P.Ø.

DNMI

DET NORSKE METEOROLOGISKE INSTITUTT

klima

DATABASE-PROSJEKTET I KLIMAAVDELINGEN
STANDARDER FOR SYSTEMUTVIKLING

Delprosjekt 4

B. Nordin, M. Moe, K.A. Iden, P.O. Kjensli

RAPPORT NR. 44/92 KLIMA

DNMI - RAPPORT

DET NORSKE METEOROLOGISKE INSTITUTT
POSTBOKS 43 BLINDERN 0313 OSLO 3

TELEFON: (02) 96 30 00

ISBN

RAPPORT NR.
44/92 KLIMA

DATO
17.11.92

TITTEL

DATABASE-PROSJEKTET I KLIMAAVDELINGEN
STANDARDER FOR SYSTEMUTVIKLING
Delprosjekt 4

UTARBEIDET AV

B.Nordin, M.Moe, K.Iden, P.O.Kjensli

OPPDRAGSGIVER

DNMI

SAMMENDRAG

* Rapporten beskriver standarder for systemutvikling i Klimaavdelingen. Det er satt opp standarder for: programsystemer, bruker-grensesnitt, data-grensesnitt, dokumentasjon og versjonsstyring.
* Standarden gir retningslinjer for hva programmer og dokumentasjon skal inneholde og hvordan disse skal være oppbygd.
  Brukerdialogen er skissert; både i form av skjermbilde-maler, navigeringsregler mellom og i skjermbilder samt behandlingsregler i skjermbildet.
  Datagrensesnittet er ikke detaljert beskrevet, men tar utgangspunkt i datastruktur / formater spesifisert i [2]. Generelt angis Oracle-, binær- og ASCII-filer som lagringsstrukturer. Det stilles også krav til dokumentasjon av disse.
  Standarden for versjonsstyring forutsetter at SCCS (kontrollsystem for kildekode) benyttes, og anbefaler bruk av biblioteksrutiner.
* Ved å benytte standarden under systemutvikling vil vi oppnå et mere enhetlig og oversiktlig system, hvilket er viktig både for brukerne av systemet og ved vedlikehold / påbygging av systemet.

UNDERSKRIFT

Bjørn Nordin
PROSJEKTLEDER

Bjørn Aune
FAGSJEF

Innholdsfortegnelse

1 Innledning.

Ved DNMI er det ikke noen formell standard for utvikling av edb-systemer. Dette dokumentet angir en standard for utvikling av edb-systemer som først og fremst er beregnet for Klimaavdelingen ved DNMI. Denne standarden bør senere kunne tilpasses instituttets standard på dette området.

Å følge en standard ved systemutvikling er viktig for å få et "helhetlig" system som det er mulig å vedlikeholde / videreutvikle uten for store ressurser.

2 Målsetning for standarden.

Målsetningen med standarden er å få et system med:
- modulvis oppbygning,
- klare / enhetlige grensesnitt,
- tilstrekkelig dokumentert

De ovenfornevnte målene må tilfredsstilles av standarden for:
- programmer,
- bruker-grensesnitt,
- data-grensesnitt,
- dokumentasjon
- versjonsstyring

Ingen produkter bør settes i drift om det medfører klare ulemper for brukerne. Dvs. om et program / system kan gjøres merkbart bedre med en rimelig ressursbruk.

3 Standard for programvare.

Denne standarden skal medvirke til at vi får datasystemer som:

- er modulvis oppbygget,
- har klare grensesnitt,
- er tilstrekkelig dokumentert.

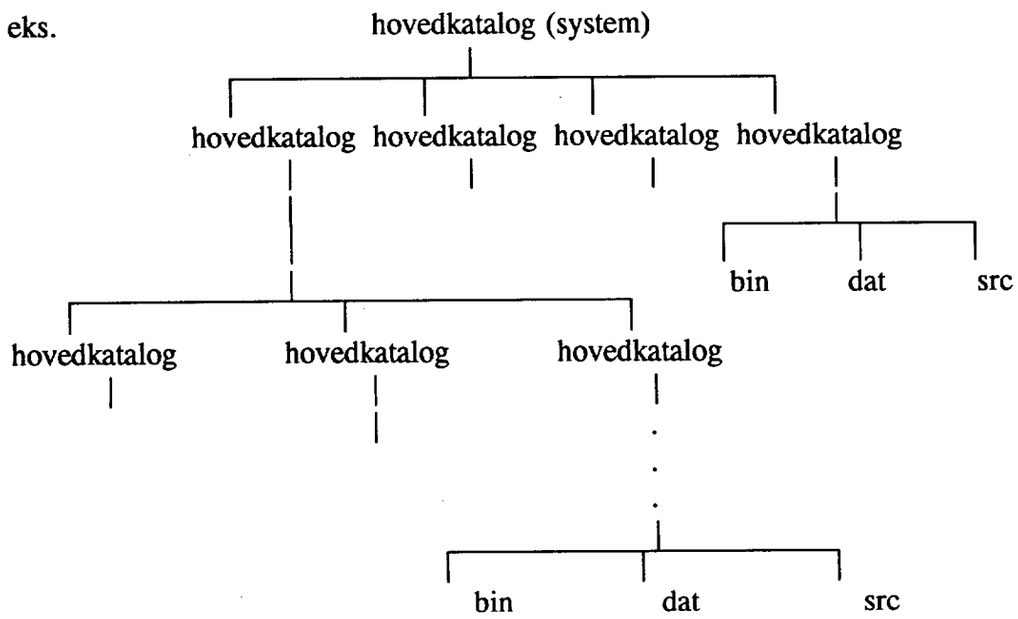
Dette kapitlet benytter enkelte delvis innarbeidede begreper for oppdelingen av en stor applikasjon:

