Handbok för programvara i säkerhetskritiska tillämpningar

FÖRSVARSMAKTEN
Försvarets materielverk

2001-12-13

FM beteckning
14910:61171
FMV beteckning
Plan 14910:3476/01
Publikation
M7762-000531

Handbok för Programvara i säkerhetskritiska tillämpningar (H ProgSäk), 2001 års utgåva (M 7762-000531), fastställs att tillämpas från och med 2002-03-01 inom Försvarsmakten samt Försvarets materielverk.

FÖRSVARSMAKTEN

Folke Rehnström
C HKV KRI

Försvarets materielverk

Staffan Näsström
C FMV:SYST

Boken är publicerad i samarbete med
Mediablocket AB

Förord

Handbok för Programvara i säkerhetskritiska tillämpningar (H ProgSäk 2001) innehåller Försvarsmaktens och Försvarets materielverks rekommendationer för anskaffning av programvara i säkerhetskritiska system. Handboken baseras på internationella standarder för programvarans livscykelprocesser (ISO/IEC 12207 1995), kvalitetshantering och kvalitetssäkring (ISO 9000-3, 1997).

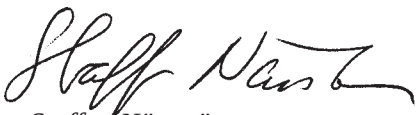
Föreskrifter för systemsäkerhetsarbetet anges i Försvarsmaktens handbok för Systemsäkerhet (H SystSäk 1996).

Systemsäkerhet är en egenskap, som ställer krav på dels de parter som är involverade i en upphandling, dels på de processer dessa använder sig av samt på systemets ingående delar.

En systematisk genomgång av programvarans roll för systemsäkerheten har strukturerats efter roll, process och produkt över hela programvarucykeln. Dessa generella egenskaper för programvarusäkerhet måste bedömas i sitt sammanhang; det system där programvaran skall ingå, den omgivning och den användning systemet är avsett för. Rekommendationer ges för hur urval och komplettering av de generella säkerhetskraven till aktuellt system kan utformas.

Handbok för Programvara i säkerhetskritiska tillämpningar är resultatet av en successiv vidareutveckling från 1997 av arbetsversioner som granskats och provats inom och utom FMV. Framtida revideringar kommer även fortsättningsvis att genomföras och grundas på erfarenheter av och synpunkter på handboken, vilka kan lämnas till FMV:SYST.

Folke Renström
Chef för HKV Krigsförbandsledning



Staffan Näsström
Chef för FMV Systemledning

Innehållsförteckning

1 Handbokens inriktning

Elektronik och programvara (EoP) övertar allt mer av ansvaret för övervakning, beslutsfattande och styrning av komplexa, precisionskrävande och säkerhetskritiska uppgifter. Många uppgifter, som tidigare sköttes av maskinist, förare eller operatör har automatiserats. Personansvaret har därvid kommit att spridas på fler roller, bl a kravställare och utvecklare. Tillämpningarna varierar till inriktning och storlek. De återfinns i alla typer av system; från apparater för privat bruk till utrustningar inom medicin, system för transport och i olika produktionsprocesser. För denna typ av system gäller, att de inte oavsiktligt får bidraga till att person, egendom eller miljö skadas. Systemsäkerhet är med andra ord en väsentlig egenskap.

Detta medför extra krav såväl på de delar, som har inverkan på systemsäkerheten, som på de personer, som är involverade vid anskaffning, produktion och användning av säkerhetskritiska system och de arbetsprocesser dessa använder sig av. Kraven spänner över ett brett fält av såväl tekniska som psykologiska aspekter och syftar till att öka förvissningen om att systemet i avsedd miljö inte kan orsaka allvarliga skador.

Behovet av rekommendationer för hur systemsäkerhetsfrågor skall hanteras inom dessa typer av system är stort. En handbok i systemsäkerhet för FM, H SystSäk, fastställdes 1996. Föreliggande handbok, H ProgSäk, behandlar speciellt programvarusäkerhet, dvs de egenskaper i programvara och dess hantering, som bidrar till att systemsäkerheten kan upprätthållas.

1.1 Syfte

H ProgSäk är avsedd att understödja anskaffning av säkerhetskritisk programvara till FM, genom att ge vägledning i hur systemsäkerhetsverksamheten kan tillämpas på programvara under hela dess livscykel. Programvarusäkerhet är därvid en del av FM:s övergripande systemsäkerhetsverksamhet.

1.2 Omfattning

I H ProgSäk formuleras de allmänna systemsäkerhetskrav som kan vara aktuella för nyutvecklad och återanvänd programvara. Vägledning ges i hur säkerhetskrav för den specifika applikationen tas fram. Innan programvaruanskaffning och utveckling påbörjas skall en anpassning till aktuellt projekt utföras genom urval av i handboken rekommenderade krav (se 1.9.).

Aspekter av betydelse för systemsäkerheten vid hantering och produktion av programvara behandlas. De krav som specificerats gäller, där inget annat utsägs, endast (säkerhets)kritiska delar i programvara och dess gränssnitt mot övriga systemdelar. Den kritiska programvaran kan vara av ändringsbar typ eller fast inkapslad i maskinvaran (s k fast program, *firmware*).

1.3 Avgränsningar

Handboken behandlar ej i detalj krav, som gäller

- generell programvara[1], produktionsprocess och personal,
- elektronik/maskinvara,
- övergripande systemaspekter[2],
- IT-säkerhet[3].

Undantag är fall, där H ProgSäk saknar styrande handböcker att referera till. När dessa föreligger kommer innehåll, som inte specifikt gäller kritisk programvara (kap 5.) att kunna rensas ut.

1.4 Giltighet

Föreliggande utgåva av handboken är en slutversion från framtagning av H ProgSäk (se 1.14). Handbokens giltighet, tillämpning och anpassning inom FMV avses att regleras enligt särskild ordning (se Förord).

1.5 Tillämplighet

H ProgSäk avser kritiska programvarusystem i olika stadier (från planering till avveckling) utan inriktning mot visst applikationsområde. För dessa system skall handboken tillämpas på programvara under utveckling och underhåll samt all typ av återanvänd programvara, såväl hyllvaror från FM (GOTS) som från industri (COTS).

Handboken skall användas inom ramen för H SystSäk efter inledande systemsäkerhetsanalys på övergripande systemnivå (dvs för det översta systemet – ett system av system) enligt H SystSäk. Detta innebär t ex, att analys av vilka delar, som skall klassas som kritiska, har utförts för varje separat system

1. Generell i betydelsen icke-säkerhetskritisk.
2. Säkerhetsaspekter på systemnivå täcks av H SystSäk, utgångspunkt för programsäkerhetsarbetet.
3. IT-säkerhetskrav hanteras enligt andra riktlinjer inom FM och FMV. Se 6.1.2., 6.9.3.

ned till delsystem och programvarunivå. Till denna klass räknas dels programdelar, som kan orsaka eller bidraga till att ett osäkert systemtillstånd uppstår, dels de, som har till uppgift att upptäcka, hindra eller lindra ett förestående osäkert tillstånd.

1.6 Kravkategorier

Specificering av säkerhetskrav syftar till att upprätthålla systemsäkerheten på den nivå, som projektet i enlighet med H SystSäk funnit vara tolerabel för systemet.

Fyra kategorier av säkerhetskrav kan identifieras för programvara:

Kravkategori/ ***Kravtyp***	**Programvaruklass**	**Kommentarer**
Grundkrav Kap 5. *Allmänna krav*	Icke-kritisk samt kritisk programvara.	Krav m a p processer, stödverktyg och dokumentation. Kraven är obligatoriska för kritisk programvara.
Generella säkerhetskrav Kap 2., 3. , 4. *Säkerhetskrav*	Kritisk programvara oavsett applikationsdomän.	Komplettering av grundkrav m a p process, personalkvalifikationer, produkt, produktionsmiljö.
Domänspecifika säkerhetskrav *Säkerhetskrav*	Kritisk programvara inom viss applikationsdomän.	Komplettering av de generella säkerhetskraven genom analys av säkerhetsrisker för aktuellt tillämpningsområde.
Systemspecifika säkerhetskrav *Säkerhetskrav*	Kritisk programvara som del av det övergripande systemet i dess avsedda omgivning och användning.	Begränsningar och tillägg av föregående kravkategorier baserade på säkerhetsanalys av applikationsproprogramvaran som integrerad del av aktuellt totalsystem.

H ProgSäk omfattar endast grundkrav och generella säkerhetskrav. En komplettering med säkerhetskrav specifika för den aktuella applikationen är därför nödvändig. Dessa identifieras under projektets inledande systemsäkerhetsanalyser och detaljeras allt eftersom den inre strukturen av de kritiska programvarudelarna tar form. I o m att den inre kärnan i varje kritisk del blir känd kan mer specifika säkerhetskrav formuleras (se 3.4.4.).

Exempel på olika typer av systemsäkerhetsanalys relevanta för programvara ges i bilaga 6.8.

1.7 Kritikalitetsklasser

Handboken innehåller krav, vilka klassats, för att ange om de avser programvarudel av hög, {H}, medelhög, {M}, eller låg kritikalitet, {L}[4]. Strängare krav ställs på delar, som kan innebära ett starkare hot mot systemsäkerheten. Detta förutsätter, att delar av olika kritikalitet isoleras från varandra och att kritiska delar begränsas så långt möjligt, för att de hårdaste kraven och de kostnader dessa medför skall kunna hållas till ett minimum.

Krav som gäller de två högsta eller lägsta klasserna har markerats med {HM} resp {ML}. Krav, som ställs för samtliga kritikalitetsklasser betecknas {HML}. Krav relevanta för en programvarudel av lägsta kritikalitet är m a o grundkrav samt generella säkerhetskrav märkta {L},{ML} och {HML}. Klassningen är ett stöd vid kravställningen och kan anpassas efter det aktuella systemet.

1.8 Kravnumrering

Varje krav har tilldelats ett unikt kravnummer bestående av tre delar.

Del	Förklaring
1	**Unik siffra** för H ProgSäk avslutad med punkt (dvs **6**.).
2	Kravets avsnittsnummer utan punkter.
3	**Löpnummer** för krav räknat från avsnittets början och inlett av bokstaven **K**.

Exempel: Numret för första kravet i avsnitt 4.3.2.2.5 är **6**.43225**K**1. Numret för första kravet under avsnitt 3.1 är **6**.31**K**1.

I denna utgåva inleds en kravsats enbart med löpnummer.

1.9 Anpassning

För varje system som visar sig innehålla säkerhetskritisk programvara skall säkerhetskrav ställas även beträffande ingående programvarudelar. Ur H ProgSäk:s rekommenderade kravlista görs ett urval av krav tillämpliga på aktuellt system. Vid urvalet skall det sammanhang i vilket ett valt H ProgSäk-krav är ställt, framgå[5]. Vilka delar som valts samt motiveringar för bortval dokumenteras. Vid anpassning skall framgå vilken risknivå, som kan betraktas som tolerabel för systemet. Detta styr i senare systemsäkerhetsanalyser av programvaran vilka delar, som skall klassas som kritiska samt graden av kritikalitet.

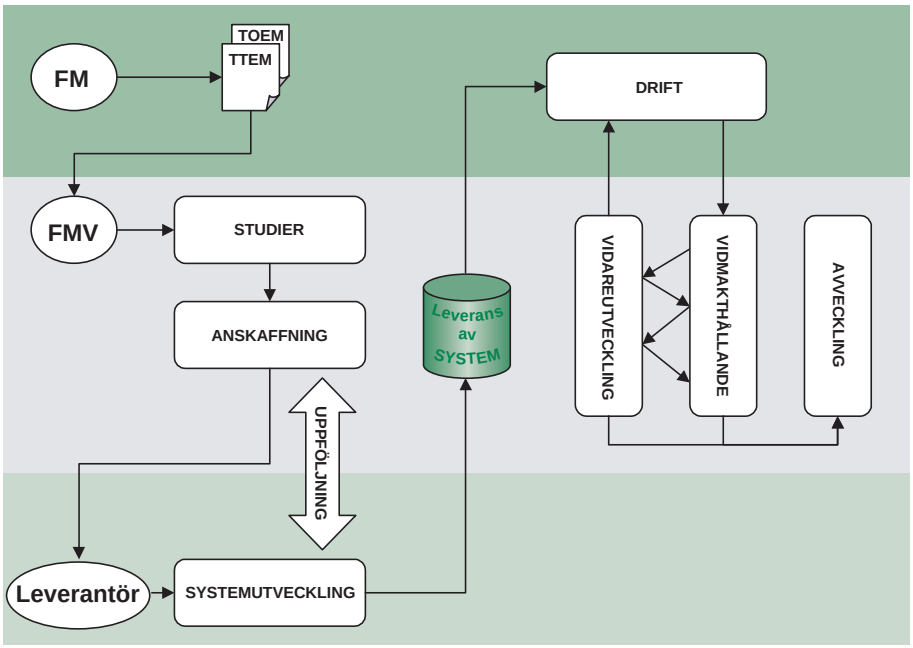
4. H SystSäk kräver att projektet definierar och klassificerar systemets risker för vådahändelser (konsekvens och inträffandefrekvens) och ger exempel på tre risknivåer.
5. Exempel: Det finns krav som berör ändringar i olika situationer. Se 3.3.3., 4.5.1. , 4.5.2.12.

I projekt, där enbart anskaffning och återanvändning av befintlig programvara är aktuell, kommer vissa delar av handboken ej att vara tillämpliga (t ex avsnitt 4.5.2. om nyutveckling).

Anpassning av H ProgSäk till enskilt projekt kan göras successivt, t ex genom att först ta fram en sektorspecifik variant gemensam för samtliga applikationer inom området, vilken därefter kan ligga till grund för det enskilda projektet. Se vidare anvisningar under avsnitt 6.4.1.

1.10 Roller och faser

En förenklad beskrivning av interaktionen mellan olika aktörer under anskaffningens faser ges i följande bild [6]:



1.11 Läsanvisningar

Programvarusäkerhet ställer en mängd kvalitativa krav på såväl själva programvaruprodukten som på personer och processer verksamma under dess realisering. För att underlätta för handbokens användare att avgöra vilka krav som är

6. Förtydligande: Vidmakthållande och vidareutveckling sker genom uppdrag till industrin.
Vidmakthållande avser aktiviteter för att bibehålla systemets ursprungliga funktionalitet och prestanda.
Vidareutveckling är en förbättring m a p dessa faktorer.

relevanta i aktuellt fall, vem de åsyftar samt vilka moment och vilka delar i anskaffningen som avses, har handboken strukturerats dels efter de huvudparter, som är involverade i ett anskaffningsärende (**Uppdragsgivare, Beställare, Leverantör**) dels för varje part efter dess **P**ersonalkvalifikationer, **P**rocess (dvs verksamhetsåtagande), **P**rodukt och i förekommande fall **P**roduktions- och måldatormiljö.

Denna huvudstruktur med dess understrukturer har visat sig mycket användbar vid jämförelser mellan olika standarder. Den kan också vara utgångspunkt för hur ett säkerhetutlåtande (*safety case*) kan detaljeras för programvara.

För att öka förståelsen för kravens innebörd och ytterligare förtydliga sammanhangen har i regel varje avsnitt inletts med informerande text före själva kravmassan. Kraven har ramats in, numrerats samt kritikalitetsklassats (se 1.7).

Kraven baseras i huvudsak på tre, kompletterande discipliner:

- **Programvaruteknik** inriktad mot en systematisk analys och förfining av givna krav, för att m h a lämplig utvecklingsmetodik, - modell och -miljö få fram en kostnadseffektiv implementation, som uppfyller kraven.
- **Tillförlitlighetsteori** med strategier för att undvika, detektera och eliminera felkällor samt kunna erbjuda korrekt funktionalitet trots resterade fel (feltolerans). Dessa strategier tillämpas *bottom-up*, dvs från enstaka komponent upp till integrerat system och ingår till stora delar i den gängse programvarutekniken.
- **Säkerhetsmetodik** vilken i stället koncentrerar sig på de riskkällor, som leder till förlust eller skada av liv, egendom eller miljö. Systematiska analyser från övergripande systemnivå och nedåt i varje separat system utförs, för att få fram vilka delar som kan utgöra säkerhetshot och leda till olycka. För dessa införs detaljerade säkerhetskrav och konstruktionsrestriktioner. Konstruktionsalternativ, som leder till eliminerade eller reducerade risker, väljs. Konstruktionen kompletteras med olika säkerhetsmekanismer, för att kvarvarande risker skall kunna bibehållas på tolerabel nivå under avsedd driftstid.

Kapitlens huvudsakliga innehåll och avsedd målgrupp:

Kap 1 Handbokens inriktning. Styrande och informativt avsnitt avsett för **alla**.

Kap 2 Generella säkerhetskrav på uppdragsgivare/slutanvändare. Avsedd för **FM**.

Kap 3 Generella säkerhetskrav på beställare. Avsedd för **FMV**:s anskaffare, kvalitetsgranskare.

Kap 4 Generella säkerhetskrav på leverantör[7] för anpassning till aktuellt projekt. Avsedd för **FMV**:s anskaffare, kravställare, granskningspersonal samt för **leverantör**ens konstruktörer och granskare.

7. Om Beställare åtar sig rollen som Leverantör, gäller kap 4 även Beställaren.

Kap 5 Grundkrav för all programvara. Avsedd för **alla**.

Kap 6 Bilagor. Informativa avsnitt för:

alla: Begrepp, Akronymer, Referenser, Användaranvisningar till H ProgSäk, Problemställningar, Publikationer.

konstruktörer och granskare: Metoder för systemsäkerhetsanalys, Checklistor.

Rekommenderad läsordning:

- Kap 1. Handbokens inriktning.
- 6.4. Anvisningar till H ProgSäk-användare
- 6.6. Problemställningar
- 6.7. Systemsäkerhetsarkitekturer
- 6.1.2. - 6.1.5. om Säkerhet, Tillförlitlighet, Risk, Kritikalitet
- 6.8.: Inledning under rubriken ”Metoder för systemsäkerhetsanalys”.
- 2. - 4.: Aktuellt kravkapitel (och därmed kap 5.).
- Resterande avsnitt: Används som uppslagsverk.

1.12 Styrande dokument

TjF-FMV 1997:11 (styrande för H SystSäk)
H SystSäk

1.13 Dokumentrelationer

En starkt förenklad bild av inriktningen hos denna och relaterade handböcker.

Inriktning	**Generell** (speciell systemtyp)	**Systemsäkerhet**	**IT-säkerhet**
Helt system (program- & maskinvara)	DIT 01 (IS/IT) H KRAVDOK (allmän) RML (militär luftfart)	H SystSäk	H SÄK IT (IT-system) ISO/IEC 15408 (CC) ITSEC
Programvarusystem		**H ProgSäk**	
Organisation			H SÄK Skydd, H Säk IT

1.14 Dokumenthistorik

Arbetsutgåvor

Version	Datum	Dokumentstatus	Etapp	Kommentarer
1.0	1997-09-30	Förstudieutgåva	1	Slutligt utkast för etapp 1.
1.1	1998-12-07	Arbetsversion för FMV-granskning	2	Klargöranden, kompletteringar, kritikalitetsklassning av krav.
2.0	1999-02-28	Remissversion	2	Kompletteringar m a a interna granskningskommentarer m m. Slutligt utkast för etapp 2.
2.1	1999-10-10	Arbetsversion (Provversion)	3	Kompletteringar m a a yttranden till version 2.0.
3.0	2000-11-17	Version för fastställelse	3	Nytt: 6.1.3, 6.4, 6.6.1, 6.7, 6.9.2.1. Uppdateringar: övrig info till aktuell status.

Fastställda utgåvor

Version	Datum	Dokumentstatus	Etapp	Kommentarer
1.0	2001-12-13	Fastställd utgåva för tryckning och översättning	4	Redigeringar av informativa delar m a p bilder, enstaka ord, referenser till aktuell status. Rättelse av löpnummer för krav under 5.2.4.1.2. Tillägg av fastställelsemissiv samt förord.

2 Uppdragsgivare/ slutanvändare (FM)

Detta avsnitt innehåller generella säkerhetskrav på personal och teknik inom FM, dels under studiefasen innan ett anskaffningsuppdrag ges till FMV, dels då produkten överlämnas av FMV till uppdragsgivare och slutanvändare (förband och enheter).

2.1 Personalkvalifikationer

Programvarusystem med säkerhetskritiska inslag kräver en samlad insats från samtliga parter involverade i dess realisering. För att undvika överraskande och onödiga kostnader behövs tidig insikt i vilka de presumtiva säkerhetsriskerna är, analys av dessa samt planering för lämpliga säkerhetsåtgärder. Detta förutsätter goda kontakter mellan involverade parter samt kontinuitet och långsiktighet i uppdragsgivarens ansvar och relationer, främst med FMV och med slutanvändarna.

Uppdragsgivarens förmåga, att kunna formulera rätt målsättningskrav, är väsentlig, för att nå fram till en fullständig, korrekt och motsägelsefri kravspecifikation, vilken i sin till tur är en förutsättning för, att **rätt system** skall kunna beställas och för att konstruktion, verifiering och validering skall kunna utföras på ett kostnadseffektivt sätt [8].

Säkerhetskraven formuleras med syfte att det slutliga systemet skall hålla sig på en tolerabel risknivå. Specificerad funktionalitet balanseras mot identifierade risker.

1. Uppdragsgivare och slutanvändare skall ha insikt i system- och programvarusäkerhetsfrågor {HML}.
2. Uppdragsgivare skall ha goda kunskaper i de systemsäkerhetsrisker som är förknippade med slutanvändares krav inom applikationsområdet {HML}.
3. Slutanvändare skall ha goda kunskaper om och erfarenheter av systemets användningsområde samt av risker, speciellt för personal, utrustning och miljö vid handhavandet {HML}.

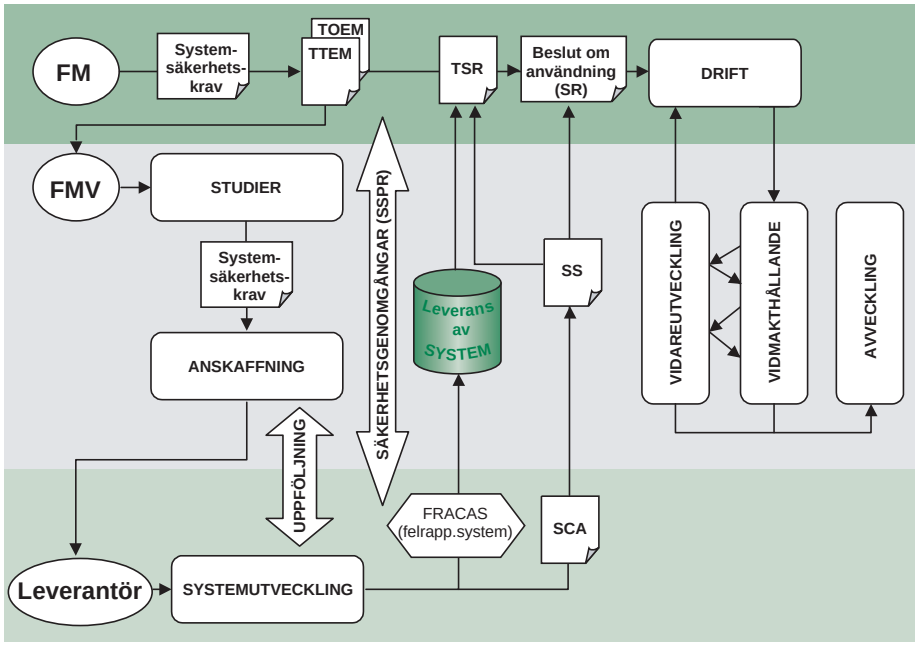
8. Analys av inträffade incidenter i programvarusystem har visat att > 44% orsakats av fel i kravspecifikationerna. Dessa fel är också de mest kostsamma att åtgärda. Stora kostnader ligger även i validering och verifiering (främst testning). För säkerhetskritiska system har siffror på mer än 50% av de totala projektkostnaderna rapporterats. Dessa system kan vara en faktor 7-9 gånger dyrare än icke säkerhetskritiska. Merkostnaden beror främst på höga test- och dokumentationskrav. Positiva effekter har dock visat sig, där man tidigt i framtagningsprocessen vinnlagt sig om tätare interaktion med berörda parter.

2.2 Styrprocesser

Detta avsnitt begränsar sig till de krav, som ställs på stödjande aktiviteter, för att möjliggöra en kostnadseffektiv materielanskaffning (se 3.3.). Dessa aktiviteter löper över materielens hela livscykel: från idé till avveckling. Kraven berör främst uppdragsgivare inom HKV.

2.2.1 Systemsäkerhetsplanering, -ledning och -uppföljning

H SystSäk beskriver systemsäkerhetsverksamheten under ett systems livstid för olika aktörer (**FM, FMV, Leverantör**). För FM:s del kan dessa sammanfattas i följande bild.



För FM:s del innebär[9] detta krav på, att

[1] Säkerhetskrav specificeras i TTEM av HKV,

[4] Säkerhetsgenomgångar, SSPR, genomförs under medverkan av bl a FM,

[9] Felrapporteringssystem, FRACAS, installeras, hanteras och uppföljs av bl a HKV,

[16] Användarmanualer och utbildning, TSR, framställs av HKV,

[19] Beslut om användning, SR, tas fram av HKV (baserat på leverantörens säkerhetsutlåtande och FSC/FMV:s säkerhetsgodkännande).

9. Hakparanteserna åsyftar motsvarande numrering under avsnitt 6.9.3.8.

Utöver detta tillkommer, att

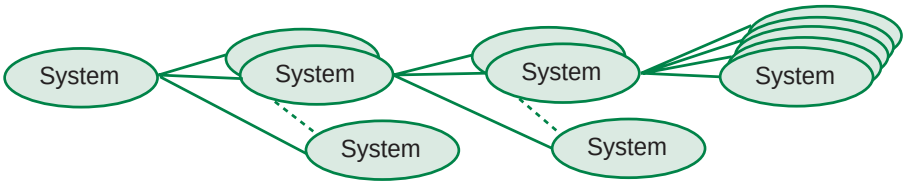
1. System som kan förväntas innehålla säkerhetskritisk programvara skall anskaffas via uppdrag till FMV[10] {HML}.
2. Regler för handhavande, vidmakthållande/underhåll skall specificeras, där det framgår {HML}
   a) vem, som äger produkten (dvs är ansvarig för innehåll och dess aktualitet)
   b) vem, som skall förvalta produkten (dvs utföra uppdateringar)
   c) vilka inskränkningar och krav som gäller för personal, som hanterar, vidmakthåller eller underhåller säkerhetskritiska delar i fred/kris/krig.

   Med produkt avses här maskin- och programvara (inkl fasta program) samt dokumentation på samtliga nivåer (avseende krav, systemsäkerhetsanalyser, konstruktion, test, drift, underhåll) .

2.3 Materielprocessen

Materielprocessen är den övergripande huvudprocessen vid anskaffningar inom FMV (jfr 3.3., <H MatProc>). Under dess inledande studiefaser tas olika styrande målsättningar fram, bland dessa FM:s Målsättningsföreskrifter, TOEM, TEMU samt TTEM[11].

Väsentligt är att dessa dokument ger en helhetsbild av i vilka sammanhang systemet under anskaffning skall användas och vilka uppgifter det har att utföra, för att överordnade behov på högsta systemnivå skall kunna uppfyllas.



10. Anskaffning av säkerhetskritisk programvara ställer större krav på produkt, process och involverade parter. Ett ensat, strukturerat förfarande med ett antal säkerhetsbefrämjande aktiviteter och tekniker under anskaffningens faser är nödvändig, för att minska systemets säkerhetsrisker. Detta kräver insikt i vilka system- och programvarutekniska aspekter, som kan vara relavanta - en kunskap, som inte ryms inom FM:s åtaganden. Anskaffning av denna typ av programvara kan alltså ej utföras enbart med FM och leverantörer som parter.
11. Föreskrifter för dessa dokument finns i <HMÅL>.

En systemarkitektur för totalsystemet på högsta nivå är nödvändig, för att ingående separata system skall kunna fogas ihop till en samverkande enhet. Arkitekturen beskriver ingående delar, deras gränsytor och interaktioner samt formulerar enhetliga principer för hur gemensamma situationer och systemövergripande uppgifter skall hanteras. Bl a fastläggs de övergripande principer, som skall gälla, för att ingående system på viss nivå skall kunna bidraga till den totala systemsäkerheten samt med vilka medel detta skall åstadkommas[12]. Den övergripande arkitekturen styr arkitekturerna för underliggande system (där programvara kan komma in på alla nivåer) samt motsvarande säkerhetsvyer för dessa (4.5.2.2., 6.6).

2.4 Produkter

Detta avsnitt behandlar krav på de produkter FM skall leverera. En av de viktigare inför FMV:s anskaffningsuppdrag är TTEM, som FM utarbetar i samverkan med FMV i successivt mer detaljerade versioner (UTTEM, PTTEM, STTEM).

2.4.1 TTEM, TEMU

TTEMs taktiskt, tekniskt och ekonomiskt inriktade krav och TEMUs taktiskt, ekonomiskt inriktade specificeringar av utbildningsmateriel skall vid anskaffning av system med säkerhetskritiska programvarudelar kompletteras med följande typ av specificeringar.

1. Säkerhetsverksamheterna skall integreras med den framtagningsprocess och utvecklingsmiljö, som används för säkerhetskritiska delar {HML}.

12. Principer: Fastlägger t ex om ingående system på nivå två skall bevaka systemsäkerheten isolerat/oberoende eller samordnat.
Medel: De säkerhetsfunktioner och -mekanismer, som tillsammans definierar arkitekturens systemsäkerhetsvy.

2. Styrande för säkerhetsarbetet skall vara tidigare specificerade uppgifter[13] om anskaffningssystemets omgivning, operationella profil[14], tolerabla risker i fred, kris resp krig för ett exemplar av systemet[15] under hela dess driftstid[16] {HML}.
3. Säkerhetskrav specifika för aktuellt system skall detaljeras m a p funktionalitet, tid, prestanda och kravverifieringar {HML}.
4. Oavsett om realisering sker i maskin- eller programvara skall del kritisk för systemsäkerheten ha ett deterministiskt[17] beteende {HML}.
5. Ett felrapporteringssystem med säkerhetsinriktad information skall finnas från projektstart till systemets avveckling. Om någon del av systemet klassats som säkerhetskritisk skall felrapporteringen avse all programvara, oavsett om denna påverkar systemsäkerheten eller ej {HML}. (Se 6.9.3.8. [9] FRACAS).

13. Dessa uppgifter besvarar frågor av typ: På vilken nivå i systemhierarkien ligger systemet under anskaffning och i vilken omgivning skall det verka? Vilka är de överliggande systemen? Vilka systemsäkerhetsarkitekturer har definierats för övergripande nivåer och hur har toleransnivån fördelats ned till aktuellt system?
Svaren ligger till grund för senare bestämning av vilka kritikalitetsnivåer som skall åsättas det aktuella systemet och dess ingående delar, för att det totala systemet skall hålla sig inom satta toleransnivåer. Kritikalitetsnivån för enskild programvarudel kommer att vara styrande för vilka säkerhetskrav, som skall ställas på personal, processer och produktdelar.
14. Exempel: Systemets användningsområden och totala livslängd, antal instanser/kopior av systemet i drift, antal timmar i kontinuerlig drift eller per uppdrag.
15. Alternativt: ... för samtliga instanser av systemet under hela den sammanlagda drifttiden
16. Detta kan innebära behov av en riskmatris per läge (fred, kris, krig).
17. I varje situation endast **en** tillståndsövergång. För varje möjlig *input* eller vid *timeout* endast **ett** beteende.

3. Beställare (FMV)

Detta avsnitt innehåller generella säkerhetskrav på personal och teknik (metoder & verktyg) inom FMV, främst vid anskaffning, uppföljningar av leverantörens arbete (analys, konstruktion, test, leverans, installation) samt inför vidmakthållandefasen.

3.1 Personalkvalifikationer

Programvarusystem med säkerhetskritiska inslag kräver en samlad insats från samtliga parter involverade i dess realisering[18]. Detta inkluderar projektledare på alla nivåer, program- och maskinvarukonstruktörer, säkerhetsanalytiker, kvalitetssäkringspersonal och operativ personal. De som arbetar med programvarusäkerhet skall även kunna samverka med personer inom andra discipliner, t ex experter på tillförlitlighet och IT-säkerhet, verifiering och validering samt beteendeforskare.

1. Beställare skall ha goda kunskaper och erfarenhet av aktuellt applikationsområde, programvarumetodik och -teknik samt systemsäkerhet {HML}.

3.2 Styrprocesser

Detta avsnitt innehåller krav på stödjande aktiviteter inför realisering och drift av ett säkerhetskritiskt, programvarubaserat materielsystem. Dessa aktiviteter är aktuella under systemets hela livstid. Kraven berör främst projektledare och kvalitetssäkrare.

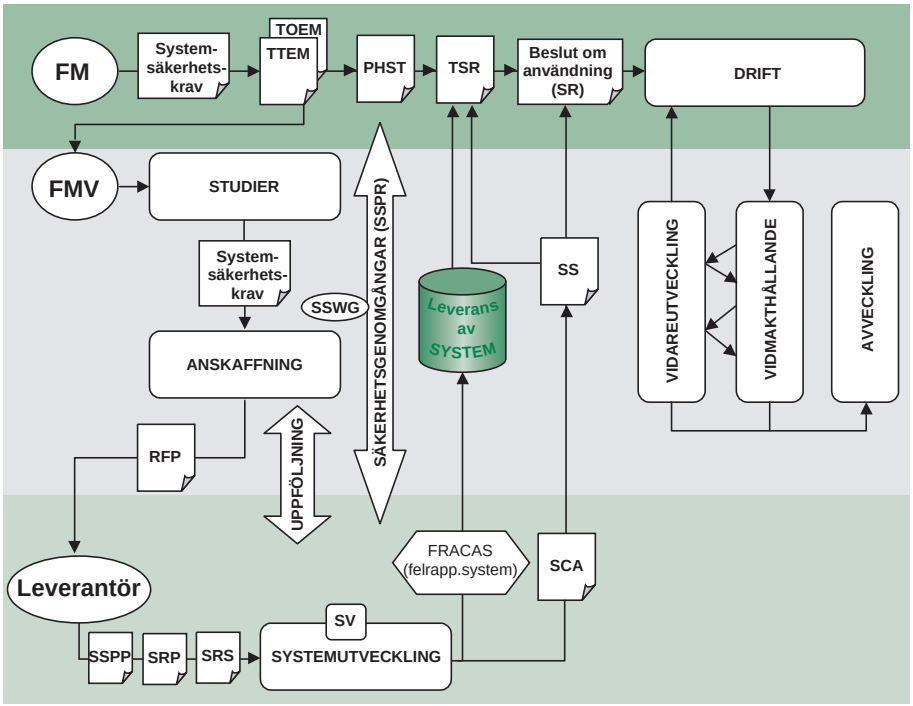
3.2.1 Projektplanering, -ledning och -uppföljning

1. Grundkrav för projektplanering enligt kap 5. skall vara uppfyllda {HML}.

18. Kompetens och etiska regler för de, som arbetar med säkerhetsrelaterade system, finns formulerade, se t ex <IEE>.

3.2.2 Systemsäkerhetsplanering, -ledning och -uppföljning

H SystSäk beskriver systemsäkerhetsverksamheten (se 6.9.3.8.) under ett systems livstid för olika aktörer (**FM, FMV, Leverantör**). För FMV:s del kan dessa sammanfattas i följande bild.



Programvarans roll i systemsäkerheten preciseras genom följande krav:

1. Programvarusäkerhetsfrågor skall ingå bland arbetsgrupp Systemsäkerhets (SSWG) uppgifter {HML}.
2. Leverantörens Programvarusäkerhetsplan[19] skall granskas {HML}.
3. Programvarans systemsäkerhetsanalyser (enligt 4.3.3.) och föreslagna riskreducerande åtgärder (4.5.2.4.) skall följas upp[20] {HML}.
4. Krav, konstruktion, realisering, återanvändning, test och dokumentation av programvara skall följas upp m a p systemsäkerheten {HML}.
5. Felstatistik och uppgifter ur leverantörens felrapporteringssystem för programvara skall bevakas m a p systemsäkerheten {HML}.

19. Se 3.4.1 ang krav på en systemsäkerhetsplan m a p programvara.
20. Exempel på checklistor: Se bilagor.

3.2.3 Kvalitetsstyrning

1. Grundkrav för kvalitetsstyrning enligt kap 5. skall vara uppfyllda {HML}.

3.2.4 Kvalitetssäkring

Projekt med kritisk programvara erfordrar stringenta insatser för att säkerställa kvaliteten i såväl utvecklingsprocess som produkt.

1. Grundkrav för kvalitetssäkring enligt kap 5. skall vara uppfyllda {HML}.
2. Planeringen för FMV skall omfatta resurser och tider för {HML}
   a) kravställning
   b) utvärdering av anbud
   c) eventuell granskning av anbudsgivare (QA audit)
   d) uppföljning av utveckling, såsom deltagande i granskningar (*reviews*)[21]
   e) deltagande vid tester, leveranser, etc.

3.3 Materielprocessen

Den övergripande Materielprocessen omfattar ett antal faser, som i varierande omfattning berör säkerhetskritiska, programvarubaserade system.

Väsentligt för samtliga dessa är, att den arkitektur, som byggs upp från översta systemnivå och ned till aktuellt programvarusystem har en väl definierad systemsäkerhetsvy (2.3.), och att denna bevaras intakt vid varje typ av förändringar i programvarusystemet.

3.3.1 Studier

Krav på Leverantörs systemsäkerhetsverksamhet ställs i beställningens studiekontrakt. Leverantören tar fram en studierapport och en säkerhetsvärdering, vilka utgör grund för beställarens studierapport till FM, vilken i sin tur är underlag för FM:s PTTEM.

De krav specifika för kritisk programvara, som kan vara aktuella under en studiefas, är en delmängd av dem, som listats nedan för anskaffning.

3.3.2 Anskaffning

Under denna fas är målsättningen att få fram rätt kravspecifikation[22]. Detta inbegriper formulering av systemsäkerhetskrav avseende systemet under an-

21. Exempel på granskningsverksamheter m m: Se bl a <TjF-FMV 1997:49>.
22. Se fotnot under avsnitt 2.1: Drygt 44% av inträffade incidenter i programvarusystem har visats bero på fel i kravspecifikationerna.

skaffning, personer involverade i dess framställning samt produktionsprocess och -miljö. Se vidare 3.4.

1. Grundkrav för anskaffning enligt kap 5. skall vara uppfyllda {HML}.

3.3.3 Drift och Vidmakthållande

Efter installation, driftsättning och Beslut om användning (SR, se 6.9.3.8.) tas systemet i drift ute på förband[23].

Under aktuell fas gäller, att bibehålla den säkerhet, som byggts in i systemet. Fel, olyckor eller tillbud[24] rapporteras och utreds. Lämpliga åtgärder vidtas.

1. Grundkrav för Drift och Vidmakthållande enligt kap 5. skall vara uppfyllda {HML}.

3.3.3.1 Ändringar av färdigt system

Olika typer av ändringar och vidareutvecklingar kan bli aktuella efter driftsättning och är i vissa fall en kontinuerlig verksamhet under driftfasen. Det har konstaterats, att renovering, modifiering eller teknisk anpassning är, näst fel i kravspecifikation, den vanligaste incidentorsaken[25].

Förändringar i ett system och dess operationella omgivning kan leda till problem, varför konsekvenserna m a p säkerheten måste analyseras på nytt, innan en implementering av förändringarna inleds. Vilka effekter får t ex ändringar i programvarusystemet på den övergripande säkerhetsarkitekturen?

Förnyad systemsäkerhetsanalys är även nödvändig för ett system, som inte modifieras, t ex om det sätt på vilket systemet används eller dess driftsförutsättningarna eller driftsmiljö förändrats. Framförallt gäller det att vara observant på om vissa antaganden (t ex de som legat till grund för beräkning av tolerabel risk) kan ha blivit inaktuella. Detta kan bl a inträffa vid förlängning av systemets livstid, beslut om ökat antal exemplar av systemet i drift, användning av ett system för kontinuerlig drift utöver avsedd maxtid[26], byte av kringutrustning etc.

Även prestandaförbättringar, som driver systemet/programvaran mot dess dimensioneringsgränser, utbyte av COTS eller ny version av delar som kommunicerar med COTS (se 4.5.1.), är riskkällor, som fordrar förnyad analys.

Problem kan också uppstå efter normal översyn (som ej behöver innebära uppdateringar av programvaran), t ex där vissa mekanismer kan ha satts om

23. Analys av inträffade incidenter har visat, att endast 6% av dessa kunnat hänföras till detta skede (*eng installation and commissioning*), som därmed svarar för det minsta incidentbidraget.
24. Analys av inträffade incidenter har visat att <15% av dessa beror på fel introducerade under detta skede.
25. Analys av inträffade incidenter har visat att >20% av dessa beror på fel introducerade under detta skede.
26. Exempel: Patroit-missilen i Dharan, se 6.9.5, <Patriot>, <FMV_2>.

för hand, utan att styrande programvara vid återstart uppdateras (jämför krav vid start och initiering, 4.5.2.8.). Genom att analysera procedurerna vid översyn och underhåll av systemet kan nya säkerhetskrav tillkomma inför konstruktion av t ex programvaran, så att diskrepans mellan dess modell av systemets status och den verkliga ej uppstår.

Vissa krav under 4.5.2.12. kan även vara aktuella under vidmakthållandefasen.

1. Den kompetens och utbildning, som krävs, för att få titta på resp gå in och ändra i kritiska delar skall specificeras {HML}.
2. Endast därför utsedd personal skall kunna skapa, ändra eller extrahera kritiska delar {HML}.
3. Förnyad systemsäkerhetsanalys skall utföras, som visar ändringars verkan på kritiska delar. Speciell uppmärksamhet skall därvid riktas mot systemets dimensioneringar och driftsvillkor samt ändringar som gäller eller berör COTS-produkter {HML}.
4. Efter underhåll skall säkerställas, att inga skillnader mellan programvarans modell av systemets processtatus och systemets verkliga status föreligger[27] {HML}.

Se även bilaga 6.5. Checklistor.

Krav på den dokumentation, som skall användas under denna fas, har specificerats i 4.5.2.13.3.

3.3.4 Avveckling

Redan under systemets anskaffning ställs krav för att möjliggöra säker avveckling samt underlätta återanvändning av ingående delar. Under själva avvecklingsfasen utför beställaren en riskanalys baserad på underlag från systemutvecklingens systemsäkerhetsanalyser och uppgifter från drifttiden. Inga systemsäkerhetsfrågor speciella för avveckling av kritisk programvara har identifierats. Om avvecklingen är partiell, se avsnitt 3.3.3.1.

3.4 Produkt

Utgående från FM:s kravdokument (se 2.4.) färdigställer FMV vid anbudsinfordran en Anbudsinfordranspecifikation (AIS), vilken omfattar dels krav på leverantörens åtaganden, tids- och verksamhetsplaner samt eventuellt driftstöd, dels tekniska krav på produkten och de stödresurser denna erfordrar. Kra-

27. Exempel på säkerhetsproblem: Ett reglage omsatt för hand inför underhåll, där systemet vid återstart ej uppdaterats ang verkligt status.

ven fördelas på de fyra dokumenten Verksamhetsåtagande, Tidplaner, Vidmakthållandestöd och Teknisk Specifikation (se 6.9.4.4.). I samband med detta utförs en anpassning av kraven i denna handbok till aktuellt anskaffningsärende.

Nedan har enbart aspekter, som berör programvarusäkerhet, behandlats.

3.4.1 Verksamhetsåtagande (VÅ)

VÅ beskriver FMV:s krav på processer och personal vid framställning av aktuell produkt samt åtaganden i samband med leverans och i tillämpliga fall även under avtalad garantitid.

VÅ utgör grund för anbudgivares offertarbete och -kalkyler.

I VÅ skall ingå krav på:

1. Leverantörens personalkvalifikationer {HML}. (Se 4.1.)
2. Hur oberoende granskare skall utses[28] {H}. (Se 4.1.)
3. Leverantörens process {HML}. (Se 4.2.-4.3.)
4. Leverantörens produktionsmiljö {HM}. (Se 4.4.).
5. Godkännanderutiner för programutvecklingsmiljön {H}.

Utöver H SystSäk:s systemsäkerhetsaktiviteter skall krav ställas på:

6. En Programvarusäkerhetsplan (del av SSPP) skall framställas {HML}[29]. Ur denna skall framgå vilka relationer som föreligger mellan systemsäkerhetsanalyser för system och för programvara samt för programutvecklingsarbetet. Speciell uppmärksamhet skall riktas mot hur säkerhetskrav genereras, implementeras, spåras och verifieras.
7. Ett felrapporteringssystem för programvara skall upprättas om någon del i systemet klassats som säkerhetskritisk. För kritiska programvarudelar skall även uppgifter om konsekvens för person, materiel och yttre miljö lagras {HML}. (Se 6.9.3.8. [9] FRACAS).

3.4.2 Tidplaner (Verksamhetsplaner) (TP)

TP anger generella krav på tidplaner, när de skall levereras och vem, som skall godkänna dessa. Den ger även exempel på planer, som kan förekomma vid anskaffning.

Krav på **vilka** av dessa, som skall ingå i aktuell upphandling samt detaljerade krav för dessa, anges däremot i VÅ.

28. Jämför IEC 61508, del 1, tabell 4-5 (oberoende m a p person, avdelning, organisation).
29. Denna plan bör ingå i SSPP (för bibehållet fokus på **system**säkerheten), d v s ej utgöra ett separat dokument.

3.4.3 Vidmakthållandestöd (VS)

VS tas fram då leverantörsåtaganden under vidmakthållandefasen är aktuella. Dessa kan avse materiella och personella resurser samt åtaganden fram till dess stödet levererats. Är behovet omfattande kan i stället separata TS och VÅ upprättas.

I VS skall ingå krav på:

1. Utrustning eller hjälpmedel för systemsäker drift och vidmakthållande inkluderande {HML}
   a) Leverantörens system för rapportering av fel / problem / förbättringar. Detta kan med fördel vara samma som det under programproduktens utveckling (se 4.4.1.2.).
   b) Leverantörens konfigurationshanteringssystem.
2. Stöd för analys och åtgärder vid identifierade systemsäkerhetsproblem {H}.

3.4.4 Teknisk Specifikation (TS)

TS beskriver FMV:s krav på aktuell produkt för tiden fram t o m leverans och vidmakthållandefas. Kraven avser funktionalitet, prestanda, tekniska egenskaper samt erforderliga stödresurser för drift. TS utgör grund för anbudgivares offertarbete och -kalkyler.

En TS skall i kapitel 6 ange systemsäkerhetskrav (avsnitt 6.3 inkluderar bl a programvarukrav). Utbildningskrav specificeras under kapitel 10.

Säkerhetskrav, att ställa vid specificering av såväl ny- som färdigutvecklade programvaruprodukter, har listats nedan. Dessa krav skall balanseras mot krav på funktionalitet, dvs en begärd funktionalitet skall uppfyllas[30] även om den innebär vissa risker.

Kraven omfattar både programvara och dokumentation på alla nivåer inklusive systemsäkerhetsanalyser, testprogramvara (dvs test*drivers*, testdata och testutfall), testspecifikationer och -protokoll samt utbildnings- och användardokumentation för olika yrkeskategorier.

Vid analys av programvarubaserade system som havererat har man funnit, att grundorsaken ofta inte ligger i själva programvaran utan i en ofullständig specificering av system, operationell omgivning eller driftsvillkor. Att i ett tidigt skede detaljspecificera ett systems agerande i varje tänkbar situation är inte möjligt, såvida det inte är frågan om ett litet eller till stora delar känt system. Specificerade säkerhetskrav måste därför förfinas allt eftersom systemets konstruktion tar form och det står klart, vilka delar, som har inverkan på systemsäkerheten. Olika systemsäkerhetsanalyser nyttjas därvid, för att finna

30. Inom formell metodik talar man om krav både m a p *safety* **och** *liveness*, d v s säkerhetskraven får ej innebära att planet ej tillåts lyfta.

beteenden, som måste förhindras för att inte systemsäkerheten skall åsidosättas. Förutom krav på vad det aktuella systemet skall utföra tillkommer p s s detaljkrav på vad systemet och ingående delar **inte** får göra. Dessa detaljkrav kommer ofta in som konstruktionskrav eller konstruktionsrestriktioner på lägre nivåer[31]. De utgör en delmängd av programvarans systemspecifika säkerhetskrav och ingår inte i denna handboks kravsatser (jfr 6.4.1.).

