senter for teknologi og samfunn

STS

Morten Hatling

MODELLAR OG METODE I SYSTEMUTVIKLINGSLITTERATUREN

STS-arbeidsnotat 20/92

ISSN 0802-3573-64

arbeidsnotat
working paper

senter for teknologi og samfunn
universitetet i trondheim
7055 dragvoll

tlf: +47 73 59 17 88
fax: +47 73 59 13 27

centre for technology and society
university of trondheim
n-7055 dragvoll, norway

1. Innleiing

Programvare er kjernen i IT-system, den viktigaste faktoren i systemkostnader og den fremste kjelda til at IT-systema ikkje verkar som dei skal. Utvikling av programvare/datasystem, *systemutvikling*, er derfor ein svært sentral del av datoriseringa av arbeidslivet. Utnytting av IT-system er assosiert med radikal transformasjon av arbeidsinnhald, arbeidsmarknad og kompetansekrav. Samtidig viser det seg at utviklinga av informasjonssystema er svært arbeidsintensivt og lite automatisert. Tradisjonelt har systemutviklingsprosessen blitt framstilt som relativt uproblematisk, virkeligheita har vist seg å vere annleis. Dei mange, kostbare og katastrofale feila ved IT-system kan oftast sporast tilbake til den programvaren som styrer dei. Ei britisk undersøking estimerte at dårlig programvare kosta landet meir enn 500 mill. pund i året (Price Waterhouse 1988). Programvareutvikling blir derfor av mange trekt fram som den viktigaste flaskehalsen for betre og meir effektiv utnytting av datateknologien. Det er også den største kostnaden, f. eks. estimerer Sommerville (1982) at det utgjør meir enn 85% av dei totale utgiftene til IT i 1982. Eg skal i dette notatet diskutere ein del av programvareutvikling, nemlig systemutvikling. Det er eit fagfelt som omfattar teknologiutvikling, (dvs. programvareutvikling, programvaretilpasning og innkjøp/tilpassing av datamaskinar og anna nødvendig utstyr som ein skal køyre desse programma på), informasjonsinnhenting, brukarkommunikasjon, prosjektorganisering, budsjettering og organisasjonsutvikling. Det er ulike forståingar av korleis ein skal vektlegge desse ulike aktivitetane og kor langt ein kan gå i den eine eller andre retninga og samtidig vere innafor ramma av systemutvikling. Det gjeld både i forhold til kven som utfører desse oppgåvene og kva aktivitetar som faktisk skal inngå som ein del av faget. Ein kan beskrive systemutvikling som ein formidlingsprosess mellom maskin og brukarbehov (Friedman 1989). Fokus innan systemutvikling har, særlig i dei siste åra, bevega seg frå maskinkjerna (dvs. maskinvare og operativsystem) og over mot brukarane og deira behov. Denne tendensen er ikkje svært gammal og ein ser derfor at den tekniske sida av systemutvikling er den som er mest utvikla, og der det har nedfelt seg størst grad av konsensus om kva systemutvikling faktisk skal dreie seg om. Det ein kan kalle dei ikkje tekniske sidene ved systemutvikling, har ikkje nådd same modningsgrad og er på mange måtar enno svært uutvikla og "kaotisk", både metodisk, teoretisk og konseptuelt (Friedman 1989, Quintas 1991).

I kjølvatnet av dei mange historiene om feilslåtte og svært kostbare

systemutviklingsprosjekta blir det i aukande grad fokusert på det metodiske og teoretiske grunnlaget som systemutviklarar stør seg på i utviklingsprosessen. Systemutviklingsprosessen er svært arbeidsintensiv og lite automatisert, med stor grad av autonomi for utviklarane. Det har medverka til at programvareutvikling har hatt mye lågare produktivitetsutvikling enn utvikling og produksjon av maskinvare. Eit hovudproblem er forholdet mellom dei ikkje-tekniske aspekta ved systemutvikling, dvs. korleis ein i utviklingsprosessen får tak i, beskriv og modellerer den kunnskapen om arbeidsoppgåver, kommunikasjonsstruktur og organisasjon som skal ligge til grunn for systemet, og den tekniske utforminga av systemet. Det viser seg ofte at edb-systemet ikkje matchar dei forretningsstrategiar det skal tene og derfor ikkje er i stand til å nå dei oppsette mål. Problemet ligg ofte i for dårlig analysearbeid, som skuldast både det teoretiske og metodiske grunnlaget systemutvikling bygger på og det faktum at leiingas krav til kostnadsstyring og hurtig avkastning på it-investeringar ofte set klare grenser for kor store ressursar utviklarane kan bruke på dei tidlige fasane i utviklingsprosessen. Ein ser derfor både ein diskusjon om kva ein fornuftig it-strategi er, samtidig som stadig fleire peiker på behovet for ein radikal omlegging og forbetring av systemutviklingsmetodar og -teoriar.

Dersom ein vender seg til lærebøker i systemutvikling blir ein slått av kor stor vekt forfattarane legg på at dette er eit felt som er svært vanskelig å avgrense og definere. Ei mye brukt lærebok i systemutvikling i Noreg, skrevet av ein av nestorane i faget, begynner slik:

> *"En kollega hevdet at det ikke er mulig å skrive en grunnbok i systemutvikling som andre aksepterer som en god førsteinnføring i emnet. Det er ikke mulig å oppnå enighet om hva en slik bok skal inneholde, sa han. Hvilke problemer skal den behandle? Hvilke begreper er det nødvendig å introdusere? Hvilke metoder og teknikker bør omtales? Han har noe rett. Systemutvikling er et nytt fagområde. Det er ennå ikke enighet om avgrensningen av faget, og hva som er det sentrale teoretiske grunnlaget i faget.(...) Den (boka) presenterer mitt syn på systemutvikling"* (Andersen 1989).

Andersens introduksjon er eit godt eksempel på kor vanskelig det er å definere systemutvikling som fagfelt, i det minste dersom ein skal stø seg på lærebøkene. Der vil ein sjå at fagfeltet ofte beskrivast som svært ueinsarta, uoversiktlig og vanskelig å definere. Det er ingen intern konsensus om korleis feltet skal avgrensast og kva aktivitetar og metodar det skal omfatte. Ja, det er ikkje eingong semje om kva fagfeltet skal heite. Den enkelte forfattar vel ofte å definere systemutvikling på ein måte som dekkjer hans hensikter med boka og slik han meiner systemutvikling skal gjørast, ofte etter mange år med praksis i databransjen/faget. Mesteparten av systemutviklingslitteraturen er svært foreskrivande og normativ. Forfattarane fortel lesarane korleis dei skal designe system, og beskriv kva verktøy, teknikkar og metodar dei foretrekk.

Det er fleire årsaker til at det er så liten grad av konsensus rundt defineringa av systemutvikling som fagfelt. Eg vil kort kommentere to av dei viktigaste. For det første er systemutvikling eit ungt fagfelt. For det andre er systemutvikling ein svært heterogen aktivitet. Vidare i dette notatet vil eg bruke tre ulike metodar i systemutvikling som grunnlag for ein diskusjon om korleis systemutviklingsmetodane handsamar det vi noe upresist kan kalle dei tidlige fasane i systemutviklingsprosjekta, som bl. a. omfattar dei fasane der kommunikasjonen med brukarane er hyppigast og der systemet skal spesifiserast i forhold til korleis det skal møte dei forretningsmessige mål.

2. Systemutvikling - framvekst og heterogenitet

Utviklinga av systemutvikling som fagfelt er tett kopla til den generelle historiske utviklinga av datorisering generelt. Maskinvareutviklinga har på mange måtar sett rammene for utviklinga av programvare. Det er ikkje mulig å gi noen fullstendig oversikt over framveksten av dataarbeid (dvs. arbeid knytt til utvikling av program- og maskinvare). I denne samanhengen vil eg berre skissere noen hovudtrekk i utviklinga av fagfeltet som er sentrale for å forstå situasjonen i dag (framstillinga i dette avsnittet bygger i hovudsak på Friedman 1989, Willcocks 1987, Norsk datahistorie 1989, Ofstad 1985). Det er mest fruktbart å ta utgangspunkt i korleis hardware og software har utvikla seg for å spore framveksten av datayrker og slik få ei forståing av kva som blir rekna som dataarbeid, og kven som gjør det. Det betyr bl. a. å analysere kva dataarbeid er i lyset av utviklinga av datasystem som raskt diffunderte varer, dvs. å studere utviklinga av datasystem som ein kommersialiseringsprosess. Friedman (1989) deler utviklinga av programvare inn i tre fasar; 1. fase frå 1954/55 til midt på 60-talet, andre fase frå midt på 60-talet til begynnelsen av 80-talet og 3. fase frå begynnelsen på 80 talet og framover.

