende fem dominerende temaer i litteraturen om big data-etik (Mittelstadt og Floridi, 2016, 311ff):

1. *Informert samtykke*, dvs. spørgsmålet om, hvorvidt folk giver reel tilladelse til, at deres data bliver brugt til de formål, de bliver brugt til.
2. *Privacy*, dvs. spørgsmålet om hvorvidt analyse af big data kan overskride retten til privatliv fx ved at udlede følsomme oplysninger fra ikke-følsomme.
3. *Ejerskab*, dvs. spørgsmålet om, hvem der ejer og har råderet over folks data (og hvorvidt folk bliver tilstrækkeligt kompenseret for den værdi, deres data besidder).
4. *Epistemologi*, dvs. spørgsmålet om troværdigheden af de konklusioner, man når frem til via en given big data-analyse.
5. „*Big data-skellet*“, dvs. den asymmetri i både magt og viden, der er omkring brugen af big data (et firma som Facebook har en langt bedre forståelse af, hvad data kan bruges til, end den typiske bruger og kan (til en vis grænse) diktere brugerbetingelserne, da man ikke kan sige nej til dem).

Af disse repræsenterer tema 1, 2, 3 og 5 klare deontologiske perspektiver, der i grove træk udtrykker bekymring for, om folks autonomi og ret til selvbestemmelse bliver respekteret (1 og 2) og for, om brugen af big data fører til, at man stjæler fra eller narre folk (3 og 5). Tema 4 adresserer derimod nytteværdien af big data-analyser og repræsenterer derfor et utilitaristisk (agent-neutralt) perspektiv. Vi vil her nøjes at gå i dybden med spørgsmålet om privacy, dvs. retten til privatliv.

Spørgsmålet om privatliv er ikke nyt, men det er i meget høj grad blevet aktualiseret af udviklingen af ny informationsteknologi, idet vi (både bevidst og ubevist) efterlader os utallige digitale spor, der meget let kan samles, deles og samkøres, og fordi machine learning og andre former for statistisk analyse gør det muligt at prediktere data, der tilhører andre kategorier af personoplysninger end de data, der er tilgængelige. Det sidste er fx det der sker, hvis man bruger folks indkøbsvaner til at prediktere om de er gravide, eller folks Facebook-adfærd til at prediktere deres personlighedstype (Youyou, Kosinski og Stillwell, 500 2015).

Til at starte med kan vi overveje, hvorfor privatliv overhovedet er værd at beskytte. Et svar går på, at vi har et ønske om selv at kunne styre, hvem der ved hvad om os (det kaldes *kontekstuel integritet*). Når vi fortæller, hvad vi lavede i weekenden, vil vi måske fortælle nogle detaljer til vores mor, andre til vores arbejdskolleger eller studiekammerater og andre igen til vores kærester. Vi opbygger forskellige narrativer om os selv i forskellige sociale kontekster, og retten til privatliv sikrer netop vores mulighed for at udfolde vores autonomi på det område. Problemstillingen bliver forstærket af, at visse typer af oplysninger

direkte kan skade én på forskellig vis, hvis de falder i de forkerte hænder; I en del lande er homoseksualitet fx belagt med dødsstraf og tilsvarende kan forskellige former for politiske aktiviteter og religiøse tilhørersforhold være en trussel mod ens liv og sikkerhed. Som et lidt mere fredsommeligt eksempel kan man også have gode grunde til ikke at fortælle sin chef, at man har været til jobsamtale i det konkurrerende firma, og man kan også have gode grunde til ikke at offentliggøre på Facebook hvilken julegave, man har købt til sin kone (som Facebook kom til at gøre det en overgang i det såkaldte *Beacon-program*, hvor indkøb fra visse butikker automatisk blev delt med alle ens facebookvenner, se Nakashima, 2007). Privacy er således væsentligt fra et 1.- og 2.-personsperspektiv, dvs. for at vi kan udøve vores autonomi og følge egne autonome projekter uden at blive reduceret til et middel for andres mål. Argumentet, at man ikke har brug for privatliv, hvis man ikke har noget at skjule, er dermed grundlæggende forfejlet: Alene det at kunne bestemme, hvad man deler hvornår og med hvem er en grundlæggende del af det at udøve sin autonomi. 510  
515  
520

Retten til privatliv er primært blevet udfordret af kommercielle hensyn og af hensynet til effektivitet og sikkerhed. I forhold til de kommercielle hensyn er det bemærkelsesværdigt, at folk i så høj grad har været villige til frivilligt at afgive persondata til private virksomheder som Google og Facebook. Spørgsmålet er selvfølgelig, hvor frivillig afgivelsen af data reelt har været, og specielt om folk forstår og er tilstrækkeligt oplyste om, hvilke informationer, de egentligt afgiver (jf. MITTELSTADT og FLORIDIS tema 1 og 2 ovenfor). Tilsvarende kan man hævde, at tjenester som Google Translate og Google Maps bliver mere effektive, jo flere data vi overgiver til dem — og hvem vil ikke gerne have adgang til gode oversætttere og trafikinformation om bilkører mv. på ens typiske vej til arbejde. Så også effektivitet kan i visse tilfælde stå i modsætning til ønsket om privatliv. Spørgsmålet er selvfølgelig, om den type tjenester bliver mere effektive til at hjælpe dem, der afgiver data, eller om de private data blot er et middel til at opnå andre (kommercielle) mål. Fra et kantiansk synspunkt er der en afgørende forskel på de to situationer pga. det praktiske imperativ. 525  
530  
535

Det er dog nok i forhold til spørgsmålet om sikkerhed, at den etisk set mest betydningsfulde og vanskeligste konflikt i forhold til privatliv opstår. Her er det typiske argument, at kriminelle har lettere spil, hvis de kan skjule sig i bag privatlivets fred, og derfor bliver vi alle — for vores egen sikkeheds skyld — nødt til at acceptere et vist niveau af overvågning. Hvis man fortolker sikkerhed til også at omfatte sundhed, ser man den samme konflikt: Hvis et system som Google FluTrends, der skulle forudsige influenza-epidemier (se Lazer m.fl., 2014), rent faktisk havde fungeret, ville der have været et klart argument for, at vi alle skulle overgive vores søgerdata (og måske også andre private data, som køn, alder, arbejdssted, sociale kontakter mv.) til Google, så modellen kunne fungere bedst muligt og dermed gøre verden til et sikkert og mere influenzafrit sted. 540  
545

Der er dermed en række klare og vanskelige konflikter mellem privatliv og andre hensyn.

