31-10-2005

Compiler-gruppen: Teknisk perfektionisme kontra nytte

Af Søren Lauesen

Trykt i Dansk Datahistorisk Forenings festskrift, 2005: Regnecentralen - dansk institut for matematikmaskiner.
Gengivet med tilladelse af redaktørerne Henning Isaksson og Ole Pedersen

Hvordan kunne det gå til at Regnecentralen med sine opfindsomme medarbejdere og elegante tekniske løsninger aldrig blev en rigtig succes? Det har jeg tænkt meget over i de forløbne år, og jeg vil i dette indlæg fortælle hvordan det ser ud i mit eget bakspejl.

Min første tid på RC

I 1960 blev jeg student og begyndte på matematik-fysik studiet ved Københavns Universitet. På Matematisk Instituts bibliotek studerede jeg alt jeg kunne finde om computere. Det mest fantastiske var en lærebog i kodning for DASK (Chr. Andersen, 1958) med en benhård forklaring af maskinkoden. Jeg studerede den intenst og skrev med blyant et par trivielle programmer uden at forestille mig at det nogensinde kunne blive alvor.

På universitetet mødte jeg året efter Bjarner Svejgaard som fortalte om den nye elektronhjerne, GIER. Kort efter troppede jeg op på RC og spurgte om jeg kunne få lærebogen i kodning for GIER, og det kunne jeg. Den var endnu mere spændende at studere, og jeg kunne hurtigt alle ordrerne udenad, selvom de godt nok var meget mere indviklede. Med den ballast søgte jeg et timelønnet job som koder i Finn Larsens gruppe og startede 9. juli 1962 i en alder af 19 år.

Den sommer lærte jeg at finde rundt mellem villaerne på Gl. Carlsberg Vej og Bjerregårds Vej. Der var snedige huller i plankeværkerne og en have som man helst ikke skulle skrå igennem fordi damen der boede der blev vred. Den dag jeg startede blev jeg præsenteret for Bech ved sådan et plankeværkshul. Han bød mig velkommen og sagde at jeg nok skulle blive til noget stort, hvilket undrede mig, for hvor pokker kunne han vide det fra?

Finn Larsen fortalte at assembler-kode var for besværligt, Algol var meget lettere. Så jeg fik udleveret Algol 60 manualen og gik straks i gang med at læse og skrive noget der skulle være et program. Så skulle jeg lære at hulle det på flexowriteren. Det var næsten værre end at skrive det med blyant. Og endelig tog Finn mig ved hånden og kørte det på DASK. Der kom en masse fejludskrifter som jeg ikke forstod meget af, især når jeg nu havde gjort mig sådan en umage. Finn kiggede i mit program og forklarede hvad der var galt. Han tilføjede at det da var et underligt program med alle de labels og switches (en switch var en slags case-statement der vælger mellem en række labels i programmet). Sådan skrev man ikke et program. Det viste sig at jeg havde læst Algol 60 manualen forfra og prøvet at bruge sprogets dele efterhånden som de blev forklaret. Og bortset fra en masse indledende snak, jeg ikke forstod, og en masse om beregningsudtryk, som var ret trivielt, så startede bogen med labels og switches. Først flere år senere udbredte Dijsktra det glade budskab om at labels og goto var fandens opfindelse.

Jeg tror ikke der gik mere end en uge førend jeg fik min første rigtige opgave. Piet Hein havde i 1945 opfundet et spil, og han vidste ikke hvordan man skulle spille det for at vinde.

Det måtte være noget som en elektronhjerne kunne finde ud af, så han henvendte sig til RC. Hvem havde ikke noget at lave? Jo, ham den nye, dvs. mig.

Spillet var en to-dimensional udgave af det klassiske NIM spil. De to spillere skiftes til at tage et antal nabo-brikker, og den der tager den sidste brik har tabt. I det klassiske NIM spil lå brikkerne (som regel tændstikker) i flere rækker, og man måtte tage så mange brikker man ville fra én enkelt række. I Piet Heins udgave lå brikkerne på linier der krydsede hinanden, og man måtte tage så mange brikker man ville fra én enkelt linie. Hver enkelt brik indgik i flere linier.

