appsw.1
1993-03-21

Software til Apparater: Industri kontra Forskning

Bodil Schrøder
Søren Lauesen
Jan Pries-Heje

Marts 1993

ISSN 0903-6571 93/1

Resumé

Dansk apparatindustri bruger mere og mere software i apparaterne, så en effektiv software-udvikling får større og større betydning. Denne rapport redegør for apparatindustriens software-udvikling og samspillet med dansk forskning.

Hovedkonklusionerne er:

- Industrien udvikler konkurrencedygtigt software, men der er også mange problemer, og derfor mulighed for forbedringer.
- De største problemer ligger i projekternes første faser, men konsekvenserne viser sig ofte først i testfaserne.
- Industrien har meget svært ved at ændre sine udviklingsmetoder. Større ændringer er fx en så stor risiko, at man ikke overvejer dem. Det giver en ond cirkel hvor nye idéer ikke kan afprøves fordi de skønnes for risikable, og derfor ved man reelt ikke om de faktisk hjælper.
- De virksomheder der bruger de mest moderne metoder har konstateret at de nok giver visse fordele, men samtidig gør udviklingen meget omstændelig. Der er stadig problemer, og de har tilsyneladende ingen lette løsninger.
- Der er meget dansk forskning vedr. software, men ikke ret meget der umiddelbart er relevant for software-udvikling i apparatindustrien. Samtidig har forskerne en meget snæver forståelse af den totale udviklingsproces i industrien.
- Flere danske forskningsresultater (fx formelle, matematiske metoder) har været prøvet i praksis, men stort set uden succes.

Det er svært at vurdere om forskerne ikke har noget relevant at byde på, eller om overførselstiden til industrien blot er meget lang. Overførselstiden kan være meget lang, for flere teknikker der er blevet udbredt de seneste år stammer faktisk fra forskningsresultater i 60'erne (fx struktureret programmering, objekt-orientering, datamodellering).

Kapitel 8 indeholder en række forslag til at forbedre situationen. Nogle af de mere overordnede er:

- Forskerne må langt bedre forstå den totale udviklingsproces og vise hvordan deres idéer kan indpasses heri.
- Der savnes forskning i kravspecifikation, fx samspillet mellem krav, muligt design, og økonomiske konsekvenser.
- Industrien må enten være villig til at tage idéer og selv "produktmodne" dem, eller vente til andre har gjort det (typisk via USA).
- Industrien må langt bedre analysere problemerne og systematisk forbedre sine metoder. Problemerne kan ikke løses blot med de rigtige værktøjer eller en mirakel-metode.
- Det er meget svært at få støtte til brobygning mellem forskning og udvikling, og det offentlige bør hjælpe her. Man bør også overveje at ændre ansættelseskriterierne for forskere, så formidling og praktisk erfaring kan få samme vægt som traditionel forskning.
- Man bør overveje at etablere en egentlig videreuddannelse i datalogiske emner, fx med aftenundervisning ligesom det kendes fra HD.

Indholdsfortegnelse

1. Indledning og konklusioner

Denne rapport er resultatet af en undersøgelse igangsat og støttet af Statens Teknisk-Videnskabelige Forskningsråd. Baggrunden var at en række personer fra dansk apparatindustri undrede sig over, at forskningen tilsyneladende ikke kunne hjælpe med at løse de problemer man har med at udvikle software til apparater.

Da apparatindustrien er en vigtig del af dansk industri, ville det være ønskeligt at forskningen kunne levere bedre metoder, osv. på dette felt. Men hvis samspillet mellem forskning og apparatindustri er problematisk, hvad er problemet så mere nøjagtigt?

For at svare på det, har vi dels undersøgt de udviklingsmetoder man bruger i apparatindustrien og de problemer man har, dels den forskning der foregår i Danmark på området. Desuden har vi undersøgt hvordan formidlingen sker mellem forskning og industri, specielt hvilken rolle Teknologisk Institut og Elektronikcentralen spiller.

Vi har koncentreret os om selve udviklingsprocessen for apparat-software. I Danmark foregår der også forskning der sigter på software til specielle anvendelsesområder, fx billedbehandling, neurale net, højhastighedsberegninger, etc. Disse specielle anvendelsesområder har vi ikke forsøgt at undersøge. Vi vil altså mest se på innovation i udviklingsprocesserne, men ikke på innovation i produkterne [Note 1].

Vi har undersøgt 17 organisationer: virksomheder, forskningsinstitutter og formidlere (nogle organisationer spiller flere roller). Desuden har vi personlige erfaringer fra ansættelse i en række virksomheder. Det har alt i alt resulteret i et stort antal observationer som er svære at sammenstille, fordi organisationerne er meget forskellige.

Hovedparten af rapporten redegør for disse observationer. Trods deres forskellighed, er der alligevel så mange fælles, overordnede træk at vi har turdet drage en række konklusioner, som vi bringer her i indledningskapitlet.

Set i et globalt perspektiv leverer dansk apparatindustri udmærkede, konkurrencedygtige produkter med et stort software-indhold. Alligevel er der problemer, og ved at angribe dem burde det være muligt at forbedre udviklingsprocessen og konkurrenceevnen. Derfor koncentrerer rapporten sig om at identificere problemerne, især fordi det har vist sig at både forskerne og industrien selv kun har en meget ufuldstændig forståelse af de egentlige problemer.

Allerede i indledningskapitlet giver vi en oversigt over de vigtigste problemer ved selve udviklingsprocessen. Endelig giver vi et par eksempler på relevante forskningsprojekter og industriens syn på dem. De senere kapitler er en mere detaljeret gennemgang af problemerne. I sidste kapitel giver vi en række forslag til løsninger.

Man kunne fristes til at tro, at de mange problemer vi ruller op er specielle for Danmark, specielt fordi vi har små virksomheder og små forskningsmiljøer. Der er dog intet der tyder på at det står bedre til i USA, såvidt vi har kunnet udlede af litteratur og personlige erfaringer. Den manglende overførsel af forskningsresultater til industrien ser ud til at være et globalt problem i software-industrien [Note 2].

Det er svært at vide om der ikke findes nyttige forskningsresultater, eller om overførslen til industrien blot tager lang tid. Formentlig gælder lidt af begge dele. Den lange indtrængningstid er der flere eksempler på: Struktureret programmering stammer fra 60'erne (fx Dijkstra) og trængte igennem sidst i 70'erne båret af programmeringssprog som Pascal og C. Teknikker som objekt-orientering og datamodellering, der trænger igennem i dag, stammer faktisk også fra forskningsresultater i 60'erne (Simula fra Norge og Bachman's datamodellering).

1.1 Undersøgelsens form

Vores hovedkilde har været interviews i følgende organisationer:

Apparatindustri:
- Brüel og Kjær
- Danfoss
- Dantec Elektronik (måleteknik)
- Eskofot
- GN-Elmi
- Lumetech
- NKT

Formidlere (ofte flere roller):
- Elektronikcentralen
- IFAD, Odense
- Keld Hornbech Svendsen, konsulent, egen virksomhed
- Dansk Teknologisk Institut, Tåstrup
- Dansk Teknologisk Institut, Århus

Forskningsinstitutter:
- DAIMI, Århus Universitet
- Datalogisk Institut, Københavns Universitet
- Institut for Datateknik, DTH
- Institut for Elektroniske Systemer, AUC

Alle interviews er blevet gennemført i efteråret 1992. Vi er overalt

blevet modtaget med stor velvilje og har haft lejlighed til at tale med både ledere og menige medarbejdere. Typisk har et interview strakt sig over 2 til 3 timer, og i nogle tilfælde har vi foretaget flere interviews i samme virksomhed. Nogle gange har vi interviewet én person ad gangen, andre gange flere. I flere tilfælde har vi desuden set på udviklingsdokumenter for at kunne foretage en uafhængig vurdering.

Da der ofte er tale om fortrolige oplysninger, har vi ikke i rapporten identificeret kilderne til de enkelte udsagn.

1.2 Lidt om forfatterne

Vores egen baggrund har gjort det muligt at supplere med yderligere oplysninger.

Bodil Schrøder er cand.scient. i datalogi fra DIKU, 1975. Hun viste i sit speciale hvordan man laver multi-programmering i praksis (MIK-kernen), og mange virksomheder har siden brugt disse resultater i deres apparater. Bodil har udviklet industrielt software af alle slags som ansat hos Rovsing A/S, Nordisk Brown Boveri (senere en del af ABB), Textware, Softcom, og Ambrasoft.

Søren Lauesen er cand.scient. i matematik og fysik fra 1965. Han har udviklet software på A/S Regnecentralen, Nordisk Brown Boveri (i dag ABB), og NCR. Desuden har han været konsulent for mange virksomheder. Projekterne har bl.a. omfattet oversættere, operativsystemer, datakommunikation, processtyring, og databasesystemer. Søren er i dag professor i anvendt datalogi på Handelshøjskolen i København og har siden 1985 siddet i STVF's E-kommission.

Jan Pries-Heje er cand.merc.-dat fra 1989. Han har både i sit speciale, og senere som licentiatstuderende, undersøgt hvordan software-firmaer i praksis udvikler produkter, og han er bl.a. meget velbevandret i litteraturen om system- og produktudvikling. Jan er i dag adjunkt i datalogi på Handelshøjskolen i København.

1.3 Om metoder, teknikker og værktøjer

I resten af rapporten kommer vi hyppigt til at bruge ordene *metode*, *teknik* og *værktøj*. I industrien bruges de lidt i flæng.

I denne rapport mener vi med en *teknik* en forskrift der dækker en mindre del af udviklingsprocessen. Som eksempler på teknikker kan vi nævne (1) dataflow som beskrivelse af et design, (2) whitebox-testing af et system, (3) systematisk opsamling af fejl og ændringsønsker.

En *metode* er de principper der styrer hele udviklingsprocessen og får de enkelte teknikker til at hænge sammen. Den mest udbredte metode

i apparatindustrien er SPU (Systematisk Program Udvikling, [Note 3]). Den foreskriver en række udviklingsfaser og giver eksempler og gode råd for hver fase. På mange områder er den ikke særlig præcis, så hver virksomhed supplerer den derfor ofte med diverse teknikker og tjeklister.

Et *værktøj* er et edb-baseret system der støtter en eller flere teknikker. Fx er der værktøjer der kan hjælpe med at tegne dataflow-diagrammer, holde styr på om alle tilfælde er dækket ved whitebox-testing, etc. Et CASE-værktøj er et edb-baseret system der støtter en række sammenhængende teknikker, så en større del af udviklingsprocessen dækkes [Note 4].

1.4 De væsentligste konklusioner

1.4.1 Problemer i industrien

- Der er mange problemer i forbindelse med software-udvikling i apparatindustrien. Det viser sig bl.a. ved at udviklingsprocessen bliver meget dyrere og mere langvarig end forventet. (Der er eksempler på problemerne nedenfor).
- I mange tilfælde konstaterer man virkningerne uden rigtig at erkende hvilke problemer de er konsekvenser af. Der er dog stor forskel fra det ene firma til det andet på erkendelsen af den slags problemer, samt på viljen og evnen til at løse dem.
- Der er et stort behov for hurtigt at kunne ændre udviklingsorganisationens brug af metoder, teknikker og værktøjer, men en lang række faktorer modvirker mulighederne for forandringer. En vigtig faktor er at risikoen ved at bruge en ny metode synes meget stor, samtidig med at de påståede gevinster virker utroværdige.
- Nogle steder anvender man nedskrevne udviklingsmetoder (typisk SPU med Struktureret Analyse/Design, [Note 5]), men de hjælper tilsyneladende kun lidt. Fx hævder enkelte virksomheder at produkterne er blevet mere fejlfrie med disse metoder. Til gengæld får man mere bureaukratisk stivhed og mere arbejde.

1.4.2 Problemer med teknikker og metoder

- Der findes en række teknikker der kan anvendes i forbindelse med forskellige dele af udviklingen, men de benyttes i praksis ikke ret meget.
- De eksisterende teknikker og metoder er ofte ikke særlig tilgængelige. Der findes ikke grydeklart undervisningsmateriale, egentlige lærebøger, eller støtte-værktøjer.
- Kendskabet til teknikkerne er ikke ret udbredt. Det gælder også

velprøvede teknikker som man kunne have lært ved uddannelserne.

- De fleste teknikker løser kun en lille del af det samlede kompleks af problemer ved systemudviklingen. Teknikkerne hænger ikke sammen med andre teknikker, eller det kan være at indlæringstærsklen er høj.
- Der mangler teknikker på nogle vigtige områder. Specielt mangler der teknikker til på den ene side at formulere behov, krav og økonomiske begrænsninger til et produkt, og på den anden side at få det udmøntet i et design og en implementation indenfor de givne rammer.
- Selv om alle løsningerne forelå ville der ikke være nogen der vidste det (man drukner i mængden af informationer og giver op).

1.4.3 Problemer vedr. forskningen

- Der er meget lidt af den forskning der foregår i Danmark der umiddelbart er relevant for apparatindustrien.
- Nogle af de forskningsprojekter der er på området er *meget* gode og yderst relevante, men resultaterne er alligevel ikke blevet brugt i apparatindustrien. KAITS og Mjølner er eksempler på sådanne projekter (se nedenfor). Den eneste rimeligt udbredte metode vi har kunnet spore (SPU) stammer ikke fra forskere, men fra nogle erfarne udviklere.
- De formelle metoder (matematisk baserede) er forsøgt brugt i praksis, men de er ikke blevet nogen succes (se nedenfor).
- Forskerne ser på udvalgte problemstillinger der er meget afgrænsede i forhold til de opgaver man møder i praksis.
- Forskerne ved meget lidt om industrien, fx mangler de forståelse af de indledende faser. Manglen på relevant viden er ikke erkendt.
- Forskningsresultaterne "oversælges", idet forskerne tror at deres løsning løser alle væsentlige problemer. Reelt ved man sjældent om de overhovedet kan løse nogle af de relevante problemer.
- Forskerne leverer ideer, men industrien ønsker grydeklare løsninger med værktøjer der løbende vedligeholdes svarende til de aktuelle markedsstandarder.
- Hverken industrien, formidlerne eller andre forskere er særligt velorienterede om hvad forskerne laver.

1.4.4 Problemer hos formidlerne

Formidlerne har en glimrende forståelse af problemerne i apparatindustrien, men deres indsats består primært i at afholde kurser i eksisterende metoder, suppleret med konsulent- og udviklingsbistand. En sjælden gang overtager man idéer fra forskere og holder kurser over dem.

- Der er ingen penge til at udvikle de grydeklare metoder der kunne løse nogle af industriens behov.
- Der er ringe samspil mellem forskere og formidlere. Fx organiserer Elektronikcentralen en række ERFA-grupper, men forskerne er ikke med, bl.a. fordi de ikke har råd.
- Formidlerne konkurrerer indædt indbyrdes.

1.5 Vigtige problemer ved software-udviklingen

En meget stor del af rapporten handler om de problemer vi har fundet ved selve software-udviklingen i apparatindustrien. Vi kan resumere de vigtigste sådan:

1.5.1 Produktafgrænsning og kravspecifikation

Alle virksomhederne har svært ved at finde en balance mellem det man prøver at opnå, og det der reelt er muligt indenfor de givne økonomiske og tidsmæssige rammer.

Samspillet mellem de egentlige markedsbehov og de krav man formulerer er vanskeligt. Der er ingen systematik til at sammenholde kravene med behovene og til at vurdere konsekvenserne for udviklingstid og økonomi.

Resultatet af produktafgrænsningen er normalt en kravspecifikation, men den typiske kravspecifikation er meget upræcis og ufuldstændig. Vigtige krav som performance, portabilitet og vedligeholdbarhed bliver fx ikke overvejet.

Problemet vanskeliggøres af at specifikationen som regel er formuleret på sådan en måde at marketingafdelingen (eller kunden) ikke kan leve sig ind i den. Resultatet bliver at man først meget sent i udviklingsprocessen opdager væsentlige krav, som nu bliver meget dyre at opfylde. Tilmed har vi oplevet at udviklerne (der formulerer kravspecifikationen) ikke har det fornødne kendskab til anvendelsesområdet.

Dette område er nok det der er mest oplagt til en forskningsindsats, men det der mangler er mest "blød" forskning, der næppe kan baseres på matematiske metoder.

1.5.2 Design

I designfasen prøver man at opdele produktet i en række "moduler", både hardware og software.

Her kan vi konstatere at designet er meget ufuldstændigt. Grænsefladerne til modulerne specificeres meget overfladisk og visse dele overvejes slet ikke. Det gælder fx konfigureringsfunktioner og brugergrænseflade.

Det er typisk at opdelingen i hardware-dele fastlægges ret tilfældigt - og inden software er overvejet.

Endelig er der en næsten total mangel på sporbarhed mellem kravene og det foreslåede design. Man forvisser sig med andre ord ikke systematisk om at kravene faktisk bliver overholdt og om at designet ikke har overflødige dele.

Matematisk baserede metoder (formelle metoder) er ofte af forskere blevet anbefalet som løsning på designproblemerne, men de bliver ikke accepteret af industrien, trods ihærdig formidlingsindsats fra forskerne. Et eksempel er MetaIV, der omtales nærmere på side 15.