I forhold til den kommercielle udnyttelse af private oplysninger er det måske ikke så meget en konflikt, men mere et spørgsmål om, hvorvidt virksomhederne respekterer grundlæggende deontologiske forpligtelser. I forhold til spørgsmålet om sikkerhed, er konflikten mere reel og udspiller sig som nok et klart eksempel på den klassiske konflikt mellem deontologiske og utilitaristiske hensyn. Her antyder historien (uden at vi vil gå i dybden med det), at det er let at hylde de deontologiske principper, når alt går godt, men så snart der viser sig en trussel – fx terrorangreb eller en pandemi – har de utilitaristiske hensyn det med at få forrang. 550

Af samme grund er det, som etikeren DEBORAH JOHNSON påpeger, væsentligt at gøre sig klart, at den rent deontologisk funderede begrundelse, vi ovenfor gav for privatliv, faktisk ikke er fuldstændig (Johnson, 2009, 95ff). Privatliv er ikke blot et personligt gode, men 555

også et socialt gode, idet den grundlæggende autonomi og integritet, beskyttelse af privatlivet muliggør, ifølge JOHNSON er en afgørende forudsætning for at have et demokrati. Et  
560 demokrati kan kun fungere, hvis der er plads til at tænke anderledes og mulighed for at gøre og afprøve nye ting, som de siddende magthaverne måske ikke billiger. Uden en privatsfære, hvor man kan tage selvstændig stilling til og indgå med andre borgere i en kritisk dialog om magthavernes beslutninger kan man ikke have et demokrati. Hvis det argument er korrekt, er spørgsmålet om privatliv vs. sikkerhed ikke blot et spørgsmål om individuelle rettigheder mod almenvellets velfærd, men også et spørgsmål om forskellige former for generel velfærd.  
565 Sat på spidsen kan man sige, at valgte også står mellem den sikkerhed man kan få med totalitær overvågning og de fordele, et oplyst, inddragende demokratisk styre giver. Man kan i længden ikke have begge dele.

## 10.7 Bias og algoritmisk transparens

570 Brugen af algoritmer giver også en anden etisk konflikt, nemlig en konflikt mellem effektivitet og retfærdighed. Det har i århundrede været helt almindeligt at bruge datadrevne sandsynlighedsberegninger til at støtte beslutninger, der går ud på at vurdere en form for risiko. Hvis man fx tegner en ulykkesforsikring, vil forsikringsselskabet bruge deres viden om, hvor hyppigt folk, der på relevante træk ligner dig, kommer til skade, til at fastsætte forsikringspræmien. Det er ret ukontroversielt. Det er straks mere kontroversielt, at man bl.a. i det amerikanske retsvæsen siden starten af den 20. århundrede har brugt lignende beregninger til at vurdere kriminelles risiko for at begå ny kriminalitet – og at den type beregninger er blevet brugt i vurderingen strafudmåling, prøveløsladelser mv. (Carlson, A. (2017). The Need for Transparency in the Age of Predictive Sentencing Algorithms. *Iowa Law Review*, 103(1), 303–329.). I de sidste årtier har lettere adgang til store datamængder og fremkomsten af diverse machine learning teknikker gjort det lettere at udvikle den type systemer og diverse former for prædiktive algoritmer bruges i dag på en lang række områder fra reklamer til sorterings af jobansøgninger.

575 Det er dog ikke helt uproblematisk at bruge prædiktive algoritmer. Som vi så tidligere, er der en række epistemiske problemer forbundet med machine learning. Der er typisk en meget direkte sammenhæng mellem epistemiske og etiske overvejelser; hvis de etiske overvejelser inddrager utilitaristiske komponenter er det i høj grad væsentligt at forstå, hvor godt en given algoritme i praksis virker. Ville den etiske vurdering af algoritmen til forudsigelse af frafald i gymnasiet fx falde anderledes ud, hvis systemet kun havde en træfsikkerhed på 67%? Der er imidlertid også et anden og vanskeligere etisk aspekt specielt hvis man bruger algoritmiske forudsigelser til at træffe afgørelser, der har betydning for enkeltindividens muligheder. Algoritmen til forudsigelse af frafald inddrog således direkte elevernes etnicitet som en faktor i beregningen, men hvad nu hvis algoritmen havde en bias så den systematisk gav elever med en etnicitet en højre risikovurdering end elever med en anden?

580 Spørgsmålet om diskrimination i algoritmer er navnligt blevet diskuteret i forbindelse med den såkaldte COMPAS-algoritme, der bruges i det amerikanske retsvæsen til vurdering af kriminelles tilbagefaltsrisiko. En gennemgang af algoritmen viste imidlertid en systematisk skævhed i de fejl, algoritmen lavede. Således havde farvede, der ikke senere begik ny kriminalitet, en langt højere chance for fejlagtigt at få en høj risikoscore end hvide, og hvide havde omvendt en højere chance for fejlagtigt at få en lav score end farvede (Angwin m.fl., 2016).

Hvis man imidlertid tager udgangspunkt i de to kategorier høj- og lav-risiko, og under-

søger, hvor god algoritmen er til at komme med korrekte forudsigelser, er der paradoksalet nok ingen forskel på farvede og hvide; blandt de, der blev bedømt som højrisiko, var procentdelen af farvede, der var blevet fejlplaceret, nogenlunde lige så stor som procentdelen af hvide, og tilsvarende for de, der blev vurderet som højrisiko. Det betyder med andre ord, at risikoscoren er nogenlunde lige pålidelig, uanset hvilken etnicitet den person, den bliver lavet for, har. Ydermere har det vist sig, at man ikke kan have lighed for begge mål samtidig. Enten vil algoritmen som nu være lige pålidelig i sine forudsigelser uanset etnicitet, men til gengæld have en systematisk skævhed i forhold til enkeltpersoners risiko for at blive fejlplaceret, eller også kan man gøre risikoen for at blive fejlplaceret lige stor uanset etnicitet, hvilket til gengæld betyder, at algoritmen vil være mindre pålidelig for den ene etniske gruppe end for den anden (for en stringent matematiske gennemgang, se Kleinberg, Mullainathan og Raghavan, 2017). Det er selvfølgelig en interessant etisk udfordring!