- system	avgrenset informasjons/EDB-system (med alle moduler)
- modul	del av systemet (består av en eller flere prosesser/rutiner)
- prosess	hoved-/styrings- script/batchjob
- subprosess	under-/sub- script/batchjob (startet opp fra en prosess)
- rutine	hovedprogram f.eks. kompilerte FORTRAN- og C-programmer (.exe-filer)
- smårutiner	biblioteksrutiner (subrutiner/funksjoner i FORTRAN- og C-programmer)

3.1 Katalogstruktur for system.

Hvert enkelt system skal fordeles over en hierarkisk oppbygd katalogstruktur. Denne tar utgangspunkt i *en* hovedkatalog med forgreninger til en / flere kataloger (betegnes også som hovedkataloger), som igjen kan ha underkataloger (se eksempel nedenfor). Antall katalog-nivåer må tilpasses hvert enkelt systems kompleksitet. Laveste katalog-nivå skal bestå av katalogene bin, dat og src. Bin inneholder eksekverbare binærfiler og temporære filer, dat inneholder datafiler, mens src inneholder kildekode. Prosesser og subprosesser (script / batchjobber) skal ligge i hovedkatalogene.

eks.



3.2 Navne-konvensjon for system.

Her inngår navnekonvensjoner for hvert enkelt system og de elementer, prosesser, filer etc. som er knyttet til det.
Hvert system må ha en unik og permanent id (/forkortelse), og innenfor systemet må konvensjonen bygges opp hierarkisk. I tillegg til bruk av kataloger, vil navnene da vise logisk gruppering ved at navnets første del viser tilhørighet.

-Systemnavn er id på 3 (4) karakterer	SSS
-Modulnavnet får et tillegg tilsvarende	SSS MMM
-Prosessnavn	ppp
-Subprosessnavnet tar utgangspunkt i prosessnavnet og har et tillegg tilsvarende	ppp sss
-Rutinenavn	rrr
-Smårutiner	rrr <name>
-Datafiler	<name>

Etter tilhørighets-id kan det henges på et mere beskrivende filnavn slik det skal gjøres for alle rutiner. Id og navn må være unike innenfor systemet, system og prosessnavnene også utenfor systemet.

Fil-type må også være inkludert i navnekonvensjonen. Det er praktisk å benytte de etablerte standarder for operativsystem etc i størst mulig grad.

f.eks.:

-kommandofiler	bat, com, sh eller "uten filtype"
-datafiler	dat, dta
-registerfiler med styringsparametere	reg
-biblioteksfiler	lib, h
-inputfiler for hjelpebiblioteker	hlp
-info og oversiktsfiler	inf, txt
-kildefiler med FORTRAN-program	for (eller) f
-kildefiler med C-program	c
-eksekverbare filer	exe (eller uten filtype)
-linke/loade-filer	mak, makefile, Makefile
-temporære filer	tmp

3.3 Systemoversikt.

Det må lages en oversikt over systemet med informasjon om organisering, funksjon og oversikt over alle moduler, prosesser og filer som inngår i systemet.

Denne oversikten og informasjonen bør, så langt det er mulig, vedlikeholdes automatisk. Operativsystemets verktøy for programorganisering bør brukes og kan danne grunnlaget for systemoversikten. Disse verktøyene er (under UNIX): 'make','ar','sccs', og er beskrevet i Versjonsstyring 7.