1.5.3 Test

Testfaserne (modultest, integrationstest og accepttest) tager for lang tid. Hovedårsagen er problemerne fra de tidligste faser, som nu dukker op og tager meget lang tid at løse.

1.5.4 Metode

En metode er de principper der styrer hele projektforløbet og får de enkelte teknikker til at hænge sammen.

Nogle virksomheder bruger slet ingen egentlig metode og arbejder uden overordnet plan. Ganske få virksomheder bruger en sammenhængende metode, typisk baseret på SPU med egne tilføjelser.

Generelt gælder det at sporbarheden gennem projektet er utilstrækkelig. Man kan ikke systematisk spore krav til design, eller design til faktiske programmoduler.

De der bruger en sammenhængende metode har erfaret at man producerer en masse dokumenter, men de bliver ikke brugt i de følgende faser. Desuden opgiver man at vedligeholde dem. Der er dog undtagelser, fx et delprojekt der bruger KAITS (side 14).

En anden erfaring er at tidsplanen, når man gør det "rigtigt", bliver uacceptabel lang. På et vist tidspunkt opgiver man derfor at følge metoden og fortsætter "som man plejer", omdefinerer projektet i al

hast, osv.

1.6 Nogle danske metoder

De mest lovende danske forskningsprojekter vi har mødt er Mjølner og KAITS. Desuden har vi talt med udviklere der har kendskab til matematiske metoder som MetaIV. Vi vil kort resumere disse metoder og prøve at forklare hvorfor de ikke har slået an, trods stor formidlingsindsats fra forskerne. Endelig vil vi omtale SPU, en dansk metode der er vidt udbredt, men ikke stammer fra forskere.

1.6.1 Mjølner

Mjølner er udviklet ved DAIMI og Forskerparken i Århus. Det er en metode med integrerede værktøjer der primært dækker design, programmering og test. Med Mjølner designes og programmeres et system som en samling generaliserede objekter. Man bruger et specielt sprog, som også bruges i selve programmerne.

Mjølner-gruppen beretter at metoden er brugt til administrative systemer i Storebælt-projektet, samt til en digital switch i Finland, men vi har ikke undersøgt det nærmere. Der er ikke brugere i den danske apparatindustri.

Metodens store styrke er at der er fin sporbarhed gennem hele projektet. Desuden er vedligeholdelsen af fx designdokumenter automatisk: Når man ændrer i et program, bliver designdokumenterne automatisk opdateret.

Metoden ser således ud til at kunne løse to store problemer i apparatindustrien: Sporbarhed og vedligehold af designdokumenter. Hvorfor bliver den så ikke brugt?

Flere af de virksomheder vi har besøgt kendte til Mjølner, men man mente ikke metoden var relevant for apparatindustrien. Nogle typiske kommentarer var:

- For at udnytte metoden må man basere sig på det særlige Mjølner-sprog. Og hvilken fremtid har det? Hvilke hardware-arkitekturer kan det støtte fremover? Og kan det kombineres med kommende CASE-værktøjer?
- Metoden er nok ikke egnet til apparat-software. I hvert fald havde man ikke set eksempler fra forskernes side på det. (Man var ikke klar over at den faktisk var blevet brugt til et apparat - den digitale switch).

Disse udsagn afspejler at metoden anses for alt for risikabel, den kan ikke kombineres med andre teknikker (fx C-programmering som er en standard overalt), og den er ikke grydeklar til apparater. Mjølner-grup-

pen beretter i øvrigt at den fik afslag på en ansøgning om udviklingsstøtte i 1985 med den begrundelse, at objekt-orientering ikke var main-stream forskning og at man ikke var ADA-baseret. I dag kan man se at begrundelsen var meget kortsynet.

Til sammenligning kan bemærkes, at det mest udbredte objekt-orienterede sprog i dag er C++. Det har også sine rødder i DAIMI, men det skulle en tur over USA for at blive udbredt. Desuden er det baseret på almindeligt C, som man frit kan kombinere det med. Begge dele reducerer risikoen tilstrækkeligt i industriens øjne.

1.6.2 KAITS

KAITS startede som et samarbejde mellem Institut for Datateknik (DTH) og Brüel & Kjær. KAITS er i dag blevet til et værktøj udviklet af IFAD i Odense. Det markedsføres af DDC-International under navnet CEDAR [Note 6].

KAITS dækker lidt af kravspecifikationsfasen samt de forskellige trin i designfasen, baseret på matematiske eller datalogiske notationer. Metoden er specielt beregnet til de dele af apparat-software der tager sig af signaler fra sensorer og til aktuatorer. Alle eksemplerne i metodebeskrivelserne drejer sig om apparater.

Metodens styrker er en meget kompakt beskrivelse af systemerne, stor sporbarhed gennem designet, mulighed for at ræsonnere sig frem til den nødvendige parallellitet, og mulighed for at regne på performance. Igen egenskaber der kunne løse væsentlige problemer i apparatindustrien.

Det er lykkedes os at finde frem til ét projekt der har brugt metoden med succes. I hvert fald var projektmedarbejderen som vi interviewede glad for det, mens afdelingslederen og andre projektgrupper ikke vidste noget om det eller mente det var ret uinteressant. Produktet hvor KAITS blev brugt er markedsført og vedligeholdes fortsat i dag. Også dokumentationen vedligeholdes, og det hænger formentlig sammen med sporbarheden og den kompakte form, der fylder meget lidt i forhold til selve koden.

Vi har hørt om et andet projekt der også forsøgte at bruge metoden, men det blev efter sigende en total fiasko fordi udviklerne helt havde misforstået metodens grundlag.

Trods ihærdig søgen har vi ikke kunnet finde andre KAITS-anvendelser. Selv IFAD, der har udviklet den til et værktøj, anvender den ikke selv, selvom de laver apparat-software.

Hvorfor er metoden ikke mere udbredt? En mulig faktor er at metoden er blevet misforstået. Fx hørte vi hos en anden virksomhed, at metoden ikke egnede sig til apparater, for det "var jo tydeligt at den krævede

en PC med skærm". Man forvekslede her at værktøjet kørte på en PC, med at det genererede specifikationer for et apparat uden skærm. (Vi forsøgte efter bedste evne at korrigere misforståelsen).

En anden mulig faktor er at metoden har været "oversolgt". Den blev fremstillet som om den løste de fleste af udviklingsproblemerne for apparater, men reelt er det kun en mindre del af hele processen den støtter. Apparater har i dag typisk også en PC-del med database, skærm og vinduesorienteret brugergrænseflade. Disse dele støttes ikke af KAITS. Desuden har KAITS efter sigende en svaghed vedrørende mere sammensatte datastrukturer, som bl.a. optræder ved apparater med mange ind- og udgange.

Endelig har KAITS nok også en ret høj indlæringstærskel.

1.6.3 Matematiske metoder (MetaIV)

MetaIV stammer også fra Institut for Datateknik (DTH) og er i familje med VDM [Note 7]. Det er en fuldt matematisk baseret metode der dækker lidt af kravspecifikationsfasen samt de forskellige trin i designfasen. Den har forsøgsvis været brugt i flere danske virksomheder.

Vi har talt med flere udviklere der har brugt MetaIV eller deltaget i projekter hvor det blev brugt. Alle udsagn har gået på at det var en hemsko, snarere end en hjælp. Man kunne ikke udtrykke det væsentlige, andre kunne ikke forstå hvad man havde specificeret, osv.

En udvikler fortalte at det var en stor lettelse for ham da han opdagede at det ikke var forbudt at skrive kommentarer i en MetaIV-specifikationen. Så kunne han da endelig udtrykke nogle af de tanker han ellers brændte inde med.

MetaIV havde oprindelig succes hos DDC, hvor metoden blev brugt som et afgørende element i udviklingen af ADA-oversætteren. Men udvikling af en ADA-oversætter er væsensforskellig fra udvikling af et apparat, bl.a. fordi hele kravspecifikationen forelå meget detaljeret i form af en specifikation af ADA-sproget.

Den manglende accept af MetaIV i andre kredse skyldes sandsynligvis at metoden blev "oversolgt", den meget høje indlæringstærskel, samt den manglende succes ved realistisk brug.

Endelig er mange ingeniører i apparatindustrien visuelt orienteret: De foretrækker tegninger og diagrammer i den kreative fase, mens MetaIV kræver matematisk notation.

MetaIV er i dag videreudviklet til et værktøj, RAISE, der markedsføres af CRI og har nogle udenlandske kunder. Der er ingen danske kunder. Vi har ikke kunnet få oplyst om nogle af kunderne faktisk bruger det.

1.6.4 SPU

I apparatindustrien findes der en ret udbredt metode: SPU (Systematisk Program Udvikling, [Note 3]). Den er udviklet i 1985 med støtte fra Teknologirådet. Den blev til på den måde, at en række erfarne folk satte sig sammen og beskrev hvordan de mente at man burde gøre når man skulle udvikle et edb-baseret produkt.

Metoden foreskriver en række udviklingsfaser og giver eksempler og gode råd for hver fase. Der er også gode råd om projektledelse og dokumentation. Metoden er ikke særlig præcis, så hver virksomhed supplerer den derfor ofte med diverse teknikker.

Metoden er blevet udbredt gennem formidlerne: både private konsulenter, Teknologisk Institut, og Elektronikcentralen. Bemærk at udbredelsen har taget mange år, og flere virksomheder er først nu ved at indføre den.

Man kan ikke kalde metoden for forskningsbaseret, og der er så vidt vi ved heller ingen systematiske undersøgelser af hvordan den virker i praksis. Vi har fået udsagn om at den tilsyneladende kun hjælper lidt. Ganske vist hævder enkelte virksomheder at produkterne er blevet mere fejlfrie med disse metoder, men til gengæld får man mere bureaukratisk stivhed og mere arbejde.

SPU's udbredelse hænger nok sammen med dens uformelle karakter. Det er primært en ramme suppleret med gode råd og simple eksempler, og man kan uden videre afvige efter behov - hvilket man i øvrigt gør i udstrakt grad.

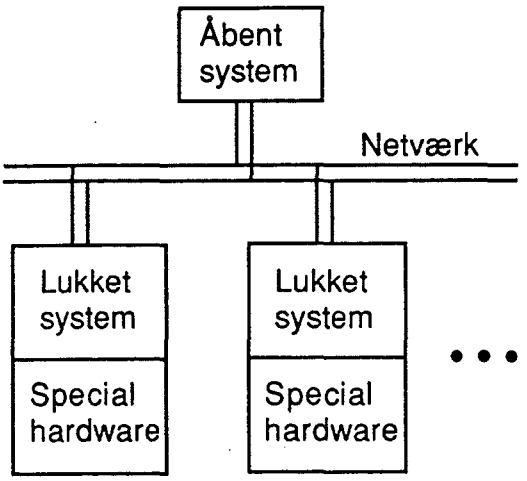
2. Apparater

Apparatindustrien er en broget skare virksomheder. I Danmark udvikles der bl.a.:

- Måleapparater
- Medicoteknik
- Teleudstyr
- Grafisk udstyr
- Billede og lyd (fx B&O)
- Intelligent udstyr inden for byggebranchen (fx Danfoss)
- Radioudstyr, fx mobilradioer
- Automater, fx til kaffe eller benzin

Betegnelsen "apparat" bruges om alt muligt fra et enkeltstående apparat (støvsuger, kaffeautomat, video), over et apparat hvor al styringen sker ved hjælp af én PC, og op til store integrerede systemer hvor enkeltapparater samarbejder i store netværk, der fx ender med en række UNIX-arbejdsstationer.

I rapporten bruger vi følgende model af et apparat. Det vigtige er at forstå, at et apparat ofte består af det vi kalder et lukket system kombineret med et åbent system. De to typer system har forskellige teknologiske udviklingstendenser.

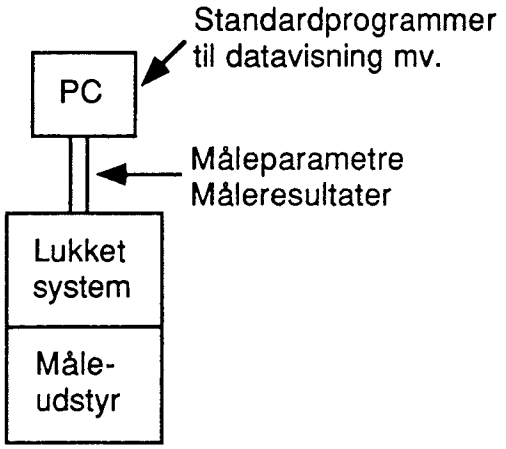


Den nederste del af figuren viser et eller flere **lukkede tidstro systemer**, der styrer det hardware der er specielt for det pågældende apparat, fx måleapparatur, aktuatorer, kommunikationslinjer. Det er typisk implementeret som en række kort med hver sin processor indenfor samme familie. På hvert kort kører der et selvstændigt system baseret

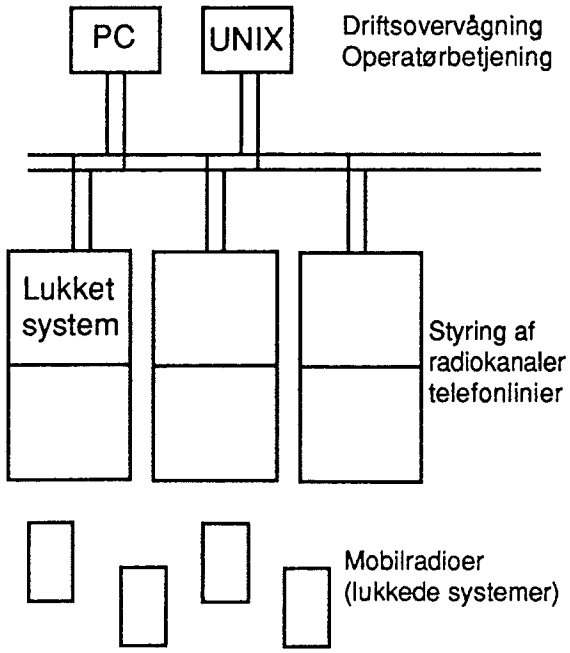
på en fælles - ofte selvudviklet - realtidskerne. Koden er lagret i PROM'er. Kortene kommunikerer med hinanden og/eller det "åbne" system via en protokol, der som oftest er en hjemmelavet beskeduddvekslingsmekanisme. Der er sjældent tilkoblet skærm og tastatur, men måske er der en anden brugergrænseflade (knapper, lysdioder, ...). Der vil normalt ikke være en disk.

Den øverste del af figuren illustrerer et **åbent system** der på den ene side kommunikerer med det lukkede system, og på den anden side kommunikerer med omverdenen. Det åbne system er karakteriseret ved at der anvendes standard hardware og software. Det har typisk en brugergrænseflade og en disk. Det vil ofte være en PC med en grafisk brugergrænseflade og en eller anden form for database. Kommunikationen med omverdenen vil være baseret på standardprotokoller, mens den interne kommunikation mellem kortene og de to systemer ofte er hjemmelavet. På figuren er det kaldt et netværk: det kan være et lokalnet, en eller flere serielle forbindelser, eller en backplane-bus med tilhørende kommunikationssoftware.

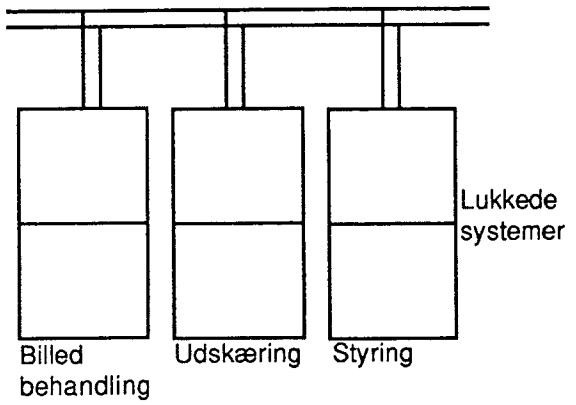
De fleste af de apparater vi har hørt om kan med lidt god vilje passes ind i ovenstående model. Her er nogle eksempler på apparater der er lidt specielle udgaver af modellen:



Eksempel 1: Et **måleapparat** hvor målingerne (fx optælling af partikler) sker i et lukket system med de kritiske dele i hardware. Der er tilknyttet en PC hvor man sætter parametre op til målingen og samler resultater op. Al lagring og efterbehandling sker i PC'en og i vid udstrækning ved hjælp af standardprogrammer. Grænsefladen kan fx være en seriel linje eller et kort med et DMA-interface.



Eksempel 2: Et **mobilradiosystem** hvor de enkelte radioer er lukkede systemer, mens de stationære systemer består af et hierarki af distribuerede systemer, hvor hvert kort betjener en bestemt type udstyr, fx en radiokanal eller en telefonlinie. De stationære systemer omfatter også PC-kort og forbindelser til standard edb-udstyr til operatørbetjening, lagring af konfigureringsinformation, driftsstatistikker, overvågning, konteringsdata osv.



Eksempel 3: En **fødevaremaskine** hvor 3 samarbejdende lukkede systemer varetager hhv. en billedbehandlingsdel, en udskæringsdel, og en styringsdel.

2.1 Den lukkede del af systemerne

Det lukkede system er ofte styret af en real-time kerne eller noget simplere, fx DOS eller måske "ingenting". Her er udviklingen gået ret langsomt.