Hvordan skulle man finde det rigtige træk i en given situation? I princippet kunne man lade computeren prøve alle træk, og for hvert af dem prøve alle træk igen, osv. for at finde vejen til gevinst. Selv om der kun var 12 brikker i Piet Heins spil, ville der være uoverkommeligt mange muligheder - selv for en computer. Jeg grublede flere dage over problemet, og på en tur med Storebæltsfærgen, gik løsningen op for mig. Jeg beviste først matematisk at man for alle NIM-agtige spil kunne dele spillets situationer i vinder- og tabersituationer. Computeren kunne starte bagfra med de slutsituationer hvor spiller A klart ville vinde fordi der kun var én brik igen til spiller B. Det var vindersituationer. Så kunne computeren regne baglæns og finde de situationer hvorfra man med ét træk kunne komme til en vindersituation. De situationer måtte være tabersituationer. Situationer hvor man kun kunne komme til tabersituationer måtte være vindersituationer. Da Piet Heins spil kun havde 12 brikker, var der kun 4096 mulige situationer. Ved at udnytte at brikkerne lå i et tre-symmetrisk mønster, kunne man reducere antallet af vindersituationer til noget der let kunne være i lageret, der var på 1024 ord (ca. 5 k bytes). Resten var programmering. (I dag ville jeg have lagret alle de 4096 situationer som 4096 bit, fx i 128 ord).

I dag er denne opgave triviel for dem der beskæftiger sig med spilteori, men det var den ikke den gang. Og jeg var helt alene om at finde løsningen.

Programmet blev lavet i assembler til GIER, og det var det første rigtige program jeg lavede. Jeg husker stadig det umådelige chok da maskinen accepterede programmet og startede det, hvorefter der skete absolut ingenting. Det tog lang tid at vænne sig til at ens tanker kan være rigtige i princippet, men de er altid forkerte i detaljen - i hvert fald i de første mange forsøg. Og computeren forstår overhovedet ikke principper, men kun detaljer.

Piet Hein og RC blev enige om at vise spillet på en udstilling hvor de besøgende kunne spille mod computeren. Vi lavede et bræt med lysende knapper svarende til brikkerne. Desuden var der to knapper med teksterne *Svar* og *Nyt spil*. Spilleren lavede et træk ved at trykke på de knapper han ville tage. Maskinen lod dem blinke indtil spilleren trykkede på *Svar*. Maskinen slukkede så knapperne og ventede et passende stykke tid som om den tænkte (Piet Hein havde lagt meget vægt på at det skulle se sådan ud). Så blinkede den med de knapper den ville tage og slukkede dem. Hardware-mæssigt havde det været svært, for input/output var computernes svage led. Faktisk var det her projektet gik langsomt. Vi endte med at løse problemet ved at koble knapperne direkte på computerens M-register (multiplikator-registret). Mit program anbragte så ettaller der hvor knapperne skulle være tændt, og aflæste ændringer i M-registret svarende til de knapper som spilleren trykkede på. Og så skulle jeg selvfølgelig undgå multiplikationer i programmet.

Samarbejdet med Piet Hein strakte sig over det meste af et år, hvor vi også mødtes privat nogle gange. Han var et fascinerende menneske, og han så en mission i at give mig en

balanceret forståelse af verden. Det var ikke teknik alt sammen, men en balance mellem kultur og teknik. Den unge mand på 20 år trængte i høj grad til den forståelse, men forstod det kun delvis. Da RC begyndte at snakke om betaling for arbejdet, blev der problemer. Piet Hein mente at en kulturel indsats som hans, skulle han ikke betale for. Snarere omvendt. Det var heldigvis ikke mit bord, og jeg ved ikke hvordan det endte.

Mine andre opgaver i disse første år var at lave røntgen-krystallografiske beregninger, optimering af sukkerroe transport i Polen, løsning af astronomiens 3-legeme-problem, optimering af sporvejsdrift, og sågar et enkelt program til ATP. Det var alt sammen noget med at prøve sig frem. Der var ingen gode metoder eller retningslinier, og man udvekslede kun i meget begrænset omfang erfaringer. Til gengæld var det også nogle rodede programmer jeg leverede, selvom de fungerede korrekt.

Compiler-gruppen
I en alder af 22 år fik jeg min kandidatgrad (1965), selvom jeg havde brugt det meste at tiden på at arbejde hos RC. Jeg følte luften under vingerne og bad om at blive overflyttet til den mest prestige-fyldte afdeling, compiler-gruppen. Forinden havde jeg sat mig grundigt ind i Algol compilerens opbygning.

Det var fantastisk at komme ind i compiler-gruppen. Man sad ikke alene og nørklede, men alle fulgte med i hvad de andre gjorde. Peter Naur og Jørn Jensen var de ubestridte guruer, og de havde fastlagt gode principper for hvordan man testede, dokumenterede koden, osv.

