

HARLANS SEMINAR 1987

90-ÅRENES KKI-SYSTEMER  
PROGRAMVARE VEDLIKEHOLD

OVERINGENIØR VEBJØRN SMÅBERG  
P 6157 OG P6071

## 90-ÅRENES KKI-SYSTEMER OG VEDLIKEHOLD AV PROGRAMVARE

FOREDRAG PÅ HARLANS SEMINAR FREDAG 6 NOVEMBER 1987

### 1. INNLEDNING

Årets tema er simulering/modellering, og det følgende foredraget har som overskrift 90-årenes KKI-systemer og vedlikehold av programvare. Begge disse områdene er på grunn av dagens teknologi-utvikling og de muligheter denne gir, paradoksalt nok både kompliserte og meget ressurskrevende. Det er derfor nødvendig å konsentrere seg om deler, og jeg har da valgt å ta for meg det som er relatert til Sjøforsvaret og spesielt til de oppgaver SFK har som ansvarsområde. Spesielt vil det bli fremhevet de problemer vi etter min mening har i dag, hva vi er i ferd med å gjøre, samt noen tanker om fremtiden.

SFK har som organisasjon gjort et kjempeløft for å tilpasse seg dagens situasjon rent organisasjonsmessig gjennom ODIN-prosessen. Spørsmålet er om man har nådd de mål man har satt seg. Jeg håper dette foredraget kan gi en indikasjon på hvilke områder man bør sette inn mer innsats.

Jeg er klar over at dere som tilhørere har en svært sammensatt bakgrunn, men jeg håper at de ord og vendinger som blir brukt vil bli forstått, selv om det nok vil bli benyttet en del gode "ny-norske" ord. Jeg vil også gjøre oppmerksom på at det ikke er nødvendig med noen EDB-bakgrunn; det det gjelder er kun sunn fornuft.

### 2. SYSTEMET

En overordnet målsetning for SFK er produksjon av Sjøforsvarets materielle stridsevne. Denne materielle stridsevnen bygger på de krav som utarbeides av de operative myndigheter - de operasjonelle stabskrav. For en marine vil den materielle stridsevne være en kampenhet, d v s et fartøy eller fort med de nødvendige omgivelser. Denne kampenheten er i dag sammensatt av mer eller mindre autonome deler som skal sammen virke effektivt over levesyklusen for å løse sine pålagte oppgaver tilpasset en stadig forandring av den operasjonelle trussel, samt være innenfor de tekniske, økonomiske, personellmessige og tidsmessige begrensninger som til enhver tid gjelder.

Tradisjonelt har man sett på kampenheten som oppdelt i ulike deler, f eks skrog, maskineri, sensorer, våpen, etc. Man anskaffet ofte gode løsninger innenfor disse delområdene, men stilte man seg noen gang spørsmålet om disse suboptimaliseringene var den optimale løsningen for plattformen som helhet? Stilte man spørsmålet om man rent logistisk kunne drive denne plattformen gjennom hele dens levetid? Ville plattformen kunne oppdateres for å ivareta forandringer i trussel, nye våpen, sensorer, o s v? Var den tilpasset de operatører som skulle betjene plattformen? Mange tilsvarende

spørsmål kan stilles, og satt på spissen så må det generelle svaret være nei. Dette ikke som noen kritikk av fortiden, men saken er at man med de teknologiske utviklinger nå har mulighet til å rette på disse forhold, men det krever at man er villig til å ta den merbelastning som er nødvendig i plattformens utviklingsfaser. Den eneste måten å få dette til på er å betrakte de enkeltheter en plattform består av som et enhetlig system. Man må slutte med sub-optimaliseringer og innse at det er systemet som et helhetlig system som skal oppfylle de pålagte og forventede oppgaver.

Hva består et systems livssyklus av? Hvor bør og kan brukerne selv sette inn sine ressurser? Den store "utgiftsposten" ligger i programvaren, som i dagens systemer er den mest fremtredende del. Man hører ofte at produktene blir levert for sent, fyller ikke kundens krav og ikke minst forventninger og budsjettene overskrides. Utallige undersøkelser viser at dette i hovedsak skyldes utilstrekkeligheter i kravspesifikasjonene (ca 50-60%) mens selve kodingen er skyld i bare 30%. Det er også et faktum og nokså enkelt å forstå at jo senere feilene finnes, jo større blir kostnadene ved utbedring. (Vi snakker i enkelte tilfeller om opptil 80 ganger fordyrelse). Dette burde derfor klart indikere at vi som kunder må forsterke innsatsen i front-ende analysene.

La oss nå se på denne første delen av et systems livssyklus. Et problem må:

- dekomponeres inn i funksjoner
- funksjonene må allokere til komponenter
- komponentene må designes og testes for å møte de opprinnelige kravene

For å kunne utføre dette er det et behov for formelle metoder og verktøy. Disse finnes, men jeg vil ikke gå gjennom disse, men konsentrere meg om konseptene. Det er av avgjørende betydning at man gjennom fasene i systemutviklingen sikrer at man beholder den overordnede oppførsels- og ytelsesmessige karakteristikkene for et system. Dette vil resultere i et design som utfører de oppgaver som opprinnelig var planlagt med de nødvendige "trade-offs" for å balansere systemeffektivitet mot kostnader. Det er kanskje nå på sin plass å definere system-effektivitet. Denne er et produkt av:

- overlevelses- evne
- ytelse
- pålitelighet
- tilgjengelighet
- treningsbehov
- vedlikeholdbarhet

målt opp mot de operative krav og tilpasset levesyklusaspektene.

For å fange opp systemets oppførsel er det nødvendig å:

- trekke opp grensene
- identifisere objekter og objektenes tilstander
- fra objektenes tilstander utforme følger av disse
- dekomponere objektenes tilstand for å identifisere karakteristikk
- assosiere objektenes karakteristikk med input informasjon i systemet
- definere sammenhengen mellom hendelser og annen informasjon samtidig som man beholder oversikten over tidssekvenser, data flyt, ytelse, o s v.

Etter som man går gjennom de ulike fasene, vil man alltid gå gjennom den samme syklus som vist på figuren med forskjellig innhold og bindinger.

Når dette er gjort, starter man selve oppdeling av oppgavene mellom maskin og operatør. Dette skaper behov for en tilsvarende analyse av grensesnittet mellom disse samt mellom flere operatører.