Den første fasen begynte samtidig med starten på den kommersielle datoriseringa. Friedman kallar denne fasen *maskinvarebegrensingar*. Den var dominert av problem knytt til kostnader, påliteligheit og særeigenheiter i samband med betre utnytting og drift av maskinvare. Dei første maskinane var sett saman av ei enorm mengde radiorør og programmeringa bestod av å konstruere fysiske koplingar i maskina. I denne tida vart programmeringa gjort av dei som bygde, vedlikeholdt og modifiserte desse store maskinene. Palaez (1988) peiker på eit interessant poeng, nemlig at all vekt vart lagt på bygginga av maskina i den første tida, slik at dei første programmerarane, dei som konverterte problem til maskinforståelige algoritmar, var kvinner. Sagt på ein annan måte, det arbeidet desse kvinnene gjorde var ikkje rekna som eit datayrke. Med introduksjonen av lagra program vart det klart at algoritmeskriving var svært vanskelig og oppgåva vart hurtig og totalt maskulinisert. Programmering hadde mange trekk av å vere ein handtverksmessig aktivitet og

det var ingen etablert utdanning. Opplæring var prega av "læring-ved-gjøring" og det var stor grad av autonomi for programmerarar. Det var allereie då ein stor mangel på programmerarar og jobb-hopping var eit alternativ for å fremme karrieren.

Frå midten på 50-åra til ut i 70-åra gjennomgikk maskinvaren ei rivande utvikling, og ein såg tre generasjonar teknologi utvikle seg og erstatte dei føregåande i løpet av denne perioden. Utviklinga gikk frå radiorør, via transistorteknologi til logiske kretskort. Den teknologiske utviklinga frå 1. til 3. generasjonsmaskinar var ujamn for ulike delar av teknologien. Utviklinga innan prosessorteknologi (CPU), inn- og utmatingsenheiter, dataspråk, verktøy og operativsystema hadde alle ulik utvikling. Dette hadde effekt på kva typar arbeid som var nødvendig innan dataarbeid. Den første generasjon av "programmerar" var, som eg var inne på, folk som fysisk kopla program direkte inn i maskina og dei var maskinvare-ingeniørar, programvare-ingeniørar, operatørar, analytikarar, programmerar og kodarar, i ein og same person. Samtidig utvikla det seg eit spesialarbeidar-yrke, hullkort-operatørar, som var kvinner. Markussen (1985) peiker på at denne inndelinga i høg-kvalifiserte og låg-kvalifiserte jobbar innan databransjen har halde seg sidan 60-åra og trulig blitt forsterka.

Den andre fasen kallar Friedman *programvare-begrensingar*. Den var prega av at kostnadene knytt til innkjøp av maskiner minska, mens personkostnadene til drift og utvikling av programvare auka. IBMs 360-serie reduserte prisane på datamaskiner betraktelig. I denne fasen vart også skiljet mellom program- og maskinvare introdusert og programmering vart på mange måtar det arketypiske uttrykket av kva dataarbeid dreidde seg om. Ein såg mot slutten av 60-åra også ei aukande forståing av problema knytt til programvareutvikling og begrepet "programvare som flaskehals" vart introdusert. Uttrykket viste til gapet mellom dei stadig aukande krava om større og betre system, og problema bransjen hadde med å møte dei. Det vart sett fokus på evnene programvare-utviklarane hadde til å levere system til den oppsette tidsfristen og innafor budsjettet. Det vart også eit aukande fokus på korleis ein skulle styre utviklings-staben og ein såg framvekst av direkte kontroll som styringsmetode. Ein såg også framvekst av ein uavhengig programvaresektor, noe som medførte at entreprenørskap vart mulig som karriereveg. Det var vidare ei utvikling av det formelle utdanningstilbodet som starta eit gryande profesjonaliseringsframstøt.

I denne perioden skjedde det også ei vidareutvikling av operativsystema. Det hadde bl. a. den effekten at oppgåva som formidlar mellom maskina og brukarane delvis vart flytta frå programmerar i brukar-organisasjonen til kjernen i maskina. Det medførte bl. a. framveksten av ein data-industri som tiltrakk seg mange programmerar. Det førte også til innovasjonar innan dataspråk. Utviklinga av kompilatorar gjorde det mulig å skilje kildeprogram (som oversett assembler-instruksjonar) og objekt-program (som gjør om desse til

maskinlesbar form). Det førstnemnde blir det framveksande applikasjons-programmerarens domene, mens den sistnemnde har utvikla seg som system-programmerarens felt.

Den tredje fasen som Friedman kallar *bruker-relasjons-begrensingar* er vi framleis inne i. Den var (er) prega av at stadig fleire system vart (blir) retta inn mot "høgare" funksjonar i bedriftene. Det førte til at leiing på høgare nivå i bedrifta vart sterkare involvert i systembruk og såleis også interesserte i utvikling av systema. Dataavdelingane kom i sterkare grad i direkte kontakt med resten av organisasjonen. Mye av den kompetansen som var nødvendig innan datautvikling/drift kom til å likne meir og meir på generelle leiarferdigheiter. Samtidig vart det nødvendig for dataavdelingane å bli betre i å selje sine tenester og kontakten med brukarar og evne til å sjå deira problem vart stadig viktigare. Ettersom computing har komme over i sin 4. generasjon, basert på mikrochip-teknologi, har utviklinga av høg-nivåspråk skutt fart. Assembler-programmering var i stor grad maskinorientert. Mange programmerar i brukarorganisasjonane fann det vanskelig å bruke datamaskina til å løyse forretningsproblem eller administrative oppgåver i forhold til maskinproblem. Cobol, som såg dagens lys i 1959, vart det dominerande språket for applikasjonsprogrammerarar som arbeidde med forretningsproblem. Presset for å ta i bruk Cobol og dens etterfølgjarar viser kor viktig brukarorientert computing har vore i utviklinga av datayrke. Det er i dag allmenn semje om at begrensingar i forhold til brukarane er ei av hovudutfordringane for programvareutvikling spesielt og feltet computing generelt i tida framover.

På same måte som ein såg at komponentar i datasystema var fragmenterte og at nye verktøy, dataspråk og teknikkar vart utvikla for å løyse spesielle del-løysingar, utvikla også handverket programmering seg frå ein esoterisk einheit til ei rekke oppdelte oppgåver og aktivitetar. Thierny (1991) relaterer denne dramatiske eksplosjonen i yrkesmessige under-spesialitetar til fleire forhold: For det første til polariteten i den organisasjonsmessige lokaliseringa av dei som arbeider med data (mellom dataindustrien og brukarorganisasjonar som var spreidde over mange ulike sektorar), for det andre utviklinga av råmateriale som dataarbeid blir utført på (maskina, operativsystemet, applikasjonar, dataspråk og problem), for det tredje den store veksten i mengden og type av datatarbeid som blir utført. Thierny legg likevel svært stor vekt på den leiarmessige effekten dette hadde i forhold til organiseringa og kontrollen av dataarbeid.

> *"From the 60s onward we find a remarkable level of manipulation of job titles, of career structures, of requirements practices, of departemental structures, and of monitoring strategies over computing work which, between them, operate to diversify and fragment computing"* (op. cit).

Eit av nøkkelproblema med å avgjøre når ein type arbeid har blitt eit yrke og

når det har etablert seg sub-yrke som er distinkt forskjellig frå kvarandre er at ein må bli samde om kva målestokk ein skal bruke. Skal ein skilje på jobb tittel, på nivå og type av kunnskap som ein innehar, på ansvarsområde, på rolle i organisasjonen, osv? Innan studiar av dataarbeid er det i liten grad blitt semje om slike målestokkar, og ein koplar ofte jobbtittel som f. eks. programmerar med det ein meiner slike folk gjør. Sjølv om det ikkje er noen klar definisjon av kva dataarbeid faktisk består av, og at det å lage ein slik har vist seg svært vanskelig, er det likevel ikkje til hinder for at vi meiner å kunne seie at dataarbeid er eit distinkt yrke som ein person anten tilhøyrer eller ikkje.

> *"The complexity of job structuring in computing work is likely to have implications for how we may conceptualize the problem of computing expertise for it is clear from the litterature that the proliferation of jobs and titles within computing refer as much to issues of status, responsibility and trust as to changes in the actual content of job task"* (op. cit.).