605

Når man skal diskutere hvorvidt brugen af prædictive algoritmer kan forsvares etisk er der er en række ting, det kan være godt at få på plads. For det første er de etiske argumenter for at bruge prædictive algoritmer typisk utilitaristiske; den grundlæggende påstand er, at en algoritme som COMPAS på den eller anden måde øger den samlede nytte i samfundet og det er på den baggrund algoritmen skal vurderes (der kan selvfølgelig også være ikke-etiske grundelser som kommercielle interesser eller andet).

610

For andet skal være klar over, at prædictive algoritmer altid diskriminerer, i den forstand at din fremtidige adfærd estimeres på baggrund af den fortidige adfærd af den gruppe, som du anses for at tilhøre. Hvis man fx vil estimere hvor længe en 25 årig, ikke-rygende, københavnsk studerende lever, vil man se på, hvor længe folk i en referenceklasse af personer, der minder om vores 25-årige studerende i gennemsnit har levet. Det er ganske simpelt sådan, statistiske forudsigelser fungerer (i det mindste i den frekventistisk tolkning af sandsynlighed, som er mest naturlig i denne sammenhæng). Spørgsmålet er altså ikke om algoritmerne må diskriminere, men i højere grad hvilke faktorer, vi vil tillade, at der indgår i referenceklassen. Hvilke træk ved dig, kan man det etisk set forsøre at diskriminere på baggrund af, når du skal have en livsforsikring, ansøger om et job eller søger ind på en uddannelse?

620

For at svare på det spørgsmål bliver vi for det tredje nødt til at inddrage begrebet retfærdighed. Begrebet retfærdighed befinner sig et sted imellem etik og politisk filosofi, idet det angår mere generelle egenskaber samfund, som fordeling af goder, adgang til politisk magt og det juridiske straffesystem. Præcis som det er tilfældet for etik findes der mange forskellige retninger indenfor politisk filosofi, og mange forskellige ideer om, hvad retfærdighed er. Det vil vi på ingen måde gå i dybden med her. Vi vil nøjes med at præsentere en enkelt og meget operationel forestilling om retfærdighed, til dels med udgangspunkt i den amerikanske filosof JOHN RAWLS (1921–2002). RAWLS hævder (Rawls, 1972), at hvis samfundet skal være retfærdigt, skal vi tage alle grundlæggende beslutninger om det indretning og institutioner, mens vi befinner os bag et *videnhedens slør*, der skjuler, hvilken position vi selv vil få i samfundet. Så man skal altså forestille sig, at man ikke ved, hvilket køn man har, at man ikke ved, om man er rig eller fattig, at man ikke kender sin etnicitet mv., og ud fra den position kan man beslutte, hvordan samfundet skal indrettes. I dette tilfælde: Hvilke former for diskrimination vil vi tillade i prædictive algoritmer? Det er let at forestille sig (men det kan selvfølgelig diskuteres) at man i den situation vil vælge et samfund der er uden kønsdiskrimination og ude diskrimination mode fx seksuelle og etniske minoriteter. Så simpelt er det.

630

640

645

Og så alligevel ikke, for der kan være principielt forskellige former for diskrimination. I nogle tilfælde diskriminerer man ud fra en faktor, der slet ikke har nogen betydning for det udkomme, man er interesseret i at optimere. Fx ved man, at videnskabelige artikler har

650

letterer ved at bliver publiceret, hvis der bruges et mandligt forfatternavn end hvis der bruges et kvindeligt. Da det udelukkende er artiklens kvalitet og ikke forfatterens køn der skal afgøre, om en artikel er publicerbart eller ej, er der her tale om direkte (køns)diskrimination.  
655 I andre tilfælde er sagen imidlertid mindre klar. Fx lever mænd kortere end kvinder (en tommelfingerregel siger, at det er lige så farligt at være mand som at være overvægtig). Så hvis man tegner en livsforsikring, burde mænds præmier være højere end kvinders. Hvis præmien er ens for begge køn kommer de i gennemsnit fornuftige kvinder til at betale for de i gennemsnit ufornuftige mænds dumheder. Man kan sige, at køn i forhold til livsforsikring er en informationsbærende dimension, så hvis man afskærer algoritmen fra at bruge din  
660 dimension, vil den naturligvis blive mindre effektiv. Og i dette tilfælde er det kvinderne, der vil komme til at betale for den manglende effektivitet. Hvis man omvendt inddrager køn i beregningen af pensionspræmie, vil en mand, der opfører sig fornuftigt og tager sit helbred seriøst, komme til at betale en højere præmie, udelukkende fordi han har samme køn som en gruppe, der – gennemsnitligt – opfører sig ufornuftigt. Og hvad gør man så? Er  
665 det mest retfærdige at kønsdiskriminere i dette tilfælde (og lade de fornuftige mænd betale prisen) eller er det mest retfærdigt at undlade at kønsdiskriminere (og lade kvinderne betale prisen)?

Det bringer os tilbage til COMPAS-algoritmen. I den population, hvor algoritmen er blevet testet, har farvede fanger reelt en større risiko for at begå ny kriminalitet. Når  
670 algoritmen diskriminerer på baggrund af etnicitet er det derfor blot et udtryk for, at den har identificeret en informationsbærende dimension i datasættet. Og lige som ovenfor vil vi have valget imellem at forhindre algoritmen i at bruge informationen, hvorved den vil blive mindre effektiv i den forstand at dens overordnede fejrlate vil stige, eller at tillade den at  
675 bruge informationen, hvorved den vil diskriminere enkeltindivider, således at farvede, der ikke vil begå ny kriminalitet har en meget større risiko for at få en høj kriminalitetsscore end ikke-farvede. Hvilket af de to scenarier forekommer dig mest retfærdigt, når du står bag uvidenhedens slør?

Der er dog også væsentlige forskelle på de to cases. Specielt kan man argumentere for, at den gennemsnitligt høje kriminalitetsrate blandt farvede til dels skyldes strukturelle forhold  
680 i det amerikanske samfund; pga. en generel diskrimination mod farvede har de sværere ved at få uddannelser og job end ikke-farvede, og derfor vil de hyppigere ende i kriminalitet. En algoritme som COMPAS, der har en større tilbøjelighed til at holde farvede i fængsel end ikke-farvede, vil bidrage til fastholde denne strukturelle ulighed. En datadrevne algoritme afspejler virkeligheden, som den er, men hvis virkeligheden er racistisk, vil algoritmen også blive det, og hvis algoritmen bliver brugt uden forbehold, kan den bidrage til at fastholde historisk betingede diskriminerende strukturer. Der er derimod ikke noget, der tyder på, at mænds kortere levealder er et udtryk for strukturel diskrimination. Så med andre ord er  
685 det vigtigt, at se på den kontekst, en algoritme skal indgå i, når man overvejer, om den er retfærdig.