Det vi nå egentlig har beskrevet er et kommando kontroll og informasjonssystem, men vi har sørget for at man har sett dette i forhold til alle de komponenter som en kampanhet består av. I tillegg er det tatt hensyn til omgivelsene allerede fra begynnelsen. Omgivelsene er her:

- logistikk-siden
- trening
- vedlikehold

Man kan også si at man har utvidet begrepet til å omfatte mer enn kontroll av de tradisjonelle sensorer og våpen. Når man har avgjort hvem som skal ta avgjørelser og hvor, må vedkommende tilføres den informasjon som er nødvendig for å øke sannsynligheten for at de avgjørelser som taes er de optimale for den aktuelle situasjon. Hva da med informasjon angående fremdriftsmaskineri?

Som dere har oppdaget, har jeg hittil lagt vekt på den integrerte system-filosofien, fordi det er den som vil være den avgjørende for systemene i 90-årene. Hvor langt ned i detaljer Sjøforsvaret som kunde skal bevege seg, må vurderes i hvert enkelt tilfelle, men de erfaringer som er trukket spesielt fra ULA-prosjektet, viser at vi som kunde bør bevege oss helt ned i system design spesifikasjonen. Det er eneste måten vi kan forsikre oss om at de operative krav og ikke minst intensjonene bak kravene blir oppfylt på en måte som tilfredsstillende oss for ikke bare levering av systemene, men også senere drift og vedlikehold. Det er også eneste måten vi kan avgjøre grensesnittet mellom de ulike komponentene og ta avgjørelser om hvem som skal ta for seg integrasjonsansvaret.

Det er kanskje ventet at jeg skal si noen ord om hvordan 90-årenes KKI-systemer vil se ut. Jeg ser egentlig ikke noe særlig hensikt med dette, for satt på spissen vil alt være mulig. Teknologien har kommet langt, det er opp til oss å utnytte den. Man må først av alt huske at vi fortsatt er avhengig av mennesket for å kunne ta visse avgjørelser. Det er derfor avgjørende at mennesket blir betraktet som en del av systemet allerede fra starten, og mennesket må få anledning til å utnytte sin erfaring og kreativitet i oppgaveløsningene så lenge tiden tillater det, og vi må sørge for at de nødvendige hjelpemidler er til stede.

Det er vel også etterhvert blitt innlysende at det er datateknologien som har muliggjort dette. For å tilfredsstille de krav som settes til sikkerhet, pålitelighet, utvidelsespotensiale og funksjonsstabilitet, har man vendt seg mot distribuert prosessering med en åpen systemarkitektur. På det rent konkrete plan ser man for seg en forandring i den grafiske maskinen, det samme gjelder for data-lasten og for databasekravene.

### 3. PROGRAMVAREVEDLIKEHOLD

Det vil etter denne delen være nokså naturlig å gå over til programvare vedlikehold for de systemene som er nevnt over. Det har vært gjentatt flere ganger at system vedlikehold er en del av et systems livssyklus, og programvaren blir mer og mer styrende for de funksjoner et system skal utføre. For å illustrere hva programvarevedlikehold innebærer relativt sett kan følgende figurer vises, og dette skulle gi en indikasjon på at programvare er mer enn bare de applikasjoner man ser til vanlig.

Vedlikehold av et moderne KKI-system er i sin natur atskillig mer omfattende enn det tradisjonelle vedlikehold vi er vant med fra eksisterende systemer, som i det vesentlige er hardware-baserte.

Hva er så programvare vedlikehold? Den kan deles inn i fire forskjellige typer:

- KORREKTIV: Retting av feil
- ADAPTIV: Modifisering av systemet for tilpasning til forandring i omgivelsene
- PERFEKTIV: Innføring av nye muligheter, modifisering av eksisterende funksjoner og generell utvidelse
- PREVENTIV: System forandringer for å forbedre fremtidig vedlikeholdsvennlighet eller funksjonsstabilitet, eller å lage en bedre basis for fremtidig utvidelse

Hovedproblemet med vedlikeholdet er at enhver forandring har iboende en risiko for uventede konsekvenser for deler av systemet som ikke skulle være berørt. Det er også umulig å teste ut hele systemet.

Det arbeidet som hittil har vært gjort i f m programvare vedlikehold innen Sjøforsvaret har ført til en del klare konklusjoner:

- programvare vedlikehold må sees i sammenheng med hele systemet p g a den nære sammenheng mellom hardware, programvare og operatørene
- programvare vedlikehold er i sin form og ressurskrav å sammenligne med programvare/system utvikling
- vedlikeholdbarhet må bygges inn i systemet fra starten, og kan ikke komme som en ettertanke
- vedlikeholdet vil bl a bestå av en løpende verifisering av systemet i hele dets livssyklus
- det er et sterk behov for modeller og verktøy som er strukturert i h t de ulike funksjonelle aspekter og de ulike vedlikeholdsoppgaver
- det er behov for referanse-systemer for å identifisere rapporterte feil, for testing av forandringer og verifisering av systemet
- det er behov for en "test-bed"
- det må eksistere aktuell dokumentasjon
- det må være muligheter for å lagre og gjenfinne kunnskaper om systemet of forandringer som er gjort
- Sjøforsvaret må utføre alt forarbeidet ned til programvare top-design, samt ha full konfigurasjonskontroll

Følgende figur er en overordnet oversikt over programvarevedlikehold sett i f m selve systemutviklings-prosessen, og den neste viser gangen i vedlikeholdet.

#### 4. STATUS OG FREMTID

Det burde være klart fra det forutgående hvor jeg mener det bør settes inn ressurser, men man må da akseptere at dette koster både m h p personell, penger og tid. Hvor er f eks SFKs EDB-strategi? Jeg gikk ikke i noen særlig detalj om hvor omfattende en systemanalyse er, men det er helt klart at vår hjerne kommer til kort når det gjelder å holde styr på mer enn 6-8 aktiviteter samtidig. Vi trenger derfor verktøy. Man hører stadig om både 4. generasjons-språk, modeller og metoder, men felles for de alle er at de ikke tar for seg hele den integrerte sammenhengen i et systems livssyklus.