Historisk har utviklinga av computing vore prega av raske og store skift og teknologien har stilt ulike krav til "operatørane". Det har medført ei aukande grad av spesialisering og diversifisering i oppgåver og kunnskapar. Samtidig har ein sett ei organisasjonsmessig og stillingsmessig oppdeling og fragmentering som ytterligare har medverka til å splitte opp feltet.

Ein annan viktig årsak til at systemutvikling er så mangelfullt definert og at ein finn så mange ulike beskrivingar av feltet, er den svært heterogene karakteren faget har. Denne heterogeniteten er i særlig grad, samanlikna med dei fleste ingeniørfag, nedfelt som ein del av faget. Systemutvikling er ein fleirfaglig disiplin. Ein systemutviklar må ideelt sett vere teknolog, sosiolog og psykolog. Teknolog i den forstand at det er teknologi han skal utforme, sosiolog fordi mye av verksemda knyter seg til organisasjonsutvikling og organisasjonsteoretiske spørsmål, og psykolog fordi systemutvikling involverer slike ting som å få arbeidsgrupper til å fungere og å utvikle gode brukargrensesnitt. Systemutvikling inneber, som mye anna ingeniør/teknolog arbeid, svært mange og til dels svært ulike oppgåver. Systemutviklaren må gjøre så ulike ting som prosjektplanlegjing og prosjektleiing, intervjuing av brukarar, analyse av brukarorganisasjonen og dei arbeidsoppgåver som skal automatiserast, systembeskriving, programmering, vedlikehald og brukaropplæring. Det betyr at det krevst kvalifikasjonar frå mange fagområde.

Den kjende engelske systemteoretikaren Michael Jackson eksemplifiserer heterogeniteten i faget utmerkt når han startar boka si om systemutvikling slik:

> *"We use the term "development" to cover the range of activities usually carried out by people whose job-titles are "system analysts", "system designer", "program designer" or "programmer". These activities include requirements spcification, functional specification, logical system design, application system design, physical system design, program specification and design,*

program implementation, and system and program maintenance. (...) There is very little agreement on the definitions of these activities, and even less on the meanings of the job titles "analyst", "designer" and "programmer". JSD (Jackson systems development) cuts across even the very small area of agreement that does exist, redrawing the boundries between activities and between jobs, so that the existing names and titles become almost entirely inappropriate (1983)".

Vi ser at Jackson beskriv systemutvikling som eit svært heterogent felt. Han trur i liten grad det er mulig å komme til einigheit om definering av faget og enno mindre kva innhald dei ulike sub-gruppene innan yrket skal ha. Sjølv om aktivitetane er breie, er begrepsbruken teknisk. Den teknologiske sida er slik sett dominerande. Generelt er systemutvikling som fagfelt svært "mangelfullt" definert. Det eksisterer nesten like mange definisjonar på kva systemutvikling er, og kva aktivitetar det inneheld som det eksisterer lærebøker og annan innføringslitteratur på området. Mackay og Lane (1990) peiker på at systemanalyse (utvikling) er ein spesielt interessant profesjon å studere samspelet mellom teknologien og det sosiale i, fordi den er sett saman av både det å kartleggje og det å modellere den sosiale verda, og samtidig å representere den sosiale verda i maskiner.

Systemutvikling som teknologiutvikling fokuserer på ulike teoriar og metodar for å representere og implementere "kodifiserbar" (eller kodifisert) kunnskap i ei datamaskin. Den første delen av dette handlar i stor grad om å representere (rekonstruere) sosiale aktivitetar i form av (oftast) dataflytdiagram. Diagramma kan sjåast som representasjonar (eller bilete) av dei oppgåvene som brukaren skal utføre og forholdet mellom den "kodifiserte kunnskapen" og "omverda". Det inneber bl. a. ei organisering av aktivitetane som skal automatiserast, både ei definering av oppgåvene i forhold til kvarandre (oppdeling i programmodular) og ei presisering av kva rekkefølgje dei skal skje i (kva sekvensar programmodulane skal utførast i). Vidare handlar teknologidelen om å implementere denne kunnskapen i maskina, dvs. overføre informasjonen frå papir til digitale, elektroniske signal. Den elektroniske representasjonen må vere i ei form som gjør at den fungerer som eit system, at det fungerer saman med resten av programvaren og med den eksisterande eller nyerverva maskinvaren slik at systemet faktisk kan fungere maskinelt, og i samarbeid med brukarane.

Systemutvikling som organisasjonsorientert forandringsarbeid framstår som å i stor grad dreie seg om organiseringa av sjølve utviklingsprosessen, dvs. prosessen som skal leie til defineringa av kva datasystemet skal omfatte og kva det skal utføre. Kven skal involverast i prosessen og korleis? Vidare er sentrale spørsmål korleis utviklingsprosessen, der medarbeidarar i ulik grad tar del, påverkar organisasjonen og dei som arbeider der. Dvs. korleis sjølve handlinga knytt til prosjektet fører til ny kunnskap og nye måtar å organisere verksemda

på, som igjen kan føre til nye krav til informasjonssystemet. Eit anna sentralt område er kva effektar sjølve datasystemet, med sine nye oppgåver og ny arbeidsorganisering får og korleis organisasjonen og menneska i den tilpassar seg teknologien eller tilpassar teknologien.

Den aktiviteten som ligg i grenselandet mellom desse (eller som er nødvendig for begge) ytterpunkta, er planlegginga av systemet, problemdefineringa og ikkje minst informasjonsinnhentinga frå den sosiale verda. Det er på mange måtar basisaktiviteten i systemutviklinga, det som skal bygge "grunnmuren og reisverket" i systemet. Korleis kan ein etablere kodifiserbar kunnskap (som representerer dei arbeidsoperasjonane som skal automatiserast) og samtidig i den prosessen, korleis kan ein skape konsensus rundt denne slik at han kan fungere som grunnlag for systemet?

All systemutviklingslitteratur legg vekt på at dette er aktivitetar som må utførast grundig og godt, men kva det faktisk inneber og korleis det praktisk skal utførast er i liten grad problematisert. Kva generelle, prinsipielle problem er til stades i den prosessen? Det blir i stor grad opp til den enkelte analytikars skjønn korleis ein vel å løyse desse oppgåvene. Litteraturen viser oftast til at dette er ein type kunnskap som kjem med erfaring i praktisk systemutviklingsarbeid og går ikkje meir i detalj i korleis oppgåvene skal løysast. Det er ein klar kontrast mellom den betydning denne fasen av systemutviklingsarbeidet blir tillagt i dei fleste lærebøker og kor lite konkret dei beskriv problem og måtar å løyse desse på. Det er også ein klar kontrast til dei meir teknisk orienterte delane av systemutviklinga som blir svært detaljert handsama med oppskrifter for korleis ein skal gå fram. I dei tidlige fasane av systemutviklingsprosessen må analytikaren handtere ei ofte kompleks og uoversiktlig virkeligheit og han må strukturere informasjonen om den sosiale verda slik at den "passar til" krava han møter frå teknologien. Det betyr bl. a. at han først må forstå, gjennom kommunikasjon med brukarane og lesing av manualar osv., den eksisterande kunnskapen om dei arbeidsoperasjonane som skal automatiserast. Det er ein aktivitet han må gjennomføre anten han er hovudsaklig teknologiorientert eller orientert mot organisasjonsutvikling. Han må også søke å skape rimelig grad av konsensus om si tolking av problema og organisasjonen som systemet skal fungere i. Det betyr at han må "overtale" oppdragsgjevarane (og brukarane) om at hans måte å sjå problema på og såleis den løysinga han foreslår for å løyse desse er korrekte. Den kanskje viktigaste bestemmande faktoren er likevel teknologien, både fordi den set i manges auge absolutte grenser (krav) og fordi systemutvikling som fag tradisjonelt er mest konsentrert om grensene som teknologien set. Slik blir det å gjennomføre (og strukturere) utviklingsprosessen er ei slags mellomaktivitet - mellom teknologien og organisasjonen/personane

3. Arbeidet med å lage ein brukarkravspesifikasjon

Eg viste ovanfor at det var ei klar motsetning mellom kor stor vekt systemutviklingslitteraturen la på dei tidlige fasane i systemutviklingsprosjekta og korleis dei blir handsama i bøkene. Eg skal i dette avsnittet kort vise eksempel på korleis desse fasane blir beskrevet og kva det er vanlig å innlemme av aktivitetar. Andersen begrunner kvifor dei tidlige fasane er så viktige slik:

> *"Systemutvikling er en meget omfattende oppgave. (...) Jeg legger størst vekt på systemeringen. Innenfor systemeringen vier jeg analysefasen størst oppmerksomhet. Begrunnelsen for dette er at analysen av informasjonssystemet kan sies å være den viktigste oppgaven ved systemutviklingen. Den er viktigst i den forstand at hvis den gjøres dårlig så blir informasjonssystemet dårlig, uansett hva som senere gjøres. Det er ikke mulig å kompensere for dårlig analysearbeid i de etterfølgende fasar"* (op. cit.).