Min første opgave var at dokumentere assembler-koden til passage 7 af Gier Algol 3 compileren, som nylig var blevet frigivet. Det geniale princip i compileren var at den bestod af 8 passager, som hver læste hele kildeprogrammet igennem og leverede det videre til næste passage i en mere hensigtsmæssig form. På den måde kunne man oversætte selv meget store programmer lynhurtigt, selvom maskinen kun havde 5 k memory, som det ville hedde i dag. Passage 6 havde omformet alle udtryk til omvendt polsk form og kontrolleret at operander typemæssigt passede med operatorerne. Passage 7 dannede nu en forkortet form af maskinkoden ud fra passage 6's resultater. Endelig udvidede passage 8 den forkortede maskinkode til den fulde form. Naur havde skrevet passage 6, og koden var eksemplarisk kommenteret. Passage 7, derimod, var skrevet af Jørn under tidspres, og den var stort set helt uden kommentarer. (Det er jo ikke altid en guru tager sin egen medicin). Når man betænker at der ikke var symbolske adresser i assembleren, men kun labels af formen a1, a2 ... k1, k2 ..., så er det utroligt at man overhovedet kunne forstå hvad der skete. Nå, men det lykkedes mig at kommentere alle 700 kodelinier til alles tilfredshed. Set i bakspejlet var det en ganske glimrende svendeprøve.

En fantastisk ting ved compileren var, at den til forskel fra datidige compilere fortsatte med at oversætte når den havde fundet en fejl i kildeprogrammet. Men som menig bruger af den havde jeg været irriteret over dens fejludskrifter. Hver af passagerne udskrev nemlig de fejl den selv fandt. Udskriften bestod af et linienummer i kildeteksten og en meddelelse om fejlens art. Når man oversatte et større program, kunne der godt være 100 fejl som man skulle finde frem til i kildeteksten. Først bladede man hele teksten igennem for at lokalisere de fejl som passage 1 havde fundet, så en gang til for at finde passage 2's fejl, osv.

Jeg spurgte Jørn hvorfor hver passage ikke bare sendte fejlangivelserne videre til næste passage. Så kunne passage 6 udskrive dem alle sammen i linienummer-orden. Så sparede man også lidt plads til at generere udskrifter i hver af de første passager. Jørn kiggede

overrasket på mig og bragte ideen videre til Naur. Næste gang jeg så Naur var han meget rosende og fortalte mig at det var den slags ideer man satte højt. *Enkle, nyttige og robuste* (under *robust* hører bl.a. at løsningen også kan klare usædvanlige situationer). Disse principper blev et ideal der ledte mig i mange år fremover, indtil det gik op for mig at det var en frygtelig grøft man var havnet i. Mere om det nedenfor.

I de følgende år kom jeg til at deltage i mange af compiler-gruppens projekter. Algol 4 til Gier, Algol 5 og 6 til RC 4000, og operativsystemerne til RC 4000. Der var masser af udfordringer og masser af succeser - i hvert fald målt med vores egen målestok.

På et tidligt tidspunkt fik Bech og Naur den idé at jeg skulle dele kontor med Naur, en ære der var uhørt på RC. Det blev en meget lærerig periode. Jeg kom fra en ganske almindelig familie hvor kun en moster var blevet student, og slet ingen vidste hvad en akademisk uddannelse var. Så der var mange ting på mange niveauer jeg skulle lære.

En af dem var at tænke lige som Naur. Det foregik på en skrivemaskine, hvor Naurs tanker kom lige så smukt og velstruktureret ud af hans hænder og ned på papiret. De få ting der skulle ændres, klarede han bagefter med saks og klisterbånd. Jeg prøvede ihærdigt i over et år, men mine tanker var åbenbart af en anden art, for de kom altid hulter til bulter og ville slet ikke rette sig ind. Til sidst opgav jeg og gjorde som jeg plejede, først tegninger på tavlen, så en disposition med mange udviskninger og streger, så et skitsemæssigt manuskript, og så renskrift. Men jeg var alligevel blevet meget bedre til at skrive under Naurs hånd, især formulere mig kort og præcist. Med tekstbehandling kan jeg i dag næsten gøre som Naur: få tankerne pænt ned på skærmen i første hug (næsten).

Jørn og Naur prægede også min åndelige udvikling på mange andre områder. Vi kunne diskutere alverdens ting indenfor naturvidenskab og filosofi, og de anbefalede mig at læse Scientific American, hvilket jeg har gjort siden.

Jørn og jeg var dog mest optaget af computeren, og vi vidste i detaljer hvad der foregik helt inde i den. Her er et lille eksempel. RC var involveret i alle folketingsvalg. RC registrerede alle valgtallene efterhånden som de ankom til Indenrigsministeriet, og computeren talte op og beregnede prognoser. Resultaterne gik direkte på skærmen, mere og mere elektronisk efterhånden som årene gik.

Ved folketingsvalget i 1966 gik det galt. Valgstudiet var sat op i Falkonércentret, og Uffe Ellemann Jensen var TV-journalist. Under en af prøverne interviewede han en stumtjener i garderoben, og hver gang han havde stillet "politikeren" et spørgsmål, trådte han over på "politikerens" plads og besvarede spørgsmålet fuldkommen som denne politiker ville have gjort. Alle i studiet lå flade af grin under disse seancer.

Nå, til sagen. Compiler-gruppen havde ikke lavet valgprogrammerne. De var lavet i Algol af andre, men Jørn og jeg var med ved Gier computeren i studiet, hvis nu noget skulle gå galt.