Programmeringen foregår overvejende i højere programmeringssprog. C er dominerende, og ellers er det ret spredt, fx Concurrent Pascal, Modula-2, PLM. Assemblere bruges til meget tidskritiske ting. Vi har ikke set eksempler på *gennemførte* projekter med C++ i objektorienteret stil, men mange tror at udviklingen går den vej.

Indenfor de lukkede systemer sker der en kraftig udvikling på hardwaresiden. Der er flere og flere opgaver der kan løses med nye former for hardware. Til gengæld skal dette hardware oftere programmeres eller konfigureres, og dermed bliver det en form for software. De værktøjer og metoder der hører til, er imidlertid helt forskellige fra dem man bruger til resten af sin softwareudvikling. Det er ofte også andre personer der udfører opgaven, typisk personer uden softwareuddannelse, men med en hardware-baggrund.

2.2 Den åbne del af systemerne

Det åbne system omfatter de opgaver man har kunnet flytte væk fra de lukkede systemer, fx adgang til standardhardware og -software herunder værktøjer til brugergrænseflader, filer, database, kommunikationsprotokoller og andre programpakker.

- Med det åbne system får man en billig løsning på mange problemer (adgang til færdigt hw/sw, software der er flytbart svarende til udviklingen på markedet)
- Konkurrencen og bevægeligheden indenfor de åbne systemer er voldsom. Man skal have fuld grafik og farver fordi de andre har det, og man skal holde sig opdateret svarende til de nyeste versioner af 3. parts efterbehandlingspakker, fx statistiske pakker med grafisk visning.
- Det åbne system giver muligheder for at lave integrerede løsninger, hvor mange standardprogrammer og -pakker kombineres.
- Kravene til de integrerede systemer er at overholde en masse standarder, hvoraf definitionen af de fleste er på et ret indledende stadium.
- Pris og leveringsdato er stadig konkurrenceparametre, men det bliver nu også væsentligt med fleksibilitet, så man kan tilpasse produktet til krav der ikke foreligger på forhånd. Pålidelighed bliver utrolig vigtigt i store integrerede systemer. Brugergrænsefladen bliver også på mange måder væsentlig: den skal leve op til nogle standarder og den skal være flot - det er den der "tegner" apparatet. Der er derimod ikke meget der tyder på at bruger-

venlighed er blevet en væsentlig konkurrencefaktor.

- Det åbne system er tit PC-baseret, men kan også være andet. Softwareløsningen baseres på det operativsystem der findes. Ofte ses DOS, Windows eller UNIX-systemer, hvor real-time problemerne er løst ad hoc eller hvor de slet ikke erkendes.

2.3 Det samlede system

Det samlede produkt består af hardware og software i både den lukkede og den åbne del, men det er væsentligt at forstå at der er tale om forskellige delsystemer med vidt forskellige krav og med vidt forskellige muligheder, traditioner, metoder og værktøjer. Der er en tendens til at betragte de enkelte delsystemer som mere uafhængige end godt er. Det giver en dårlig basis for designet - specielt af real-time aspekterne - og det giver masser af problemer i forbindelse med integration.

Med de åbne systemer får man en masse systemmæssige muligheder relativt billigt via standard software og hardware, men prisen for at udnytte dem til et mere ambitiøst produkt er en øget kompleksitet, en højere indlæringstærskel, og nye pålidelighedsproblemer.

Vi hørte flere steder, at markedet bevægede sig væk fra enkeltstående apparater, som var lukkede systemer, mod integrerede systemer, hvor de traditionelle apparater indgår som knuder i et større netværk med standardiserede grænseflader. Det betyder for leverandøren, at man vover sig ud i store softwareudviklingsprojekter helt op i størrelsesordenen 100 mandår.

Alt i alt leverer dansk apparatindustri gode produkter med konkurrencedygtigt software. Alligevel er der mange problemer med software-udviklingen. De viser sig først og fremmest ved at udviklingen tager meget længere tid end forventet. Det vil vi se nærmere på i næste kapitel.

3. Apparatsoftware

Interviewrunden har - sammenholdt med vores egne erfaringer - vist os at der trods store forskelle er en del fælles træk i den måde der arbejdes på i apparatindustrien.

I det følgende gennemgås de typiske faser i et softwareudviklingsprojekt. For hver fase beskriver vi nogle karakteristika ved forløbet, og vi forsøger at påpege de problemer vi har set og hørt om. Desuden ser vi på tværgående emner som ledelse, metoder, og teknologioverførsel.

3.1 Produktafgrænsning og kravspecifikation

I apparatindustrien laver man *produkter* under en eller anden form. Kravene til fx mobilradioer, der laves ens i store styktal, er ganske vist forskellige fra kravene til avanceret måleudstyr i millionklassen, men softwareudviklingen har mange fælles træk:

- Der er som regel *ikke* en kunde der stiller krav eller som man bare kan spørges til råds.
- Fejl er meget dyre at rette i det endelige produkt.
- Produktet skal kunne tilpasses til mange forskellige kundekrav uden programændringer (konfigurerbarhed).
- Produktet kommer til at eksistere i et antal forskellige versioner.
- Vedligeholdelsesfasen er lang, og produktet kommer til at ændre sig væsentligt på en måde man ikke kan forudsige i definitionsfasen.
- Funktioner til tuning og fejlsøgning skal være en del af produktet.
- Brugergrænsefladen er dels en egenskab ved produktet, dels et potentielt salgsargument/blikfang.
- Den totale udviklingstid (time-to-market) er en afgørende faktor for produktets succes.

En af formidlerne berettede at tidsforløbet for et produkt typisk er således:

- idéfase: 1-2 år
- overordnet kravspecifikation: 1 år
- udvikling: 1-2 år
- levetid: 10 år (med mange versioner undervejs).

Det passer godt med vores observationer, idet vi fx så produkter der havde været på markedet i ca. 10 år og andre hvor en ca. to-årig udviklingsfase netop var afsluttet. Det mest bemærkelsesværdige er den lange levetid og det lange forarbejde til selve udviklingen.

3.1.1 Produktidé

Idéen til nye produkter opstår bl.a. ved at kombinere kendskab til:

- anvendelsesområdet (nye behov)
- markedet (hvad konkurrenterne og kunderne gør)
- begrænsninger i de eksisterende produkter
- ny teknologi (hardware og software)

Udviklingen i hardware til de lukkede systemer kan være afgørende, enten fordi den eksisterende hardware ikke længere kan skaffes, eller fordi den nye giver nye muligheder.

Udviklingen i software kan betyde at man kan lave helt nye produkter, fx med avanceret billedbehandling i forbindelse med grafisk udstyr. Men det kan også være at man "bare" er tvunget til at leve op til markedskrav om standard hardware og software, fx farver, grafik, levering af data på diskette eller en anden standardgrænseflade.

Eksempler på hvor produktidéer kan komme fra:

- Produktfornyelse indenfor firmaet (mange eksempler, [Note 8]).
- Ideen til produktfornyelse ligger klar, men produktet defineres først for alvor når der kommer en konkret kunde (flere eksempler).
- Lægerne på en sygehus-afdeling fik en idé til et system de selv kunne lave på PC'er med støtte fra amtet (frikøbt tid). Via kontakt til det firma der leverede overvågningsudstyr blev det til et produkt, der nu bliver markedsført af dette firma. Undervejs blev software-udviklingen meget større end forventet, og det blev nødvendigt at få konsulent- og udviklingshjælp udefra.
- Forskere der tager et forskningsresultat og forsøger at opbygge et produkt (Mjølner og KAITS er eksempler).

Vi har ikke set tegn på at gode produktidéer er en mangelvare. Det passer godt med at flaskehalsen er udviklingsomkostningerne og -tiden.

3.1.2 Kravspecifikation

Hvis der er en kunde involveret, vil kundens krav være med som en del af kravene til produktet, ellers defineres produktet på basis af en kombination af den viden der er hos udviklingschefen, sælgerne, udviklerne og evt. kundesupport. Idéen diskuteres måske gennem længere tid inden man for alvor definerer produktet, men som regel bliver der på et eller andet tidspunkt skrevet en form for **kravspecifikation**.

Kundens krav kommer ofte som et velkomment spark. Man har måske længe tænkt på produktfornyelsen, men det er blevet ved snakken. Når den første kunde står der, beslutter man at gå efter denne kundes krav, plus så meget som muligt af det man havde tænkt sig i det mere generelle system.

Kravspecifikationen kan have forskellige former, men det er som oftest et dokument i almindelig tekst, undertiden suppleret med et antal skærmbilleder, der skal illustrere brugerens interaktionsmuligheder. Vi har dog også set "kravspecifikationer" der kun består af skærmbilleder (en slags prototype), og i så fald mangler der helt alle de funktionelle krav og krav vedr. performance, vedligeholdbarhed, etc.

3.1.3 Problemer

Den økonomisk stramme situation betyder at man tøver med at lave nye produkter. Det betyder samtidig at når man så endelig laver noget, skal det kunne en masse nyt på én gang.

Hovedproblemet er dog at **kravspecifikationen ikke er god nok**. Dette punkt fremhæves af alle vi har snakket med, og det fremgår også af diverse andre undersøgelser [Note 9]. Der er for meget der ikke bliver taget stilling til før man går igang. Kravspecifikationen bliver til en ønskeseddel, hvor det i stedet burde være arbejdsgrundlaget for at specificere et system, hvor en række krav og muligheder er afvejet mod hinanden.

- Kravene bliver ikke formuleret præcist nok.
- Kravene kan ikke forstås og vurderes af ikke-teknikere, fx marketingafdelingen eller kunden. Det kan yderligere vanskeliggøres af manglende kontakt mellem udviklere og aftagere (marketing, kunden eller potentielle brugere).
- Der er væsentlige krav der ikke formuleres eksplicit som en del af kravspecifikationen, fx krav til færdiggørelsestermin, ressourceforbrug, performance, acceptabelt fejlniveau, flytbarhed, udvidelighed.
- Man springer for let hen over krav der senere viser sig at blive meget omfattende, fx krav til brugergrænseflade og konfigurerbarhed.
- Man får ikke defineret forventninger til produktet af frygt for at stille for præcise krav. Fx risikerer man at kravene til vigtige egenskaber ved systemet (fx dimensionering eller performance) aldrig defineres præcist, fordi alle krav bliver af typen "så hurtigt som muligt". Her kunne man i stedet komme med nogle typiske beskrivelser af situationer systemet skal kunne klare (eksempel for et mobilradiosystem: antal opkald, antal knuder, varighed af samtaler). Den slags krav kan være nødvendige hvis man ikke i

designfasen skal afveje forskellige løsningsmuligheder i blinde.

- Man stiller ikke eksplicitte krav til værktøjer til vedligeholdelse, testomgivelser, belastningssimulation, osv.
- Kravene er ikke prioriterede eller sat i forhold til kundebehov og udviklingsomkostninger.

Selvom der er en kundespecifikation som grundlag, ser man mange af de samme fejl, idet man ikke klart formulerer kravene til det system man faktisk har tænkt sig at udvikle. Man ønsker som regel at udvikle noget mere generelt end det den første kunde kræver, men det bliver ikke defineret præcist nok.

Man ser også at de eksterne krav fra en kunde sammenblandes med de interne krav til produktet. Aftalerne med kunden kan være bevidst uklare: man lover mere eller mindre noget man godt ved man ikke kan holde. Det er en måde at få ordren og en måde at få gang i udviklingsprojektet. Eksempelvis talte vi med et firma der var utilbøjelige til at beskrive oppetidskrav, da man havde erfaring for at målingen af oppetid var meget afhængig af velviljen hos dem der betjente apparatet. I den slags situationer er det vigtigt at kravene stadig defineres internt.

Der foreligger ofte en løsningsstrategi sammen med produktidéen. Den omfatter typisk helt præcise specifikationer af det specialhardware man har tænkt sig at bruge til den lukkede del af systemet. Der kan også foreligge løsrevne beslutninger om at man fx vil køre Windows eller måske UNIX i det åbne system. På disse punkter lægger man sig således fast på en bestemt løsning, der ikke nødvendigvis er begrundet i noget krav. Et krav om en bestemt grad af flytbarhed kunne være et godt supplement i disse tilfælde.

Man ser også krav der ikke viser noget fornuftigt om systemet. Der er så stærkt et ønske om at få noget der kan testes objektivt, og så meget afsmitning fra de traditionelle metoder til at teste hardware, at man fx insisterer på at ryste apparatet i timevis. Det krav man havde brug for var måske, at det stadig skal fungere efter hensigten selvom der kommer en ny softwareversion - eller at man skal kunne rette en fejl og installere en ny version på x timer.

3.1.4 Løsninger

Generelt er der to holdninger til hvad det er der går galt i forbindelse med kravspecifikationen:

- Den ene skole siger at problemet er at man beslutter for lidt. Man skal se at få truffet nogle beslutninger og holde fast ved dem, selvom de måske ikke altid viser sig at være optimale.
- Den anden skole mener at man træffer sine beslutninger på for dårligt et grundlag. Den prædiker at man skal indrette sine udvik-

lingsmetoder med henblik på hyppige ændringer. Nøgleord er prototyper, spiralmodel, risikoanalyse [Note 10]. Idéen er at man skal udvælge dele af det totale system og implementere det: fx et rammesystem, et brugergrænsefladeforslag, de svære funktioner som man er usikker på. På basis af dette skal man så genoverveje sit mål og udvælge sig den næste etape. Metoden er ikke nødvendigvis hurtigere, men man mister i mindre grad kontrollen og fleksibiliteten.

Vi observerede ingen der konsekvent forsøgte at bruge den anden model, men den blev nævnt flere gange som en forventet bedre måde at tackle problemerne på.

Ofte er den første leverance til en specifik kunde, eller der leveres specielle udgaver til enkelte kunder. Her forsøger mange leverandører at opdele projekterne i en undersøgelsesfase, der betales separat, og et efterfølgende bindende tilbud om implementation. Idéen er god, men ikke iterativ nok. Når kontrakten er skrevet er man forpligtet - og berettiget - til en vis mængde arbejde. Det kan være svært senere at ændre dette til en mere formålstjenlig løsning.

Alt i alt er dette et område der er egnet til forskning. Internationalt foregår der meget på dette område [Note 11], men den eneste danske forskningsindsats vi har fået øje på er de matematiske metoder, der kun ser ud til at kunne bidrage meget lidt i denne fase.

3.2 Design

I designfasen skal man nedbryde det system der er beskrevet i kravspecifikationen i komponenter og beskrive komponenternes virkemåde og deres grænseflade. Hvis designet er godt, får man simple grænseflader og "pæne" komponenter. Når designfasen er overstået, forventes det at der foreligger en designspecifikation der kan danne grundlag for

- et designreview
- en detaljeret projektplan
- detailspecifikation og implementation
- testplan

Hovedparten af designspecifikationen er som regel almindelig tekst. Formelle metoder - fx VDM eller MetaIV - har vi ikke mødt nogen der har brugt, selvom mange kendte til dem. Vi har dog mødt et enkelt eksempel hvor KAITS var brugt.

Ofte bliver en væsentlig del af designet slet ikke beskrevet, men eksisterer blot som ikke nedskreven "fælles viden". Systemet designes normalt af nogle få udviklere - evt. kun en enkelt for mindre systemer.

Når vi har villet kigge i dokumenter, er netop designbeskrivelsen ofte fremhævet som et ømt punkt. De dokumenter vi har set har typisk

været forskellige former for modulspecifikationer eller dataflow-diagrammer, mens den overordnede struktur og begrundelsen for modulopdelingen savnes. Fx er opdelingen i parallelle tasks nok beskrevet, men ikke begrundet eller overvejet.

De objekt-orienterede tankegange vinder indpas som en måde at specificere modulerne på, men de løser ikke umiddelbart de overordnede designproblemer.

3.2.1 Problemer i designfasen

Specifikationens form

- Den dårlige kravspecifikation får nu konsekvenser: Den kan ikke bruges til at afgøre om man skal vælge den ene eller den anden løsning.
- Sporbarhed: De oprindelige krav kan ikke spores i designspecifikationen, så man er ikke sikker på at alle krav er dækket. Der er flere, der har udtalt at man ønsker sig et værktøj der understøtter dette.
- Man savner en god form: I mange af de projekter vi hørte om brugte man en del tid på et opfinde en form til at skrive specifikationer. Typisk beskriver man formatet for input og output til et modul, men ikke sammenhængen mellem input, tilstand og output.
- Man udskyder problemer man ikke kan overskue lige nu til senere, da der ikke er nogen regler for hvad der skal beskrives.
- De punkter man gik let hen over i kravspecifikationen går man også let hen over her: performancekrav, værktøjer til test, fejlfinding og simulation, brugergrænsefladen, konfigureringsmuligheder.

Nedenfor uddyber vi nogle af disse problemer.

Sammenhæng mellem delsystemer
Den opdeling der er bestemt af hardware-arkitekturen bestemmer hovedopdelingen på softwaresiden, men designet skal omfatte det samlede system, herunder både det "åbne" og det "lukkede". Designprocessen skal på den ene side opdele systemet i enkelte dele der udfører veldefinerede opgaver med veldefinerede grænseflader, på den anden side skal samspillet mellem delene opfylde de ydre krav. Hyppige problemer her er:

- Designet sikrer ikke at de centrale begreber i applikationen behandles konsistent i alle dele af systemet.
- Tabte ressourcer, baglåse, og flaskehalse analyseres ikke for det samlede system. De forsvinder ikke automatisk ved at indføre en transmissionslinje med en standardprotokol - og slet ikke fordi

man lader forskellige grupper implementere det.