Gjennom P 6071 og P 6157 har det blitt iverksatt en del ting for å skyve SFK i riktig retning i f t de oppgaver vi har foran oss på dette feltet. I f m anskaffelsen av trenere til ULA-klassen, vil Singer Link-Miles benytte et system som gir omgivelser for integrert prosjektstøtte. Systemet kalles ISTAR IPSE ("Integrated Project Support Environment") og er utviklet av det engelske firmaet "Imperial Software

Technology (IST)". Systemet dekker hele produktets livssyklus med språk-uavhengige verktøy for prosjektstyring, utvikling og konfigurasjon/data kontroll. Settet av verktøy kan utvides av brukerne, og eksisterende verktøy kan inkorporeres uten modifikasjoner. Selve design-målet med ISTAR var å oppnå muligheter til å integrere verktøy som arbeidsstasjoner som deler ISTARs bruker-grensesnitt og data kontroll fasiliteter.

Fordelene ved valget av ISTAR kan summeres som følger:

- utvidelses-muligheter for nye eller modifikasjon av verktøy for å forbedre programvare vedlikehold og samtidig beholde en konsistent bruker-grensesnitt
- mulighet for holde kontroll med og se sammenhengen i de ulike systemkomponenter, dokumenter og verktøy
- mulighet for å forandre planlagte aktiviteter med tilgjengelige ressurser
- ensartet bruk som letter forståelse og operering av de nødvendige oppgaver

I f m de arbeider som er gjort og gjøres i p 6071 spesielt i f m MMI-design i samarbeid med FFI, er det benyttet en metodikk som dekker de feltene som er beskrevet. Dette systemet er nå i ferd med å bli automatisert under navnet RDD-100.

Målet med dette systemet er at et system må bli dekomponert i funksjoner, funksjonene må bli overført til komponenter, og disse komponentene må designes, bygges og testes for å tilfredsstille de opprinnelige krav. Dette gjøres med den forutsetning at problemene er klart formulert på alle nivå samtidig som tids-sekvenser (rekkefølge av aktiviteter/prosesser), tilstander (operativ/defekt) og krav til utførelse (performance) har avgjørende betydning, videre er handlinger/aktiviteter rettet mot definerte objekter.

Fordeler med metoden kan sies å være:

- den lar oss spesifisere dekomponeringen av kravene til utførelsen (performance)
- den lar oss beskrive nivået hvor en skal ta seg av feil og derved systemets evne til å være feil-tolerant
- lar oss spesifisere hvordan man skal møte begrensede ressurser
- den understøtter et strengt definert system utviklings-mønster
- den gir oss en konstruktiv metode for å bevare avgjørelser som er tatt i kravspesifiseringen ut i kodingen
- den muliggjør en modell-beskrivelse av systemet
- den muliggjør identifisering av de beslutningene som ligger bak omformingene av systemkravene til distribuerte prosesser

Har man nå to metoder man går inn for? Nei, RDD-100 vil være et utviklingsverktøy som går inn i ISTAR som en arbeidsstasjon for å holde bl a full konfigurasjonskontroll. Den er altså et av verktøyene.