## 690 Pandoras black box

I en traditionel statistisk model vil man have nogenlunde styr på de variable, modellen inddrager. Det er dog værd at bemærke, at tingene selv med traditionelle algoritmer ikke altid er så klare; COMPAS-algoritmen inddrager således ikke direkte etnicitet som en variabel. Den inddrager til gengæld flere variable, der er tæt korrelerede med etnicitet, og dermed indgår etnicitet indirekte i algoritmen. Det gør naturligvis tingene mere besværlige,  
695

at man på den måde kan komme til at bruge variable, man ikke direkte har indbygget i sin algoritme.

Det problem bliver imidlertid kraftigt forstærket, hvis man træner en algoritme med machine learning. Her har vi styr på de data, algoritmen er trænet med, og vi kan opstille forskellige mål for, hvor godt den virker, men resten er en black box; vi aner reelt ikke hvorfor den virker, og som vi berørte ovenfor, kan machine learning algoritmer med lethed diskriminere langs variable, der er af en helt anden type end de data, vi har trænet algoritmen med. Hvis vi vil sikre, at de algoritmer, vi udvikler, ikke diskriminerer på en etisk kritisabel måde, bliver vi nødt til at gøre dem transparente. Det er imidlertid ikke altid let, at gøre en blackbox transparent!

700

705

## 10.8 Systemisk etik

Efter at have set på de normative teorier vil vi nu vende tilbage til den deskriptive etik og beskrive et interessant og i høj grad overset aspekt af etikken, nemlig det tilsvyneladende empiriske faktum, at vores etiske beslutninger kan påvirkes af selv små ændringer i vores miljø. Som et enkelt eksempel testede JOHN M. DARLEY (1938–2018) og DANIEL BATSON teologistuderende i et eksperimentelt paradigme, hvor en forsøgsperson (typisk en studerende) bliver bedt om at forberede et kort oplæg om den barmhjertige samaritaner, som er en lignelse fra Det Nye Testamente, der i korte træk går ud på, at man ubetinget skal hjælpe andre i nød, også selv om det er en vildt fremmet af et andet folk (Darley og Batson, 1973). Forsøgspersonen skulle herefter gå hen til et andet lokale for at holde sit oplæg. På vejen kom forsøgspersonen forbi en skuespiller, der, forklaædt som hjemløs, fingerede et ildebefindende. Eksperimentet gik nu ud på at undersøge, hvor mange af forsøgspersonerne, der faktiske stoppede op for at hjælpe den fremmede i nød. Resultatet var ikke så ringe—i den første setup forsøgte 63% af forsøgspersonerne at hjælpe den hjemløse. I den anden variant af forsøget lavede man imidlertid én lille ændring: Netop som forsøgspersonen gik ud af forberedelseslokalet kiggede forsøgslederen på sit ur og fortalte forsøgspersonen, at vedkommende havde travlt. Det fik antallet af forsøgspersoner, der standsede op, til at falde til 10%. Med andre ord viser eksperimentet, at selv små ændringer i vores miljø kan have en stor indflydelse på de moralske valg, vi tager, her om man skal hjælpe eller ikke-hjælpe en fremmed i nød.

710

715

720

725

Man skal absolut ikke lægge for meget i et enkelt adfærdsspsykologisk eksperiment (med lav statistisk kraft) som det ovenstående. Eksperimenter er primært en illustration. Det slående er snarere, at effekten tilsvyneladende er robust og kan gentages i forskellige varianter og eksperimentelle setups (se fx Pascual-Ezama m.fl., 2015).

Det er ikke helt oplagt, hvordan man skal fortolke den slags resultater fra et etisk synspunkt. Undersøgelser af, hvordan folk faktisk handler siger jo, som nævnt i indledningen, ikke noget om, hvordan vi bør handle. Og så alligevel. Fra en overfladisk betragtning kan man hævde, at kontekstafhængigheden af vores moralske valg mindsker vores personlige etiske ansvar: Hvis vores etiske beslutninger styres af konteksten, er det jo ikke rigtig vores skyld alligevel. Og dog. Man kan lige så vel sige, at den type resultater gør os opmærksom på et nyt og anderledes etisk ansvar, nemlig vores medansvar for, at den kontekst, vi befinner os i, tilskynder os til at opføre os etisk korrekt og ikke omvendt. Med andre ord kan man hævde, at man har et ansvar for at være opmærksom på, om de strukturer og den kultur, der er i den virksomhed eller det fagfællesskab, man befinner sig i, på den ene eller anden måde tilskynder til tvivlsom eller uetisk adfærd. Og hvis det er tilfældet, må man forsøge at ændre

730

735

740

kulturen eller flytte sig. For at vende tilbage til figur 1 ovenfor, må man med andre ord have en opmærksomhed på, om den lokale praksis, man befinner sig i, er i overensstemmelse med, hvad der er etisk korrekt.

I en forskningsetisk sammenhæng er det interessant, at man i den *code of conduct*, som alle danske universiteter har ratificeret, til dels har forladt det entydige fokus på individets ansvar (*Danish Code of Conduct for Research Integrity* 2014). Således betones det, at det ikke kun er individuelle forskere, der har et ansvar for at optræde ordentligt, men at institutionerne også har et ansvar for at skabe strukturer og arbejdsbetingelser, der gør det muligt for forskerne at optræde etisk og redeligt. Spørgsmålet er så selvfølgelig, i hvilket omfang de strukturer er på plads, hvilket peger hen på temaet om redelighed og forskningspolitik.

## 10.9 IT-professionelles etiske ansvar

Foreløbig har vi set på, hvordan man kan håndtere etiske valgsituationer, men vi har endnu ikke undersøgt, om IT-professionelle i det hele taget har et etisk ansvar eller hvor langt det ansvar egentlig går.