Multi-tasking
For det lukkede system fastlægger man hvilke tasks der skal være og hvordan de kommunikerer. Kommunikationen mellem de lukkede og åbne systemer specificeres også.

Derimod har vi set flere eksempler på at man for den åbne del af systemet ikke overvejede hvilke processer, buffere, osv. der skulle være. Man stoppede ved beslutningen om at køre Windows - eller DOS - i tiltro til at drivere, kommunikation, og multi-tasking hermed havde en standardløsning - hvad de ikke havde.

Under integrationstesten dukker problemerne så op: Baglåse, sammenbrud ved spidsbelastninger, buffere der bliver væk, etc.

I nogle tilfælde havde man viden, så man kunne have løst problemet ved at eksperimentere eller ved at beskrive problemet og løsningen. I andre tilfælde har man måtte erkende, at man ikke havde fantasi til at forestille sig hvad der kunne gå galt.

Der tænkes således over nogle af sandtidsproblemerne, men ikke samlet og konsekvent.

Brugergrænseflader
Brugergrænsefladerne defineres ofte ad hoc af programmøren selv. De tilbagemeldinger der kommer er fra andre udviklere og - når det går højt - fra marketingafdelingen. Det skyldes tilsyneladende ikke uvidenhed om, at man kunne få meget bedre brugergrænseflader hvis man gjorde sig umage med at lave prototyper og teste dem med repræsentative brugere. Vi har dog set enkelte eksempler hvor man gjorde sådan.

Standardiserede grafiske brugergrænseflader er blevet meget populære som løsningsgrundlag. De giver en ensartet måde at betjene systemerne på, men de kan også gøre betjeningen stiv og besværlig. Og desværre kan det stadig være umuligt at finde ud af at betjene systemet.

Ofte er der ingen brugergrænseflade til de lukkede systemer udover nogle lamper og evt. simple testterminaler til udviklere/teknikere. Hvis der er en egentlig brugergrænseflade, er den typisk baseret på knapper, lamper, simple displays, og sjældnere skærm og tastatur. De problemer der kan konstateres er i høj grad de samme som for skærmbaserede brugergrænseflader: De er lavet af teknikere for teknikere uden erkendelse af brugernes forskellige behov.

Vi har fx hørt en gribende beretning om kølerumspersonale der skulle vende deres apparat midt i en indviklet knaptryksekvens for at læse vejledningen på bagsiden (og så var den tilmed trykt med små hvide bogstaver på sort baggrund). Et andet, velkendt eksempel på et apparat

der ofte er urimeligt indviklet at betjene er en videobåndoptager.

Datakommunikation og andre standardgrænseflader
De eksterne protokoller og grænseflader defineres tilsyneladende pænt med tilstandstabeller, syntaks-notation, eller undertiden objektorienteret. Ikke desto mindre viser det sig ofte under testen at der er problemer med at få dem til at virke.

Et andet problem man kan konstatere er, at det er svært at vælge de rigtige protokoller og snitflader. Det er ikke nok at vælge standard - man skal også kende modetendenserne indenfor sideordnede standarder. Desuden er det ofte meget vanskeligt at skaffe beskrivelser af det der udråbes til standarder et eller andet sted.

Forudsigelser af performance
Der foretages meget få beregninger på forventet performance. Den gennemgående holdning er "at det kan man ikke forudsige - man må vente til det kører".

- Simple beregninger af svartider eller cpu-forbrug er sjældne, selvom de er ret enkle at foretage.
- Man regner ikke på spidsbelastninger.
- Der er ingen der gør noget særligt ud af simulation.
- Der er ingen der opstiller matematiske modeller, selv ved ret avancerede løsninger.
- Køteoretiske overvejelser - som ofte kunne være yderst relevante - bruges ikke.
- Man regner ikke på performance af ny-designede lukkede systemer: Hvis processoren er hurtigere og der er lagt flere funktioner ud i special-hardware, stiller man sig tilfreds med det, selvom hele softwarearkitekturen er ny.

Resultatet af den manglende performance-forudsigelse er, at man ikke på forhånd ved hvor skoen trykker. Derfor kan man ikke vælge et design der undgår problemet. Når problemet viser sig sidst i testfasen, kan man kun lave lokale småforbedringer.

Planlægning af konfigurerbarhed
Produktet skal typisk på længere sigt kunne konfigureres svarende til forskellige kunders krav. Det er sjældent med i designet og heller ikke i første version af produktet.

Konfigurerbarheden kan sagtens være en del af kravspecifikationen - det er nemlig meget populært og nemt at stille krav som fx "alting skal være konfigurerbart" og at systemet "selv skal kunne finde ud af hvordan det er konfigureret når det bliver tændt".

Den slags krav kan kun opfyldes, hvis de er designet med ind i sy-

stemet fra starten - og de sparer en masse tid når først de er på plads. På den anden side kan overdreven dynamisk konfigurering nemt vise sig at sluge meget mere tid end man ønsker at ofre på det.

I de eksempler vi har gravet i har der været ambitioner om at lave en pæn løsning, men typisk var den i hvert fald ikke med fra starten, og det var heller ikke helt klart hvordan den skulle se ud i detaljer.

Ofte ser man at der efter første leverance bliver flere og flere varianter af produktet, og en facilitet der udvikles til én variant kan ikke uden videre indbygges i de andre. Det bliver særligt slemt når sælgeren tilbyder sådanne hybride løsninger, fordi han tror at de findes eller let kan laves. Koordineringen af varianter overvejes sjældent på design-tidspunktet, og som regel heller ikke når man begynder at lave dem.

Brug af specifikationsværktøjer
MetaIV er et værktøj der er ret kendt i apparatindustrien, hvor mange har lært at bruge det i deres uddannelse. Vores stikprøver viste ingen anvendelse af det. Dog hørte vi om et sted hvor der var én der brugte det til visse ting for sig selv - men kollegerne sagde at "det var i øvrigt en meget fornuftig person, så det var svært at vide om det var hans fornuft eller metoden der hjalp ham".

Begrundelserne for den manglende brug er forskellige. Nogle steder tror man ikke på metoden, andre steder ville man egentlig gerne, men der er bare aldrig lige en opgave der passer. Konkrete indvendinger går på at det giver dobbeltarbejde at skulle formulere alting en gang til i et helt andet sprog med tilhørende muligheder for at lave flere fejl.

KAITS bruges heller ikke. Ved ihærdig søgen fandt vi dog en projekt-gruppe på to i et firma, der i øvrigt har gjort en ihærdig indsats for at oplære sine medarbejdere i de bedste metoder de kunne finde frem til. De to havde udviklet et lukket system (et enkelt kort), som del af et større system. Resten af projektet havde opgivet at bruge KAITS, men de to havde holdt ved og var meget tilfredse med metoden. Som den væsentlige fordel nævnte de, at systemet kørte fejlfrit fra starten. Resten af virksomheden var ikke så begejstret. Enten kendte de slet ikke metoden, eller de mente at den var uinteressant, bl.a. fordi projekt-gruppen havde brugt lang tid.

Prototypeværktøjer til brugergrænseflader anvendes flere steder. De vurderes som nyttige.

3.3 Programmering

Selve programmeringsfasen anses som regel for ret hurtig og problem-fri.
Programmeringen betragtes for afsluttet med en eller anden form for modultest. Flere steder nævnes at man havde tænkt sig at gøre mere

ud af modultesten, men i praksis ender det med lidt ad hoc test.

Detaildesign og programmering af forskellige dele af systemet kører ofte parallelt og ret uafhængigt.

C og C++ er helt dominerende som programmeringssprog, men ingen af dem vi har snakket med har endnu væsentlige erfaringer med C++ (udover bøvl med oversætterfejl).

Der er en udbredt tiltro til at objektorienteret design og programmering er godt. Vi tilskriver det dels en ret udbredt tradition for undervisning i objektorienterede sprog på uddannelserne, og dels en ny - mere modepræget - tiltro til C++ som det sprog man vil satse på. Vi har ikke talt med nogen der har lavet noget færdigt i C++, men adskillige der er gået i gang med at bruge det. Som kuriosum kan nævnes, at stort set alle de forskere og formidlere vi har snakket med har været skeptiske overfor hvor velegnet C++ er til den slags opgaver - og det uanset om de i øvrigt selv troede på objektorienterede metoder eller ej.

Tidligere tiders udstrakte brug af assemblerkode er nu erstattet med C, undtagen i helt specielle tilfælde. C er så dominerende på markedet at man vælger at se stort på fordelene ved andre programmeringssprog. C++ må ses som et forsøg på at komme videre end C uden at brænde sine broer.

Der anvendes også en række værktøjer og biblioteker til fx databaser, brugergrænseflader, og datakommunikation. Der er et udtalt, fornuftigt ønske om at slippe for at skrive kode der allerede findes.

3.3.1 Problemer

En del tyder på at grunden til at programmeringsfasen går så nemt, er at mange fejl først findes i integrationsfasen eller senere. Mange af fejlene - også de der findes sent - er i øvrigt ret banale kodefejl som indeksfejl og uinitialiserede variable. Hvis man i stedet for C brugte sprog og oversættere med streng typekontrol, initialiseringskontrol, indekscheck, stakcheck og tilsvarende teknikker, der har været kendt og udnyttet de sidste 20 år, kunne mange af den slags fejl undgås, fanges af oversætteren eller i det mindste være nemme at finde og rette når de optrådte.

Vi har flere steder set at sporbarheden til design-specifikationen helt mangler. Alle de "moduler" der blev fastlagt ved designet findes ikke i koden (eller detailspecifikationen), og det hele er reelt blevet tænkt om igen. Vi uddyber det i afsnittet om metoder, side 37.

I forbindelse med programmering - og i øvrigt også på de fleste andre punkter vi her kalder programudvikling - er den dygtige håndværker én der kender en række forskellige teknikker der kan bruges til at løse en række forskellige problemer - og samtidig ved han meget om hvad der

plejer at gå galt. Vores fornemmelse er at det er svært at blive en dygtig håndværker i dette fag. Det er et punkt der i nogen grad falder tilbage på uddannelsessystemet, men også på manglende erfaringsudveksling mellem grupper i virksomheden. Vi har fx bemærket en ret begrænset teoretisk viden om multiprogrammering og ressource-administration hos iøvrigt meget erfarne og kompetente udviklere.

3.4 Test og integration

3.4.1 Systemintegration

Programmeringsfasen glider mere eller mindre veldefineret over i en systemintegrationsfase. Denne fase omfatter den tid hvor systemet afprøves i sammenhæng på det rigtige hardware. Den afsluttes typisk når systemet er godt nok til at den første kunde kan bruge det. Det betyder ikke nødvendigvis at der ikke er flere fejl, men snarere at systemet kan køre nogenlunde stabilt, når det bruges i normal drift.

Nogle steder i udlandet bruger man en arbejdsdeling hvor udviklerne afleverer deres kode til nogle andre personer, der står for integrationstesten. Det giver ikke gode resultater, og heldigvis har vi ikke hørt om det i Danmark.

Der går uventet meget tid i denne fase af mange forskellige grunde:

- Misforståelser mellem de forskellige programmører pga. en uklar eller manglende designspecifikation.
- Misforståelser med kunden pga. en uklar eller manglende kravspecifikation.
- Nedbrydningen var forkert. Det viser sig måske nu, at man ikke havde lavet opdelingen i tasks rigtigt, og det skal rettes en masse steder.
- Performance-problemer, flaskehalse uventede steder.
- Real-time fejl: baglåse, bufferoverløb.
- Kommunikationsprotokollerne virker ikke når de prøves af mod "virkeligheden" - måske på grund af fejl i ens eget program, men ofte også fordi der er fejl i det man skal kommunikere med.
- Det rigtige specialhardware er først til rådighed på dette tidspunkt, så der er mange ting der først kan testes her. Hardwaren virker anderledes end forventet (specifikationen af chippen var fx forkert eller misforstået). På dette tidspunkt kan problemerne kun løses ved at tilføje omgåelser i programmerne.
- Banale kodefejl kan være meget svære og tidskrævende at finde - ikke mindst da man alle steder koder i C.
- Systematisk aftestning mod kravspecifikationen giver anledning til

at man nu finder fejl og misforståelser.

- Dårlige arbejdsbetingelser: Der laves flere dumme fejl, og de er sværere at finde, når der arbejdes under tidspres i længere tid.
- Mangel på testressourcer kan også betyde at man spænder ben for hinanden i projektgruppen. Det kræver både gode arbejdsmetoder og høj arbejdsmoral at undgå konflikter.
- Manglende disciplin: I paniksituationer gribes der nogle gange til hurtige løsninger. Det kan give bagslag - fx er versionsproblemer meget dominerende i denne fase: "Jamen var det ikke den fejl vi fandt ude hos kunden i sidste uge? ". Dokumentationen bliver ikke opdateret.

3.4.2 Aflevering, accept-test

Hvis der leveres til en specifik kunde, foreligger der ofte en accept-testplan. I mange eksempler vi har set har der været sporbarhed, så man kan følge kundens krav fra kravspecifikationen til accepttesten.

Der er ofte masser af problemer alligevel. Det skyldes bl.a. at specifikationerne som tidligere nævnt ikke er gode nok, og det giver anledning til diskussioner, som ikke bliver nemmere af at tidsplanen også ofte er skredet. Resultatet er at systemet bliver afleveret selvom det ikke er helt færdigt - og det giver ofte nye problemer, fordi udviklerne - der allerede er stressede - nu ved siden af at lave systemet færdigt, også skal installere et system hos kunden, hvilket giver nye problemer: Nye hardware-konstellationer, nye grænseflader, brugere der anvender systemet anderledes, versionsproblemer, performanceproblemer på grund af uventede belastningsmønstre - og endnu mere pres på testressourcerne.

3.5 Vedligeholdelse - genbrug - flytbarhed

3.5.1 Gamle systemer

Vedligeholdelse af gamle systemer er et irritationsmoment præget af uforudsete udgifter, utilfredse kunder, uvillige udviklere, og dårlige løsninger, bl.a. fordi ingen tør lave radikale ændringer i noget der kører nogenlunde og som man kun delvis forstår.

De gamle systemer er ofte endt i et virvar:

- Folkene er væk
- Dokumentationen er ikke ført ajour (hvis den blev lavet)
- Der er gjort vold på det oprindelige design, fx med rettelser overalt der vedrører enkelte kunders specielle konfiguration.
- Hardwareplatformen er død: Der kan ikke skaffes reservedele,

man kan kun teste hos kunderne, osv.

- Softwaren kan ikke flyttes. Måske findes sproget ikke andre steder, men selv om det gør (fx ANSI C), er det meste af koden speciel for det aktuelle hardware og det aktuelle styresystem, så det er kun viden om applikationen og måske nogle kodestumper der kan flyttes.
- Fejlsøgningen og rettelser har været panikprægede og er måske udført af personer med ringe kendskab til systemet.

For nye systemer forsøger man at foregribe disse kendte problemer ved en indsats der som oftest ligger i forlængelse af en hektisk afleveringssituation. Opfølgningen går ud på at få dokumentationen - og ikke mindst specifikationerne - rettet op svarende til virkeligheden.

Ved at lægge funktionalitet over i den åbne del af systemet, håber man også på at få noget software der er nemmere at vedligeholde. Dels er der flere udviklere der kender platformen, og dels kan man håbe på portabilitet for sprog, standardprogrammer, og styresystemer. Under alle omstændigheder vil mange af problemerne være nogle man deler med mange andre.

3.5.2 Dokumentation

Det væsentligste problem med dokumentationen er at få den lavet. Udviklerne er måske motiverede en kort periode efter færdiggørelsen, men ofte venter der nye (allerede forsinkede) projekter eller udskudt ferie. Dokumentationsarbejdet er det der bedst kan udskydes. Resultatet er velkendt - det bliver aldrig lavet. Her kunne det være, at en mere udbredt anvendelse af stringente metoder og kvalitetssikring kunne komme til hjælp.

Endelig må man nævne, at det langt fra er klart hvad dokumentationen skulle omfatte - hvis den blev lavet. Meget af den dokumentation der faktisk er blevet lavet samler blot støv. De der skal vedligeholde systemet savner en anden form for dokumentation, men det er uklart hvad. Erfaringen viser, at den dokumentation der faktisk bruges, er den der står i selve koden.

3.5.3 Genbrug af kode

Det viser sig i praksis at planlagt genbrug af kode er svært. Udover teknik og erfaring kræver det en fornemmelse for hvor meget generalisering der kan betale sig. Vi har set eksempler på at man i designfasen har brugt mange kræfter på at lave et generelt design, der kan tåle visse typer ændringer og udvidelser, men de faktiske ændringsønsker, der kommer senere, viser sig at være af en anden art.

3.6 Projektledelse

Projektledelsen går ud på at afvikle et projekt - svarende til en given specifikation - indenfor rammerne af et budget (nogle medarbejdere) og en tidsramme.

3.6.1 Tidsplaner

Projektplanen bygger på nogle tidsplaner, der igen bygger på nogle tidsestimater for de enkelte aktiviteter.