Analysearbeidet blir med andre ord avgjørande for kor godt eit informasjonssystem kan bli og feil i denne fasen av utviklingsprosjektet blir svært kostbare å rette opp i seinare fasar. Yordon beskriv forholdet mellom analysefasen og resten av systemutviklingsaktivitetane slik:

> *"The most important activity in a software development project is analysis. If analysis is done well, a mediocre design can be tolerated and the code can be written by trained chimpanzees. But if systems analysis is done poorly, the best design and the best code will simply allow the project to arrive at a disaster sooner than would otherwise have been possible"* (1988).

Utgangspunktet for alt systemutviklingsarbeid er, som eg har vore inne på, å utvikle ein spesifikasjon over kva problem systemet skal løyse og korleis det skal løyse det. Yourdon definerer det slik:

> *"The purpose of the analysis is to determine presicely what the users want the system to accomplish - regardless of how it is implemented"* (op. cit.).

Sommerville (1992) utvidar denne definisjonen noe med å også ta med dei omgivelsane som systemet skal fungere innafor og analyse er for han:

> *"The process of establishing the services the system should provide and the constraints under which it must operate"* (1992).

Det sentrale i arbeidet med å definere kva systemet skal gjøre og forholdet til andre delar av organisasjonen (rammevilkår) er innhenting og analyse av brukarkrav. Resultatet av denne prosessen er ein brukarkravspesifikasjon som er det første formelle dokumentet i utviklingsprosessen. Brukarbehov og brukarkrav er ikkje det same. Behova er ofte vage og funksjonsorienterte, f. eks. ein bedrift kan finne ut at dei vil ha eit rekneskapssystem. Det er da nødvendig å innhente og analysere informasjon om det behovet og på basis av

den informasjonen designe eit system som kan løyse det. Denne løysinga må da i størst mulig grad vere i samsvar med dei krava brukarane stiller til systemet. Rittel og Webber (1989) beskriv dette som "wicked problems", dvs. problem som ikkje har noen definitiv formulering eller definering. Det er fleire årsaker til at spesifisering av problem som systemet skal løyse ofte er ein komplisert prosess.

For det første, store informasjonsteknologi-system blir ofte utvikla for å forbetre status quo i tilfelle der det ikkje eksisterer noe tidligare system eller der det noverande systemet ikkje er godt nok. Sjølv om ein er i stand til å peike på feil ved det eksisterande systemet er det langt vanskeligare å vurdere effektane av det nye systemet på organisasjonen. For det andre, store system har ofte ei svært heterogen brukargruppe som har forskjellige og noen gonger motstridande behov og prioriteringar. Det endelige systemet må derfor bli eit kompromiss. For det tredje, dei som bestiller eit system (dei som betaler for det) og (slutt)brukarane er sjeldan dei same personane. Bestillarane har krav i forhold til organisasjonsmessige og budsjettmessige vilkår. Dei er ofte i konflikt med brukarbehov.

Det er ei uklar skiljeline mellom systemmål og systemkrav. Ein vanlig distinksjon er at systemmål er generelle eigenskapar som systemet skal ha og systemkrav skal kunne testast. Eit eksempel på mål kan vere at systemet skal vere brukarvenlig, mens det samanfallande kravet kan vere at alle kommandoar skal kunne utførast ved hjelp av menyar. Denne uklårheita er også tilstades i spesifiseringa av kva brukarkravspesifikasjon faktisk skal vere. Det kan variere frå ei generell beskriving i naturlig språk som spesifiserer kva systemet skal utføre til matematikkbaserte formelle systemspesifikasjonar.

Det er ei stadig kjelde til forvirring og problem at systemutviklaren skal dokumentere systemet på ein slik måte at det er forståelig for potensielle brukarar og samtidig skal produsere ein systemspesifikasjon som kan fungere som basis for kontrakt mellom bestillar og utviklar av systemet. Generelt ønskjer brukarar ei høgare grad av generell beskriving av systemet enn det som ein spesifikasjon som skal fungere som kontrakt kan gi. Det er også umulig å konstruere ein detaljert spesifikasjon utan å gjøre litt design først, noe som ytterligare tilslører skiljet mellom brukarkrav- og designspesifikasjon.

Sommerville viser til at det medfører at spesifikasjon må produserast på ulike abstraksjonsnivå med ulike lesarar som målgruppe. Ein brukarkravdefinisjon er ei klargjøring, i naturlig språk, av kva brukartenester systemet skal gi. Dette må skrivast på ein slik måte at det er forståelig for prosjektleiinga i både produsent og brukar (i kontraktssamanheng) og for bestillar og brukar av systemet. Ein kravspesifikasjon er eit strukturert dokument som penslar ut kva systemet skal gjøre i større detalj. Dette dokumentet skal vere så presist at det skal kunne fungere som kontrakt. Det må vere forståelig for den tekniske staben i både produsent og brukar av systemet. I utviklinga av dette kan ein bruke formelle spesifikasjonsteknikkar. Ein

programvarespesifikasjon (designspesifikasjon) er ei abstrakt beskriving av den programvare som er basis for design og implementering. Det bør vere ei klar samanheng mellom denne og kravspesifikasjonen, og målgruppa for denne er dei som designar systemet.

Ofte vil ein støyte på store problem i å definere krava til systemet, bl. a. fordi ein har for liten kunnskap om applikasjonsdomenet. I slike tilfelle er det meir passande, i følgje Sommerville, å bruke ein prosessmodell basert på prototyping heller enn den klassiske forssefallsmodellen. Den sistnemnde kan f. eks. innebere følgjande arbeidsoperasjonar:

Muligheitsstudie. Det inneber å utvikle eit estimat over om det er mulig å tilfredsstille dei identifiserte brukarbehova med eksisterande program- og maskinvare, om det foreslåtte systemet vil bli kost-effektivt frå eit forretningsmessig vurdering og om det er mulig å utvikle innafor dei budsjetta som er stilt til rådvelde.

Innhenting og analyse av brukarkrav. Det er prosessen som skal komme fram til brukarkrava. Det gjørast gjennom observasjon av det eksisterande systemet, samtalar med potensielle brukarar osv.

Definering av krav. Det inneber å definere ein systemmodell og bruke denne som basis for ei abstrakt beskriving av systemkrava. Dette skal primært vere eit dokument som beskriv systemet sett frå brukarens ståstad.

Spesifisering av krav. Det er utvikling av ei detaljert og presis beskriving av systemkrava som skal fungere som basis for å utarbeide ein kontrakt. Utarbeiding av denne vil ofte gå parallelt med design og desse to aktivitetane vil påverke kvarandre undervegs.

Kravspesifikasjonen definerer systemet og er derfor ofte basis for kontrakten mellom utviklar og bestillar. Alle krav som er spesifiserte i ein slik kontrakt skal i prinsippet vere komplette og konsistente. I praksis er dette umulig å få til, særlig dersom dokumentet er i naturlig språk. Heininger (1980) hevdar at eit kravspesifikasjonen må tilfredsstille seks krav:

1. Den må berre spesifisere eksterne eigenskapar ved systemet.
2. Den må spesifisere rammevilkåra på implementeringa.
3. Den må vere lett å endre.
4. Den må fungere som eit referanseverktøy for vedlikehald.
5. Den må nedteikne antakelsar om systemets livssyklus.
6. Den må karakterisere akseptable tiltak på uønskte effektar.

Dette dokumentet er ein kombinasjon av kravdefineringa og kravspesifikasjonen og krev ofte eit svært omfattande arbeid for å utvikle.

Kravdefineringa er ein abstrakt beskriving av det systemet skal gjøre, og dei rammevilkåra det må arbeide under. Det skal berre beskrive dei eksterne funksjonane til systemet og ikkje vere opptatt av korleis dei kan realiserast. Det er vanlig å dele krav inn i to. For det første, *funksjonelle systemkrav*, som er dei tenestene brukarane forventar seg av systemet. For det andre, *ikkje-*

funksjonelle krav, som er ei beskriving av dei rammevilkåra som systemet må arbeide under, f. eks. i forhold til responstid.