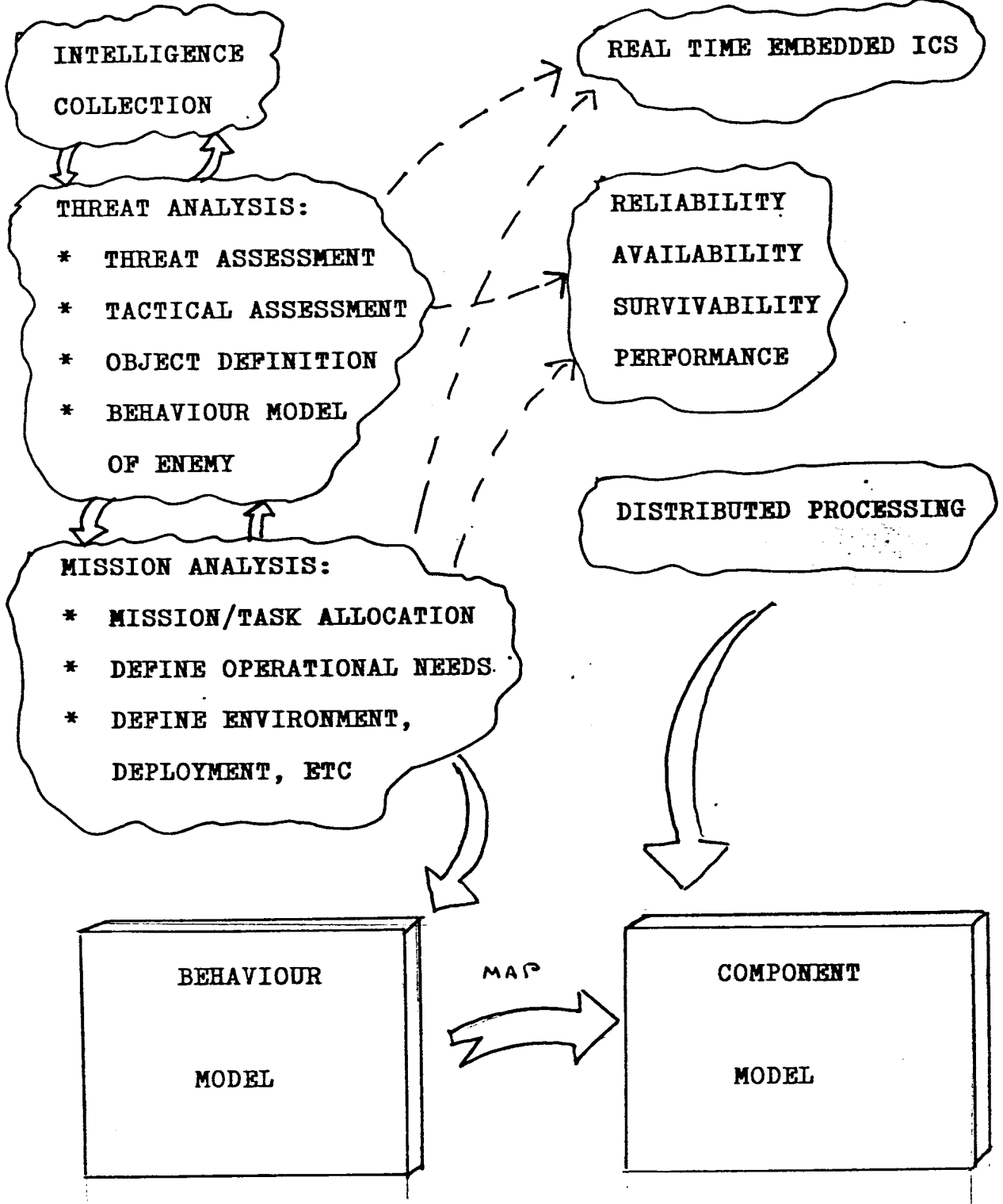
## 5. AVSLUTNING

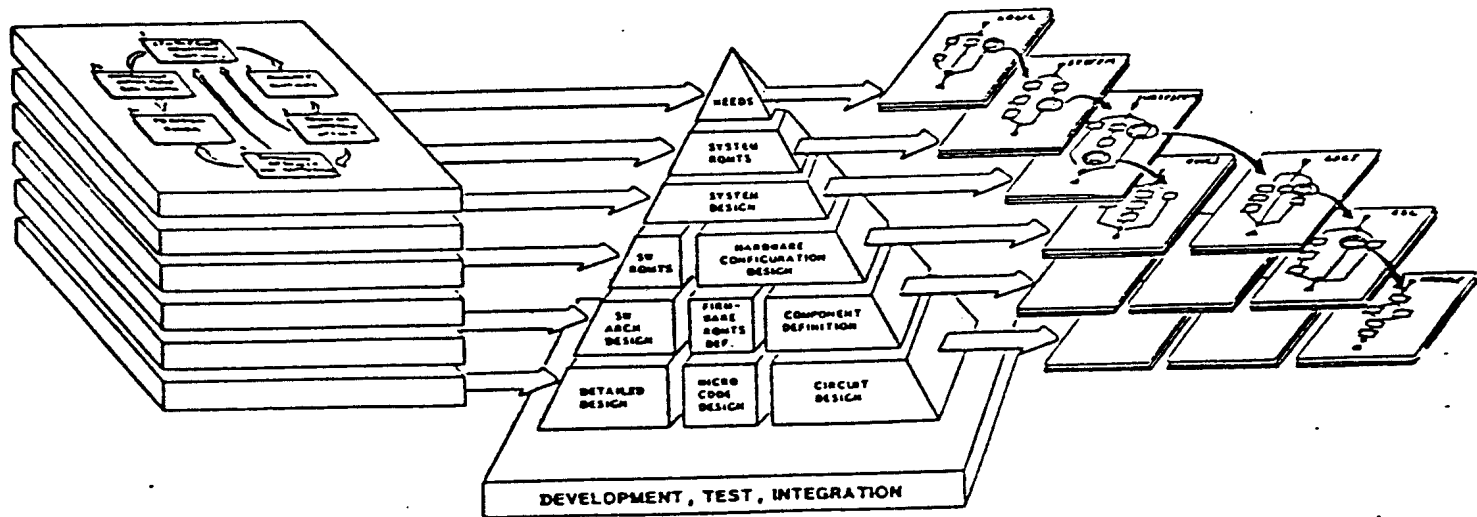
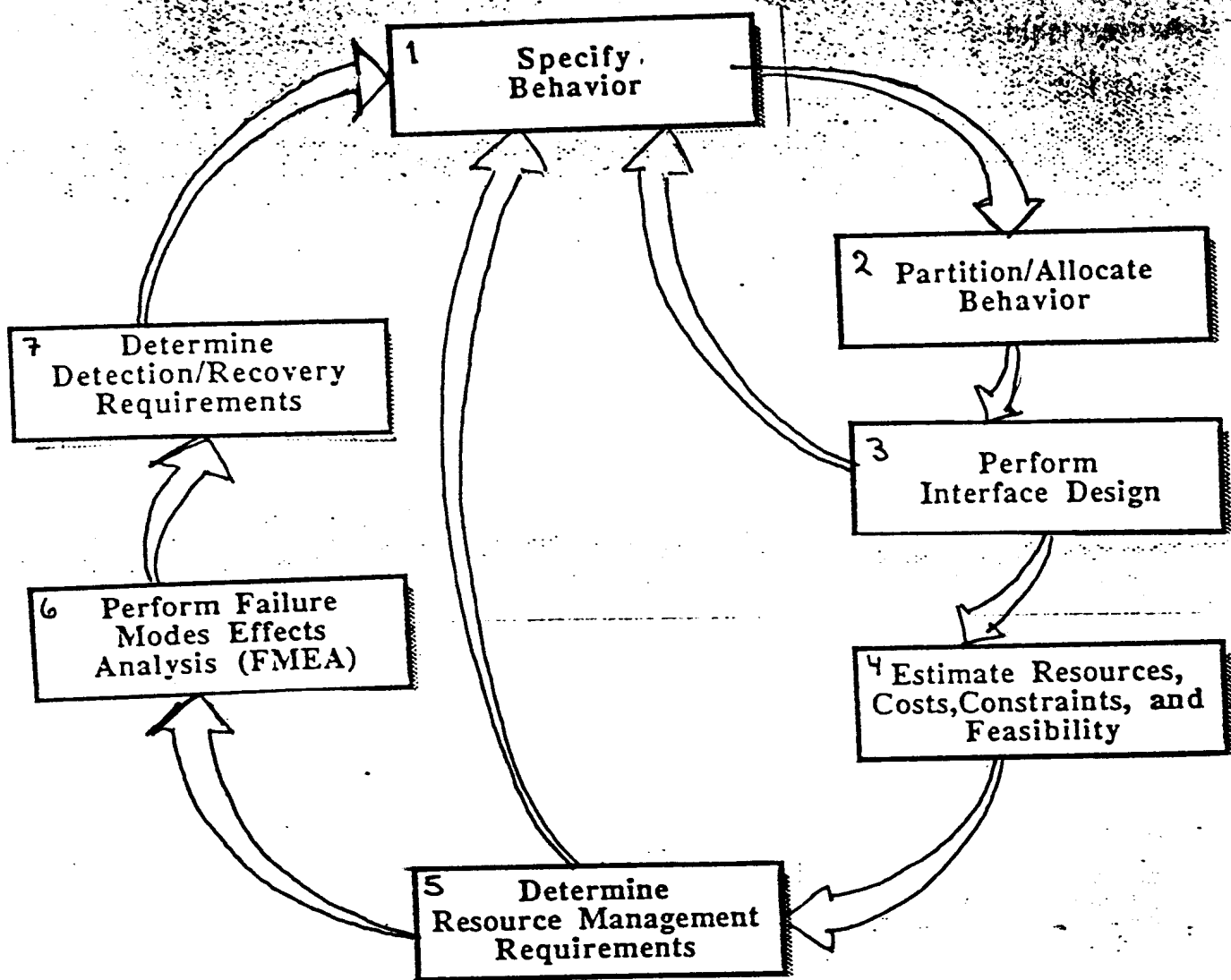
Tiden tillater dessverre ikke at man går i detaljer innen de nevnte områdene. Jeg håper bare at jeg har vært i stand til å gi en viss forståelse for at vi først og fremst snakker om integrerte systemer hvor ingen enkelt del må få eksistere i isolasjon. Sub-optimalisering vil aldri få oss frem til målet: utvikling, drift og vedlikehold av systemer som oppfyller de stadig foranderlige operative krav og som ligger innenfor de tilgjengelige ressurser.