3.4 Organisering av programmene i et system.

Oppgaven systemet skal utføre, må deles opp i enkelte enheter slik at feks styrende parametre etc skilles ut fra den mere generelle algoritmen, I/O samles, logisk feilkontroll utføres ett sted og at oversiktlige registre ligger enkelt tilgjengelig for korrigering av f.eks. formater, beregningskonstanter eller testverdier. Disse enhetene er:

- eget styreprogram
- egne moduler/..../rutiner for I/O som henter filnavn, devicenavn etc fra separerte registere. Aliaser kan/bør benyttes.
- egne moduler/..../rutiner for terminal-kommunikasjon eller innhenting av styrende/overordnede parametre
- egne moduler/..../rutiner for initiering
- egne registre adskilt fra algoritmen
- utførende generelle algoritmer
- feilkontroll
- styring av prosesslogg og feillogg

Hver rutine bør begrenses i størrelse og kompleksitet (bør ikke overstige 300 utførbare programsetninger).

Målet er å få til en logisk hierarkisk organisering hvor subrutinen inngår som minste selvstendige enhet og at denne minste enheten er enkel å få oversikt i og å rette på.

Enheter som brukes av mer enn en modul/..../rutine, skal ligge i flerbruker-biblioteker.

Ett sett biblioteker av hver kodetype:

Kildekode	.for .f og .c	
Linkbar kode	.obj og .lib	(Archive library (ref. kap.7))
Eksekverbar kode		(Shared library (ref. kap.7))

Et system for linking av enkeltrutiner fra biblioteket (bare de programmet trenger), skal brukes, om dette er mulig.

3.5 Kommentarer.

3.5.1 Krav til program og rutine-heading.

Hvert system skal ha felles mal for program og rutine-heading:

```
Rutinenavn: .....

Rutinens oppgave: .....

Metode:.....
Evt. sideeffekter:.....

Rutiner som kalles: .....
                    .....

Databaseaksess: .....
                .....

Fil-aksess: .....
            .....

INPUT: .....
OUTPUT: .....
```

Da rutinene varierer i kompleksitet vil ikke alle punkter alltid være aktuelle.

Forklaring til enkelte av punktene:

* Sideeffekter beskrives hvis rutinen påvirker andre deler av systemet.
* Databaseaksess angir tabeller som rutinen aksesserer.
* Fil-aksess angir filer som aksesseres.

Rutine-headingen skal også inneholde informasjon om **hvem** som har utviklet og oppdatert rutinen, samt **hva** som er oppdateringens innhold, hvis dette ikke dokumenteres på annen måte (f.eks. vha. UNIX sitt SCCS system).

3.5.2 Generelle kommentarer.

Alle rutiner (programmer) og prosesser (script / batchjobber) skal være gjennomgående kommentert. Spesielt viktig er det at rutinekall, løkker, variable og viktige "utførelser" kommenteres. Kommentarene skal beskrive innhold / logikk, og **ikke** "oversette" statements.

3.6 Standard for feilbehandling.

All feilbehandling skal samles. Hver prosess skal ha kun en feilutgang. Feil fra flere prosesser skal tolkes samlet, og gi en feilmelding (ett samlet sett meldinger) i feillogg. Feilen(e) skal klassifiseres, og de enkelte feiltypene telles opp, og avhengig av feiltype og mengde gi operatør melding direkte.

I tolkningen skal det også inngå hvilken del av prosessen som kan/skal fortsette og hvilke meldinger som skal gå ut til andre systemer.

Hvert enkelt system må inneholde:
- en beskrivelse av gangen i feilrapporteringen, samt en beskrivelse av feilmodulens logikk.
- en liste over de koder som inngår (med feks aliaser for tall) m.h.p. feiltyper (fatal, warning, ...), feilkilder (operativsystem, database, nett, ...), prosess og rutinekoder, kontinuitetskriterier mm. Med kontinuitetskriterium menes en oversikt over hvilke (del)prosesser / systemer som kan fortsette etter gitte feil (og mengde feil).
- en beskrivelse av overføringsmekanismer for feilmeldinger innen prosessen og mellom forskjellige prosesser (også dersom den ene dør).
- beskrivelse av feilmeldings-formatet som skal benyttes i hver enkelt rutine. La helst formatet avsluttes med en tekststreng på maksimalt 80 karakterer med beskrivende tekst fra den feilende rutinen.
- beskrivelse av formatet av 1) direkte melding til operatør, 2) melding til logg.
- beskrivelse av eventuelle timeout-mekanismer. Dette er viktig dersom systemet involverer flere prosesser som går delvis uavhengig av hverandre, og/eller det benyttes perifere device eller prossessorer med mulighet for heng mot/ved disse.
- initiering av deler av systemet, eventuelt startet opp av feil-tolkeren.

Om man på tross av dette mister viktige feilmeldinger, skal alle feilmeldinger logges kontinuerlig.

3.7 Standard for intern oppbygging av rutiner.

Rutinen skal så langt mulig inneholde en samlet logisk operasjon. Den skal deles opp i program-blokker og hver blokk skal kommenteres samlet i starten av blokken.