Projektgruppen fremkommer med nogle tidsestimater, der efter diskussion med ledelsen bliver til en officiel tidsplan for projektet. Selvom alle parter betragter det som "bedst mulige gæt" er tallene i tidsplanen i virkeligheden resultatet af en forhandling, hvor den enkelte udvikler fortæller hvornår han tror han kan være færdig.

Denne metode sikrer at han føler sig presset til at blive færdig - ikke nødvendigvis på det estimerede antal arbejdstimer, men måske til den rigtige dato. Hvis det ikke er tilfældet så gentager man processen: Hvornår er du **så** færdig? Og så er det psykologiske pres større. Metoden gentages hierarkisk mellem projektledelsen og de enkelte (del)projektgrupper. Visse steder praktiseres den sådan at man kun forhandler datoen - og overlader til projektgruppen eller den enkelte at erlægge så meget arbejde som nødvendigt.

Denne metode anses som oftest for rimelig set fra begge parters synspunkt, så længe forholdene i øvrigt er rimelige: Ydre hændelser og fejl man ikke er herre over skal accepteres som lovlige undskyldninger for ikke at blive færdig til tiden. De meget pressede perioder der opstår i forbindelse med afleveringer og delafleveringer, forventes i nogen grad at modsvares af perioder hvor der er tid til at rydde op, uddanne sig, osv.

Ulempen ved metoden er at estimaterne bliver for optimistiske. Der er meget ønsketænkning, for man ved godt hvor man gerne ville ende.

Erfaringstal findes sjældent, men der er flere der nævner at de er i gang med at samle data op fra de nuværende projekter.

Der er tilsyneladende ingen der sammenligner deres tal med de tommelfingerregler der findes, fx antal kodelinjer pr. arbejdsmåned, hvor meget af den samlede tid der bruges til en bestemt fase, antal fundne fejl pr. linje.

Tidsplanen bliver således i praksis et diktat eller et forhandlingsresultat, snarere end en konsekvens af det man vil udvikle. Årsagen er at det vigtigste overordnede krav er at få produktet leveringsklart til tiden, men man ser ikke i øjnene at det så begrænser hvor omfattende pro-

duktet kan blive.

Myten går på at softwareudviklere alligevel aldrig kan gøre sig færdige, så derfor bliver færdiggørelse en kvalitet som firmaet i virkeligheden ikke ønsker at betale for, selvom det sjældent formuleres eksplicit. Ledelsens interesse er at få noget færdigt der er "godt nok". Som afskrækkende eksempel nævner nogle det firma hvor udviklerne havde så meget magt, at de fik udsat produktleveringen så længe at konkurrenterne overhalede det. Firmaet findes ikke mere.

Nogle formidlere hævder at SPU-metoden sikrer at man bliver færdige til tiden, men det holder bestemt ikke i praksis.

3.7 Kvalitetssikring

Der er et udbredt ønske om at blive ISO 9000 certificeret, og mange er igang. Den ene grund er forventede kundekrav. Den anden grund er ønsket om at få styr på kvaliteten af softwareudviklingen. Der er nogle udviklere der ser det som en metode til at sikre sig, at man ikke får hevet halvfærdige produkter ud af hænderne.

Selve processen med opbygning af kvalitetshåndbøger og analyse af hvad man gør og kunne gøre, anses for frugtbar og meget lærerig. Nogle firmaer har set det som anledning til at lære nyt og indføre nye metoder på en række områder, hvor det ellers aldrig ville blive gjort.

Bagsiden er at kvalitetssikring let kommer til at give nogle meget stive metoder, og det kan blive et problem, for man hænger på dem. Hvis de ikke overholdes, risikerer man at miste sin certificering. Det kræver derfor løbende arbejde at vedligeholde dem, så de passer til en fornuftig praksis.

Et andet problem er at stive metoder og bureaukrati kan mindske den enkeltes ansvarlighedsfølelse, samtidig med at de uoverskuelige mængder af papir kan skjule for alle involverede at der i virkeligheden ikke er spor styr på kvaliteten. Dette har vi faktisk set eksempler på.

Endelig har vi set, at man udmærket kan få ISO 9000 certificering uden at bruge en fornuftig metode. Blot man gør hvad man har beskrevet, er det i orden. Ulempen er, at man nu har fået en sovepude, selvom alt er som det plejer.

3.8 Metoder og værktøjer

En metode er en samling overordnede principper der får hele udviklingsprocessen til at hænge sammen. Den vil bl.a foreskrive at en række nærmere angivne teknikker bruges i de forskellige faser.

Der er tilsyneladende en udbredt, men ret ny, erkendelse af at man har brug for metoder til udvikling af apparatsoftware. Det skyldes nok til dels dårlige erfaringer, men også tendensen mod ISO 9000, hvor det bliver væsentligt med velbeskrevne metoder.

SPU-metoden er meget udbredt blandt de firmaer vi har besøgt. Mange har været på kurser, og mange taler om metoden som en referencemodel.

I praksis siger de fleste af dem vi har snakket med, at de ikke følger metoden slavisk, men bruger den som en fælles basis, hvorfra man definerer sin egen metode. Typisk fylder man detaljer på i form af dataflow mv. som i struktureret analyse og design, eller - i nyere forsøg - i form af objektspecifikationer.

Objekt-orienterede metoder er meget populære blandt udviklere, formidlere og forskere, men det er endnu svært at se hvad det betyder i praksis - bortset fra at C++ er blevet meget udbredt som programmeringssprog.

Tilfredsheden med SPU og SPU-lignende metoder er dog ret blandet. De positive kommentarer går på at man har fået en metode og føler man går den rigtige vej. De negative går på at metoden ikke er god nok af forskellige grunde.

Nogle indvendinger går på at metoden ikke er operationel: Man skal selv omsætte den til noget brugbart. Det er en opgave formidlerne er ved at tage op. Fx har DTI i Århus udbygget metoden med en kursusbog for hver fase, men materialet er fortroligt og vi har ikke haft lejlighed til at se det.

Det viser sig også, at det at bruge SPU ikke er nogen garanti for at udviklingen går hurtigere eller er mere forudsigelig.

Andre kritikpunkter er, at der bruges meget tid på at producere papir, og at mange af de dokumenter der produceres er overflødige. De fleste andre indvendinger går på den stive opdeling i faser. Det er lidt svært at se om disse kritikpunkter er berettigede. Måske giver metoden en hjælp, men virksomheden rykker så op på et nyt modenhedsniveau og får øje på andre problemer [Note 12].

Fase-problemet og sporbarhed

Fase-problemet formuleres på forskellige måder, men det har alt sammen at gøre med det sekventielle forløb. Metoden dikterer et sekventielt forløb hvor man laver hver fase helt færdig før man fortsætter, men der er tydeligvis behov for at kunne eksperimentere, lave prototyper, lave visse dele af systemet - og så vende tilbage og lave om i noget der hører til en tidligere fase.

De der følger metoden slavisk har dels konstateret at der er meget af det der er lavet tidligt som skal laves helt om, fordi det alligevel ikke skulle være sådan. De konstaterer også, at de får blandet krav og løsningsforslag i de samme dokumenter, og de konstaterer at det er svært at holde dokumenterne opdaterede svarende til de ændringer der sker undervejs. Det hænger bl.a. sammen med at sporbarheden fra fase til fase er utilstrækkelig: Det hele tænkes ofte om fra fase til fase.

Når man så er færdig, ligger der et stort stykke arbejde i at få alle de tidlige dokumenter bragt i overensstemmelse med virkeligheden og på den rigtige form.

Vi har kun set ét projekt baseret på SPU hvor sporbarheden så ud til at være i orden. Dokumentationen var her baseret på dataflow-diagrammer som kunne spores fra krav/design til færdig kode. Til gengæld havde man observeret at dokumentionen og bureaukratiet var blevet for omfattende og uflexibelt.

Såvidt vi har kunnet se, er det der er gået bedst en ganske simpel teknik fra før SPU: Man specificerer modulerne i form af dataerklæringer og funktionserklæringer med tilhørende kommentarer. Det betyder at design-specifikationerne kan genfindes i selve koden, og derfor bliver de også læst og vedligeholdt. Flere virksomheder har med succes gennemført dette gennem mange år af et produkts levetid. Teknikken er nok lidt i strid med et af idealerne fra SPU og diverse metode-guruer: "Man skal beskrive *hvad* man vil have gjort, men ikke *hvordan*". På den anden side ser teknikken ud til at virke.

Værktøjer og ønsker

Vi har hørt mange udtrykke ønske om bedre metoder og værktøjer til systemudvikling. Ønskerne om bedre udviklingsmetoder formuleres ret kraftigt: Vi vil gerne gøre det bedst muligt denne gang, men hvad er den rigtige metode?

Én type ønsker går på understøttelse af spiralmodeller, risikoanalyse og andre måder at tilgodese tilbagemeldinger og løbende ændringer undervejs i udviklingsprocessen.

En anden type ønsker går på værktøjsunderstøttelse: Man ønsker ikke at skrive det samme mange gange på forskellig form. Man ønsker et CASE-værktøj der understøtter at kravspecifikationen kan genbruges i design- og testspecifikationer - og helst også i koden og i automatisk genererede test-cases, så man slipper for kontrolarbejde og fejlmuligheder, og så man bevarer sporbarheden undervejs.

De værktøjer der findes er ikke professionelle. De løser kun et hjørne af opgaven, og de passer ikke sammen med andre værktøjer på markedet.

De lukkede systemer, der indgår i de fleste apparater, sætter grænser for hvilke værktøjer der kan bruges, fordi der er tale om små CPU'er med beskedent lager og specielle programmerbare enheder. I mange tilfælde er man henvist til de værktøjer hardwarefabrikanten tilbyder.

Der er en kraftig tendens til at sætte lighedstegn mellem værktøjer og metoder [Note 13]. Det betyder bl.a. at leverandørerne af PC-produkter bliver afgørende for hvilke metoder der kan benyttes. Det gælder integrerede programudviklingsmiljøer, projektstyringsværktøjer, versionsstyring og meget andet.

Formidlerne klager over at firmaerne kun vil have værktøjer, men ikke er villige til at følge de tilhørende metoder. I mange tilfælde starter man på at bruge en metode, men opgiver på halvvejen. Vi har selv observeret tilfælde hvor man siger at man bruger metode X, men når vi så beder om at se nogle af udviklingsdokumenterne, viser det sig at man bare gør som man plejer. Man forklarer det med at man ikke har haft tid lige nu, men når man er færdig med programmeringen har man tænkt sig at lave de manglende dokumenter!

3.9 Organisation og personale

Følgende iagttagelser er med til at tegne et billede af forholdene i apparatindustrien.

3.9.1 Personer

Apparatfirmaerne har ofte en udviklingschef. Han er typisk civilingeniør (eller sjældnere cand.scient). Hans viden om software stammer fra "noget programmering engang", og han har ikke selv nogen detaljeret indsigt i nyere teknikker, men forventer at kunne overlade det til sine medarbejdere.

Større firmaer kan have en metode-guru, der er den nøgleperson som samler viden om nye teknikker og metoder. Han deltager i alt koordinerende arbejde (QA, reviews, metodediskussioner). Eksternt vil det være ham der sidder i standardiseringskommiteer og forskningsfølgegrupper. I større firmaer kan det være en gruppe snarere end én person. Der kan være stor divergens mellem den viden og holdning der er i metodegruppen og den der er hos den menige udvikler.

Med undtagelse af meget små firmaer, vil der være et niveau i organisationen bestående af projektledere og/eller gruppeledere. Det er udviklere med en passende mængde erfaring og evner der placeres her. Det er ikke nødvendigvis en fast rolle og det er normalt kombineret med udviklingsarbejde. Lederuddannelser og personlighedsudviklingskurser, som er blevet ret populære i andre typer virksomheder, har vi ikke hørt nævnt i apparatindustrien.

Den typiske udvikler er en eller anden slags ingeniør, men andre uddannelser forekommer også. Kvinder og folk over 40 er undtagelser. Da mange ansættelser i branchen sker via jungletrommerne, ser man en vis sammenklumpning af folk med samme alder og uddannelse.

Der har tidligere været en generation af mere eller mindre selvlærte software-udviklere. Nogle har antydet at der kommer en ny: PC-freaks, der er eksperter i at lave smarte vinduessystemer, men ikke har en bredere software-uddannelse. De har en endnu højere tærskel over for "matematiske" metoder end andre, og de mangler viden om nødvendige grundteknikker som fx multi-tasking, test, versionsstyring osv.

3.9.2 Projektgrupper

Medarbejderne er som oftest fast allokerede til projekter for længere tid ad gangen, typisk ½-2 år. Vedligeholdelsesarbejde stikkes ind imellem projekterne. Fejlretning i gamle systemer kommer nogle gange pludseligt og med højeste prioritet og kan virke meget forstyrrende.

Denne organisation skal ses i modsætning til den der stadig findes visse steder i den administrative verden, hvor man i højere grad opererer med planlæggere, programmører, vedligeholdere, brugerservice-konsulenter, osv.

Projektgrupperne er normalt organiseret med en projektleder og evt. nogle undergrupper. Typisk opbygges den omkring nogle få gode kernefolk, og så bliver gruppen velegnet til oplæring af nye medarbejdere.

Projektgruppen arbejder meget selvstændigt, og ofte er der næsten ingen kontakt til andre projektgrupper. Projektgruppen arbejder tæt og uformelt sammen, og ofte ses en stærk følelse af fællesskab og engagement i produktet. Det er måske den væsentligste drivkraft i udviklingsarbejdet.

Små projekter, hvor én person arbejder alene, anses ofte for mindre attraktive. Udvikleren savner modspil, og ledelsen savner dokumentation af såvel design som af det færdige resultat. Vedligeholdelsesarbejde rangerer lavest - specielt vedligeholdelse af gamle systemer der er udviklet af andre.

Forskellige projektgrupper kan køre ret forskelligt og ret lukket overfor hinanden. Udvekslingen af metoder og erfaringer sker tit først når medarbejderne bliver sat sammen på nye projekter.

3.9.3 Personalegrupper

Vi har hørt nogle antydninger af at samarbejde mellem hardwarefolk og softwarefolk kan være svært pga. organisatoriske og kulturelle forskelle. Det er værst hvor der er forskellige afdelinger - specielt hvis de skal aflevere til hinanden: hardwareudvikling til softwareudvikling, eller

udvikling til produktion. Hvis der er tættere samarbejde, går problemerne mest på forskellige traditioner og metoder.

I nogle virksomheder har vi hørt software-folk sige at hardware-folk ikke har styr på metoder og specifikation. Og i andre virksomheder har vi hørt hardware-folk sige det samme om software-folk. Noget tyder på at de to parter kunne lære af hinanden.

3.10 Teknologioverførsel - hvordan kommer ny viden ind i firmaerne?

Teknologioverførsel inden for software er et helt centralt problem for apparatindustrien, da det er afgørende for forbedring af produktiviteten. Vi vil se på kilderne til ny viden, og de barrierer der er ved indførelse af noget nyt.

3.10.1 Hvorfra får man ny viden?

Følgende nævnes som kilder til ny viden:

- Konsulenter
- Nye medarbejdere (nyuddannede eller fra andre virksomheder)
- Videreuddannelseskurser
- Tidsskrifter - danske og udenlandske
- Leverandørkurser, seminarer osv.
- Lærebøger, værktøjer/manualer
- Foredrag i faglige foreninger eller ERFA-grupper
- Personlige kontakter - herunder private
- Interne og eksterne kollokvier eller studiekredse
- Rapporter fra formidlere eller forskere
- Medvirken i forskningsprojekter

Det er svært at vurdere hvad der giver bedst videnoverførsel, men vi har flere udsagn om at metoder bedst indføres af konsulenter der i en længere periode giver "on-the-job" træning.

Vi har også hørt at rapporter er den mindst effektive måde at overføre viden på.

3.10.2 Problemer og barrierer ved indførelse af nyt

Nedenstående er en liste af mange forskellige mekanismer der - mere eller mindre uafhængigt - modvirker indførelse af nye teknikker og metoder i apparat-software.