Systemet må allerede i idefasen ta hensyn til hele livs-syklusen, og forhold som trening, logistikk og vedlikeholdbarhet må bygges inn i systemet fra første stund. For SFK er det av avgjørende betydning at man innser at innføring av ny teknologi adderer et nytt aspekt, og at man er villig til å forlate bokstenkningen og gå over til en integrert systemtenkning.

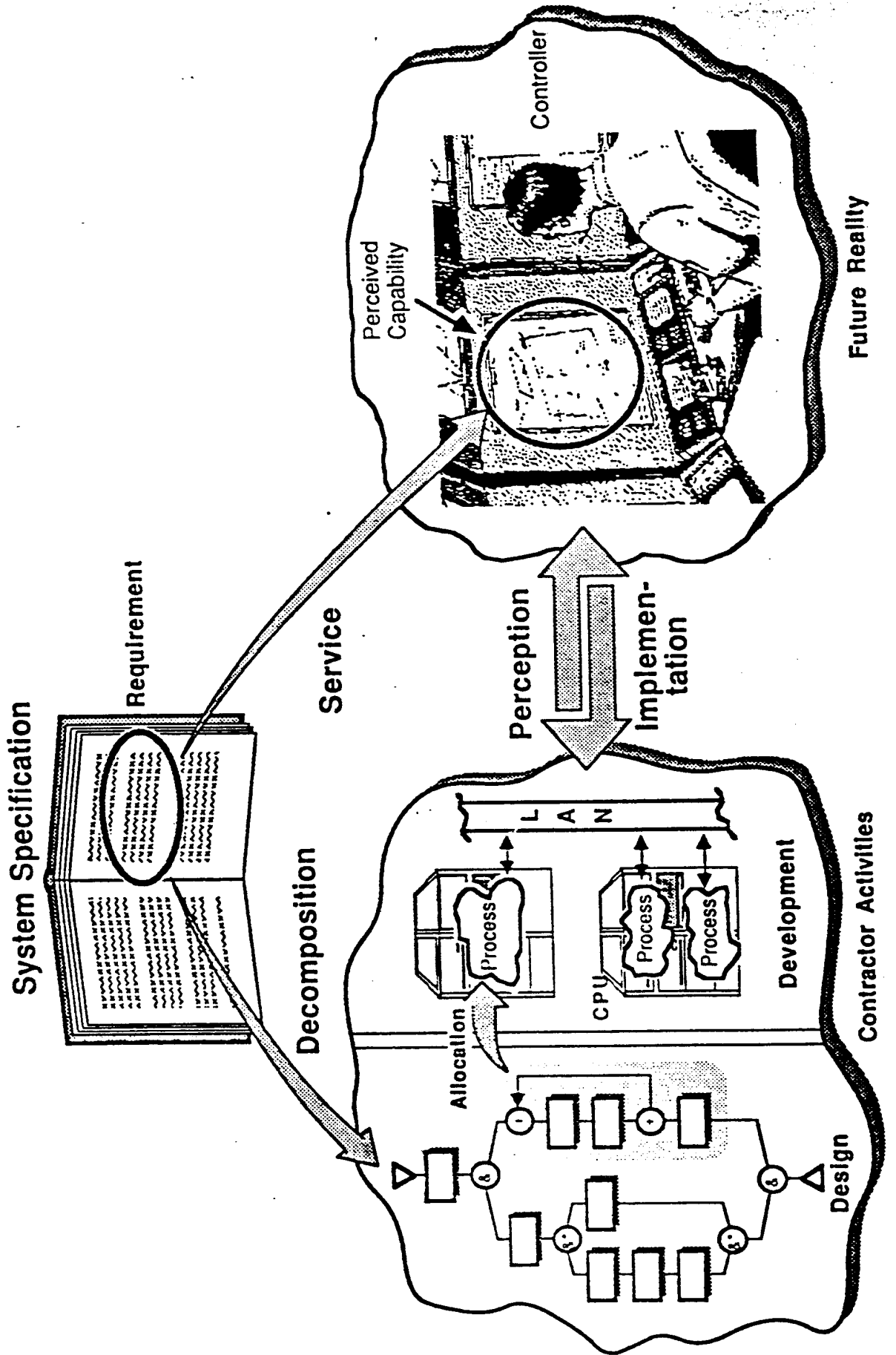
OPERATIONAL

TECHNICAL