Og det gjorde det. Da de sidste valgtal var blevet registreret, skulle computeren straks skrive landsresultatet ud, men intet skete. TV-producerne blev nervøse og måtte holde seerne hen, og de der kendte valgprogrammerne gik i gang med at finde fejlen. Christian Gram sørgede for at alle fik arbejdsro, og efter en halv time var fejlen fundet. En rettelse foretaget i sidste øjeblik var ikke blevet testet ordentligt. Programmets centrale løkke blev afsluttet når antallet af resterende valgkredse, n, var nede på 0, og så kom valgresultatet. Men rettelsen havde

bevirket at når sidste kredsresultat kom, blev n ikke talt ned fra 1 til 0. Programmet kørte derfor rundt i en evig løkke.

I princippet var det let at klare. Programmet skulle bare rettes lidt og oversættes. Men det ville desværre slette alle kredsresultaterne, så de måtte indlæses igen. Skønsmæssigt ville det tage et par timer og det kunne let gå galt. Jørn og jeg blev sat ind i sagen. Vi konfererede kort med hinanden og blev enige om at vi kunne rette n direkte i lageret mens programmet kørte. Med vores kendskab til compileren kunne vi regne ud hvilken lagercelle n lå i. Vi satte computeren på pause og sammen med Knud Bruun fra Præstø, kiggede vi via teknikerpanelet på celle 853 (eller hvad det nu var). Der skulle n være. Ganske rigtigt, der var et ettal. Vi kontrollerede lige nabocellerne for at se at det var det rigtige ettal vi havde fundet. Okay. Så ændrede vi cellen til nul. Det var ikke et almindeligt heltalsnul, men et flydende nul, så det skulle gøres med omhu. Det hele tog vel højst 10 minutter. En sidste kontrol - og med stor spænding trykkede vi på *kør*. Jubi! ud kom landsresultatet.

Elfenbenstårnet

Som bekendt var RC ikke nogen økonomisk succes, og firmaet måtte have livredning adskillige gange. På RC trivedes adskillige konspirationsteorier om IBM's infiltrering af staten, regeringens svigt af dansk forskning, osv. Det har Per V. Klüver skrevet indsigtsfuldt om i IEEE Annals of the History of Computing (Vol. 21, No. 2, 1999). Jeg var jo bare en stor dreng og var ikke blandet direkte ind i alt dette, men hørte om det. Det var jo uforståeligt, for vi - især compiler-gruppen - var jo fantastiske, og ingen i verden kunne lære os noget. Så hvorfor blev vi svigtet?

Først meget sent begyndte det at dæmre for mig at RC måske ikke havde de rigtige produkter. Compiler-gruppen sad i sit elfenbenstårn og forstod ikke hvad brugere og kunder havde brug for. Og selv om Bech havde endda meget store visioner om samfundsomspændende databehandling, formåede ingen at styre de kreative kræfter i den retning. Hans signaler til compiler-gruppen var at vi bare skulle lave noget genialt, så skulle han nok finde nogen der kunne sælge det.

En af de første gange det dæmrede for mig var i begyndelsen af 1970. En potentiel amerikansk kunde havde henvendt sig fordi han var interesseret i at købe RC 4000, som han på mange måder syntes om - ikke mindst på grund af prisen. Der var dog en ting han savnede - et indeks-sekventielt filsystem. Sådan et havde vi ikke. Brinch Hansens dominerende indflydelse på RC 4000 havde fortonet sig, og Bech indkaldte derfor Naur, Jørn og mig selv til et møde med kunden. Han satte mødet i gang og overlod os til os selv. Hverken Jørn eller Naur havde erfaringer med et indeks-sekventielt filsystem, mens jeg dog nogenlunde vidste hvad det gik ud på, fordi jeg havde opbygget et kursus i administrative systemer som et led i den nye datalogi-uddannelse på Københavns Universitet og derfor havde læst litteraturen.

Kunden forklarede kort hvad sådan et filsystem gik ud på og hvorfor han havde brug for det. Til læserens orientering er ideen at man lagrer alle records i en fil ordnet efter deres sorteringsnøgle. Man anbringer huller passende steder så det ofte er muligt at skubbe nye records ind det rigtige sted, uden at skulle skubbe alle efterfølgende records. Det svarer til at man på et bibliotek ikke stiller bøgerne tæt på hylderne, men lader der være lidt tom plads sidst på hver hylde. For hurtigt at kunne finde en bestemt record, har man et indeks der giver sorteringsnøglen for første record i hver af filens hovedafsnit (cylindre på disken). I starten af hver cylinder er der også et indeks der peger på den første record i hvert segment af cylinderen. På den måde kan computeren ved én flytning af diskhovederne til den rigtige

cylinder og to læsninger af et disksegment, få fat i den rette record selvom filen er umådelig lang.