Ved bruk av gren-konstruksjoner (do,for,if...) økes lesbarheten ved å skille denne ut ved et fast innrykk på 3 plasser for grenens utførende og event kommenterende statements. Grenens avslutnings-statement og delavslutning (else...) rykkes tilsvarende ut (se eks). Faste løkke-konstruksjoner(do,for,...) skal benyttes og enkelthopp unngås. Enkelthopp inn i en løkke og hopp opp tilbake tillates ikke.

Dimensjonering av arrayer, grenseverdier for løkker og tester skal aldri settes i det aktuelle statementet, men hentes fra registre eller satt i en initierings-rutine (FORTRAN feks vha. parameter-statementet).

Hver blokk har kun ett entry-punkt, mens det tillates flere returpunkter (i motsetning til prosess-nivå,hvor kun to tillates). Det anbefales å benytte logiske blokker (i FORTRAN og C), hvor rutinen kalles vha en test om den logiske rutinen har utført jobben.

Eksempel:

```
hvis subrINPUT(..) da utfør
  hvis subrDECODE(..) da utfør
    ......
    ......
  ellers rapporter feil ved dekoding
ellers rapporter feil i input
```

3.8 Navne-konvensjon for variable.

Ethvert system kan ha en standard for rutinenavn, variabelnavn, tellere, pekere, indekser, styrende parametre, etc. Dersom et stort antall av systemene hadde en slik felles innarbeidet standard, ville lesing av andres programmer kunne forenkles.

Ved å benytte prefix til variabelnavn kan dette oppnås på en enkel måte.

- rutinenavn er behandlet over og prefixet er ikke krevd for små rutiner. Likevel kan et tilhørighets-prefix være nyttig.
- variabelnavn, generell integer i,j,k,l,m,n
  - " - logical b
  - " - character c
  - " - pekere p
  - " - real de øvrige

3.9 Språkavhengige standarder.

De språk som i denne omgang er aktuelt å sette standarder for er FORTRAN, C og C++.

3.9.1 Programmeringsspråket FORTRAN.

- Strukturert FORTRAN pålegges, med bruk av gren-statements og flere lag subrutiner.
- Unlabled common forbys, og common begrenses.
- Variable deklareres selv om prefixet er implisit deklarert.
- Goto statementet begrenses til der det ikke finnes mulighet til annet, feks ved hopp ut av en løkke til umiddelbart etter løkkas avslutning. Det er ikke tillatt å hoppe ut av en definert blokk.

3.9.2 Programmeringsspråket C.

- Minst mulig programmering, mest mulig benyttelse av UNIX-scripts og programbiblioteker.
- Programmer fordeles på flere filer for å hindre tilfeldig aksess av variable og sikre modularitet. Hver fil inneholder en håndfull prosedyrer.
- Programfiler bygges opp av prosedyrer. Hver prosedyre av størrelse ca. 1 skjermbilde.
- Idiomatiske uttrykk og kompakt notasjon brukes i størst mulig grad der dette vil øke lesbarheten for språkkyndige.

- Globale variable forsøkes unngått.

- Dynamisk plassallokering foretrekkes fremfor statisk der dette er effektivt gjennomførbart.

3.9.3 Programmeringsspråket C++.

Språket C++ benyttes for objektorienterte problemer. Hovedprinsippene for FORTRAN og C gjelder også her, men problemer som krever objektorienterte løsninger setter vanligvis ekstra strenge krav til ryddig programkode.

- Bunting av assosierbare objekter og morfismer (prosedyrer).

- Benyttelse av subobjekter i så stor grad som mulig.

- Utnyttelse av Private/Public.

- Spredning av programkode på flere filer, minst en fil for hvert objekt i større applikasjoner.

4 Standard for brukergrensesnitt.

Brukergrensesnitt er i videste forstand et meget omfattende begrep da dette jo kan omfatte alt som brukeren ser mot EDB-systemene (programmer, skjermbilder, dokumentasjon osv.).

Brukergrensesnitt her vil kun omfatte den befatning brukeren får med:

Skjerm
Tastatur

i egenutviklede (DNMI-Klimaavdelingens) applikasjoner.

All spesifikasjon / uttesting av brukergrensesnitt skal, så langt det er praktisk mulig, foregå sammen med brukerne.

4.1 Språk.

* Norsk skal benyttes i alle skjermbilder og meldinger til interne brukere.

* I rapporter beregnet for eksterne brukere skal det kunne velges mellom norsk og engelsk.

  Språk i skjermbilder / rapporter beregnet for eksterne brukere bør forøvrig tilpasses disse i størst mulig grad.

4.2 Brukerdialog.

Brukerdialogen er den kontakten brukeren har direkte med systemet. Denne skal være oversiktlig og selvdokumenterende (dvs. at brukeren skal "ledes" gjennom systemet og evt. få hjelp underveis).