1. Generella säkerhetskrav på upphandlad programvaruprodukt skall ingå (se 4.5.) {HML}.
2. Säkerhetskrav [32] specifika för upphandlad programvaruprodukt skall ställas {HML}.
3. För de delområden, som låter sig beskrivas formellt, skall krav på användning av formella metoder (se 4.3.2.1.) ställas {H}.
4. Systemuppgifternas inbördes prioritet skall fastläggas {HML}.
5. Acceptabel degraderingsgrad skall definieras för varje övergripande systemfunktion[33] {HM}.
6. Kända, framtida ändringar skall anges, så att hänsyn kan tas till dessa vid analys och konstruktion av säkerhetskritiska delar (se även 3.3.3.1.) {HML}.

Nya beställningar kommer, av ekonomiska och tidsmässiga skäl, att innehålla ett allt större inslag av tidigare utvecklade komponenter. Även om de barnsjukdomar, som nyutvecklade delar drabbas av, därmed kan undvikas, är återanvändning inte problemfri, i synnerhet ej i samband med kritiska system. Några av de problem, som kan uppstå, har listats under 6.6.5.

De krav som skall ställas vid återanvändning av programvara i kritiska delar finns under avsnitt 4.5.1. Se för övrigt bilaga 6.5. Checklistor.

31. Exempel: Riskkällor: Bussdörr slår igen vid på-/avstigning. Buss i rörelse med öppen dörr. Konstruktionskrav: Hinder i dörröppning inhiberar stängning. Buss ej körbar med ostängd dörr (jfr krav under 4.5.2.4: I kritisk del ingen funktionalitet utöver den krävda).
32. Exempel: Prestandakrav för funktioner. Övre gräns för det totala minnesutnyttjandet. Antal timmar systemet skall kunna vara i kontinuerlig drift utan omstart. Krav på att undvika riskkällor och olyckor specifika för applikationen, t ex genom krav på speciella funktioner för övervakning och skydd (se 6.7).
33. Vilka uppgifter/vilken funktionalitet skall prioriteras högst? Vilka alternativ skall finnas vid nödläge? När är det acceptabelt med *fail silent, fail safe, fail soft* (se 6.1)? Var är det rimligt att kräva självrättande kod eller möjlighet till omkonfigurering under drift till andra, intakta maskinvaruenheter?

4. Leverantör

Detta avsnitt innehåller generella säkerhetskrav på leverantör av produkt med kritiska programvarudelar. Personal, process samt nyutvecklad och återanvänd produkt behandlas.

Det bör noteras, att leverantör, som uppfyller krav i detta kapitel ej därmed befrias från ansvar för konsekvenser av programvarans och systemets beteende i säkerhetskritiska situationer.

4.1 Personalkvalifikationer

Programvarusystem med säkerhetskritiska inslag kräver en samlad insats från samtliga parter involverade i dess realisering[34]. Detta inkluderar projektledare på alla nivåer, program- och maskinvarukonstruktörer, säkerhetsanalytiker, kvalitetssäkringspersonal och operativ personal. De som arbetar med programvarusäkerhet skall även kunna samverka med personer inom andra discipliner, t ex experter på tillförlitlighet och IT-säkerhet, verifering och validering samt beteendeforskare.

1. Personalen skall bestå av programvaruutvecklare med god kännedom om etablerad utvecklingsteknik (metoder, verktyg, programmeringsspråk), tillämpliga standarder och systemets applikationsområde. Detta gäller särskilt system- och programvaruarkitekter samt konstruktörer {HML}.
2. System- och programvaruarkitekter, konstruktörer samt utvecklare av kritiska delar skall ha god kunskap om system- och programvaruteknik för konstruktion av säkra system {HML}. Dessa personer skall
   a) ha kunskap och erfarenhet av teknik för konstruktion och utveckling, verifiering och validering samt säkerhetsanalys av säkerhetskritiska system med tyngdpunkt på programvaran i denna typ av system,
   b) förstå principerna för ovanstående teknik, kunna avgöra hur dessa skall tillämpas för att åstadkomma säkra konstruktionslösningar samt inse effekterna på systemet i avsedd driftmiljö och driftsituation.
3. Minst två personer skall för varje kritisk programvarukomponent ha grundlig kännedom om dess konstruktion, implementation, test och operation {H}.
4. Minst två personer skall för varje kritisk programvarukomponent ha kännedom om dess konstruktion, implementation, test och operation {M}.

34. Kompetens och etiska regler för de, som arbetar med säkerhetsrelaterade system, finns formulerade, se t ex <IEE>.

5. Integratör skall ha god kännedom om systemets olika delar samt veta, vilka delar, som är kritiska för systemsäkerheten {HM}.
6. En ansvarig person skall finnas för varje tilldelad uppgift i produktionsprocessen {HML}.
7. En person med ansvar för programvarusäkerhet i projektet skall vara utsedd {HML}.
8. Personer som utvärderar eller verifierar en produkt skall vara oberoende av dem, som deltagit i dess produktion {HML}. Graden av oberoende [35] beror av produktens kritikalitet:
   a) Granskare skall tillhöra annat företag eller organisation {H}
   b) Granskare skall ej väljas bland konstruktörer i det aktuella projektet {M}
   c) Granskare skall vara annan person än den, som utvecklat delen {L}.
   d) Testgrupp för säkerhetstest skall ha minst en person, som ej deltagit i utvecklingen av systemet {HML}

4.2 Styrprocesser

Detta avsnitt innehåller krav på stödjande aktiviteter vid utveckling av ett säkerhetskritiskt, programvarubaserat materielsystem. Dessa aktiviteter är aktuella under programvarusystemets hela livscykel.

Kraven berör främst projektledare och kvalitetssäkrare.

4.2.1 Projektplanering, -ledning och -uppföljning

1. Grundkrav för projektplanering enligt kap 5. skall vara uppfyllda {HML}.
2. För varje roll beskriven under 4.1. skall namn, kunskap och erfarenhet på utsedd person anges, innan resp aktivitet påbörjas {HML}.

Vid successiva systemutgåvor bör planeringen inriktas mot att i första hand inkludera de väsentligaste delarna och kommunikationsvägarna (med begränsad men stabil funktionalitet tillräcklig för att kunna nyttja in- och utmatningsdelar, databaser etc.) för att senare bygga på med allt mer komplett funktionalitet (*back-bone*-principen).

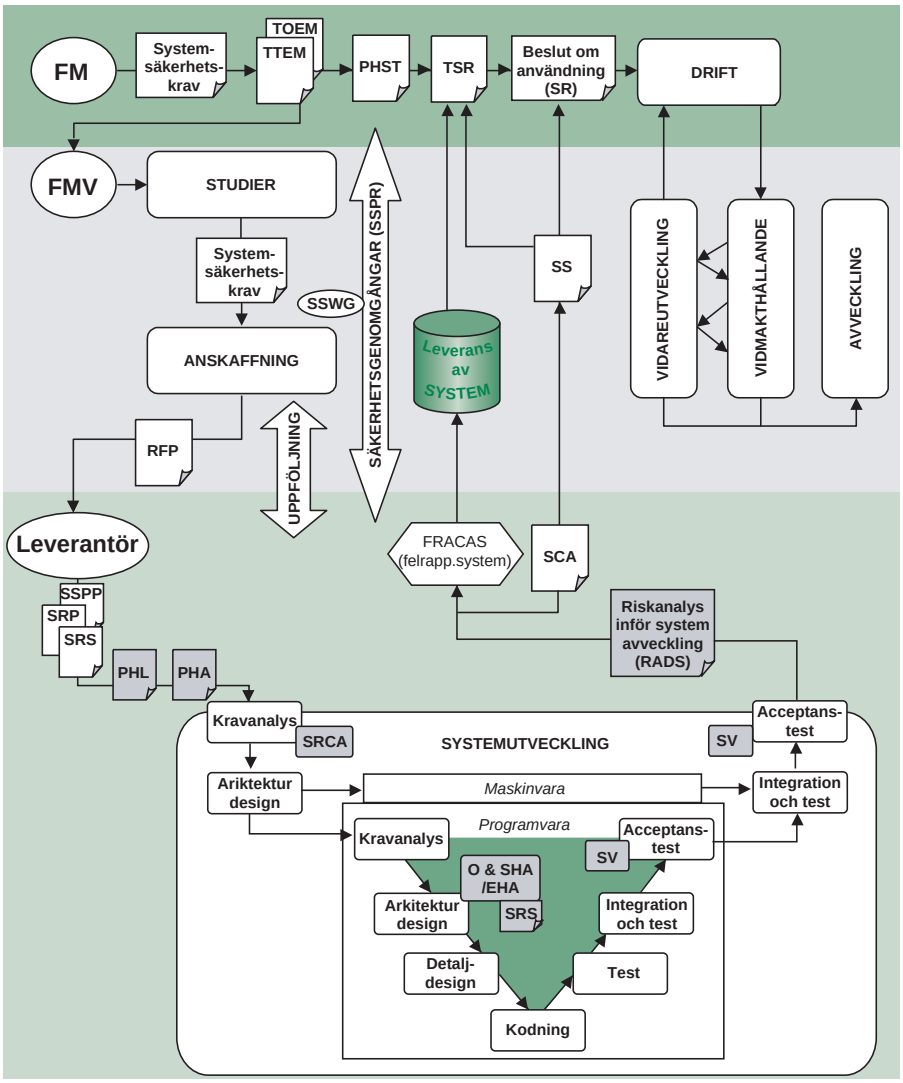
För kritiska programvarusystem bör realisering av kritiska delar planeras in före delar av lägre eller ingen kritikalitet. Detta ger utrymme för upprepade systemsäkerhetsanalyser och omkonstruktion i fall där kritikaliteten ligger över toleransgränsen.

35. Se även 3.4.1. Jämför IEC 61508, del 1, tabell 4-5 (oberoende m a p person, avdelning, organisation).

4.2.2 Systemsäkerhetsplanering, -ledning och -uppföljning

Allmänna krav på Leverantörs systemsäkerhetsarbete enligt H SystSäk (se 6.9.3.8.) kan sammanfattas m h a den klassiska V-modellen, där aktiviteter i den vänstra skänkeln inriktas mot att bygga in säkerhet och planera för hur denna skall valideras medan de i den högra syftar till att successivt verifiera säkerhetskravens realisering.

Bilden nedan är starkt förenklad och skall inte tolkas som att en viss utvecklingsmodell förordras (t ex vattenfall, se 6.6.1.). Större system tas i regel fram iterativt och i successiva kundutgåvor (se 4.2.1.). V-modellen upprepas för varje ny systemutgåva och utvecklingsmodellen skulle därmed kunna betraktas som evolutionär eller spiralformad, snarare än linjär eller sekvensiell, <Sommerville>.



Utöver H SystSäks krav tillkommer, att

1. Programvarans säkerhetsarbete skall planeras och dokumenteras i en Programvarusäkerhetsplan (se 3.2.2.).

4.2.3 Kvalitetsstyrning

1. Grundkrav för kvalitetsstyrning enligt kap 5. skall vara uppfyllda {HML}.

4.2.4 Kvalitetssäkring

1. Grundkrav för kvalitetssäkring enligt kap 5. skall vara uppfyllda {HML}.
2. Granskningar och genomgångar samt testplaner och testprocedurer skall inkludera aspekter på programvarusäkerhet {HML}.

4.2.5 Konfigurationsstyrning

1. Grundkrav för konfigurationsstyrning enligt kap 5. skall gälla {HML}
   a) all programvara,
   b) de verktyg, som använts för att utveckla, producera, styra eller verifiera programvaran
   c) all dokumentation inklusive den över olika analys- och verifieringsresultat[36].

4.3 Produktionsprocess

Detta avsnitt innehåller krav på aktiviteter vid utveckling av ett kritiskt, programvarubaserat system. Kraven berör främst konstruktörer och utvecklare.

En inledande diskussion angående olika modeller i produktionsprocessen finns under 6.6.1.

1. Grundkrav för produktionsprocess enligt kap 5. skall vara uppfyllda {HML}.
2. Systemsäkerhetsanalys skall utföras på produktionsprocessen, för att identifiera de risker processen kan medföra {H}.
3. Analysresultaten skall dokumenteras {H}.
4. Kompletterande säkerhetsåtgärder skall införas i produktionsprocessen för reduktion av identifierade säkerhetsrisker i processen till tolerabel nivå {H}.

4.3.1 Utvecklingsmodell

Detta avsnitt inkluderar krav på den livscykelmodell, som används vid systemutvecklingen. Fokus ligger på de programvaruinriktade faserna och närmast omgivande skeden.

1. Grundkrav för utvecklingsmodell enligt kap 5. skall vara uppfyllda {HML}.

4.3.2 Utvecklingsmetodik

Detta avsnitt inkluderar krav på de metoder, som används vid analys, konstruktion, test och integration av programvara.

1. Grundkrav för utvecklingsmetodik enligt kap 5. skall vara uppfyllda {HML}.

Principen felfri konstruktion[37] inriktar sig mot att fånga fel redan under konstruktionsfasen. Syftet är att minska de risker och kostnader, som sena felupptäckter, rättelser och omtest medför. Formella metoder och i någon mån vissa strukturella metoder[38] bygger på principen felfri konstruktion.

36. Exempel: Formella, statiska, dynamiska, säkerhetsinriktade.
37. Exempel: *Cleanroom* baserad på bl a formella metoder och statistiska systemtest utförda av en oberoende testgrupp, <Mills_2>, www.dacs.dtic.mil/.
38. Exempel: Semiformella (eller formaliserade) metoder med verktyg, som helt eller delvis kan generera kod resp konstruktionselement ur resultat från föregående fas (konstruktion resp analys). Detta förutsätter dock verifierade genereringsverktyg. Se 4.4.2.

4.3.2.1 *Formella metoder*

Formella metoder är ett sammanfattande begrepp för en mängd tekniker[39], som använder sig av formell logik och diskret matematik för specificering och verifiering av krav och motsvarande implementation. Ett formellt system definieras m h a en uppsättning primitiver, axiom (givna påståenden) och inferensregler (satser, teorem). Formalismen kan drivas olika långt och avse:

– enbart specificering (eventuellt kompletterad med manuell verifiering)

– både specificering och verifiering (med helt eller delvis automatiserad kontroll eller genom bevisgenerering).

I sin mest stringenta form kan formella metoder användas inte bara för att verifiera att en realisering uppfyller sina specifikationer utan också för att påvisa frånvaron av vissa fel[40] och för att identifiera implicita, odokumenterade antaganden i system och omgivning. Tekniken kan däremot inte avgöra om andra funktioner (eller beteenden) utförs än de som specificerats eller om själva den formella specifikationen är korrekt[41]. Likaså saknas möjlighet att uttrycka och föra i bevis vissa realtidskrav (prestanda). Detta kräver kompletteringar med annan teknik (prov, test och granskningar).

De formella metoderna kan indelas i två huvudklasser: de baserade på formell bevisföring (*Theorem Provers*) samt de som utför någon typ av modellkontroll (*Model Checkers*). Den första kategorin har bättre möjligheter att klara något större system, men dess verktyg är inte helt automatiska och förutsätter i högre grad användare med formell skolning. Detta kan vara en anledning till att den senare kategorin vunnit större acceptans inom industrin, trots att den är mer resurskrävande (minne och tid) och därmed svår att tillämpa på storskaliga system. I båda fallen krävs förmåga att kunna översätta och abstrahera aktuell systembeskrivning (t ex en informell kravtext eller en semiformell konstruktion i UML) till den representation den formella metoden och verktyget fordrar. Integrering av formell teknik med den utvecklingsmetodik som används vid industriell programvaruproduktion idag skulle t ex kunna bestå i, att de olika vyer CASE-verktygen tillhandahåller för att beskriva systemets statiska och dynamiska egenskaper kompletteras med formella vyer för delar, som låter sig beskrivas formellt[42]. Utvecklingen går mot detta håll -verktyg som konverterar mellan flera olika formalismer finns redan.

Formell verifiering av en konstruktion och motsvarande formella specifikation kan bli komplicerad. Bevisverktyg är därför av stort värde. Dock kan

39. Se t ex <NASA-1-97>, <FME-Guide>, <FM-State> samt hemsidor under avsnitt 6.9.5.
40. Exempel: Logiska fel, syntaktiska och semantiska inkonsistenser, t ex odeklarerade namn, tvetydigheter.
41. Exempel: Rena felaktigheter, underspecificering (överspecificering däremot medför inkonsistens och kan fångas med formell analys).
42. Formell verifiering är speciellt värdefull för delsystem med extremt höga krav på tillförlitlighet. Skyddssystem (t ex säkerhetsskal och brandväggar, se 6.7) hör till denna kategori. Formell teknik har använts industriellt bl a för nödstängningssystem, automatiserade vågskydd, spärrsystem, antikollisionssystem.

förekomma att ett verktyg (för att förenkla bevisföringen) underkänner fler konstruktioner än vad som är motiverat ur korrekthets- och konsistenssynpunkt. Detta kan vara ett övergående fenomen, som gäller tidiga versioner av verktyget (jfr 4.5.2.6.).

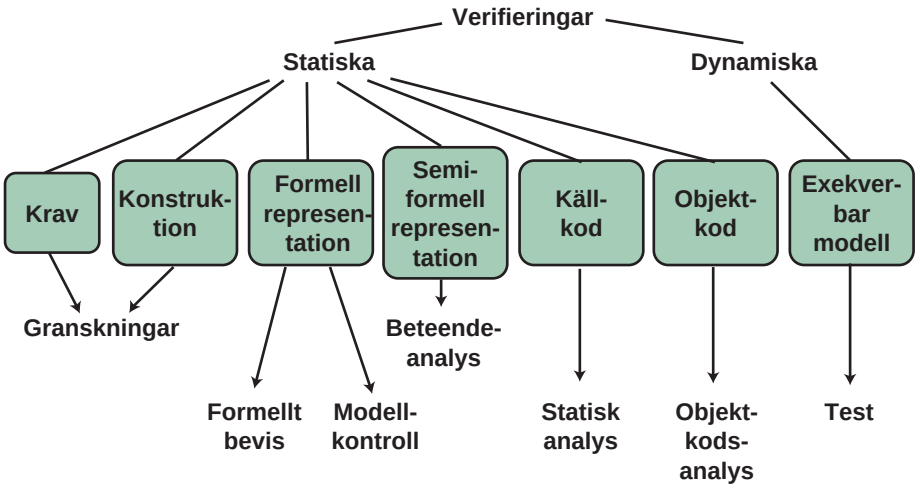
För använda bevisverktyg gäller (liksom för övriga verktyg för produktion och drift av kritiska delar) att dessa visats vara korrekta (se 4.4.2.1.).

1. Formella metoder för specificering och verifiering skall övervägas för delområden, som låter sig beskrivas formellt [43] även om verktygsstöd för verifieringar inte föreligger[44] {HM}.
2. Formell verifiering skall kompletteras med test i målmiljö {HML}.

4.3.2.2. *Verifieringar*

Olika teknik används för verifieringar under utvecklingsprocessen. Vilken man väljer beror på programvarans karaktär, vilket stadium den befinner sig i och vilka aspekter, som skall kontrolleras. En del kan vara mer kostnadskrävande att tillämpa, men nödvändiga om inga alternativ finns (granskning, är t ex den enda tekniken för verifiering av att kravformuleringarna är korrekta). För några finns dock verktygsstöd, som avlastar den manuella insatsen.

Där formell specificering tillämpats, är det mest kostnadseffektivt med formell verifiering (se föregående avsnitt). Där detta ej är möjligt måste annan teknik tillgripas. Det går inte att utse någon enskild teknik, som den entydigt bästa, eftersom ingen ensamt klarar, att verifiera alla aspekter. Detta talar för ett diversifierat angreppssätt, där samtliga tekniker nedan tillämpas.



43. D v s med någon matematisk/logisk notation.
44. Insatsen för den formella specificeringen skall vägas mot de vinster i säkerhet, som kan erhållas genom möjligheten till stringent och automatiserad verifiering av såväl program- som maskinvara (och framförallt kombinationen av dessa).

4.3.2.2.1 Granskningar (manuella verifieringar)

Granskningar utförs av personer oberoende av dem, som utvecklat materialet för granskning. Graden av oberoende är avhängigt materialets kritikalitet (se 3.4.1.).

Granskningen kan vara mer eller mindre formell[45] och vara inriktad på krav, konstruktion, kod, test eller analysresultat.

1. Oberoende granskningar och uppföljningar skall utföras under hela utvecklingsfasen enligt fastställd kvalitetsmanual (se 4.2.4.) {HML}.

4.3.2.2.2 Statisk analys (källkodsverifieringar)

Vid statisk analys kontrolleras ett program mot en modell av programmet härledd från dess källkod, för att bedöma om det är välformulerat[46], följer fastslagna programmeringsregler etc. Statisk analys kan utföras manuellt och m h a verktyg (se 4.4.2.). Tekniken har visat sig vara mycket kostnadseffektiv. Flera typer av metriker kan vara aktuella för mätning av olika programvaruegenskaper.

1. Statisk analys skall utföras på all kritisk källkod {HML}.
2. Analyserna skall omfatta
   a) källkodens kritiska data-, styr- och informationsflöden {HML},
   b) avsteg från fastlagd Kodningsföreskrift (4.5.2.7.) {HML},
   c) avsteg från föreskrivet språksubset {HML},
   d) källkodens komplexitet[47] {HM},
   e) semantisk analys/symbolisk exekvering[48], där så är möjligt {H}.
3. Analysresultaten skall dokumenteras {HML}.

45. T ex baserad på Fagans inspektionsteknik <Fagan> med väldefinierade procedurer för olika roller eller med en mer informell genomgång (*Walkthrough*). Se även <Gilb>,<NASA-2202>, <NASA-A302>. En utveckling mot modellbaserade i stället för dokument- och textbaserade granskningstekniker pågår f n.
46. Av speciellt intresse här är analys av **kritiska** flöden:
Dataflödesanalys verifierar, att programvariabler satts och använts korrekt enl specifikationen (avslöjar t ex använda men oinitialiserade variabler, variabler som satts om utan mellanliggande läsning). Denna analys utförs för starkt typade språk av kompilatorn.
Styrflödesanalys (t ex analys av anropsgraf) kontrollerar, att sekvensialiseringen implementerats enl specificerad konstruktion. Den upptäcker även ur säkerhetssynpunkt olämpliga egenskaper som rekursion, loopar med multipla ingångar, avsnitt utan utgång, död kod samt delade subprogram mellan programvarans säkerhetskritiska delar och övriga delar. Analysen utförs av vissa kompilatorer (Ada95) eller via separata verktyg.
Informationsflödesanalys kombinerar dataflödes- och styrflödesanalys för varje variabel inom en modul eller mellan flera moduler. Kan t ex identifiera beroenden mellan en komponents in- och utvariabler, att dessa är satta och ej har några oväntade beroenden.
47. Avser bedömning av komplexitet dels för subprograms styrflödesgrafer dels för programs anropsgrafer. Analys, som visar på låg komplexitet, är **ett** sätt att visa att kravet på enkelhet enl 4.5.2.4 är uppfylld.
48. Successiva substitutioner, där utvariabler uttrycks m h a invariabler, t ex för kontroll av den matematiska relationen mellan in- och utvariabler, av begränsningar inom språket (t ex indexgränser eller *overflow*).

4.3.2.2.3 Beteendeanalys

Beteendeanalys utgår från en beskrivning/vy/modell över systemets beteende (t ex Petri-nät, Tillståndsmaskiner) för att kontrollera om denna överensstämmer med förväntat beteende. Varje etablerad utvecklingsmetodik tillhandahåller ett antal vyer för beskrivning av både statiska och dynamiska systemegenskaper. Flera av dessa vyer avser systemets dynamiska beteende och kan med fördel användas som underlag vid t ex riskkälleanalys (se 6.8.2.).

4.3.2.2.4 Objektkodsanalys

Analys av objektkod syftar till att verifiera, att kodgenereringssystemet inte introducerat fel vid översättning av källkoden. Genom att välja ett språk, där både krav och mostvarande valideringstester är standardiserade (<Ada 95>, <IEC 18009>) och en kompilator, som passerat testerna med godkänt resultat, underlättas analysarbetet väsentligt. Ett ytterligare plus är om kompilatorn har certifierats m a p någon säkerhetsstandard (se 4.4.2.2.). Inga testsviter kan dock garantera felfrihet. För kritiska applikationer är därför kraven på täckningsgrad hårda och skall vid högsta kritikalitet uppfyllas på objektkodsnivå (se nästa avsnitt). Andra sätt att kontrollera överföringen till objektkod är, att nyttja olika kompilatorer på samma källkod (se 4.5.2.4.5.: Diversifiering).

1. Objektkodsanalys skall ha utförts, som verifierar att källkoden implementeras korrekt på aktuella målmaskiner {H}.
2. Analysresultaten skall dokumenteras {H}.

Verfieringarna kan utföras eller styrkas på flera sätt:

I. Använd kompilator är formellt specificerad och verifierad för aktuell plattform. Dess transformation av källkod till objektkod är matematiskt korrekt.

II. Statisk analys[49] på objektkoden visar, att transformation från motsvarande källkod är korrekt.

III. Verifieringsvillkor (*proof obligations*) definierade m a p icke-korrekt transformation av källkod till objektkod har kunnat förkastas m h a formella argument.

IV. Verifieringar genom test har utförts med godkänt utfall, där

   a) kompileringssystemet uppfyller villkor enligt 4.4.2. och 4.4.2.2.,

   b) objektkod från komplex/svår/ovanlig källkodssyntax har kontrollerats speciellt,

   c) en mappning mellan styrflödet i källkod resp objektkod genererats,

   d) objektkodens slutliga version använts,

   e) full testtäckning på objektkodsnivå uppnåtts (se nästa avsnitt).

49. För kontroll av bl a programvarans styr-, data- och informationsflöde. Se föreg avsnitt.

4.3.2.2.5 Dynamisk analys (verifiering genom test)

Vid dynamisk analys kontrolleras ett program utgående från dess exekvering. Tekniken är ett komplement till formell verifiering, granskning och statisk analys.

Viss analys under exekvering kan utföras av *run-time*systemet (t ex kontroll av variabelvärden m a p definitionsgränserna). Adas exceptionhantering är ett sådant exempel[50].

Test och simulering är till karaktären empiriska verifieringar och kan som sådana ej ge garanti för felfrihet. Om testfallen väljs klokt kan utförda test öka konfidensen att programvaran uppför sig tillförlitligt (dvs enligt specifikationerna).

Dynamisk analys syftar till att kontrollera dels verkan av operationella villkor på programegenskaper som korrekthet, prestanda och resursbehov, dels det skydd systemet kan ge vid säkerhetshotande situationer eller tillstånd.

Programvaran testas på olika nivåer under dess uppbyggnad (enhets-, komponent-, integrations- samt systemnivå) och i en allt mer verklighetsnära testmiljö. Under integration byggs och testas systemet successivt från parametersatta, färdigtestade komponenter. Integrationstesten fokuseras mot den systemövergripande funktionaliteten (dvs interaktionen mellan ingående komponenter och mot omgivningen) med syfte att avslöja avsteg från avsedd funktionalitet.

Vanligast är att starta med de delar, som svarar för de viktigaste kommunikationsvägarna, för att snabbt få till stånd en tillräcklig och stabil funktionalitet och kunna använda sig av in- och utmatning, databaser etc. Detta ger successiva systemutgåvor med allt mer komplett funktionalitet. I planeringen ingår, att kritiska delar produceras i ett tidigt skede (4.2.1.). Detta kan möjliggöra tidig test av kritiska kommunikationsdelar.

Krav vid test av kritiska programvarudelar och kommunikationsvägar:

1. Grundkrav för verifieringar enligt kap 5. skall vara uppfyllda {HML}.
2. Säkerhetstest, som verifierar säkerhetskraven, skall planeras, utföras, granskas samt presenteras under säkerhetsgenomgångar på systemnivå {HML}.
3. Test skall utföras konsekutivt på minst tre nivåer: enhets-/ komponent-, integrations- och systemnivå samt vara formella åtminstone på
   a) systemnivå {HML}
   b) integrationsnivå {HM}
   c) enhets- och komponentnivå {H}.
4. Valda testfall skall verifiera, att {HML}
   a) programvarans samtliga säkerhetskrav är uppfyllda,
   b) programvarusystemet hanterar identifierade risker på ett säkert sätt
   c) programvarusystemet inte initierar säkerhetsrisker.
5. Endast kod, som genomlöpts under test, skall ingå i kritisk programdel[51] {HM}.

50. Innebär ändringar i programmets styrflöde samt tillägg av extrakontroller med ökad tids- och minnesförbrukning som följd. Detta kan komplicera verifieringarna.
51. Innebär att efter bortresning av otestade delar en ny, fryst version skapas för omtest. Se 5.

6. Vid graftest skall testfallen minst en gång exekvera/täcka in objektkodens samtliga noder (och mellanliggande vägavsnitt)[52] {H}.
7. För generisk enhet skall kravet på täckningsgrad enlig föregående krav även uppfyllas för systemets samtliga instansieringar av enheten {H}.
8. Vid graftest av kritisk del skall testfallen genomlöpa såväl typiska vägar under normal uppstart och drift som vägar, vilka leder till felsituationer/feltillstånd, i synnerhet säkerhetskritiska situationer eller tillstånd[53]. Speciell uppmärksamhet skall riktas mot systemets hantering vid flera samtidiga felsituationer {HML}.
9. Integrationstesten skall planeras för tidig test av kritiska kommunikationsvägar och inleds så snart systemets grundläggande funktionalitet byggts upp {HML}.
10. Vid integrationstest skall samtliga delar av de kritiska kommunikationsvägarna ha genomlöpts minst en gång och med tillfredsställande resultat {HML}.
11. Integrationstesten skall visa, att varken begärd kommunikation eller störningar och fel i gränssnitten[54] kan leda till olycka[55] {HML}.
12. Inlagd programvara för provning, som ej krävs för operationell drift, skall rensas bort från kritiska programdelar[56] {HM}.
13. Test skall visa, att funktioner, kommadon eller andra faciliteter endast avsedda för visst driftläge/systemmod inte kan exekveras under annat driftläge/systemmod[57] {HML}.
14. Slutverifiering av säkerhetskrav skall utföras i rätt miljö[58] under riktiga betingelser och med programvarusystemet i slutlig objektkodsversion {HML}.
15. Efter avslutad test skall programvarans modell av systemets processtatus överensstämma med verklig status[59] {HML}.
16. Inga säkerhetskritiska testscenarier skall exekveras på system i skarp driftmiljö[60] {HML}.

52. Syftet är inte bara att kontrollera, hur pass väl testen täckt in objektkoden, utan att finna ev logiska fel, t ex i form av död kod eller onödig, kompilatorinlagd kod (t ex för kontroll av redan tidigare kontrollerade varibelgränser - en form av suboptimering, som resulterar i extra grenar, svåra att täcka in vid test).
Exempel på en testteknik, som kan användas med detta syfte är MC/DC. Se <Leveson_2>, <MCDC>.
Onödig extrakod skall klassas som död eller deaktiverad och motsv åtgärder vidtagas. Se 4.5.2.4.
53. S k robusthetstest. Exempel på hur felsituationer kan simuleras ges under Felinjiceringar (6.8.7).
54. Se t ex 6.8.7 Felinjiceringar.
55. Stora inmatningsfiler, snabbförändrad inmatning, många avbrott kan vara lämpliga testfall.
56. Normalt ingår ej testprogramvara i det operativa systemet. Jämför testdriver (6.1), BIT (4.5.2.10).
57. Jfr 6.9.5: Ariane 5.
58. Även delar verifierade med formella metoder skall testas i målmiljö. Se 4.3.2.1.
59. Innebär t ex att parametrar specialsatta för test skall återställas. Jämför krav under 3.3.3.
60. För starkt komplexa system är det svårt att försäkra sig om, att systemets alla lägen återställts till normal mod. Det finns exempel, där resttillstånd funnits kvar i maskinen efter test, vilket aktiverat alerttillstånd och avfyrningar (se <Leveson> och varningssystemet vid NORAD med två allvarliga tillbud 1962 och 1979).

4.3.2.2.6 Statistisk felanalys – Felprediktering

Statistisk felanalys är en tillförlitlighetsteknik. Data samlas under systemets utveckling och drift, för specificering, prediktering eller skattning av programvarans tillförlitlighet (trendanalys).

Då skattade, återstående fel ligger inom ett förutbestämt acceptansområde anses tillförlitligheten tillräckligt hög[61], för att test- och korrektionsfasen skall kunna avslutas.

Till förutsättningarna hör, att felrapporteringssystemet[62] redan från början av programutvecklingen förses med data för felanalys samt att använd process, utvecklingsmiljö, system och användningprofil hålls oförändrade under den tid data samlas in, för att beräknade modellparametrar fortfarande skall vara giltiga (varken rättad eller ny kod kan m a o tillföras). Vilken typ av data, som skall sparas, beror på vald predikterings- eller estimeringsmodell (i sin tur avhängigt det tidsspann prognosticeringen skall täcka och aktuellt applikationsdomän) [63]. Trimning av vald modell mot aktuellt system och dess förutsättningar kan visa sig nödvändig.

Ett vanligt krav för kritiska system är, att återstående allvarliga felyttringar för systemets kalkylerade livs- eller driftstid skall understiga 1. I regel görs dock predikteringar m a p samtliga felyttringar, oavsett allvarlighetsgrad.

1. Statistisk felanalys skall utföras m a p fel i kritiska delar, där tillförlitlighet är väsentlig för systemsäkerheten[64] {H}.
2. Beslutsunderlag för när testarbetet kan avslutas skall hämtas från den statistiska felanalysen och baseras på att skattade, återstående fel befinner sig på en säkerhetsmässigt tolerabel nivå {H}.

4.3.2.2.7 Resursanalys

Med resursanalys avses här analys av tids- och minnesbehov. Analysen används, för att undersöka nödvändigheten av konstruktionsändringar p g a att resursbehovet i vissa lägen kan komma att överstiga möjlig tilldelning[65].

1. Minnesanalys skall utföras för varje resurs, som delas mellan olika programvarudelar [66] {HML}.

61. Tillförlitlighten **kan** vara hög, trots ett stort antal kvarvarande fel (t ex om dessa ej aktiveras i avsedd användning). Även det omvända kan vara möjlig: en mycket liten andel av de kvarvarande felen kan stå för en majoritet av de observerade felyttringarna, <Fenton_2>.
62. Se 4.4.1.2. Jämför även fotnot 384.
63. En mängd modeller finns inriktade mot olika applikationsområden och olika klasser av data. Exempel: Musas modell är anpassad mot telekommunikation, Schneidewinds mot rymdtillämpningar. En del modeller skattar MTBF (t ex MUSA) andra ger statistik över antal felyttringar (t ex Schneidewind).
64. Exempel: Gäller t ex skyddssystem.
65. Se 4.5.2.4.
66. Exempel: Analys av minne/*heap*, I/O-portar, vissa maskinvaruberäkningar, tidsövervakare. Specialfall: Analys av stackanvändningen för kontroll av, att maximal stackstorlek under exekvering ryms inom tilldelat minne. Ada ger stöd, t ex genom väldefinierade anrop/återhopp, klar separering mellan statiska resp dynamiska typer, olika *pragma* (*No_Allocators, No_Implicit_Heap_Allocation, Reviewable, Inspection_Point*).

2. Tidsanalys skall utföras, för att kontrollera, att funktionalitet kommer i rätt tid och i rätt ordning[67] {HML}.
3. Objektkodens realtidsbeteende i en deterministisk, värsta-falls-situation[68] skall bestämmas {HML}.
4. Resursanalyserna skall kompletteras med test i målmiljö[69] {HML}.

4.3.3 Systemsäkerhetsanalys på programvara

Principiellt är det inte någon skillnad på den typ av säkerhetsverksamhet, som utförs på programvara jämfört med den, som tillämpas på andra systemdelar eller det övergripande totalsystemet. Det är programvarans inneboende egenskaper (6.6.2.), som gör, att andra åtgärder behöver vidtas, för att försäkra sig om ett tillräckligt säkert slutsystem.

Säkerhetsanalyserna inleds på övergripande systemnivå och ned i varje enskilt system. Den risknivå, som bedöms vara tolerabel för det enskilda systemet i alla dess exemplar under beräknad drifttid fastställs. Potentiella olyckor identifieras. Syftet med dessa inledande analyser är, att ge underlag till sådana förbättringar i systemkonstruktionen, att riskkällor undanröjs helt eller att riskerna reduceras till tolerabel nivå. Detta resulterar i, att en högsta nivå fastläggs för den kritikalitet, som kan tolereras för enskilt system och ingående delsystem, att kompletterande säkerhetsstrategier och konstruktionsprinciper definieras eller att extra säkerhetsmekanismer specificeras.

Då säkerhetsaktiviteter genomförts på enskild systemnivå, fortsätter analyserna ned på underliggande nivåer (bl a till programvarunivå). Resultaten återförs till systemnivå. Ur det löpande projektarbetet växer p s s en allt mer ingående säkerhetsanalys fram. För varje nivå gäller att riskkällor, konsekvenser etc är relaterat till närmast överliggande system. Detaljeringen leder till insikt om vilka delar, som i vilken grad inverkar på den övergripande systemsäkerheten.

Olika metoder för säkerhetsanalys (t ex felträdsanalys) tillämpas på programvaran, för att utreda var och hur programvara kan vara involverad i de samband inom system och omgivning, som leder till olycka. Analysresultaten används, för att detaljera säkerhetskraven (vilket leder till ytterligare konstruktionsrestriktioner)[70]. Syftet med dessa analyser på programvarunivå är att avgöra,

67. Exempel: Oacceptabelt ur tidshänseende är resultat som inte kommer alls, för tidigt, för sent eller i fel ordning. För nödstängningssystem där delsystem skall tas ned i prioritetsordning är ordningsföljden kritisk.
68. Väsentlig information är t ex värsta exekveringstid för nyttjade realtidsprimitiver (se 4.4.2.2) och längsta nätverksförsening. Nyare teorier finns, som med **viss sannolikhet** garanterar att samtliga *deadlines* kan mötas (istället för den **absloluta visshet**, som en värsta-falls-analys måste räkna med), <Burns_2>.
69. Exempel: Stresstest, volymtest och exekvering av värsta-falls-scenarier. Jämför krav under [4.3.2.2.5].
70. Exempel: Riskkällor: Bussdörr slår igen vid på-/avstigning. Buss i rörelse med öppen dörr. Konstruktionskrav: Hinder i dörröppning inhiberar stängning. Buss ej körbar med ostängd dörr.

vilka programvarudelar, som är kritiska, och var det är nödvändigt att få ned graden av kritikalitet, för att inte överskrida den för systemet tolerabla nivån.

1. Säkerhetsanalys skall utföras på programvara och baseras på resultat från närmaste systemnivå {HML}.
2. Ändringar i säkerhetsanalyser på systemnivå, skall föras ned till motsvarande analyser på programvarunivå {HML}.
3. Successiva säkerhetsanalyser skall utföras under programvarans livscykel med utgångspunkt från krav, arkitektur, gränssnitt, konstruktion och implementation samt vid ändringar, som påverkar redan genomförda analyser {HML}.
4. Gränssnittsanalyser skall utföras på programvarans snitt mot externa enheter, mot olika aktörer/roller, samt på snittet mellan kritiska och icke-kritiska delar {HML}.
5. För kritiska programvarudelar eller -funktioner skall klarläggas, att en programvaruimplementering inte ökar säkerhetsrisken jämfört med en icke-programvarubaserad realisering {H}.
6. Analysresultaten skall presenteras vid planerade granskningar av programvaran samt vid säkerhetsgenomgångar på systemnivå {HML}.
7. Spårbarhet mellan programvaran samt systemets identifierade riskkällor och vådahändelser skall föreligga {HM}.
8. Genomförda analyser skall visa, vilka delar av programvaran, som är kritiska samt graden av kritikalitet {HML}.
9. Analysen skall drivas så långt, att den kritiska kärnan i varje kritisk del identifierats {H}.
10. Om en övervakande / styrande funktion inte analyseras i detalj skall den tilldelas samma kritikalitet som övervakad resp styrd funktion {HML}.
11. Programvara, data, kommandosekvens eller operatörsprocedurer, som levererar information till kritisk kärna eller på annat sätt påverkar denna, skall räknas som kritisk av samma grad som kärnan {HML}.
12. Förekomst av sidoeffekter och konsekvenser från andra delar än de kritiska skall utredas under säkerhetsanalyserna och elimineras {HML}.
13. Analysresultaten skall användas för att komplettera säkerhetskrav och motsvarande verifierande testfall samt för att införa eventuella konstruktionsrestriktioner {HML}.

Exempel på checklistor vid säkerhetsanalys av programvarans olika representationsformer[71] ges i bilaga 6.5.

71. D v s all dokumentation beskrivande programvaran (krav, konstruktion, implementation, analysresultat).

4.4 Produktionsmiljö

Detta avsnitt inkluderar krav på hjälpmedel vid utveckling av kritiska, programvarubaserade system. Kraven berör främst verktygsspecialister.

4.4.1 Stödverktyg

4.4.1.1 Konfigurationshanteringssystem

1. Ett verktyg för konfigurationshantering skall användas {HM}.
2. Verktyget skall uppfylla grundkrav för konfigurationshanteringssystem enligt kap 5.

4.4.1.2 Felrapporteringssystem

1. Ett verktyg för felrapportering skall användas fr o m start av programvaruarbetet till dess programvaran avvecklas {HML}.
2. Verktyget skall uppfylla grundkrav för felrapporteringssystem enligt kap 5.
3. För varje enskilt fel i system eller komponent, skall framgå felyttring, observationsdatum, fas, berörd leverans/konfiguration/komponent, felrapportör, allvarlighetsgrad/kritikalitet/prioritet samt uppgifter, som stödjer felanalys[72].
4. Informationen skall kunna kompletteras allt efter ärendets hantering med troliga felorsaker, ansvarig person, prioritet, kostnadsskattningar, rekommenderade/beslutade/utförda åtgärder, felrapportstatus[73], verifieringar[74], utgåva där beslutad ändring införts samt datum för utgåvan.

4.4.1.3 Kravspårningsverktyg

1. Ett verktyg för kravspårning skall användas {H}.
2. Verktyget skall uppfylla följande egenskaper{H}:
   a) Skydd mot uppdatering av fruset krav skall kunna åsättas[75],
   b) Back-up / recovery-mekanismer skall finnas.

Följande är exempel på önskvärda egenskaper för denna typ av verktyg:

Inmatning/utmatning/dokumentation

- Krav kan skrivas in direkt via verktyget.
- Krav kan automatiskt extraheras ur färdiga kravdokument[76].

72. Se felanalys 4.3.2.2.6.
73. Exempel: Öppen/analyserad/beslutad/stängd och motsv datum.
74. Verifieringar att utförda ändringar eliminerat felyttringen och inte infört nya fel eller nya säkerhetshot.
75. Vissa operationer (frysning, godkännande etc) skall kunna förbehållas en viss kateori användare.
76. T ex i MS-Wordformat.

- Färdiga kravdokument kan hanteras[77].
- Kravtext med bilder och tabeller kan hanteras smidigt.
- Generering av dokumentation ur kravverktyget är möjlig.
- Utmatning av selekterad information är möjlig.

CCC

- Motsägelsefrihet, fullständighet och felfrihet föreligger mellan och inom olika kravbeskrivningar och relationer[78].
- Verktyget varnar/hindrar, där ovanstående kan äventyras.
- Relationer, som saknar associationer, kan markeras[79].

Spårbarhet

- Relationer kan definieras mellan olika klasser av objekt.
- Spårbarhet är möjlig på olika nivåer i en hierarki [80].
- Möjlighet till expandering/hopslagning av ett/flerakrav till flera/ett, föreligger.
- Spårbarhet mellan en objektklass och element i andra verktyg/aktiviteter är möjlig[81].
- Attribut/kommentarer/frågeställningar/historik/datum/kravid kan knytas till ett krav.
- Stöd för konsekvensanalys/ändringsanalys ges.

Stora system

- En stor kravmängd/flera kravdokument kan hanteras och lagras.
- Flera användare kan samtidigt nyttja systemet.
- Svarstider vid fleranvändning och stora system är acceptabla.
- Nästlade abstraktionsnivåer kan definieras[82].
- Distribuerad kravhantering understödjs[83].
- Återanvändning av gamla krav, t ex från ett kravbibliotek, understödjs.
- Flexibel kravnedbrytning är möjligt[84].

77. Se föreg fotnot.
78. D v s **ccc**: *correctness, concistency and completeness.*
79. T ex krav, som inte kopplats till något acceptanstest, eller till någon implementationsdel.
80. T ex mellan ett övergripande krav och motsv underliggande delkrav.
81. T ex mellan krav och testdokumentation, analys- och designdokument, implementationsdel, felidentitet i en feldatabas. Spårbarheten skall föreligga i båda riktningarna.
82. T ex för att möjliggöra gruppering av krav, nyckelordstilldelning.
83. För fördelning på geografiskt spridda arbetsgrupper, underkontraktörer, anläggningar och en säker, tillförlitlig (ccc: korrekt, konsistent och motsägelsefri) återhämtning till *master*-versionen.
84. Krav skall ej behöva detaljeras till samma nivå (t ex skall ej samtliga detaljkrav för ett COTS behöva specificeras).

Utbyggbarhet
- Mekanismer för koppling till andra verktyg eller databaser finns.

Konfigurationshantering
- Nya versioner av kravdokument kan hanteras.
- Enstaka krav kan versionshanteras inom verktyget.
- Revisionshistorik kan sparas för tidigare versioner.
- Baselines över tidigare objektversioner kan definieras.

Säkerhet (*safety and security*)
- Behörighetskontroll kan åsättas objekt [85].
- Skydd mot uppdatering av fryst krav ges [86].
- Verktyget kan installeras på IT-säker plattform [87].
- Back-up/recovery-mekanismer finns.

Användarvänlighet
- En ny användare kan lära sig systemet på några dagar.
- Användardokumentationen och användarsnitt är anpassade till novis och expert.
- Användardokumentationen ger en lista över kända problem.
- Grafisk presentation av objekt, länkar, spårkedjor är möjlig.
- Verktygets hantering kräver ej specialexpertis eller speciell administratör.

Leverantör
- Professionell leverantör med erfarenhet av verktyget på stora system.
- Leverantören har förståelse för CM-hanteringsproblem vid stora kravmängder.
- Leverantören kan ge kompetent och snabbt stöd samt god utbildning.

4.4.2 Programvaruverktyg

1. Verktyg, som används vid produktion eller drift av kritisk programvara
   a) skall ej medföra hot mot systemsäkerheten {HML},
   b) skall vara certifierad av en oberoende instans mot en adekvat, officiell specifikation eller standard {H}.

85. Access skall kunna begränsas till den grupp av krav, objekt, länkar, för vilka en person ansvarar.
86. Vissa operationer (frysning, godkännande etc) skall kunna förbehållas en viss kategori användare.
87. OS skall ej ha brister i IT-säkerheten (H SäkIT:s riktlinjer för IT-säkerhet skall vara uppfyllda).

2. Verktyg, som används vid drift av kritisk programvara skall uppfylla samma krav som gäller för kritisk, nyproducerad programvara {H},
3. Systemsäkerhetsanalys m a p verktyg skall utföras för att identifiera risker för kritiska delar {HML}.
4. Skydd skall införas, där säkerhetsrisker identifierats, för att få ned dessa till tolerabel nivå {HML}.

Certifiering är inte något bevis för felfrihet. Godkända prov och granskningar samt en spridd användning är faktorer, som bidrar till ökad konfidens för att produkten är välutvecklad.