Problemet kommer når man indsætter en record og der ikke er et tilstrækkeligt stort hul i nærheden. Så kan man fx placere den overskydende record sammen med andre overskydende records et helt andet sted. Men nu kan man ikke længere være sikker på at man kan finde recorden med kun én hovedflytning. Når filen bruges i lang tid, kan det blive så rodet at man må få computeren til at omstrukturere det hele.

Nå, hvordan reagerede guruerne på det? Naur så først opgivende ud over denne amatøragtige fremgangsmåde. Med slet skjult arrogance spurgte han kunden *om han havde overvejet at bruge rekursive procedurer* i stedet. Denne akademiske disciplin var kunden ikke bekendt med, og inden det blev alt for pinligt, greb jeg ind og forklarede at rekursive procedurer af flere grunde var totalt uegnede til denne problematik, og at kundens ønske var vigtigt for de fleste administrative systemer. Jeg antydede også at vi ville overveje at lave et sådant filsystem.

Senere var Naur stadig afvisende, mens Jørn efter mange overvejelser tog udfordringen op og endte med at lave et nydeligt og veldokumenteret system.

Hvorfor var der sådan en modvilje mod at implementere en velkendt løsning, der var uhyre nyttig i disk-baserede, administrative systemer? Der var selvfølgelig noget *not-invented-here* over modstanden, men det stak dybere. Husk på vores uskrevne idealer: *enkelt, robust og nyttigt*. Dette var i høj grad nyttigt, men ikke enkelt. Det værste var at det slet ikke var robust.

Ordet *robust* var ikke så præcist defineret, men vi mente noget med at ingen fejl eller uventede situationer måtte få systemet til at gå ned. Intuitivt gik vi videre og mente at det heller ikke måtte forringe systemets hastighed. Der måtte heller ikke være modstrid mellem hurtighed og pladsbesparelse. Kunne vi ikke opnå begge dele var det en afvejning som vi ikke turde lave. Et indeks-sekventielt filsystem krævede sådan en afvejning, og tilmed ville det gradvis blive langsommere og langsommere - uacceptabelt for purister som os.

Et lidt tidligere eksempel er compiler-gruppens arbejde med at lave en mere generel håndtering af input/output enheder til RC 4000, så det hele ikke skulle bygges ind i operativsystemet, inklusive strategier for at bruge multiple buffere og håndtere tekniske fejl på disse sårbare enheder. Sådanne strategier var hverken enkle eller robuste. Det skulle derfor være muligt for programmøren at styre det selv. Inspireret af Naurs beretninger om Dijkstras arbejde med samarbejdende parallelle processer, designede vi en avanceret mekanisme i Algol til håndtering af parallelle processer. De blev kaldt *zoner* efter vores daværende russiske gæsts forslag. Zoner kunne man også bruge til at behandle magnetbånd, mv. - hvis man ellers var dygtig nok.

Kort tid efter at denne løsning var annonceret rundt om på RC, mødte min gamle chef, Finn Larsen, op hos os. Han og hans gruppe programmerede administrative anvendelser og han havde med bekymring set vores forslag. *Det kan godt være at I har løst et problem for jer selv, men I har gjort det meget sværere for os der skal bruge det*, sagde han. Jeg tror der blev meget tavst i vores gruppe - selv jeg var vist mundlam. Det tog lang tid at fordøje det, for egentlig havde han jo ret. Jeg tror løsningen blev at vi indførte en standardbehandling af

input/output enhederne som den "avancerede" programmør kunne koble fra. Men vi løste ikke Finn's egentlige problem, fordi vi ikke forstod det.

Administrativ databehandling
Generelt havde gruppen en uvilje mod at beskæftige sig med administrativ databehandling, selvom det nok var den største indtægtskilde for RC. En del af gruppen havde tidligt med succes brugt Algol-principperne til at lave en Cobol-compiler til Siemens, men ingen ville høre tale om at vi skulle have Cobol til egne maskiner. Man hævdede, vistnok især Naur, at Algol udmærket kunne bruges også til administrativ databehandling.

RC's servicecentre havde i lang tid udviklet administrative systemer som man kørte for kunder på servicebasis. Karakteristisk nok ved jeg dårligt om de blev skrevet i Algol eller assembler, og compiler-gruppen gjorde sig mig bekendt aldrig den anstrengelse at sætte sig ind i hvad servicecentrenes behov kunne være. Når vi en gang imellem så lidt af et program fra servicecentrene, sad vi og godtede os over deres primitive programmeringsstil, fx kontrol af om et varenummer var tastet rigtigt:

```
if c=20 or c=25 or c=32 or c=34 then . . .
```

Næ, var man rigtig datalog, så lavede man en tabel over alle tegn, der straks kunne give tegnets type:

```
if type[c] = forbogstav then . . .
```

Her havde vi en enkel, robust og nyttig løsning. Vi glemte ganske vist hvor enormt besværligt det var at opbygge sådan en tabel, hvilket Algol ikke gav støtte til. Det var sådanne detaljer vi kiggede på - ikke på de store behov for at håndtere fx record-strukturer og databaser.