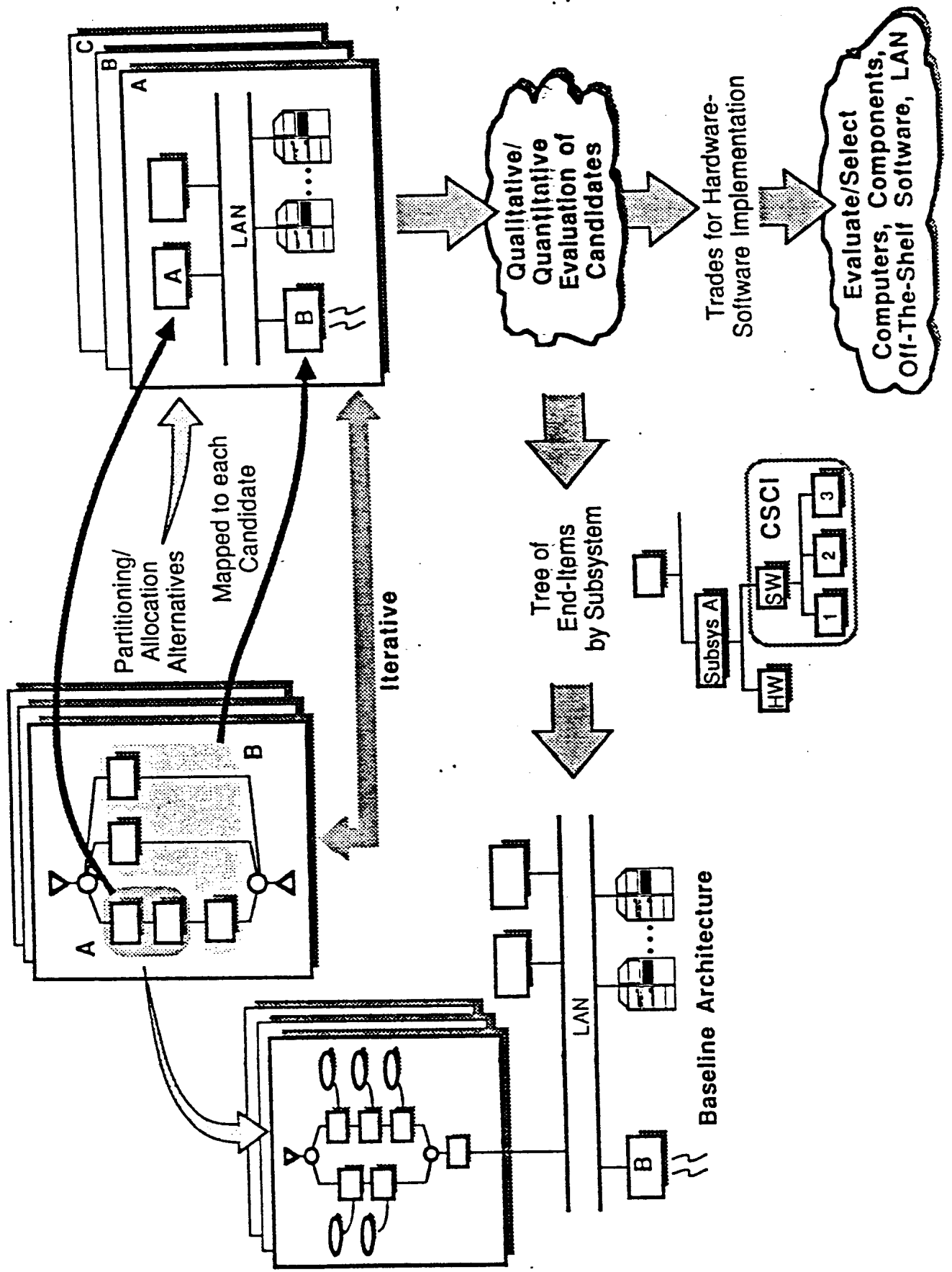


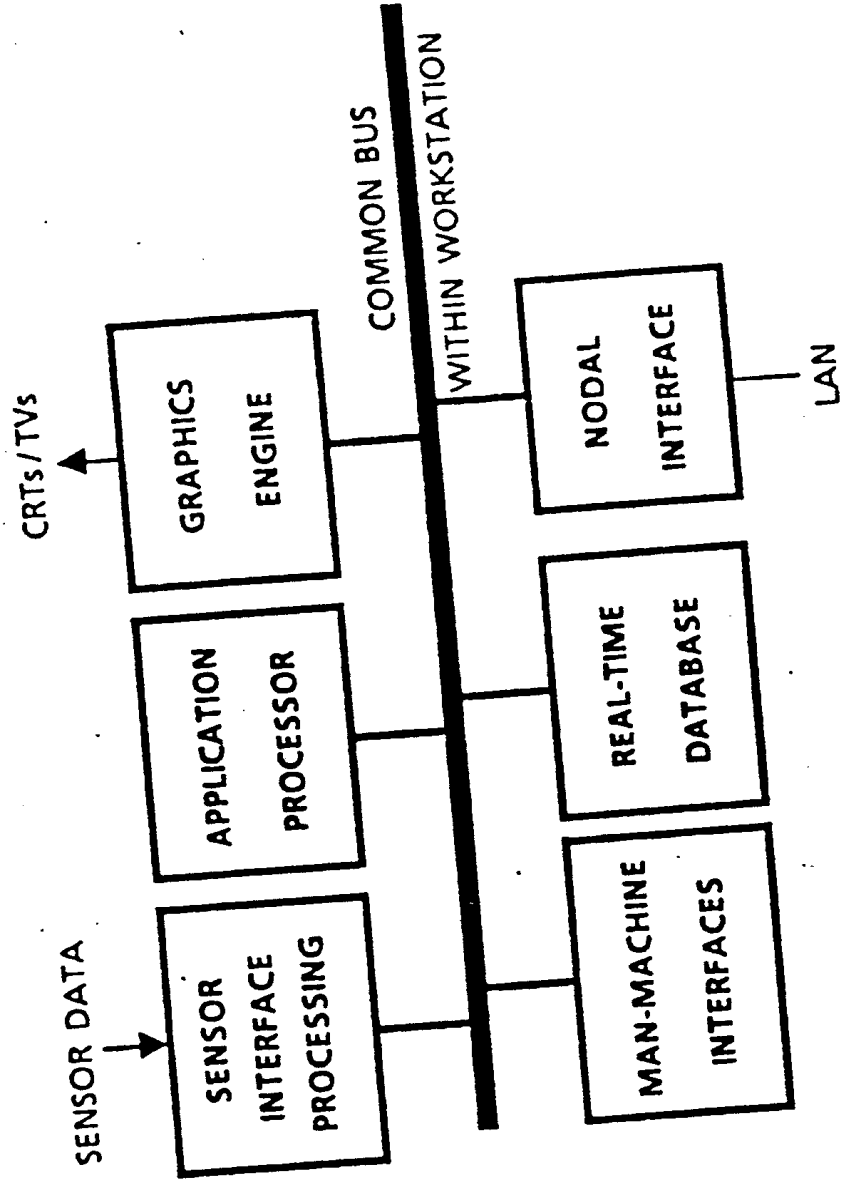


# ALIGNING INTERNAL MODELS WITH FUTURE REALITY



# ARCHITECTURE DEFINITION





	<u>Present</u>	<u>Future</u>
Resolution	1280 x 1024	2K x 2K x N
Refresh Rate	60Hz	60Hz
Video Output B.W.	>100MHz (RGB)	>300MHz (RGB)
Screen Update Rate	<1 Sec.	<0.5 Sec.
Graphics Interface	GKS/PHIGS	GKS/PHIGS
Scarf Overlays	1K x 1K x N	
Graphics Processing	Clipping, Windowing, Transformations, Fill	