Verktyg är att betrakta som COTS-programvara. Se därför rekommendationer under avsnitt 4.5.1.

4.4.2.1 *Formella verktyg*

1. Använt bevisverktyg skall visas vara korrekt {HML}.

4.4.2.2 *Kodgenereringssystem*

Under denna rubrik hanteras kompilatorer, länkare, programladdare och övriga typer av automatiska kodgenererare.

En kompilator är en komplex programvaruprodukt. Ytterst få har certifierats mot någon säkerhetsstandard (eller validerats mot sin språkdefinition). Detta är skälet till att en kritisk applikation behöver verifieras ned på objektkodsnivå (se 4.3.2.2.4.). För några språk utförs dock regelbundna kompilatorvalideringar av nationella, auktoriserade organ. Detta gäller t ex Ada83, Ada 95 och Pascal[88].

Utöver kraven i föregående avsnitt skall följande uppfyllas:

1. Använda kompilatorer skall generera korrekt objektkod från given källkod[89] {H}.
2. Rapport över kända, rättade kompilatorproblem skall införskaffas {HM}.
3. Ur rapport över kända, rättade kompilatorproblem skall framgå att rättelserna verifierats samt att inga nya fel introducerats[90] {H}.

88. Exempel: Ada-kompilatorn C-SMART för kompilering antingen m a p det kompletta språket eller m a p ett säkerhetskritiskt subset, där konstruktioner, som kan leda till indeterministiskt beteende eliminerats eller begränsats (se 4.5.2.6). Kompilatorn är certifierad mot nivå A enl DO-178B. Se <Aonix>.
89. Exempel: Kompilator validerad och godkänd av (inter)nationell organisation eller analyserad enl 4.3.2.2.4. Där risk för kompilatorfel inte kan uteslutas kan flera kompilatorer användas för generering av målkod.
90. Exempel: Testrapporter och andra kvalitetsdokument utarbetade efter rättning och test, som ger belägg för att kända fel eliminerats samt att regressionstest utförts med korrekt utfall även för tidigare problemfria delar.

4. Aktuell lista över kända, icke rättade kompilatorproblem skall införskaffas och utgöra underlag vid bedömning av språkkonstruktioner som kan hota säkerheten och som därmed skall uteslutas. Vilka dessa är, skall framgå av fastställd Kodningsföreskrift (4.5.2.7.) {HML}.
5. Den kompilatorversion och de kompilatoroptioner som gällde vid validering av kompilatorn skall vara identiska med de, som används vid generering av programvara för frysning och test samt för operativ drift {HML}.
6. Ny version av genereringsverktyg, kompilator, länkare, laddare eller ändrad kompilatoroption kräver omverifiering av kritiska programvarudelar[91] {HML}.
7. Ingen optimeringsoption skall vara vald vid kompilering och länkning av kritiska delar {HM}.
8. Kompilatoroptimeringar skall verifieras m a p korrekthet, om ej testfall kan väljas, så att testtäckning av objektkod uppnås {L}.
9. För språk, där generiska enheter ingår, skall vald kompilator realisera instansieringar genom macroexpansion och ej genom koddelning[92] {H}.
10. Källkod skall kunna spåras till objektkod[93] {H}.
11. Objektkod ej spårbar till källkod[94] skall identifieras och verifieras {ML}.

4.4.2.3 *Statiska och dynamiska analysverktyg*

Verktyg för statisk analys söker igenom källkodens styr-, data- och informationsflöden i jakt på ologiska konstruktioner, t ex onåbara delar i programflödet, omotiverade definitioner, användningar eller tilldelningar av variabler (t ex tilldelade värden, vilka skrivs över före första användning)[95].

Dynamiska analysverktyg registrerar beteendet hos ett system eller en komponent under exekvering, t ex vilka programvaruvägar som genomlöpts (test-

91. Undantag: Där resulterande objektkod ej påverkats/förändrats.
92. Kravet på testtäckning av alla koddelar är svårt att uppfylla vid koddelning. Se <IEC 15942>.
93. För språk, som ej klarar detta, krävs testtäckning ned på maskinspråksnivå. Bruk av Ada´s kortformer för logiska relationer (*and then, or else*) medför, att ev krav på MC/DC-test kan undvikas i dessa uttryck. *Pragma Reviewable*, *Pragma Inspection_Point* understödjer spårbarhet till objektkod.
94. Exempel: Initialisering, inbyggd fel-/exceptionhantering.
95. Några verktyg utför även semantisk analys, t ex SPARK, vilken från annoteringar (Ada-kommentarer med speciell SPARK-syntax), pre- och postvillkor samt *run-time*kontroller verifierar programmets verkliga beteende mot en modell av förväntat beteende. Detta förutsätter ett programbeteende helt predikterbart från koden. SPARK har därför uteslutit oklara eller osäkra konstruktioner (-oklara, där en kompilatortillverkare ges flera implementeringsalternativ, -osäkra, där minnes- och tidsåtgång ej säkert kan förutsägas, se 4.5.2.6). SPARK är formellt definierad m h a relationsalgebra i Z. Se även <SPARK>.

täckningsverktyg), graden av processorutnyttjande (last), prestanda och minnesförbrukning.

1. Verktyg skall användas för statisk och dynamisk analys {HM}.
2. För test skall verktyg användas, som kan generera normala, felbenägna och säkerhetskritiska betingelser (t ex simulatorer, stimulatorer, generatorer) {HM}.
3. Instrumenteringsverktyg, som lägger till instruktioner i programvaran, skall ej användas på slutlig programvaruversion {HML}.

4.4.3 Emulerad målmaskin

En emulator för aktuell målmaskin kan bli nödvändig att använda i lägen, där målmaskinen ej är tillgänglig för test. Emulatorn kompletterar, men ersätter ej exekvering i målmiljö (t ex är prestandaegenskaperna ej representativa). Slutvalideringar måste därför alltid köras i komplett målmiljö.

1. Använd emulator skall korrekt efterlikna måldatorns beteende {HML}.
2. Skillnader i beteende mellan emulator och målmaskin skall dokumenteras {HML}.
3. Risk för anpassning mot emulatorns specifika egenskaper skall analyseras och åtgärder definieras {HML}.

4.5 Produkt

Säkerhetsmässiga krav på produkt / system som helhet behandlas nedan. Fokus ligger på återanvänd eller nyutvecklad programvara (och motsvarande dokumentation) sett i ett övergripande systemsäkerhetsperspektiv.

4.5.1 Standardprodukter – Återanvända Komponenter – Hyllvaror

Avsnitt 6.6.5. diskuterar allmänna problem vid återanvändning av programvara. Detta avsnitt innehåller krav på återanvänd produkt avsedd att ingå i en kritisk programvarudel. Dessa säkerhetskrav berör såväl systembeställare, systemleverantör som leverantör[96] av återanvänd komponent (nedan kallad underleverantör).

1. En tidigare utvecklade produkt, som antingen är avsedd att ingå i en kritisk programvarudel, eller som genererar kod, data eller exekverar kritisk programvara i målmiljö, skall uppfylla samma säkerhetskrav, som gäller vid nyutveckling till säkerhetskritiska system {H}.

96. Denna leverantör behöver m a o ej vara identisk med systemleverantören.

2. En tidigare utvecklade produkt, som antingen är avsedd att ingå i en kritisk programvarudel, eller som genererar kod, data eller exekverar kritisk programvara i målmiljö, skall uppfylla samtliga delkrav nedan {ML}.
   a) Produkten skall ha använts i större omfattning och i liknande miljöer, utan alarmerande problem.
   b) Leverantörsbedömning skall visa, att kompetens, produktionsprocess och produkt är tillfredsställande[97].
   c) Tillfredsställande dokumentation skall föreligga över produkt och produktverifieringar eller ha tagits fram i efterhand, där dessa saknats[98].
3. Produkt, som modifieras[99] före återanvändning, skall uppfylla de krav som gäller för en nyutvecklad produkt (se 4.5.2.) {HML}.
4. Produkter från olika underleverantörer skall undvikas i samma system[100] {HML}.
5. Återanvända komponenter avsedda för icke-kritiska delar skall isoleras från kritiska delar {HML}.
6. Genomförd systemsäkerhetsanalys, där konsekvenserna av tänkt bruk och produktunderhåll[101] utretts, skall visa att riskerna ej överstiger de, som klassats som tolerabla för systemet {HML}.
7. Granskning av systemets och produktens kravspecifikationer skall ha utförts och produkten provats, för att utreda risker vid användning av

97. Exempel: uppfylls minst ISO 9001, 9000-3? Vilka metoder och verktyg har använts? Finns feldatabas?
98. *Eng revers engineering.*
99. **Modifieringar** ej utförda eller sanktionerade av produktleverantör (eller motsv) innebär ett ingrepp i köpt produkt och kan skapa problem i synnerhet om det inte klart fastlagts, vem som äger kraven och svarar för verifiering av dessa. I stället bör produkt väljas, som tillåter **anpassningar** i form av parametersättning och konfigurering.
Begränsade modifieringar, t ex i form av ett **enkelt skal** kring produkten kan vara både acceptabelt och nödvändigt, t ex för att avskärma oönskad funktionalitet, lägga till saknad funktionalitet, förhindra felaktig inmatning eller för att vidtaga lämpliga åtgärder, då COTS-produkten utför andra funktioner än vad applikationen kräver. Denna typ av COTS-modifiering kan dock komplicera ansvarsförhållandet underleverantör-systemleverantör-systemkund och försvåra möjlighet till *support* vid problem. Ansvaret för skalets underhåll hamnar hos modifierande part. För att underlätta verifiering och underhåll skall skalet hållas enkelt (dess komplexitet skall ej vara högre än den funktionalitet som nyttjas). Ändras COTS-produkten eller delar som interagerar med denna, fordras även ny analys för försäkra sig om, att skalets skydd fortfarande är adekvat.
100. De flesta säkerhetsproblem visar sig i interaktion mellan komponenter och med omgivningen. Olika, tidigare ej integrerade produkter från flera leverantörer med olika utgivningsplaner kan leda till komplicerade knytningar, kompatibilitetsproblem och t o m låsningar, vilka är starkt kostnadsdrivande.
101. T ex den typ av åtaganden, som förutsätts från produktleverantören för säker livstidsanvändning.

produkten i avsedd situation och målmiljö. Speciell uppmärksamhet skall därvid ha riktats mot förekomsten av explicita krav i produktens kravspecifikation, vilka ej föreligger vid aktuell återanvändning, härledda krav under kravnedbrytningen samt implicita / odokumenterade antaganden gjorda under dess realisering[102]. Effekten av dessa faktorer i aktuell återanvändning skall ha utretts {HM}.

8. Vid integrationstest före provning skall speciellt studeras komponentens gränssnitt (nyttjande av standarder och protokoll)[103], instrumentering (parametersättningar och konfigureringsmöjligheter), reaktioner under systemstress samt felåterhämtning[104] {HML}.
9. Komponentprov skall utföras med användningsprofiler relevanta för aktuellt system och omfatta komponentens funktioner och mekanismer samt prestanda under såväl normala som extrema betingelser {HML}.
10. Slumrande[105] kod resp funktionalitet skall identifieras och dokumenteras {HM}.
11. Möjliga interaktioner mellan produkt och system, icke-tolererad interaktion[106] samt systemdelar involverade i dessa fall skall utredas {HM}.
12. Ett okomplicerat programvaruskal kring produkten skall användas, för att förhindra möjligheter till felaktig inmatning eller riskfylld, onödig, oönskad utmatning/funktionalitet samt för att i dylika fall vidtaga lämpliga åtgärder[107] {HML}.
13. Krav på skalets funktionalitet skall specificeras och verifieras, innan avskärmad produkt integreras med systemet för vidare test {HML}.

Krav inför ändringar under systemrealisering:

14. En ny version av en återanvänd komponent eller en ny version av programvara som kommunicerar med återanvänd komponent kan accepteras i en kritisk del, endast om {HML}
    a) Alla ändringar i förhållande till den föregående versionen är dokumenterade och kända.

102. Exempel: Dolda faktorer i tidigare användning kan gälla antaganden om datatakt, använda funktioner, exekveringsmod, storlekar, komplexitet, noggrannhet, systemkapacitet (*throughput*) och kringutrustnings prestanda. Jfr 6.9.5. : Ariane 5.
103. Följer t ex flyttalsprocessorn IEEE:s standard för flyttalsrepresentation?
104. Samintegreringen och efterföljande test syftar till att utreda komponentens möjligheter till interaktion under nya betingelser, där underleverantörens ansvar är betydligt svagare än vid problem inom komponenten.
105. *Dormant code*: Oönskad men känd av kunden eller okänd för kund (men känd av underleverantören). Exempel: *Easter Egg*.
106. Ger krav på vilken typ av in-/utmatning ett **skal** runt produkten skall förhindra. Kan verifieras enl 6.8.7.
107. Ignorera eller begära ny inmatning? Avstå eller ge korrigerad funktionalitet? Logga fel?

b) Utförda systemsäkerhetsanalyser uppdaterats m a p införda ändringar

c) Integrationstest upprepats med den nya versionen[108]

d) Granskningar, analyser och test visar, att den nya versionen ej påverkar systemsäkerhetskraven negativt[109].

e) Krav under 3.3.3.1. samt 4.5.2.12. är uppfyllda.

Exempel på produktegenskaper[110], vilka hindrar återanvändning i delar med {HM}-kritikalitet:

- Källkod saknas, har ej gjorts tillgänglig för kvalificeringsinstans eller är av tvivelaktig kvalitet.
- Dokumentation saknas eller är av tvivelaktig kvalitet.
- Oklara beskrivningar beträffande den omgivning, där komponenten är avsedd att verka.
- Tidigare verifieringar, granskningar eller test av komponenten är ofullständigt utförda med tanke på den aktuella återanvändningssituationen.
- Tidigare – eller inför återanvändningen nya – verifieringar, granskningar eller test av komponenten visar på oacceptabla brister hos produkten.
- Analys saknas av specifikationer, förutsättningar, villkor eller förhållanden i omgivningen, för vilket den återanvända komponenten skapades.
- Några förutsättningar, villkor, eller förhållanden i omgivningen, där komponenten ingått tidigare, skiljer mot aktuell tillämpning[111] och kan inte uteslutas ha inverkan på säkerheten.
- Implicita antaganden har gjorts i kod, under granskningar eller analyser, att vissa förhållanden ej kommer att föreligga, och därför ej behöver beaktas.
- Död kod ingår i exekveringskod [112].
- Koden kan exekvera delar eller komma åt data, som är irrelevanta för aktuellt driftläge eller systemmod[113].
- Oklara åtaganden från komponentleverantörens sida beträffande underhåll/vidmakthållande av komponenten.

108. Bevaka bl a: Är tidigare nyttjad funktionalitet oförändrad, ingen ny funktionalitet tillförd, prestanda OK?
109. Är t ex det säkerhetsskydd **skal**et ger intakt? Är någon skyddsfunktion inaktuell?
110. Uttrycken ”saknas”, ”ej tillgänglig” nedan avser åtkomst för kvalificeringsinstans. Kvalificeringsinstans kan vara anskaffare eller annan myndighet (t ex USAF, US Navy).
111. Exempel: Aktuell maskinvara med annan prestanda än vad som gällde tidigare.
112. För distinktionen mellan död kod resp deaktiverad kod, se 6.1.
113. Detta ett exempel på exekvering av s k deaktiverad kod.

Exempel på produktegenskaper[114] som underbygger krav under 4.5.1.:

- Produkten är typcertifierad av tredje part mot en fastlagd standard (t ex DO-178B), där
  - Certifieringsintyg föreligger med uppgifter om hur, när, av vem, mot vilken standard typcertifiering utförts,
  - Dokumentation föreligger[115] och beskriver tillämpade specificerings-, utvecklings- och verifieringsmetoder.
  - Krav- och gränssnittsspecifikationer, testplaner/-specifikationer/-rapporter är tillgängliga.
  - All funktionalitet är dokumenterad och saknar sidoeffekter (bieffekter).
  - Eventuella dolda gränssnitt har dokumenterats.
  - Från dokumentationen skall speciellt framgå:
    - om det finns programvarudelar, som kan aktiveras endast under vissa betingelser (s k deaktiverad kod),
    - om beroende till andra produkter föreligger.
- En oberoende bedömning av leverantören har utförts, som visar på hög kvalitet betr kompetens och produktionsprocess, produkter, kundstöd, uppgraderingsplaner och tidigare ansvars- och garantiåtaganden.
- Avtal är tecknat med underleverantören, som ger kunden (systemleverantören) möjlighet att påverka långsiktiga planer och strategier för produkten, information om kommande utgåvor, upptäckta fel, planerade åtgärder.
- Underhållsavtal, vilket garanterar felrättningar inom specificerad tid och utan krav på t ex byte av OS.

4.5.2 Nyutvecklad programvara

Detta avsnitt behandlar säkerhetskrav på programvara i dess olika former från programvaruspecifikation till konstruktionslösning och slutlig realisering i avsedd processor och målmiljö.

4.5.2.1 Specifikation

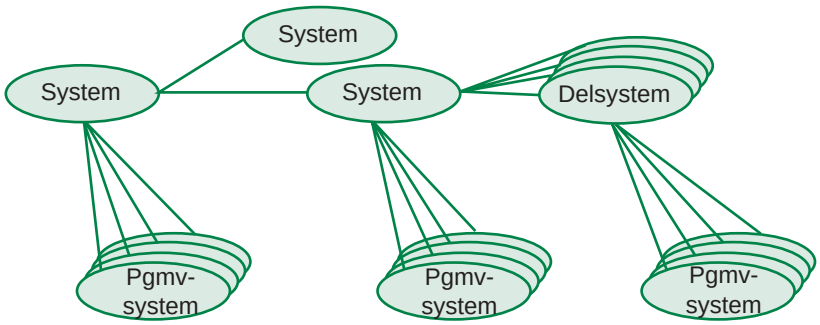
Krav från närmaste systemnivå fördelas till programvaran, för att analyseras och dokumenteras som programvarukrav. Parallellt med detta påbörjas testplaneringen, där metoder för hur enskilda krav skall verifieras fastläggs. Risker identifieras och mekanismer för hur dessa skall hanteras upprättas.

114. Egenskaperna underlättar verifieringsarbetet för en del krav under 4.5.1.
115. Där dokumentation saknas kan ett alternativ vara att återskapa denna i efterhand (*reverse engineering*).

1. Grundkrav för specifikation av programvara enligt kap 5. skall vara uppfyllda {HML}.
2. Spårbarhet mellan ett säkerhetskrav och dess realisering i källkod skall finnas i båda riktningar {H}.
3. Spårbarhet mellan ett säkerhetskrav och dess realiseringkomponent(er) skall finnas i båda riktningar {M}.

4.5.2.2 *Programvaruarkitektur/övergripande konstruktion*

Arkitekturen på den översta nivån i en systemhierarki för ”system av system” fastlägger bl a de gränsyteprinciper, som skall gälla för interaktion mellan ingående, enskilda system och hur IT-säkerhet resp systemsäkerhet skall hanteras. Topparkitekturens systemsäkerhetsvy baseras på säkerhetsanalyser inledda på denna översta systemnivå och ligger till grund för underliggande arkitekturer samt säkerhetsanalyser och säkerhetsvyer för ingående system (se 2.3.).



Inom det enskilda systemets representerar programvaruarkitekturen den första implementationsnivån för programvarukraven. Denna beskriver programvarusystemets övergripande struktur samt en gemensam filosofi (4.5.2.3.) för hur ingående delar skall agera i olika situationer. Speciellt identifieras de delsystem och komponenter, som innehåller eller interagerar med kritiska delar och omgivning.

Säkerhetsanalyserna på denna nivå ligger till grund för beslut om vilka förändringar som är nödvändiga för att få till stånd en säkerhetsinriktad programvaruarkitektur (6.7., 4.5.2.8.) och möjliggöra ett säkert totalsystem. Principer för säkerhetsinriktad konstruktion samt teknik och metoder för det fortsatta konstruktionsarbetet fastläggs. Val av färdigutvecklade komponenter övervägs. Säkerhetsanalyserna förfinas nedåt i strukturen. Planering inför verifiering av kritiska delar påbörjas – allt med syfte, att redan från början beakta och bygga in säkerhet i hela systemkedjan.

1. En säkerhetsinriktad arkitektur skall finnas för programvarusystemet som del i en övergripande systemsäkerhetsarkitektur {HML}.

4.5.2.3 *Grundläggande konstruktionsprinciper*

Dokumentation över den gemensamma filosofin för hur ingående systemkomponenter skall kommunicera (internt och mot omvärld), hantera information (data, tid), allokera resurser (minne, process, prioriteter) och agera i normala resp extrema situationer (feltillstånd, överlast, nodbortfall) upprättas. I denna fastläggs även de speciella procedurer, metoder och tekniker som kommer att användas för konstruktion och verifiering av kritiska programdelar.

1. Konstruktionsprinciper styrande för hur säkerheten programvarumässigt kommer att realiseras i aktuellt system skall definieras och dokumenteras före programkonstruktion[116] {HML}.
2. Konstruktionsprinciperna skall fastlägga vilka strategier för felupptäckt, feltolerans och felsäkerhet, som skall tillämpas. Redogörelsen skall ange valda konstruktionsprinciper, motiveringar för gjorda val[117] och vilka säkerhetskrav, som därmed uppfylls {HML}.
3. Konstruktionsbeslut fattade under hela utvecklingsarbetet skall dokumenteras och inkludera förutsättningar, antaganden samt motiveringar för valda resp förkastade konstruktionsalternativ, kvarstående hot[118] och motsvarande åtgärdsförslag, verifieringsteknik[119], godkännandeinstans och beslutsdatum samt utgåvor/versioner, där valt alternativ införts[120] {HML}.

4.5.2.4 *Systemsäkerhetsinriktade konstruktionsprinciper*

Tidig säkerhetsanalys indikerar vilka delar, som i planerad konstruktion kommer att vara kritiska. Dessa är kostsamma att producera, hantera och uppdatera. Varje annan del, som påverkar en kritisk, måste också betraktas som kritisk. Det är därför ytterst angeläget, att när väl dessa identifierats, alla ansträngningar vidtas för att bringa ned kritikaliteten. Där riskerna trots detta visar sig vara för stora, kan det bli nödvändigt med speciella säkerhetsfunktioner

116. Referenser till Programvarusäkerhetsplanen (3.2.2.) kan ges t ex beträffande valda procedurer, metoder, språk, analys- och verifieringsteknik, ändringshantering och verktyg.
117. Även motiveringar för val av visst implementationsspråk och vid vilka situationer olika språk får användas.
118. För kvarstående hot ingår även att göra avvägningar beträffande riskerna för t ex falsklarm resp missade larm samt riskerna för otillräcklig information resp nedlastning av system och operatör i samband med larm.
119. Exempel: Verifieringsstrategier enl <McNeil>.
120. Jämför DS 00-55 s k *safety-case*-argumentation.

och upprepad analys på förändrade delar, för att visa att riskerna reducerats till tolerabel nivå.

För införda säkerhetsfunktioner avgörs huruvida dessa skall realiseras i programvara eller annan teknik, hur många av dem som behövs, vilken tillförlitlighet dessa måste uppnå osv.

4.5.2.4.1 Allmänna principer

En allmän riktlinje vid val mellan alternativa konstruktionslösningar är att prioritera enkelhet samt säkerställa testbarhet och determinism.

Exempel på ökad testbarhet:

- Kombinationer av tillgängliga faciliteter/optioner/parameterval, som inte kan testas eller får utnyttjas samtidigt rensas bort eller reduceras till ett fåtal testbara varianter.
- Ingång till komponent[121] läggs till, för att ge möjlighet att löpa genom samtliga kommunikationsvägar mellan olika komponenter.
- Ingångar svarande mot speciella felhändelser[122] skapas till komponenten.
- Datastrukturer byggs med operationer för åtkomst av dess innehåll.

Exempel på ökad determinism/minskad komplexitet:

- Endast nödvändiga funktioner implementerade.
- Händelsestyrd lösning[123] ersatt av klockstyrd (statisk).
- Förgreningspunkter med parallella val/grenar i programflödesgraf eller transaktionsflöde ersatta med sekvensiella val/villkor[124].
- Indeterministiska språkkonstruktioner ersatta med deterministiska/statiska[125].
- Multiplikativa/Kombinatoriska villkor/effekter eliminerade. Beroenden frikopplade[126].

1. Konstruktionen av en kritisk del skall vara enkel, testbar och deterministisk {HML}.

121. Export av en operation, som inget utför (s k *dummy*-operation, *No-Op*).
122. Export av en operation, som t ex svarar mot en viss typ av maskinvarufel, vilket påverkar komponentens indata och/eller dess fortsatta exekvering.
123. Dvs med dynamisk, prioritetsbaserad schemaläggning.
124. T ex vid fler än två transaktioner, som uppstår eller upphör samtidigt, reduceras till högst två samtidiga.
125. T ex *for-loop* i stället för *while-loop* och rekursioner (med antalet iterationer bestämd först under exekvering, med osäker prediktering av minnes- och tidsutnyttjande som följd).
126. T ex att på en lägre nivå ersätta beroenden mellan *loop*index (som tillhör olika *loop*ar) med *loop*ar i sekvens, att rensa bort nästlade *loop*ar eller föreskriva ett maximalt djup för nästlade *loop*ar eller för anropskedjor.

2. Ingen funktionalitet utöver de specificerade skall finnas i kritiska programvarudelar[127] {HML}.
3. Vid konstruktion av kritisk del skall hänsyn tas till specificerade, framtida ändringar, så att dessa kan införas utan att äventyra vald säkerhetslösning (se även 3.3.3., 3.4.4.) {HML}.
4. Prototyp skall ej utgöra del av kritisk programvara {HML}.
5. Den enhet, som rymmer ett kritiskt delsystem/komponent/funktion/egenskap skall tilldelas en unik CI-identitet {HML}.
6. För varje konfigurationsenhet (SWCI) skall kritikalitetsnivån fastställas {HML}.
7. Delar, som kan leverera information i konflikt skall identifieras och riskkonflikten skall elimineras {HM}.
8. Prioritering av vilken information, som skall gälla vid konflikt, skall göras utgående från en detaljerad säkerhetsanalys samt motiveras och dokumenteras {HML}.
9. Död kod skall ej ingå i kritiska delar {HML}.
10. Där död kod eliminerats skall berörda säkerhetsanalyser uppdateras {HM}.
11. Deaktiverad kod skall ej kunna aktiveras oavsiktligt {HML}.
12. Mellanformer av död, deaktiverad resp aktiv kod[128] skall eliminieras och berörda säkerhetsanalyser uppdateras {HM}.

Flera av de konstruktionsprinciper, som listats nedan (t ex defensiv programmering, felhantering och feltolerans) är i grunden baserade på tillförlitlighetsteknik, dvs inriktade på att hindra samtliga fel snarare än säkerhetshot. Detta är relevant för skyddsfunktioner, där tillförlitlighet är det primära kravet för vidmakthållen säkerhet. För övriga säkerhetskritiska delar gäller däremot att i första hand eliminera säkerhetshoten (se 6.7.). Även i dessa fall är nedanstående principer relevanta och tillämpas främst vid felsituationer, där säkerheten kan åsidosättas (jämför 6.1.3.).

4.5.2.4.2 Riskreduktion

Principerna för riskreduktion i programvara är gemensamma med de på systemnivå. Tillämpningen av dessa kan dock skilja och för programvara kan de riskreducerade åtgärderna vara enklare att realisera än för maskinvara.

127. Dvs inga sido- eller bieffekter.
Exempel på sidoeffekt: Uppdatering av delad datastruktur, dvs ändring i ett subprogram av en struktur, vilken även används av annat subprogram.
Exempel på bieffekt: En okontrollerad uppdatering enl föreg, där påverkan och effekt är oavsiktlig.
128. Exempel: Se bil Begrepp: *Easter Egg*.

Riskreduktionen inriktas mot att finna lösningsalternativ, som medger att riskkälleexponering, olyckssannolikhet och -konsekvens kan föras ned till tolerabla nivåer. Detta innebär förändringar i ursprunglig programvarukonstruktion. Huvudprincipen är att i första hand undvika de allvarligaste olyckorna och att utföra aktiviteterna i följande prioritetsordning:

1. eliminera identifierade säkerhetshot
2. reducera resterande hot
3. hantera kvarstående hot
4. håll kvarvarande risker under den för systemet fastlagda toleransnivån.

Risken kan elimineras t ex där det går att undanröja

a) tillstånd eller omständigheter, som leder till olycka (dvs riskkällan själv eller något av de övriga villkoren, som gör att riskkällan resulterar i en olycka),

b) riskkällans negativa konsekvenser. Ett sätt att uppnå detta senare kan vara att använda annan teknik, utan dessa risktillstånd eller konsekvenser (se ”Diversifiering” nedan).

Rent allmänt kan riskerna reduceras, genom att begränsa säkerhetskritiska delars storlek, uppgifternas omfattning och komplexitet.

För att speciellt minska sannolikheterna för kvarvarande olyckor granskas händelsekedjorna mot dessa. Den kedja, där risken är störst väljs först. I första hand undersöks om någon enstaka händelse eller situation kan vara olycksorsak (enkelfel). I dessa fall utreds om **ytterligare oberoende villkor**[129] kan införas i olyckskedjan. Detta svarar mot tillägg av ”och”-grindar i motsvarande felträd (se 6.8.3.). Tillagt villkor kan företrädesvis vara baserad på annan, icke mjukvarubaserad teknik. Nästa steg är att granska olyckskedjor med enstaka ”eller”-grindar, dvs där flera alternativa orsaker leder till samma olycka. Strategin här är att se om konstruktionslösningen kan ändras till ett beroende av flera **samtidiga oberoende orsaker**, så att ”eller”-grindar kan ersättas med ”och”-grindar. Olycksrisken reduceras genom att en summa av sannolikheter för olyckan ändras till en produkt (”Diversifiering” och ”Redundans” under 4.5.2.4.5. är exempel på detta).

Ytterligare sätt är att begränsa **riskkälleexponeringen** (tid, volym, närhet och frekvens), att minska riskfrekvensen eller att lägga in ”marginaler” före ett osäkert tillstånd, t ex **fler steg**, flera **moder**, vilka ger operatör eller styrsystem längre tid för åtgärder och fler handlingsalternativ (t ex för övergång till full men säker mod, reducerad och säker mod eller nödtillstånd).

129. Dvs ett eller flera händelser (hur många som behövs för att få ned sannolikheten för aktuell vådahändelse beror på det aktuella systemets toleransnivå och kan ej fastläggas här). För visst system eller applikationsdomän kan detta resultera i krav eller restriktioner av typ ”Vådahändelse *y* skall kunna inträffa endast om minst *n* st oberoende fel föreligger”.

Hantering av kvarstående risker kan bestå i att hålla delar av högre resp lägre (eller ingen) kritikalitet oberoende (en del kan ej påverka data eller exekvering i annan del)[130], att isolera dem fysiskt eller logiskt (skilda datorer, minnen, brandväggar) eller att införa olika **säkerhetsanordningar**, skydds- och **övervakningssystem**, **varningsmekanismer**, självkontrollerande delar, skal mot andra applikationsdelar/COTS/OS.

Till de sista riskreducerande åtgärderna hör, att parallellt med själva konstruktionsarbetet ta fram **dokumentation över kvarvarande identifierade risker** (se 4.5.2.13.), att framställa **material för utbildning och träning** samt att eventuell medverka i utbildningen.

1. För varje kvarvarande risk skall referens finnas till {HML}:
   a) den utredning, där möjligheten att eliminera eller reducera denna klarlagts,
   b) registrerad identitet i felrapporteringssystemet samt
   c) handhavandedokumentation[131] i fall där denna risk sammanhänger med slutanvändares interaktion eller hantering av systemet.
2. Varningsmekanismer skall vara oberoende av kritiska delar {HM}.
3. Inget programvarufel skall kunna vara enda orsak till en olycka {HML}.
4. Ingen enstaka omständighet (händelse, situation, villkor, inmatning i programvaran) skall kunna vara enda orsak till en olycka[132] {HML}.
5. Kritiska delar skall isoleras från eller göras oberoende av icke-kritiska delar eller delar av lägre kritikalitet[133] {HML}.
6. Minnesareor, register och andra delar i exekveringsomgivningen som nyttjas av kritisk programdel skall separeras, isoleras eller göras oåt-

130. Ett exempel på verktyg, som kan analysera beroendet mellan olika delar är SPARK.
131. Se 4.5.2.13.2.
132. Exempel: En konstruktion som nyttjar informationsredundans (i form av diversifierade *input*källor) kan gardera sig mot inkorrekt bestämning av höjdläge, genom att använda tre olika tekniker (tröghetsnavigering, tryck- resp radarhöjdmätning) och jämföra resultat. Gardering mot operatörsfel: två piloter i kabinen.
133. Att förhindra säkerhetshotande interferenser från delar av lägre (eller ingen) kritikalitet innebär att dessa ej negativt skall påverka den mer kritiska delens resursbehov (minne, processortid, I/O-kanaler) och resultat. Flera förslag på angreppssätt finns, <Vito>, <Rushby>. Ett oberoende är lättare att visa vid fysisk separering (t ex uppdelning på olika processorer, olika kanaler för kommunikation, brandväggar) och totalförbud mot interaktion mellan delarna, men medför större kostnader och grövre restriktioner på arkitekturen. Exempel: Logisk separation: **Inkapsling** av de kritiska i paket/modul/subsystem med hög interaktion **inom** strukturen (hög modulstyrka) och låg interaktion utåt (låg modulkoppling). Paket erbjuder olika grad av interaktionsskydd för både data och kod via specifikationens öppna resp privata del och totalt skydd av paketkroppens delar. **Separat kärna** för varje kanal/port, dvs oberoende mekanismer för kommunikation mellan olika partitioneringar (IPC). **Skeduleringsalgoritm**, som låter kritisk funktion exekvera vid avsedd tidpunkt utan säkerhetshotande fördröjningar från andra delar. Se även 4.5.3.1.

komliga från areor använda av delar med ingen eller av lägre kritikalitet[134] {HML}.

7. Säkerhetsövervakande delar skall separeras från driftsstyrande delar[135] {ML}.
8. Säkerhetsövervakande delar skall fysiskt separeras från driftsstyrande delar {H}.
9. Där oberoende händelser eller mekanismer utnyttjats för att sänka kritikalitetsgraden skall oberoendet styrkas[136] {HML}.

4.5.2.4.3 Resurs- och tidshantering (realtid) – Skeduleringsalgoritmer

Resursanalys utförs, för att bedöma behov av konstruktionsändringar m a p resurstilldelningen (se 4.3.2.2.7. samt 4.5.3.).

1. Vald allokeringsalgoritm för fördelning av systemets processer på tillgängliga resurser eller processorer skall garantera, dels att processerna håller sig inom tilldelade tidsramar (*deadline*), dels att prioriterad information kommer fram i rätt ordning och kan hanteras inom tilldelade tidsramar[137]{HML}.
2. Implementerad minneshantering skall tillse, att tilldelade resurser ej överskrids och att omfördelning av resurser ej inverkar menligt på kritiska delar[138] {HML}.

134. Inkluderar förbud mot att dela maskinvara för speciella beräkningar eller med speciell funktion, t ex en s k *watchdog timer*.
135. Exempel: Uppdelning i delsystem/komponenter/moduler och fördelning på olika datorer. NASA föreskriver t o m separering i olika driftsstyrande delar, vilka hålls oberoende, <NASA>.
136. Exempel: Analys av gemensam felorsak (*CCA*), verifierande test.
137. Exempel: Säkra modeller:
   - **Statiska** med cykliska exekutiver är helt deterministiska, men kan vara inflexibla och svårunderhållna.
   - ***Rate Monotonic Scheduling*** är en *preemption*-teknik med fixa prioriteter. Hårda *deadlines* klaras för oberoende *task* av period T_i, om prioriteten P_i sätts > P_j för $T_i < T_j$.. **Tillräckligt** (jfr nästa exempel nedan) är att processorlasten ($\Sigma_i C_i/T_i$) ej överstiger n ($2^{1/n}$-1), där n = antalet *task*, C_i = värsta exekveringstid för ensamexekverande *task* i. För stora n motsvarar detta en last < 69% (för n=2 eller 3 en last < 80%).
     Mer sofisitikerade varianter på RMS finns, där hänsyn tas bl a till respons- och blockeringstider, <Sha>.
   - **Ravenscar-modellen**, dels utan *preemption* dels med *premption* teknik, är naturliga utvidgningar av ovanst. två modelltyper. *Preemption*varianten kan tackla prioritetsinversion. Stöd för modellen kommer att ges i Ada via dess *pragma Restrictions* (<Ravenscar>). Modeller för att klara samtliga *deadlines* med viss sannolihet har också tagits fram, se <Burns_2>. Exempel: Andra analystekniker: **Exakt analys** (tabelldriven) kan visa, att ett schema är skedulerbart, trots att ovanst. lastvillkor ej uppfylls. Exakt analys är också en alternativ metod, om modellförutsättningarna ej gäller (t ex vid *task*-beroenden). Analyserna kan behöva baseras på egna mätningar av värsta-falls-situationer, eftersom fabrikantens uppgifter ej säkert är tillförlitliga (upp till 50% avvikelser har konstaterats).
138. Exempel: Ej virtuellt minne. Ej okontrollerad tidsåtgång t ex för hantering av *overflow* och minnesåtervinning (jfr 4.5.2.6.).

4.5.2.4.4 Defensiv programmering

En teknik avsedd att skapa robusta delar, som kan upptäcka feltillstånd och hindra dess spridning. Kontrollerande kod i form av satser eller exekverbara påståenden (*assertions*) läggs till, exempelvis för att förhindra överskridna indexgränser, *overflow*/*underflow* eller oacceptabla avrundningsfel. Tilläggen kan göras av programmeraren. En del kompilatorer har möjlighet att lägga in viss kontrollkod (I det senare fallet avgör programmeraren enbart lämpliga åtgärder vid felupptäckt). Vid egenprogrammerade kontroller bör dock kompilatorns hantering av statiskt oförändrade kodavsnitt studeras, då dessa kan komma att optimeras bort från avsedd plats. Vissa språk har mekanismer, som stöder defensiv programmering (s k exceptionhantering).

Tekniken kan vara lämplig t ex för att skydda programvaran mot fel i antingen dess inmatning eller dess beräkningar (t ex vid division med noll) eller för att skydda andra delar mot fel i programvarans utmatning.

Felhantering kombinerad med felåterhämtning, se nedan, är exempel på defensiv programmering.

1. Systemet skall kunna fortsätta exekvera säkert trots fel i programkod, data, överföringar eller omgivning[139] {HML}.
2. Programvaran skall kunna hantera kända fel i kompileringssystem eller maskinvara {HML}.
3. Möjlighet att realisera kritisk del i annan, icke-programvarubaserad teknik skall övervägas, antingen för att ersätta eller för att operera parallellt[140] med denna {HML}.
4. Om defensiv programmering används skall den kommenteras och följa fastlagda Konstruktionsprinciper {HML}.
5. All inläst eller till programvaran överförd information skall hanteras av systemet {HML}.
6. Hanteringen skall klara såväl korrekt som inkorrekt[141] information {HML}.

139. Egenskapen kallas felsäker (*failsafe*). Kravet innebär bl a att:
    a) Säker funktionalitet prioriteras över en till alla delar fullständigt korrekt funktionalitet.
    b) Systemet skall vara felsäkert trots fel i använda program- och maskinvarukomponenter eller i verktyg (inkl bl a COTS, GOTS, OS, kompileringssystem) och från användare.
140. Diversifiering. Principen ”så låg risknivå som praktiskt är möjlig” ges i H SystSäk 1.12.2.2 (Jämför ALARP).
    Exempel: Både säkringsbygel till avfyrningsknapp och säkert uformad avfyrningslogik.
141. Giltiga/ogiltiga värden givna/mottagna inom/utom stipulerat tidsintervall inom/utöver angiven min-/maxlast eller mängd.
    Exempel: Värde, som är felaktigt, i felaktig ordning, vid fel tidpunkt (för tidig, för sen, ej samtidig med samhörande data), överflödig, oväntad eller irrelevant. Förväntat värde som uteblir. (Ett värde kan i kombination med andra värden vara irrelevant för visst mod: Flygplan i markläge+landningshjul upp).

7. Tidsgränser för giltighet hos in- och utvärden skall sättas[142] {HML}.
8. Ofullständigt genomförd, kritisk transaktion eller kommandosekvens skall automatiskt inhiberas {HML}.
9. En gräns för antalet (automatiska eller manuella) inhiberingar och hantering då dessa gränser överskrids skall definieras för olika typer av kommandosekvenser {HML}.
10. Funktionsövervakning skall utföras på inaktiva kommunikationsvägar {HML}.
11. Behovet av att sända om information skall baseras på den klassning som gjorts av informationens kritikalitet {HML}.
12. Vid lagring av kritisk information skall robust teknik och flera oberoende media användas {H}.

4.5.2.4.5 Felhantering – Felåterhämtning – Feltolerans

- **Felhantering** innebär, att ett system i felaktigt[143] tillstånd överförs till ett korrekt. Efter felupptäckt avbryts den normala exekveringen, och i stället för att exekvera nästa sats, överförs kontrollen till en alternativ gren. Mekanismer för felhantering finns inbyggda i vissa språk. Ett exempel är Adas exceptionhantering, där feltillstånd/felsituationer kan deklareras i koden och matchande felhantering läggas in, antingen i samma enhet eller i den anropskedja, som leder till enheten. Vid feltillstånd, för vilka felhanterare saknas, avslutas programexekveringen. I större system konstrueras ofta (som en del av dess arkitektur) ett specialdesignat felhanteringssystem för att ensa systemets felhanterings- och felåterhämtningsstrategi över samtliga komponenter (Exceptionhanteringen är där en detalj i den gemensamma filosofin).
- **Felåterhämtning** är en sammanfattande benämning på de olika tillvägagångssätt, som kan tillämpas, för att forsätta exekvering i normal och säker driftmod, då ett felaktigt systemtillstånd identifierats. Två grundläggande modeller för felåterhämtning finns:

  (a) Forward recovery

  Exekveringen fortsätter efter felupptäckt, under det att feleffekterna åtgärdas.

142. Giltiga värden kompletteras med en övre och undre gräns för hur länge värdena är giltiga. Incident p g a obefintlig *timeout*: En kombinerad mekanisk-elektronisk öppningsmekanism till en bomblucka provades under ett flygpass, men utlöstes ej p g a anbringad spärr . Kommandobegäran låg dock kvar i systemet outförd och verkställdes efter avslutad flygning i samband med underhåll (då spärren frigjordes).
143. Fokus här är de felaktiga systemtillstånd, som är säkerhetshotande, dvs kan leda till olycka.

Ett lämpligt alternativ t ex, där interaktionen med omgivningen inte kan tas om, eller där felrättande kod, robusta datastrukturer etc. kan införas.

(b) Backward recovery

Exekveringen backas till ett tidigare tillstånd eller tidpunkt, eventuellt med degraderad funktionalitet.

1. Varje systemkonfiguration skall kunna inta åtminstone ett säkert tillstånd för varje operativt mod {HML}.
2. För väg (*path*), som oundvikligen leder till ett osäkert tillstånd, skall krävas flera samtidiga villkor {HML}.
3. Systemets pågående uppgifter skall kunna avslutas på ett säkert sätt, även om systemet kommer in i osäkert tillstånd {HML}.
4. Från varje osäkert tillstånd skall finnas vägar till ett eller flera[144] säkra tillstånd {HML}.
5. Identifierbara felsituationer eller feltillstånd i operativ omgivning eller i program- och maskinvarukomponenter skall hanteras, även om dessa inte bedöms kunna inträffa i den miljö, där programvaran är avsedd att verka[145]. Hanteringen skall innebära, att systemet bringas till ett säkert tillstånd inom acceptabel tid, för att därifrån fortsätta säker exekvering[146] {HML}.
6. Vald felhanterings- och felåterhämtningsstrategi skall baseras på analyser, där det utretts var systemet skall återta säker och fullständig funktionalitet samt var säker och s k degraderad funktionalitet[147] är enda möjligheten {HM}.
7. Endast icke-kritiska fel skall kunna ignoreras utan felåterhämtning[148] {HML}.
8. Felkällor, feltillstånd, felyttringar eller andra oegentligheter i kritisk programvara skall loggas[149], oavsett om dessa leder till osäkert tillstånd eller ej {HML}.
9. Spårbarhet skall finnas mellan utlösande felsituation och det tillstånd, systemet överförts till[150] {HML}.

144. Flera t ex p g a systemmod, villkor i omgivning, typ av riskkälla.
145. En följd av att inga indeterministiska tillstånd accepteras. Jfr 6.9.5. : Ariane 5.
146. Säker exekvering i full mod där möjligt eller i degraderad (t ex tillräcklig funktionalitet för säker flygning hem).
147. Dessa två alternativ svarar mot engelskans *fail safe* resp *fail soft* (se bil Begrepp).
148. Eng *fail silently.* Vid t ex förlorad information, kan omsändning vara en lösning, se nästa avsnitt.
149. Lagrad information bör (förutom t ex systemkonfiguration, felande enhet, identifierande enhet, diagnos, allvarlighetsgrad) även innehålla data som stödjer tillförlitlighetsanalys och prediktering enl vald predikteringsmodell (t ex total exekveringtid från releasedatum till felupptäckt). Se 4.4.1.2.
150. I fallet Ada innebär detta t ex, att ett fördefinierat eller anonymt *exception* ej får propageras utanför paketets räckvidd. Detta skall i stället gå via ett unikt *exception* definierat i paketet, såvida inte en lämplig hantering kan utföras inom paketet.

- **Feltolerans** möjliggör, att avsedd funktionalitet genomförs korrekt, trots fel i program- eller maskinvara. Detta förutsätter en hypotes om felorsaker och en mekanism, som förhindrar vidare spridning i systemet. Tekniken tillämpas oavsett om fel föreligger och är ett alternativ till felhantering (vilken vid felupptäckt överför systemet till ett säkert, eventuellt degraderat tillstånd). Eventuella inträffade fel döljs därmed för operatören. Olika typer av feltolerant teknik finns:
- **Redundans** innebär, att samma funktionalitet realiseras i flera olika enheter, så att feltillstånd eller felyttringar i en av dessa inte påverkar avsedd funktionalitet i en annan. För att redundans skall vara meningsfull gäller, att de olika realiseringarna är oberoende, dvs ej har några gemensamma felorsaker. Detta beaktas bl a under felträdsanalysen (6.8.3.). Redundans kan åstadkommas genom diversifiering eller identiska kopior. För programvara kan redundansen avse kod, data eller tid (informations-/tidsredundans).
- **Diversifiering** innebär att olika angreppssätt tillämpas, för att minska risken, att feltillstånd eller felyttringar knutna till en viss lösningsteknik sprids vidare i systemet. I programvarusammanhang kan diversifiering åstadkommas, genom att införa en icke-programvarubaserad variant (t ex mekanisk spärr, *watchdog timer*), en annan utvecklingsprocess/språk/kompilatorversion/lagringsmedium/processor/algoritm/lösningsprincip, flera beskrivningssätt (t ex både kod och s k *assertions*) eller genom olika typer av skydd i flera nivåer (*defence-in-depth*). Exempel på tillämpningar: Olika tekniker och informationskällor för höjdbestämning[132]. Skilda detekteringsmekanismer (t ex temperatur- resp radioaktivitetsmätningar) för att undvika ett osäkert tillstånd (läckande kylvätska). En skyddsanordning i flera varianter vid inträde i osäkert tillstånd. Samma typ av verktyg från olika fabrikanter, t ex för att under viss fas verifiera transformationen från en representation till en annan (från specifikation till källkod, från källkod till objektkod[151]).
- **Identiska kopior** (*replication*) är ett bra alternativ, t ex vid risk för slumpmässiga fel i maskinvara eller omgivning. Identisk programvara exekveras på flera maskiner (av samma typ eller av olika varianter). Tekniken är däremot ej verksam mot systematiska fel, t ex programvarufel (vars felorsak kan ligga i kravspecifikation eller konstruktion). Avsedd effekt uppnås ej av att flytta exekveringen till en annan programvarukopia[152].
- **Alternativa programvarulösningar** [153] innebär, att varianter av systemets programdelar utvecklas parallellt i oberoende och åtskilda grupper från

151. Exempel: Diversitet på objektkodsnivå: samma källkod körd genom olika kompilatorer.
152. Jfr 6.9.5. : Ariane 5.
153. Eng *Dissimilar software, Diverse software, Multiversion software, Multiple-version dissimilar software*.

samma specifikation m h a olika språk, plattformar och verktyg. Distinktion görs ibland mellan:

(a) Funktionell redundans -Alternativa lösningar med identisk funktionalitet [154]

(b) Analytisk redundans -Alternativa lösningar med likartad funktionalitet [155].