Alle systemer *skal* være menystyrt. Dvs. at man navigerer mellom ulike funksjoner ved valg fra menyer. Systemet skal i tillegg tillate direkte "hopp" mellom funksjoner via kommandoer der dette er hensiktsmessig (ut i fra bruken av systemet). Ingen informasjon skal imidlertid gå tapt, og man skal når som helst kunne gå over til å navigere vha. menyene igjen.

4.2.1 Skjermbilder.

Det som hører naturlig sammen skal være i ett og samme bilde dersom dette er mulig. Blir bildet for overlesset, bør det deles opp i flere sider, evt. bør det vurderes å ta i bruk menyer (flere nivåer). Alle menybilder skal ha "layout" som vist i fig. 1, inklusive de angitte felt (elementer). Andre skjermbilder skal ha første linje og de to siste linjene likt med meny-bildet (fig. 1), mens "innholdet" må variere fra bilde til bilde uti i fra funksjon (fig 2).

* Ved innlegging / oppdatering av data, skal innleggingen / oppdateringen alltid gjøres endelig (commit) pr. skjermbilde (eller oftere). Dvs. en transaksjon skal aldri foregå over mere enn ett skjermbilde.

* All sletting av data må verifiseres i skjermbildet.

* Så sant mulig ut i fra utviklingsverktøy, skal det alltid angis i skjermbildet at "arbeid pågår" når "programmet går"(evt. hvor mange % som er utført).

* Bruker skal alltid kunne velge ut-medium for resultatet av en applikasjon (fil, skjerm, printer osv.).

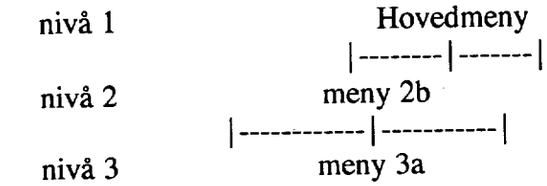
* Informasjon om utdata (f.eks. enhet, statusflagg osv.) skal kunne presenteres ved behov.

4.2.2 Oppsett av menybilder / skjermbilder.

Definisjoner:

Nivå: Plass i hierarkiet.

Eksempel:



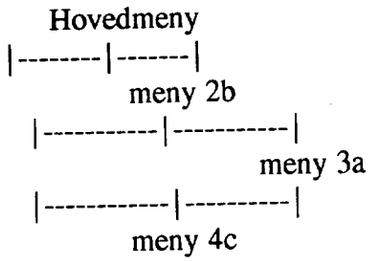
Sti: Den veien en går / har gått i et programsystem.

Valgene i alle menyene må være tilstrekkelig beskrevet. Hvis ikke må det være en hjelptast med nok beskrivelse / veiledning.

En meny (et nivå) kan ha direkte valg av programmer / funksjoner eller flere menyer (hierarkisk oppbygning).

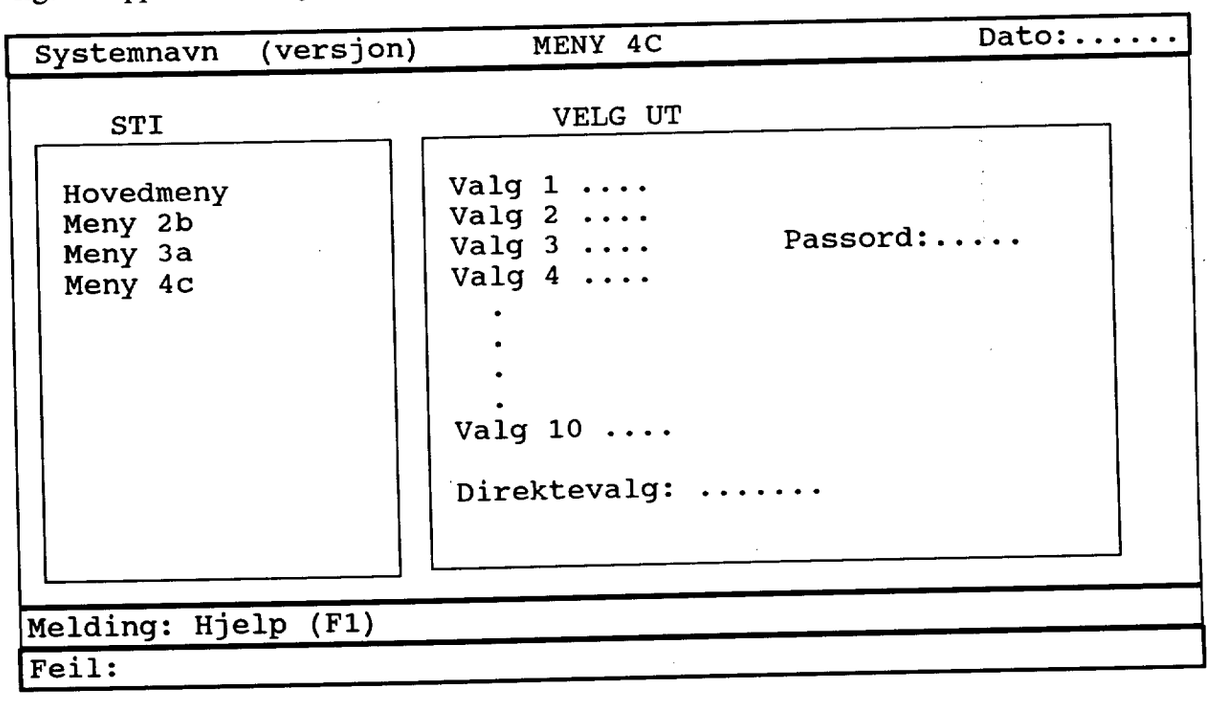
I hver enkelt meny skal det være angitt hvor man er i systemet.