I en artikel, der diskuterede forskeres ansvar for militære anvendelser af deres forskning, behandlede den danske etiker JESPER RYBERG en række af de undskyldninger, forskere typisk kommer med, for at unddrage sig et etisk ansvar for deres forskning (Ryberg, 2003). Den første undskyldning går på, at staten håndterer moralen, idet staten gennem lovgivning sætter grænser for, hvad man må og hvad man ikke må. Som vi så ovenfor, holder denne undskyldning ikke, specielt for områder med hastig teknologiudvikling, da undskyldningen forudsætter, at lovgiverne både har perfekt moralsk og teknisk indsigt. En anden mulig undskyldning påpeger, at det ligger i videnskabens natur, at man hverken kender udfaldet af et eksperiment eller kan forudsige dets anvendelser. Det er korrekt, at man ikke altid ved, hvor man ender, når man bedriver videnskab, men i mange tilfælde har man faktisk klare mål og en klar forventning om, hvad der kommer ud af et givet forskningsprojekt. Og hvad angår anvendelserne, kan man som minimum gøre en fokuseret indsats for at overveje, hvad den teknologi, man udvikler, faktisk vil kunne bruges til. Så den undskyldning har også begrænset rækkevidde. En tredje undskyldning går på, at det jo ikke er os som teknologiudviklere, der bestemmer, hvad resultaterne af vores forskning skal bruges til. Vi gør det kun muligt. Den undskyldning fungerer heller ikke rigtigt. Hvis vi husker på casen med KANTS udspørgende morder, var problemet jo netop, at vi ikke bare kan hjælpe andre med at udføre en handling, vi ved er etisk forkert, uden selv at pådrage os et vist ansvar. Så med mindre man er parat til at blive en stålsat kantianer med alt, hvad deraf følger, virker 'vi gør det bare muligt'-undskyldningen ikke. Som en sidste undskyldning kan man hævde, at hvis ikke jeg havde gjort det, ville en anden bare have gjort det. Eller med andre ord, at man er *udskiftbar*. Den undskyldning fungerer dog heller ikke. Hvis man stjæler en pung, som nogen har glemt i bussen, vil man ikke nå langt ved at hævde, at hvis jeg ikke havde stjålet den, ville en anden have gjort det. Man tage ansvar for sin handling uanset hvad.

Der kan naturligvis være andre undskyldninger, men med mindre de fungerer bedre, end de fire, vi har set ovenfor, har man tilsyneladende som teknologiudvikler et ret udstrakt ansvar for konsekvenserne af den teknologi, man udvikler. Dominerende faglige foreninger som Institute of Electrical and Electronics Engineers (IEEE) og (specielt) Association for Computing Machinery (ACM) har da også meget klare etiske kodeks, der tydeligt tilskriver

foreningernes medlemmer et ganske udstrakt etisk ansvar. Ifølge IEEE's etiske kodeks accepterer medlemmerne således bl.a. „to hold paramount the safety, health, and welfare of the public, to strive to comply with ethical design and sustainable development practices, and to disclose promptly factors that might endanger the public or the environment“ (History of Computing, u.å.). Hos ACM finder man tilsvarende krav om, at IT-professionelle skal  
790  
bruge deres viden og kunnen til samfundets bedste og til at fremme menneskerettigheder og individers autonomi (ACM, u.å.). Og tilsvarende punkter findes også i dansk sammenhæng implementeret i PROSA's etiske retningslinjer (PROSA, 2000).

Begge de amerikanske kodekser indeholder dog også paragraffer, der rækker ud over den enkeltes eget arbejde. ACM's kodeks rummer således et eksplisit krav om, at man skal  
795  
'blow the whistle' (ACM, u.å., sektion 1.2), hvis man ser tegn på, at et system kan gøre skade, og hos IEEE hedder det, at man skal „seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors, and to credit properly the contributions of others“ (History of Computing, u.å.).

Det er en interessant diskussion, hvor langt dette ansvar egentlig går. Man kan argumentere for, at man som IT-professionel sidder inde med en særlig viden, og at man af den grund har en særlig pligt til at reagere, hvis man bliver opmærksom på misbrug eller tvivlsom brug af computerteknologi, også selv om man ikke selv har været med til at designe de pågældende systemer. Med store kræfter kommer et stort ansvar! Det er dog også klart, at det kan have store omkostninger for den enkelte IT-professionelle at påtage sig det ansvar  
800  
både i form af ekstraarbejde og repressalier, og derfor må den utilitaristisk funderede pligt til at handle afvejes i forhold til den enkeltes autonome projekter og mål.  
805

## 10.10 Redelighed

Spørgsmålet om videnskabelig redelighed kom for alvor på dagsordenen i årene lige efter årtusindskiftet, hvor en række store skandaler viste, at videnskaben langtfra var så selv-korrigende, som man hidtil havde troet, men at bevidst og grov uredelighed faktisk fandt sted. Det var skandaler, som sagen om den tyske faststoffysiker JAN HENDRIK SCHÖN, der en overgang publicerede omkring en artikel om ugen i de mest prestigefyldte tidsskrifter, indtil det viste sig, at mange af artiklerne var skrevet på baggrund af delvist konstruerede data. Eller sagen om den nederlandske psykolog DIEDERIK ALEXANDER STAPEL, der publicerede  
810  
mere end 50 artikler på baggrund af frit opfundne data, og den koreanske læge HWANG Woo-SUK, der publicerede en række banebrydende artikler inden for stamcelleforskning, men også på baggrund af konstruerede data. I Danmark er den mest prominent sag til dato formentlig sagen om hjerneforskeren MILENA PENKOWA, der på linje med STAPEL og SCHÖN publicerede artikler på baggrund af konstrueret data fra forsøg, der aldrig havde  
815  
fundet sted.  
820

Nogenlunde samtidig med disse store skandaler blev man klar over, at direkte uredelighed måske slet ikke var det største problem i forhold til videnskabens integritet. En række store spørgeskemaundersøgelser af forskeres adfærd viste, at alvorlige overskridelser som plagiering og fabrikation af data kun forekommer sjældent (eller i hvert fald indrømmede  
825  
kun få forskere i undersøgelserne at de havde gjort den slags), men til gengæld var mindre overskridelser af normerne forbavsende udbredte. I den mest berømte af undersøgelserne, der involverede mere end 3000 amerikanske forskere, indrømmede kun 0,3% af forskerne, at de havde forfalsket eller manipuleret med data, men 15,5% svarede, at de havde ændret  
830  
på en undersøgelses metode eller resultatet efter pres fra en sponsor, 15,3% svarede, at de

havde slettet afvigende data udelukkende ud fra en mavefornemmelse af, at datapunkterne var upræcise og 10,0% svarede, at de havde givet eller fået ufortjente forfatterskaber (Martinson, Anderson og Vries, 2005). Andre undersøgelser havde nogenlunde sammenlignelige resultater (Fanelli, 2009), og man begyndte at bruge betegnelsen *tvivlsom forskningspraksis* til at beskrive den type afdærd, der ligger i gråzonen mellem acceptabel forskningspraksis og direkte uredelighed.