Erfarenheterna från system, där programdelar utvecklats parallellt i oberoende och åtskilda grupper från samma specifikation m h a olika språk, plattformar och verktyg (a), har visat att den ökade kostnaden inte uppvägts av högre säkerhet. Komplexiteten ökar i och med att skiljaktiga resultat måste utvärderas och att två produkter i stället för en behöver underhållas. Båda drabbas också av samma specificeringsfel och ofta även samma typ av logiska misstag. Det har därför visat sig bättre att satsa på diversitet i form oberoende teknik (t ex en maskinvaruvariant) eller skilda fysikaliska principer i stället för funktionella varianter framtagna genom oberoende utveckling.

4.5.2.5 *Språk och språkkonstruktioner*

Valet av språk har stor betydelse vid implementering av kritisk programvara. I princip skulle lågnivåspråk kunna användas, om en starkt stringens och konformitet tillämpades vid val av språkelement och konstruktionslösningar. Dock blir uppgiften att verifiera koden m a p olika säkerhetsaspekter samt kostnaderna för verifieringarna närmast övermäktiga. Den låga abstraktionsnivån ökar vidare risken för degenerering av strukturen efter en tids underhåll, (ofta utförd av andra än de, som ursprungligen konstruerade programvaran) och kräver kostsamma omverifieringar. Bruket av lågnivåmekanismer bör därför hållas lågt och begränsas till delar, där en direkt mappning mot målsystemet är nödvändig.

Det finns språk, som ger möjlighet till större säkerhet och god konstruktionsteknik, t ex genom att inskränka möjligheterna till integritetsbrott [156] samt erbjuda en väldefinierad semantik (även i felsituationer), stark typning [157], kopplingar från källkod till motsvarande objektkod[158] och som automatiskt eliminerar oanropad kod.

Ada 95 är närmast unik i detta avseende och för övrigt det enda, som speciellt adresserar säkerhet[159]. Dess kompilatordirektiv (*pragma*) medger t ex, att

154. *Eng N-version programming*, se även Feltolerans. Vid skilda resultat röstas om vilket som skall negligeras.
155. *Eng Modelbased redundancy*. Hit kan även s k fysikalisk diversitet räknas (skilda fysikaliska principer, var och en med sina parameteruppsättningar och därtill hörande tröskelvärden. Då dessa överskrids ersätts aktuell princip med annan t ex genom övergång från det primära till det sekundära systemet. Alternativt tillämpas båda principerna samtidigt och då ett av dem överskrids aktiveras nödstoppet).
156. Exempel: Överskrivning av data, ändringar av styrflöde.
157. Typning reducerar räckvidd och kostnad för analys och verifiering.
158. Denna spårbarhet underlättar detaljgranskningar och krav på testtäckning ned till maskinspråksnivå.
159. Se <Ada95> Annex H: *Safety & Security*.

biblioteksenheter kan läggas i skilda partitioneringar, vilka ej delar variabler eller fysisk adress och som kan kommunicera på ett väldefinerbart och analyserbart sätt. *Pragmas* underlättar vidare analys av objektkod (m a p exekveringstid och minnesutnyttjande) och kan utestänga vissa konstruktioner[160]. Språket har redan visat sig mycket lämpat för säkerhetskritiska applikationer.

1. Systemsäkerhetsanalys skall ha utförts[161] på de språk, som planeras för kritiska delar {HML}.
2. Analyserna skall speciellt belysa möjligheten att kunna bygga konstruktioner som är deterministiska, modulariserade, spårbara, robusta och väldefinierade.

Goda abstraheringsmöjligheter och kraftfulla språkmekanismer[162] kan dock inte hindra, att ett språk används på ett ur säkerhets- och verifieringssynpunkt olämpligt sätt. En säker programvarukonstruktion kräver, oavsett språk, att följande egenskaper är uppfyllda:

Deterministisk/Predikterbar

3. Endast språkkonstruktioner, vars effekter är kända eller kan bestämmas från källkoden skall användas[163] {HML}.

Modulariserbar

4. Konstruktioner för inkapsling och skydd av koden skall användas[164] {HML}.

Spårbar

5. Endast konstruktioner som inte försvårar spårbarhet och analys av källkod resp exekverbar kod skall användas[165] {H}.

160. Exempel: Restriktioner kan läggas in betr *tasking* (*protected types*), minnesshantering (pekarstrukturer, minnesåterlämning), *exception*hantering, okontrollerade typkonverteringar, *dispatching*, rekursion.
161. Denna analys kan ha utförts av annat projekt, instans eller organisation.
162. Exempel: Möjligheter till inkapsling, abstraktion, felhantering (för att bl a undvika spagettikod, inhopp i interna delar, komplicerade felfallsförgreningar).
163. Exempel: Konstruktioner med sidoeffekter eller där elaboreringsordning, parameteröverföringsteknik (kopiering/referens) etc. beror av kompilatorvariant och därmed kan ge olika utfall, kan ej godkännas. Ej heller implicita gränser för variabler eller *loop*ar, *records* med diskriminanter, varianter, *default*värden.
164. Medger oberoende analys av olika koddelar. Minimerar möjligheten, att ändring i en del får oavsedda effekter i en annan del. Analysen underlättas betydligt om dessutom möjlighet finns till separatkompílering av moduler och beroendekontroll mellan moduler. Dessa faciliteter utgör dock ej krav.
Exempel: Modul/paketbegrepp, Separering av specifikation och implementationsdelar, Privatdeklarationer, Begränsningar i deklarationers räckvidd.
165. Exempel: *Goto*-satser, *loop*ar med multipla utgångar samt konstruktioner, som kräver omfattande *run-time*stöd eller extra kod skall undvikas (t ex *dispatching*, *variantrecord*). Dessa försvårar bl a analys av styrflöde och av uppnådd täckningsgrad. Oändliga *loop*ar (t ex *loop* utan utgång) försvårar tidsanalys, men kan tillåtas givet, att uförd analys ej påvisar problem (t ex i fallet, att en kontinuerlig process skall upprätthållas). Vissa *tasking*mekanismer.

Robust

6. Typinformation skall ges m h a olika sorters deklarationer[166], så att kompilatorn kan upptäcka felaktiga tilldelningar, operationer och anrop {HML}.
7. Delade datastrukturer skall vara skyddade mot samtidiga eller oavsiktliga uppdateringar[167] {HML}.

Väldefinierade språkkonstruktioner

8. Högnivåspråk skall användas {H}.
9. Språkets syntax skall vara formell definierad {H}.
10. Språket skall vara standardiserat av ISO, ANSI eller IEEE {H}.
11. Språket skall vara standardiserat av en internationell eller nationell organisation {M}.
12. Lågnivåspråk eller assembler får endast användas[168] där något av följande gäller {HML}:
    a) Nära interaktion med maskinvara kan inte åstadkommas med ett högnivåspråk.
    b) Prestandakraven kan ej uppfyllas med ett högnivåspråk.
    c) Högnivåspråk, kompileringsystem och motsvarande processorer snarare ökar än minskar säkerhetsriskerna för en liten applikation .
13. Om lågnivåspråk används enligt ovan skall samtliga delkrav nedan dessutom vara uppfyllda {HML}:
    a) Koddelarna hålls mycket små eller applikationen är mycket liten
    b) All funktionalitet och alla effekter är väl definierade och dokumenterade
    c) Koddelarna är inkapslade[169] och isolerade från övriga delar
    d) Koden är okomplicerad, välstrukturerad och analyserbar
    e) Föreskrivna Konstruktionsprinciper och Kodningsföreskrifter följs[170].

För språkkonstruktioner som huvudsakligen berör kompilatorsystemet, se 4.4.2.2.

166. Exempel: Objekt och typdeklarationer för variabler, subprogramparameterar. *Unchecked_Conversion* mellan dessa skall undvikas.
167. Globala variabler är inte på något sätt skyddade. Programmeraren måste m a o bygga ett skydd m h a mer grundläggande språkelement, såvida inte språket innehåller språkelement med detta skydd. Ada erbjuder kontrollerad åtkomst av delade datastrukturer m h a s k *protected types*.
168. Jämför motsvarande krav i DS 00-55.
169. Exempel: Inkapslad i kroppen till ett paket eller ett subprogram.
170. Se 4.5.2.2. samt 4.5.2.7.

4.5.2.6 Språkrestriktioner

Ett sätt att minska risken för säkerhetshotande konstruktioner är att lägga restriktioner på användningen av vissa språkelement, t ex genom att definiera en säkerhetskritisk delmängd av språket. Detta kan förenkla verifieringarna. Det kan dock finnas skäl, att tillåtna vissa kraftfulla språkelement, t ex då konstruktion med en begränsad mängd tillåtna element skulle leda till en mer komplex (och därmed svårverifierad) struktur. En förutsättning är dock, att detta inte bryter mot kraven i föregående avsnitt. En ytterligare förutsättning är, att Kodningsföreskrifter, som beaktar säkerhetsinriktade aspekter, finns framtagna och tillämpas (se nedan). I realiteten kan även statiska analysverktyg vara nödvändiga, för att underlätta alltför betungande verifieringar.

Ada95 har utvidgats med en mängd kraftfulla mekanismer och erbjuder vissa lågnivåfaciliteter[171]. En del av dessa kan vara tveksamma i kritiska tillämpningar. Exempel på detta är *protected types*[172] samt *tagged types*[173]. För andra krävs vissa restriktioner[174]. Bruket av parallellism komplicerar verifiering av att exekveringen är deterministisk. Risk för låsningar, osynkroniserade uppdateringar och kapplöpningssituationer (*race condition*) etc. måste undvikas. Möjlighet, att kunna välja funktionsvariant vid anrop (*dynamic dispatching*), försvårar också verifieringarna. Användaren måste i detta fall kunna styra och begränsa dynamisk bindning. Hur man kan lösa dessa typer av problem framgår av <IEC 15942>.

Även C++ är standardiserad. Kompilatorn tillåter dock inblandning av C-kod, vilket öppnar för en mängd osäkra konstruktioner[175], C-dialekter och bibliotekskomponenter, som även dessa måste säkerhetsgranskas m a p val av språkelement, dynamiskt minne (*heap*), pekare, etc. Det finns visserligen sammanställningar över restriktioner på C i kritiska system[176], men dessa förutsätter ingående kodgranskningar (ett omfattande, kostsamt arbete om ej extra verktygsstöd utvecklas). Lämpligheten i att tillhandahålla regler och språksubset för C i kritiska system har ifrågasatts[177]. Oro finns, att detta kan tas som intäkt för att språket lämpar sig för kritiska applikationsdelar (se 6.9.2.11.).

Det finns kompilatorer och andra statiska analysverktyg, som kontrollerar, att inmatad kod endast innehåller en definierad, säkerhetsinriktad delmängd

171. Lågnivåfaciliteter underlättar åtkomst av grundläggande mekanismer i målsystemet.
172. Ett alternativ till *rendezvous* för parallellexekvering.
173. Ett objektorienterat tillägg av funktionalitet.
174. Exempel: *Generics, exceptions, tasking*. En begränsning av *tasking*modellen ges av <Ravenscar>, där styrflöden mellan processer omöjliggörs, genom att *rendezvous* och *abort* utesluts.
175. <Romanski>, en granskning av C och <Hatton> ur ett säkerhetsperspektiv (ca 150 osäkra konstruktioner identifierade, <FMV_1>).
176. Exempel: <MISRA_C>: Använd ej objekt som upphört existera. Inför kontroll mot indexöverskridanden. Omreferera ej nollpekaren.
<TÜV>: Använd ej *loop*-variabel utanför *for-loopen*. Nyttja ej implicita typkonverteringar eller nästlade tilldelningar.
177. Se <Wichmann>.

av språket[178]. Standardiserade snitt mot kompilatorbibliotek[179] möjliggör också en marknad för mer sofistikerade analysverktyg. Flera av de mekanismer, som idag måste uteslutas p g a en alltför omfattande manuell verifieringsbörda, kan i framtiden – med verktyg stödda på en förbättrad verifieringsteknik – tillåtas även för system av högsta kritikalitet (se *protected types* ovan).

1. Språkrestriktioner skall fastställas för att ange vilka konstruktioner som ej får användas i kritiska delar {HML}.
2. Icke-deterministiska språkkonstruktioner skall ej användas[180] {HML}.
3. Pekare och processer skall ej användas, såvida ej allokering sker vid programstart och därefter bibehålls statiskt under exekveringen[181] {HM}.
4. Automatisk minnesåtervinning (*Garbage collection*) skall ej användas[182] {HML}.

4.5.2.7 *Kodningsföreskrift*

1. En kodningsföreskrift[183] med åtminstone följande innehåll skall finnas för samtliga språk i kritiska delar {HML}:
   a) Tillåtna och otillåtna konstruktioner,
   b) Regler för märkning, kommentering och namngivning av kritiska delar,
   c) Anvisningar för minimering av komplexitet
   d) Restriktioner p g a problem i kompilator eller målsystem med mera
   e) Detaljerade regler för säker konstruktion i använt lågnivåspråk[184].
2. Kodningsföreskriften skall följa de Konstruktionsprinciper (se 4.5.2.2.), som fastlagts för aktuell applikation {HML}.
3. Avsteg från kodningsföreskriftens rekommendationer skall vara dokumenterade, motiverade samt godkända vid granskningsgenomgångar {HML}.

178. Exempel: För Ada C-SMART (4.4.2.2.) och SPARK (se 4.4.2.3., 6.9.3.1. samt <SPARK>.).
179. Exempel: <ASIS>, ett standardiserat snitt mot Ada-kompilatorns bibliotek.
180. Detta utesluter bl a obegränsad dynamisk allokering av processer och pekarobjekt samt konstruktioner med bieffekter eller som är installations- eller implementationsspecifika (försvårar plattformsbyte).
181. Statiska allokeringar medger predikterbara minnesbehov.
182. Automatisk minnesåtervinning kan innebära opredikterbar tidsåtgång i oförutsägbara situationer.
183. En företags- eller projektgemensam föreskrift. Hänvisningar kan göras till analysverktyg (som rensar bort otillåtna språkkonstruktioner före kompilering) eller säkerhetscertifierad kompilator, där sådana använts.
184. Exempel: Modulär uppbyggnad, där varje modul hålls liten. Strukturerat styrflöde liknande det för högnivåspråk (utan t ex *goto*:s eller multipla *entry/exit*-punkter). Inga kryptiska eller obefintliga kommentarer etc.

En handledning vid utveckling av säkerhetskritisk programvara i Ada´95 har utarbetats (<IEC 15942>, 6.9.3.). Denna klassar språkelement bl a efter den insats, som krävs, för att verifiera säker användning.

Kärnkraftsindustrin (<NUREG-6463>, 6.9.3.) och den europeiska rymdindustrin, ESA, har också tagit fram handledningar över olika språk (Ada, C/C++ osv.).

4.5.2.8 *Gränsytor*

Gränsyteprinciperna för interaktion mellan systemen i en systemhierarki definieras på en övergripande nivå (se 4.5.2.2.). Följande avsnitt innehåller generella säkerhetskrav för gränssnitt mellan ett programvarusystem och dess omgivning på en lägre nivå. För vissa tillämpningar kan några av dessa krav vara irrelevanta, svåra att tillgodose eller överspelade m a a att mer specifika gränsytekrav definierats. En översyn och komplettering av nedanstående kravmängd baserad på säkerhetsanalyser av det aktuella systemet i avsedd omgivning och användning är givetvis nödvändig[185].

Snittet mot Start- och Initieringsdelar

1. Systemet skall vara i säkert tillstånd under uppstart {HML}.
2. Vid start av systemet skall programvaran kontrollera att avsedd säkerhetsnivå uppnåtts, innan kritiska delar kraftsätts {HM}.
3. Oavsiktliga startkommandon skall ej kunna aktiveras {HML}.
4. Intermittenta felorsaker, fluktueringar i kraftsystemet, kraftbortfall, överföring till operativ drift eller nedstängning skall ej kunna föra systemet till ett osäkert tillstånd {HML}.
5. Vid uppstart samt återgång efter temporär nedstängning, skall programvarans modell av systemets processtatus överensstämma med verkligheten {HML}.

Den mänskliga faktorn nämns ofta som bidragande orsak vid olyckor eller tillbud. Mera sällan tillskrivs denna en positiv roll, trots att operatörsingripanden kunnat leda till att konsekvenserna lindrats. Den ökade automatiseringen har dock medfört att en del av det ansvar, som tidigare låg på kontrollrumspersonal, maskinist, pilot osv. nu fördelats över till övriga medverkande i ett systems produktion, drift och underhåll. Nya säkerhetshot kan uppstå ur ett olämplig operatörsgränssnitt i kombination med en svårbemästrad situation. Detta motiverar extra insatser (t ex säkerhetsanalyser på operatörsscenarier), för att försäkra sig om logiskt konsekventa man-maskin-gränssnitt. Arbetet utförs på en högre nivå (ovanför de delsystem, som realiserar interaktionen) som ett led i att få till stånd en säkerhetsinriktad arkitektur (4.5.2.2.).

185. Se även avsnitt [4.3.3.C Gränsytor] i bilaga 6.5.

Nedanstående sammanställning utgörs av allmänna säkerhetskrav på lägre nivå för programvarans interaktion med operatör[186] :

Snittet mot Operatör: Inmatning

6. Felaktig inmatning[187] samt uppgift om felets karaktär skall, där detta är möjlig att detektera, anges direkt efter sådan inmatning {HML}.
7. Där inmatning kan vara kritisk för systemsäkerheten skall detta framgå {HML}.
8. Osäkra tillstånd och farliga aktiviteter[188] skall kunna lämnas eller avbrytas med en knapptryckning eller någon annan enkel åtgärd {HML}.
9. Farlig aktivitet skall kräva minst två separata åtgärder eller flera steg, för att verkställas {HML}.
10. Inga genvägar skall kunna tas i en operatörsprocedur, där en hel sekvens är väsentlig för systemsäkerheten {HML}.
11. Nollställning av ett säkerhetskritiskt larm skall inte vara möjlig, innan korrigerande åtgärder vidtagits {HML}.

Snittet mot Operatör: Utmatning

12. Presenterad information skall vara klar, konsis och otvetydig samt ha en logiskt konsekvent utformning {HML}.
13. Presenterad information skall ge tillräckligt beslutsunderlag, där åtgärder väsentliga för systemsäkerheten krävs {HML}.
14. Aktuell systemmod samt övergång till ny mod skall framgå, även då detta inte direkt orsakats av en operatörsinmatning {HML}.
15. För presenterad information skall det vara möjligt att få fram dess källa, ålder, giltighetstid, orsak till att informationen visas, ändras eller tas bort[189] {HM}.
16. Presenterade värden, som närmar sig resp ligger i säkerhetskritiskt intervall, skall markeras på avvikande sätt {HML}.
17. Om systemet befinner sig i – eller riskerar hamna i – osäkert tillstånd, skall operatör larmas[190] {HML}.

186. Allmänna säkerhetskrav på in- och utmatning ges i 4.5.2.4.4. För generella konstruktionskrav på MMI se t ex <MIL-STD-1472>.
187. Inkluderar även fallet att inmatning uteblivit inom det stipulerade tidsspannet.
188. Vissa system befinner sig ständigt på gränsen till osäkert tillstånd och fordrar kontinuerlig, programvarukontrollerad styrning, för att hindra övergång till osäkert tillstånd. Exempel på enkel åtgärd vid farlig aktivitet: att utlösa raketstol (katapult-).
189. Exempel: Tidstämplat sensordata genererad p g a yttre händelse/operatörsbegäran/fixt tidsintervall. Ändring av valbar knapp/meny/formulär p g a inträffad, högprioriterad händelse.
190. Aktuellt systemmod, deltillstånd, systemets planerade åtgärder, vad som förväntas av operatör, vilket säkert tillstånd systemet överförs till samt vilken konfiguration och status systemet intar efter överföringen skall därvid anges. Gäller t ex vid varningsmeddelanden om lastreducerande åtgärder, avbrottsblockeringar, högre granularitet i prestanda, noggrannhet eller funktionalitet.

18. Säkerhetskritiska varningar eller alarm skall klart skilja sig från rutinmässig information {HML}.
19. Vid motstridig information från olika delsystem, sensorer eller aktörer skall klart framgå hur systemet hanterar detta och vad som förväntas av operatören [191] {HM}.
20. Vid kritisk aktivitet, där både system och operatör kan styra, skall systemet ha prioritet, om tidsfristen inte medger beslut och agerande från operatör. I övriga fall skall klart framgå, om system eller operatör har prioritet [192] {HM}.
21. Effekten av vidtagna åtgärder skall återmatas till operatören {HML}.
22. Operatören skall informeras vid degraderad funktionalitet. För varje tillstånd i systemet, då degradering kan inträffa skall det specificeras hur ofta information om degraderingsmod skall ges[193] {HML}.
23. Operatörsåtgärder skall loggas {HM}.
24. Presenterad information skall loggas {M}.

Snittet mot In- och Utenheter

25. In- resp utmatning av kritisk information skall ombesörjas av minst två oberoende enheter {H}.

Snittet mot Maskinvara

26. Maskinvaran skall ge skydd mot att fel i densamma propageras in i programvaran {HML}[194].
27. Programvaran skall kunna hantera fel, vars ursprung ligger i maskinvaran[195] {HML}.

191. Exempel: Är det t ex piloten eller systemet som bestämmer hur dragkraften skall fördelas mellan motorerna?
192. Boeing prioriterar piloten. Airbus ger systemet högsta kommandot.
193. Hur mycket information kan en stressad pilot tillgodogöra sig i olika lägen?
194. Exempel på sådana mekanismer (s k *hardware failure logic*) är *watchdog timers*, olika typer av redundans, felrättande kod ”inuti” datorn, för att hantera slumpmässiga maskinvarufel i instruktioner och data.
195. **Förutsättningar**: Tillgång till aktuell lista över konstruktionsfel i använd maskinutrustning (t ex fel i buss- eller registeradress/-värde, tidshantering, styrflöde) med uppgifter om ”*workarounds*” och rättningsplaner (se 4.5.3.).
Exempel: Endast för några enstaka kommersiella microprocessorer lämnas fellistor ut. Mest information ger Intel (http: //developer.intel.com/design/litcentr/). Analys av 68 fel rapporterade 9705-9811 för Pentium II, <Avizienis>, visade att felen ledde till systemkrasch i 28%, till hängning i 18%, till funktions-/servicebortfall i 22% , till smärre fel i 32%.
Förfarande: Analysera effekten av maskinvarufel på programvara (t ex FTA på snittet, se 6.8.3.) Anpassa konstruktionen, så att inget enstaka fel kan hota säkerheten (se Riskreduktion under 4.5.2.4.). Implementera lämplig felhanteringsteknik i programvaran (se 4.5.2.4.). Verifiera programvarans reaktion på maskinvarufel (efter kodning t ex via Felinjektioner 6.8.7.).

4.5.2.9 Detaljerad konstruktion

Programvaruarkitekturen förfinas under den detaljerade konstruktionen[196]. Tidigare identifierade kritiska komponenter bryts ner i mindre enheter, vilka antingen har en motsvarighet i programvarans säkerhetskrav eller är enheter, som påverkar kritiska delar.

Samma krav på val av säkra konstruktionslösningar gäller, som under övergripande konstruktion (en av de mer betydelsefulla därvidlag är principen för riskreduktion).

Framkomna aspekter på säker användning av systemet överförs till motsvarande användardokumentation (se nedan). Testprogramvara för verifiering av programvarans säkerhetskrav tas fram (se nedan).

4.5.2.10 Testprogramvara för drift och underhåll

Krav på test av programvaran under produktionsfaserna har listats under 4.3.2.2.5. Detta avsnitt avser test, som användare eller operatör kör på färdigt system, t ex i samband med uppstart eller under drift för kontroll av systemets hälsotillstånd. S k *Built in Test* kontrollerar maskinvaran. Funktionsövervakning används för kontinuerlig kontroll av systemets funktioner och bygger bl a på maskinvarans BIT-funktioner.

1. Inbyggda testmöjligheter och självtest skall finnas {HML}.
2. BIT-funktioner ej avsedda att användas under drift skall ej kunna aktiveras under drift {HML}.
3. BIT-funktioner för kontroll av säkerhetskritiska delar skall betraktas som kritisk av samma grad {HML}.
4. Om det finns säkerhetsspärrar, som måste kunna brytas för att tillåta test eller underhåll, skall de konstrueras så, att de inte kan brytas oavsiktligt, inte kan lämnas i brutet tillstånd vid återgång till drift och inte kan brytas av programvarusystemet {HML}.

4.5.2.11 Implementation/Kod

Under implementeringen realiseras den detaljerade konstruktionen i valt programmeringsspråk. Av tidigare formulerade krav framgår, att kritisk kod skall följa säkerhetsprinciper enligt dokumenterade Konstruktionsprinciper (4.5.2.3.) samt uppfylla säkerhetsanvisningar i fastlagda Kodningsföreskrifter (4.5.2.7.). Verifiering av kritisk kod utförs mot programenhetens säkerhetskrav och mot dokumenterad konstruktion enligt 4.3.2.2.5.

196. Analys av inträffade incidenter har visat att < 15% av felen introducerats under konstruktion och implementation.

4.5.2.12 *Ändringar under produktion*

Detta avsnitt behandlar krav vid ändringar av programvarusystemet med ny utgåva innehållande

a) Enbart rättelser till föregående utgåva,

b) Andra ändringar enligt kravspecifikation

c) Ändringar i genererad kod p g a att en ny version av ett kodgenereringsverktyg använts på oförändrade källkodsdelar (se 4.4.2.2.)

d) Ändringar för att skydda mot nyupptäckta fel i använda kodgenereringsverktyg eller i integrerad COTS-produkt.

e) Enbart utvidgad funktionalitet i förhållande till föregående utgåva.

Här kan tänkas två fall:

1: Större system, som tas fram i form av ett antal kundutgåvor med successivt mer komplett funktionalitet enligt ursprungskraven.

2: Komplettering av kravmängden, p g a att denna visat sig ofullständig.

För krav vid ny COTS-version eller vid ändringar av delar som interagerar med en COTS-produkt, se 4.5.1.

Nedanstående krav är även aktuella under vidmakthållandefasen vid ändringar av färdigt och levererat system (se 3.3.3.1.).

1. Vid ändringar under produktion avseende rättelser eller tillägg till en föregående utgåva gäller följande krav {HML}:
   a) Ändringsanalys skall utföras, som utreder säkerhetseffekter på system och dokumentation[197].
   b) Ny säkerhetsgenomgång, SSPR, skall genomföras.
   c) Ändringarna skall införas i källkod, varifrån ny objektkod genereras[198] .
   d) Ändrad programvaruenhet skall lagras som ny version.
   e) Verifieringar skall utföras för att kontrollera
      i) ändringarnas korrekthet
      ii) att effekten på berörda delar är i överensstämmelse med utförd ändringsanalys
      iii) att ingen effekt kan konstateras i delar, som enligt analysen skall vara opåverkade[199].

197. Exempel: Innebär ny version för fler kunder, att död kod kan ha introducerats?
198. Detta innebär, att *patch*ar direkt i objektkod ej är tillåtet för säkerhetskritiska delar.
199. **Regressionstest** och sparade testresultat är viktiga hjälpmedel. Testfallen ses över för att kolla om dessa kan upptäcka ev. påverkan på kritiska delar vid omtest efter ändring. Detta kan kräva nya testfall. Dock bör en effektiv uppsättning testfall eftersträvas och rensning kan vara motiverad, där flera testfall kan ersättas av ett (utan att komplexiteten ökar eller beroenden till andra testfall införs).

f) Om maskinvaran ändras, skall kritiska delar testas om, för att kontrollera att ställda prestandakrav fortfarande är uppfyllda.

4.5.2.13 Dokumentation/Information

Detta avsnitt avser de speciella krav på dokumentation eller datorlagrad information, som system innehållande kritiska programvarudelar ställer.

1. Grundkrav för dokumentation enligt kap 5. skall vara uppfyllda {HML}.
2. Dokumentation relevant för bedömning av systemets säkerhet skall på begäran tillställas beställaren {HML}.
3. Varje kritisk konfigurationsenhet skall dokumenteras ned till minsta ändringsenhet[200] {HM}.
4. Varje kritisk konfigurationsenhet skall dokumenteras {L}.

4.5.2.13.1 Utveckling

Programvarudokumentation tas fram kontinuerligt under utvecklingen och omfattar bl a detaljerade kravspecifikationer, arkitekturbeskrivningar, dokumentation över ingående delsystem och komponenters struktur, gränssnittsbeskrivningar (mot interna delar, externa enheter och operatörer), testspecifikationer och testresultat, säkerhetsanalyser.

1. Utvecklingsdokumentationen skall beskriva {HML}:
   a) Alla förutsättningar, antaganden och begränsningar
   b) Deaktiverad kod, globala variabler och andra konstruktioner, som kan äventyra systemets robusthet.

4.5.2.13.2 Handhavande

Allmänna krav på innehåll i dokumentation riktad till personal, som hanterar system med kritiska programvarudelar. (Jämför även 4.5.2.8.).

Olika typer av dokumentation är aktuella för olika personalkategorier. Driftsdokumentation riktar sig till de, som ansvarar för installation, drift och hantering. Användar- eller operatörsdokumentation vänder sig till dem, som skall interagera med programvarusystemet under drift. Dokumentinformationen byggs upp successivt, främst under den detaljerade konstruktionen.

1. Handhavandedokumentation skall finnas för varje personalkategori och beskriva {HML}:
   a) Den kompetens och utbildning som krävs för säkert handhavande.

200. Se även Konfigurationsstyrning under kap 5.

b) Alla förutsättningar, för att få ta systemet i bruk och procedurer för verifiering av att dessa är uppfyllda [201].

c) De återställningar av säkerhetsspärrar, -kontroller eller -mekanismer, som krävs efter olika underhållsmoment samt de verifieringar och attester, som skall utföras vid kontroll av återställningen.

d) Säkerhetsaspekter, som måste vara kända, för att upprätthålla säker drift / operation[202].

e) Vilken hantering, vilka kommandon eller handgrepp, som är möjliga att utföra i olika systemmoder och driftlägen samt vilken effekt dessa har.

f) Begränsningar enligt utvecklingsdokumentation som påverkar handhavandet och hur dessa kan spåras mellan dokumentation över utveckling samt handhavande.

g) Beskrivningar, varningar och restriktioner.

h) Kritiska moment eller moment, som skulle kunna leda till säkerhetskritiska situationer eller tillstånd[203], hur sådana situationer kan undvikas samt hur man kan ta sig ur en sådan situation, om den skulle uppkomma.

4.5.2.13.3 Underhåll

Krav på innehåll i dokumentation riktad både mot personal, som skall underhålla och de som skall vidareutveckla kritisk programvara. Denna dokumentation används under fasen drift och vidmakthållande (se 3.3.3.1.).

1. Underhållsdokumentationen skall beskriva:

   a) Den kompetens och utbildning som krävs för att få tillstånd att utföra underhåll.

   b) Referenser till kritiska delar i utvecklingsdokumentationen.

4.5.2.13.4 Dokumentationslista

En sammanställning över de dokument och den information, som krävs av ett säkerhetskritiskt programvarusystem utöver vad som fordras av H SystSäk samt refererade standarder enligt kap 5.:

201. Exempel: Om möjlighet finns att åsidosätta ev. säkerhetsspärrar/-kontroller/-mekanismer (t ex vid test/underhåll), skall de kontroller, som skall ha utförts efter återställande eller senast före driftsättning, beskrivas samt av vem och var motsv. signering betr utförd återställning tecknas.
202. Exempel: Hur programvaran hanterar fel från externa delar (maskinvara, operatörer etc.). Hur länge ett system för kontinuerlig drift kan exekveras säkert/tillförlitlig utan mellanliggande omstart (jfr 6.9.5. och < Patriot>).
203. Detta inkluderar även möjligheter att kunna driva systemet mot och förbi dess dimensioneringsgränser.

Dokumentation	Avsnitt	Kritikalitet	Kommentarer
Felrapporterings-system	2.4. 3.4.1. 3.4.3. 4.4.1.2.	HML HML HML HML	För all programvara
Loggar	4.5.2.4. 4.5.2.8. -"-	HML H HM	Säkerhetsloggar, felloggar Presenterad information Operatörsåtgärder
Programvaru-säkerhetsplan	3.4.1.	HML	Del av SSPP
Analysrapporter: - Säkerhetsanalys - Formell verifiering - Granskning - Statisk analys - Objektkodsanalys - Dynamisk analys - Felanalys - Resursanalys	 4.3. 4.3.3. 4.4.2. 4.5.1. 4.5.2.5. 4.5.2.12. 3.3.3.1. 4.5.3. 4.3.2.1. 4.3.2.2.1. 4.3.2.2.2. 4.3.2.2.4. 4.3.2.2.5. 4.3.2.2.6. 4.3.2.2.7.	 H HML HML HML HML HML HML HML HM HML HML H HML H HML	Avseende: – Produktionsprocess – Programvara – Programvaruverktyg – Återanvänd programvara – Språk – Ändrad programvara – Ändrat programsystem – *Run-time*system – Formell specifikation – Programvarudokumentation – Källkod – Objektkod – Exekverbar kod – Fel i kritisk kod – Tids- & Minnesbehov
Konstruktions-principer	4.5.2.3.	HML	Grundläggande och säkerhetsinriktade
Kodningsföreskrift	4.5.2.7.	HML	(O)Tillåtna konstruktioner
Dokumentation av - Utveckling - Handhavande - Underhåll	4.5.2.13. 4.5.2.13.1. 4.5.2.13.2. 4.5.2.13.3.	HML HML HML HML	Programvarudokumentation All typer av dokument Dokument för utveckling ... handhavande ... underhåll

4.5.3 Måldatormiljö

För att systemsäkerheten skall kunna tillgodoses i ett kritisk målsystem, måste hela systemrealiseringen från applikationnivå ned till måldatorutrustning uppfylla ställda krav på pålitlighet.

En förutsättning för att programvarumässigt kunna undvika eller hantera konstruktionsfel i måldatormiljön är att aktuella listor över dessa görs tillgängliga (se 4.5.2.8.).

1. För måldatorutrustning i kritiska system gäller att {HML}
   a) Mekanismer skall finnas för upptäckt av fel under drift.
   b) Åtgärder skall vidtagas för under drift upptäckta fel.
   c) Aktuell lista över kända fel skall föreligga

För inkluderad programvaruprodukter gäller dessutom krav under 4.5.1.

4.5.3.1 *Operativ- och run-timesystem*

Ett säkert operativsystem (OS) skall följa samma stringens vid konstruktion och användning, som det säkerhetskritiska system, som nyttjar detta. Möjlighet måste finnas, att kunna spärra ut alla typer av konstruktioner, som kan ge ett icke-deterministiskt beteende. Exempel på detta är virtuellt minne, minnesåtervinning och delat minne mellan flera processer.

I dagens läge finns få kommersiella operativsystem, som är utvecklade efter och typcertifierade m a p någon säkerhetsstandard. Vissa målsystemet byggs på naken maskin (dvs en liten kärna med grundläggande *run-time*stöd i stället för ett komplett OS)[204].

I och med att även *run-time*systemet (RTS) ingår i den operativa programvaran, är det ej tillräckligt att certifiera detta isolerat (en produkt, som ”certifierats” fristående från applikationen betecknas av denna anledning som certifier**bar**[205], **typ**certifierad eller **pre**certifierad). Analys m a p säkerhet och verifiering (anpassad till den kritikalitetsnivå som gäller för motsvarande systemdel) måste därför utföras med RTS som integrerad del av systemet och med en processorlast representativ för applikationen. I detta arbete kan även felinjicering (6.8.7.) vara en lämplig teknik för kontroll av, att simulerade fel inte kan sprida sig till applikationen.

För att kunna utföra tillförlitliga resursanalyser (4.3.2.2.7.) och kunna garantera rätt funktionalitet i rätt tid, är det väsentligt, att ha tillgång till värsta exekveringtider för ingående primitiver samt att kunna utesluta eventuella realtidsprimitiver i OS-systemet/*run-time*kärnan, som kan leda till indeterministiskt beteende för de kritiska delarna eller som inte används. Om det senare inte är möjligt tillkommer uppgiften att visa, att onödig funktionalitet ej används eller kan skapa säkerhetsproblem. Önskvärt är även någon typ av

204. Exempel: (1) C-SMART, en Ada´ 83 kompilator för kompilering antingen m a p det kompletta språket eller m a p ett säkerhetskritiskt subset, där konstruktioner, som kan leda till indeterministiskt beteende eliminerats eller begränsats (se 4.5.2.6.). Kompilatorn är certifier**bar** mot nivå A enl DO-178B. Se <Aonix>
(2) <RAVEN>, en realtidskärna baserad på Ravenscars taskingprofil för Ada´95, <Ravenscar>, är certifier**bar** mot nivå A enl DO-178B. En väsentlig egenskap hos kärnan är dess deterministiska beteende.
(3) <OSE>, en programvarukärna, har **pre**certifierats mot SIL3-krav i del 1,3,6 i standardförslaget IEC 1508.

205. En produkt som är certifierbar mot viss kritikalitetsnivå enl någon säkerhetsstandard är anpassad för certifiering i applikationer med krav på att uppfylla denna kritikalitetsnivå.

stöd för analys av vald skeduleringsmetod. Några forskningsinstitut och företag har utvecklat verktyg för analys av egna realtidskärnor[206] eller av kommersiella microkärnor, men endast ett fåtal finns kommersiellt tillgängliga.

Vissa krav under 4.4.2.2. har bäring på operativ och *run-time*system.

1. Valt operativ- och *run-time*system skall utnyttja systemets resurser på ett predikterbart sätt {HML}.
2. Systemsäkerhetsanalyser och -verifieringar av operativ- och *run-time*-system skall göras med dessa integrerade i applikationen {HML}.
3. Verifieringarna skall göras med den stringens som motsvarar applikationens högsta kritikalitet {HML}.
4. Möjlighet skall finnas, att kunna blockera användningen av primitiver, som leder till osäkra konstruktioner {HML}.

Kravet på att isolera programvarudelar av olika kritikalitet[207] handlar egentligen om att förhindra negativa effekter från delar av lägre kritikalitet. Problemet kan lösas på olika sätt. Enklast kan vara fysisk separering. För att logisk separering skall vara acceptabel krävs garantier för att partitioneringen är robust mot otillåten interaktion mellan olika kritikaliteter. Forskningen är aktiv inom området, bl a inom NASA och RTCA SC182 (jfr 6.9.2.6., 6.1.2., <Vito>, <Rushby>). En förutsättning för att logisk separering skall vara möjlig är att bl a nedanstående krav är uppfyllda.

Om programvara av olika kritikalitet avses att exekveras på samma plattform, gäller att:

5. Grundkrav för operativ- och *run-time*system enligt kap 5. skall vara uppfyllda {HML}.
6. Stöd för isolerad exekvering skall ges av operativsystemet, genom att {HML}
   a) Maskinvaruresurser skall ej kunna användas av exekverande programenhet annat än via kärnan
   b) Kärnan och dess resurshantering skall ej kunna manipuleras från exekverande programenhet.
   c) Kärnan skall ha minimal funktionalitet och ej bryta fastställd separation.

206. Exempel: <SPRING>, MAFALDA <Fabre>, FINE <Kao>, BALLISTA <Kropp>, Xception <Carreria>.
207. Se krav under 4.5.1., 4.5.2.4. (Riskreduktion + fotnot) samt partitioneringsteknik under 6.1.2.

4.5.3.2. *Maskinutrustning*

Problem här kan (liksom för programvaru-COTS) vara bristande insyn i dokumentation, förändringar i nya versioner, som ej är kända för kund och odokumenterade särdrag[208].

1. Vid ny version av maskinvaruutrustning, som exekverar kritisk programvara, skall information om ändringar i förhållande till tidigare version samt resultat från integrationsanalyser och test av den nya versionen tillställas berörda programvaruutvecklare.

För viss säkerhetskritisk elektronik, som styrs av programvara, kan krav komma att ställas på t ex processorns konstruktion, så att det är möjligt att testa och mäta på programexekveringen, utan att modifiera eller instrumentera exekverbar kod. Detta kan t ex realiseras, genom införandet av ”avlyssnare” på intern busstrafik, som inte styrs av den exekverande koden. Mängden av test- och avlyssningselektronik får naturligtvis inte bli så stor, att apparatens totala tillförlitlighet/säkerhet blir lidande.

208. Exempel: (1) Tre funktionellt identiska processorer, som vid prov visade sig ha skillnader i tidsstegningen.
(2) Microprocessorn med kvarlämnade testinstruktioner, vars aktivering gjorde datorn oemottaglig för avbrott och där återgång till normalt mod fordrade återställning av processorn.

5. Grundkrav

Detta kapitel innehåller grundläggande krav[209] gemensamma för kritisk och icke-kritisk programvara. Numreringarna inom hakparentes hänvisar till motsvarande avsnitt i handboken.

5.1 Beställare

5.1.1 Personalkvalifikationer (tomt avsnitt)

5.1.2 Styrprocesser

5.1.2.1 [3.2.4. Kvalitetssäkring]

1. IEC 12207:s krav under ”6.3 *Quality Assurance Process* ”
   ”6.4 *Verification* -”- ”
   ”6.5 *Validation* -”- ”
   ”6.6 *Joint Review* -”- ”
   ”6.7 *Audit* -”- ”

   skall vara uppfyllda.

5.1.3 Materielprocessen

5.1.3.1 [3.3.2. Anskaffning]

1. IEC 12207:s krav under ”5.1 *Acquisition Process*” skall vara uppfyllda.

5.1.3.2 [3.3.3. Drift och Vidmakthållande]

1. IEC 12207:s krav under ”5.4 *Operation Process*” skall vara uppfyllda.
2. IEC 12207:s krav under ”5.5 *Maintenance Process*” skall vara uppfyllda.

5.2 Leverantör

5.2.1 Personalkvalifikationer (tomt avsnitt)

209. Grundkrav refererar till fastlagda standarder och kan ersättas av motsvarande krav i senare efterträdare till dessa.
Kravlistan är ej komplett (t ex ingår ej krav m a p IT-säkerhet).

5.2.2 Styrprocesser

5.2.2.1 [4.2.1. Projektplanering, ledning och uppföljning]

De procedurer för projektplanering, ledning och uppföljning som tillämpas skall vara väldefinierade, dokumenterade och väletablerade.

1. IEC 12207:s krav under ”7.1 *Management Process*” skall vara uppfyllda.
2. Planer för hela systemet och ingående delsystem skall föreligga och vara godkända av beställare och uppdragsgivare innan utvecklingen startar. Planerna skall åtminstone omfatta
   a) resurser, tider
   b) process/metodik
   c) utvecklingshjälpmedel/-miljö
   d) plan för programvaruutvecklingen och dess integration i det totala systemet.

   Speciellt skall framgå
   e) hur successiva programvaruleveranser avses att tas fram (funktionalitet),
   f) hur en för leveransen ny funktionalitet införs och kontrolleras,
   g) hur rättelser i förhållande till föregående leverans införs och kontrolleras[210].

5.2.2.2 [4.2.3. Kvalitetsstyrning]

1. En kvalitetsmanual skall föreligga, som beskriver företagets kvalitetssystem dvs kvalitetspolicy, organisation/roller/ansvar, regelverk (i form av företagsgemensamma handböcker och instruktioner) och verksamhetsmodell från kundkontakt till färdig leverans och eftermarknadsverksamhet.
2. Kvalitetssystemet skall åtminstone omfatta krav enligt ISO 9001 med förtydliganden enligt ISO 9000-3.

5.2.2.3 [4.2.4. Kvalitetssäkring]

1. IEC 12207:s krav under ”6.3 *Quality Assurance Process*”
   ”6.4 *Verification* -”- ”
   ”6.5 *Validation* -”- ”
   ”6.6 *Joint Review* -”- ”
   ”6.7 *Audit* -”- ”
   skall vara uppfyllda.

210. Det senare innebär bl a krav på verifiering av, att rättelser eliminerat avsett fel samt inte introducerat nya fel -speciellt inte sådana som kan försämra systemets säkerhet. En teknik, som underlättar detta är regressionstestning baserad på väl valda testfall.

5.2.2.4 *[4.2.5. Konfigurationsstyrning]*

Såväl aktiviteter under styr- och framtagningsprocess, som programvarans delar skall dokumenteras och konfigurationshanteras. Detta inbegriper bl a källkod, testprogram, testdata (invärden samt resultat), testomgivning, resultat från olika säkerhetsanalyser, säkerhetsloggar och säkerhetsutvärderingar samt övrig dokumentation från samtliga programvarufaser. Ändras en frisläppt systemdel innebär detta, att samtliga relaterade delar skall granskas för att avgöra, var dessa skall uppdateras.

1. IEC 12207:s krav under ”6.2 *Configuration Management Process*” skall vara uppfyllda[211].
2. Återtag skall kunna göras till en tidigare konfigurationsenhet, utan att resulterande system blir inkonsistent.

5.3.2 [4.3. Produktionsprocess]

1. IEC 12207:s krav under ”5.3 *Development Process*” skall vara uppfyllda.

5.2.3.1 *[4.3.1. Utvecklingsmodell]*

1. En systematisk, väletablerad, dokumenterad och av utvecklarna känd utvecklingsmodell skall användas.
2. Faser samt motsvarande indata, aktiviteter och resultat skall finnas dokumenterade.

Att praktiserade procedurer skall vara beskrivna och kända samt att samtliga roller och motsvarande uppgifter skall vara beskrivna och tilldelade ingår i ISO 9001-kraven.

5.2.3.2 *[4.3.2. Utvecklingsmetodik]*

1. En systematisk, väletablerad, dokumenterad och av utvecklarna känd framtagningsteknik för programvara[212] skall användas.

5.2.3.3 *Verifieringar [4.3.2.2.5. Dynamisk analys]*

Följande allmänna krav vid test av programvarusystem förutsätter – då det i de flesta fall ej är möjligt att i inledande utvecklingsskeden testa systemdelar i sin slutliga miljö– att verifieringarna utförs i successivt mer verklighetsnära test-

211. Bl a skall framgå hur systemets olika delar identifieras och lagras, vilka procedurer för ändringshantering, som tillämpas, hur historik över tidigare versioner kan tas fram.
212. Exempel: En namngiven strukturerad eller objektorienterad utvecklingsmetodik.

miljöer: initialt på processorer i simulerad omgivning hos leverantör, därefter i målmiljö hos leverantör, i målmiljö hos kund på mark och slutligen i taktisk miljö.