F.eks. skal følgende struktur:



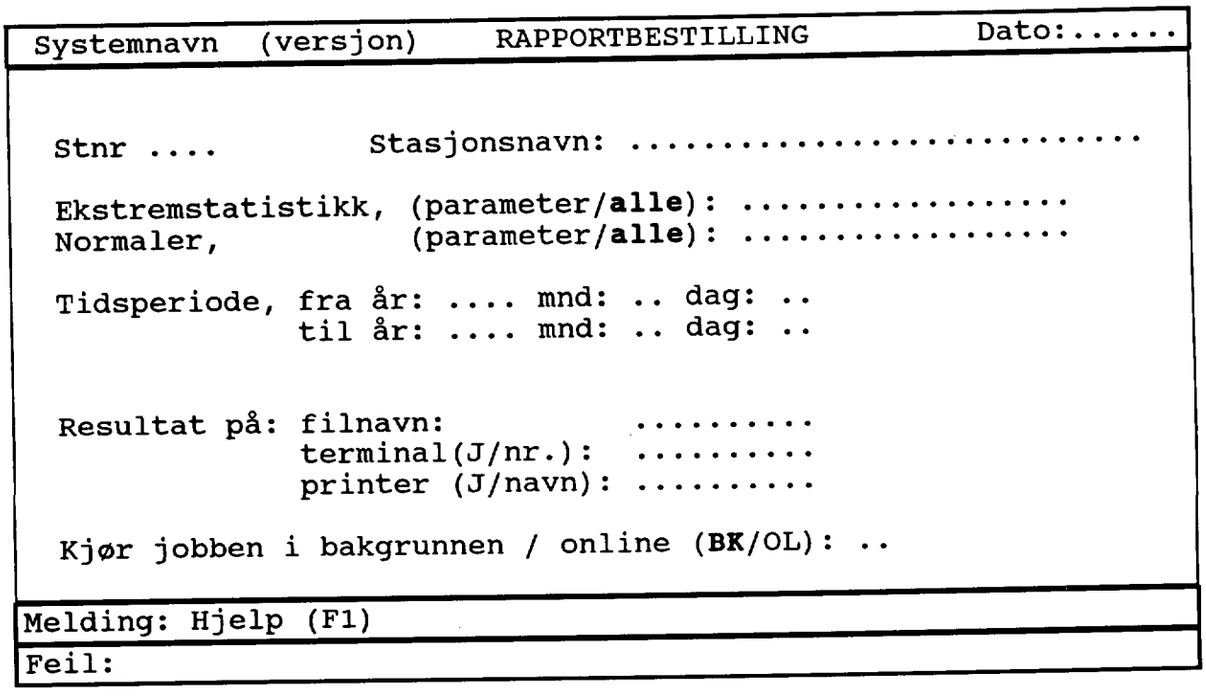
"gjenspeiles" i skjermbildet (meny 4C) ved at stien angis:

fig. 1 Oppsett av menybilde.



For "windows" applikasjoner og bruk av rullgardinmenyer vil ovenforstående "layout" kanskje ikke være hensiktsmessig. Dette dokumentet vil imidlertid ikke angi noen standard for bruk av windows / rullgardinmenyer.

fig. 2 Andre skjermbilder



4.2.3 Navigering mellom menyer / skjermbilder.

Man navigerer mellom menyer og skjermbilder ved å velge fra menyen. Unntatt på nivå 1 skal det alltid være et menyvalg til "forrige" nivå. Eventuelt kan man flytte seg direkte mellom menyer og skjermbilder ved direkte valg fra en meny eller skjermbilde.

Behovet for slike snarveier kan variere fra system til system og fra bruker til bruker og hvor ofte man bruker systemet. Det anbefales at direkte-valg implementeres slik at bruker selv kan velge om han / hun vil bruke dette eller navigere vha. menyene. Brukeren skal imidlertid alltid (fra et hvilket som helst nivå) kunne navigere videre vha. menyer.