Utviklinga av krav til systemet fokuserer ofte på muligheitene programvare har, forretningsmål og interaksjon med eksisterande system. Ettersom desse krava blir spesifiserte etablerast det også ei betre forståing av brukarbehova. Dette vil få ein tilbakeførings-effekt som fører til at krava vil bli endra. Sidan store prosjekt går over mange år vil krava naturlig endre seg i løpet av denne perioden, noe som utviklaren må vere merksam på. Konteksten systemet utviklast innafor kan også endrast i løpet av utviklingsprosessen. Brukaranes auka kunnskap om muligheitene teknologien gir, kan også føre til nye eller endra krav. Dette er ein iterativ prosess med stadig endring og betre presisjon på krava osv. Det er derfor nødvendig at kravspesifikasjonen organiserast på ein slik måte at det er mulig å inkorporere endring utan svært omfattande omskriving.

Det er ofte hevda at dårlig spesifikasjon av brukarkrav er ei av hovudårsakene til at utvikling av datasystem slår feil. Det blir peika på fleire årsaker til at tradisjonelle og mye brukte strukturerte metodar er spesielt svake på dette området, dels at det kjem deira systemorientering og dels at dei ikkje har adekvate teknikkar for å hente inn brukarkrav.

Fleire har peika på at strukturerte metodar er system- eller dataorienterte, og at dei neglisjerer sluttbrukaren (sjå f. eks. Gould og Lewis (1985) og Sutcliff (1988)). Eit problem er at strukturerte metodar ikkje skil tilstrekkelig mellom krav frå sluttbrukaren, og forretningsmessige krav eller systemkrav. Eit meir grunnleggande problem er at strukturerte metodar ikkje tilbyr særlig i form av metode for å samle inn og organisere brukarkrav. Dersom ein ser på dei viktigaste strukturerte metodane i bruk i dag finn ein at dei er generelt svake i beskrivinga av kravinnhentingsfasen, særlig når det gjeld det å forstå brukaranes krav i forhold til å forstå systemets krav.

SSADM (Structured System Analysis and Design Methodology) generer f. eks. eit dokument som heiter "problems/requirements list", som i hovudsak er ei liste over systemproblem og systemkrav. Det tar i liten grad omsyn til brukarkrav. Brukarane er involverte i å revidere denne lista, men den er ikkje oppgåveorientert og det er vanskelig å sjå korleis brukarane kan gi fornuftige kommentarar til dei abstrakte systemorienterte momenta som den inneheld. Det er sentralt at brukarperspektivet blir tatt med i betraktning gjennom heile analysefasen og at det blir reflektert tilbake til brukaren når dei skal vurdere kor adekvat eller komplett analysen er.

Andersen deler dei ulike hjelpemidla i systemutvikling inn i fire kategoriar, modell, metode, beskrivingsteknikk og verktøy. Begrepa kan seiast å danne eit hierarki og han beskriv samanhengen mellom dei slik:

> *"En modell består av et antall fasar (eller steg eller områder). En metode kan bli brukt i en eller flere fasar av modellen. En beskrivelsesteknikk kan bli brukt i en eller flere metodar. Et verktøy kan*

støtte bruken av en eller flere metodar og/eller en eller flere beskrivelseteknikker" (op. cit.).

Ein modell[1] er ein oversikt over utviklingsarbeidet og den beskriv i grove trekk kva arbeid som skal utførast og kven som skal gjøre det. Modellen er ofte bygd opp av deler (fasar, steg, område). Ein metode er ei meir detaljert beskriving av framgangsmåte for å løyse eit bestemt problem. Ein metode kan karakteriserast av kva bruksområde den har, kva arbeid som skal gjørast og korleis det skal organiserast og kva beskrivingsteknikkar som skal brukast. Ein beskrivingsteknikk er ei oppskrift på korleis ein skal lage ei beskriving. Oppskrifta er eit sett med reglar for korleis noe i virkeligheita kan uttrykkjast i ei beskriving. Dei kan vere formaliserte eller ikkje-formaliserte og dokumenterte eller ikkje-dokumenterte. Formaliserte har presise reglar for samanhengen mellom kva som skal beskrivast og dei symbol eller andre uttrykksformer som teknikken bruker. Ein teknikk kan ha i seg både formaliserte og ikkje-formaliserte. Dokumenterte beskrivingar har ei form som gjør dei permanente, dei kan såleis studerast seinare i prosessen. Systemutvikling har innslag av både formaliserte og ikkje-formaliserte teknikkar, men hovudvekta er på den førstnemnde. Dokumentasjon er sentral i all systemutvikling, sjølv om det er eit punkt det ofte blir synda mot. Symbolske beskrivingsteknikkar gir formaliserte beskrivingar som er relativt lette å endre og dette er den mest vanlige innan systemutvikling. Ein formalisert og symbolsk beskrivingsteknikk er kjenneteikna ved; kva type symbol som er tillate, kva forbindelsar som er tillate mellom dei ulike symbola, korleis naturlig språk blir brukt i samband med symbola og korleis ein bruker nivå (hierarki) i beskrivingane. Beskrivingsteknikkane er eit av dei viktigaste hjelpemidla for systemutviklarane og eit sentralt kommunikasjonsmiddel overfor brukarane og til andre (og dei sjølve) i prosjektgruppa. Verktøy er eit fysisk hjelpemiddel og som kan vere alt frå papir og blyant til dataprogram, f. eks. CASE (Computer Aided Software Engineering).

Eg har i dette avsnittet kort diskutert noen sentrale begrep og modellar i forhold til dei tidlige fasane av systemutviklingsprosessen. Som det går fram av drøftinga rundt desse er det ei spenning mellom maskinas krav til formalisering og presisjon og brukaranes krav til informasjonsrikdom og forståelig presentasjonsteknikkar i samband med utvikling av brukarkravspesifikasjonar. Dette er ei spenning som i liten grad blir problematisert i litteraturen.

3. Drøfting av noen modellar for systemutvikling

Eg har vald å vise tre ulike modellars syn på og handsaming av dei tidlige

[1] Den påfølgande diskusjonen baserer seg i hovudsak på Andersen (1989).

fasane i systemutviklinga. Desse modellane er alle svært utbreidde i undervisningssamanheng, slik at det kan kanskje gi ein indikasjon på kva ballast studentar i systemutvikling får med seg i desse spørsmåla. Dei tre modellane er SASD (Structured Analysis and Structured Design), ISAC (Information Systems work and Analysis of Changes), og JSD (Jacksons System Design)[2].

SASD er ein av dei mest kjende systemutviklingsmodellane og kan stå som eksempel på ei heil gruppe av beslekta modellar, f. eks. SSADM, SSA og Gane og Sarson. Den er utvikla i USA, i konsulentfirmaet Yourdon som kanskje er det miljøet som mest forbindast med SASD. Sentrale namn er Tom DeMacro, Chris Gane, Trish Sarson og Ed Yourdon. Modellen vart utvikla på slutten av 1970-åra og har fått vid anvendelse seinare (Andersen 1989).

Yourdons utgangspunkt for å utvikle ein eigen modell var stor skepsis til det han kallar konvensjonelle metodar, dvs. at systemutviklaren gjør seg ferdig med ein fase i utviklingsprosjektet før neste fase innleiast. Han hevdar at det er tre hovudårsaker til at konvensjonelle tilnærmingar til systemutviklingsprosjekt feilar. Det første og mest alvorlige problemet er dårlig systemanalyse. Det betyr at arbeidet med å framskaffe ei god brukarkravspesifikasjon er for dårlig. Det endelige resultatet blir deretter. Men Yordon forklarer ikkje dårlig systemanalyse med problem med å faktisk skaffe seg slik kunnskap, for dårlig kommunikasjon med brukaruniverset, og ulike krav og "forståingsmodellar" mellom det sosiale og det tekniske. Han fokuserer på at arbeidet med å *dokumentere* brukarkravspesifikasjonen er for dårlig. Det gjør det umulig å forstå det som er produsert, både av utviklar og brukar. Det gjør det sjølvsagt også svært vanskelig å endre kravspesifikasjonen etter at dei er ferdige.