1. Alla krav skall verifieras.
2. Planer för testarbetet under programvaruutvecklingens olika faser skall föreligga.
3. Formella test skall utföras på fryst version/utgåva före frisläppning till intressenter utanför utvecklingsgruppen[213].
4. Vid ändringar av fryst version skall en ny, fryst version skapas och testas om.
5. Test skall utföras konsekutivt på minst tre nivåer: enhets-/komponent-, integrations- och systemnivå samt vara formella åtminstone på systemnivå.
6. Test på ny nivå skall startas först, då ingående delar testats på föregående nivå med godkänt resultat.
7. Varje testnivå skall inriktas mot aspekter, som inte kunnat testas på tidigare testnivå eller mot fel, som kan ha introducerats i den utvecklings/integrationsfas, som testen utgör en avslutning av.
8. Vald metrik för testtäckning och täckningsgrad samt stoppkriteria för test skall för varje testnivå anges i motsvarande testplan.
9. Två grundläggande, sinsemellan kompletterande angreppssätt skall tillämpas på de olika testnivåerna: kravtest resp strukturella test[214].
10. Den strukturella teknik, som skall tillämpas under utveckling är graftest[215].
11. Vid graftest skall testfallen minst en gång exekvera/ täcka in källkodens samtliga noder och mellanliggande vägavsnitt[216].
12. Den kravbaserade teknik, som skall tillämpas på programvarans komponent och integrationsnivå är
    a) Ekvivalenstest
    b) Randvärdestest
13. Den kravbaserade teknik, som skall tillämpas på enskild systemnivå är
    a) Syntaxtest

213. Detta innebär, att ev. ändringar måste införas i ny version, som fryses före omtest och frisläppande. (Separering i olika utgåvor med enbart rättelser eller enbart ny funktionalitet underlättar verifieringarna) .
214. De strukturella testen dominerar under de inledande enhets-/modultesten. Kravtesten överväger under integrations- och systemtest.
215. Se bilaga Begrepp: Test: Graftest.
216. 100% *statement coverage* + 100% *branch coverage*.

b) *Feature testing*
c) Stresstest
d) Volymtest
e) Konfigurationstest
f) Prestandatest
g) Återhämtningstest

14. Spårbarhet skall finnas mellan krav och testfall i båda riktningar.

5.2.4 Produktionsmiljö

5.2.4.1 Stödverktyg

5.2.4.1.1 [4.4.1.1. Konfigurationshanteringssystem], CM-system

1. Ett verktyg för konfigurationshantering av system och ingående delar skall användas under systemets hela livscykel som stöd för verksamhet enligt [4.2.5.].

5.2.4.1.2 [4.4.1.2. Felrapporteringsystem]

1. IEC 12207:s krav under ”6.8 Problem resolution process” skall vara uppfyllda.
2. Ett system för lagring av fel-/problem-/förbättringsrapporter skall föreligga och användas så fort en programvarukomponent färdigställts och frisläppts av ansvarig utvecklare oavsett om denna är kritisk eller ej.
3. Behöriga användare av systemet eller av enskilda komponenter skall kunna rapportera fel/problem upptäcka under dess utveckling eller efter leverans.
4. Ur kvalitetssystemet skall framgå vilken instans, som beslutar i ärendet, vilka som kan införa kompletterande information i systemet och hur informationen skall vidarebefordras/göras tillgänglig.
5. Behöriga användare skall kunna ta del av vilka fel som rapporterats och vilket status enskilt ärende befinner sig i.

5.2.5 Produkt

5.2.5.1 Standardprodukt (tomt avsnitt)

5.2.5.2 *Nyutvecklad programvara*

5.2.5.2.1 [4.5.2.1. Specifikation]

1. Krav på från närmaste systemnivå skall fördelas till programvaran på ett korrekt och motsägelsefritt sätt, så att inga krav, som berör programvaran tappas bort. Detta gäller även den successiva nedbrytningen av krav till underliggande programvarudelar.
2. Spårbarhet skall föreligga från systemkrav till motsv programvarukrav och ned till komponentnivå[217].
3. Programvarukraven skall kunna verifieras.

5.2.5.2.2 [4.5.2.13. Dokumentation/Information]

1. IEC 12207:s krav under ”6.1 *Documentation Process*” skall vara uppfyllda.
2. All dokumentation skall vara korrekt, komplett, motsägelsefri samt aktuell.
3. Systemets moder[218] och beteenden under normala och onormala betingelser samt möjliga övergångar mellan dessa skall vara beskrivna.

Vad som skall dokumenteras framgår av resp kapitel i IEC 12207.

5.2.5.3 *Måldatormiljö*

5.2.5.3.1 [4.5.3.1. Operativ- och *run-time*system]

I system, där oberoende exekvering mellan olika programenheter på samma plattform nyttjas, skall OS uppfylla nedanstående grundkrav:

1. Operativsystemet skall sörja för, att
   a) Ett standardiserat gränssnitt skall finnas mot exekverbara, oberoende programenheter.
   b) Resurser till oberoende programenhet skall allokeras oberoende av andra programenheters existens.
   c) Oberoende skall föreligga mellan programenheter och de resurser dessa exekverar på.

217. Kravet på spårbarhet gäller såväl källkod, testprogramvara som dokumentation, dvs kravspecifikationer, säkerhetsanalyser, konstruktions- och testbeskrivningar samt felrapporter.
218. Exempel: Uppstart/Operation/Underhåll, vart och ett med ett antal delmoder för olika kombinationer av betingelser (normal/extrem belastning, manuell/automatisk hantering, olika operationsfaser eller driftlägen).

d) En exekverande, oberoende programenhet skall inte kunna inkräkta på en annan programenhets exekvering.

e) Resurser tilldelade oberoende programenheter skall inte leda till konflikter eller krockar i minnes-, tids- eller avbrottshantering.

f) Delade resurser skall tilldelas oberoende programenheter, så att resursernas integritet bibehålls och programenheterna kan hållas separerade.

6. Bilagor

6.1 Begrepp

Begrepp i denna handbok och dess bilagor förklaras och exemplifieras. Skillnader och likheter mellan termer i refererade systemsäkerhetsstandarder beskrivs.

Begrepp inom tillförlighet och säkerhet definieras även i:

- H SystSäk, Försvarsmaktens handbok för Systemsäkerhet, <H SystSäk>.
- *Definitions for Hardware/Software Reliability Engineers*, <Meulen>[219].
- IEC 61508-7: *Overview of techniques and measures*, <IEC 61508>[220].
- *International Electrotechnical Vocabulary*, Ch 191: *Dependability and quality of service*, <IEC 60050>[221].
- *Dependability: Basic concepts and terminology*, <Laprie>.
- Tillförlitlighet - Ordlista, Svenska Elektriska Kommissionen, <SEK>.
- Pålitliga system, <Erikson>.

Innebörden hos en term kan skilja mellan olika källor – i synnerhet om dessa tillhör skilda tillämpnings- eller teknikområden.

För svenskan tillkommer att finna en adekvat översättning, vilket kan kompliceras av att samma ord används för flera engelska begrepp (t ex ”säkerhet”: *safety-security* samt ”fel”: *fault-error-failure*). För att undvika missförstånd ger handboken motsvarande engelska term (i *kursiv stil*). ”Systemsäkerhet” (*safety*) är ett centralt begrepp i handboken och förkortas ibland till enbart ”Säkerhet”. För *Security* har i stället benämningen ”IT-säkerhet” valts. Om ett och samma engelska begrepp förekommer i flera motsvarigheter på svenska har den term valts, som används inom den svenska programvarusfären. Detta gäller t ex ”tillförlitlighet” (*reliability*) och ”pålitlighet” (*dependability*). Sammansatta engelska begrepp, där fastlagd svensk term saknas, har härletts genom översättning av ingående delord (t ex ”riskkällenivå” som beteckning för *hazard level*). Några termer har i brist på bra alternativ lämnats oöversatta.

Exempel på begrepp med skiftande betydelse mellan olika systemsäkerhetsstandarder är **säkerhetsrelaterad** resp **säkerhetskritisk**. Denna handbok an-

219. En sammanställning definitioner i flera varianter och med anvisningar till över 90 olika källor. För t ex *reliability* ges nio olika definitioner, för felbegreppen *fault - error - failure* förekommer 7-15 varianter.
220. En beskrivning av begrepp och teknik för säkerhetsrelaterade programvarusystem.
221. En mycket omfattande ordlista, där kap 191 på ca 135 sidor behandlar begrepp inom området *dependability* på fyra olika språk.

vänder sig av termen säkerhetskritisk (eller kritisk) som en sammanfattande beteckning för delar som är kritiska för systemsäkerheten oavsett graden av kritikalitet.

Även kritikalitetsbegreppen skiljer (se 6.9.1.2.). Detta kan gälla både benämning och ingående faktorer. Några standarder använder samma benämning (SIL), men ger olika definition av **kritikalitetsnivåerna**. För att undvika låsning mot någon viss standard har handbokens kravtext rangordnats efter en 3-gradig skala, **kritikalitetsklasserna** {H}, {M}, {L}, se 1.7. Kravtextens termer har vidare begränsats till en logiskt konsekvent mängd.

Exempel på termer med delvis gemesamma attribut är **driftssäker** och **pålitlig**. Den första omfattar egenskaper väsentliga för maskinvara, den andra motsvarande för programvara. Ytterligare exempel på närbesläktade begrepp med gemensamma faktorer är **riskkällenivå** och **risk**. Det första är en delmängd av det andra, men sannolikhetsfaktorn avser olika händelser. Även **riskkälleanalys** och **riskanalys** skiljer. Om det i texten inte är väsentlig att ange vilken typ av systemsäkerhetsanalys som avses, används ett sammanfattande begrepp, **säkerhetsanalys**.

Begreppslistan i nästa avsnitt kan i vissa fall ge flera formuleringar med i princip samma innebörd för en och samma term. Vilken som är att föredra kan variera beroende på situation, ändamål eller tillämpningsområde. Bilagans avslutande avsnitt diskuterar mer ingående begreppen säkerhet, risk samt kritikalitet.

6.1.1 Begreppslista

Detta avsnitt innehåller begreppsförklaringar med hänvisningar till besläktade termer samt exempel. Några uppslagsord har samlats under gemensamt begrepp. Exempel: **Kod**, **Mod**, **Språk**, **Test**.

Ackreditering	Formellt beslut att ett visst informationssystem kan godkännas för drift med användning av en beskriven uppsättning säkerhetsfunktioner. Ett formellt driftgodkännande från IT-säkerhetssynpunkt[222]. Även: Formellt intyg från auktoriserad instans att en organisation eller person är kompetent att utföra en speciell uppgift.
Allvarlighetsgrad (*severity level, worst case consequence*)	Olyckas konsekvens rangordnad efter värsta möjliga skadeutfall (t ex Katastrofal, Svårartad, Marginell). Se även 6.1.4.

222. För systemsäkerhet motsvaras detta av beslutskedjan Säkerhetsutlåtande-Säkerhetsgodkännande-Beslut om användning, <H SystSäk>.

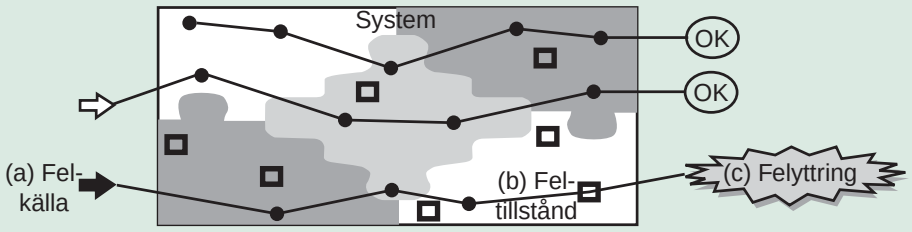
Applikation	Ett oberoende, exekverbart program, som utför ålagda uppgifter[223].
Arkitektur	Ett systems övergripande funktioner och delar samt interaktioner mellan dessa och med omgivningen.
• säkerhetsinriktad	– En arkitektur, som även beskriver de säkerhetsfunktioner och säkerhetsprinciper som möjliggör realisering av ett säkert system.
• tidsstyrd	– En systemlösning, där systemets aktiviteter initieras av en klocka efter ett i förväg uppgjort tidsschema.
Assertions	Satser före och efter ett kodavsnitt, som anger förutsättningar och villkor giltiga före resp efter kodens exekvering. Dessa används för kontroll av kodens korrekthet.
Beslutspunkt (*decision point*)	Punkt i en programflödesgraf, där styrflödet förgrenar sig. Exempel: *if*- och *case*-satser.
Bysantinsk felyttring	Generering av information i konflikt.
Cleanroom	En programutvecklingsmodell, som föreskriver formella metoder vid specifikation och konstruktion, funktionell verifiering m a p korrekthet, programutveckling utan exekvering samt statistiska systemtest baserade på representativa användarprofiler utförda av oberoende testgrupp, <Mills_2>.
Certifiering *certification* *conformity certification*	Vitsordande från en auktoriserad, oberoende instans (tredje part), att en produkt, process eller provning uppfyller föreskrivna krav, företrädesvis enligt en eller flera specificerade, fastlagda standarder[224].
Conformity assessment *Conformance -"-*	Bedömning (ej nödvändigtvis av 3:e part) att relevanta krav är uppfyllda[225]. Jfr "Certifiering", "Revision".
Datorsystem	System bestående av ett eller flera datorer med tillhörande programvara.
Deadline	Den maximala tidsram inom vilken resultat skall levereras.

223. En applikationen nyttjar -men är distinkt från- datorns operativsystem (OS).
224. Certifiering mot en ännu ej fastlagd standard benämns **villkorlig** certifiering (*conditional certification*). Certifiering m a p systemsäkerhet förutsätter utvärdering av hela systemet. En fristående säkerhetscertifiering av en produkt innan den integrerats med avsedd applikation brukar därför benämnas **typcertifiering**, **precertifiering** eller att produken är **certifierbar** och fordrar m a o en kompletterande certifiering i avsett system. (Inom IT-säkerhet: Formellt fastställande av resultatet från en evaluering m a p IT-säkerhet.)
225. Inbegriper bl a tekniker som sampling, test, inspektion, utvärdering, verifiering och försäkran om överensstämmelse, registrering, ackreditering och godkännande (*ISO/IEC Guide, 2nd edition*, 1994). Används t ex betr kompilatorvalideringar.

Defence-in-depth	Hierarkiskt skydd. Integrering av flera mekanismer, för att felyttringar i en mekanism skall kunna hindras av en annan att spridas till systemnivå.
Deterministisk	Med förlopp/beteende/effekt helt bestämd ur givna förutsättningar[226].
Diversifiering Diversitet	Olika sätt att realisera en viss funktionalitet eller att lösa en viss uppgift. Se 4.5.2.4.5..
Domängemensamma säkerhetskrav	Systemsäkerhetskrav specifika för kritisk programvara inom skilda applikationsdomän. Ett komplement till ”Generella säkerhetskrav” (se 6.4.1.4.).
Domänspecifika säkerhetskrav	Systemsäkerhetskrav unika för kritisk programvara inom en viss applikationsdomän. Ett komplement till ”Generella säkerhetskrav”. Se 1.6.
Driftsäker (*availability performance*)	Sannolikhet för tillfredsställande funktion under angivna betingelser vid en given tidpunkt (alt för viss tidsperiod) förutsatt att erforderliga underhållsåtgärder utförs av fastställd underhållsorganisation. Driftsäker innebär: Tekniskt system är funktionssäker och underhållsmässig. Underhållsorganisation är underhållssäker (jfr ”Pålitlig”).
Enkelfel	En felorsak, som ensamt kan leda till felaktigt systembeteende.
Estimeringsmodell	En tillförlitlighetsmodell tillämpad på ett programvarusystem under utveckling för prognosticering av sannolikheten för felyttringar. Statistiskt baserade skattningar görs ur insamlade data över felyttringar under test och drift. Jfr ”Predikteringsmodell”.
Evolutionär anskaffning / utveckling	En gradvis utveckling av kravmängden, där nya och modifierad krav implementeras i successiva systemversioner för bättre kravförståelse och återmatning från kund.
Fail active	Att vid felyttring vidtaga korrigerande åtgärder, som medger fortsatt exekvering (felåterhämtning).
Fail halt	Att vid felyttring stanna systemet (t ex nödstopp), för att kunna fortsätta exekvering först då faran undanröjts (jfr ”*Fail passive*”).

226. Exempel: En finit tillståndsmodell, där i visst ögonblick endast en typ av *input* är möjlig till ett tillstånd, där för varje giltig/ogiltig inmatning i/till ett visst tillstånd endast en tillståndsövergång är möjlig samt där oberoende föreligger mellan tillståndsövergångarna.

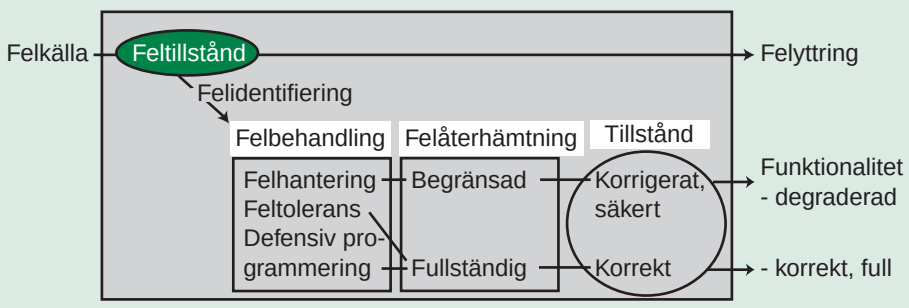
Fail operational	Att vid felyttring fortsätta i degraderad mod och med kända säkerhetsrisker (jfr ”*Fail soft*”).
Fail passive	Att vid felyttring stänga ned systemet (t ex nödstängning), för att kunna forsätta exekvering först efter ny uppstart av systemet.
Fail safe	Att vid felyttring övergå till ett säkert tillstånd (t ex rött ljus i alla riktningar). Olika varianter finns för om och hur återgång till operativt mod görs.
Fail silent	Att vid felyttring inte utföra någon felåterhämtning.
Fail soft	Att vid felyttring övergå till ett säkert tillstånd och fortsätta exekvera i degraderad mod (s k partiell nedstängning).
Fail stop	Att vid felyttring först signallera om fel och sedan stanna systemet.
Fara	Något, som kan leda till en olycka. Del av ett säkerhetshot[227].
Fast program (*firmware*)	Maskinvara med ej ändringsbar programvara (*read-only* software).
Fel	En sammanfattande term för följande begrepp:
(a: *fault*	**a: Felkälla**, felorsak, felbetingelser, t ex felaktiga indata eller användning av aktuell komponent under icke föreskrivna villkor.
b: *error*	**b: Feltillstånd** i en komponent (*Programming error* = konstruktionsfel).
c: *failure*)	**c: Felyttring** eller felbeteende hos komponent som följd av ett konstruktionsfel eller som effekt av ett föregående feltillstånd.



Felkällor och feltillstånd kan förbli okända pga att valda indata/testfall ej lyckas aktivera dem (för olika typer av fel i programvara: se 6.6.2.).

227. Exempel: Riskkälla, Riskkälleexponering, Annan omständighet. Se exempel under 6.1.4.

Felbehandling (*error processing*) — Åtgärder vidtagna efter felidentifiering (felupptäckt).



Felbeteende (*failure*) — Se ”Felyttring”.

Felbeteende med gemensam felorsak (*common cause failure*) — Samtidiga felyttringar hos flera (skenbart) oberoende komponenter härrörande från en gemensam felkälla (en indikation på beroende, som i händelse av redundant komponentrealisering gör vald redundans verkningslös)[228].

Feleffekt — Konsekvenser av en specifik felmod för system eller omgivning.

Felhantering (*error handling*) — Teknik, som syftar till att överföra ett system från ett felaktigt till ett korrigerat tillstånd.

Felkälla (*fault*) — Felorsak. Jfr ”Fel”.

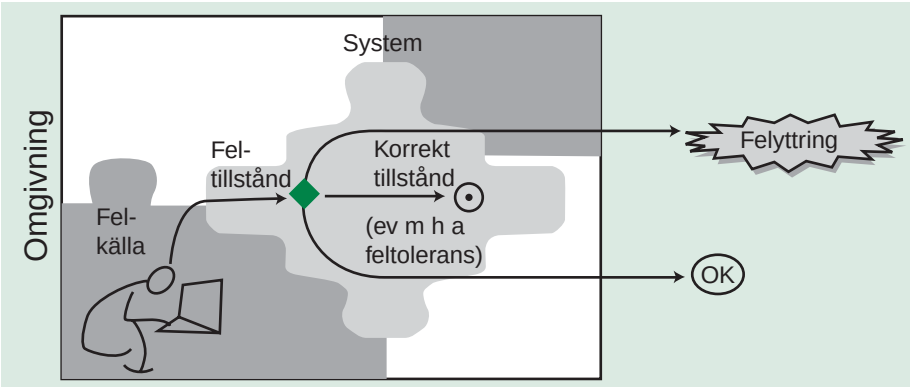
Felmod (*failure mode*) — De sätt en komponent kan fela (*fail*). Felsätt[229].

Felsituation — En kombination av faktorer, som leder till feltillstånd eller felyttringar.

Felsäker — Se ”*Fail safe*”.

228. Exempel på gemensam felorsak: Kortslutning-avbrott-brand, kommunikations- och funktionsavbrott. *Common cause failure:* enl <IEC 812>, <Smith> synonym till *common mode failure*. <Leveson> skiljer dock på dessa begrepp.
229. <SEK> anger ***fault*** *mode*, som motsvarande engelska term. <Leveson> m fl använder termen ***failure*** *mode*.

Feltillstånd (*error*) — En avvikelse från förväntat tillstånd. Kan orsaka en felyttring i en annan del av systemet. Jfr ”Fel”.



Feltolerans (*fault tolerans*) — Inbyggd systemegenskap, som möjliggör, att avsedd funktionalitet kan utföras trots ett begränsat antal förutsedda fel i program- eller maskinvara.

Felyttring (*failure*) — Ett beteende eller resultat, där tillförlitligheten ej är tillgodosedd[230].

Felåterhämtning (*error recovery*) — Att överföra programmet från ett felaktigt tillstånd till ett korrigerat, från vilken normal och säker drift kan fortsätta.

Firmware — Se Fast program.

Formell metod — En matematiskt baserad teknik för att beskriva och dra slutsatser om system och dess egenskaper. En formell metod klassas efter den formalism den baserats på, t ex modell, algebra, process algebra, logik eller nätverk.

Frysning, fryst version — Att skapa en skrivskyddad version. Se ”Formella test”.

Funktionalitet (*feature*) — Ett sammanfattande term för beteenden och tjänster hos ett system, delsystem eller en komponent.

Funktionellt säker — Där funktionaliteten ej bidrar till olycka. (Skilj från ”Funktionssäker”).

Funktionssäker (*reliable*) — Sannolikhet att begärd funktion/tjänst utförs tillfredsställande under föreskriven tid[231] och specificerade omgivnings- och driftsförhållanden. Mått på förväntad frihet från felyttringar under givna villkor (för maskinvara ofta skattad ur data över slumpmässiga

230. Notera: (1) En felyttring behöver ej vara säkerhetskritisk.
(2) Ett felbeteende hos en komponent kan utgöra en felkälla till en annan. Se även ”Felbeteende” Jfr ”Felbeteende med gemensam felorsak”, ”Gemensam felyttring”.

231. Tid är en naturlig måttenhet i fallet kontinuerlig drift. Alternativa måttenheter kan vara: per användningstillfälle, per visst antal transaktioner.

	fel (uttryckt i MTBF, MTTF). Typiskt krav för ett system av högsta kritikalitet kan vara att antalet felyttringar/tim skall understiga 10^{-9} (< 10^{-5}/år, MTTF = 115* 10^{3} år). För programvara används i stället begreppet tillförlitlig.
Gemensam felorsak (*common cause fault, common mode fault*)	Enstaka felkälla eller händelse, som kan orsaka felyttringar hos flera (oberoende) komponenter eller kommunikationskedjor[232].
Gemensamt feltillstånd (*common mode error*)	Felmod (i samma eller i redundanta strukturer), vars orsak kan ligga i olika felkällor och som resulterar i samma typ av feltillstånd (ev samtidiga)[233].
Gemensam felyttring (*common mode failure*)	Samma felbeteende hos flera komponenter. Exempel: ”fastnad i öppet läge”, <Leveson>.
Generella säkerhetskrav	Systemsäkerhetskrav giltiga för kritisk programvara oavsett applikationsdomän. Ett komplement till ”Grundkrav”. Se 1.6.
Generisk enhet	Mall för en programenhet, där tilldelning av aktuella parametervärden (instansiering) resulterar i en ordinär, icke-generisk enhet.
Graceful degradation	En stegvis reduktion av systemets funktionalitet vid identifierade felyttringar, där de väsentligaste funktionerna hålls operationella.
Grundkrav	Allmänna krav för all typ av programvara, som är obligatoriska krav för kritisk programvara. Se 1.6. samt kap 5.
Haveri	Ett fortskaffningsmedel, som kraschat/blivit vrak. Maskin/motor, som blivit manöverodugligt och ej kan utföra sina ordinarie funktioner[234].
Heap	En global area för dynamisk allokering och minneshantering av objekt.
Hindermängd (*cut set*)	De logiska relationerna i ett reducerat felträd som visar en topphändelse och de primära händelser, som leder till denna.

232. Gemensam felorsak innebär att beroende föreligger, vilket påverkar sannolikhetsberäkningarna och gör eventuell redundans meningslös. Se även ”Felbeteende med gemensam felorsak” ovan.
Exempel: Blixtnedslag (orsakar kortslutning-avbrott-brand), grävning eller annat underhållsarbete, vilka oavsiktligt skadar parallellt dragna kablar – parallellt lagda hydraulikssystem etc.
233. Exempel: Motorstopp, där orsaken bl a kan vara bränslebrist, fel i tändspole eller fel i brytarkontakt.
234. Haveri förekommer som svensk term för *failure*. Så ej i denna handbok.

• minimal	– Ett reducerat felträd med den minsta uppsättningen av de basala villkoren för topphändelsens inträffande. En hindermängd som vid ytterligare reduktion inte kan leda till samma topphändelse.
Incident	Se ”Tillbud”. (Inom IT-säkerhet: Händelse som kan leda till sekretessförlust, minskad kvalitet eller tillgång till data i ett IT-system.)
Inkrementell utveckling	En etappindelad utveckling och leverans i form av operativa systemversioner med en successivt större mängd av systemkraven implementerade.
Integratör	Person, som bygger ihop systemets delar och samtestar dessa.
Isolera	Göra oåtkomlig. Avskärma. Skydda mot påverkan från omgivande delar. Olika grad av skydd kan åstadkommas, s.k. fysisk, logisk eller temporal separering. Fysisk separering åstadkommes genom användning av olika processorer (skydd både av processhantering och adressrymd), logisk t ex genom inkapsling, separerade adressrymder eller separata processer, temporal genom att förhindra tidsmässig påverkan (åtkomstkontroll, synkroniseringar, avbrottsskyddad exekvering).
IT-säkerhet (*security*)	Egenskap eller tillstånd som innebär skydd mot okontrollerad insyn, förlust eller påverkan; oftast i samband med medvetna försök att utnyttja eventuella svagheter[235].
Kod	En sammanfattande term för data, källkod, objektkod samt exekverbar kod.
• deaktiverad	– Exekverbar objektkod/data konstruerad enbart för vissa målkonfigurationer (t ex kod, som aktiveras genom parameterval vid konfigurering av systemet) eller ej avsedd att exekveras/användas (t ex del av en återanvänd komponent). Funktionalitet avsedd för andra driftslägen eller systemmod (t ex intrimningar och självtest före start eller vid underhåll) betraktas som deaktiverad utanför dessa lägen/mod.
• död	– Exekverbar objektkod/data, vilken p g a konstruktionsfel ej kan exekveras eller användas i avsedd målkonfiguration. (Undantag: inkapslade variabler). Jfr ”Deaktiverad kod”.

235. IT-Säkerhet omfattar bl a begreppen **konfidentialitet** (sekretess), **riktighet** (integritet, dvs ingen obehörig ändring samt informationskvalitet) och **tillgänglighet**. Se även <ITS 6> samt 6.1.2. En utredning om samordning av begrepp för sambands- och informationssystem pågår inom FM.

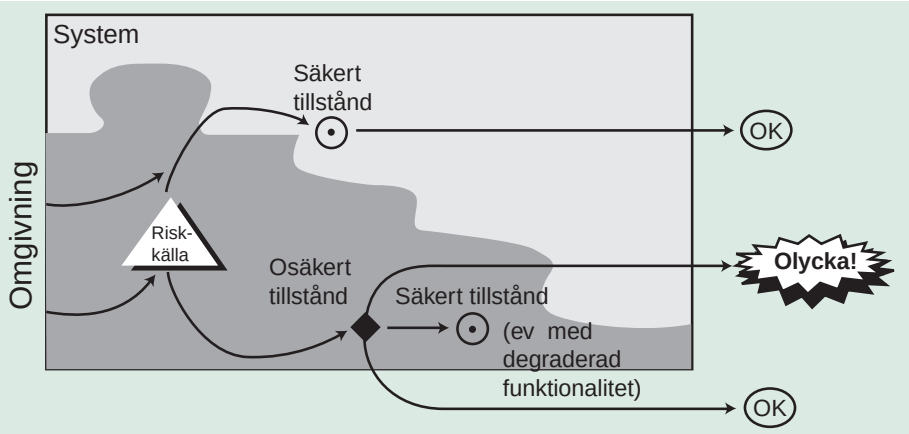
• *Easter egg*	– Odokumenterad kod avsiktligt inlagd av programmeraren. Koden kan varken klassificeras som död eller deaktiverad (men ligger på gränsen mot det senare, då det fordras kunskap om vilka sekvenser, som aktiverar den). Finns i vissa COTS-produkter[236].
• exekverings-	– Ett datorprogram vars instruktioner direkt kan utföras av en viss dator. Exempel: exekverbar objektkod, maskinkod.
• käll-	– Ett datorprogram uttryckt i det språk det programmerats i.
• objekt-	– Ett datorprogram (med relokerbar information eller direkt exekverbar) genererat ur källkod skriven i ett programmeringsspråk.
Konfigurationsenhet (*configuration item, CI*)	En samling maskin- eller programvaruelement hanterade som en enhet i konfigurationsstyrningen.
Konsekvens	Utfallet vid en olycka. Konsekvensskattningar baseras på värsta möjliga skada under de mest ogynnsamma förutsättningarna (konsekvensens allvarlighetsgrad). Se 6.1.4.
Konstruktionsrestriktion	Inskränkning p g a krav på konstruktionen från överliggande och angränsande nivåer (t ex arkitektur- och systemsäkerhetsbegränsningar).
Kooperativ preemption	Frisläppning endast under processorernas tidsluckor, där tidsluckorna motsvarar blockeringsfaktorn.
Kravspecifikation	Ett dokument, som beskriver krav på en produkt under anskaffning. Kravspecifikationen kan avse FMVs ursprungliga Tekniska specification (se detta ord) eller detaljeringar och vidareutvecklingar av denna utförda under produktionsprocessen.
Kritikalitet	Ett relativt mått på den verkan ett krav, modul, annan enhet eller ett fel (felkälla/-tillstånd/-yttring) har på systemsäkerheten. Se 6.1.5., 6.9.1.2.
Kritikalitetsklass -grad	Kritikaliteten rangordnad efter konsekvens och sannolikhet (rimlighet) för olycka. I denna handbok för gradering av programvara och dess säkerhetskrav i hög, medel och låg kritikalitet, dvs {H}, {M}, {L}. Se 1.7.
Kritikalitetsnivå	Kritikaliteten rangordnad enligt någon standard (t ex för STANAG 4452 i väsentlig/måttlig/låg insats för säkerhetsbedömning).

236. Exempel: Spelprogram (Atari) från 70-talet, tidiga system till Macintosh och Amigas, Windows95 och NT, Excel97.

Kritisk	Se säkerhetskritisk.
Kritisk kärna	Den del i en enhet som är mest kritisk.
Kvarvarande risk (*residual risk*)	Identifierade risker som förklarats tolerabla för systemet och därför tillåts kvarstå samt risker som förblivit oupptäckta och därför ej eliminerats eller reducerats. Den totala risk användare och omgivning utsätts för.
Kärna (*kernel*)	En central komponent i en processors operativsystem ansvarig för fördelningen av processorns resurser till olika processer.
Låsning (*deadlock*)	Den situation då datorns exekvering hängt sig p g a att någon enhet eller process väntar på resurs reserverad för en annan, vilken i sin tur är låst i direkt eller indirekt väntan på resurs tilldelad den förste.
Minnesåtervinning (*garbage collection*)	Program startad under exekvering av OS för insamling av outnyttjat minne.
Mod (*mode*)	Ett systems olika operations- eller användningssätt under olika tidsskeden[237].
• degraderad –	– Ett systemmod, där begränsningar införts i funktionalitet, prestanda, noggrannhet eller feltolerans (t ex graden av redundans).
• *mode confusion*	– Situation, där det är oklart i vilket mod ett system befinner sig i.
Multipla felkällor	Flera samtidiga felkällor, vilka antingen är (a) oberoende, leder till olika feltillstånd och skilda felyttringar eller är (b) relaterade, leder till liknande feltillstånd och felyttringar av samma typ/felmod (*common mode failure*).
Oberoende	
• händelser	– Där en händelse är opåverkad av om en annan händelse inträffar.
• programdelar	– Delar, som ej kan påverka varandra (varken direkt eller indirekt)[238].

237. Exempel: **Systemmod**: Uppstart/Normal/Degraderad/Under nedstängning, **Datormod**: Normal/Överlastad,
Normala flygmod: Stigning/Transport/Landning, **Styrmod**: Manuell/Automatisk, **Operativa mod**: Jakt/Attack/Spaning.
238. Exempel: Ingen påverkan under exekvering via data, styrflöde eller delad resurs (för olika realiseringar, se ”Isolera”).
Inget beroende p g a gemensam bakomliggande mekanism, komponent, felorsak, tillverkningsprocedur (*Diversity*).

• utvärdering	– Objektiv utvärdering utförd av person (eller verktyg) fristående från den, som utvecklat granskad produkt.
Olycka	En oönskad (serie av) händelse(r), som orsakar icketolerabel skada på person, egendom eller miljö.
Osäkert tillstånd	Ett tillstånd, som inte är säkert.



Plattform	Datorutrustning med maskinvara, kärnprogramvara (operativ system, maskinvarugränssnitt etc) och programvarustöd för applikationer.
PLC (*Programmable Logic Controller*)	Specialbyggd processor för styrning av industriella applikationer inom kategori 4 enligt <prEN954-1> [239].
Predikterbar	Med förutsägbart utfall, effekt eller beteende.
Predikteringsmodell	En tillförlitlighetsmodell tillämpad på ett programvarusystem under utveckling för prognosticering av sannolikheten för felyttringar. Jämförelser mellan parametriserade egenskaper i aktuell programvaruprodukt och utvecklingsmiljö samt i ett liknande, färdigutvecklat system används vid predikteringarna. Jfr ”Estimeringsmodell”.
Preemption	Teknik, där en process av högre prioritet omedelbart får ersätta en exekverande[240].
Prioritetsinversion	En lägre prioriterad process nyttjar en resurs krävd av en högre prioriterad process.

239. Standard, handböcker: <IEC 1131-3> (behandlar ej säkerhet), <PLC-Guide>. Definition av säkerhetskriteria för PLC:<SEMSPLC>.
240. *Non-preemption*-teknik innebär att en process tilldelas i förväg schemalagda tidpunkter för exekvering.

Programvaruprodukt	All programvara och dokumentation för en produkt
Program(varu)system	eller ett system (t ex dataprogram, fast program (*firmware*), data, testrutiner inkl testfall, dokumentation på alla nivåer: t ex av krav, analysresultat, konstruktion, test, drift, underhåll). Avser en viss konfiguration av produkten.
Programvarusäkerhet	Egenskap hos programvara och programvaruhantering[241], som bidrar till systemsäkerheten.
Progressiv anskaffning	En evolutionär anskaffning av ett komplett program- och maskinvarusystem kombinerad med en inkrementell systemplanering, -finansiering, -kontraktering och -utveckling.
Provning	Undersökning under olika betingelser av egenskaper hos materiel, system, konstruktionselement och konstruktioner samt av funktionalitet, prestanda, funktionssäkerhet, underhållmässighet hos materiel eller metoder för materielens handhavande.
Pålitlig (*dependable*)	Förmåga hos ett system att utföra sina tjänster på ett tillfredsställande sätt. Termen inrymmer egenskaperna tillförlitlig, systemsäker (person-, egendom-, miljösäker), IT-säker (konfidentiell, okränkbar, tillgänglig), underhållsmässig (reparer- och utvecklingsbar), <Laprie>. En gradering av förmågan görs kvalitativt i termer av rimlighet snarare än sannolikhet. Jfr ”Driftsäker” (**sannolikheten** för motsv förmåga hos maskinvara: inkluderar även underhållssäker (–förmåga) men ej IT-säker. Pålitlighet ställer krav på hela det realiserade systemet (från applikationsnivå till basprogramvaror/*middleware*, operativ- och *run-time*system och ned till fasta program (*firmware*) och maskinvara).
Race-condition	Kapplöpningsföreteelse, t ex där det vid två meddelanden från oberoende källor till samma destination inte går att avgöra i vilken ordning dessa två meddelanden anländer[242].
Reaktivt system	Ett system, som reagerar på händelser från sin fysiska omgivning.
Realtidssystem	Ett system, vars korrekthet beror både av beräkningarnas logiska resultat och tidpunkt då resultaten levereras. Klassning efter tidskravets stränghet:

241. Exempel: Egenskapen hos programvara att (1) inte orsaka eller bidraga till att systemet når ett osäkert tillstånd, (2) kunna upptäcka och vidtaga åtgärder vid osäkert tillstånd samt (3) bidra till att lindra skada om olycka inträffar.
242. Denna företeelse var en orsak till Therac-25-olyckan, se <Leveson>.

• Hårda –	– Tidskravet är absolut med katastrofala konsekvenser om det ej uppfylls.
• Fasta –	– Tidskravet är väsentligt, då resultat levererade senare är värdelösa.
• Mjuka –	– Tidskravet är viktigt, men inte väsentligt för systemets funktion.
Redundans	En viss funktionalitet eller uppgift realiserad i flera enheter. Syftet är att vid felfunktion eller feltillstånd i en/flera enheter kunna erbjuda feltolerans, genom att låta en annan, oberoende enhet ta över. Redundans kan åstadkommas genom identiska eller diversifierade enheter (se 4.5.2.4.5.). För programvara: Flera oberoende sätt att utföra viss funktion eller uppgift.
Resurs	En ingångsfaktor till en process[243]. En entitet, som applikationen behöver för att utföra sina funktioner.
Revision (*audit*)	En systematisk och oberoende undersökning, för att bedöma om procedurerna m a p kraven överensstämmer med planerade arrangemang, är effektivt implementerade och i övrigt lämpliga för att uppnå specificerade mål. (Att skilja från kvalitetsprocessens inplanerade granskningar, *reviews*, vilka huvudsakligen är inriktade mot själva produkten).
Risk	Sannolikhet att en olycka inträffar samt dess konsekvens[244]. Riskkällenivå (se nedan) kompletterad med sannolikheten för att riskkällan leder till en olycka samt riskkälleexponeringen[245].
Riskanalys (*risk analysis*)	En systematisk identifiering av potentiella olyckor för bestämning av deras sannolikhet och konsekvens. (I första hand inriktad mot att finna tidigare ej kända konsekvenser, jfr ”Riskkälleanalys”).

243. Exempel: Person, verktyg, exekveringstid i processor, minne, diskutrymme, kommunikationslänkar.
244. En **övergripande formulering**, där sannolikheten (**sh**) inkluderar dels sh för att en riskkälla föreligger, dels sh att denna leder till olycka. Skattning av den senare sh:en kräver att systemets inre struktur är bestämd, så att mer ingående analys kan utföras (se 6.1.4.). ”Rimlighet” används ofta i stället för ”sannolikhet” (i synnerhet om endast kvalitativ bedömning är möjlig, t ex för programvara.
245. En **detaljerad formulering** enl <Leveson>., där sannolikheten avser **från riskkällan**, vilken befinner sig i början av den händelse- eller tillståndskedja, som leder vidare mot olyckspunkten. Beräknas sannolikheten från en senare punkt i denna kedja, är det väsentligt att explicit ange varifrån. Leveson använder termen ***likelihood*** (rimlighet). Se 6.1.4.

Riskkälla (*hazard*)	Ett tillstånd eller en serie omständigheter hos ett system, vilka tillsammans med andra förhållanden i dess omgivning kan leda[246] till en olycka. Ett villkor eller en förutsättning för en olycka.
Riskkälleanalys (*hazard analysis*)	En systematisk identifiering av riskkällor och bedömning av motsvarande riskkällenivåer. (I första hand en inriktning mot att finna tidigare okända säkerhetshot, se 6.1.4.).
Riskkälleexponering	Uttryck för i vilken grad person, egendom eller miljö är utsatt för en viss riskkälla. Detta kan t ex mätas i styrka/dos/volym, varaktighet/tid, närhet till riskkällan, antal exponeringstillfällen.
Riskkällenivå (*hazard level*)	Sannolikhet att en riskkälla föreligger samt konsekvensens allvarlighetsgrad[247].
Robust	Med förmåga att även under onormala omständigheter kunna vara operativ och utföra föreskriven funktionalitet.
• datastruktur	– Struktur, som garderar mot fel, genom att nyttja redundant information[248].
• konstruktion	– Konstruktion, som tillfredsställande kan hantera oväntade signaler, ändringar i omgivning eller fel i dess modell.
• partitionering	– Uppdelning av programvara efter kritikalitet, där del av lägre kritikalitet inte negativt kan påverka säkerheten vid interaktion med del av högre kritikalitet.
Safety case	En dokumenterad argumentation för att ett system är säkert i föreskriven användning och omgivning[249].
Safety Integrity Level *Software Integrity Level*	Uttryck för den risk programvaruprodukten bidrar till vid användning av det system, där den är avsedd att ingå.

246. Exempel ”andra förhållanden”: Riskkälleexponering. Se exempel under 6.1.4.
Kommentar till ”kan leda till”: Levesons definition har i stället skrivningen ”*that ... will lead inevitably to an accident (loss event)*”. Ordet ”*inevitably*” har orsakat en del missförstånd och somliga har hävdat, att formuleringen ”*could lead to*” skulle vara mer relevant. Leveson har därvid förklarat, att inget i hennes definition uttrycker att det föreligger visshet om en olycka, utan enbart, att det finns åtminstone **en** väg mellan riskkälla (vid systemgränsen) och olycka, där yttre händelser blir avgörande för om den verkligen inträffar.
247. Riskkällenivån (till skillnad från risk) kan fastställas utan detaljkunskap om och analys av det aktuella systemets inre struktur. Den beaktar händelsekedjan till en olycka i dess ingång och utgång till/från systemet. Skillnaden mellan begreppet riskkällenivå och riskklass ligger i vad sannolikheten avser. Se 6.1.4.
248. Exempel: Checksummor, dubbellänkade listor.
249. Se t ex <DS 00-55> del 2, B-2. En motsvarighet till H SystSäk:s SCA (*Safety Compliance Assessment*).

Samresident programvara (*co-resident software*)	Multipla programvarupaket i samma laddmodul eller enhet, var och en avsedd att exekveras på en specifierad delmängd av värdprocessorn under vissa betingelser.
Scenario	Beskrivning av ett möjligt händelseförlopp.
Sekundär felkälla	Felkälla till ett system (komponent) utanför detta (denna)[250].
Sekundär felyttring	Felyttring p g a förändringar i omgivning eller användning i strid mot vad som specificerats[251].
Semiformell metod	En blandteknik, som använder sig av ett strukturerat, naturligt språk, diagram, matematiska notationer och genereringsverktyg för specificering av krav och överföring av dessa till kodfragment.
Skedulerbart schema	En definition av den ordning systemets processer får tillgång till delade resurser, som möjliggör, att varje process kan utföra sina uppgifter inom föreskriven tidsrymd. Schemat kan vara statiskt/dynamiskt, *preemtive/non-preemtive*.
Skyddsanordning	Mekanism eller funktion avsedd att lindra eller förhindra skador vid en förestående olycka[252].
Skyddssystem	Ett system, som efter upptäckt av ett potentiellt osäkert tillstånd, kan överföra systemet till ett mer säkert tillstånd.
Sneak circuit analysis	En teknik att identifiera dolda grenar, vilka orsakar oavsiktlig funktionalitet eller hindrar avsedd funktionalitet p g a att alla komponenter förutsatts fungera korrekt[253].
Spiralformad livscykel	En riskorienterad modell, där varje spiralslinga representerar en fas och passerar 4 kvadranter: 1) fasens målsättning, begränsningar, projektrisk, 2) riskanalys & -reduktion, 3) val av utvecklingsmodell, 4) planering av nästa fas. Avståndet från origo är ett mått på nedlagt arbete <Boehm>.

250. Exempel: Direkt solljus på termostat för reglering av rumstemperatur – en förutsättning missad vid konstruktion av termostaten. En ogiltig inmatning till ett system, som byggts för att endast hantera specade inmatningssekvenser.
251. Exempel: Användning utanför avsett/specat användningsområde. Handhavande i strid mot angivna föreskrifter. Se <Patriot>, 6.9.5.
252. Exempel: Avbrottshanterare vid farlig process (t ex Nödstopp, ABS-bromsar, flygplansstabilisator), återstartshinder efter nödstopp innan fel åtgärdats, larmutlösare, åtkomsthinder till farligt område (skyddskåpa, bommar, viadukt).
253. Oklart om tekniken i praktiken lyckats finna några fel.

Språk

• programmerings-	– Språk med exakt syntax och semantik, som efter översättning kan avkodas och exekveras av en dator.
• högnivå-	– Språk oberoende av datortyp med abstraktioner för objekt/moduler, subprogram, operationer, datatyper, sekvenser, kommunikation m m. Statisk kodanalys (t ex typkontroll) utförs vid översättning (kompilering). Avancerade högnivåspråk har stöd i *run-time*-systemet för olika typer av dynamisk kodanalys under exekvering.
• lågnivå-	– Språk för viss datortyp som medger direkt manipulering på bitnivå av register, villkor, grenar, kanaler, diskar etc. Ingen kodanalys tillhandahålls. Exempel: assembler- och maskinspråk.
• assembler-	– Maskinnära språk för viss datortyp med syntax i form av symboliska namn för instruktioner, register och minnesadresser, vilka direkt motsvarar och kan översätts till maskinspråk.
• maskin-	– Binärkodade maskininstruktioner för viss datortyp med absolutadressering av minne och register direkt avkodningsbara för exekvering.
Strukturerad programmering	En programmeringsteknik lanserad i slutet av 1960-talet, där konstruktion utförs *top-down* och koden skrivs utan hoppinstruktioner (*go-to*-satser).
System	En samling komponenter organiserade för att utföra en eller flera speciella uppgifter[254].
Systemarkitektur	Se ”Arkitektur”.
Systemspecifika säkerhetskrav	De unika systemsäkerhetskrav, som det övergripande systemet ställer på ingående programvara, för att kunna utföra ålagda uppgifter på ett säkert sätt i avsedd omgivning och användning. Se 1.6.

254. Komponent kan även avse operatör, ett delsystem eller ett fristående system. Det senare innebär att vi har ett system av system. Där betydelsen inte framgår av sammanhanget görs förtydliganden i texten genom formuleringar av typ:

”Övergripande systemnivå”, ”det totala”/”kompletta” systemet	för systemet högst upp i systemhierarkien.
”Enskilt system(nivå)”	för ett autonomt system nedanför högsta nivån.
”Närmaste system(nivå)”	för närmast ovanliggande system, där aktuellt programvarusystem ingår.

Obs: Systemsäkerhet, riskkälla, konsekvens etc avser alltid systemet på högsta nivån i avsedd omgivning och användning.

Systemsäkerhet[255] (*Safety*)	Egenskap hos ett system att under föreskrivna villkor och i avsedd omgivning ej oavsiktligt skada personal, egendom eller miljö. (En säker konstruktionen är inriktad mot att undvika olyckor, en tillförlitlig mot att undvika fel. Balans mellan säkerhet och funktionalitet måste eftersträvas, där säkerhet kan komma att prioriteras på bekostnad av funktionalitet: övergång till säkert tillstånd kan fordra en degradering av funktionaliteten och därmed av tillförlitlighet och tillgänglighet).
Systemsäkerhetsverksamhet (*System safety*)	Det totala arbete som bedrivs för ett visst system från studiefas till avveckling i syfte att identifiera och kvantifiera risker, eliminera dessa eller reducera dem till en tolerabel nivå.
Säker, Säkerhet	I svenskan en samlingsterm för olika typer av säkerhetsbegrepp (se 6.1.2). I detta dokument avses, där inget annat anges, systemsäkerhet.
Säkerhetsanalys	Sammanfattande begrepp för olika typer av systemsäkerhetsanalyser vilka utgående från interaktion mellan system, användare och omgivning studerar dels händelsekedjor mellan riskkälla och olycka dels inträffade olycksscenarier för att identifiera möjliga riskkällor, olyckshändelser, kritiska komponenter samt värdera värsta utfall och motsvarande sannolikheter.
Säkerhetshot	Sammanfattande begrepp för de förhållanden i system och omgivning, vilka tillsammans med riskkälla kan leda till olycka[256].
Säkerhets**kritisk**	System/delsystem/komponen/funktion, där felyttringar eller interaktion mellan samverkande delar kan leda till olycka[257].