- Uddannelse nedprioriteres af medarbejdere og ledelse, fordi der altid er tidspres.
- Uddannelse sker kun på eget initiativ. Der er ingen tilskyndelse i form af fx karriereplan eller uddannelsesplan.
- Holdning: Vi er gode, vi kan ikke lære noget af de andre, det er dem der kan lære af os.
- Holdning: Det er ikke svært at lave software. Software er det samme som programmering, og det kan vi jo allesammen efterhånden.
- Holdning: Med alle de fine nye værktøjer må det da være nemmere at lave software. Forkert: kompleksiteten og ambitionsniveauet er steget mere end der spares.
- Softwareudviklere er ret konservative - og vant til selv at bestemme deres metoder.
- Artikler om nye teknikker - specielt med matematiske formuleringer - er noget man kommer ud af vane med at læse.
- Nyuddannede ved måske en masse, men det kan vare længe inden de får gennemslagskraft nok til at deres viden udbredes.
- Kurser i konkrete teknikker man står og har brug for virker godt nok, men når enkelt-personer kommer på mere generelle kurser fungerer det ikke særlig godt. Hvis man ikke bruger det man har lært og ikke kan snakke med nogen om det i firmaet, bliver det nye bare noget der står på hylden når man kommer hjem - **uanset hvor godt det var**. Formidlerne snakker om et typisk forløb på 2 år for indførelse af en ny teknik: Først sender firmaerne enkelte teknologispejdere, og hvis de så beslutter for alvor at hoppe på vognen, får man lavet et firmakursus eller sender alle medarbejdere afsted.
- Tidshorisonter: Formidlerne påstår at der går to år før nye metoder slår igennem. Det skal ses i sammenhæng med at produkterne holder 10 år. Et produkt er typisk bundet til de samme metoder og værktøjer i hele sin levetid. Det betyder at der er folk i virksomheden der er tvunget til at arbejde med ret forældede teknikker.
- Behovshierarkiet: Når man har andre - mere presserende - problemer er der ingen interesse for at løse de lidt mere langsigtede [Note 14]. Det gælder *til trods* for at man har behovet og at der findes løsninger.
- Manglende risikovillighed: Man tør ikke prøve noget nyt. Det er for dyrt, og fordelene er alt for tvivlsomme og langsigtede.
- Som specielt risikofyldte betragtes nye teknikker, der ikke er kompatible med andre teknikker og værktøjer. For man kan jo ikke skifte tilbage uden at have mistet det hele. Fx er C++,

objektorienteret design, og SPU acceptable, mens KAITS og Mjølner ikke er det.

- Teknikker med høj indlæringstærskel betragtes også som en større risiko. Hvis man satser forkert, hænger man på at skulle uddanne folk i metoden i lang tid fremover for at kunne vedligeholde sit produkt.
- Teknikker og metoder der er lavet i USA af et stort firma anses for at være bedre end andre. Holdningen er yderst forståelig, men det er bestemt ikke oplagt at den øger ens egen konkurrenceevne. Værktøjer sælges i dag mere på markedsføringen end på kvaliteten - og kvaliteten af nogle af de udbredte produkter er overraskende ringe - endda på grænsen af de pålidelighedskrav man må stille i apparatindustrien.

4. Dansk forskning

I rapporten har vi tidligere sagt, at der ikke er ret meget dansk forskning som umiddelbart er relevant for apparatindustriens software-udvikling. Men faktisk er der meget dansk forskning på softwareområdet, og en stor del af det er grundforskning der senere kunne gå hen og blive relevant.

I dette kapitel giver vi en kort oversigt over projekter vi er stødt på gennem vores besøg eller på anden vis. Vi kan sagtens have overset nogle relevante, men vi mener dog selv at vi har de væsentligste med.

Desuden præsenterer vi vores observationer om forskerens rolle og holdninger til industrien, samt industriens og formidlernes holdning til forskerne.

4.1 Eksempler på forskningsprojekter

4.1.1 DAIMI, Århus Universitet

Mjølner

Mjølner-projektet har udviklet en metode og tilhørende værktøjer, der dækker design, programmering og test. Metoden er i princippet meget relevant for apparatindustrien. (Den er nærmere omtalt på side 13).

VLSI

En gruppe arbejder med VLSI-design i samarbejde med en stor virksomhed.

Petri-net

Petri-net er en måde at specificere funktionaliteten på. Teknikken er specielt god til at opstille og bevise invarianter. Den er på DAIMI blevet videreudviklet så den har et meget større anvendelsesområde. Det er interessant at projektet ikke kunne finde partnere og økonomisk støtte i Danmark, men via USA blev det til et værktøj, der nu bruges flere steder.

Andet

DAIMI har flere andre projekter, bl.a. et visionært projekt i tæt samarbejde med en stor virksomhed om fremtidens apparat-anvendelser.

4.1.2 DIKU, Københavns Universitet

Distribuerede, objekt-orienterede systemer

DIKU har igennem mange år arbejdet med objekt-orientering, specielt objekt-orienterede operativ-systemer hvor objekterne kan flyttes fra en maskine til en anden i et netværk. Der lægges stor vægt på en høj performance. Projektet er nu i gang med samarbejde med en industriel partner i EF.

Partiel evaluering

Partiel evaluering går ud på at tage en generel algoritme og "fryse" en del af dens input. En compiler genererer så automatisk en langt mere effektiv algoritme der kan håndtere resten af input. Som et praktisk eksempel kan nævnes billedgenerering for komplekse 3-dimensionale scener. Den 3-dimensionale scene er input der "fryses", og den optimerede algoritme kan nu meget hurtigt genere billeder svarende til forskellige perspektiver.

DIKU har leveret et afgørende gennembrud på området ved at kunne håndtere partiel evaluering af en realistisk delmængde af C. Herved har metoden fået praktisk relevans.

Billedbehandling

DIKU har leveret mange resultater vedr. automatisk analyse af billeder, fx en automatisk rekonstruktion af den 3-dimensionale scene ud fra to billeder fra forskellig vinkel. Der er gode industrielle muligheder, men det falder uden for denne rapports emne.

Andet

DIKU har flere andre projekter, fx om beregninger fordelt på mange parallelle maskiner, og om systemudvikling mere generelt.

Generelt har DIKU en rimelig erhvervskontakt, men ikke specielt til apparatindustrien. Man vil gerne have sådanne kontakter, men synes at man gør hvad man kan. Resten må være industriens initiativ.

4.1.3 Institut for Datateknik, DTH

Formelle specifikationer

ID har mange projekter om formelle specifikationer. Vi har tidligere nævnt KAITS (side 14) og MetaIV (side 15). Der har her været meget tæt erhvervskontakt og forsøg på at bruge resultaterne i praksis.

VLSI

ID har også projekter indenfor design af VLSI-kredse - igen i tæt kontakt med firmaer.

I forhold til andre danske forskningsinstitutter har nogle af ID's medarbejdere nok den bedste kontakt til apparatindustrien og den bedste forståelse af de tekniske problemer. Alligevel har man svært ved at forstå den totale udviklingsproces og at forskningsresultaterne ikke finder mere accept i industrien.

4.1.4 Stærkstrømsafdelingen, DTH

Dette institut regnes af en eller anden grund ikke for en del af den danske programmelverden, men faktisk er der udviklet et meget bemærkelsesværdigt værktøj her, Beologic, som nu bruges og markedsføres af B&O.

Beologic

Beologic gør det muligt at lave en meget præcis og kompakt beskrivelse af et apparats funktion. Det er muligt at undersøge den logiske konsistens af specifikationen, finde sammenhænge mellem input og output i form af logiske udtryk, etc. Specifikationen kan oversættes til kode der fylder ganske lidt.

Flere andre virksomheder har vist interesse, men vi har ikke mødt andre der bruger det, vistnok fordi det kun kan håndtere relativt få diskrete signaler ind og ud (knapper, mv.), og ikke måledata og datakommunikation.

4.1.5 Institut for Elektroniske Systemer, AUC

Objekt-orienteret design

AUC arbejder med udviklingsmetoder generelt og prøver specielt at bruge objekt-orientering i processen. Det forekommer dog at være på et meget generelt niveau, som apparatindustrien har svært ved at omsætte til deres eget behov.

Udviklingsværktøjer

AUC har bl.a. beskæftiget sig med CASE-værktøjer og prøver fx at bruge Hypertekst til dokumentation af udviklingsprocessen.

Distribuerede systemer

Man prøver at udvikle et generelt koncept for distribution af data og beregninger i et netværk. Et af problemerne er at gøre systemet tilstrækkeligt effektivt til realistisk brug.

Andet

AUC har en betydelig indsats og succes på billedbehandlingsområdet, talegenkendelse, medicinske anvendelser, mobilkommunikation og reguleringsteknik. Det har alt sammen stor industriel betydning, men der er tale om specielle anvendelser der falder uden for denne rapports område.

Generelt gælder at AUC har god lokal erhvervskontakt, men på software-området har man meget lidt kontakt med apparatindustrien.

4.1.6 Handelshøjskolen i København

HHK har bl.a. projekter om metoder og værktøjer til design af brugergrænseflader, undersøgelse af produktudvikling i edb-branchen, og undersøgelser af teknologi-overførsel i industrien generelt.

Man har god kontakt til erhvervslivet generelt, og interesserer sig specielt for de organisatoriske aspekter.

4.1.7 IFAD, Odense

IFAD (Institut for Anvendt Datateknik) spiller flere roller: Formidlere, forskere, og udviklere af apparat-software. På forskningsområdet har man bl.a. lavet en omhyggelig undersøgelse af eksisterende udviklingsmetoder, man udvikler værktøjer (fx baseret på KAITS), og man leder flere ESPRIT-projekter om udviklingsmetoder.

Generelt gælder at IFAD har en usædvanlig god forståelse af apparatindustriens udviklingsproblemer, og man gør en stor indsats for at formidle bedre metoder.

4.2 Holdninger mellem parterne

4.2.1 Forskerens roller og holdninger

- Forskeren skal primært meritere sig ved at producere internationalt anerkendte forskningsresultater. Det betyder at han skal lave noget nyt og publicere det for andre forskere.
- Forskerens faglige miljø består af andre forskere, undertiden suppleret med lidt erhvervskontakt.
- Forskeren kan tillade sig at begrænse problemstillingen meget kraftigt og fordybe sig.
- Forskeren får ikke kredit for at lave noget der kan bruges i praksis, for at formidle til industrien, skrive lærebøger, etc.
- Forskeren har som regel ikke prøvet andet end at være forsker.
- Senior-forskere bruger en meget stor del af deres tid til administration, projektansøgninger, projektledelse, mv.

4.2.2 Forskernes holdning til apparatindustrien

- Nogle forskere erkender blankt at de ikke har kendskab til hvordan udvikling foregår i praksis i erhvervslivet.

- Andre forskere mener at de godt ved hvad der foregår, men reelt bygger deres opfattelse for en stor del på fordomme, og de erkender ikke deres manglende viden. Samtidig har de ikke megen respekt for industriens arbejdsformer og -vilkår. De mener fx at der laves en meget usystematisk udvikling og at produkterne er meget fejlbehæftede.
- Hvis nogen fra industrien henvender sig, er forskeren til gengæld meget positiv og vil gerne hjælpe.

4.2.3 Industriens holdning til forskerne

- Apparatindustrien er generelt ikke interesseret i forskningsresultater før de er tilgængelige i form af et værktøj eller en metode. Den skal tilmed passe godt sammen med det man har allerede.
- Apparatindustrien er generelt positivt indstillet over for forskningen, der betragtes som et uundværligt grundlag for den teknologiske udvikling. Men man tror ikke på at forskerne kan levere noget man kan bruge direkte. Så der er snarere tale om at stille sig til rådighed som prøveklude og med råd og viden.
- Forskerne skal have lov til at forske i fred.
- Man vil gerne kunne trække på forskerne som eksperter.
- Man mener ikke at forskerne ved hvad der foregår "i den virkelige verden".
- Forskerne er meget kloge, men desværre også meget uforståelige.

4.2.4 Formidlernes holdning til forskerne

- Formidlerne synes at forholdet er godt.
- Man holder sig til dem man kender og skaber ikke nye kontakter.
- Man synes at der foregår meget lidt relevant forskning.

4.3 Viden om forskningsprojekter

- Ingen - hverken forskere, formidlere eller folk i industrien - mener at have overblik over dansk forskning på software-området.
- Den viden man har stammer fra fagpressen, kursusmateriale, møder i selskaber og foreninger samt personlige kontakter. De personlige kontakter er meget bestemt af geografi og fælles rødder i en uddannelsesinstitution.
- Der udtales ønsker om at blive informeret, men der er ingen viden om de forskellige databaser der faktisk findes. Fx er der

tilsyneladende ingen i industrien eller hos formidlerne der kender til DANDOK forskningsdatabasen.

- De projekter hvor der virkelig er gjort noget for at forsøge at "sælge" dem (fx KAITS og Mjølner) er forholdsvis kendt - dog mest på niveauet "det navn har jeg hørt".

5. Formidlerne

Indenfor det område vi snakker om har Elektronikcentralen og Teknologisk Institut en væsentlig rolle som formidlere.

SPU, som er en de facto standard på området, er blevet udviklet og udbredt af formidlerne. KAITS er ligeledes blevet produktmodnet og forsøgt udbredt via formidlerne. Udformningen af KAITS til et værktøj (CEDAR) er ligeledes sket ved et samarbejde mellem en formidler (IFAD) og et software firma.

I de senere år er bevillingssystemet ændret så TI og EC nu får mellem 8 og 15% i offentlige tilskud, hvor de før fik 40%. Nogle af pengene fås via deltagelse i projekter, fx ESPRIT. Det betyder at man stort set arbejder på linje med ikke-offentlige formidlere og konsulenter. Det betyder måske også at TI og EC ikke er så hyppige sparringpartnere i forskningsprojekter som tidligere. TI og EC har dog stadig et image som offentlig virksomhed, hvilket betyder at firmaerne henvender sig til dem i forventning om gratis assistance.

Der er ikke løse penge til fx fremstilling af lærebogsmateriale à la SPU, eller modning af forskningsidéer så de kan bruges i industrien.

Begge steder ved man meget om apparatindustrien og dens problemer, da man har mange kontakter såvel via kurser som via forskellige former for konsulentvirksomhed. ERFA-grupperne, som omtales senere, administreres af Datateknisk Forum via EC.

De øvrige formidlere på markedet lægger sig i forskellige nicher, typisk i form af konsulentarbejde, fx firmakurser, bearbejdning af en metode så den passer til firmaet, assistance i projektforløbet ved indførelse af nye metoder. En anden måde at profilere sig på er ved at hyre forskere som gæsteundervisere.

IFAD (Institut for Anvendt Datateknik) i Odense er formidlere, men samtidig arbejder man tæt sammen med en række firmaer, hvor de fleste har geografisk tilknytning til regionen. I nogle tilfælde virker IFAD som en udviklingsafdeling, i andre som konsulent for udviklingsafdelingerne. Sideløbende med disse aktiviteter deltager man i EF-projekter.

Formidlernes viden om dansk forskning ligger på linje med de mere velorienterede firmaer. Det betyder at de har hørt om de væsentligste projekter og kender et par stykker mere detaljeret, men at deres viden er lige så usystematisk indsamlet som firmaernes.

Vi har kunnet konstatere at der er en ret skarp konkurrence mellem formidlerne. Det kan man uddrage af de udtalelser de kommer med om hinanden. De fortrolige metodebøger, som er en udvidelse af SPU, er et andet eksempel. Der er tilsyneladende også flere kurser på markedet end der er kunder til.

En af konsekvenserne af den skarpe konkurrence og den minimale offentlige støtte er, at der ikke er kræfter til at udvikle nye metoder, højst til at tilpasse lidt på de eksisterende.

6. Samarbejde

Samarbejde mellem forskere og industrifolk kan have mange former. Vi gennemgår de vigtigste nedenfor.

6.1 ERFA-grupper

Datateknisk Forum er en forening med et halvt hundrede firmaer som medlemmer. Det har som hovedformål at køre de såkaldte "ERFA-grupper" indenfor forskellige emner.

Der er for tiden ca. 8 grupper. Formen kan være foredrag eller virksomhedsbesøg. Nogle steder er det som om emnerne går i ring efter et par år, hvorefter gruppen har udtjent sit formål. Herefter nedlægges den eller medlemmerne udskiftes. Gruppedeltagerne er ansatte i medlemsfirmaerne. Forskere ses højst som foredragsholdere. Kvaliteten af det arbejde der foregår er meget svingende, afhængig af emnet og af den person der fungerer som leder og indpisker.

Der er udbredt enighed om at idéen med ERFA-grupper er god, men også at resultatet er meget svingende.

Prisen for firma-deltagelse i ERFA-grupper er ca. 9.000 til 20.000 pr. år, afhængig af hvor mange grupper man vil være med i. Det er et ret stort beløb for et forskningsinstitut - i hvert fald er ingen institutter medlem. Undervisningsministeriet har fået et tilbud om et fælles medlemsskab for alle forskningsinstitutter, men det er ikke blevet modtaget og de enkelte forskere kender det næppe.

6.2 Følgegrupper til forskningsprojekter

En følgegruppe er en samarbejdsform hvor repræsentanter fra en række firmaer er medlemmer af en gruppe der løbende bliver holdt underrettet om et konkret forskningsprojekt og ved jævnlige møder kan kommentere projektet. Bl. a. KAITS- og Mjølner-projekterne har haft følgegrupper. ERFA-grupperne har også virket som følgegrupper i nogle tilfælde. Formen betragtes positivt fra industrifolkene, uden at vi har fået det konkretiseret yderligere.

Følgegruppearbejdet er uforpligtende og uden økonomisk støtte. Det er ofte vanskeligt at holde firmaerne engageret i arbejdet, specielt fordi de har andre, mere presserende opgaver.

6.3 Erhvervsforskere

Erhvervsforskere er typisk unge forskere der ansættes i en tidsbegrænset stilling (fx 2 år), hvor de lønnes halvt af et firma og halvt af ATV med henblik på at drive forskning indenfor et område af umiddelbar interesse for firmaet. Forskeren forventes at yde forskning svarende til en normal produktion, og forventes også at være til gavn for firmaet.

Ordningen betragtes positivt af såvel forskere som industri. Forskerne frygter dog firmaer der bare er ude efter billig arbejdskraft og lader forskeren bruge det meste af sin tid på almindelige projekter i firmaet. Erhvervsforskere er en meget anvendt ordning for hardwarefolk, mens vi kun har hørt om ganske få indenfor software, og det har været specielle anvendelser snarere end metoder.