Et af servicecentrene var baseret på en CDC 1604, som jeg selv arbejdede på et par år inden jeg kom i compiler-gruppen. Maskinen var anskaffet fordi Gier ikke kunne levere den nødvendige kapacitet til administrative opgaver. Planen havde været at bruge CDC's Algol til opgaverne, men compileren var ubrugelig, og man endte med at bruge Fortran, som var langt mere egnet. Cobol var stadig et fy-ord. Hvordan havde man dog kunnet forestille sig at Gier kunne løse de opgaver som Stat og Kommuner havde?

På et sent tidspunkt begyndte Paul Lindgreen, der kom fra compiler-gruppen, sammen med Bjørn Ørding Thomsen at udvikle bedre Algol-baserede værktøjer til administrative systemer. Jeg ved ikke i hvor høj grad de fik succes, men de fik i hvert fald ikke moralsk støtte fra resten af compiler-gruppen.

Når vi nu hævdede at Algol var egnet til administrative systemer, hvad gjorde vi så for at håndtere variable af typen *text*? Denne datatype findes ikke i Algol, men det er den allervigtigste type data i administrative systemer. I Cobol findes der grundlæggende ikke andre typer.

Vi gjorde stort set ingenting! Der fandtes ikke engang typen *char*, men der var da en procedure der kunne indlæse et tegn og lagre det som et heltal, og en anden der kunne udskrive et heltal som et tegn. Ellers var tekster kun tekstkonstanter; sådan nogle man fx brugte til at skrive overskrifter på matematiske tabeller.

Og hvorfor havde vores Algol ingen tekstvariable? Dels fordi det ikke var en del af Algol 60 standarden, som Naur havde været dybt involveret i, men nok lige så meget fordi vi ikke kunne finde en enkel og robust måde at håndtere det på. Vores maskiner var bygget til talbehandling. I GIER kunne man adressere et ord som var på ca. 40 bit, i RC 4000 et ord på 24 bit. I administrative anvendelser er tekster det der fylder mest i lageret, og lagerplads var et evigt problem, både i centrallageret og på disken. Man var derfor nødt til at pakke flere tegn sammen i ét ord, men så kunne man ikke behandle en tekst som et array af tegn. Eller rettere sagt, man kunne godt, men det ville ikke være en enkel og robust løsning. Vores Algol tilbød ikke engang standardprocedurer til at pakke tegn ind og ud, sådan som det fra tidernes morgen har været muligt i C. Det var noget folk selv måtte programmere.

Renhed frem for nytte

I dag kan jeg tydeligt se mønstret i hvad vi gjorde. Vi elskede at finde "rene" løsninger, enkle og robuste. Kunne vi ikke det, opgav vi at løse problemet, selv hvis nytten ville være stor. Vi overlod løsningen til vores brugere, de der programmerede anvendelserne som vi levede af. Og vi kunne endda finde på at se hånligt på deres "snavsede" løsninger på de problemer som vi selv havde opgivet.

Hvad med vores højt besungne operativsystem, *monitoren* til RC 4000? Brinch Hansen gjorde det berømt, og jeg selv var en af hovedmændene bagved. Det stod såmænd ikke bedre til. I lang tid diskuterede vi operativsystemets strategi for lager- og tidsdeling, men kunne selvfølgelig ikke finde en enkel og robust løsning. Jeg foreslog derfor en mekanisme hvor operativsystemet kunne uddelegere ansvaret til underordnede operativsystemer, der så kunne implementere deres egen strategi. Det faldt øjeblikkeligt i god jord og blev det bærende princip. Endnu engang havde vi undgået at løse de egentlige problemer, men givet dem videre til andre.

En anden gruppe blev sat til at lave et sådant underordnet operativsystem, Boss 1, men det mislykkedes. Kort tid efter fik jeg til opgave at forsøge en gang til. Det blev til Boss 2, som tog enormt lang tid at lave. Det viste sig hurtigt at ideen med at kunne lave et underordnet operativsystem nok var god i princippet, men der manglede en masse mekanismer i det overordnede operativsystem, for at det kunne lykkes. Der måtte forhandles en løsning med Brinch og de andre der lavede monitoren, og som nødigt så deres enkle, robuste løsninger saboteret. Det lykkedes i nogen grad.