255. Jfr Leveson: "*Safety is freedom from accidents or losses*" -ett eftersträvansvärt idealtillstånd förodrat framför t ex "*freedom from unacceptable levels of risk*", bl a för att undvika tolkningstvister av vad som är tolerabelt/acceptabelt (och i så fall för vilka) och för att inte ge utrymme för att riskeliminerande/-reducerande designalternativ ignoreras.
256. Riskkälla + riskkälleexponering + andra omständigheter (t ex person i riskområde, se exempel under 6.1.4.).
257. I denna handbok ett **sammanfattande begrepp** för säkerhetskritiska **delar av alla kritikalitetsklasser**, vilka även kan innehålla säkerhetsrelaterade **styr- eller skyddsfunktioner** med uppgift att hindra eller lindra skada om hot mot systemsäkerheten upptäcks. **DS 00-55** använder termen säkerhetskritisk enbart för system eller funktioner av **högsta kritikalitet** (S4).

Säkerhetskritiskt testscenario	Testscenario, vilket med systemet i verklig anläggning skulle kunna leda till att en olycka inträffar (jfr ”Säkerhetsinriktat test”).
Säkerhets**relaterad**	System/delsystem/funktion med uppgift att uppnå eller bibehålla säkert tillstånd, för den del, som den är avsedd att styra eller skydda[258].
Säkerhetsövervakning	Funktion med syfte att upptäcka potentiellt osäkra tillstånd och som vid upptäckt vidtager åtgärder, för att undvika dessa.
Säkert tillstånd	Ett väldefinierat tillstånd, som är säkert[259] m a p systemsäkerheten.
Teknisk specifikation	Krav på den produkt FMV har i uppgift att anskaffa åt FM.

Test, -ning, -teknik

- Beteende- – Test m a p beteende. Jfr ”Kravtestning”.
- *Built in Test (BIT)* – Test, realiserad mestadels i programvara som del av levererad produkt. I BIT ingår t ex
  - a: säkerhetskontroll, SK (under uppstart),
  - b: funktionskontroll, FK (under klargöring, service, uppstart),
  - c: prestandakontroll/fellokalisering, PK/FL (vid service, felsökning),
  - d: funktionsövervakning, FÖ (under drift).

258. Exempel: Säkerhetssystem eller -funktioner för **styrning** resp **skydd** (se 6.7.).
- Säkerhetsrelaterat system kan antingen vara fristående från det ordinarie, övervakade styrsystem eller utgöra en del av detta.
- Styrsystemet kan skydda mot felyttringar eller avvikelser i övervakat system, t ex genom avvärjande styråtgärder eller genom att inkludera skyddsanordningar som alarm och nedstängningsfunktioner.
- En operatör kan vara del av ett säkerhetsrelaterat system, t ex genom att vidtaga lämpliga säkerhetsåtgärder.
- Det finns skyddssystem, som inte är säkerhetsrelaterade, t ex de, som hindrar eller lindrar skada (skyddsbarriär, nödtelefon).

DS 00-55 inbegriper i termen säkerhetsrelaterad alla system eller funktioner av **någon kritikalitet** (dvs S1-S4). **IEC 61508** avser med säkerhetsrelaterad programvara de säkerhetsfunktioner (dvs **skyddsfunktioner**), som implementerats för att nå eller upprätthålla ett säkert tillstånd i den styrda utrustningen. En implementering av dessa säkerhetsfunktioner med syfte att upprätthålla den tolererade risknivån kallas för det säkerhetsrelaterade systemet.

259. Vid övergång från osäkert till säkert tillstånd kan systemet passera ett antal mellanliggande tillstånd, där säkerhetshoten gradvis reduceras. Det finns system, där upprätthållandet av ett säkert tillstånd fordrar kontinuerlig, programvarukontrollerad styrning (t ex styrsystem till flygplan med aerodynamiskt instabila flygegenskaper). Jfr ”*Fail safe*”.

- Domän- – Test baserad på I/O-variablernas domän (dvs skiljelinjen tillåtna – otillåtna värden). Exempel på domäntest är ekvivalenstest och randvärdestest.

  Domäntest är vanligtvis kravorienterade (baserade på specifikation), men kan vara strukturella (baserade på implementation).

- Ekvivalens- – Testteknik baserad på en ekvivalensindelning av in- och utvärdesmängderna, där ett representativt testfall väljs från varje partition av tillåtna resp otillåtna värden.

- Enhets- – Test av programvarans minsta, separat testbara delar (t ex subprogram, subrutiner, procedurer) m a p krav och konstruktionsrestriktioner.

- Formella test – Test enligt specificerade testplaner och -beskrivningar granskade och godkända av kund, användare eller annan administrativ instans (kvalitetsavdelning). Testen utförs på fryst version. Testad utgåva frisläpps utanför utvecklingsgruppen då testresultaten granskats och godkänts.

- Funktionella- – Test m a p funktion. Jfr ”Kravtestning”.

- Funktionella system- – Funktionella test på ett integrationstestat system. Ingår i systemtest.

- Graf- (*path testing*) – En klass av testmetoder baserade på grafer från olika konstruktionsvyer för enhet, komponent eller system över programflöde, kommunikationsflöde (dataflöde, transaktionsflöde), finita tillstånd etc. **Noder**na i dessa representerar flödets förgrenings- och föreningspunkter. Förbindningslinjerna mellan noderna (dvs vägavsnitten) definierar flödets vägar (*paths*) genom grafen.

Graf över	Noder i grafen
styrflöde	start & slut för *if-*, *while-*, *for-*, *case-*, *loop*satser
transaktions-flöde	start & slut för parallella flöden i en kedja av händelser/systemfunktioner/ användningsfall
tillstånds-diagram	start & slut för tillståndsövergångar, dvs tillstånd före och efter övergång

- Integrations- – Test av integrerade komponenter m a p data, gränssnitt och interaktion mellan komponenter utförda efter avslutad komponenttestning.

• IT-säk- (*security* -) – Test av att systemet kan skydda sig mot IT-säkerhetshot.

• Komponent- – Test av en programvarukomponent (bestående av en eller flera integrerade programvaruenheter) m a p krav och konstruktionsrestriktioner utförda efter avslutad enhetstestning.

• Konfigurations- – Test av ett konfigurerbart system m a p att varje möjlig kombination av maskin- och programvara fungerar enligt specificerade system. Där antalet möjliga systemkonfigurationer är så stort att samtliga inte kan testas med rimlig insats (vanligt för operativ- och databashanteringssystem), utförs åtminstone test av systemet i dess minimala och maximala konfiguration med varje möjlig typ av maskinvaruenhet.

• Krav- – Test av realiserad programvara för att verifiera specificerade krav. Exempel: Beteendetest, funktionella test, *black-box* test. Tillämpas i huvudsak vid integrations- och systemtest.

• MC/DC- [260] – Strukturell testteknik, där utförda test minst en gång skall ha exekverat

a: programmets samtliga ingångs- och utgångspunkter,

b: varje villkor i en beslutspunkt, så att varje möjligt utfall antagits,

c: varje beslut lett till samtliga möjliga utfall,

d: varje villkor i en beslutspunkt skall oberoende ha påverkat utfallet.

Testen resulterar i att objektkodens samtliga grenar genomlöps.

260. *Modified Condition/Decision Coverage*
(d) kan åstadkommas, t ex genom att variera ett villkor, medan de övriga hålls konstanta. Visas lätt för t ex ”*short-circuit-forms*”.
(b-c) förenklat uttryckt: Varje villkor i ett boolskt uttryck skall utvärderas till både *true* och *false*.
Varje villkor skall ha visat sig oberoende ha påverkat uttryckets utfall.
Förenklingen leder till samma mängd testfall (för komplexa operationer med *n* boolska variabler vanligtvis endast *n*+1 test).
MC/DC-test har jämfört med övriga strukturella testmetoder (se 4.3.2.2.5.) de hårdaste täckningskraven.
Testfall, som genomlöper samtliga satser (dvs med 100% *statement coverage*) kan visa sig motsvara 60% täckning enl MC/DC. Testen är mycket kostsamma -i synnerhet där rättning och omtest visar sig nödvändiga. För utvärdering av MC/DC, se <Leveson_2>.

- Prestanda- – Test av att specificerade tidskrav på genomströmning (*throughput*) och maximal fördröjning (*delay*) är uppfyllda.
- Randvärdes- – Test baserad på in- och utvärdesmängdernas randvärden. Ur varje **tillåten partition** (se ekvivalenstest) väljs ett testfall på och omedelbart intill randen (men inom domänen). Ur varje **otillåten partition** väljs ett värde omedelbart utanför.
- Regressions- – Omtest med samma inmatning och startvillkor som vid tidigare test (ofta använd för att visa, att ändringar inte infört oavsedda effekter).
- Stress- – Test med syfte att utvärdera ett system/komponent på eller utanför dess specificerade gränser, t ex test under orimlig last och med otillräckliga resurser för att hantera överlasten.
- Strukturell- (*white-box test, glass-box test*) – Testteknik baserad på programvarans interna struktur. Hit räknas olika typer av graftest samt vissa domäntest. Testtekniken är inriktad mot att finna icke-exekverade vägar i koden.
- Syntax- – Test av hur systemet klarar att hantera såväl korrekt som inkorrekt inmatning från externa och interna delar, t ex kommandon från operatör/användare, kommunikationsprotokoll, *scripts*, dolda språk.
- System- – Test av ett integrerat system m a p dess övergripande beteende samt egenskaper, som inte kunnat verifieras på underliggande testnivåer (dvs under tidigare enhets-, komponent- och integrationstestning)[261].
- Säkerhetsinriktad- (*safety test*) – Test (under betryggande former) av att systemet kan skydda mot säkerhetskritiska incidenter (jfr ”Säkerhetskritiskt testscenario”).
- Volym- – Test vid gränsen för maxlast.
- Återhämtnings- (*recovery test*) – Test av systems förmåga att kunna behålla kontroll och integritet.

Testdriver – En separat procedur, som startar en modul för test, förser den med indata, styr och övervakar exekveringen samt rapporterar testresultat[262].

261. Exempel: Funktionalitets-, prestanda-, stress-, volym-, konfigurations-, uppstarts-, återhämtnings- och säkerhetsinriktade test.
262. En *testdriver* lägger alltså inte in testinstruktioner i testad enhet. Vid rapportering av testresultat kan en *testdriver* använda sig av regressionsteknik (dvs ange om resultatet från ett upprepat test skiljer sig från tidigare testresultat).

Testfall	En tupel (*testinput*, testutfall) definierad för en enhet under test.
Testnivå	Indelning av testarbetet i konsekutiva test på successivt större delar av systemet: enhetstest (komponenttest), integrationstest och systemtest.
Testprocedur	Se *testdriver*.
Testscenario	Beskrivning av ett antal förutsättningar för test av ett system (dvs omständigheter i omgivning och ingångsvärden till systemet) samt förväntat utfall (händelseförlopp, resultat, effekt på omgivningen).
Testsession	En trippel (*testdriver*, t*estinput*, testutfall) definierad för en enhet under test.
Testtäckning	Testinsatsens omfattning m a p den totala kravmassan eller viss programvarustruktur uttryckt i täckningsgrad för en given måttenhet[263].
Tillbud	En händelse, som under andra omständigheter skulle kunna leda till olycka.
Tillförlitlig (*reliable software*)	Med förmåga att utföra begärd funktion eller tjänst tillfredsställande under föreskriven tid[264] och specificerade omgivnings- och driftsförhållanden. För programvara ett kvalitativt uttryck för frihet från syste-

263. Exempel på olika måttenheter: Exekverade sekvensiella block, satser, beslutsvägar, möjliga villkorsutfall, eller kombinationer av dessa (t ex MC/DC). En stark måttenhet (t ex MC/DC) innebär lägre risk för oupptäckta fel och kräver fler testfall för 100%-ig täckning än en svag måttenhet (t ex statement coverage). Att tala om 100%-ig testtäckning, utan att nämna måttenhet, är meningslöst.
Exempel på mått efter kritikalitet: DO-178B ställer bl a följande täckningskrav för programvara:

Nivå A	MCDC
Nivå B	*Decision Coverage*
Nivå C	*Statement Coverage*
Nivå D	-

Rekommenderad teststrategi är, att fastlägga måttenhet och täckningsgrad redan under tidig testplanering utgående från betydelse och kritikalitet hos komponenten för test, att sträva mot en testbar konstruktion, att välja effektiva testfall (baserade på krav och struktur), att kolla täckningsgrad efter test -och slutligen- att komplettera/reducera testfall för optimerad testtäckning. Testprocedurer, testdata och resultat sparas inför regressionstest.

264. I stället för specificerad tid kan felfriheten avse per någon annan enhet förknippad med resultatet (antal sidor, meddelanden etc.).

	matiska fel under givna villkor[265]. Jfr ”Funktionssäker”, ett begrepp för maskinvara ofta kvantifierat i termer av medeltid mellan fel (MTBF). (Tillförlitlighet beaktar felyttringar i gemen, medan systemsäkerhet endast berör de som kan leda till olyckor. I båda fallen kan faktorer utanför komponent och system inverka). Se 6.1.3. och 6.1.5.
Tillgänglig (*available*)	Erbjuda kontinuerlig service.
Täckningsgrad	**Vid strukturell testning**: mått på hur stora delar av koden, som den totala uppsättningen testfall genomlöpt[266]. Olika måttenheter finns. Vald täckningsgrad utgör ett stoppkriterium för den strukturella testningen. **Vid kravtestning**: mått på i vilken utsträckning specificerade krav verifierats[267].
Underhållsmässig (*maintainable*)	Möjlig att vidmakthålla eller återställa i tillfredsställande skick (för maskinvara ofta kvantifierad i medelreparationstid, reparationsintensitet).
Underhållssäker (*performable mainten. support*)	Förmåga hos en underhållsorganisation att kunna tillhandahålla erforderligt underhåll (för maskinvara ofta kvantifierad i logistisk medelväntetid).
Uppdragskritisk (*mission critical*)	System/delsystem, där felyttringar kan äventyra möjligheterna att fullfölja systemets uppdrag. Se även 6.1.5.
Uppstart[268]	Systemmod då systemets delar successivt kopplas in. Initialt skede från starttidpunkt till dess systemet inträtt i full, operativ mod.
Utvecklare	Konstruktör, programmerare, som implementerar och testar programvarusystemets delar.
Validering	Den process, som säkerställer, att implementerade krav är korrekta och fullständiga. (Svarar på frågan: Har vi byggt rätt produkt?)

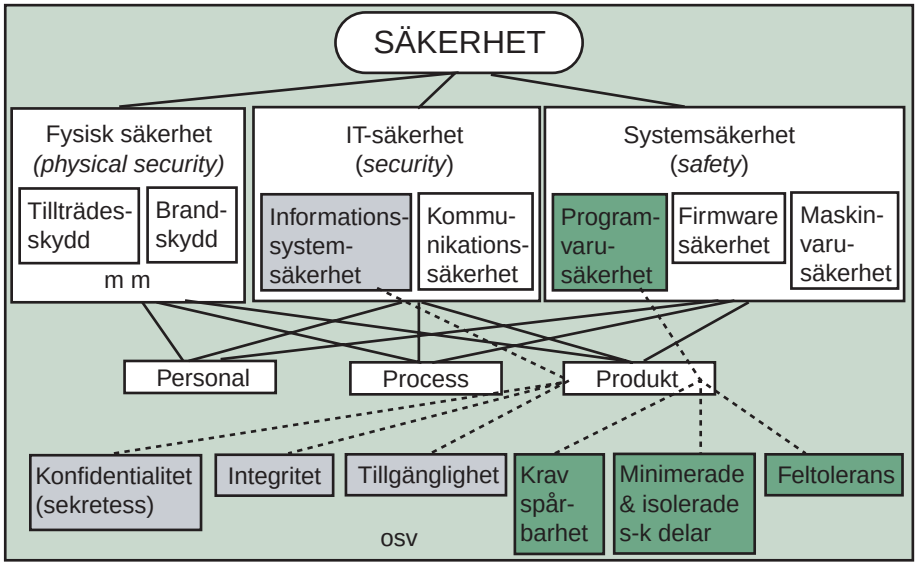
265. Hög tillförlitlighet för en programvaruprodukt i visst sammanhang kan bero på att kvarvarande ”fel” ej aktiveras (vilket dock kan inträffa under andra förhållanden). ”Fel” kan f ö vara en missvisande term för programvara: vad som är tillräckligt tillförlitligt i **ett** sammanhang, kan vara undermåligt i ett annat, utan att det beror på ett direkt ”fel”,< FMV_2>,<Patriot>. Kopplingen tillförlitlig-felfri kan även ifrågasättas, <Fenton_2>: en liten andel av de kvarvarande felen kan stå för nästan alla observerade felyttringar (**Pareto**-principen).
266. Måttet uttrycks i kvoten = antalet exekverade enheter: totala antalet exekverbara enheter.
267. Måttet en kvot= antalet satisfierade krav: totala antalet krav. Krävd täckningsgrad för kravtest är 100%.
268. Ordet start används ej, då det ger ett mer momentant intryck (det ögonblick då startvredet vrids om).

Vattenfallsmodell	En beskrivning av programvarans livscykelfaser i strikt sekvens (Kravdefinition, System och programvarukonstruktion, Implementation och enhetstest, Integration och systemtest, Drift och underhåll), <Royce>.
Watchdog timer	En oberoende, extern klocka, som bevakar att ett system (vid fel i program eller utrustning) inte loopar i oändlighet eller träder in i ett inaktivt tillstånd.
Verifiering	Den process, som säkerställer, att implementationen uppfyller ställda krav (dvs utvärdering av resultat från en fas/steg/process, för att försäkra sig om att dessa är korrekta och konsistenta med fasens ingångskrav). (Svarar på frågan: Har vi byggt produkten rätt?)
Verksamhetsåtagande	Krav på leverantörs personal, process, och produktionsmiljö.
Väg (*path*)	Se Graftest ovan.
Z	Ett modellbaserat specifikationsspråk för främst data-orienterade, sekvensiella system. Genom att specificera pre- och post-villkor för systemets operationer går att visa, om dessa bevarar definierade invarianter.
Ändringsanalys	Analys, som undersöker, vilka delar som påverkas av en ändring.

6.1.2 Säkerhet

Det gemensamma säkerhetsbegreppet som finns i många språk är ett mångfacetterat uttryck för frihet från förluster vid olika typer av hot. Det täcker in flera områden, vart och ett med sina karakteristika – några unika, några gemensamma. Nedan ges exempel på tre områden. Fysisk säkerhet och IT-säkerhet inbegriper i olika grad skydd mot otillbörlig åtkomst, loggning och incidentrapportering. Båda är en förutsättning för att Systemsäkerhet skall kunna upprätthållas, utan att därmed hänföras till dess verksamhet. Andra egenskaper kan skenbart synas vara gemensamma, men visar sig vid närmre granskning avse olika aspekter. Detta gäller t ex kravet på spårbarhet inom IT-säkerhet och Systemsäkerhet[269].

269. **Spårbarhet** i samband med IT-säkerhet uttrycker möjligheten, att kunna knyta utförda operationer till enskilda individer, men innebär vid System- och programvarusäkerhet, att kunna finna samband mellan t ex ett krav och dess realisering samt motsvarande tester. För samtliga gäller, att krav och realisering skall vara **korrekta, konsistenta** och **fullständiga** (de tre c:na ***c**orrect, **c**omplete, **c**onsistent*).



För språk med gemensamt säkerhetsgegrepp är sammanblandningar inte ovanliga. Det kan det därför vara motiverat att peka på några särskiljande drag mellan IT-säkerhet respektive System- och programvarusäkerhet. (De mest uppenbara skillnaderna återfinns i de krav, som resp egenskap ställer på en produkt, se 4.5.).

- IT-säkerhet avser främst att skydda ett system mot dess omgivning (angrepp från en aktiv angripare). Skyddet är inriktad mot att upprätthålla sekretess, riktighet och tillgänglighet.
- Systemsäkerhet syftar till att skydda omgivningen (person, egendom, miljö) mot oavsiktlig skada från systemet.
- Programvarusäkerhet innebär i sin tur att programvara och programvaruhanteringen ges egenskaper, som bidrar till att systemsäkerheten kan upprätthållas. Tekniken är en integrering av:
  - ⇒ **Programvaruteknik** – själva ramverket för programutvecklingsarbete baserat på en systematisk uppbyggnad av kvalitet i personalkvalifikationer, process, produkt, produktionsmiljö.
  - ⇒ **Säkerhetsmetodik** – startpunkten för all systemsäkerhetsverksamhet syftande till att bryta händelsekedjan från riskkälla till det osäkra tillstånd, som omedelbart föregår en olycka.
  - ⇒ **Tillförlitlighetsstrategier** – inriktad mot att bryta händelsekedjan från felkälla över till feltillstånd och slutligen till felyttring (på närmaste systemnivå). Tekniken ingår till en del redan i den traditionella programvarutekniken.

Egenskaper, krav och hotbild för de olika säkerhetsbegreppen gör det svårt att finna en gemensam strategi för t ex systemkonstruktion. Även om forskning pågår inom området pålitlighet[270] behandlas dess ingående egenskaper separat. Influenser från ett område kan dock stimulera ett annat.

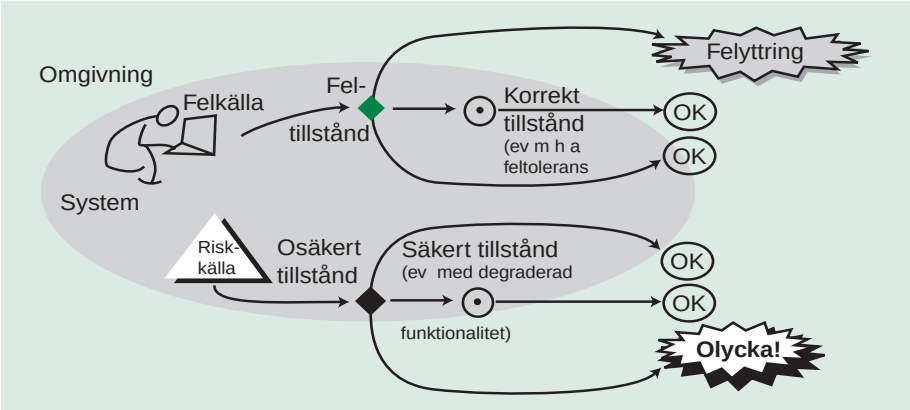
270. Ett paraplybegrepp för IT-säkerhet, systemsäkerhet, tillgänglighet, tillförlitlighet, underhållbarhet. Se även <SQUALE>.

Några gemensamma angreppssätt och mönster (se 6.6.5.) som visat sig användbara är:

- **Partitionering**: Informationsskydd i form av en logisk separering mellan användare av olika behörighet samt brandväggar, säkerhetsportar etc har börjat tillämpas inom programvarusäkerhet för att tillse, att delar av olika kritikalitet, som exekverar på samma maskin, inte interfererar med varandra[271]. Tekniken erbjuder ekonomiska (och viktmässiga) fördelar jämfört med fysisk separering på olika processorer. Den minskar även behovet av total omtest. Största fördelen är troligen att den undviker problem som kan uppstå, då funktioner, som dynamiskt sett är tätt kopplade kommit att separeras fysiskt – vilket inom bl a civil *avionik* i flera fall lett till s k *mode confusion*.
- **Angreppsskydd**: Även här har IT-säkerhet varit inspirationskälla – i detta fall till nya tillförlitlighetsstrategier vid **felinjicering**. Ett exempel är en feltolerant konstruktion kompletterad med komponenter, som påverkar värden eller tidsförlopp i form av s k sabotörer, <MEFISTO>.
- **Identifiering av riskkällor**: Här används likartade metoder för samtliga säkerhetsbegrepp (4.3.3.).

6.1.3 Systemsäkerhet – Tillförlitlighet

Systemsäkerhet och tillförlitlighet är besläktade. Båda teknikerna söker genom analys och verifiering finna var konstruktionsförbättringar är nödvändiga, för att begärd säkerhet/tillförlitlighet skall kunna upprätthållas. Arbetet inriktas i första hand mot att förebygga och eliminera riskkällor/felkällor, i andra hand att bryta händelsekedjan från riskkälla/felkälla över till osäkert/felaktigt tillstånd, för att förhindra olyckor/felyttringar eller se till att kvarvarande risker kan hållas på en för systemet tolerabel nivå i dess avsedda omgivning och användning.



271. Exempel: Robust partitionering, <Vito>. Utvecklingen inom flygindustrin av IMA, *Integrated Modular Avionics,* <ARINC 651>, <Rushby>. Säkerhetsportar, <Watt>.

Skillnaderna består bl a i att:

- Säkerhetsverksamheten inleds på övergripande systemnivå (dvs för det översta systemet i ett system av system) och fortsätter nedåt i varje separat system för analys av säkerhetshot i delsystem, komponenter samt i gränssnitt mellan dessa och mot omgivningen.
- Tillförlitlighetsarbetet utförs i stället nedifrån och upp till närmaste systemnivå, t ex genom att resterande felyttringar komponentvis skattas ur statistik över alla typer av felyttringar (inklusive de icke säkerhetshotande).

Även om tillförlitlighetstekniken begränsas till att successivt eliminera enbart säkerhetshotande felyttringar i allt större systemdelar, så har den svårt att finna samverkande effekter mellan i och för sig korrekt fungerande komponenter och omgivningen[272]. Prioriteringarna i säkerhetsarbetet måste därför alltid inriktas på den totala säkerheten. I detta arbete kan tillförlitlighet vara av primärt intresse, t ex då det gäller systemets skyddsanordningar.

- Korrekt funktionalitet är en faktor i den totala säkerheten för skydds- och övervakningsfunktioner, medan
- Säker funktionalitet är den dominerande egenskapen för delar, som i sig kan utgöra ett hot mot systemsäkerheten (se 6.7.).

6.1.4 Risk

Risk används i vardagligt språk ofta som uttryck för det troliga, att någon typ av skada kan bli följden. I systemsäkerhetssammanhang omfattar begreppet både rimlighet eller sannolikhet ***p***, att en olycka inträffar[273] **samt** dess värsta möjliga konsekvens (allvarlighetsgrad), ***k***, och betecknas därför ibland med tupeln (p, k). Sannolikheten p inkluderar dels sannolikheterna för att en riskkälla föreligger, $\boldsymbol{q_0}$, dels att denna leder till en olycka, $\boldsymbol{\Pi p_i q_i}$, vilket ger ett mer detaljerat uttryck för risk, ($q_0 * \Pi p_i q_i$, k), se bild på sid 123.

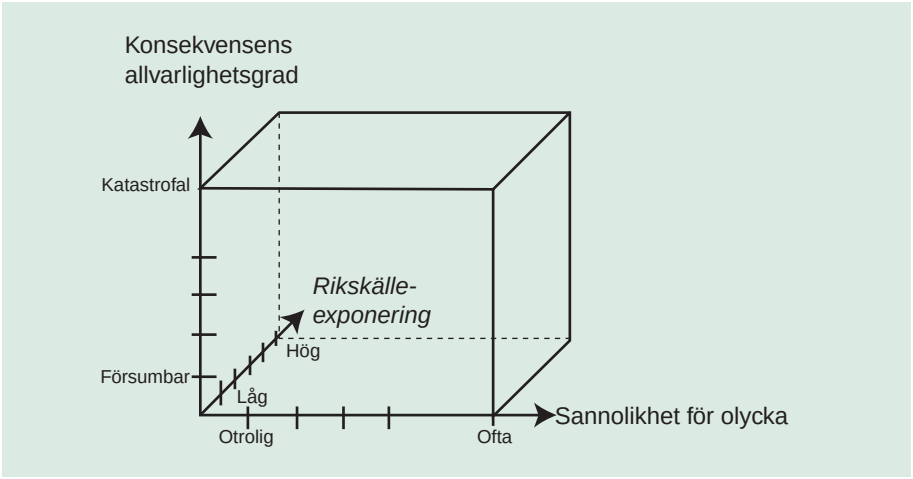
Ytterligare faktorer kan påverka olycksrisken, bl a komplexitetsgrad hos systemet och riskkälleexponering (m a p volym, tid, närhet, antal gånger). Flera av de tiotal riskdefinitioner som kan påträffas i litteraturen tar också med dessa faktorer (riskkälleexponering finns som en tredje komponent i riskbegreppet i <Leveson> och <Brauer>).

För det mesta kan det räcka med den enklaste riskbeskrivningen, (p, k). Det torde t ex vid riskbedömning inte vara meningsfullt, att försöka bryta ned sannolikheten för viss olycka i mer eller mindre svårkvantifierade delsannolikheter (f ö är siffermässiga skattningar vanskliga för programvara). En mer detaljerad formulering kan däremot vara av värde, då det gäller att avgöra vilka faktorer, som bör åtgärdas, för att få ned riskerna på en tolerabel nivå. I vissa

272. Se exempel i <Leveson>.
273. Väsentligt är att klargöra vad sannnolikheten avser (per viss tidsenhet, händelse, population, enhet, aktivitet).

sammanhang kan t ex införandet av skyddssystem som minimerar riskkälleexponeringen vara en lösning (se exempel nedan).

Ett närbesläktat begrepp till risk är **riskkällenivå** (sannolikheten att en riskkälla föreligger samt dess allvarlighetsgrad), vilken kan betecknas (q_0,k). Ofta används en **matris** för att beskriva vilka kombinationer av de två komponenterna, som är tolerabla resp icke-tolerabla för aktuellt system. Detta ligger till grund för **var** i konstruktionen ansträngningarna vid riskeliminering/-reducering skall prioriteras, för att riskerna skall kunna hållas inom fastlagd toleransnivå. Där riskkälleexponeringen utgör en starkt bidragande faktor och det traditionella, 2-dimensionella riskbegreppet används, är det brukligt att gå upp en nivå i riskmatrisen vid klassning av inträffandefrekvens. Med ett 3-dimensionellt synsätt, där riskkälleexponering ingår som en komponent, kommer denna automatiskt med i riskbedömningen. Riskmatrisen övergår från ett plan till en riskkub, där riskbedömingen består i att fastlägga vilka av de ingående delkuberna, som ej skall betecknas som tolerabla.

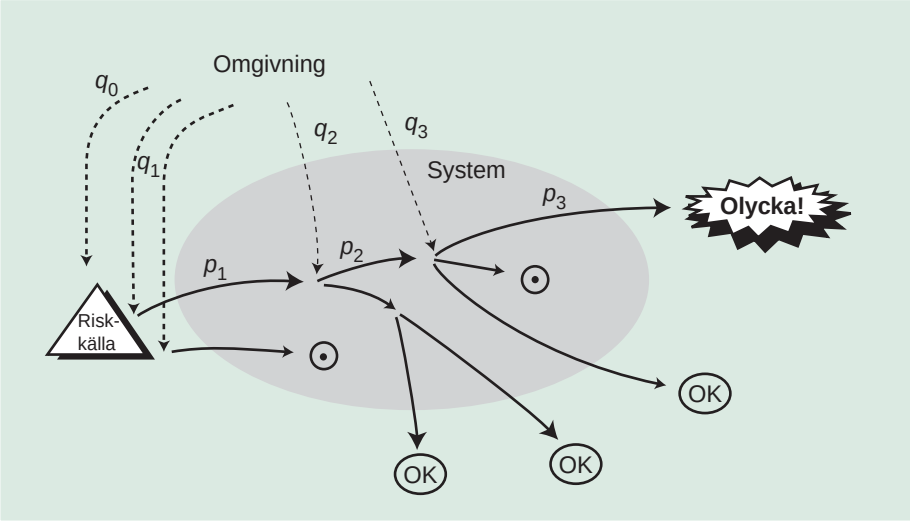


Exempel på några riskbegrepp:

Riskkälla	– Tåg på bana.
Övriga omständigheter	– Aktiverade ljud-/ljussignaler.
	– Person / fordon i spårområde (fara).
Riskkälleexponering	– Tid i spårområdets träffzon.
	– Närhet till spår (avstånd från träffzon).
	– Antal personer i spårområdet.

Konsekvens (k: värsta utfall)	– Dödsfall.
Sh för olycka ($q_0 * \Pi p_i q_i$)	– Sh för riskkälla (q_0) & Sh att riskkälla leder till olycka ($\Pi p_i q_i$).
Olycka	– Person i korsning men inom träffzon för passerande tåg.
Incident	– Person i korsning men utanför träffzon då tåget passerar.
Riskreducering	– Järnvägsbommar (om ljud- och ljussignaler redan finns).
Riskeliminering	– Järnvägsviadukt kombinerad med inhägnat spårområde.
	– Järnvägsövergång avstängd/bortbyggd.

För att undvika missförstånd beträffande olika sannolikhetsbegrepp är det väsentligt att klargöra vilket avsnitt i händelsekedjan, som avses.



$\boldsymbol{q_i}$: Den samlade sh från de bakgrundsfaktorer i omgivningen, vilka bidrar till att efterföljande systemstatus intas (sociala, tekniska och organisatoriska aspekter).

$\boldsymbol{p_i}$: Sh – givet q_i – för övergång till nästa systemstatus i den händelsekedja som leder till olycka.

$\boldsymbol{\Pi p_i q_i}$: Den sammanvägda sh – givet q_0 – baserat på bakgrundsfaktorer fram till olycka. (**Π** betecknar en **produkt** av sh:er **endast om oberoende** föreligger mellan bakgrundsfaktorerna).

Olika säkerhetsanalyser utförs för att kartlägga ett systems risker och riskkällor. Efter identifiering av möjliga olyckor utförs s k **riskkälleanalys** för att utreda säkerhetshot (omständigheter, förutsättningar, tillstånd som tillsammans med riskkälla och andra förhållanden i omgivningen kan leda till en olycka) samt motsvarande riskkällenivå.

Först när systemets inre struktur börjat ta form och händelsekedjorna in i systemet från riskkälla till olycka kan studeras är det möjligt att komplettera en riskkälleanalys till en s k **riskanalys**, vars sannolikhet omfattar hela kedjan fram till olyckan och vars huvudsyfte är att i detalj utreda konsekvenserna.

6.1.5 Kritikalitet

Kritikalitet är ett diskret mått på hur stor effekt en del (eller ett fel) i system och avsedd omgivning kan ha på systemsäkerheten. Olika uttryck för kritikalitet förekommer (se 6.9.1.2.). De flesta är mer eller mindre direkt kopplade till riskbegreppet. Stark associering föreligger t ex i H ProgSäk, H SystSäk och 00-55, svagare i DO-178B och IEC 61508.

Kritikalitetsklassning startar, liksom allt säkerhetsarbete, på systemnivå. Analys av systemet och dess delar leder till insikt om vilka, som är mest säkerhetskritiska. Ur högsta tolererade kritikalitet för systemet fastläggs hur stort bidrag varje enskild del högst får ge. Fördelning av krav (restriktioner) angående högsta tolererade kritikalitet görs m a o *top-down*, medan bedömning av om det realiserade system uppfyller kritikalitetsrestriktionerna utförs *bottom-up*.

För programvara får man nöja sig med en rent kvalitativ kritikalitetsklassning, tillräcklig för att avgöra var ändringar är nödvändiga, såvida inte tillfredsställande statistisk (se 4.3.2.2.6.) för skattningar av felfrekvens föreligger (se 6.8.).

Klassning av programvarans kritikalitet fyller flera behov:

1. Att finna vilka krav som måste ställas på programvaran, för att systemets säkerhetskrav skall kunna uppfyllas.
2. Att finna var det är mest optimalt, att ändra programvarans konstruktion, för att få ned dess kritikalitet. (Olika sätt att åstadkomma detta framgår ur 4.3.3. och 4.5.2.2.).
3. Att utvärdera alternativa konstruktionslösningar m a p kritikalitet.
4. Att göra troligt, att tolerabel kritikalitetsnivå uppnåtts (s k *safety case* argumentering).

Aktiviteterna under punkt 1–3 syftar till att åstadkomma en säker programvaruprodukt. Under punkt 4 gäller det att visa, att realiserad produkt är tillräckligt säker, vilket kan kräva en omfattande insats (i synnerhet om många nya delar ingår och konstruktionen inte systematiskt inriktats mot system-

säkerhet redan från början). Vägledning i hur denna bedömning skall utföras är mycket knapphändig i de säkerhetsstandarder, som f n föreligger, <Fenton_1>.

Hur skall programvarans kritikalitet bestämmas?

Endast de delar av programvaran, som kan påverka en kritisk systemdel behöver kritikalitetsbestämmas. Vilken effekt på systemsäkerheten en kritisk programvarudel kan ha är i högsta grad avhängigt under vilka villkor och i vilken omgivning programvaran är avsedd att exekveras. Kritikaliteten kan m a o inte klassas för en programvara ryckt ur sitt sammanhang. Klassningarna avser risk för olycka (dvs konsekvensens allvarlighetsgrad och sannolikhet/rimlighet för olycka) och uttrycks för programvara enbart i **kvalitativa** termer.

Några allmänna anvisningar:

- Initialt tilldelas
  - Övervakande programvara: Samma kritikalitet, som den del den övervakar.
  - Programvarukomponent: Samma kritikalitet, som den del, där den ingår.
  - Programvara, data, kommandosekvens och operatörsprocedurer: Samma kritikalitet, som den del informationen levereras till.
  - Delar av lägre kritikalitet, som beror av eller påverkar delar med högre kritikalitet: Den högre kritikaliteten.
- Riskreducerande konstruktionstekniker tillämpas för att få ned kritikaliteten (i första hand där största kritikalitetsbidrag föreligger).

Leveson (en ledande auktoritet på säkerhetsfrågor) har ifrågasatt värdet av ett kritikalitetsbegrepp. Genom att specificera kraven formellt och samtidigt förståeligt för den applikationskunnige, kan dessa granskas m a p korrekthet. Verktyg tas fram för kontroll av kravens fullständighet och motsägelsefrihet, för generering av kod och för säkerhetsanalys. I o m att generatorerna är verifierade m a p korrekthet kan granskning- och testmoment under hela livscykeln utgå. I stället styrs realiseringen genom en successiv detaljering av kraven, där resultat från olika säkerhetsanalyser ger kompletterande krav och konstruktionsrestriktioner. Underhåll och ändringar utförs på kravnivå, följd av ny kravverifiering och kodgenerering. Detta är några av målsättningarna vid bygget av en uppsättning verktyg för komplexa styrsystem, <SpecTRM>.

Även om frågan kvarstår huruvida det är möjligt, att specificera användarinteraktion, avbrottshantering, synkroniseringar, tidskrav, prestanda hos system, kompilatorer och plattformar så detaljerat, att genererad kod får de önskvärda egenskaperna, skulle denna typ av hjälpmedel givetvis vara till stor nytta på de delar, vars egenskaper ej är av realtidskaraktär.

6.2 Akronymer

ABS	*Anti Brake System*
ACE	*Adaptive Communication Environment*
AIS	Anbudsinfordranspecifikation
ALARP	*As Low As Reasonably Practicable*
ANSI	*American National Standards Institute*
ARINC	*Aeronautical Radio Inc.*
ASIC	*Application Specific Integrated Circuit*
ASIS	*Ada Semantic Information System*
ATM	*Air Traffic Management*
BCAG	*Boeing Commercial Aircraft Group*
BIST, BIT, BITE	*Built-in Self Test, Built-In Test, Built-In Test Equipment*
BSI	*British Standards Institution*
CACM	*Communications of the Association for Computing Machinery* (ACM)
CAS	*Commercially Acquired Software*
CCA	*Cause Consequence Analysis*
CCA	*Common Cause Analysis*
CCB	*Change Control Board*
CCC	*Correctness, Consistency and Completeness*
CD	*Committee Draft*
CDI	*Commercially Developed Items*
CENELEC	*European Committee for Electrotechnical Standardisation*
CHAZOP	*Computer HAZOP*
CI	*Configuration Item* (Konfigurationsenhet)
CM	*Configuration Management*
CMA	*Common Mode Analysis*
CNS	*Communication, Navigation and Surveillance*
CORBA	*Common Object Request Broker Architecture*
COTS	*Commercial Off The Shelf* [274]

274. Några analoga begrepp: OTS, GOTS, MOTS, CAS, CDI, NDI, *3rd party product, public domain.*

DACS	*Digital Aviation Systems Conference*
DCCA-7	*Dependable Computing for Critical Applications* (konf.)
DFM	*Dynamic Flowgraph Methodology*
DS	*Defence Standard* (UK)
ECCS	*European Corporation for Space Standardization*
E/E/PES	*Electrical/Electronic/Programmable Electronic Systems*
EoP	Elektronik och programvara
ESA	*European Space Agency*
ETA	*Event Tree Analysis*
EUROCAE	*European Organisation for Civil Aviation Equipment*
EWICS	*European Workshop on Industrial Computer Systems*
FAA, FAR	*Federal Aviation Authority, Federal Aviation Regulations*
FCD	*Final Committee Draft*
FFA	*Functional Failure Analysis*
FIB	Försvarsmaktens interna bestämmelser
FM	Försvarsmakten
FME(C)A	*Failure Modes, Effects (and Criticality) Analysis*
FMES	*Failure Modes and Effects Summary*
FMV	Försvarets Materielverk
FPGA	*Field Programmable Gate Array*
FRACAS	*Failure Reporting, Analysis and Corrective Action* (Felrapporteringssystem)
FSI	Flygsäkerhetsinspektören (en s k beslutsförig instans)
FSC	Försvarets sjukvårdscentrum
FTA	Felträdsanalys
GIS	*Geographic Information System*
GOTS	*Governmental Off The Shelf*
HAZOP	*Hazards and Operability Analysis*
HKV	Högkvarteret
{HML}	Hög, Medel, Låg (se 1.7.)
HOL	*High Order Language*
HRG	*Annex H Rapporteur Group*
HW	*Hardware* (Maskinvara)
IEC	*International Electrotechnical Commission*
IEE	*Institution of Electrical Engineers*
IEEE	*Institution of Electrical and Electronic Engineers* (USA)

IEV	*International Electrotechnical Vocabulary*
IPC	*Interpartition Communication*
ISSC	*International System Safety Conference*
IV&V	*Independent Verification and Validation*
IMA	*Integrated Modular Avionics*
ISO	*International Organization for Standardization*
IS/IT	Informationssystem/Informationsteknik
JAR	*Joint Aviation Regulations*
JAWS	*Java Adaptive Web Server*
LAN	*Local Area Network*
LOT	*Level of Trust*
MC/DC	*Modified Condition/Decision Coverage*
MISRA	*Motor Industry Software Reliability Association*
MIRA	*Motor Industry Research Association*
MOD	*Ministry of Defence* (UK)
MOTS	*Military off the Shelf*
MTBF, MTTF, MTTR	*Mean Time Between Failure, ... To Failure, ... To Repair*
NASA	*National Aeronautics and Space Administration* (USA)
NATO	*North Atlantic Treaty Organization*
NDI	*Non-Developmental Item*
NORAD	*National Command Center*
OL	Operationsledning
ORB	*Object Request Broker*
O&SHA	*Operating and Support Hazard Analysis* (Säkerhetsanalys för användning och underhåll
OS	Operativsystem
OTS	*Off The Shelf*
PES	*Programmable Electronic System*
PGA	*Programmable Gate Arrays*
ProgSäk	Programvarusäkerhet
PHA	*Preliminary Hazard Analysis* (Preliminär riskkälleanalys)
PHL	*Preliminary Hazard List* (Preliminär riskkällelista)
PHST	*Package Storage and Handling Requirements* (Förslag till hanterings- och förvaringsbestämmelser)

PLC	*Programmable Logic Controller*
PTTEM	Preliminär Taktisk Teknisk Ekonomisk Målsättning
RADS	*Risk Assessment at Disposal of System* (Riskanalys för avveckling av system)
Ravenscar	*Reliable Ada Verifiable Executive Needed for Scheduling Critical Applications in Real-time*
RDB	*Reliability Block Diagram*
RI	*Risk Index*
RFP	*Request for Proposal* (Kravställning vid offertförfrågan)
RHA	*Requirements Hazard Analysis*, se i stället SRCA
RML	Regler för Militär Luftfart
RTCA	*Requirements and Technical Concepts for Aviation*
SAFECOMP	*Computer Safety, Reliability and Security*
SAR	*Safety Assessment Report* (Säkerhetsrapport)
SCA	*Safety Compliance Assessment Report* (Säkerhetsutlåtande)
SESC	*Software Engineering Standards Committee* (IEEE)
SEI	*Software Engineering Institute* (vid Carnegie Mellon)
SEK	Svenska Elektriska Kommissionen
SHA	*System Hazard Analysis* (Säkerhetsanalys för system)
SIL	*Safety (/Software) Integrity Level*
SIS	Standardiseringskommissionen i Sverige
SOW	*Statement of Work* (Verksamhetsåtaganden)
SR	*Safety Release* (Beslut om användning)
SRCA	*Safety Requirements/Criteria Analysis* (Säkerhetskravanalys)
SRP	*Safety Requirements Proposed* (Industrins säkerhetskrav)
SRS	*Safety Restrictions* (Användningsrestriktioner)
SS	*Safety Statement* (Säkerhetsgodkännande)
SSHA	*Sub System Hazard Analysis* (Säkerhetsanalys för delsystem)
SSPP	*System Safety Program Plan* (Systemsäkerhetsplan)
SSPR	*System Safety Progress Review* (Säkerhetsgenomgångar)

SSWG	*System Safety Working Group* (Arbetsgrupp för systemsäkerhet)
STTEM	Slutlig Taktisk Teknisk Ekonomisk Målsättning
SV	*Safety Verification* (Kravverifiering)
SVC	*Safety Verification Condition*
SVVP	*Software Verification and Validation Plan*
SW	*Software* (Programvara)
SWCI	*Software Configuration Item*
TAO	*The ACE ORB*
TEMU	Taktisk Ekonomisk Målsättning för Utbildningsmateriel
TES	*Test and Evaluation Safety* (Provningsvärdighet)
TjF	Tjänsteföreskrift
TOEM	Taktisk Organisatorisk Ekonomisk Målsättning
ToSE	*Transactions on Software Engineering*
TS	Teknisk Specifikation
TSR	*Training Safety Regulations* (Användningsmanualer och utbildning)
TTA, TTP	*Time Triggered Architecture, Time Triggered Protocol*
TTEM	Taktisk Teknisk Ekonomisk Målsättning
TÜV	Technischer Überwachungs Verein (Tyskl.)
USAF	*US Air Force*
UTTEM	Utkast till Taktisk Teknisk Ekonomisk Målsättning
V&V	*Verification and Validation*

6.3 Referenser

Givna referenser är de, som gällde vid handbokens färdigställande. Endast första författarnamn har i regel angivits.

<Ada 95> Ada 95 Reference Manual, International Standard, ISO/IEC 8652:1995-05 (E).

<AIAA> Recommended Practice for Software Reliability, AIAA-R013, 1992.

<ANEP-54> Guidelines for COTS Insertion, NATO Industrial Advisory Group, ANEP-54, okt 1997.

<Aonix> Safety Critical Handbook, Romanski, Aonix, ADOC-SCHB-60321, 2nd ed. 1996.

<Ariane5> Ariane 5, Flight 501 Failure, Report by the Inquiry Board, J.L. Lions, 1996-07-19, http://subs.esa.int:8330/apress/plsql/news.list?p_doctypeid=1&p_year=1996&p_adminid=

<ARINC 613> Guidance for Using the Ada Programming Language in Avionics Systems, ARINC Rep 613.

<ARINC 651> Design Guidance for Integrated Modular Avionics, Aeronatic Radio inc, nov 1991.