6.4 Rekvireret hjælp

Mange af de forskere vi har snakket med bliver nu og da kontaktet af firmaer der ønsker hjælp indenfor forskerens område. Firmaerne fortæller om det samme, og begge parter giver udtryk for tilfredshed. Det drejer sig både om allerede etablerede kontakter og om nye henvendelser. Som eksempler kan nævnes assistance i forbindelse med nogle reguleringsalgoritmer, og assistance til at vurdere om det design man har lavet er godt nok.

Industrifolkene mente dog, at det var et begrænset antal forskere der kunne levere den slags hjælp.

6.5 Tilbud om hjælp

Der er også eksempler på at forskerne selv tilbyder sig som hjælpere ved at opfordre firmaer til samarbejde. Fx har AUC kontakt til lokale firmaer. Her har man den erfaring at de formelle samarbejdsprogrammer sjældent bliver til noget, mens uformelle henvendelser virker fint.

6.6 Eksamensprojekter

Forskningsinstitutionerne ønsker sig eksamens- og praktikprojekter for at give de studerende realistiske opgaver. Det koster ikke firmaerne løn, men til gengæld kan de ikke forlange noget, og desuden koster det tid fra den person der fungerer som vejleder fra firmaet (ofte en nøgleperson).

Ordningen lader til at være velset, dels som en måde at holde kontakt med en forskningsinstitution, og dels som en god rekrutteringskilde.

6.7 Industriel afprøvning af forskningsresultater

Forskerne kan have brug for industrielle partnere for at kunne afprøve forskningsresultater. På overfladen er der velvilje fra industrien, men når det kommer til stykket bakker man ud: Man har ikke velegnede opgaver lige nu, eller de projekter der ser velegnede ud bliver udsat eller omdefineret. Det er på dette punkt forskerne klager over industriens manglende lyst til at eksperimentere.

I ESPRIT-projekter indgår der ofte såvel universiteter som industrivirksomheder, men her får firmaerne betalt (noget af) deres tid. Vi hørte alligevel om et større firma der ikke ønskede at deltage i den slags arbejde, fordi man ikke ønskede at kaste sig ud i en aktivitet der alligevel ville blive prioriteret væk næste gang man fik travlt.

6.8 Personoverførsel mellem forskning og industri

Selvom det hører til sjældenhederne, er der enkelte erhvervsfolk der bliver ansat som forskere. Normalt bliver de så hængende dér, men på et institut havde man også et vellykket eksempel på en "gammel" kandidat der blev ansat på instituttet i en periode og derpå vendte tilbage til sit firma igen.

Det modsatte sker også, men som regel kun i form af yngre forskere der bliver ansat i industrien. Vi har ikke hørt om etablerede forskere indenfor software, der skiftede til erhvervslivet.

Forskerne kan forestille sig ordninger hvor erhvervsfolk har sabbatordninger, så de kan komme "tilbage" og blive klogere.

Erhvervsfolkene forestiller sig, at forskerne kunne have glæde af at arbejde med på et "rigtigt" projekt indenfor det område de beskæftiger sig med.

Ingen af parterne er særlig interesseret i de andres forslag. En mulig forklaring er, at man efterhånden opbygger en identitet og trives med det man er god til. Hvis man flytter, kan man forvente først at skulle gennem en længere periode hvor man bliver "kigget an" og skal vise om man har noget at byde på. Det vil i sig selv være ubehageligt, men dertil kommer en forestilling om at det ikke er særlig interessante opgaver der tilbydes det nye sted.

6.9 Uformelt samarbejde

Det samarbejde der lader til at fungere bedst er det uformelle. Det er meget personbundet - som oftest mellem gamle studiekammerater eller forhenværende kolleger.

Der er intet i vejen med det uformelle samarbejde, udover at der ikke

er så meget af det, og at det måske har en tendens til at holde viden og information inde i nogle kredse der kender hinanden i forvejen.

7. Uddannelse

Uddannelsen indenfor software har ikke været emnet for denne rapport, men vi har da gjort nogle iagttagelser.

Uddannelserne forsøger at give grundlæggende kunnen og en teoretisk overbygning. Der er ingen af de akademiske uddannelser der forsøger at sikre en fast basisviden ud over et absolut minimum. Konkrete teknikker er ikke noget forskerne mener hører hjemme på de højere uddannelser.

SPU er et godt eksempel. Denne metode - der har vist sig at være en de facto standard i apparatindustrien - undervises der stort set ikke i på de datalogisk prægede uddannelser (Handelshøjskolens datalogi-økonomiuddannelse er fx en undtagelse), men nok på nogle teknika og lignende. Det betyder at industrien så selv må give en efteruddannelse. Tilsvarende gælder for mange andre nyttige metoder og teknikker.

7.1 Typer af uddannelser

Vi har været overraskede over hvor lidt vi selv vidste om hvilke typer af uddannelser der findes for dataloger og ingeniører. Fx er der dataloger med kredsløbsteknik (Odense Universitet), dataloger med informationspsykologi (Københavns Universitet) og dataloger med økonomi (Handelshøjskolen). Det lader til at både forskere og industrifolk er lige så uvidende om uddannelserne.

7.2 Indhold af uddannelserne

Det viser sig, at de kandidater der kommer fra forskellige steder er meget forskellige både i deres obligatoriske og deres tilvalgte uddannelse. De enkelte uddannelsesinstitutioner ved sjældent hvor kandidaterne bliver af og hvad de savner af viden. Nogle steder prøver man dog at sende spørgeskemaer ud til de færdige kandidater, men det er uklart hvad man bruger svarene til.

Målsætningen for uddannelse af fx dataloger går meget lidt på konkrete teknikker, men mere på holdninger, kritisk sans og fornemmelse for kvalitet - og så mere generelle teknikker. Specielle forskningsområder på institutionen vil typisk præge uddannelsen i en grad der ikke står mål med behovet for viden på dette område.

Meget af undervisningen betragtes som et tilbud hvor det er op til den enkelte at vælge.

Der anvendes projektarbejde i større eller mindre grad i flere uddannelser, mens fx teknikum-uddannelserne er mere skole- og teknikprægede.

Hvis man ser på industriens behov, er det oplagt at en lang række emner mangler i uddannelserne. Fx undervises der ikke i følgende emner på de højere uddannelser (igen med enkelte undtagelser):

- Hvad er en kravspecifikation? Hvad skal der stå i en kontrakt?
- Store softwaresystemer: Hvad er det der går galt?
- Kvalitetssikring.
- Performance-beregninger.
- Brugergrænseflader.
- Konkrete metoder (fx SPU) og state-of-the art.
- Samarbejde i teori og praksis.

7.3 Aftagernes indflydelse på uddannelserne

På edb-skolerne har aftagerne (erhvervslivet) stor indflydelse på undervisningens indhold. For de højere uddannelser har vi ikke set eksempler på at aftagerne har nogen indflydelse på hvad der undervises i.

Aftagerne tror at kandidaterne kender de nyeste teknikker, men det er der ikke nogen som helst garanti for.

Hvis man ser på en mere etableret uddannelse, som fx lægeuddannelsen, har man en fælles teoretisk universitetsuddannelse efterfulgt af flere års "mesterlære", hvor kandidaten skal arbejde på bestemte typer hospitalsafdelinger og som føl hos en praktiserende læge før uddannelsen betragtes som færdig.

I sammenligning med dette er uddannelsen til "softwareudvikler" det rene anarki:

- Der er mindst 10 forskellige uddannelser, og de fleste af dem er så tilvalgsorienterede, at det kun er kandidaten selv der ved hvad det er for en slags uddannelse han har. Der er dog en vis garanti for uddannelsens bredde og lødighed.
- Når uddannelsesforløbet er slut, er der ingen koordinering eller mulighed for at sikre sig en fortsat vedligeholdt basisviden. Formidlerne og fagforeningerne tilbyder en række kurser og foredrag, men det er ofte mere modeprægede emner.
- Mesterlæreprincippet kan fungere nogenlunde i firmaer hvor en projektgruppe med erfarne folk kan oplære en nyuddannet. Men der er ingen garanti for at den man ansætter kan tilføre noget nyt. Tværtimod er der en tendens til at ansætte folk der ligner dem man har: samme uddannelse og faglige fokus.

- Der er ingen fælles identitet hos softwareudviklere. Der er ingeniører, dataloger, folk fra andre uddannelser, autodidakter, osv. Hvis de er fagligt organiseret, kan de tilhøre flere forskellige fagforeninger (Ingeniørforeningen, Magisterforeningen, PROSA m.fl.), som hver for sig ikke betragter softwareudviklere i apparatindustrien som deres kernemedlemmer. Det ses også af at der ikke findes et fælles fagtidsskrift som "Tidsskrift for Softwareudviklere". Nogle læser "Ingeniøren" og nogle læser "Computerworld", men tilsyneladende ret tilfældigt.

7.4 Videreuddannelse

Videreuddannelse - i betydningen vedligeholdelse af den uddannelse man har - er et problem som forekommer meget væsentligt, men der er ingen bud på hvordan man løser det.

Videreuddannelse er noget der overlades til den enkeltes initiativ i ærgrelse over at "sidde og blive dummere og dummere", som vi hørte det udtrykt. Ledelsen er ofte passiv omend positiv indstillet, hvilket giver sig udtryk i at man budgetterer med uddannelsesomkostninger. Der findes dog også steder hvor firmaet ikke støtter videreuddannelse.

Leverandørkurser og kurser i konkrete teknikker, som man lige står over for at skulle bruge, kan komme på tale, men vedligeholdelsen af teoretisk viden om emner der ligger uden for det konkrete arbejde må ske gennem bøger og tidsskriftsartikler, der er ret svært tilgængelige for de fleste.

Der er ingen tilbud om videreuddannelse fra uddannelsessystemet. Der er fx ingen aftenkursus i "Datalogi 1 - revisited". Datalogistudierne har i praksis den efterlyste funktion for de der tager et andendelskursus i ny og næ, men der er mangel på pladser, og der er i princippet kun adgang for studerende. Færdige kandidater har ikke adgang.

Kollokvier eller tilsvarende korte foredrag hvor gæster, lærere og studerende præsenterer deres aktuelle arbejde er en glimrende institution man har flere steder, men den sigter mere snævert på lokale specialer end på en bredere orientering om mere veletablerede teknikker.

8. Forslag

I dette kapitel opstiller vi en række forslag vi har mødt eller selv er kommet på under arbejdet. Vi har kun vage forestillinger om hvorvidt de er realistiske, og hvordan de evt. kunne gennemføres. Nogle af dem er måske allerede prøvet uden at vi ved det. Forslagene må altså betragtes som idéer til videre behandling.

8.1 Prøvede forslag

En del af de forslag vi har hørt om er faktisk allerede prøvet eller udført. Det drejer sig fx om følgende:

(A1) En database over forskningsprojekter. Den findes allerede i form af DANDOK-databasen, men ingen vi har mødt har været klar over dens eksistens eller hvordan man får adgang til den. En bedre formidling af den og andre tilsvarende baser kunne ønskes.

(A3) En database over hvilke forskere der laver hvad og har hvilke interesser. Også disse oplysninger er vist tilgængelige via DANDOK.

(A2) Bedre orientering fra forskerne om deres projekter. Det findes allerede i flere former. Mange forskningsinstitutter holder eller har holdt årlige "virksomhedsdage" hvor man giver den slags informationer. Større forskningsprojekter har ofte en følgegruppe med repræsentanter fra virksomheder. Ingen af delene giver noget væsentligt udbytte.

(A3) Teknologi-overførselsprojekter. Det har bl.a. været prøvet for metoder som KAITS, Mjølner og MetaIV, men uden de store resultater. Idéen skulle nok prøves igen, men for andre, mere jordnære teknikker.

(A4) Teknologi-overførsel ved on-the-job træning. Det bruges faktisk af flere konsulenter, men mest i forbindelse med metoder uden det store forskningsindhold. SPU er typisk sat i værk på denne måde, og det virker fint, men er ret dyrt for virksomheden. Små virksomheder har derfor svært ved at bruge det. EF's nye ESSI program giver direkte støtte til den slags, men vil mindre virksomheder opdage og udnytte det?

(A5) Støtte til forskere der vil udvikle værktøjsprodukter. Det findes allerede i flere former, men det er svært at få støtte og produkterne slår i øvrigt sjældent an.

8.2 Forslag til forskerne

De følgende forslag kræver et tæt samarbejde mellem forskere og en eller flere virksomheder, hvis der skal komme noget erhvervsrelevant ud af det.

(B1) Forskerfølgegrupper i udviklingsprojekter. Lad forskere kigge kritisk på nogle eksisterende udviklingsprojekter - i detaljer. Dels kan forskerne lære noget om de faktiske udviklingsvilkår, dels kan de få inspiration til bedre teknikker og metoder, og endelig kan de måske levere idéer til virksomhederne.

(B2) Forskning i produktafgrænsning, kravspecifikation og design.

(B3) Udbygning af SPU-metoden med flere detaljer og innovative teknikker, fx med mere realistiske eksempler på kravspecifikation, designspecifikation efter forskellige teknikker (herunder de matematiske), risikostyring (spiralmodel), etc. Resultatet kunne blive en metode-håndbog, der bl.a. kunne anvendes af mange virksomheder som led i ISO 9000 certificering.

(B4) CASE-værktøjer til apparatindustrien, bl.a. med mulighed for sporing fra krav til design, program, og test.

(B5) Udbygning af objektorienterede metoder så der angives konkrete løsninger på problemer der optræder i apparatindustrien, fx protokolspecifikation via objekter, konfigurering, multi-tasking, mikroprocessorer.

(B6) Tilskyndelse til orlov der tilbringes i virksomheder.

8.3 Forslag til formidlerne

(C1) Få forskerne inddraget i ERFA-grupper. Søg evt. endnu engang om bevillinger til fælles medlemskab for forskere.

(C2) Samarbejd mere systematisk med forskere for at udbygge SPU-metoden.

(C3) Udarbejd en bibliografi over eksisterende teknikker og lav realistiske eksempler på teknikkens anvendelse. (IFAD har et tilløb til det).

(C4) Etabler et fælles fagblad for apparatsoftware - i stedet for de mere reklameorienterede blade som hver formidler i øjeblikket laver.

(C5) Etabler brugergrupper om afprøvning og brug af værktøjer.

(C6) Etabler en samling detaljerede arkitekturmodeller for apparater.

8.4 Forslag til industrien

(D1) Etabler samarbejde om en fælles apparat-platform med tilhørende værktøjer.

(D2) Etabler teknologi-overførselsprojekter på virksomhedens præmisser, men start med en omhyggelig analyse af de problemer man faktisk har og forventer i de kommende projekter.

(D3) Lad være med at tro på at løsningen bare er det rigtige værktøj. Alle formidlerne er enige om at værktøjer alene ikke løser nogle problemer. Man må selv systematisk arbejde på at forbedre sine metoder.

8.5 Forslag til uddannelsessystemet

(E1) De datalogi-orienterede uddannelser bør langt bedre dække state-of-the-art, fx kravspecifikationer, kvalitetssikring, performanceberegning.

(E2) Der er behov for uddannelser der kombinerer datalogi og kredsløbsteknik (i Odense er der en).

(E3) Giv de færdiguddannede mulighed for at følge dele af uddannelserne.

(E4) Der er behov for egentlig efteruddannelse der strækker sig over en længere periode. HD uddannelsen har på økonomiområdet dækket dette behov i mange år, men der er et tilsvarende behov indenfor datalogi-området.

8.6 Offentlig forsknings- og uddannelsesstyring

(F1) Læg vægt på at meritering også kan ske via erhvervserfaring og formidling. De nuværende regler for besættelse af forskningsstillinger lægger ensidig vægt på et forskningsniveau svarende til licentiatgraden (Ph.D. niveau). Det gør at vigtige praktiske erfaringer ikke kan blive videreforarbejdet på forskningsniveau. Desuden låser det de faste lærere i et rent forskningsmiljø.

(F2) Der mangler offentlig støtte på områder der slår bro mellem forskning og udvikling. Typisk vil Industriministeriet synes at et sådant projekt er for forskningspræget, mens Undervisnings-/Forskningsministeriet vil synes at det er et udviklingsprojekt. Et symptom på dette er formidlernes manglende mulighed for at lave nyudvikling af metoder.

(F3) Flere af de nye EF-programmer er meget relevante for apparatsoftware, men det er svært for mindre forskningsgrupper at komme med. Dels er det svært at finde partnere, dels er rejsebevil-

linger og andre opstartmidler ikke tilgængelige til sådanne formål. Støtte på dette område ville være en stor hjælp.

(F4) Forskningsrådene har svært ved at finde plads til den "blødere" forskning, der ligger mellem de matematiske/tekniske aspekter og de samfundsmæssige eller humanistiske. Emner som kravspecifikation, brugerdialog, og kvalitetssikring falder fx mellem to eller flere forskningsråd, der henviser til hinanden. En bevillingspulje til sådanne, mindre projekter ville hjælpe, men den skal startes langsomt op for at undgå "hovsa-projekter". Området er fx endnu ikke modent til at slå større rammeprogrammer eller centre op.

Noter

[Note 1]

Ifølge Christensen (1992) opdeles teknologisk innovation normalt i kategorierne produktinnovation og procesinnovation.