Stier skal tas vare på av og i systemet så lenge det ikke er avsluttet. Dvs. at brukeren skal kunne gå tilbake den veien hun / han allerede har gått gjennom menyene / skjermbildene.

4.2.4 Navigering i menyer / skjermbilder.

Cursor skal i utgangspunktet flyttes sekvensielt mellom alle inputfeltene, eller slik det er hensiktsmessig for applikasjonen. Hvis det oppstår feil, bør cursor bli stående (evt. gå tilbake til første mulige feil utfylte felt) og feltet "lyse opp" (revers video / farge el.) og feilmelding / rettledning gis.

4.2.5 Felt i menyen / skjermbildet.

* Dato skal alltid inngis og vises på formen: dd.mm.åååå

* Tid skal alltid inngis og vises på formen: tt:mm

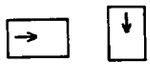
* I søk etter stasjon skal alltid enten stasjonsnummer eller navn / del av navn kunne gis inn. Det skal dessuten alltid være mulig å plukke ut stasjon(er) fra en stasjonsliste som vises ved behov (ref. neste punkt).

* Inndata til felt i skjermbildet skal, så sant informasjonen er lagret i systemet, kunne velges fra lister som viser aktuelle valg / muligheter, presentert ved behov.

4.3 Tastatur.

* TAB flytter cursor sekvensielt mellom inputfeltene i skjermbildet.
* ESC går ett nivå ut (til forrige meny / skjermbilde).
* HOME går til starten (første input-felt) av menyen /skjermbildet.
* END går til slutten av skjermbildet / menyen.
* PGUP blar en tekstside tilbake.
* PGDN blar en tekstside frem.
* INSERT slår av og på insert modus.
* DELETE sletter tall / tekst / tegn.
* PILTASTENE navigerer i og mellom inputfelt i menyen /skjermbildet.

går til forrige inputfelt



går til neste inputfelt

går til forrige kolonne (tegn) i inputfeltet

går til neste kolonne (tegn) i inputfeltet

går til starten på inputfeltet

går til slutten på inputfeltet

Der det benyttes kommersielle produkter for å lage menyer /skjermbilder, kan ovenfornevnte om nødvendig fravikes til fordel for produktets tastaturdefinisjoner.

5 Datagrensesnitt.

Oracle.

Data som er beskrevet i "Etablering av valgt datastruktur på Typhoon" [1] og "Utarbeiding og testing av ulike datastrukturer på Typhoon" [2] skal lagres slik det er beskrevet i disse rapportene.
Dokumentasjon skal finnes.

Binærfiler.

Direkte aksess skal brukes.
Dokumentasjon skal finnes.

ASCII-filer.

Dokumentasjon skal finnes.

6 Dokumentasjon.

Kravet til dokumentasjon kan variere ut i fra oppgavens art og kompleksitet. Denne standarden vil angi hvilke dokumenter som imidlertid skal inngå, og skissere innhold (i stikkords form) av disse. Standarden angitt her, danner grunnlaget for kvalitetssikringen angitt i "Kvalitetssystem for prosjektarbeid" [3].

Enhver større oppgave (delprosjekt) skal bestå av følgende faser og tilhørende produserte dokumenter:

*	Initiering	Kvalitetssikringshåndbok opprettes.
*	Forstudie	Prosjektplan / Forstudie
*	Spesifikasjon	Kravspesifikasjon
*	Konstruksjon / Realisering	Systemdokumentasjon, Brukerdokumentasjon, Kvalitetssikringsskjema for programvare.
*	Verifikasjon	Testplan / testskjema Avviksrapport.
*		Endringsdokumenter / Revisjoner
*		Prosjektrapport

6.1 Kvalitetssikringshåndboken.

Kvalitetssikringshåndboken skal: dokumentere hele prosjektet og er en del av prosjektets kvalitetsplan (ref. [3] kap. 4.2 og 5).

6.2 Prosjektplanen.

Prosjektplanen skal: kartlegge dagens situasjon, behov og analysere mulige løsninger. Videre skal den spesifisere mål og ambisjonsnivå, metode, aktiviteter, tidsestimat, ressursestimat, Gantt-diagram. Prosjektplanen skal ha en revisjonsforside og evt. en revisjonsfotnote (ref. [3] kap. 4.7)

6.3 Kravspesifikasjonen.

Kravspesifikasjonen skal:	Beskrive behov / krav /ønsker. Beskrive løsninger og funksjonalitet i et "brukertilpasset" språk, samt brukergresesnitt.