Det andre problemet i følgje Yordon er design av kode og dokumentasjon. Mange har sin eigen stil i design av programmet og er berre opptatt av om det verkar, ikkje korleis det ser ut for brukaren eller for andre systemutviklarar som skal gjøre oppdateringar seinare. Det gjør det umulig å definere ein objektiv kvalitetstandard og prosjektleiinga kan ikkje på ein meiningsfull måte overvake framgangen til dei einskilde programmerarane og i prosessen totalt. Korleis det i det heile er mulig å definere ein objektiv kvalitetsstandard for programmering er meir uklart hos Yordon. Fleire har hevda at styring av systemutviklingsarbeidet er svært vanskelig, bl. a. fordi det ikkje er enkelt å definere noen meiningsfulle kriteria for å overvake kvaliteten og framgangen i arbeidet (Mackay og Lane 1990).

Det tredje problemet er at konvensjonelle prosjekt blir utvikla nedanfrå og opp. Det vil seie at når systemet er designa og koda blir det testa først på modulnivå, så på programnivå, så på subsystemnivå og til slutt på systemnivå.

[2] Framstillinga i denne delen bygger på Andersen (1989), Jackson (1975 og 1983), Langefors (1967 og 1970) og Yourdon 1988.

Alle desse problema er i følgje Yourdon tekniske problem som kan løysast ved hjelp av strukturert analyse, strukturert design og strukturert programmering. Strukturert metode er karakterisert av betre verktøy for å uttrykkje brukarkravspesifikasjonar, vekt på kvalitet i design og betre kode, og top-down systemutvikling. Det sentrale begrepet innan strukturert metode er den strukturerte livssyklusmodellen. Skiljet mellom denne og den klassiske går i hovudsak på at den sistnemnde er ein nedanfrå og opp implementering av systemet, og at modellen insisterer på ein lineær, sekvensiell progresjon frå ein fase til den neste. Yourdon beskriv problema med den slik:

> *"The bottom-up implementation approach is a good one for assembling automobiles on an assembley-line - but only after the prototype model has been thoroughly debugged. Unfortunately, most of us in the computer field are still producing one-of-a-kind systems"* (op. cit.).

Det betyr bl. a. at det ikkje er mulig å produsere feilfrie prototypar som så skal masseproduserast. Dei fleste system som utviklast er unike og må gjennom heile prosessen frå start til slut.

Livssyklusmodellen, dvs. oppdeling av utviklingsprosjektet i fasar eller aktivitetar er eit av dei mest berande elementa i all systemutvikling. Yordons bok heiter da også "Managing the System Life Cycle". Friedman beskriv livssyklusmodellen slik:

> *"The term "computer system life cycle" is often used in the computing litterature to describe an ordred set of activities, or phases, which combine to make up the conception, development, use and eventual replacement of a new computer-based system"* (op. cit.)

Livssyklusmodellen gir eit bilete av arbeidsoppgåvene, dei forskjellige typar av beskrivingar som må utarbeidast, kva tidsrekkefølgje dei har og når dei skal vere ferdige (milepælar), og dei forskjellige interessentane sin rolle i utviklingsarbeidet. Bakgrunnen for begrepet er at systemutviklinga følgjer informasjonssystemets "liv", frå tanken om eit nytt informasjonssystem oppstår til eit ferdig innført system ligg føre.

Peters (1988) peiker på at livssyklusmodellen har svært mange og til dels ganske ulike definisjonar, men trekk fram følgjande fellestrekk. Ein livssyklusmodell blir brukt til å forklare og hjelpe oss til å forstå utviklings- og vedlikehaldsprosessen av programvare. Den inneber ein stegvis nedbryting av utviklingsprosessen. Den inkluderer ei liste over ting som må gjørast. Den er både eit leiingsmessig og eit teknisk verktøy for å organisere, planleggje, tidfeste og å kontrollere dei aktivitetane som er knytt til programvare-utvikling og -vedlikehald. Vi ser at han beskriv livssyklusmodellen både som ein aktivitet og som ein plan eller guide. Modellen er prosess og aktivitetsorientert, og nyttig både for dei som leier arbeidet og dei som utfører det.

Det er i følgje Yourdon tre hensikter med livssyklusmodellen. For det

første å definere aktivitetane som skal utførast i utviklingsprosjektet. For det andre å sikre at alle utviklingsprosjekt følgjer noenlunde same lest. Og for det tredje å etablere milepælar for viktige avgjerder og som hjelpemiddel for leiinga. Yourdon identifiserer tre ulike entitetar i systemutviklingsprosjekt som er sentrale i livssyklusmodellen. Det er brukarar, leiing og operasjonar.

Det er i følgje Yordon tre ulike typar brukarar, den strategiske, den taktiske og den operasjonelle. Brukaromgrepet generelt vel Yourdon å forstå slik: *(...) this is the person who ultimately pays for the system (and thus, indirectly, pays the salaries of the people who build the system).*
Den strategiske brukaren er på høgt nivå i organisasjonen og *"primarily concerned with the long-term profitability or return on investment that will be generated by the proposed new system"*. Den strategiske brukaren er involvert i dei tidlige fasane av utviklingsprosjektet. Den taktiske brukaren er på mellomleiarnivå og er mest opptatt av at utviklingsprosjektet ikkje skal forstyrre "hans" arbeidarar i jobben. Derfor vil dei ofte seie: *"Don't talk to my people - they're far too busy! I'll tell you everything you need to know about their jobs."* Den operasjonelle brukaren er funksjonær. Dette er personen som har mest kontakt med systemet når det er i bruk. Operasjonelle brukarar er opptatt av slike ting som menneske-maskin dialog og feilmeldingar.

På same måte som brukar kan bety fleire ting er også leiing fleirtydig. Noen gonger er leiing og brukar same person, f. eks. ein strategisk brukar. I andre tilfelle er leiing ei styringsgruppe eller styret i bedrifta. For Yourdon er det sentrale å vere i stand til å identifisere ein person (eller ei gruppe av personar) som avgjør finansiering av prosjektet og som gir organisasjonsmessige avgrensingar, f. eks. bestemmer at all programvare skal køyre på same type maskinar.

Den tredje entiteten er operasjonar eller operator. Med det meiner Yourdon den personen som er ansvarlig for den daglige køyringa av datateknologien i bedrifta som systemet vil bli ein del av. Oftast vil systemet gå saman med mange andre system, og interaksjonen med dei må identifiserast. Koplinga til resten av datateknologien gir ofte føringar/avgrensingar for korleis systemet kan byggast.

Dei tre entitetane i den strukturerte modellen er alle i interaksjon med dei ni aktivitetane som Yourdon deler livssyklusmodellen opp i. Dei er følgjande;

Survey. Denne aktiviteten startar med førespurnaden om at noe skal automatiserast. Det viktigaste målet med survey er å identifisere manglar i brukarmiljøet, å etablere nye mål, å avgjøre om det er mulig å automatisere verksemda, å foreslå noen akseptable scenario og å lage eit prosjektcharter som kan fungere som ei rettleiing i resten av prosjektet.

Analyse. Målet med analysen er å transformere dei to viktigaste inspela, brukarstrategi og prosjektcharter, til ein strukturert spesifikasjon. Det betyr bl. a. å modellere brukarmiljøet ved hjelp av dataflytdiagram,

entitetsrelasjonsdiagram og andre verktøy.

Design. Designaktiviteten konsentrerer seg om å allokere delar av spesifikasjonen til passande prosessorar (CPU og/eller menneske) og til passande oppgåver innan kvar prosessor. Innafor kvar oppgåve skal design utvikle eit hierarki av programmodular og grensesnitt mellom desse for å implementere spesifikasjonen frå analysen.

Implementering. Denne aktiviteten inkluderer koding og integrasjon av modular i eit stadig meir komplett skjelett av det ferdige systemet. Det betyr bruk av både strukturert programmering og top-down utvikling.

Generering av aksepttest. Den strukturerte spesifikasjonen skal innehalde all nødvendig informasjon for å definere eit akseptabelt system for brukaren.

Kvalitetssikring. Det er den endelige testen på om systemet er akseptabelt. Innspel til denne aktiviteten er data frå aksepttesten og eit integrert system.

Prosedyrebeskriving. Det inkluderer ei formell beskriving av både dei manuelle delane av systemet og korleis brukarane skal samhandle med den automatiserte delen av systemet.

Databasekonvertering. Denne aktiviteten krev innspel frå den eksisterande databasen og designspesifikasjonen.

Installering. Den endelige aktiviteten treng innspel frå brukarmanualen, frå den konverterte databasen, og frå det aksepterte systemet. Installeringa kan i noen tilfelle vere ein avslutta aktivitet, i andre tilfelle kan det vere ein langvarig, gradvis prosess der ulike delar av verksemda blir brukarar av systemet etterkvart.