Og så blev det underordnede operativsystem vel godt? Næ, det lod vi som om, men det var ikke særlig nyttigt til moderne administrative opgaver, hvor data kom ind via terminaler. Vi havde ikke forstået behovene. Jeg kan se i dag, at vi troede at alle brugte computerne til det samme som vi: Man rettede i nogle tekstfiler og fik dem derefter oversat eller behandlet på anden måde. Derfor havde Boss 2 en indbygget kommando-baseret teksteditor der kunne betjene mange brugere samtidig. Når de så skulle have deres tekster behandlet, skete det med en slags batch-behandling, der dog kunne køre nogle få jobs parallelt. Det var bestemt ikke egnet til datidens moderne databehandling og ville ikke have været konkurrencedygtigt.

Hvis vi havde forstået behovet i databehandling, ville vi have opdaget at der ikke var brug for tekstediteringen, men for udskrift og indtastning via skemaer. Vi kunne så have lavet en mange-bruger editor til det. Den efterfølgende behandling af denne transaktion kunne godt være sket med vores form for batch. Havde vi gjort det, kunne vi have været et par år før IBM's 3270 terminal, som kom midt 1971. Vi kunne have gjort i software, hvad den gjorde i

hardware. Den kom til at præge markedet i mange år fremover, og den emuleres stadig på PC og UNIX.

Det kan være lærerigt at se på hvordan en meget succesrig virksomhed som Navision, nu Microsoft Business Solutions, blev skabt. På det tidspunkt da grundlæggerne lavede det første produkt, var der opstået en masse små administrative systemer som konkurrerede indædt om markedet. Navision folkene skaffede sig et eksemplar af hvert af dem og studerede detaljeret deres styrker (og i mindre grad deres svagheder). Desuden allierede de sig med en revisor der havde god forstand på alle de administrative arbejdsopgaver i virksomheder. Og så byggede de et system der kunne alt det bedste fra de andre, og som effektivt støttede de opgaver som revisoren var ekspert i. De gjorde sig stor umage for at lave systemet rimeligt enkelt og robust, men gik ikke af vejen for at tilføje specielle mekanismer som var særligt nyttige. Kort sagt, de gjorde meget af det som vi også forsøgte i compiler-gruppen, men var mere pragmatiske, og først og fremmest drevet af en indgående forståelse af kundernes behov.

Kunne vi have gjort det anderledes?
Set i bakspejlet burde vi have handlet anderledes, men det ville have krævet en indsigt vi ikke havde. Finn Larsen og Peer Lorentzen fortæller at vi ikke var det eneste elfenbenstårn. RC var fuld af dem, fx flere salgs- og servicecentre.

Hele RC var baseret på den idé at hvis en masse dygtige folk alle sammen gør det de synes er rigtigt, så bliver det samlede resultat godt. Nej, det gør det ikke, for der kommer slet ikke noget samlet resultat, kun en masse isolerede øer, der let kan komme til at bekrige hinanden.

I compiler-gruppens tilfælde skulle ledelsen have gjort det klart for os, at vores kunder både var computer-markedet og servicecentrene. Vi skulle sætte os ind i deres behov og finde på geniale løsninger der støttede disse behov.

Hvad vi bragte videre ud i verden
Vi kan godt lide at tænke på at vi var pionerer også i arbejdet med at skabe universitetsuddannelser på IT-området. Ja, det gjorde vi på mange måder godt, men vi bragte altså også smitten videre.

Naur, H.B. Hansen og jeg selv grundlagde datalogiuddannelsen på Københavns Universitet i 1969. Naur var blevet professor der, og H.B. Hansen blev lidt senere professor på RUC. Jeg selv forblev fuldtidsansat på RC mens jeg som ekstern lektor opbyggede og underviste i flere fag. Jeg underviste også i et fag der handlede om administrativ databehandling, men det var tydeligt at jeg helt manglede praktisk erfaring med at lave sådan nogle systemer.

Jeg var stadig besat af idealet om det enkle og robuste, som i hænderne på folk der havde endnu mindre praktisk erfaring end jeg, hurtigt blev til den rene datalogi. Man definerede sin egen verden og fandt rene, robuste, og gerne matematiske løsninger på sine selv-definerede problemer, uden hensyn til omgivelsernes egentlige behov. Det var ikke blot et dansk fænomen, men noget der bredte sig overalt i den akademiske verden som *Computer Science* (CS).

Til gengæld opstod der også et akademisk fagområde hvor man kigger på de egentlige behov i virksomheder. Det går under navnet Information Systems (IS). Her interesserer man sig til gengæld meget lidt for de tekniske løsninger. Dette skisma mellem CS og IS er en stor

barriere for at forskning kan bidrage til fortsat udvikling af IT-industrien, både her og i den store verden.

Jeg kom selv til at være medgrundlægger af først en udviklingsafdeling i Brown Boveri, hvor vi lavede systemer til overvågning af el-nettet, og senere en international udviklingsafdeling i NCR. Brown Boveri afdelingen var bemandet med udbrydere fra det smuldrende RC. Her lavede vi de mest succesrige produkter jeg har været med til. Vi havde tæt kontakt til kunderne og fik samtidig mulighed for at lave teknisk elegante løsninger. Jørgen Green, som kom fra RC's hardware-afdeling, kom til at spille en afgørende rolle. Han fik os compiler-folk til at forstå hvad kundens behov var.