<Arnborg> Information Awareness in Command and Control: Precision, Quality, Utility, Arnborg et al, 2000.

<ARP 4754> Certification Considerations for Highly-Integrated or Complex Aircraft Systems, jun 1996.

<ARP 4761> Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment, (http://www.sae.org), dec 1996.

<ASIS> Ada Semantic Interface Specification, ISO/IEC 15291, 1999, www.acm.org/sigada/wg/asiswg/

<Avizienis> The Taxonomy of Design Faults in COTS Microprocessors, Avizienis, DCCA-7, jan 1999.

<Barnes> Programming in Ada 95, J Barnes, Addison-Wesely, ISBN 0-201-87700-7.

<BCAG_1> BCAG Airborne Software Product Requirements, D6-81925.

<BCAG_2> BCAG Digital Avionics Ada Standard, BCAG, D6-53339.

<BCG>	Guidance for the Adoption of Tools for use in Safety Related Software Development, www.iee.org.uk/PAB/SCS/tools.html
<Bobbio>	Comparing Fault Trees and Bayesian Networks for Dependability Analysis, A. Bobbio, SAFECOMP 1999.
<Boehm>	A spiral modell of software development and enhancement, IEEE Computer, 1988.
<Brauer>	3-D System Safety Engineering Process Simultaneously Simplifies and Improves Characterization of a Hazard and Its Risk Identification/ Assessment, M.M. Brauer, Proceedings of the 17th ISSC, 1999.
<Burns_1>	Concurrency with Ada, A Burns, A Wellings, ISBN 0-521-41471-7.
<Burns_2>	Probabilistic Scheduling Guarantees for Fault-Tolerant Real-Time Systems, Univ. of York, YCS 311(1998).
<Carreira>	A Technique for the Experimental Evaluation of Dependability in Modern Computers, Carreira, IEEE ToSE, Vol 24, 1998.
<CCA>	Probabilistic Risk Assessment: Reliability Engineering, Design & Analysis, Henley & Kumamoto, IEEE Press. 1992.
<Chudleigh>	Safety assessment of computer systems using HAZOP and audit techniques, Chudleigh, SAFECOMP 1992.
<CMM>	The Capability Maturity Model, Software Engineering Institute, 1994.
<Crisys>	http://ais.gmd.de/~ap/crisys/
<Daily Build_1>	Microsoft Secrets, Cusumano et al, 1995.
<Daily Build_2>	Rapid Development, S McConnel, 1996.
<Daily Build_3>	Daily Build -the Best of Both Worlds, Rapid Development and Control, Swedish Engineering Industries, 1999.
<DEF(AUST)5679>	The Procurement of Computer-Based Safety Critical Systems, DEF(AUST)5679, www.dsto.defence.gov.au/esrl/itd/safety/, aug 1998.
<DFM>	Programmable Electronic System Design & Verification Utilizing DFM, Houterman, SAFECOMP 2000.
<Digest>	COTS and Safety Critical Systems, IEE Digest 97/013.
<Dijkstra>	Goto statement considered harmful, CACM 11, 1968.
<DIT 01>	Direktiv för Försvarsmaktens IS/IT-verksamhet (DIT 01), 09 626:62156, 19 feb 2001.

<DO-178B> Software Considerations in Airborne Systems and Equipment Certification, dec 1992.

<DO-248> First Annual Report for Clarification of DO-178B, okt 1999.

<DOD 5000.2-R> Mandatory Procedures for Major Defense Acquisition Programs and Major Automated Information Systems, www.deskbook.osd.mil/, okt 2000.

<DOD-STD-Software2167A> Military Standard Defense System Development, feb 1988.

<DS 00-55> Requirements for Safety Related Software in Defence Equipment, Part 1 : Requirements, Part 2: Guidance, MOD Defence Standard, 1 aug 1997.

<DS 00-56> Safety Management Requirements for Defence Systems, Part 1 : Requirements, Part 2: Guidance, MOD Defence Standard, 13 dec 1996.

<DS 00-58> HAZOP, 1996.

<ECCS> Guide for Space Applications, ECCS-E-40 (under utarbetande).

<EIA-6B> System Safety Engineering in Software Development, Electronic Industries Association, 1990.

<EN50128> Software for Railway Control and Protection Systems, CENELEC, EN50128:1997.

<Erikson> Pålitliga system, Christer Erikson, Henrik Thane, 960414[275].

<Fabre> Assessment of COTS Microkernels by Fault Injection, DCCA-7, jan 1999.

<Fagan_1> Design and code inspections to reduce errors in program development, M.E.Fagan, IBM Systems Journal Vol.15, No.3, 182-211, 1976.

<Fagan_2> Inspecting Software Design and Code, Datamation, 133-144, okt 1977.

<Fagan_3> Advances in Software Inspections, IEEE Transactions on S/w Engin., SE-12(7), 744-751, juli 1986.

<Fenton_1> A Strategy for Improving Safety Related Software Engineering Standards, Fenton, IEEE ToSE, Vol. 24, No 11, nov 1998.

<Fenton_2> A Critique of Software Defect Prediction Models, IEEE ToSE, Vol 25, No 3, jun 1999.

275. OH-underlag över begrepp och metoder vid konstruktion av säkra och tillförlitliga (*reliable*) system.

<FME-Guide> Guidance on the use of Formal Methods in the Development and Assurance of High Integrity Industrial Computer Systems, Part I-III, WP4001-4002, EWICS TC7, www.ewics.org, juni 1998.

<FM HIT 97:3> Försvarsmaktens handbok för informationsteknik, M7757-784601, 1998.

<FM-State> The State-of-the-Art in Formal Methods, Barjaktarovic, www.wetstonetech.com, jan 1998.

<FMV_1> Development and Certification fo Safety Critical Software, FMV, ELEKTRO:19448:14128/96.

<FMV_2> Programvarusäkerhet (en introduktion, Inspektion 14910:9284/00, nov 2000.

<FOA> Värdering av säkerhetskritisk elektronik och programvara, FOA C 30636-3.8, 1991.

<formalWARE> (www.cs.ubc.ca/formalWARE/).

<Gilb> Software Inspection, T. Gilb, D. Graham, Addison Wesley, ISBN-0-201-63181-4.

<Hairston> Multi-Phase Fault Tree Analysis, J. Hairston, Proceedings of the 16th ISSC, 1998.

<Hatton> Safer C, Developing SW for High Integrity and Safety-Critical Systems, Les Hatton, McGraw-Hill Book Company, ISBN 0-07-707640-0.

<Hermann> Software Safety and Reliability, Debra S. Hermann, IEEE Computer Society, ISBN 0-7695-0299-7, 1999.

<HiP-HOP> Hierachical Performed Hazard Origin and Propagation Studies, Papadopoulos, SAFECOMP 1999.

<H Kravdok> Handbok KRAVDOK FLYG 075/96, utgåva 1.0, 1996-04-03.

<H MatProc> Handbok Materielprocessen, FMV, S-115 88 Stockholm, Sweden.

<HMÅL> Handbok för FM:s framtagning av målsättningar för förband, förnödenheter och anläggningar.

<H ProgSäk> Handbok för Programvara i Säkerhetskritiska tillämpningar (denna handbok).

<H SystSäk> Försvarsmaktens handbok för Systemsäkerhet, M7740-784851, 1996.

<H SäkIT> Handbok för Försvarsmaktens Säkerhetstjänst, Informationsteknik, M7745-734061, under tryckning, 2001.

<H Säk IT 96>	Handbok för Försvarsmaktens Säkerhetsskyddstjänst, Informationsteknologi, M7745-774041, 1996.
<IEC 15942>	Guide for the use of the Ada Programming Language in High Integrity Systems, ISO/IEC 15942:2000.
<IEC 60050>	International Electrotechnical Vocabulary. Ch 191: Dependability and quality of service, IEC 60050-191, 1990-12.
<IEC 812>	Analysis techniq:s for system reliability - Procedure for Failure Mode and Effects Analysis, 1985.
<IEC 60880>	Software for computers in the safety systems of nuclear power stations, Publ. 880, sep 1986.
<IEC 1025>	Fault Tree Analysis (FTA), 1990.
<IEC 1131-3>	Standards for Programmable Controllers Language, Part 3, first edition, 1993-03.
<IEC 12207>	Information technology - Software life cycle processes, ISO/IEC 12207, 1995-08-01.
<IEC 15026>	System and Software Integrity Levels, ISO/IEC 15026, ed.1, nov 1998.
<IEC 15288>	Systems Engineering – System Life Cycle Processes, ISE/IEC CD 15288 CD3, 2000.
<IEC 15408-1>	Information technology–Security techniques–Evaluation criteria for IT security, Part1: Introduction and general model, ISO/IEC 15408-1, dec 1999[276].
<IEC 15408-2>	Information technology–Security techniques–Evaluation criteria for IT security, Part2: Security functional requirements, ISO/IEC 15408-2, dec 1999.
<IEC 15408-3>	Information technology–Security techniques–Evaluation criteria for IT security, Part1: Security assurance requirements, ISO/IEC 15408-3, dec 1999.
<IEC 18009>	Information technology - Programming languages - Ada: Conformity assessment of a language processor, ISO/IEC 18009 ed. 1, 1999-12.
<IEC 60812>	Analysis techniques for system reliability - Procedure for failure mode and effects analysis (FMEA), IEC 60812, 1985-07.
<IEC 61025>	Fault tree analysis, IEC 61025, 1990-10.

276. ISO/IEC 15408 ersätter ITSEC för evaluering av IT-säkerhetsprodukter.

<IEC 61508>	Functional Safety of E/E/PE Safety-related systems, Part 1-7, dec 1998 - maj 2000.
<IEC 61511>	Functional Safety of electrical/../.. safety instrumented syst:s for the Process Industry, draft 1998.
<IEC 61713>	Software dependability through the software life-cycle processes - Application guide, IEC 61713, 2000-06.
<IEC 61882>	Hazard and Operability (HAZOP) Studies - Guide word approach, 56/653/CD[277].
<IEC 60300-3-6>	Dependability management -Part 3: Application guide - Sec. 6: Software aspects of dependability, IEC 60300-3-6, nov 1997.
<IEE_1>	Competences for those involved in computer based, safety-related systems[278].
<IEE_2>	Hazards Forum:Safety-related systems:Guidance for engineers, www.iee.org.uk/PAB/SCS/hazpub.htm, 1995.
<IEEE 1012>	IEEE Standard for Software Verification and Validation, IEEE Std 1012-1998.
<IEEE 1228>	IEEE Standard for Software Safety Plan, IEEE Std 1228-1994.
<ISO 9000-3>	Quality management and quality assurance standards – Part 3: Guidelines for the application of ISO 9001: 1994 to the development, supply, installation and maintenance of computer software, ISO9000-3, 1997[279].
<ITK 01>	Försvarsmaktens informationstekniska katalog, planerad utgivning 2001.
<ITSEC>	Information Technology Security Evaluation Criteria, version 1.2, jun 1991.
<ITSEM>	Information Technology Security Evaluation Manual, version 1.0, sep 1993.
<ITS 6>	Terminologi för Informationssäkerhet, ISBN 91-630-2483-7, utgåva 1, mars 1994.
<JA 1003>	Software Reliability Program Standard, Society of Automotive Engineers (SAE), okt 1998.
<JA 1004>	Software Reliability Implementation Guide, SAE, okt 1998, juli 1999.
<JSSSC>	Software System Safety Handbook, Joint Software System Safety Committee, www.nswc.navy.mil/safety/handbook.pdf. dec 1999.

277. Förslag som förväntas antas som IEC-standard med angivet nummer – eventuellt med annan titel.
278. Tas fram inom IEE på uppdrag av BSI. Planerad utgivning 9803.
279. Nya versioner av ISO 9000, ISO 9001 och ISO 9004 har publicerats i december år 2000.

<Kao> FINE: A Fault Injection and Monitoring Environment for Tracing the UNIX System Behaviour under Faults, Kao, IEEE ToSE, Vol. 19, 1993.

<Kopetz> The Transparent Implementation of Fault Tolerance in Time-Triggered Architecture, Kopetz, DCCA-7, jan 1999.

<Kropp> Automated Robustness Testing of OTS Software Components, Kropp, Proc, 28th IEEE Symp. on Fault Tolerant Computing, 1998.

<Laprie> Dependability: Basic concepts and terminology (in English, French, German, Italian and Japanese, Laprie, Springer-Verlag, ISBN 0-387-88296-8, 1992.

<Lawson_1> Guidelines for the assessment of safety critical computer based systems, H W Lawson, dec 1996.

<Lawson_2> Twenty Years of Safe Train Control in Sweden, H W Lawson et al, jun 2000.

<Leveson_1> Safeware: System Safety and Computers, Nancy G. Leveson, ISBN 0-201-11972-2, 1995.

<Leveson_2> An Empirical Evaluation for the MC/DC Coverage Criterion on the HETE-2 Satellite Software, Dupuy, Leveson, http://sunnyday.mit.edu/papers.html, okt 2000.

<Lyu> Handbook of Software Reliability Engineering, M Lyu, McGraw-Hill, ISBN 0-07-039400-8,1996.

<MANA> A Formal Model of the Ada Ravenscar Tasking Profile, Protected Objects, Lundqvist et. al., Ada-Europe´99.

<MCDC> Applicability of modified condition/decision coverage to software testing, Chilensky et al, Software Engineering Journal, sep 1995, Vol 9, No 5.

<McNeil> Safety Critical Software System Verification Strategy, J McNeil et al, Proc. of the 18th ISSC, sep 2000.

<MEFISTO> Fault Injection into VHDL Models: The MEFISTO Tool, Jenn et al, Chalmers Univ, TR 277, 1995.

<Meulen> Definitions for Hardware/Softw. Reliability Engineers, v d Meulen, ISBN 1852331755, 2000.

<Microsoft> Writing Solid Code, Romanski.

<Mills_1> The Management of Software Engineering, Mills et al, IBM System Journal 24, 1980.

<Mills_2> Cleanroom Software Engineering, Mills et al, IEEE Software 4, 1987.

<MIL-STD-498> Software Development and Documentation, dec 1994.

<MIL-STD-882C> System Safety Program Requirements, Jan 1993 (ersatt av 882D).

<MIL-STD-882D> DoD Standard Practice for System Safety, www.afmc.wpafb.af.mil/HQ-AFMC/SE/ssd.htm, feb 2000.

<MIL-STD-1472> DoD Design Criteria Standards Human Engineering, MIL-STD-1472F, (ca 220 sid), aug 1999.

<MIL-STD-1521> Technical Reviews and Audits for Systems, Equipments, and Computer Software.

<MISRA_C> Guidelines For the Use of The C Lang. in Vehicle Based Softw., ISBN 0-9524156-9-0, apr 1998.

<MISRA_G> Development Guidelines for Vehicle Based Software, www.misra.org.uk.

<NASA> Computer-Based Control System Safety Requirements, SSP 50038, Rev.B, ISSA, 1995.

<NASA-A302> NASA Software Formal Inspections Guidebook, NASA-GB-A302.

<NASA-1-97> Formal Methods Specification and Analysis Guidebook..., v 1.0, NASA-GB-001-97, http://eis.jpl.nasa.gov/quality/Formal_Methods/.

<NASA-871> Software Safety, NASA Technical Standard, NASA-STD-871, Feb 12 1996.

<NASA-1740> NASA Guidebook for Safety Critical Software -Analysis and Development, NASA-GB-1740.13-96.

<NASA-2202> NASA Software Formal Inspections Standard, NASA-STD-2202-93.

<NUREG-0492> Fault Tree Handbook, NUREG-0492.

<NUREG-6463> Review Guidelines on Software Languages for use in Nuclear Plant Safety Systems, NUREG/CR-6463, maj 1997, www.nrc.gov.

<ORA AdaAnalys> Analysis of Ada95 for Critical Systems, ORA TR-96-5499-03, draft, Apr 1996.

<ORA AdaGuidance> Guidance on the user of Ada95 in the Development of High Integrity Systems, draft, aug 1996.

<OSE> OSE Realtime Kernel, ENEA, Box 232 Täby, www.enea.se/OSE/products/RTK.

<PALBUS>	Comparative Analysis of Dependability Properties of Communication Protocols in Distributed Control Systems, Sivencrona et al, www.sp.se/pne/software&safety/palbus/eng/report.htm, apr 2000.
<Parnas>	Evaluation Standards for Safety Critical Softw., Parnas, TR 88-220, ISSN-0836-0227, maj 1988.
<Pascoe>	A Survey on Safety-Critical Multicast Networking, J S Pascoe, SAFECOMP 2000.
<Patriot>	Notiser i <RisksDigest> under åren 1991-92.
<PLC-Guide>	Guidelines for the use of Programmable Logic Controllers in Safety-related Systems, WP6009, EWICS TC7, www.ewics.org, okt 1997.
<prEN 954-1>	Safety related parts of control systems, Part 1:General principles for design.
<Pullen>	Analyzing Multi-Phased Dependencies in Fault Trees using Markov Models, Pullen, Proc. of the 17th ISSC, 1999.
<Pyle>	Developing Safety Systems, A guide using Ada, I C Pyle, ISBN 0-13-204298-3, 1991.
<RAVEN>	ObjectAda Realtime/Raven, Aonix AB, Romansv.2, 13140 Nacka, www.aonix.com.
<Ravenscar_1>	Tasking Profiles, Proc. of the 8th Internat. Real-Time Ada Workshop, ACM Ada Letters, 1997[280].
<Ravenscar_2>	The Ravenscar Tasking Profile for High Integrity Real-Time Programs, Burns, SIGAda´98, nov 1998.
<RisksDigest>	The RisksDigest: Forum on Risks to the Public in Computers and Related System, ACM Committee on Computers and Public Policy, P.G. Neumann, http://catless.ncl.ac.uk/Risks/
<RML-G>	Regler för militär luftfart - Grunder, Försvarsmakten, utg.1, M7748-754101, dec 1997.
<RML-V-1>	Regler för militär luftfart - Verksamheter, Del 1 Ledning, Försvarsmakten, utg.1, M7748-754211, jun 1998.
<RML-V-5>	Regler för militär luftfart - Verksamheter, Del 5 Utveckling, certifiering och produktion av luftfartsprodukter, Försvarsmakten, utg. 2, M7748-754251, jun 1999.

280. Se även *Ada-Europe News*, dec-97.

<RML-V-6> Regler för militär luftfart - Verksamheter, Del 6 Flygunderhållstjänst, Försvarsmakten, utg.1, M7748-754261, dec 1998.

<Romanski> Review of Safer C (by Les Hatton), G. Romanski, Jan 1996 © [281].

<Royce> Managing the development of large software systems: concepts and techniques, Proc. IEEE WESTCON, Los Angeles, 1970.

<RT-CORBA> An Overview of the Real-time CORBA Specification, Kuhn, Schmidt, IEEE Computer, jun 2000.

<Rushby> Partitioning in Avionics Architectures: Requirements, Mechanisms and Assurance, Rusby, NASA/CR-1999-209347, jun 1999.

<SA-CMM> Software Acqusition Capability Maturity Model, ESC-TR-96-020, v 1.01, dec 1996.

<SACRES> The SACRES Design Methodology for Safety Critical Systems, Baufreton, 1998, www.tni.fr/sacres.

<Safety shell> New Method of Improving Software Safety in Mission-Critical Real-Time Systems, Proc. of the 17th ISSC, 1999.

<Scheer> Towards Dependable Software Requirement Specifications, Scheer et. al., SAFECOMP 1997.

<SDA> Software Deviation Analysis: A ”Safeware” Technique, J D Reese, N G Leveson, Univ. of Washinton, 1996.

<SEK> Tillförlitlighet - Ordlista, Svenska Elektriska Kommissionen, Svensk Standard SS 441 05 05, Förslag 1999-01-14 [282].

<SEMSPLC> Software Engineering Methods for Safe Programmable Logic Controllers, UK Project.

<Sha> Priority Inheritance Protocols: An Approach to Real-Time Synchronization, Sha, Rajkumar, Lehoczky, IEEE Transaction on Computers, sep 1990.

<SID News> Standards in Defence News, Directorate of Standardization, fax 0141-224 2503, (the.editor@dstan.mod.uk).

<Sinclair> Use of Commercial off-The-Shelf (COTS) Software in Safety-Related Applications, HSE Books, CRR80,1995.

281. Kopia genom e-mail till cawthron@aonix.com, ange format: RTF/Word7/HTML/papper.
282. Den första svenska ordlistan för tillförlitlighetsteknik utgavs av SEK 1972. Den nu gällande är från 1985. Föreliggande förslag är en harmonisering med IEV 50, men innehåller ytterligare termer, vilka därför saknar IEV-nummer. Motsvarigheten till de svenska termerna ges på engelska, franska och tyska.

<Smith> Engineering Quality Software, Smith et. al, Elsevier Applied Science, 1989.

<Sommerville> Software Engineering, Ian Sommerville, Addison Wesley, ISBN 0-201-42765-6, 1995, www.comp.lancs.ac.uk/computing/resources/ser/.

<SP> Säkerhet i datorbaserade maskinstyrningar, H Carlsson, J Jacobson, SP Rapport 1995:58.

<SPARK_1> High Integrity Ada: The SPARK approach, Barnes, Addison-Wesley, 1997.

<SPARK_2> www.praxis.co.uk/technologies.

<SpecTRM_1> SpecTRM: A Toolset to Support the Safeware Methology, N Leveson, Proc. of the 16th Intern. System Safety Conference, sep 1998.

<SpecTRM_2> SpecTRM: A CAD System for Digital Automation, DASC, okt 1998.

<Spring> The Spring Kernel, A new Paradigm for Real-Time Systems, Stankovic, IEEE Software, Vol.8 s 62-72, 1991.

<SQUALE> SQUALE Dependability Assessment Criteria, Draft, ACTS95/AC097, www.newcastle.research.ec.org/squale, jan 1999.

<SSS> System Safety Analysis Handbook, syssafe@ns.gemlink.com, www.system-safety.org/handbook.html, juli 1997.

<STANAG 4404> Safety Design Requirements and Guidelines for Munition Related Safety Critical Computing Systems, 4th draft[283].

<STANAG 4452> Safety Assessment of Munition-Related Computing Systems, 1st draft.

<STARTS> The STARTS Purchasers' Handbook: Software Tools for Application to Large Real Time Systems, National Computers Centre Publ, 1989.

<Storey> Safety-Critical Computer Systems, Neil Storey, ISBN 0-201-42787-7, 1996.

<TjF-FMV 1997:49> Bestämmelser för FMV typgranskning, provning och godkännande av system, utrustningar, apparater och datorprogram i militära luftfartyg.

<TjF-FMV 1997:11> Regler för Systemsäkerhetsverksamheten vid FMV (refererar till H SystSäk).

283. Kontroll 9709 har visat, att senaste version fortfarande utgörs av denna preliminärutgåva.

<Thane> Safe and Reliable Computer Control Systems, Concepts and Methods, TRITA-MMK 1996:13.

<Tracey> Integrating Safety Analysis with Automatic Test-Data Generation for Software Safety Verification, N.J. Tracey et al., Proceedings of the 17th ISSC, 1999.

<TÜV> Rules for programming in ”C”, TÛV Rheinland, Flick, www.isep.de/c-rules.htm, nov 1996.

<UL 1998> Standard for Software in Programmable Components, Underwriters Laboratories Inc[284].

<Vito_1> A Formal Model of Partitioning for Integrated Modular Avionics, Di Vito, NASA/CR-1998-208703, aug 1998.

<Vito_2> A Model of Cooperative Noninterference for Integrated Modular Avionics, Di Vito, DCCA-7, jan 1999.

<Voas> Prediciting How Badly ”Good” Software Can Behave, IEEE Software, Vol. 14, No 4, aug 1997.

<Watt> Firewalls in Safety-Critical Software Systems, G Watt, Proc. of the 18th ISSC, sep 2000.

<WEAG> Guidance on the Use of Progressive Acquisition, WEAG TA-13 Aquisition Programme, 1997.

<Wichmann> Can we learn anything from the MISRA C subset?, B. Wichmann, prel. version, aug 1997.

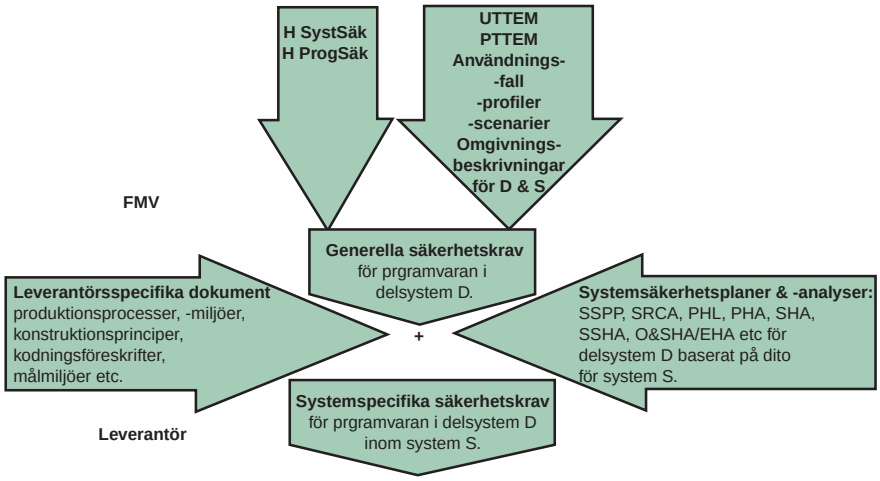
284. ANSI-förslag.

6.4 Anvisningar till H ProgSäk-användare

Detta vägledande avsnitt anger hur H ProgSäk kan anpassas till aktuellt projekt – ett arbete, som berör både beställare och leverantör. Sista delen ger förslag på allmänna förberedelser inför nya säkerhetskritiska projekt och vänder sig enbart till leverantör.

6.4.1 Anpassning av H ProgSäk

H ProgSäk innehåller i princip samtliga generella säkerhetskrav som kan ställas vid anskaffning av programvara i säkerhetskritiska tillämpningar (se 1.6.). Anpassning till enskilt projekt och system är därför nödvändig (jfr 1.9.). Krav relevanta för aktuell tillämpning väljs ut. Dessa kompletteras med systemspecifika säkerhetskrav baserade på resultat från fördjupade säkerhetsanalyser (jfr 3.4.4.). Arbetet inleds av beställaren och kompletteras successivt av leverantören.



Säkerhetskrav för D:s programvara i system S

6.4.1.1 *Ingångsmaterial*

– Initialt (FMV)

- Systembeskrivningar[285]
- Användningsprofil för aktuellt system[286]
- Omgivningsbeskrivningar[287]
- Anpassad H SystSäk
- H ProgSäk

285. Beskrivning av ett system på högsta nivå eller ett självständigt (del)system inom detta (t ex UTTEM, PTTEM).
286. Exempel: Användningsfall, dimensionerande scenarier, skottprofiler.
287. Omgivning: Närmast övergripande system samt det aktuella systemets övriga omvärld (t ex miljöprognos, typmål).

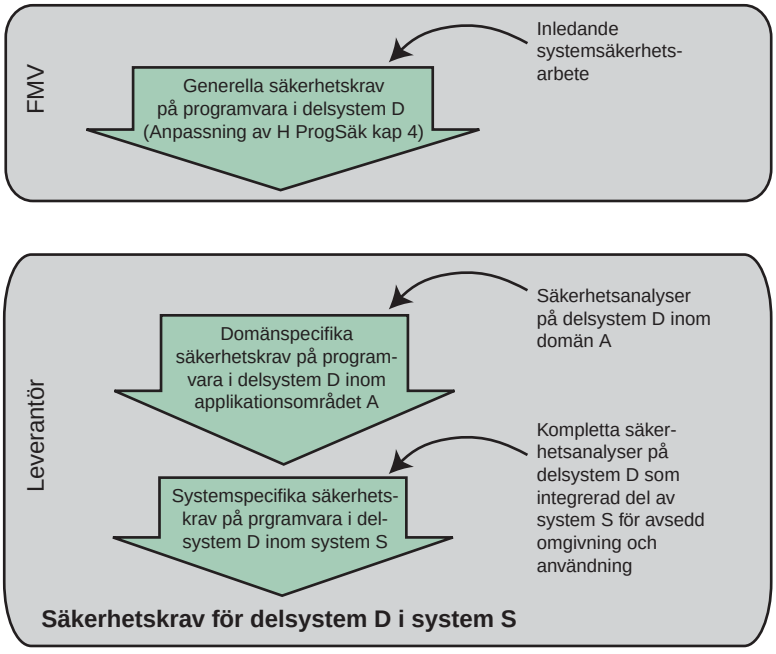
(FMV & Leverantör)
- Dokumentation över det inledande systemsäkerhetsarbetet[288].
- Leverantörsdokumentation över programvaru process, -produkt, -miljö[289].

– Systemrealisering (Leverantör)
- Resultat från fördjupade säkerhetsanalyser av successivt detaljerade kritiska delar.

6.4.1.2 *Resultat*

– FMV: Anpassad H ProgSäk med
- generella säkerhetskrav baserade på H ProgSäk kap 4.[290]

– Leverantör: Dokumentation över
- generella och systemspecifika säkerhetskrav



288. Enl anpassad H SystSäk för aktuellt system eller närmast högre systemnivå. Exempel: SSPP, PHL, PHA, Definition och värdering av konsekvens-sannolikhet, riskmatris.
289. Inkl Konstruktionsprinciper, Kodningsföreskrifter etc.
290. Dokumentet kan antingen refereras från (eller dess krav inkluderas i) TS, VS, VÅ. Det första alternativet underlättar för leverantören att ta fram en detaljerad variant av dokumentet.

6.4.1.3 *Aktiviteter (FMV)*

Ange:

- Dokumentets:
  - Inriktning — Systemsäkerhetskrav relevanta för applikationsprogramvaran.
  - Förutsättningar — Systemsäkerhetsverksamhet utförd före anpassning av H ProgSäk.
  - Omfattning — Generella säkerhetskrav anpassade från H ProgSäk kap 4. (Leverantör).
  - Avgränsningar — Grundkrav för programvara enl H ProgSäk kap 5. inkluderas ej i anpassad H ProgSäk, men gäller (se punkt d nedan).
  - Giltighet — a) Giltighet för dokumentet i relation till andra aktuella handböcker/regelverk.

    b) Företrädesval vid konflikt mellan dessa (alt: hänvisning till kontrakt). Notera att

    c) H ProgSäk är utgångspunkt för de generella säkerhetskraven och skall användas vid eventuella oklarheter i anpassade säkerhetskrav samt

    d) Grundkrav enl H ProgSäk kap 5. gäller eller annat angivet, likvärdigt dokument.
  - Referenser — Exempel: Referenser till använda ingångdokument.
  - Historik — Exempel: Nr, datum, dokumentstatus, upprättande instans, typ av ändringar för aktuella och tidigare versioner.
- Aktuellt system:
  - Bakgrund — En översiktlig beskrivning av aktuellt system.
  - Tillämpningsområden — Ett eller flera applikationsdomän (t ex fartyg, flyg, fordon, ledning).
  - Riskmatris — Avser aktuellt system och härleds från motsvarande matris på översta systemnivå.

– Programvarans kritikalitet	Mappning av H ProgSäk:s kritikalitetsklasser H, M, L på riskmatris för aktuellt system[291].
– Preliminär identifiering av kritiska program-delar	Uppgifter hämtas ur resultat från tidigare säkerhets-analyser och utgör utgångpunkt för djupare analyser.
– Principer för risk- och kostnads-reduceringar	Grundläggande principer att poängtera eller ta ställ-ning till i tidigt skede, t ex: • Graden av formalism vid specificering och verifie-ring av högkritiska delar • Beredskap och öppenhet för alternativa implementa-tionstekniker[292] • Säkerhetsinriktade arkitektur- och konstruktions-principer – En genomtänkt systemsäkerhetsarkitektur från högsta nivå – Okomplicerade, deterministiska och testbara systemlösningar – lsolering av kritiska delar[293] – Redundans och diversitet för minskade olyckssannolikheter[294] – Separata skyddssystem som ytterligare barriär mot ev riskkälleexponering[295]

291. Exempel: För risk (*p*, *k*), där *p* är sannolikhet för olycka (t ex vanlig, trolig, tillfällig, försumbar, osannolik) och *k* konsekvens (katastrof, kritisk, marginell, försumbar) gäller:

H ProgSäk kritikalitetsklass	Risk *K*: konsekvens &	*P*: Sannolikhet
H	Katastrofal	Vanlig, Trolig
M	Katastrofal Kritisk	Tillfällig Vanlig, Trolig
L	Kritisk Marginell	Tillfällig Vanlig

Notera: Övergripande krav på systemsäkerhet innebär att säkerhetskraven på eventuell
- styrning av säkerhetskritiska delar avser systemsäkerhet
- övervakning (av styrd enhet resp av styrning) samt skydd av person, egendom och miljö avser tillförlitlighet

Exempel: Ett vapensystem har funktioner för styrning och övervakning. Systemsäkerhet är därmed en väsentlig egenskap.
Det kan också ses som ett skyddssystem med uppgift att identifiera och nedkämpa angrepp. I dessa funktioner är tillförlitlighet (funktionssäkerhet) en nödvändig förutsättning för att systemsäkerheten skall vara tillgodosedd.

292. Avvägningar mellan att realisera i programvara resp maskinvara: Kan vissa risker undvikas med annan lösningsteknik?
293. Syftet är att till omfång och antal få ned delar som kan påverka de kritiska delarna (t ex genom att begränsa informationsflöden och beroendekedjor genom kritiska delar, fysisk/logisk separering).
294. Jfr H ProgSäk 4.5.2.4.2.: Riskreduktion genom redundans (t ex alternativa höjd-bestämningar via tröghetsnavigering, radar, tryck).
295. Exempel: Motmedel, aktiv bullerdämpning, magnetskydd, raketstol.

– Förslag på risk- och kostnads-reduceringar	Innehållet till denna rubrik kan lämnas tom. Om exempel inkluderas – föreskriv inga lösningar. Det är leverantören som tar ansvar för dessa, inte beställaren. Eventuella förslag värderas, kompletteras/förkastas/fördjupas och fastläggs av leverentör under systemrealisering och konstruktion.

- Specifika säkerhetskrav[296]:

– Prioriteringar vid resurs-begränsningar	Fastlägg systemuppgifter att prioritera vid plötsliga resursbortfall under drift, t ex: – Inbördes prioritet mellan systemets säkerhetskritiska uppgifter[297]. – Acceptabel degraderingsgrad för övergripande kritiska systemfunktioner[298]. – Eventuella möjligheter till omkonfigureringar under drift[299]
– Kända ändringar i senare serier	Om det redan är känt att slutleverans kommer att följas av a) ny beställning byggd på aktuellt system b) där vissa tillägg, modifieringar etc redan är kända c) dessa ändringar kan tänkas påverka kritiska delar Ange: – Kända, framtida ändringar Kräv: – Säkerhetsanalyser skall även utreda kända ändringars verkan på kritiska delar. – Vald konstruktion skall kunna utestänga eller minimera ändringars påverkan.

- Generella säkerhetskrav

– Tillämpliga krav från H ProgSäk:s kap 4.	– Använd för anpassad H ProgSäk helst samma kapitelnummer som i H ProgSäk. – Ange orsak till utelämnat H ProgSäk-krav[300]. Vid riktad upphandling och där ett krav ersatts med referens till motsvarande leverantörsdokumentation bör motsvarande, likvärdiga krav i detta anges.

296. Leverantör kompletterar dessa efter fördjupade säkerhetsanalyser.
297. Gradering av vilka uppgifter, som i alla lägen skall kunna utföras full ut, vilka som kan utföras delvis (eventuellt med lägre precision, prestanda, feltolerans etc) och vilka som kan utgå helt.
298. En mer detaljerad kartläggning av vilka försämringar som kan tolereras betr precision, prestanda, feltolerans etc utan att de mest högprioriterade uppgifterna äventyras. Ge hänvisning till motsvarande information i TS.
299. Exempel: Omlagring av systemet på färre noder i händelse av nodbortfall.
300. Exempel: ”Ej aktuellt p g a xxx”, ”Likvärdigt krav ställs i yyy”.
Möjliggör spårbarhet till uteslutet krav att aktualisera vid förändringar, som gör tidigare motiveringar inaktuella.

6.4.1.4 *Aktiviteter (Leverantör)*

Komplettera dokumentationen över programvarans säkerhetskrav m a p:
Dokumentets
- Omfattning Ingående säkerhetskrav i slutlig dokumentversion:
  - Generella (anpassade från H ProgSäk kap 4.)
  - Domängemensamma (vid flera möjliga tillämpningsområden)
  - Domänspecifika (för **ett** av dessa tillämpningsområden)
  - Systemspecifika krav (unika för aktuellt system).
- Referenslista över använda ingångsdokument
- Historik

• Identifierade kritiska programvarudelar
- Sök den kritiska kärnan i varje kritisk del, genom fördjupad säkerhetsanalys.
- Utred vilka dataflöden som löper genom kritiska delar och dess kärna.
- Finn vilka delar dessa kritiska dataflöden interagerar med.

• Risk- och kostnadsreduceringar
- Utred var riskreducering skall genomföras.
- Analysera ur risk- och kostnadssynpunkt nödvändiga och möjliga lösningar[301].

• Specifika säkerhetskrav
- Härled dessa ur utförda säkerhetsanalyser.
- Ge referens till den säkerhetsanalys (eller motsvarande dokument), som motiverar kravet.

6.4.2 Leverantörsförberedelser

Programvarans systemspecifika säkerhetskrav är unika för ett system i dess avsedda användning och omgivning. De härleds genom anpassning och komplettering av de generella säkerhetskraven för programvara enl H ProgSäk (ett arbete, som initieras av beställaren och slutförs av leverantören, se 6.4.). De unika kraven berör till stor del själva programvaruprodukten och kan m a o inte preciseras eller bedömas förrän ett aktuellt system föreligger.

För säkerhetskrav som avser personal, processer, produktionsmiljö och till någon del de generella säkerhetsprinciperna för en programvaruprodukt, är det dock möjligt att göra en förberedande genomgång av kravuppfyllnaden baserad på företagets dokumenterade regelverk och kompetensprofil. Denna

301. Alternativt: Referera till annat dokument där detta framgår.

inventering behöver inte upprepas inför varje ny säkerhetskritisk produktion, såvida inte profil, verksamhetsrutiner eller någon annan produktionsförutsättning förändrats efter inventeringstillfället.

Förberedande aktiviteter inför kommande produktion och leverans av säkerhetskritiska system:

- Inventera företagets dokumenterade regelverk (rutiner, metoder, tekniker, verktyg) och kompetensprofil m a p programvaruteknik och systemsäkerhet.
- Undersök i vilken grad dessa möjliggör, att generella säkerhetskrav enl H ProgSäk kap 4. och projektgemensamma, obligatoriska grundkrav enl avsnitt 5.2. kan uppfyllas.
- Identifiera på vilka punkter brister i systemsäkerheten föreligger.
- Lämna förslag på åtgärder, som undanröjer bristerna.

  Exempel:

  – Komplettera företagets kompetensprofil genom utbildning/rekrytering
  – Förbättra styr- och produktionsprocesserna, t ex beträffande
    ⊕ viss säkerhetsmetodik
    ⊕ formell specificering och verifiering
    ⊕ testteknik m a p viss fas/aspekt
    ⊕ resursanalyser
  – Anskaffa stöd- och programutvecklingsverktyg
  – Dokumentera policy och programmeringsprinciper vid återanvändning
  – Se över grundläggande och säkerhetsinriktade konstruktionsprinciper
  – Uppdatera Kodningsföreskrift för använda implementationsspråk
- Genomför förbättringsförslag och uppdatera dokumentationen m a p dessa
- Upprätta en korsreferenslista över krav och den dokumentation, som visar att visst krav är uppfyllt
- Ange vilka generella säkerhetskrav, som ej beaktats eller utretts klart vid inventeringen, orsak till detta samt tidpunkt när arbetet kommer att slutföras[302].

För en leverantör av säkerhetskritisk programvara är det väsentligt att systemsäkerhetsverksamheten och det traditionella programvaruutvecklingsarbetet är väl integrerade (jfr 4.3.). En investering i teknik för programvarusäkerhet är inte bara nödvändig utan också fördelaktig – sett både från ett ekonomiskt och ett marknadsmässigt perspektiv.

302. Exempel: Kravet/avsnittet ej utrett p g a resursbrist, men kommer att behandlas till/före/senast xx-xx-xx.
Kravet/avsnittet ej utrett: förutsätter tillgång till projekt-/systemspecifik information.
Kravet/avsnittet uppfyllt för projekt/system ABCD, se XXX.

6.5 Checklistor

Detta avsnitt innehåller checklistor att användas vid granskning av olika delprodukter beskrivna under kapitlen 2.–4. Listorna är strukturerade i huvudfrågor och i vissa fall följdfrågor. En huvudfråga besvarad med nej bör följas upp, för att utreda var problem kan föreligga och om behov av uppdateringar föreligger, t ex av säkerhetskrav, konstruktionsrestriktioner eller tidigare analyser. Listorna är ej fullständiga, varför kompletteringar kommer att bli aktuella. Strykningar kan vara motiverat i den takt analysverktyg, som kan överta en del av kontrollerna, blir tillgängliga. Väl genomarbetade checklistor kan i det senare fallet utgöra underlag vid specificering av sådana verktyg.

Verktyg för specificering och kontroll (t ex m a p fullständighet, motsägelsefrihet) finns[303], som skulle kunna minska frågelistan under [3.4.4. Teknisk specifikation] nedan. Fortfarande saknas dock möjlighet att uttrycka samtliga krav (t ex vissa realtidsegenskaper[304]) formellt eller att med verktyg bedöma kravens korrekthet.

Varje krav kan omformuleras till en motsvarande kontrollfråga. Detta har inte utförts, i huvudsak för att undvika dubbelskrivningar och därmed sammanhängande konsistensproblem.

Rubriken till varje checklista refererar till det avsnitt i handboken, som beskriver den delprodukt eller aktivitet, som är föremål för granskning.

[3.3.3.1. Ändringar av färdigt system]

Granskningsunderlag: Ändringsspecifikation samt dokumentation över kritiska programvarudelar, tester samt säkerhetsanalyser.

Checklista:

- Är specificerade ändringar precisa, otvetydiga, motsägelsefria och fullständiga?
- Har specificerade ändringar förutsatts eller förberetts i tidigare specificeringar och konstruktionslösningar?
- Är specificerade ändringar grundade på interaktion / kommunikation mellan ändringsställare, godkännande ändringsråd och möjliga implementatörer?
- Har en analys av ändringarnas inverkan utförts eller infordrats?
- Har en förnyad säkerhetsanalys utförts m.a.a. dessa ändringar?

303. Exempel: SpecTRM-verktygen med logiskt användarsnitt (*and-or* tabeller) på formell bas ger läsförståelse även åt en icke formellt skolad användare samt formell stringens, analys och exekvering. Första leverans går till NASA 9909.
304. Exempel: Tidskrav, *overflow*, interaktion SW/HW.

- Har den förnyade säkerhetsanalysen förts tillräckligt långt?
  - Finns brister i säkerhetsanalysen, som kan äventyra säkerheten?
- Vilka ändringar i system, omgivning eller användningssätt har bäring på programvaran?
  - Har dessa effekt på programvarans kritiska delar?
  - Kan dessa medföra, att tidigare säkerhetskritiska moment/delar minimeras/elimineras?
  - Går det att undvika att nya, säkerhetskritiska moment/delar tillförs?

Se även kontrollfrågor nedan under avsnitt [4.3.3.F Ändringar under produktion].

[3.4.4. Teknisk specifikation]

Granskningsunderlag: Teknisk specifikation

Checklista:

- Vilka krav går ej att verifiera? Vilka kan förenklas eller elimineras?

Säkerhetsanalyser

- Vilka systemsäkerhetsanalyser har utförts?
- Vilka vådahändelser har identifierats och vilka riskkällor ligger bakom dessa?
- Vilka riskkällor kan undvikas genom komplettering av kraven?
- Vilka antaganden har gjorts angående hur kontrollerat system resp nyttjade processorer skall operera?

Gränsytor

- Vilka förhållanden gäller för systemets omgivning?
- Finns entydigt definierat hur systemet skall agera vid olika förhållanden i omgivningen[305]?
- Vilka aktörer har inte konsulterats betr sin roll i system och operativ omgivning?

Operatör

- Vilka uppgifter skall operatör ha i systemet[306] ?
- Är operatörsrollen aktiverande och kompetensuppehållande?
- Har för varje operatörsuppgift bestämts, vad systemet skall göra, om operatör ej hinner (eller kan) agera eller utför en oplanerad handling?

305. Exempel: Specificera systemets respons som följd av olika systemstimuli under skilda externa villkor. Kontrollera: Har samtliga möjliga stimuli/externa villkor/respons täckts in?
306. Övervakare, mänskligt *back-up* system, partner?

- Har en riskkälleanalys utförts på de operatörsroller som ingår?
- Se även frågor nedan under [4.3.3.C Gränsytor].

Systemtillstånd

- Vilka tillstånd/moder kan kontrollerat system inta?
- Finns entydigt definierat vilka övergångar som systemet kan göra till / från varje tillstånd?
- Har för varje tillstånd maximal/minimal väntetid på inmatning resp resultat angetts samt hantering vid missat tidsspann?
- Finns för varje tillstånd föreskrivet hantering vid överlast (t ex för mycket data under givet tidsintervall)?
- Framgår under vilka villkor presenterad information, knappar och menyer skall ändras?

Framtida ändringar

- Vilka ändringar av systemets uppgifter kan förväntas?
  - Vilka av dessa har bäring på programvaran?
  - Vilka berör de kritiska delarna?
- Finns specificerade krav på en konstruktion, som klarar förväntade förändringar utan onödiga kostnadsökningar?
- Finns specificerade krav på utveckling av stöd, för att underlätta införandet av förväntade förändringar?

För varje svar på ovanstående frågor:

- Är dessa korrekta, fullständiga, motsägelsefria?
- Framgår dessa uppgifter klart i specificerade krav?

Ovanstående frågeställningar är aktuella under hela systemutvecklingen och svaren blir styrande för ingående komponenter på alla nivåer.

[4.3.3. Systemsäkerhetsanalys på programvara]

Checklistorna nedan kan användas vid genomgång av resultat från olika säkerhetsanalyser, dels av dem som utför själva analyserna, dels av dem som har till uppgift att granska analysresultaten. Utgångpunkt för analyserna är de representationsformer av programvaran, som föreligger vid tidpunkten för analys, dvs allt från krav, arkitektur, gränsytespecifikationer, konstruktion, implementation och fram till ändringsspecifikationer.

[4.3.3.A Programvarukrav]

Granskningsunderlag: Dokumentation av utförda systemsäkerhetsanalyser, systemarkitektur och systemgränsytor samt härledda programvarukrav.

Checklista:

- Har säkerhetsanalys av aktuella programvarukrav utförts och dokumenterats?
- Är analysen baserad på aktuella resultat från säkerhetsanalys från närmaste systemnivå?
- Har relevant dokumentation nyttjats vid analysen (t ex av programvarukrav på systemnivå, gränsytor och aktuell programvaruspecifikation) ? (Har uppmärksamhet riktats mot
  a) speciella krav,
  b) gränser/randvillkor,
  c) händelseordning,
  d) tidsaspekter,
  e) beroenden mellan gränssättande parametrar,
  f) röstningalgoritmer,
  g) krav på farliga kommandon,
  h) krav på övervakning av farlig maskinvara,
  i) varningar och restriktioner från inkopplade enheter/gränsytor,
  j) feldetektion och -isolering/-eliminering,
  k) feltolerans och -återhämtning?
- Var kan det finnas oklara/osäkra faktorer eller bristande underlag, som kan inverka på analysresultatens tillförlitlighet? Vad är orsaken till bristerna?
  – Kan orsaken vara ofullständig kravspecificering?
  – Vilka åtgärder skulle kunna förbättra analysens tillförlitlighet?
- Har säkerhetskrav för programvaran identifierats?
  – Finns programvarudelar, som kan orsaka/leda till olycka?
  – Finns programvarudelar, som del i säkerhetsövervakning och -skydd?
  – Finns riskkällor, som saknar ett matchande säkerhetskrav[307]?
- Föreligger spårbarhet mellan programvarans säkerhetskrav och systemets säkerhetskrav?
  – Har systemets säkerhetskrav fördelats till programvaran korrekt, fullständig och motsägelsefritt?
  – Kan säkerhetskritiska hot ej identifierade på systemnivå föreligga?
  – Är tidigare fördelningar ned till programvara från systemnivå och gränsytor korrekta?