6.4 Systemdokumentasjon.

Systemdokumentasjonen skal:	Dokumentere faktisk implementering av funksjoner, beskrevet i kravspesifikasjonen. Systemdokumentasjonen skal videre være så detaljert at den fullstendig dokumenterer systemet, beregnet for andre systemutviklere. Programflyt / logikk med flytdiagram skal angis sammen med skjermbilder og program / subrutine oversikt.

6.5 Brukerdokumentasjon.

Brukerdokumentasjonen skal:	Være skrevet for brukerne av systemet ! Den skal være så detaljert at det ikke kreves forkunnskaper eller annen veiledning i tillegg. Den skal angi "oppskrift", nødprosedyre, navigering, input, output og en oversikt over systemet.

6.6 Kvalitetssikringsskjema for programvare.

Kvalitetssikringsskjema for programvare kreves utfylt og signert av kvalitetssikrings-ansvarlig før programvaren tas i ordinær bruk (ref. [3] kap. 5.3.3 og 5.6).

6.7 Testplan / testskjema.

Testplan / testskjema skal:	angi hva som skal testes og forventet resultat. Avviksrapportering, sjekkliste for oppretting, retesting og godkjenning.

6.8 Endringsdokumenter / revisjoner.

Endringer både under utvikling og etter at systemet er satt i drift skal dokumenteres i form av endringsdokumenter / lister eller ved revisjon av eksisterende dokumentasjon. Versjonsnummerering er derfor viktig (se kap. 7).

6.9 Prosjektrapport.

Prosjektrapporten skal inneholde en oppsummering av prosjektet, "måling" av suksess for prosjektet, status mv. Rapporten skal ha en revisjonsforside (ref. [3] kap 5.7).

7 Versjonsstyring.

Med versjonsstyring menes et verktøy eller metode som sikrer at utviklingen av programvare går i ønsket retning og er oversiktlig. Versjonsstyringen er et ledd i kvalitetssikringen.

Versjonsstyring skal brukes på programvare og dokumentasjon. Med dokumentasjon menes her alle dokumenter som beskriver programvaren. Det kan være alt fra systemoversikter til brukerveiledning.

For versjonsstyring av programvaren skal UNIX sitt verktøy brukes. Det skal også brukes på brukerveiledningen som følger programvaren. For annen dokumentasjon kan UNIX-verktøyet også brukes om ikke annet finnes. Det kreves at filene er ASCII-filer.

Versjonsstyringen skal medføre:

nummerering:	utgave.nivå.gren.rekke (tilstrekkelig med utgave.nivå)
datering:	dato, tid
endring:	formål, konsekvenser
konsistens:	I eget og mot andre system/dokument(er)

For at disse punktene skal være oppfyllt er det nødvendig å bruke et sett av verktøy.

Konsistens mot andre systemer (programmer) krever at moduler er samlet i biblioteker. Biblioteker kan bygges på to måter (v.h.a. ar-systemet):

Archive Library: samler objekt-kode-moduler som kan kopieres inn i programmer. Archive filer genereres med **ar** -kommando. Etter vedlikehold av moduler på biblioteket må alle eksekverbare programmer som bruker biblioteket genereres på nytt. Denne metoden er usikker m.h.p konsistens.

Shared Library: samler eksekverbare moduler som lastes inn i programmer ved eksekvering. Vedlikehold på biblioteket gir konsistens mot andre systemer automatisk

Konsistens mot eget system krever en metode for å vedlikeholde siste versjon av systemet (som består av flere filer som har variert opphav). Verktøyet for dette formålet er:

make: en generell rutine for å kompilere kildekode, linke objektmoduler eller generere eksekverbare moduler. **make** leser en Makefilesom inneholder spesifikasjonen av sammenhengen mellom filer og hvordan disse skal behandles.

De øvrige punktene i versjonsstyringen var som nevnt: Nummerering, datering og opplysninger om endring. Dette tar SCCS seg av:

SCCS: et verktøy til vedlikehold og sporing av utvikling på UNIX systemer. Det gir kontroll med lagring og oppdatering av filer. Det kan brukes både på programfiler og rene dokumentasjonsfiler (ASCII). Sikrer at kun en kopi (av samtlige versjoner) kan redigeres om gangen. SCCS gir følgende:

- Den lagrer en original + evt. endringer.
- Nummerering og forgreninger. (utgave.nivå.gren.rekke)
- Dato,tid og identifikasjon av utøver.
- Kommentarer til endring.

Referanser

[1] Etablering av valgt datastruktur på Typhoon, Delprosjekt 3,
DNMI-Klima rapport nr. 40/92

[2] Utarbeiding og testing av ulike datastrukturer på Typhoon, Delprosjekt 2
DNMI-Klima rapport nr. 42/92

[3] Kvalitetsstyring for prosjektarbeid, Delprosjekt 5
DNMI-Klima rapport nr. 45/92