I 1981 fortalte PHILIP HANDLER (1917–1981), der på det tidspunkt var præsident for det amerikanske National Academy of Sciences den amerikanske kongres, at videnskabelig snyd sjældent forekommer, og at det, hvis det forekommer, uundgåeligt bliver opdaget, fordi videnskaben er effektiv, demokratisk og selvkorrigerende (Stroebe, Postmes og Spears, 2012, §677). 30 år efter stod det imidlertid klart, at den påstand ikke længere holdt vand. Videnskabelig uredelighed og tvivlsom forskningspraksis er virkelige og alvorlige fænomener, som videnskaben bliver nødt til at tage alvorligt, og som man kollektivt må finde metoder til at håndtere.

I Danmark nedsatte man allerede i 1998 tre stående udvalg, der kunne håndtere sager om uredelighed (BEK nr 933 af 15/12/1998). Udvalgene og de bekendtgørelser og lovtekster, der sætter rammerne for dem, har udviklet sig en del siden da, og udviklingen er et godt eksempel på, hvordan praksis, etik og jura gensidigt påvirker hinanden. Udvalgene blev i 2017 omdannet til et nævn, Nævnet for Videnskabelig Uredelighed (NVU), og definitionen af uredelighed blev ved samme lejlighed væsentligt revideret. I den nuværende lov beskriver man videnskabelig uredelighed som: 'Fabrikering, forfalskning og plagiering, som er begået forsætligt eller groft uagtsomt ved planlægning, gennemførelse eller rapportering af forskning' (*Lov om videnskabelig uredelighed m.v.* 2017, §3.1), hvilket svarer til den typiske internationale definition, hvor uredelighed netop forstås som fabrikering, forfalskning og plagiering (FFP). I den danske lov defineres de tre centrale begreber som:

Fabrikering: Uoplyst konstruktion af data eller substitution med fiktive data.

Forfalskning: Manipulation af forskningsmateriale, udstyr eller processer samt ændring eller udeladelse af data eller resultater, hvorved forskning fremstår misvisende.

Plagiering: Tilegnelse af andres ideer, processer, resultater, tekst eller særlige begreber uden retmæssig kreditering (*Lov om videnskabelig uredelighed m.v.* 2017, §3.2–3.4).

Loven skelner desuden skarpt mellem uredelighed og tvivlsom forskningspraksis, hvor det sidste defineres som: 'Brud på alment anerkendte standarder for ansvarlig forskningspraksis, herunder standarderne i den danske kodeks for integritet i forskning og andre gældende institutionelle, nationale og internationale praksisser og retningslinjer for integritet i forskning.'

NVU tager sig udelukkende af sager om uredelighed, mens sager om tvivlsom forskningspraksis bliver håndteret af universiteternes praksisudvalg. Det kodeks for integritet, der henvises til i lovteksten, er kodekset *Danish Code of Conduct for Research Integrity*, der blev ratificeret af alle danske universiteter i 2014. Dette kodeks betoner specielt, hvordan man som forsker skal forsøge at leve op til de tre grundlæggende værdier *ærlighed, transparens* og *ansvarlighed*. Så hvor loven om videnskabelig uredelighed beskriver de ting, man som forsker absolut *ikke* må gøre, beskriver *Danish Code of Conduct for Research Integrity* et ideal, man så vidt muligt skal stræbe efter. De to tekster beskriver med andre ord hver sin ende af et spektrum for, hvordan man som forsker bør opføre sig.

Endelig skal det bemærkes, at NVU eksplisit *ikke* behandler spørgsmål om forskningskvaliteten af et videnskabeligt produkt (*Lov om videnskabelig uredelighed m.v.* 2017, §3, stk. 2). Spørgsmålet om kvalitet er uddelegeret til peer-reviewsystemet og de mekanismer til organiseret skepsis, det videnskabelige samfund i øvrigt har etableret. Der er med andre ord forskel på sjusk og fusk, og det er kun det sidste, NVU tager sig af. 880

I en fremsynet artikel beskrev den amerikanske sociolog STEPHEN HILGARTNER i 1990 fire forskellige mulige tiltag, man kunne gøre mod uredelighed, nemlig:

1. **Law enforcement**, dvs. afskrækkelser gennem hurtig og hård straf.
2. **Oversight**, dvs. mere kontrol med forskeren, fx gennem skærpede krav om opbevaring af forsøgsprotokoller og rådata. 885
3. Mere **oplysning og uddannelse** om god videnskabelig praksis, så folk ikke uforvarende kommer til begå uredelighed.
4. **Revision af belønnings-** og meriteringssystemerne i forskningsverdenen, så de i højere grad opfordrer til god praksis (Hilgartner, 1990, §2). 890

De første tre af disse strategier er tydeligt blevet bragt i spil. NVU er således et klart udtryk for *law enforcement*, både universiteter og tidsskrifter stiller i dag meget højere krav om publicering og opbevaring af rådata og forsøgsdesign end tidligere (*oversight*), og kurser i „Fagets Videnskabsteori“ er en styrkelse af uddannelsen i god videnskabelig praksis. Hvad angår den fjerde strategi er det til gengæld et mere tvivlsomt om belønnings- og meriteringssystemet er blevet ændret, så de i højere grad opfordrer til god praksis. Specielt kan man diskutere, om overgangen til postakademiske forskning og det øgede fokus på forskningsproduktion hos de enkelte forskere fremmer god praksis eller det modsatte. 895

Vi er dermed tilbage ved spørgsmålet om systemisk etik og den rolle, de strukturer, vi befinner os i, spiller for vores moralske valg (afsnit 10.8). I hvilket omfang ligger ansvaret for uredelighed og tvivlsom forskningspraksis hos de individuelle forskere og i hvilken grad ligger ansvaret på de overordnede strukturer, man fra politisk og institutionelt hold har skabt til at bedrive forskning under? Mens loven om videnskabelig uredelighed udelukkende beskæftiger sig med forskernes individuelle ansvar, er det bemærkelsesværdigt, at *Danish Code of Conduct for Research Integrity* også tillægger forskningsinstitutionerne et vist (omend begrænset) ansvar. I forhold til publicering af forskningsresultater lyder det fx således (og der findes lignende paragraffer for andre aspekter af forskningsprocessen): 905