Graphics Engine Requirements for C<sup>3</sup> Applications

Tracks  
Symbology  
Alphanumerics  
Ellipses & Circles  
Vectors  
Maps  
Screen Update  
Graphics Interface

>1000  
>200  
>4K  
>10  
>500  
>10K overlay vectors  
<1 Sec.  
High Level Language

Current database characteristics may be summarized as:

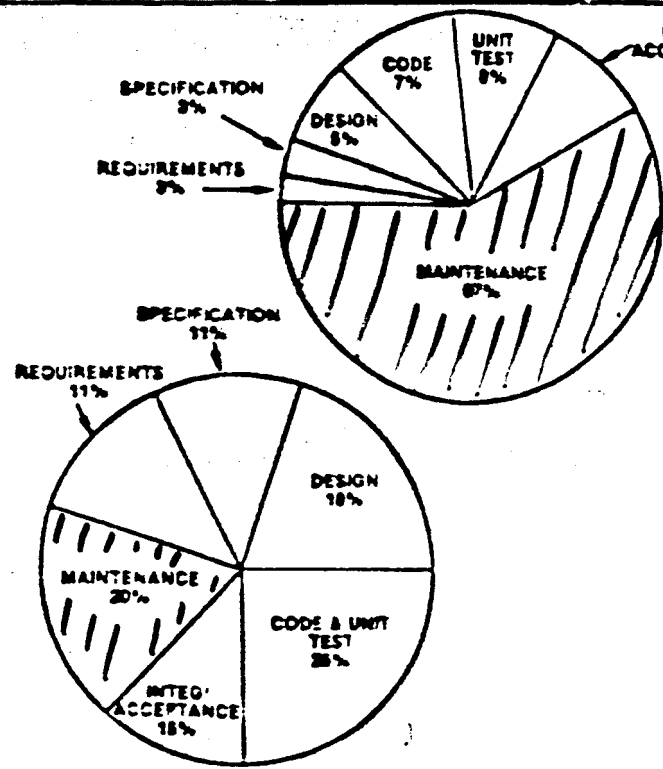
Track Capacity.....200 to 500  
Size.....150Kbytes to 1Mbyte  
Write to Read Access Ratio.....3 to 1  
Simple Access Frequency.....up to 500 per sec

A 1990's system workstation database may require:

Track capacity.....500 to 2,000 tracks  
Size.....2 to 20Mbytes  
Access Frequency.....> 1000 per sec

6

# THE COST OF PRODUCT ASSURANCE



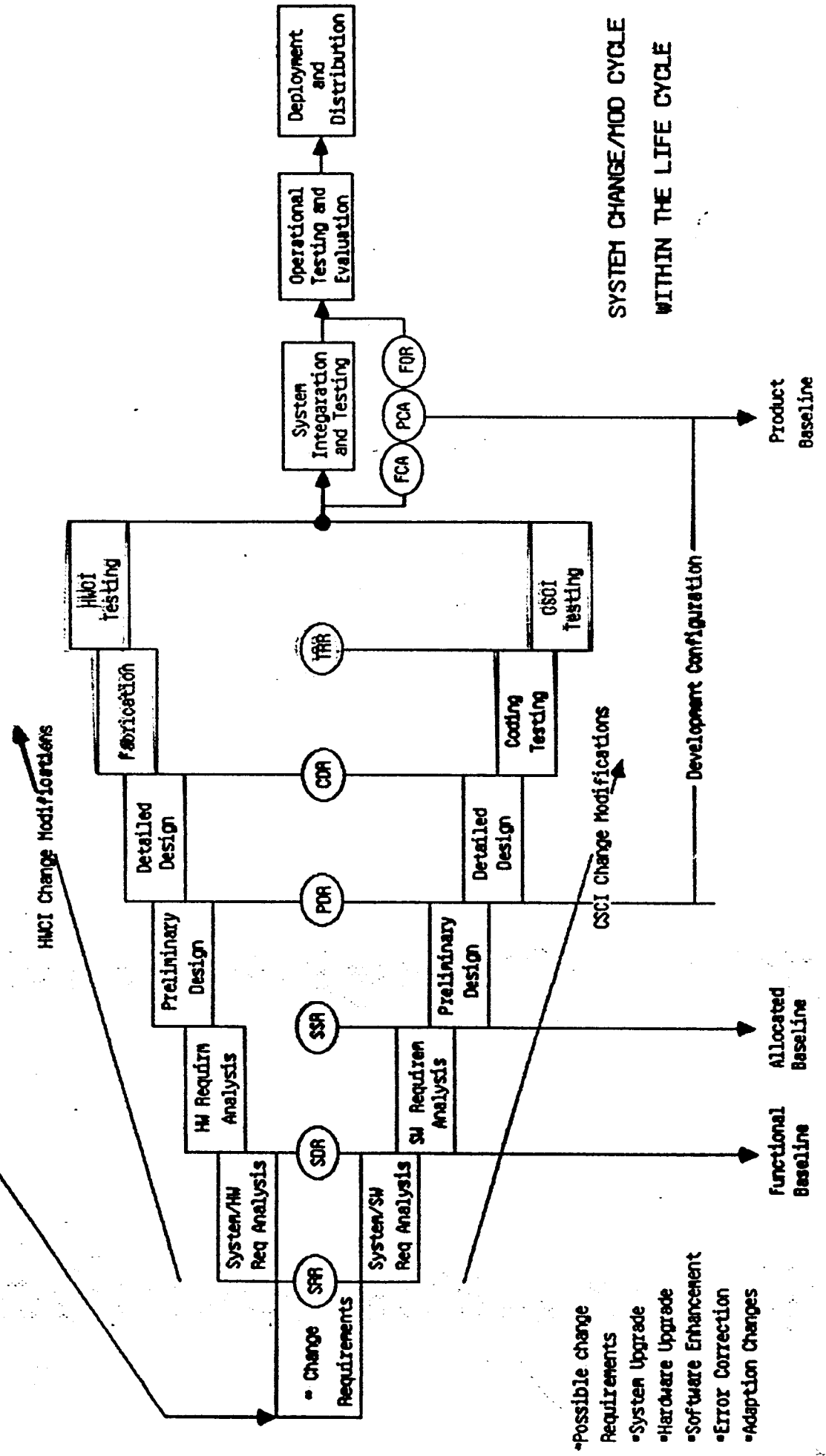
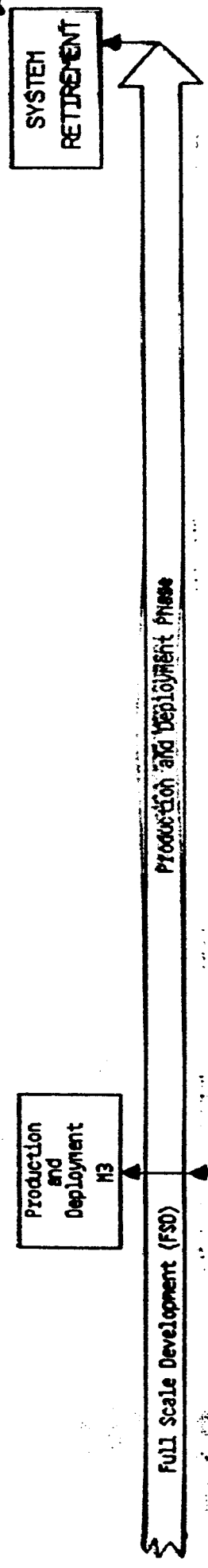
<u>SYSTEM DEVELOPMENT</u>		<u>\$19,000,000</u>
REQUIREMENTS	80.0M	
SPECIFICATION	8.0M	
DESIGN	1.5M	
CODE	2.15M	
UNIT TEST	2.4M	
INTEG / ACCEPT	2.15M	
<u>SYSTEM MAINTENANCE</u>	<u>\$20,000,000</u>	
<b>TOTAL</b>	<b>\$39,000,000</b>	