Stikkordsmessig kan vi oppsummere dei viktigaste trekka ved modellen slik: Det er ein instrumentell modell, den er relativt tydelig leiingsorientert og dominert av eit syn på systemutviklaren som ekspert meir enn som medarbeidar og lagspelar. Den er som dei fleste slike modellar innretta mot å presentere handlingar og val i oppskriftsform og legg stor vekt på struktur i systemet og å vere strukturert i organiseringa av arbeidet.

ISAC vart utvikla på 1970-talet ved Stockholms Universitetet med Børje Langefors og Mats Lundeberg som sentrale namn. Modellen legg størst vekt på systemeringsdelen av utviklingsarbeidet og var ein av dei første som etablerte formaliserte systemeringsmetodar. Metodane var nært knytt saman med beskrivingsteknikkar med presise syntaktiske reglar. Tankegangen i ISAC er parallell med livssyklusmodellen når det gjeld kva fasar ein skal gjennom i arbeidet og skil klart mellom analysearbeidet og utformingsarbeidet. ISAC var også ein av dei første modellane som sa at ein vesentlig del av arbeidet må liggje i analysefasen.

Utgangspunktet for systemeringa er den verksemda som skal betjenast av informasjonssystemet. Denne verksemda kan delast inn i delverksemder. I følgje ISAC kan ein ut frå ei presis og detaljert beskriving av verksemda avleie behovet for informasjon. Dersom beskrivinga er presis nok er det mulig å gjøre

det utan "skjønnsmessige vurderingar". Det er ikkje praktisk mulig og derfor er ein avhengig av kunnskapsrike og erfarne brukarar som skal hjelpe til med å presisere informasjonsbehovet med utgangspunkt i dei ulike delverskemdene.

ISAC deler inn beskrivingsteknikkane i tre delar, som beskriv henholdsvis verksemda, informasjonssystemet, og edb-systemet og eventuelle manuelle rutinar. Beskrivinga av verksemda er delt i to, ein strukturbeskrivande del og ein eigenskapsbeskrivande del. I den førstnemnde beskriv ein verksemda og dei meldingar og dei fysiske objekt som straumar mellom delverksemdene og mellom verksemda om omgivelsane. I den eigenskapsbeskrivande delen gir ein kvantitative opplysningar om det som har flytt i ein periode eller det som kan tenkjast å komme.

ISAC er ein funksjonsorientert modell og deler beskrivelsa av informasjonssystemet i tre. Informasjonssystemets informasjonsmengder og relasjonane mellom dei, innhaldet i den enkelte informasjonsmengde og informasjonsprosessen. Desse blir beskrevet ved hjelp av ulike beskrivingsteknikkar som er regel- og symbolorienterte. Modellen er svært detaljert og gir detaljerte anvisningar på korleis arbeidet skal organiserast og utførast. Analysedelen eller forandringsanalysen er delt i to metodeområder, analyse av nosituasjonen og analyse av framtidig verksemd. Systemeringsdelen delast inn i fire metodeområder, verksemdstudie, informasjonsanalyse, datasystemutforming og utstyrstilpasning.

ISAC har ei tosidig begrunning for at brukarmedverknad er viktig. For det første bidrar kompetente brukarar med kunnskap som er viktig om den eksisterande verksemda og er såleis ein ressurs i arbeidet med det framtidige systemet. For det andre fører deltaking i utviklingsarbeidet til større aksept av det utvikla systemet. Modellen gir derimot liten grad av tilrådingar for korleis brukarmedverknad skal organiserast, men antyder på meir generelt nivå at systemering må sjåast på som ein kommunikasjons- og læringsprosess. Det står i klar kontrast til den svært detaljerte bruksanvisninga for resten av systemeringsarbeidet og korleis resultatet av det skal beskrivast.

Ein av kritikkane mot ISAC er at den er svært arbeidskrevjande og såleis betre egna der det er stor grad av stabilitet, det gjeld både i kva delar av verksemda den er egna for og kva typar verksemder den er egna for.

Jackson System Design (JSD) er ein av dei mest utbreidde modellane innan systemutvikling og er utvikla av Michael Jackson og John Cameron. Modellen har eit hendingsorientert perspektiv og integrerer på ein særskilt måte store delar av livssyklusen for systemet. JSD tar sikte på å dekke både systemeringa og realiseringa av informasjonssystemet. Utgangspunktet er at informasjonssystemet skal innehalde ein beskriving av virkeligheita, men JSD er i liten grad opptatt av å analysere om i kva grad denne virkeligheita krev endring, f. eks. i form av organisasjonsutvikling osv. Det ligg utanfor modellen.

ISAC og SASD er rutine- og funksjonsorienterte systemutviklingsmodellar og tar derfor utgangspunkt i ei beskriving av

verksemda og informasjonssystemet saman. JSD ser på informasjonssystemet som eit bilete av verksemda og informasjonssystemet må derfor innehalde ei beskriving av den delen av virkeligheita ein er interessert i. Derfor startar JSD med å beskrive verksemda utan informasjonssystemet og denne beskrivinga dannar så stammen i informasjonssystemet. Det blir så supplert med funksjonar, dvs. dei informasjonar som verksemda (brukarane) ønskjer å trekkje ut og prosessere.

Eit av dei mest sentrale begrepa i JSD er *entitet*. Ein entitet er eit objekt som skaper hendingar eller utsett for hendingar. Entitetar kan vere både ting, enkeltpersonar eller grupper av menneske. Det sentrale er at alle hendingar i livsløpet til ein entitet skal kunne registrerast, og samanhengen mellom hendingane (f. eks. kronologien) må beskrivast. Informasjonssystemet som skal beskrive virkeligheita, må derfor innehalde både ei beskriving av den prosessen som ein entitet skal gjennomløpe, og ei beskriving av kva som har skjedd med kvar enkelt entitet.

Dei viktigaste teknikkane innan JSD er strukturdiagram, strukturtekst, systemspesifikasjonsdiagram og systemimplementeringsdiagram. Strukturdiagram brukast til å beskrive virkeligheita som systemet skal automatisere. I strukturdiagramma blir entitetane beskrevet og også forholdet mellom dei, som kan vere sekvensiell, iterasjon eller seleksjon. I strukturdiagrammet blir ikkje samanhengen mellom beskrivingane tatt med, ein får i staden ei rekke beskrivingar på same nivå. Strukturdiagrammet kan også brukast til å beskrive prosessar i informasjonssystemet og beskrive strukturen i det datamaskinprogrammet som skal lagast. Strukturtekst uttrykkjer det same som strukturdiagrammet, men med ord. Strukturteksten blir ofte brukt på eit seinare tidspunkt i utviklinga og fungerer ofte som eit grunnlag for å skrive sjølve programkoden. Det betyr at JSD alt ganske tidlig i utviklingsarbeidet legg opp til realisering av systemet.

JSD har seks metodesteg; *Entitets-/hendelsessteget*, der systemutviklaren må avgjøre kva som skal vere entitetane i systemet og kva hendingar det er knytt til dei. Valet av entitetar vil samtidig vere avgrensinga av systemet og avgjøre kva det kan vise av informasjon. Det blir ikkje gitt noen reglar for korleis ein kan finne fram til dei aktuelle entitetane, det blir eit spørsmål om erfaring. *Entitetsstrukturs teget*, der ein lager strukturdiagram for entitetane. *Grunnmodellsteget*, der ein konstruerer og viser samanhengen mellom "virkeligheita"og modelleringa i informasjonssystemet, *Funksjonsmodellsteget*, der dei funksjonane som brukarane vil ha i systemet blir beskrevet, dvs. dei prosessane som lager dei data som brukarane er interesserte i. *Synkroniseringssteget*, der spørsmål knytt til eksekveringshastigheit og synkronisering mellom prosessane blir handsama. Realiseringssteget, der systemet blir teknisk realisert. Utviklingsarbeidet er organisert på ein slik måte at desse metodestega er sekvensielt arrangerte, men i praksis vil det vere iterasjon mellom stega.

JSD er i svært liten grad opptatt av korleis arbeidet skal organiserast. Det peikast på at dei fem første metodestega må skje i eit samarbeid mellom brukar og utviklar, men det blir ikkje gitt noen antydningar om korleis dette samarbeid skal skje. Det er i stor grad opp til den enkelte systemutviklar som har ein heilt sentral ekspertrolle i utviklingsarbeidet.