- v. Institutions should promote and maintain an environment that supports honesty, transparency, and accuracy when disseminating research findings, e.g. through policies and training relating to publication and communication. 910
- vi. Institutions should ensure that sponsors and other funders of research fully respect the duty of researchers to publish research and research results honestly, transparently, and accurately. (*Danish Code of Conduct for Research Integrity* 2014, §11)

Dermed er der åbnet for en diskussion, hvor hele ansvaret ikke pr. automatik placeres hos de individuelle forskere. 915

## God studiepraksis

Som studerende er man bl.a. underlagt regler om god studiepraksis, der i et vist omfang minder om de regler for uredelighed og god forskningspraksis, forskere er underlagt (ifølge 920 ordensreglerne på Københavns Universitet er man faktisk underlagt præcis de samme regler). Det er dog værd at bemærke, at der er nogle væsentlige forskelle mellem den situation, som hhv. forskere og studerende befinner sig i, specielt i forhold til plagiering og forfatterskab. Som bemærket ovenfor, er den officielle definition af plagiering faktisk ganske bred, og indeholder ikke bare tekstplagiering, men også plagiering af ideer, processer, resultater 925 og begreber. Selv om det kan være svært i praksis at bevise ideplagiering, er det bestemt noget, man tager alvorligt i forskningsverdenen; det er ikke velset at benytte andres ideer uden at kreditere dem.

For studerende ser situationen imidlertid lidt anderledes ud. En væsentlig del af meget undervisning består netop i at man tilegner sig og bruger ideer fra undervisere, undervisningsmateriale og medstuderende. Det ville ikke give mening, hvis man til eksamen skulle kreditere underviseren, hver gang man brugte en ide hun havde præsenteret til forelæsninger, eller hvis man skulle nævne sine medstuderende, hver gang man trak på en ide, de var kommet med i løbet af undervisningen. Overordnet set er de plagieringskrav, studerende er underlagt, dermed lempeligere end dem, der stilles til forskere. Der stilles dog højere 935 og højere krav til originalitet, jo længere man kommer på sin uddannelse, så dermed kan man sige, at universitetsstuderende er i en overgangsfase fra det simple plagieringsbegreb, men benytter i gymnasiet (hvor plagiering opfattes som synonym for tekstplagiering) til det mere omfattende plagieringsbegreb, man bruger i forskningsverdenen. Den slags overgange kan let skabe gråzoner og misforståelser, så derfor er det altid en god ide at spørge 940 sin underviser, hvis man er i tvivl om, hvordan plagieringsreglerne fortolkes på et specifikt kursus. Specielt skal det bemærkes, at der på nogle uddannelser er enkelte eksaminer, hvor de studerende under ingen omstændigheder må dele ideer med hinanden og hvor reglerne om ideplagiering håndhæves strengt.

Hvor plagieringsbegrebet generelt er mindre restriktivt for studerende end forskere, er 945 kravene til forfatterskaber til gengæld strengere. Som vi var inde på i forbindelse med undersøgelsen af tvivlsom forskningspraksis (Martinson, Anderson og Vries, 2005), er det ikke ualmindeligt, selvom det er uærligt, at forskere angiver sig selv eller andre som forfattere på videnskabelige artikler, de egentlig ikke har bidraget noget videre til. Kravene til forfatterskab fremgår af den såkaldte Vancouver-protokol, hvor det bl.a. fremgår at man skal have 950 ydet et 'substantial contributions to 1) conception and design, or analysis and interpretation of data; and to 2) drafting the article or revising it critically for important intellectual content; and on 3) final approval of the version to be published' („Uniform Requirements for Manuscripts Submitted to Biomedical Journals“ 1997, §38). Forbavsende nok, betragtes brud på reglerne om forfatterskaber i almindelighed ikke som uredelighed, men kommer 955 under reglerne for tvivlsom forskningspraksis, hvor de dog tilsyneladende håndhæves særlig lempeligt. Her er reglerne til de studerende til gengæld soleklare: Hvis man ikke har bidraget tilstrækkeligt til en gruppeopgave, må man ikke optræde som medforfatter. Det vil være eksamenssnyd.

Så der er altså visse forskelle på hvilke krav, der præcist bliver stillet til studerende og 960 forskere i forhold til redelighed. Det står dog også klart, at præcis som der er uklarheder og gråzoner for forskere, så er der også uklarheder og gråzoner for studerende, og derfor er det altid en god ide at snakke om reglerne og at spørge, hvis man er i tvivl.

Endelig skal det nævnes, at der kan opstå nye arbejdsformer, hvor ingen endnu har

besluttet, hvad standarderne for god praksis helt præcist er. Hvis jeg bruger en andens tekst er der nogle (nognelunde) klare standarder for, hvordan jeg skal citerer og henvise, men hvordan ser det ud, hvis jeg bruger en andens kode? I datalogien er det helt almindelig praksis, at folk deler og bruger hinandens kode via platforme som GitHub, men der har endnu ikke etableret sig klare og alment accepterede standarder for, hvor meget og hvordan man krediterer andres kode. Der er dermed en gråzone, hvor man let kan komme i klemme, hvor man let kan komme i klemme, hvis man ikke lever op til andres uudtalte forventninger om, hvordan der krediteres - eller hvis andre bruger ens egen kode uden at kreditere sådan som man forventede det. Derfor er det vigtigt, at man har en åben dialog, hvor spillereglerne for brug af andres kode fastlægges klart. 965