Christensen skriver (1992, side 11): "Teknologisk betingede produktivitetsstigninger kan dels tilskrives procesinnovationer, dvs. forbedrede eller nye produktionsprocesser, der reducerer de reelle enhedsomkostninger ved at producere eksisterende produkter, dels produktinnovationer, der tenderer mod at skabe større ydeevne, bedre funktionsvaretagelse eller behovstilfredsstillelse."

Set i lyset af dette, betyder vores koncentration om udviklingsprocessen, at vi fokuserer på procesinnovation og kun sekundært inddrager produktinnovation i vores undersøgelse.

[Note 2]

At de fundne problemer med teknologi-overførsel ikke er specielt danske fremgår f.eks. af Nilakanta & Scamell (1990) der i et studie af overførslen af forskningsresultater mht. database-udvikling bl.a. konkluderer: "At the individual level we see that even the more familiar and less technically sophisticated innovations are not readily influenced by communication of innovation."

Baron (1990) rapporterer at teknologi-overførslen i USA er meget sporadisk - ligesom i Danmark. Han hævder at den væsentligste hindring for overførsel af teknologi er kulturforskelle, og efter en gennemgang af en række amerikanske tiltag konkluderer han (1990, side 43): "There is really no easy solution or simple answer to overcoming the cultural barriers between Government and industry."

Vedr. de problemer vi har fundet i udviklingsprocessen, så er der også her lignende udenlandske undersøgelser, f.eks. Curtis et al. (1988), der på basis af en interviewundersøgelse i større amerikanske virksomheder identificerede tre fremtrædende problemområder ved udvikling af softwareprodukter, herunder apparater: (1) Sparsom udbredelse af domæneviden. (2) Ustabile og modstridende krav. (3) Kommunikations- og koordinations-sammenbrud.

Relevant er også en række artikler af Jonathan Grudin (1991a, 1991b og 1991c) omhandlende ti hindringer for brugerinddragelse i organisationer der udvikler softwareprodukter, og 19 målsætninger for udviklingsorganisationer der systematisk skulle ødelægge softwareprodukters brugervenlighed.

Endelig hævder Quintas (1991), på basis af erfaringer fra det engelske Alvey-program, at udbredelsen af Software Engineering hæmmes af sociale, organisatoriske, kulturelle og institutionelle faktorer. F.eks.

konkluderer Quintas om årsagen til den ringe udbredelse af formelle metoder (1991, side 373): " Software Engineering approaches such as formal methods ... challenge current development practice and the existing human resource base ... but also the degree of control and autonomy exercised by software personnel."

[Note 3]

SPU-modellen (Struktureret Program Udvikling) er beskrevet i Biering-Sørensen et al. (1988).

[Note 4]

Denne definition af CASE-værktøjer er så bred at næsten alle typer værktøjer til udvikling af edb-systemer som støtter mere end to teknikker bliver et CASE-værktøj. En anden langt mere afgrænsende definition gives af Chris Gane (1990: xii): "... the distinguishing characteristic of a CASE product is that it builds within itself a *design database*, at a higher level than code statements or physical data element definitions." Denne definition svarer også til det der kommercielt sælges som CASE-værktøjer. Her kaldes den centrale database ofte for et "repository".

[Note 5]

Struktureret analyse og design af "real-time systems" er fx beskrevet i Ward & Mellor (1986) og i Hatley & Pirbhai (1987). Mere generelle anvendelser er beskrevet i Yourdon (1989) om struktureret analyse, eller i Page-Jones (1988) om struktureret design.

[Note 6]

KAITS - senere kaldet CEDER eller CEDAR - er beskrevet i Rischel, Mortensen & Ravn (1987). En kort, god illustration af metoden findes i Elmstrøm (1989).

[Note 7]

Vienna Development Method (VDM) blev oprindeligt udviklet på IBM's laboratorium i Wien (Bekic et al. 1974), og blev videreudviklet op igennem 80'erne (Jones 1980 & 1986). VDM anvender et metasprog kaldet Meta-IV (Bjørner & Oest, 1980), baseret på domæneteori (Stoy, 1977) og lambda-kalkuler (Church, 1941).

En pædagogisk gennemgang på dansk af MetaIV kan f.eks. findes i en lærebog/undervisningsnote af Hans Rischel (1989).

Den nævnte udvikling af en ADA-oversætter er bl.a. beskrevet af Clemmensen & Oest (1984).

[Note 8]

I projektet "Dynamisk Specialisering" (Christensen et al. 1990, Valentin 1990, Christensen 1990, Lotz 1990, Karnøe 1990) undersøgte man hvilke anledninger der satte produktudvikling igang i avancerede mindre og mellemstore danske virksomheder.

Centrale konklusioner var: (1) Nye markedsbehov havde væsentlig større betydning end nye tekniske muligheder. (2) Den centrale kilde til innovation var søgeadfærd ud fra eksisterende viden, typisk initieret af nye behov hos nuværende kunder. (3) Kun sjældent kunne starten af en produktudviklingsproces knyttes til én markant begivenhed. Som regel var der tale om mange anledninger, som i fællig samlede interessen om en bestemt produktidé.

[Note 9]

Kravspecifikationer er et af de problemer som Curtis et al. (1988) også identificerede som centralt (se evt. note 2).

For ganske nylig (november 1992) har Elektronikcentralen offentliggjort en rapport hvor 11 af 13 firmaer, der hver blev bedt om at udpege fem kilder til problemer, pegede på Software kravspecifikationen som en af de hyppigste kilder til problemer (Thaysen 1992, side 23).

[Note 10]

Prototyping er en teknik til kravspecifikation der er karakteriseret ved en høj grad af iteration, en høj grad af brugerinddragelse i udviklingsprocessen, samt ved udstrakt brug af prototyper (Vonk 1990, side 22).

Denne definition af prototyping kaldes af Christine Floyd (1984) for *exploratory prototyping*. En anden type er *experimental prototyping* som finder sted i den tekniske designfase, og hvor prototypen bruges til at afgøre anvendeligheden af en foreslået løsning før implementering i det "rigtige" edb-system. Endelig taler Christine Floyd (1984) om *evolutionary prototyping* hvor idéen er, at edb-systemet udvikles i mange versioner; hver gang man er færdig med en version, så går man igang med den næste. Så her er prototypen det færdige system.

Barry Boehm foreslog i 1986 et skift af perspektiv fra det han kaldte dokumentdrevne modeller, hvor en aktivitet (fase) afsluttes med et dokument, til en risikodreven model.

Den risikodrevne udviklingsmodel som Barry Boehm foreslog illustreres med en spiral, hvorfor den ofte kaldes en *spiralmodel* for softwareudvikling.

Udviklingen tager sit udgangspunkt i centrum af spiralen. Afstanden fra centrum er proportional med ens foreløbige investering. For hver tur rundt i spiralen laver man en tilbundsgående vurdering af risikoen ved at fortsætte.

Efter den første risikovurdering vælger man udviklingsmodel, f.eks. kan man vælge at lave en prototype af brugergrænsefladen såfremt der er knyttet væsentlig risiko til den. Boehm & Papaccio (1988) skriver: "Once the risks are evaluated, the next step is determined by the relative risks remaining. If performance or user-interface risks strongly dominate program development or internal interface-control risks, the next step may be an evolutionary development step: a minimal effort to specify the overall nature of the product, a plan for the next level of prototyping, and the development of a more detailed prototype to continue to resolve the major risk issues.

Efter næste tur i spiralen vurderer man igen risikoen og vælger en passende model til den videre fremfærd. Alt i alt er idéen i spiralmodellen, at man på et hvilket som helst tidspunkt kan vælge nøjagtig den fremgangsmåde der bedst minimerer risikoen ved at fortsætte.

Det springende punkt i den risikodrevne spiralmodel er, at det er svært at få konkrete anvisninger på hvordan risikoen skal vurderes i forskellige situationer, samt på hvordan vidt forskellige typer risici bringes på sammenlignelig form med henblik på at fastslå hvad der er den største risiko.

[Note 11]

Kravspecifikationer behandles f.eks. på den årlige "International Conference on Software Engineering" (ICSE), næste gang i Baltimore, maj 1993. Mere specifikt kan nævnes en konference som ICRE'94 der afvikles parallelt i Colorado Springs og på Taiwan, hvor emnet er "Requirements Engineering".

[Note 12]

Det er vores opfattelse at en stor del af forskellene i reaktioner hos forskellige af de virksomheder vi var ude i kan henføres til forskelle i modenhed.

Software Engineering Institute (SEI) i Pittsburgh udviklede i slutningen af 80'erne en modenhedsmodel, med fem niveauer (Humphrey 1988, Humphrey et al. 1991), hvor den bagved liggende filosofi er, at man kun har styr på en ting når man kan måle på den og sætte tal på, altså en meget fabriks- og automatiseringspræget tankegang, der ikke desto mindre har vundet stor udbredelse, ikke mindst i USA. De fem modenhedsniveauer kan kort beskrives på følgende måde:

(1) "Initial", hvor udviklingsprocessen er ad-hoc eller måske ligefrem kaotisk. Andre kendetegn: Ingen styring der sikrer ensartet udvikling fra gang til gang. Meget svingende kvalitet medfører ofte store installations- og vedligeholdelses-problemer. Mangler ofte overblik pga. ringe erfaring med udvikling.

(2) "Repeatable" hvor den grundlæggende projektstyring er på plads så man kan gentage en type aktivitet man tidligere har udført. Kendetegn: Projektstyring indført således at der findes planer for tids- og resurseforbrug, som også kontrolleres. Mindst én leder inddrages / orienteres løbende under projektet. Der er indført kvalitetssikring og ændringskontrol.

(3) "Defined" hvor udviklingsprocessen er stabil men ikke målbar. En særlig gruppe/afdeling er etableret "to focus and lead the process improvement efforts, and to facilitate the introduction of a family of software engineering methods and technologies." (Humphrey et al. 1991, s. 41).

(4) "Managed" niveauet indikerer at man har statistisk kontrol over udviklingsprocessen. Der er opstillet målekriterier for kvalitet og omkostninger i alle faser, og målingerne bliver altid udført. Man har en procesdatabase til registrering af målingerne, som også anvendes til at overvåge kvaliteten af de udviklede produkter/systemer, og holdt op mod den målsætning om kvalitet som man måtte have opstillet.

(5) "Optimizing" hvor data/målinger automatisk indsamles, så man kan justere og optimere sin udviklingsproces ud fra det omfattende statistiske materiale. F.eks. vil man kunne konstatere om man er bedre til at finde fejl med én review-teknik frem for en anden review-teknik.

Det gode ved modenheds-modellen er, at man som virksomhed kan få vurderet sin egen situation og få nogen konstruktive forslag til at komme videre til næste niveau. Vi vil dog betvivle at det overhovedet er muligt at nå niveau 4 eller 5, populært kaldet software-fabrikken, fordi metoder, værktøjer, og teknologi hele tiden ændres så erfaringer fra et tidligere projekt ikke uden videre kan bruges i nye projekter.

I en sammenligning af japansk og amerikansk softwareudvikling (Humphrey 1991) konstateredes det, at langt over halvdelen af alle virksomheder stadig var på niveau 1. Der var ingen, hverken fra USA eller Japan på niveau 4, men alligevel en enkelt udviklingsgruppe (fra Japan) på niveau 5.

Det sted hvor vi mest udtalt mødte kritikken om at man genererede for meget papir var en virksomhed der iøvrigt havde udmærket styr på deres udviklingsproces, modenhedsniveau "Defined". Så lidt populært kan man sige at det var et "luksusproblem" i forhold til de mange virksomheder på lavere modenhedsniveauer som vi mødte.

[Note 13]

Troen på værktøjer er ikke ny, men kan bl.a. genfindes i et interview med Les Belady (Myers 1985) fra det amerikanske forskningsinitiativ MCC: "... even if some new method has been proven to be more useful than another method, it has to be implemented in some kind of

tool and environment where the new method is enforced automatically by the machine."

I forbindelse med værktøjer er der dog flere der har konstateret at værktøjet ikke er nok i sig selv. Man er nødt til samtidig at gøre et bevidst forsøg på at ændre rutiner og fremgangsmåder så det passer med værktøjet. (Aaen & Sørensen 1991)

[Note 14]

Det behovshieraki vi taler om her er ikke Maslows, kendt fra psykologien, men en software-version af Alan M. Davis (1992): "The most basic need is a task - we can't do much unless we have something to do. Once we have a task to perform, we next need resources ... Once resources are in place we need principles. It is only with a thorough understanding of underlying rules or oracles that we can possibly create process ... Only after we have all four needs (tasks, resources, principles, process) met, can we hope to use tools effectively."

Litteratur

Aaen, I. & C. Sørensen (1991). A Case of great expectations. Scandinavian Journal of Information Systems, Vol. 3, side 3-23.

Baron, Seymour (1990): Overcoming barriers to technology transfer. Research & Technology Management, January-February 1990, side 38-43.

Bekic, H., D. Bjørner, W. Henphal, C.B. Jones & P. Lucas (1974). A Formal definition of a PL/I Subset. IBM Vienna, TR25.139.

Bjørner, Dines & Ole N. Oest (eds.) (1980). Towards a Formal description of Ada. Lecture Notes in Computer Science, Vol. 98, Springer Verlag.

Biering-Sørensen, S., F.O. Hansen, S. Klim & P.T. Madsen (1988). Håndbog i struktureret programudvikling (SPU). Teknisk Forlag, København.

Boehm, B.W. (1986). A spiral model of software development and enhancement. ACM sigsoft, software engineering notes, 11(4), side 14-23.

Boehm, B.W. & P.N. Papaccio (1988). Understanding and controlling software costs. IEEE transactions on software engineering, 4(10), side 1462-1477.

Christensen, J.F. (1992). Produktinnovation - proces og strategi. Handelshøjskolens Forlag.

Clemmensen, Geert B. & Ole N. Oest (1984). Formal Specification and Development of an ADA Compiler - A VDM case study. Dansk Datamatik Center, Lyngby.

Church, A. (1941). The Calculi of Lambda-Conversion. Annals of Mathematical Studies, Vol. 6, Princeton University Press, New Jersey.

Curtis, B., H. Krasner & N. Iscoe (1988). A field study of the software design process for large systems. Communications of the ACM, 31(11), side 1268-1287.

Davis, Alan M. (1992). Why industry often says 'No Thanks' to research. IEEE Software, November 1992, side 97-99.

Elmstrøm, René (1989): A taximeter design project using CEDER. IFAD, Odense, november 1989.

Floyd, C. (1984). A systematic look at prototyping. In Budde, Kuhlenkamp, Mathiassen & Züllighoven (eds.). Approaches to prototyping. Springer Verlag, Berlin, side 1-19.

Gane, C. (1990). Computer-aided software engineering: The methodologies, the products, and the future. Prentice-Hall, Englewood Cliffs, New Jersey.

Grudin, J. (1991a). Interactive systems: Bridging the gaps between developers and users. IEEE Computer, April 1991, side 59-69.

Grudin, J. (1991b). Systematic sources of suboptimal interface design in large product development organizations. Human-computer interaction, 6(2), side 147-196.

Grudin, J. (1991c). Obstacles to user involvement in software product development, with implications for CSCW. International journal on man-machine studies, No. 34, side 435-452.

Hatley, D.J. & Pirbhai, I.A. (1987): Strategies for real-time system specification. Dorset House Publishing, New York, 1987.

Humphrey, Watts S. (1988). Characterizing the Software Process: A Maturity Framework. IEEE Software, Vol. 5, No. 2, side 73-79.

Jones, C.B. (1980). Software development. A rigorous approach. Prentice-Hall, London.

Jones, C.B. (1986). Systematic software development using VDM. Prentice-Hall, London.

Nilakanta, Sree and Richard W. Scamell (1990): The Effect of Information Sources and Communication Channels on the Diffusion of Innovation in a Data Base Development Environment. Management Science, Vol. 36, No. 1, side 24-40.

Page-Jones, Meilir (1988). Practical guide to Structured Systems Design, 2nd edition. Yourdon Press, Englewood Cliffs, New Jersey.

Quintas, Paul (1991): Engineering Solutions to Software Problems: Some Institutional and Social Factors Shaping Change. Technology Analysis & Strategic Management, Vol. 3, No. 4, side 359-376.

Rischel, Hans & Mortensen, Benny Graff & Ravn, A.P.: Konstruktion af formålsbundne systemer. Teknisk Forlag, 1987.

Rischel, Hans (1989) Programdesign sat på formler. Institut for Datateknik, Danmarks Tekniske Højskole, Lyngby.

Stoy, J.E. (1977). Denotional Semantics: The Scott-Strachey Approach to Programming Language Theory. MIT Press, Boston, Massachusetts.

Thaysen, Berit (1992). Rapport over Datateknisk Forums teknologi- og aktivitetsundersøgelse. DF-07. Elektronikcentralen, Hørsholm.

Vonk, R. (1990). Prototyping: The effective use of CASE technology. Prentice-Hall, London, England.

Ward, Paul & Steve Mellor (1986). Structured Development for Real-Time Systems, Volumes 1-3. Yourdon press, New York.

Yourdon, E. (1989). Modern Structured Analysis. Prentice-Hall, Englewood Cliffs, New Jersey.