Hos NCR fik jeg mange roller, en af dem var at lede kvalitetssikringen af software. Jeg blev også leder af afdelingen for Advanced Development, som skulle udvikle teknologier som kunne blive til fremtidens produkter. Her fik jeg endnu engang lov til at lege med avancerede løsninger på selvdefinerede problemer. Purisme var stadig den drivende kraft, som kom til at ødelægge flere projekter. I et af projekterne havde vi fået en medarbejder fra DTU, som kunne trække alt i langdrag, fx ved at insistere på at vi skulle definere et sprog hvor der slet ikke var reserverede ord. Stadig befængt af den gamle purisme var jeg ude af stand til at stoppe dette tidsspild.

Først i 1983 begyndte mit verdensbillede at skifte drastisk. Handelshøjskolen i København havde besluttet at oprette en uddannelse der kunne slå bro mellem IT og forretning. De studerende skulle simpelthen undervises 50% i hvert område. Jeg blev lokket til at være professor i anvendt datalogi og stå for den datalogiske del af uddannelsen. Som professor ved Handelshøjskolen stod dørene åbne til virksomhederne, og jeg lærte hurtigt langt mere om IT-industrien end jeg havde lært da jeg var ansat i den. Jeg lærte også en masse om ledelse, marketing, organisation, osv., men det krævede også tilpasning til en anden kultur. En af de ting der var svær at vænne sig til var, at man ikke skulle udtrykke sig kort og præcist. I handelshøjskoleverdenen taler man i timevis uden at sige ret meget fra en datalogs synspunkt. Hvor en videnskabelig datalogisk artikel højst må være 8 sider, er 8 sider bare et kort abstract i handelshøjskoleverdenen. Udtrykker man sig kort og præcist, tror de naturligt nok at man ikke har noget at sige.

Senere udnyttede jeg min handelshøjskoleviden til at finde på bedre måder at beskrive software-krav. Hvor idealet havde været at kravene detaljeret og præcist skulle beskrive hvad IT-systemet skulle gøre, gik jeg den modsatte vej. Man skulle beskrive de arbejdsopgaver som bruger og computer skulle udføre tilsammen. Kravet var så at computeren skulle støtte disse opgaver effektivt. Herved gav man udviklerne frie hænder uden at de tabte målet af syne. Metoden var også meget effektiv til store offentlige udbudsforretninger, hvor de tilbudte systemer i stor udstrækning fandtes allerede. Havde compiler-gruppen fundet på sådanne krav, havde RC måske set meget anderledes ud. Men det fandt vi ikke på fordi vi ikke kendte vigtigheden af at forstå behovene. Desuden var metoden nok enkel og nyttig, men bestemt ikke robust og præcis.

Trods mine forsøg på kulturel tilpasning, blev mine kolleger på Handelshøjskolen ved med at anse mig for specialisten der mente at IT var vigtigere end forretning - når alle vidste at det jo forholdt sig omvendt.

I 1999 blev jeg udlånt til det nye IT universitet, der fik Mads Tofte som direktør. Mads var ærke-datalog fra DIKU, men viste sig at have et naturtalent for ledelse. Under hans lederskab

opstod et nyskabende universitet med en ånd der på mange punkter kunne minde om det gamle RC. Det betød desværre også at man arvede syndromet med at alle gjorde hvad de syntes var rigtigt, og at den store sammenhæng derved vantrivedes. Fx opstod der også på ITU den traditionelle kløft mellem CS og IS. Blandt mine CS kolleger blev jeg anset for ham der anså økonomi og ledelse for vigtigere end datalogi - når alle jo vidste at det forholdt sig omvendt.

I dag udnytter jeg ofte mine mange erfaringer til at hjælpe IT opstartsvirksomheder. De lider næsten alle sammen af samme syndrom som compiler-gruppen gjorde: teknisk perfektionisme uden sans for kundernes og markedets egentlige behov. De unge mennesker bilder sig ind at de ved hvad markedet har brug for, og det kan være *meget* svært få dem til at tage kontakt med virkelige, potentielle kunder. Når det lykkes, bliver de til gengæld chokerede over de virkelige behov. Lykkes det tilstrækkeligt tidligt, er det muligt at redde situationen og udnytte den tekniske kreativitet.

Tilværelsen har mange grøfter og det kan være svært at holde sig midt på vejen. Jeg har lært hvor vigtigt det er at holde balancen, især når grøfterne råber om kap: *her går vejen, ikke der hvor du går*. En anden vigtig ting jeg har lært er at når en guru bliver for stor, spærrer han for udsynet.