970

## Litteratur

- ACM (u.å.). *ACM Code of Ethics and Professional Conduct*. URL: <https://www.acm.org/code-of-ethics>. 975
- Adrian, Stine Willum (2016). „Subversive Practices of Sperm Donation. Globalising Danish Sperm“. I: *Critical Kinship Studies*. Red. af C. Krolykke m.fl. Bd. 1. Rowman & Littlefield International, s. 185–202.
- Angwin, Julia m.fl. (maj 2016). *Machine Bias*. URL: <https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing> (bes. 04.04.2018). 980
- Behrendt, M. (jul. 2014). *Big Data-værktøj rykker ind i gymnasierne og fortæller hvem der dropper ud*. URL: <https://www.version2.dk/artikel/big-data-værktøj-forudsiger-gymnasie-dropouts-med-imponerende-traefsikkerhed-59048> (bes. 11.01.2019). 985
- Bentham, Jeremy (2000). *An Introduction to the Principles of Morals and Legislation*. Batoche Books.
- Christiansen, F. V. (2015). „Rudimentær farmaceutisk etik“.
- Danish Code of Conduct for Research Integrity* (nov. 2014). København.
- Darley, John M. og C. Daniel Batson (1973). „From Jerusalem to Jericho‘. A Study of 990 Situational and Dispositional Variables in Helping Behavior“. *Journal of Personality and Social Psychology*, bd. 27, nr. 1, s. 100–108.
- Fanelli, Daniele (2009). „How Many Scientists Fabricate and Falsify Research? A Systematic Review and Meta-Analysis of Survey Data (How Many Falsify Research?)“ *PLoS ONE*, bd. 4, nr. 5, e5738. 995
- Hilgartner, Stephen (1990). „Research Fraud, Misconduct, and the IRB“. *IRB: Ethics & Human Research*, bd. 12, nr. 1, s. 1–4. DOI: 10.2307/3563681.
- History of Computing, IEEE Annals of the (u.å.). *IEEE Code of Ethics*. URL: <https://www.ieee.org/about/corporate/governance/p7-8.html>.
- Johnson, Deborah G. (2009). *Computer Ethics. United States Edition*. 4. udg. Upper Saddle River (NJ): Pearson Education. 1000
- Kant, Immanuel (1996). „On a Supposed Right to Lie from Philanthropy“. I: *Practical Philosophy*. Red. af Mary J. Gregor. Cambridge University Press, s. 605–615.
- (1999). *Grundlæggelse af sædernes metafysik*. København: Hans Reizels Forlag.
- Kleinberg, Jon, Sendhil Mullainathan og Manish Raghavan (2017). „Inherent Trade-Offs in 1005 the Fair Determination of Risk Scores“. I: *8th Innovations in Theoretical Computer Science Conference (ITCS 2017)*. Red. af Christos H. Papadimitriou. Leibniz International

- Proceedings in Informatics (LIPIcs) 67. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 43:1–43:23. DOI: 10.4230/LIPIcs.ITCS.2017.43.
- 1010 Lazer, D. m.fl. (2014). „The Parable of Google Flu. Traps in Big Data Analysis“. *Science*, bd. 343, nr. 6176, s. 1203–1205. DOI: 10.1126/science.1248506.
- Lov om videnskabelig uredelelighed m.v.* (Apr. 2017).
- Martinson, Brian C., Melissa S. Anderson og Raymond de Vries (2005). „Scientists behaving badly“. *Nature*, bd. 435, s. 737–738. DOI: 10.1038/435737a.
- 1015 Maxmen, Amy (okt. 2018). „A moral map for AI cars“. *Nature*, bd. 562, s. 469–470. DOI: 10.1038/d41586-018-07135-0.
- Mittelstadt, Brent Daniel og Luciano Floridi (2016). „The Ethics of Big Data. Current and Foreseeable Issues in Biomedical Contexts“. *Science and Engineering Ethics*, bd. 22, nr. 2, s. 303–341. DOI: 10.1007/s11948-015-9652-2.
- 1020 Møllerhøj, Jakob (aug. 2015). *It-system til varsel af elevfrafald blev øjeblikkeligt standset af gymnasierne*. URL: <https://www.version2.dk/artikel/noejagtig-frafalds-algoritme-blankt-afvist-i-udbredt-gymnasie-it-320922> (bes. 09.07.2019).
- Nagel, Thomas (1979). *Limits of objectivity. Tanner Lectures on human values*. URL: [https://tannerlectures.utah.edu/\\_documents/a-to-z/n/nagel80.pdf](https://tannerlectures.utah.edu/_documents/a-to-z/n/nagel80.pdf).
- 1025 — (1986). *The view from nowhere*. Oxford: Oxford University Press.
- Nakashima, E. (nov. 2007). *Feeling Betrayed. Facebook Users Force Site to Honor Their Privacy*. URL: <https://www.washingtonpost.com/wp-dyn/content/article/2007/11/29/AR2007112902503.html>.
- 1030 Pascual-Ezama, David m.fl. (aug. 2015). „Context-dependent cheating. Experimental evidence from 16 countries“. *Journal of Economic Behavior & Organization*, bd. 116, s. 379–386. DOI: 10.1016/j.jebo.2015.04.020.
- PROSA (2000). *God Databehandlingsskik: En etisk guide for professionelle IT-folk*. København: PROSA.
- Rawls, John. (1972). *A theory of justice*. Oxford: Clarendon Press.
- 1035 Reverby, S. (2011). „Normal Exposure and Inoculation Syphilis. A PHS 'Tuskegee' Doctor in Guatemala, 1946–1948“. *Journal of Policy History*, bd. 23, nr. 1, s. 6–28.
- Ryberg, Jesper (2003). „Ethics and Military Research“. I: *Mathematics and War*. Red. af Bernhelm Booß-Bavnbek og Jens Høyrup. Basel, Boston og Berlin: Birkhäuser Verlag, s. 352–364.
- 1040 Stroebe, Wolfgang, Tom Postmes og Russell Spears (2012). „Scientific Misconduct and the Myth of Self-Correction in Science“. *Perspectives on Psychological Science*, bd. 7, nr. 6, s. 670–688. DOI: 10.1177/1745691612460687.
- Swartz, Aaron (2020). *Guerilla Open Access Manifesto*. URL: <https://archive.org/details/GuerillaOpenAccessManifesto> (bes. 26.05.2020).
- 1045 „Uniform Requirements for Manuscripts Submitted to Biomedical Journals“ (1997). *JAMA: The Journal of the American Medical Association*, bd. 277, nr. 11, s. 927–934.
- Youyou, W., M. Kosinski og D. Stillwell (2015). „Computer-based personality judgments are more accurate than those made by humans“. *Proceedings of the National Academy of Sciences of the United States of America*, bd. 112, nr. 4, s. 1036–1040.
- 1050 Zuboff, Shoshana (2019). *The age of surveillance capitalism. The fight for a human future at the new frontier of power*. London: Profile Books.