<u>SYSTEM DEVELOPMENT</u>		<u>\$15,000,000</u>
REQUIREMENTS	\$2.2M	
SPECIFICATION	2.2M	
DESIGN	3.6M	
CODE		
UNIT TEST	3.8M	
INTEG / ACCEPT	3.8M	
<u>SYSTEM MAINTENANCE</u>	<u>\$4,000,000</u>	
<b>TOTAL</b>	<b>\$25,000,000</b>	

## COST TO REPAIR SOFTWARE IN RELATION TO LIFE CYCLE

Phase	Relative Cost of Repair
Requirements	.1 - .2
Design	0.5
Code	1
Unit Test	2
Acceptance Test	5
<u>Maintenance</u>	<u>20</u>





19 2/2

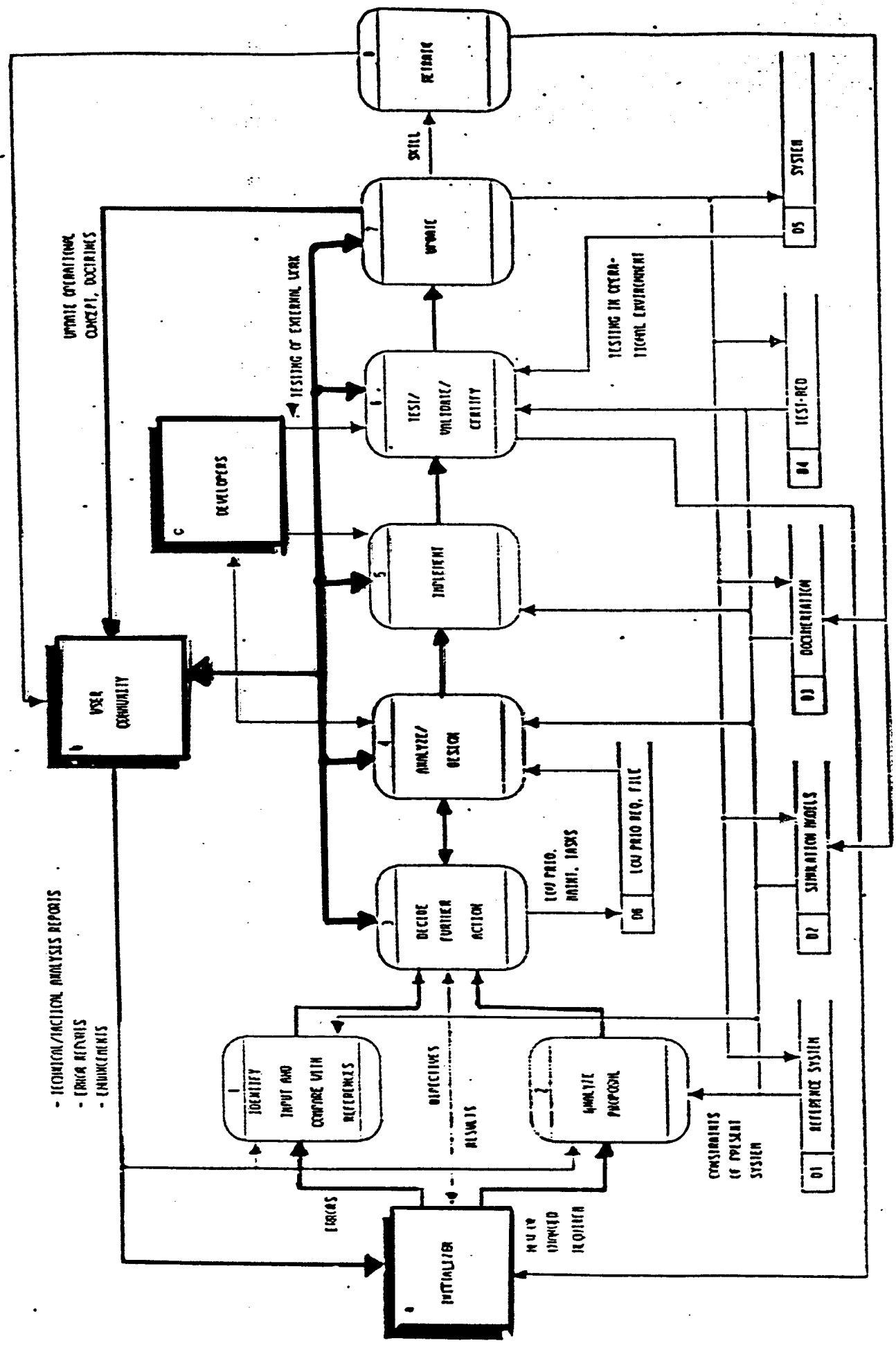


Figure 10-1 ICS Overall Maintenance

# ISTAR FRAMEWORK

# ISTAR TOOLSET