Jackson anerkjenner at kommunikasjonen med brukarane er viktig og da særlig å ta med i betraktning i løpet av analysen deira perspektiv på systemet. Han seier bl. a. at *"the input must be extensive and rich"*, men gir få om noen konkrete anvisningar for korleis dette skal gjennomførast. Det er derfor symptomatisk når han seier at: *"we are not concerned (...) with the specific techniques of interviewing, of gathering evidence, but rather the use the developer should make of the evidence gathered"* (op. cit).

Eg har i dette avsnittet gjort ei kort og summarisk presentasjon og drøfting av korleis tre sentrale modellar innan systemutvikling. Det er openbare skilje mellom dei tre, men samtidig er likheitene klare. I SASD, ISAC og JSD har aktivitetane ulike namn og dei blir delte opp etter relativt forskjellige skiljeliner. Det er likevel rimelig å hevde at sjølv om aktivitetane har ulike namn, blir vektlagt forskjellig og forskjellig sekvensiell inndeling, så er det svært mange likheitstrekk mellom dei når det gjeld innhald. Eit viktig poeng når ein skal studere systemutvikling er at det er stor grad av samanfall innan fagfeltet i den forstand at dei aller fleste beskriv systemutvikling som ein prosess der desse aktivitetane er sentrale og at utviklingsprosessen har ein form for lineær framgang. Mange legg riktignok vekt på at det ikkje er klare skilje mellom aktivitetane, at dei kan komme i forskjellig rekkefølge og at det må vere iterasjon og "vandring" mellom dei undervegs i prosjektet. Det betyr bl. a. at fleire aktivitetar vil foregå samtidig og i ekstremtilfella at alle aktivitetane pågår til same tid. Likevel skin det gjennom at ein lineær tankegang er dominerande innan desse modellane.

I handsaminga av dei tidlige fasane viser det seg at modellane er relativt like i den forstand at dei alle reflekterer eit struktureringsbehov, at alle har islett av det eg vel å kalle binærspøkelset, dvs. at maskinas krav slik det blir oppfatta frå forfattarane legg premissa for utviklingsarbeidet. Ein alternativ modell ville vore å tatt utgangspunkt i kunnskap om det sosiale for slik å la "krav" frå den dimensjonen vere premissgivande.

4. Oppsummering

Systemutvikling er, som eg var inne på innleiingsvis, eit relativt nytt fagområde/vitskaplig felt, der det enno ikkje er etablert ei felles definering av innhaldet. Faget inneheld ei rad konkurrerande definisjonar av kva det skal omfatte, av ulike teoriar, metodar og teknikkar. Det gjør det umulig å snakke om systemutvikling og da tru at alle forstår kva du snakkar om konkret, men

samtidig har det nedfelt seg ei relativt klar oppfatning av at systemutvikling er noe som skil seg klart frå andre yrke. Det er slik lettare å definere feltet etter aktivitetar eller at det er eit felt som kvalitativt skil seg frå andre ting, enn det er å definere det som teoretisk disiplin eller metodisk eining.

Samtidig som systemutvikling har fokus både på det sosiale og det teknologiske, eller på grensesnittet mellom dei, er disiplinen prega av ein teknisk rasjonalitet. Det har minst to konsekvensar. For det første legg det føringar på synet systemutviklarane har på det generelle potensialet i teknologien. Systemutvikling som fag har ei stor tiltru til kapasiteten it har til å løyse menneskelige eller sosiale problem. For det andre tenderer systemutviklarane til å ta berre marginale omsyn til dei menneskelige, sosiale eller politiske elementa av systemet. Det får sjølvsagt konsekvensar for korleis dei nærmar seg og løyser utviklingsprosjekt.

Det er bl. a. slik at det er dei tekniske eller metodemessige spørsmåla i samband med dataflytdiagram osv. som blir diskuterte i litteraturen. Det blir svært sjeldan drøfta grundig og eller eksemplifisert korleis den type representasjonsteknikkar forheld seg til brukarorganisasjonen og den svært heterogene virkeligheit systemet skal virke i.

Det blir i tradisjonell systemutviklingslitteratur vektlagt at svært mange av føresetnadene for å lage eit godt og levedyktig system blir etablerte i analysedelen av utviklingsprosjektet og at det er i denne delen at ein må legge inn størstedelen av ressursane. Det ser likevel ikkje ut til at denne forståinga har fått følgjer for korleis dei presenterer og gjør greie for den praktiske delen av systemutvikling. Der er framleis hovudvekta på det formaliserbare, på boksar og pilar, og det blir ofte ikkje peika på at systemutvikling også handlar om noe anna. Dvs. den sosiale/organisasjonsmessige dimensjonen i utvikling av informasjonssystem blir ikkje gitt same handsaming og det er grunn til å tru at det same forholdet mellom den tekniske og den sosiale dimensjonen blir formidla i undervisning innan feltet. Det kan bety at systemutviklarar som er utdanna ikkje har den nødvendige ballasten til å møte ei heterogen virkeligheit og at andre som vender seg til systemutviklingslitteratur får ei klar forståing av kvar ein bør legge vekta. Nemlig i det teknologiske fordi det er der det er konkrete råd å få.

Det kan vere eit problem for analytikarane at teorien ikkje gir dei eit godt nok apparat som kan sette dei i stand til å møte den utfordringa det er å kartleggje den sosiale verda - som kan hjelpe dei til å forstå brukarverda. Det kan vere ein medverkande årsak til "programvare-krisa", som er prega av at utviklingsprosessar blir meir problematisk, langvarig og dyrare enn nødvendig.

LITTERATURLISTE:

Andersen E. S. (1989): **Systemutvikling**, NKI-forlaget, Oslo.

Friedman A. (1989): **Computer Systems Development: History, Organization and implementation**, John Wiley, Chichester.

Gould D.J. og C. Lewis (1985): **Designing for usability: Key principles and what designers think**, Communications of the ACM, 28, 300-311.

Heininger K. L. (1980): **Specifying Software Requirements for Complex Systems. New Techniques and Their Applications**, IEEE Trans. software engineering., SE-6(1) 2-13.

Jackson M.A. (1975): **Principles of Program Design**, Academic Press, London.

Jackson M.A. (1983): **System Development**, Prentic Hall International, London.

Kirby M.A.R., C.H.J. Fowler, og L.A. Macaulay (1988): "Overcoming Obstacles to the Validation of User Requirements Spesifications", i M. D. Harrison og A. F. Monk, **People and computers IV**, Cambridge University Press, Cambridge.

Langefors B. (1967): **Theroretical Analysis of Information Systems**, Studentlitteratur, Lund.

Langefors B. (1970): **System för företagsstyrning**, Studentlitteratur, Lund.

Markusen A. (1985): "High Tech Jobs: Evidence from California", i Hall og Markusen (red.) **Silicon Landscapes**, Allen og Unwin.

Norsk datahistorie (1989), Ingeniørforlaget, Oslo.

Oftstad E. (1985): **Systemutviklingens tornefulle vei 1958-1983**, Sosiologisk institutt, Universitetet i Bergen (hovudoppgåve), Bergen.

Palaez E. (1988): **A Gift from Pandora's Box: The Software Crisis**, Ph.D. Thesis, Edinburgh universitet, sitatet er henta frå M. Thierny (1991): The Formation and Fragmentation of Computing as an Occupation: a Review of Shifting "Expertise", Working paper no. 25, University of Edinburgh.

Peters L. (1988): **Advanced Structured Analysis and Design**, Prentic-Hall International, Hertfordshire.

Price Waterhouse (1988): **Software Quality Standards: The Costs and Benefits**, Price Waterhouse, London.

Quintas P. (1991): "Engineering Solutions to Software Problems: Some Institutional and Social Factors Shaping Change", **Technology Analysis & Strategic Management**, vol. 3 no. 4.

Rittel H. og M. Webber (1973): **Dilemmas in a General Theory of Planning**, Policy Sciences, 4 (155-169).

Sommerville I. (1992): **Software Engineering**, Addison-Wesley, Wokingham.

Sutcliff A. (1988): "Some experiences of integrating specification of human-computer interaction within a structured development method", i D.M. Jones og R. Winder (red.): **People and computers IV**, Cambridge University Press, Cambridge, England.

M. Tierney (1991): **The Formation and Fragmentation of Computing as an Occupation: A Review of Shifting Expertise**, Working Paper no. 25, Edinburgh PICT, Edinburgh.

Willcocks L. og D. Mason (1987): **Computerising Work - People, Systems Design and Workplace Relations**, Paradigm, London.

Yourdon E. (1988): **Managing the Life Cycle**, Yourdon Press, New Jersey.