307. Exempel: Kravet att riskkällan ej skall föreligga.

- Har identifierade risksituationer eliminerats / minimerats till tolerabel nivå enligt 1.5.

  (t ex genom ändringar på system-, gränsyte- eller programvarunivå, utbyte av programvaruteknik mot annan teknik)?
- Har graden av kritikalitet för programvarudelarna bestämts?
- Har säkerhetsmässiga rekommendationer för efterföljande fas, speciellt konstruktion och test, tagits fram?

[4.3.3.B Arkitektur]

Granskningsunderlag: Programvarans kravspecifikation, testplan, dokumentation av programvaruarkitektur och Konstruktionsprinciper[308], samt resultat från tidigare säkerhetsanalyser.

Checklista:

- Har säkerhetsanalys av aktuell programvaruarkitektur utförts och dokumenterats?
- Är analysen baserad på aktuella resultat från säkerhetsanalys av programvarukraven?
- Har relevant dokumentation nyttjats vid analysen (t ex av programvaruarkitektur och tidigare systemsäkerhetsanalyser)?
- Var kan det finnas oklara/osäkra faktorer eller bristande underlag, som kan inverka på analysresultatens tillförlitlighet? Vad är orsaken till bristerna?
  - Kan orsaken vara ofullständiga eller motsägande specifikationer?
  - Hur pass heltäckande för aktuellt system är den policy för felhantering som definieras i Konstruktionsprinciper? Har kompletteringar gjorts för aktuellt system?
  - Förekommer avsteg från föreskrivna Konstruktionsprinciper?
  - Vilka åtgärder skulle kunna förbättra analysens tillförlitlighet?
- Har kritiska programvarukomponenter identifierats?
- Har programvarukomponenter, som påverkar kritiska delar, identifierats som kritiska?
- Hur är det tänkt att program- och maskinvara skall samverka för att upprätthålla den tolerabla systemsäkerhetsnivån?
- Vilken lösning för kommunikation, distribution, synkronisering, skedulering och resursfördelning, avbrotts- resp felhantering har valts och hur påverkar dessa de kritiska delarna?

308. Se 4.5.2.2.

- Vilka marginaler finns för kritiska delars behov av minne och processorutnyttjande?
- Har identifierade risksituationer eliminerats/minimerats till tolerabel nivå enligt 1.5.
  (t ex genom ändringar på system-, gränsyte- eller programvarunivå, utbyte av programvaruteknik mot annan teknik)?
- Har graden av kritikalitet för programvarudelarna bestämts?
- Föreligger spårbarhet mellan säkerhetskrav på denna övergripande konstruktionsnivå och närmaste systemnivå?
  - Har en korrekt, fullständig och motsägelsefri fördelning gjorts av programvarans övergripande säkerhetskrav till aktuell konstruktionsnivå?
- Har säkerhetsmässiga rekommendationer för efterföljande fas, speciellt detaljerad konstruktion, tagits fram?
- Har täckande testfall för programvarans säkerhetskrav tagits fram?
- Har rekommendationer för motsvarande testprocedurer förberetts?

Fler kontrollfrågor kan härledas ur 4.5.2.2.

[4.3.3.C Gränsytor]

Granskningsunderlag: Programvarans kravspecifikation, gränssnittspecifikationer, testplan, dokumentation av programvaruarkitektur, resultat från tidigare säkerhetsanalyser samt information ur felrapporteringssystem.

Checklista:

- Har säkerhetsanalys av aktuella gränssnitt utförts och dokumenterats?
- Är analysen baserad på aktuella resultat från föregående säkerhetsanalys?
- Har relevant dokumentation nyttjats vid analysen (t ex av programvaruarkitektur och tidigare systemsäkerhetsanalyser)?
- Var kan det finnas oklara/osäkra faktorer eller bristande underlag, som kan inverka på analysresultatens tillförlitlighet? Vad är orsaken till bristerna?
  - Kan orsaken vara ofullständiga eller motsägande specifikationer?
  - Vilka åtgärder skulle kunna förbättra analysens tillförlitlighet?
- Har för systemsäkerheten kritiska gränssnitt identifierats, t ex
  a) riskkällor som påverkar mer än ett delsystem (och kritisk interaktion mellan dessa),
  b) externa komponenter/(del)system, som levererar kritiska indata eller vars felbeteenden eller synkroniseringar med andra delar kan utgöra en riskkälla[309],

309. Exempel: Felbeteende i form av inkorrekt funktionalitet eller funktionalitet utanför stipulerat tidsspann.

c) säkerhetskritiska avbrott och avbrottshanterare[310],

d) kritiska funktioner, vilka behöver kontrolleras genom inbyggda test (BIT),

e) protokoll för överföring av kritiska data mellan systemets processorer,

f) minne/databas, som delas mellan processer och som lagrar/förmedlar kritiska data,

g) systemsäkerhetskritisk information presenterad för operatör,

h) operatörens styrning/påverkan av kritiska funktioner,

i) mänskliga felgrepp, som kan inverka på säkerheten?

– Finns möjlighet att reducera riskerna med ovanstående gränssnitt ytterligare?

- Har analys utförts av operatörs, installations-, drifts- samt underhållspersonalens aktivitetskedjor för identifiering av aktiviteter[311] med säkerhetskritiska konsekvenser?
  - Kan användning av systemet under felaktiga betingelser ge kritiska effekter?
  - Vilka kritiska uppgifter klaras ej vid kontinuerlig drift utöver fastlagd tidsgräns?
  - Kan flera samtidiga operatörsfel leda till kritisk situation?
- Har analys av varje operatörsuppgift utförts för varje operationell systemmod?
- Är avvägningen mellan datorstyrda resp operatörsstyrda kritiska funktioner lämplig?
- Vilken roll skall operatörerna ha i systemet[312] ?
  - Har en riskkälleanalys utförts på samtliga operatörsroller?
  - Är operatörsrollen aktiverande och kompetensuppehållande?
  - Är operatörsuppgifterna genomförbara? Finns tid att reagera?
  - Används flerstegskommando[313] inför farlig aktivitet?
  - Ges *feed-back* och möjlighet till självkorrigering?

310. Vilka är skälen till att avbrottsteknik valts? Är avbrottshanteringen enkel och lätt att förstå? Klarar den en säker hantering av register och delade variabler? När och var kan avbrott inträffa? Vilken är den maximalt tillåtna frekvensen för avbrott?
311. Kan utfört moment i samband med någon extern händelse vid visst programtillstånd leda till fara? Jämför H SystSäk:s O&SHA/EHA
312. Övervakare, mänskligt *back-up* system, partner?
313. Ladda, osäkra, sikta, avfyra.

  - Är informationsmängden smältbar?
  - Vilka fel eller oegentligheter uppenbaras ej för operatör?
  - Vilka åtgärder kommer att vidtagas för att minska möjligheten till dolda fel?
  - Informeras operatören om fel, tillstånd eller händelser, som kan leda till fara?
  - Har informationskedja och menysekvenser presenterade i olika nödlägen simulerats och analyserats m a p risk för överlast av operatörens uppfattningsförmåga?
  - Är informationen i nödläge nödvändig och tillräcklig?
  - Har operatören i dessa lägen möjlighet, att återföra systemet till säkert tillstånd?
  - Är operatörens möjligheter till återställning av systemet i riskläge bra utformade?
  - Har konsekvenserna vid total blockering av operatörens interaktionsmöjligheter utretts?
- Är de implementerade degraderingsmoderna på tolerabel risknivå?
- Finns barriärer inrättade mellan potentiellt riskfyllda systemtillstånd och andra tillstånd?
- Har identifierade risksituationer eliminerats/minimerats till tolerabel nivå enligt 1.5.

  (t ex genom ändringar på system-, gränsyte- eller programvarunivå, utbyte av programvaruteknik mot annan teknik)?

  - Vilka risker har inte hanterats genom konstruktionsändringar?
  - Har de säkerhetsrisker, som kvarstår och måste övervakas, omvandlats till programvarukrav?
- Har rekommentationer för motsvarande testprocedurer förberetts?
- Har information väsentlig för säkert handhavande införts i motsvarande användarmanualer?

Fler kontrollfrågor beträffande gränsytor kan härledas ur 4.5.2.8., 4.5.2.13. samt från exempel under 4.5.1.

[4.3.3.D Detaljkonstruktion]

Granskningsunderlag: Programvarans kravspecifikation, testprocedurer, dokumentation av den detaljerade konstruktionen, resultat från tidigare säkerhetsanalyser samt information ur felrapporteringssystem.

Checklista:

- Har säkerhetsanalys av aktuell detaljerad konstruktion utförts och dokumenterats?
- Är analysen baserad på aktuella resultat från föregående säkerhetsanalys?
- Har relevant dokumentation nyttjats vid analysen (t ex av detaljerad konstruktion och tidigare säkerhetsanalyser)?
- Var kan det finnas oklara/osäkra faktorer eller bristande underlag, som kan inverka på analysresultatens tillförlitlighet? Vad är orsaken till bristerna?
  - Kan orsaken vara ofullständiga eller motsägande specifikationer?
  - Vilka åtgärder skulle kunna förbättra analysens tillförlitlighet?
- Förekommer avsteg från föreskrivna Konstruktionsprinciper[314]?
- Har tidigare identifierade kritiska programvarukomponenter brutits ned till kritiska samt icke-kritiska enheter på lägre nivå?
- Har samtliga programvaruenheter, som påverkar kritiska delar, identifierats som kritiska?
- Har identifierade risksituationer eliminerats/minimerats till tolerabel nivå enligt 1.5.

  (t ex genom ändringar på system-, gränsyte- eller programvarunivå, utbyte av programvaruteknik mot annan teknik)?
- Har graden av kritikalitet för programvarudelarna bestämts?
- Föreligger spårbarhet mellan säkerhetskrav på denna detaljerade konstruktionsnivå och närmaste systemnivå?
  - Har en korrekt, fullständig och motsägelsefri fördelning gjorts av programvarans tidigare säkerhetskrav och av säkerhetsinriktade konstruktionsrekommendationer till aktuell konstruktionsnivå?
- Har säkerhetsmässiga rekommendationer för efterföljande fas, speciellt kodningsfasen, tagits fram?
- Har täckande testfall för programvarans säkerhetskrav tagits fram?
- Har säkerhetsmässig information till användarmanualer etc. förberetts?

[4.3.3.E Implementation]

Granskningsunderlag: Programvarans kravspecifikation, testprocedurer, dokumenterad kod, Kodningsföreskrifter, resultat från tidigare säkerhetsanalyser samt information ur felrapporteringssystem.

314. Se 4.5.2.3.

Checklista:

- Har säkerhetsanalys av aktuell kod utförts och dokumenterats?
- Är analysen baserad på aktuella resultat från föregående säkerhetsanalys?
- Har relevant dokumentation nyttjats vid analysen (t ex av kod och tidigare säkerhetsanalyser)?
- Har hänsyn tagits till begränsningar förknippade med aktuell implementation, t ex unika plattformsegenskaper (antal processorer, minne avbrottshantering), teknik för process- och prioritetshantering, minnesallokering etc. ?
- Vilka negativa effekter för systemsäkerheten kan uppträda vid långvarig systemdrift?
  - Var finns risk för degradering av prestanda, störningar i skeduleringsalgoritmerna p g a prestandaförluster, avdrift vid tidmätning och dataöverföringar, samlad effekt av successiva beräkningsfel etc?
  - Vilka garderingar har gjorts mot kritiska effekter vid långvarig drift?
- Var kan det finnas oklara/osäkra faktorer eller bristande underlag, som kan inverka på analysresultatens tillförlitlighet? Vad är orsaken till bristerna?
  - Kan orsaken vara ofullständiga eller motsägande specifikationer?
  - Vilka åtgärder skulle kunna förbättra analysens tillförlitlighet?
- Förekommer avsteg från fastställda Kodningsföreskrifter[315]?
- Föreligger spårbarhet mellan säkerhetskrav på denna implementationsnivå och närmaste systemnivå?
- Finns funktionalitet implementerad, som ej kan spåras till kravspecifikationen?
  - Har en säkerhetsanalys beaktat effekten av eventuell extra funktionalitet?
  - Ingår även denna extra funktionalitet i objektkoden?
  - Är den exekverbar?
- Finns krav som bara delvis eller inte alls är implementerade?
- Har en korrekt, fullständig och motsägelsefri fördelning gjorts av programvarans tidigare säkerhetskrav och av säkerhetsinriktade konstruktionsrekommendationer till aktuell implementationsnivå?
- Har kritiska delar analyserats och verifierats m a p korrekt, fullständig och motsägelsefri implementation?

315. Se 4.5.2.7.

- Finns delar, som
  a) om de inte utförs, utförs inkorrekt, oavsiktligt, i fel ordning eller i vid fel tidpunkt, kan leda till ett osäkert tillstånd eller olycka?
  b) utövar direkt eller indirekt kontroll/styrning över potentiellt farliga funktioner eller maskinvara?
  c) övervakar kritiska maskinvarukomponenter?
  d) övervakar systemet m.a.p, möjliga kritiska situationer och/eller tillstånd?
  e) utför automatisk test under systemstart eller drift (t ex Funktionsövervakning, BIT) ?
- Har eventuella osäkra tillstånd identifierats (t ex sådana orsakade av datafel, för mycket/för lite data, tidsmissar vid I/O, multipla händelser, händelser i felaktig ordningsföljd, uteblivna förväntade händelser, negativa effekter från omgivningen, låsningar (*deadlock*), fel händelser, felaktiga storlekar, felaktig polaritet, enkelfel, gemensamma felorsaker, maskinvarufel)?
- Har identifierade risksituationer eliminerats/minimerats till tolerabel nivå enligt 1.5.
  (t ex genom ändringar på system-, gränsyte- eller programvarunivå, utbyte av programvaruteknik mot annan teknik)?
- Har interaktionen/kopplingarna mellan kritiska och icke-kritiska delar utrett?
- Har interaktionen mellan kritiska delar realiserats och dokumenterats korrekt, fullständigt och motsägelsefritt?
- Har de kritiska delarna kommenterats och dokumenterats tillfredsställande?
- Har täckande testfall för programvarans säkerhetskrav tagits fram?
- Har säkerhetsmässig information till användarmanualer etc. förberetts?

[4.3.3.F Ändringar under produktion]

Granskningsunderlag: Tidigare säkerhetsanalyser av berörda delar, ändringsbegäran samt information ur felrapporteringssystem[316].

Checklista:

- Har alla föreslagna programvaruändringar analyserats m a p eventuella negativa effekter, t ex att kritikaliteten höjs för tidigare identifierade kritiska delar eller tillförs tidigare icke-kritiska delar ?
  – Finns ändring, som kan
  a) leda till nytt, osäkert tillstånd

316. Jämför kap 5.:s [4.4.1.2.] samt 6.9.3.8.

b) öka sannolikheten för att hamna i osäkert tillstånd,

c) öka exponeringen vid osäkert tillstånd (m a p volym, tid, närhet, antal gånger),

d) negativt påverka kritiska programvarudelar

e) ändra/förvärra kritikalitet hos programvarukomponenter

f) medföra att nya säkerhetkritiska delar uppstår direkt eller indirekt

g) påverka säkerhetskontroll/riskövervakning?

- Har säkerhetsanalys utförts och dokumenterats för de fall, där ändringar kan ha effekt på kritiska delar?
  - Har dessa analyser baserats på aktuella säkerhetsanalyser av berörda delar?
  - Har relevant dokumentation nyttjats vid analysen (t ex ändringsunderlag, rapporter över fel, olyckor eller tillbud samt dokumentation, kod, analysresultat för ändrade delar och för delar påverkade av ändring)?
  - Har samtliga systemdelar, som påverkas av ändringarna, identifierats?
  - Har de utvecklingsgrupper, som påverkas eller behöver kännedom om genomförda ändringar konsulterats?
  - Har konsekvenserna även för påverkade delar utretts?
- Var kan det finnas oklara/osäkra faktorer eller bristande underlag, som kan inverka på analysresultatens tillförlitlighet? Vad är orsaken till bristerna?
  - Kan orsaken vara ofullständiga eller motsägande specifikationer?
  - Var behövs fördjupade säkerhetsanalyser?
  - Vilka åtgärder skulle kunna förbättra analysens tillförlitlighet?
- Vad krävs för att spårbarheten av systemets säkerhetskrav till berörda delar skall kunna upprätthållas och verifieras efter genomförda ändringar?
  - Hur garanteras att fördelningen av programvarans säkerhetskrav blir korrekt, fullständig och motsägelsefri?
- Har alternativa lösningar övervägts vid ändringar med negativ effekt på systemets säkerhet? Vilka av dessa resulterar i en tolerabel risknivå för systemet?
- Finns ändring, som minskar kritikaliteten i någon del? Vilka ändringar, vilka delar?
- Vilken dokumentation påverkas av ändring och behöver uppdateras?
- Hur skall den uppdateras, för att korrekt spegla effekter på systemsäkerheten?
- Vilka testrutiner påverkas och måste ändras?

Fler kontrollfrågor m a p ändringar kan härledas ur 3.3.3.1., 4.5.1. och 4.5.2.12.

6.6 Problemställningar

Allt fler utrustningar i vår omgivning styrs av programvara. Säkerhetskritiska tillämpningar vinner därmed terräng inom applikationsområden, där erfarenheterna av de problem denna typ av system medför kan vara begränsade. Nya programmeringstekniska landvinningar ger också tidigare oprövade realiseringsmöjligheter, vars effekter för systemsäkerheten ännu inte är fullt utredda. Detta kapitel beskriver översiktligt några allmänna problem för programvara i system med krav på systemsäkerhet.

6.6.1 Anskaffnings- och utvecklingsmodeller

Ett flertal etablerade anskaffnings- och utvecklingsmodeller för programvarusystem finns idag, som försöker komma tillrätta med programvarusystemens brister ifråga om kostnader för utveckling, underhåll, leveranstider och tillförlitlighet. Dessa bygger på tankar från 1960-talet och framåt – en utveckling från fri, artistisk kodning till en mer organiserad, professionell programvaruproduktion.

Till de tidiga och banbrytande begreppen hör strukturerad programmering, <Dikstra> 1968, och inkrementell krav- och systemutveckling, <Mills> 1980. Båda dessa kom senare tillsammans med formell teknik och statistisk testning att ingå i *Cleanroom*, <Mills> 1987, i dess inriktning mot felfri konstruktion (se 4.3.2.). En av de dominerande livscykelstandarderna blev DOD-STD2167A, 1988, en vattenfallsmodell med en funktionellt orienterad systemuppdelning och (som grund för systembygget) ett hårt reglerat, strikt sekvensiellt, dokumentstyrt flöde baserat på frysta kravspecifikationer, formella granskningar och milstenar. En populär variant av den sekvensiella modellen blev den V-formade, <STARTS> 1989. Idéer om iterativ och inkrementell utveckling av successiva systemversioner i spiralformade och evolutionära livscykler med objektorientering, prototyping och återanvändning som nya ingredienser kom att prägla 1990-talet. MIL-STD-498 (1994) begränsar sig följdaktligen till att definiera aktiviteter och leveransinnehåll (för anpassning till enskilt projekt av beställaren: ”vad”) och överlåter därmed till leverantören att välja lämpliga metoder, procedurer och dokumentationsformer (”hur”).

En reaktion mot dokumentstyrning och sena system- och integrationstest ledde till nya metoder för iterativ och inkrementell systemutveckling, där tidig produktion av kod och kundfunktionalitet poängteras. Exempel på detta är olika varianter av dagligt byggande, <Daily Build> 1995. I grundvarianten produceras exekverbar kod genom automatiserat systembygge och test. Vid felyttringar spåras felkällan till den systemfunktionalitet, som ändrats inför de senaste bygget, varvid funktionsansvarig får uppgiften att omedelbart rätta till, bygga upp och testa om systemet. Likartade idéer finns i XP, Extrem Programmering (med en aktiv slutanvändare för test av successiva systeminkrement) samt i Parprogrammering.

Svårigheter att i ett tidigt skede få fram en fullständig, komplett och motsägelsefri kravmängd för komplexa system, och behov av att kunna anpassa dessa till ny teknik, ledde genom ett västeuropeiskt samarbete till definition av en progressiv anskaffningsstrategi för försvarsinformationssystem, <WEAG> 1997. Denna modell kombinerar en evolutionär utveckling av kravmängden med en inkrementell utgivning av successiva systemversioner, för att möjliggöra ändringar av krav, utvecklingsplaner, finansiering och kontrakt inför varje nytt inkrement.

Ytterligare en trend under 1990-talet var en förskjutning av fokus från produkt till process och professionell kompetens. Olika studier av organisationers förmåga att effektivt kunna utveckla programvarusystem resulterade i ett antal mognadsmodeller, t ex SEI:s CMM (1991) och ISO/IEC JTCA/SC7:s SPICE (ca 1995), samt livscykelstandarder. ISO/IEC 12207 (1995) är ett ramverk för programvarans processer. ISO/IEC CD 15288 (2000) är motsvarigheten på systemnivå.

Hur går dessa modeller att tillämpa på system med säkerhetskritisk programvara? Ett väsentligt, generellt säkerhetskrav är, att leverantören väljer en etablerad modell, som inkluderar de systemsäkerhetsaktiviteter, som fordras, för att kunna försäkra sig om att levererat system uppfyller ställda krav på funktionalitet och säkerhet (4.3.). Har modellen ej tidigare använts för säkerhetskritiska system innebär detta en större bevisbörda (vilket lätt leder till en mer konservativ syn på valet).

Samtidigt pågår en ständig förbättring. Nya modeller och metoder utvecklas som svar på problem de tidigare inte kunnat tackla. De flesta av dessa är också anpassbara till olika projektbehov.

Låt oss se på exemplet Progressiv anskaffning (PA). Några av svårigheterna är förändrade krav inför ett nytt systeminkrement. Detta medför t ex att resultat från tidigare utförda systemsäkerhetsanalyser måste omprövas och analyserna göras om, för att utreda förändringarnas effekt (se 4.5.2.12., 3.3.3.1.). Ett annat är byte av leverantör inför ett nytt inkrement. En utredning av ansvarsförhållandena vid vidareuteckling av tidigare levererade delar måste utföras före överlämning till ny leverantör (jfr 6.6.5.). Situationen blir analog med fallet att en produkt modifieras före återanvändning (se 4.5.1. krav 3-4). Ett tredje problem är arkitekturändringar. Dessa är alltid kostsamma och medför dessutom att säkerhetsarkitekturen måste omprövas (4.5.2.2.).

Å andra sidan kan modellen ge möjligheter att styra upp ändringsbehoven. Vissa säkerhetsmässigt nödvändiga specifikationsändringar kanske kan sparas till de enstaka, i förväg definierade tillfällen, då ett nytt inkrement skall specificeras. En avvägning av PA-modellens konsekvenser för säkerhetskritisk programvara kan leda till att en modifierad variant tillämpas, t ex att PA endast används för de icke säkerhetskritiska delarna. I ett sådant fall tillkommer att visa, att de kritiska delarna verkligen är oberoende av de icke-kritiska, dvs att förändringar inför nytt inkrement inte har någon inverkan (se 4.5.2.4.2.).

Flera moderna utvecklingsmodeller verifierar systemet *top-down* i stället för den traditionella testordningen *bottom-up* (komponent-, integrations- och systemtest). Detta gäller exempelvis *Cleanroom*, dagligt byggande och XP. Fördelarna är snabbare besked om i vilken grad övergripande systemkrav är uppfyllda. Hur går denna *top-down*-testning ihop med säkerhetskrav under 4.3.2.2.5. och mostvarande grundkrav (5.2.3.3.)? Tidigareläggning av systemtesten utesluter inte, att test på lägre nivå för kritiska delar utförs före systemtesten. Krav på testtäckning av kritiska delar kan också tillgodoses före leverans av systeminkrement. Även här finns alltså möjlighet att tillämpa top-down på icke-kritiska delar kombinerat med *bottom-up* för kritiska delar.

Ovanstående exempel ger bara en antydan om hur nyare modeller kan anpassas för att kunna tillämpas på system med säkerhetskritisk programvara. De skall inte ses som en fullständig utredning. Dock illustrerar de återigen vikten av att hålla nere mängd och volym för säkerhetskritiska delar samt att satsa på en arkitektur, där kritiska delar är isolerade från icke-kritiska (6.7.).

6.6.2 Programvara

En programvaruimplementation skiljer sig från andra tekniska realiseringar på en rad punkter: I programvara dominerar konstruktionsaktiviteterna. Trots ökat verktygsstöd ingår många manuella moment, och därmed en exponering för mänskliga försummelser. Fel i programvara är därför huvudsakligen systematiska[317] och ej som för maskinvara slumpmässiga. Produktkvaliteten kommer till följd härav att i högre grad bero av produktionsprocess och personalkvalifikationer. Felens karaktär leder också till att andra typer av konstruktionslösningar måste sökas. Redundans i form av duplicering är t ex inte lämplig teknik (4.5.2.4., 6.9.5.: Ariane 5).

Tillverkningsprocessen är konceptuellt inriktad och ger upphov till en produkt utan fysisk påtaglighet, visualiserad m h a olika beskrivningar. Produktens mångfaldigande skapar identiska kopior utan variationer bland dess egenskaper. Underhåll säkras ej genom välfyllda reservdelslager utan beror på procedurer tillämpade vid utveckling och test samt egenskaper inbyggda i programvaruprodukt, testprogramvara och dokumentation. Programvaran kan visserligen bli föråldrad (t ex p g a ändringar i annan utrustning, omgivning eller

317. Programvarufelen är antingen **systematiska** och beror på brister i
   a) specifikation av system, driftsförhållanden och gränssnitt
   b) konstruktion
   c) produktionsprocess (som missat identifiera felen)
   d) produktionsverktyg

eller på **felaktig användning**, t ex
   e) återanvändning under andra förutsättningar
   f) användningsområde skilt från det som specificerats för systemet
   g) handhavande i strid mot specar och instruktioner.

Exempel:	Ariane 5	feltyp:	a-c, e
(Se 6.9.5.)	Patriot		f, g.

gränssnitt), men ökad ålder medför inte att nya fel uppstår. Fel orsakade av slitage efter upprepat eller långvarigt bruk uppstår inte. Däremot kan begränsningar i den numeriska representationen medföra, att små fel efter en tids exekvering ackumuleras till oacceptabla proportioner (vilket konstaterats då analoga system överförts till digitala, <Parnas>).

Programvarutekniken erbjuder enklare och billigare sätt att bygga in logik i systemen. Långt fler tillstånd kan realiseras än vid en renodlad maskinvarulösning. En högre grad av självkontroll kan utövas (vilket kan bidra till ökad tillförlitlighet). Mer information kan hämtas ut från systemet. Avgöranden och val av handlingsalternativ baserad på aktuell driftsituation kan automatiseras och utföras betydligt snabbare än vid mänsklig hantering. Allt mer av ansvaret för systemsäkerhet flyttas därmed över till programvaran. Samtidigt ökar komplexiteten (t ex vid styrning och övervakning i realtid, där fördelningen av processorresurser lätt kan medföra konflikter, se 6.6.3.). Programlogiken kan också ändras snabbare (såvida den inte kapslats in som fast program), även om själva genomförandet (dvs att analysera hur beslutade ändringar skall realiseras samt att verkställa och verifiera dem) tar tid. Införda ändringar kan dock vara svåra att upptäcka för en utomstående användare.

Programvarans abstrakta realisering innebär att de tekniska hinder som följer av en renodlat mekanisk lösning inte automatiskt ingår. Inga naturliga spärrar mot att generera resultat i strid med de fysikaliska lagarna föreligger. Ett syfte vid säkerhetsanalys av datoriserade system är därför att identifiera denna typ av lagbrott och införa begränsningar, för att förhindra dem. Många abstrakta egenskaper kan inte verifieras fullt ut förrän efter nedladdning i avsedd målmiljö, och i vissa fall först efter lång drifttid (även om en del kan analyseras under specificering, konstruktion och test). Detta är en kostsam process, som talar för värdet av tidiga, väl specificerade och strukturerade aktiviteter.

Egenskapen **säker programvara** är än mer abstrakt och uttrycker inte, att koden som sådan är säker, utan att de delar av systemet programvaran har till uppgift att **styra eller övervaka** ej hotar liv, egendom eller miljö, samt att programvaran till de delar, som har till uppgift att **skydda** mot dessa hot, fungerar tillförlitligt (se 6.7.). Säkerheten avser m a o programvaran som integrerad del av systemet i avsedd miljö och under specificerade förutsättningar.

Den abstrakta karaktären medför krav på medföljande dokumentation över produkt, process och utvecklingshjälpmedel (krav, som är hårdare, ju högre säkerhetsrisker, som föreligger.

Ett uttryck för de säkerhetsrisker systemets olika delar utgör är **kritikalitet**[318]. Samma resonemang gäller här: programvaran i sig är inte kritisk, men den effekt en felande programfunktion har på systemets säkerhet är ett mått på programvarudelens kritikalitet. Ju högre kritikalitet, desto hårdare krav på produkt, process, produktionsmiljö och personalkompetens.

318. Se vidare 6.1.5.

De säkerhetsanalyser och verifieringar, som krävs av nyproducerad programvara, gäller även återanvända programvarukomponenter samt verktyg, som använts vid produktion av koden. För icke programvarubaserade system torde ej samma stringens föreligga t ex beträffande tillverkningsverktyg. Oavsett om programvarukomponenterna är unika för systemet eller återanvända, så är det svårt att genomföra säkerhetsanalyser på samma vis som för massproducerade, ofta standardiserade maskinvarukomponenter. Statistik över felfrekvenser är svåra att få fram eller kan ej skattas från ett tillräckligt tillförlitligt underlag[319]. (För ytterligare aspekter på COTS-baserade system se avsnitt 6.6.5.).

6.6.3 Realtidssystem

Realtidssystem består i regel av ett antal delsystem uppbyggda av maskin- och programvara, där delar av programvaran kan vara inkapslad i maskinvaran (fast program, *firmware*). Delsystemen är ofta distribuerade över ett eller flera lokala nätverk (LAN). Upplagan, dvs antalet kopior som produceras av ett visst system, kan vara liten[320].

Realtidssystemets uppgift är, att m h a olika sensorer kontrollera och styra olika typer av utrustningar i verklig tid. Liksom traditionella, icke-realtidsbetonade system, skall det uppfylla krav på korrekt funktionalitet. Därutöver skall det kunna reagera på händelser i omvärlden, dvs ta emot data, utföra sina uppgifter och leverera resultat inom föreskrivna tidsramar och med de begränsade resurser, som föreligger[321]. Systemklockans tid, skall synkroniseras över noderna i nätet, så att observationer från olika sensorer av samma objekt kan tidsrelateras. Resultat skall presenteras i rätt ordning, även om de härrör från olika processer. Fel i (eller bortfall av) funktionalitet, data eller noder skall upptäckas och hanteras. Framför allt gäller det att isolera fel och störningar, så att dessa inte sprids vidare i nätet till andra, fortfarande funktionsdugliga delar (och i synnerhet ej till kritiska delar). De partitioneringar, som nyttjas behöver därför fylla två huvudsyften, – dels att möjliggöra en distribuerad funktionalitet[322], – dels att kunna separera delar av olika kritikalitet, så att fel i delar av låg eller ingen kritikalitet hindras påverka de av högre kritikalitet.

Med distribuerade realtidssystem blir problemställningarna mer komplexa. Både funktionalitet och tidskrav kan vara kritiska. Distribution av funktionalitet och data ställer krav på tidssynkronisering av avsänd och mottagen infor-

319. Se t ex analysmetoder i 6.8.
320. Gäller främst militära system konstruerade mot kundkrav till skillnad från konsumtionsvaror (som t ex bilar).
321. Klassningen i hårda, fasta eller mjuka realtidssystem anger hur pass allvarlig en sen eller utebliven respons är.
322. Fördelar med partitioneringar ligger bl a i möjlighet till prestandaförbättringar, genom att subsystem kan lokaliseras med samhörande enheter på nätet, att partitioneringen kan nyttjas för feltolerans (genom t ex redundanta subsystem).

mation från/till olika noder. Tidskrav kombinerade med begränsningar i minnes- och beräkningskapacitet ställer hårda krav på vald resursallokering. Utvecklingen är vital inom området: förbättrade eller nya algoritmer kommer fortlöpande[323] och realtidsstöd i form av s k *middleware* vidareutvecklas i nya varianter[324].

Antalet dimensioner och tillstånd blir mycket stort – i synnerhet för avbrottsstyrda system med flera parallella processer[325]. Denna typ av system kan för vissa applikationer ge en klarare lösning med bättre svarstider än ett klockstyrt system, men kan också lasta ned processorn vid felaktiga avbrott (p g a brus från någon sensor el. dyl.). Avbrott innebär i praktiken ett obegränsat antal kombinationer av möjliga händelsevägar genom systemet, vilket ökar komplexiteten. Detta gäller i synnerhet osynkroniserade avbrott, vilka är svåra att förutsäga, återskapa, felanalysera och hantera. Flera avbrottsstyrda processer, som tävlar om begränsade resurser, skapar vidare risk för låsningar (*deadlocks*) och prioritetskonflikter (t ex prioritetsinversion). Det gäller m a o, att välja väl beprövade och säkra algoritmer vid fördelning av systemets resurser och att inför val av modell kontrollera, att modellens förutsättningar är uppfyllda[326].

Säkerhetshoten är av ovanstående skäl svårare att förutsäga och avvärja[327] för realtidssystem än för icke tidsberoende system, i synnerhet, där nya komponenter, unika för applikationen, ingår.

Dessa problemställningar ställer höga krav, inte bara på systemen, utan även på kompetensen hos de, som skall bygga och verifiera realtidssystem – oavsett om dessa är säkerhetskritiska eller ej – och på de processer, som nyttjas i detta arbete. Säkerhetskritiska, distribuerade realtidssystem representerar m a o den mest komplexa typen av system att konstruera. Denna komplexitet går inte att undvika, då den är en inneboende egenskap hos den uppgift som skall lösas, och skall skiljas från den komplexitet, som oavsiktligt kan ha introducerats genom val av olämplig lösningsteknik (se 4.5.2.4.).

Intresset för att snabbare och billigare kunna producera de olika typer av inbyggda, reaktiva (och därmed säkerhetskritiska) system som industri och privatpersoner numera efterfrågar, har stimulerat forskningen att undersöka hur dessa system skall kunna specificeras och byggas med formella tekniker (<SACRES>). En dynamisk utveckling kan förväntas inom gränsskiktet formalism och realtidssystem.

323. Exempel: (1) <RAVEN> -på marknaden redan ett år efter definition av *tasking*profilen <Ravenscar> (1997). Se 4.5.2.4.
(2) Forskningsprojektet MANA, en beskrivning av Ravenscarprofilen för bl a formell verifiering, implementering och eventuell senare typcertifiering, <MANA>.
324. Exempel: CORBA och Java för realtid (<RT-CORBA>, www.j-consortium.org/rtjwg/index.html).
325. Jfr resonemang under 6.8.3. , 4.3.2.2.7., 4.5.2.4. samt <Lawson_1>.
326. Se fotnot vid ”Resurs- och tidshantering” under 4.5.2.4.
327. Jämför 6.1.2. Säkerhet.

6.6.4 Informationssystem / Ledningssystem

Informationssystem är i regel databasorienterade. Väl etablerad teknik och bra produkter i form av olika databashanteringssystem finns sedan länge. Standarder för gränssnittsspecificering (ISO IDL), distribution, objektorientering (t ex CORBA), API (applikationsprogramgränssnitt, t ex ISO DIS 14750) har utarbetats via organisationer som IETF (*Internet Engineering Task Force*) och betydligt snabbare än de officiella standardiseringsorganen. Gemensamma arkitekturer börjar specificeras. Ett exempel är JTA (*Joint Technical Architecture*) som definierar ett antal prestanda-orienterade, primärt kommersiella standarder för informations- och transaktionssystem, format, IT-säkerhet m m.

De flesta informationssystem som är säkerhetskritiska har anknytning till någon realtidstillämpning[328]. Ju mer sofistikerad information som behövs, desto sårbarare blir systemen för störningar och fel i insamlad data eller plötsliga förändringar i den operationella miljön. Att databasens IT-säkerhet är en väsentlig förutsättning för systemsäkerheten är uppenbar. Transportsystemens säkerhet beror t ex av att information i färdplaner, spår- och vägnätsscheman är korrekt, komplett och motsägelsefri. (Motsägelsefriheten innebär bl a att data har en tidsstämpling som är enhetlig för samtliga användande system). Ett bilnavigeringssystem kan behöva aktuella uppgifter om vägarbeten och omläggningar av trafikriktningen. För mer stabil information finns i många fall validerade data att tillgå. Där informationen är färskvara blir säkerheten beroende av, att kritiska data ständigt uppdateras och valideras. Valideringen stödjer sig fortfarande till stor del på manuella kontroller, men torde kunna underlättas genom Formell teknik (t ex för specificering av regler och samband mellan parameterar samt för verifieringar av dessa).

På sikt torde ett mer utvecklat verktygsstöd bli nödvändigt för att kunna analysera säkerhetskritiska dataflöden (flöden som påverkar säkerhetskritiska funktioner och aktiviteter), i synnerhet vid dynamiska förändringar, som kräver snabb validering av uppdaterad information. För framtida militära ledningssystem med målsättning att ge ett tidigt informationsövertag inför beslutsfattande och insatsledning tillkommer att klara insamling, analys och sammanställning av betydligt större informationsmängder väsentligt snabbare än dagens system, <Arnborg>. Ensad lägesinformation skall kunna lämnas med

328. Av denna anledning brukar sådana informationssystem benämnas **indirekt säkerhetskritiska**.
Exempel: (1) Signalsystem baserad på uppgifter om spårnät, spårsektioner, signalpunkter, tillåtna rutter, okuperade spårsektioner. (2) Kartdatabas (ev med GIS) för navigering (och ev även styrning) i realtid. (3) Hastighetshållare/-varnare baserad på aktuell hastighet samt krökningsradie för annalkande vägavsnitt. (En vägdatabas kan ha ca 150 attribut knutna till ett vägsegment). (4) Mindatabas för olika mintyper och fabrikat, uppgifter om röjda/oröjda områden m m. (5) Realtidsinformationssystem, där data från omgivningen tas emot, bearbetas och presenteras för operatör, vilken på grundval av detta fattar säkerhetskritiska beslut (t ex ATM, kemisk processindustri). (6) Varningssystem för marinhelikoptrar (larm vid sensorvärden utanför tillåtna gränser för aktuell manöver). (7) Medicinskt databassystem över läkemedel, patienter (diagnoser), blodgivare (blodstatus) etc.

en detaljeringsgrad anpassad efter användarens behov över olika typer av nätverk (stationära/mobila/trådlösa) – en utmaning såväl m a p system- som IT-säkerhet.

6.6.5 Standardprodukter – Återanvända Komponenter – Hyllvaror

Att inkludera tidigare utvecklad programvara i nytt system förutsätter en modifierad syn på systemutvecklingsprocessen. Större vikt måste läggas på kravspecificering, arkitektur, gränssnitt/standardiserade protokoll och integrationstest. I o m utvecklingen av industristandarder för s k *middleware* (basprogramvara i skiktet mellan applikation och underliggande operativsystem, protokollstackar och maskinvara) och återanvändbara generella komponenter baserade på dessa har detta arbete underlättats. System kan nu sammansättas från färdiga standardkomponenter i stället för att specialkonstrueras från grunden. Vinster i tid, pengar och kvalitet förväntas – inte minst därför att en intensivare användning i flera system leder till att färre fel kan leva vidare oupptäckta. En standardkomponent skulle m a o snabbare kunna nå högre tillförlitlighet än en nyutvecklad specialkomponent.

Trots detta kan återanvändning innebära ett hot mot systemsäkerheten. Tillförlitlighet påvisad i ett visst system, omgivning och användning behöver inte föreligga vid en förändring i någon av dessa faktorer (<Ariane>, <Patriot>). Att dra slutsatser från statistik över påträffade fel i ett visst sammanhang och tillämpa på ett annat är ytterst vanskligt, även om förändringarna kan verka betydelselösa (se 4.3.2.2.6. Felanalys).

Ansvarsfrågan vid brister i en kommersiell komponent kan också vara ett problem, t ex där producenten friskrivit sig från ansvar vid återanvändning i icke avsedd miljö[329]. Ansvarsförhållandena vid producentuppköp och konkurs kan också visa sig oklara. Andra allmänna problem är bristande insyn och kontroll över produktens framtagning, underhåll och egenskaper. Det senare gäller inte enbart produktens funktionalitet och prestanda utan också dess krav på processorprestanda och lagringskapacitet. I den mån dokumentation över krav, konstruktion och verifieringar finns, kan åtkomsten av dessa samt av källkoden vara begränsad. Detta försvårar möjligheterna att kunna spåra systemkrav till motsvarande COTS-kod och test, att avgöra vilken täckningsgrad som uppnåtts, att följa en felkedja, att försäkra sig om determinism i återanvänd del och de komponenter denna i sin tur bygger på etc. Saknas kravspecifikation helt ökar kostnaderna för bedömning av i vilken grad systemets krav på produkten är tillgodosedda och om okänd funktionalitet, bi- eller sidoeffekter kan

329. Exempel ur licensavtal: Programvaruprodukten kan innehålla stöd för program skrivna i Java. Javatekniken är ej feltolerant och är ej utformad, tillverkad eller avsedd för användning eller vidareförsäljning som kontrollutrustning i farlig miljö med krav på felfri funktion, t ex kärnkraftsanläggningar, navigations- eller kommunikationssystem för flygplan, flygledning, livsuppehållande system eller vapensystem, i vilka fallerande Java-teknik direkt kan orsaka dödsfall, annan personskada eller allvarlig fysisk skada samt miljöskada.

föreligga (<BCG>, <Digest>, <Sinclair>). Ytterligare problem kan vara vaga, obefintliga eller alltför frekventa utgivningsplaner, knytningar till andra produkter (eventuellt från andra leverantörer) och därmed utgivningsplanerna för dessa. Låg beredskap hos producenten att snabbt åtgärda identifierade fel kan skapa låsningar i systemutvecklingen. Frisläppning av felrättad version, som förutsätter byte till ny OS-utgåva, kan äventyra systemets stabilitet och kräva omkompilering av hela systemet. Komponent återanvänd på annan plattform än avsedd kan vara omöjlig, om den inte baserats på ett standardiserat gränssnitt på högre nivå.

En konfigurationshantering som även inkluderar använda COTS-produkter är nödvändig. Förutom att hålla rätt på olika versioner, ändringshistorik[330], skillnader mellan dem, vilka som finns installerade var, behövs kontroll över vilka versioner, som är (in)kompatibla med varandra.

Ett av skälen till ökad återanvändning är möjlighet till besparingar och ett snabbare systembygge. Ovanstående faktorer kan motverka detta syfte. För långlivade system får man räkna med, att kostnaderna för underhåll på sikt kommer att dominera över de för anskaffning och utveckling.

IEEE har inlett en studie för att utreda möjligheten av en ensad, internationell procedur för certifiering och konformitetsbedömning av kritisk programvara. Därmed skulle ett bibliotek av pålitliga och verifierade komponenter kunna byggas upp. Detta förutsätter dock att dessa redan från början konstrueras och dokumenteras för återanvändning, vilket kräver en större insats än specialkonstruktion för en enstaka applikation med välkänd användning och omgivning. Investeringarna lönar sig endast om återanvändningen är av tillräckligt stor volym, dvs även intresserar en civil marknad. För att dra nytta av möjligheten att bygga system på redan färdiga produkter krävs en modifierad livscykelmodell och en systemarkitektur anpassad för återanvändning[331]. Systemutvecklare, som lyckats i sin återanvändning har investerat i en ensad arkitektur för en hel familj av produkter inom samma applikationsdomän.

Bilindustrin (vars produkter i regel används över en kortare tid än vad som hittills gällt för den militära sektorn) har börjat inkorporera mer programvara i sina produkter. Detta har öppnat en enorm marknad för säkerhetskritiska programkomponenter. Industrins ökade intresse inom området återspeglas bl a i satsningar på tidsstyrda feltoleranta arkitekturer och protokoll (TTA, TTP) för distribuerade tidskritiska realtidssystem, ett m a p funktionalitet och prestanda välspecificerat gränssnitt mellan nätverkets noddatorer och kommunikationssystem. Chip-baserade, feltoleranta TTA-komponenter och TTP-kret-

330. Exempel på problem: Icke dokumenterade processorändringar (se fotnotsexempel under 4.5.3.2.).
331. Exempel: Dialogorienterad anskaffning: ett iterativt förfarande för att fastställa de väsentligaste kraven med förändrade relationer till COTS-producenterna. Större tonvikt på produktval, arkitektur och integration. Arkitektur anpassad för systemförändringar och -uppgraderingar, som en ”kontinuerlig” verksamhet i systemets livscykel. Isolering av COTS mot kritiska delar. <ANEP-54>.

sar kan med tanke på de miljontals broms- och styrsystem (*break-by-wire*, *steer-by wire*) som produceras årligen visa sig mycket lönsamma, och kommer att konkurrera med CAN-teknikens händelsebaserade protokoll[332].

Återanvändning har visat sig fruktbar framför allt på högre abstraktionsnivåer som arkitekturer för olika klasser av applikationer, generella principer och angreppssätt, lösningsfilosofier, ramverk, programmeringsidiom (s k *patterns* eller mönster). Dessa tekniker har med framgång tillämpats på realtidsapplikationer, basprogramvara (*middleware*)[333], standardkomponenter och programmeringsstöd[334].

332. <Kopetz>, <PALBUS>, www.ttech.com, www.ttpforum.com, www.kvaser.com, <Pascoe>.
333. Exempel: ACE -ett fritt tillgängligt, objektorienterat ramverk för kommunikationsprogramvara, bas bl a för ett realtidsstöd i CORBA (TAO) och för en adaptiv *web*-server (JAWS), se t ex www.cs.wustl.edu/~schmidt/patterns.html.
334. Exempel: Ett verktyg för verifiering av feltolerans m h a återanvändbara felinjiceringskomponenter (sabotörer) med mutering av ingående komponenters beteende baserad på ett mönster för s k mutanter (jämför 6.1.2.)

6.7 Systemsäkerhetsarkitekturer

Säkerhetsarkitekturen för ett programvarusystem beskriver de funktioner och mekanismer, som har till uppgift att bevaka systemsäkerheten. Att generellt ange hur denna ser ut är ej möjligt, då angreppssätten varierar med applikationsområde. Däremot skulle det vara tänkbart att ur ett antal framgångsrika realiseringar kunna syntetisera fram mönster (*patterns*) för olika tillämpningar[335]. Även säkerhetsarkitekturen blir hierarkisk och löper från högsta systemnivå ned till enskilt system. Inom varje system med dess olika arkitekturbeskrivningar över nätverk, informationsstruktur och applikation osv kan sedan en ortogonal systemsäkerhetsvy läggas.

För ett enskilt system kan en applikationsoberoende säkerhetsarkitektur beskrivas mycket översiktligt ur de generella säkerhetsprinciperna. Principerna för riskreduktion, som syftar till att motverka säkerhetshot genom att (i prioritetsordning):

- Eliminera — Hot från omgivning eller annan del av systemet
- Övervaka och styra — Kritisk del för att hålla kvarstående hot på tolerabel nivå.
- Avvärja och lindra — Skador som kvarstående hot trots allt kan medföra.

kan realiseras i form av delsystem för:

- Styrning[336] — Signal-/reglersystem för aktiv, säker styrning av övervakad del.
- Övervakning
- Skydd[337] — Säkerhetssystem som oberoende av styrfunktionerna i förväg kan upptäcka osäkra tillstånd och svara genom att överföra systemet till säkert tillstånd eller på annat sätt minska olycksrisken[338].

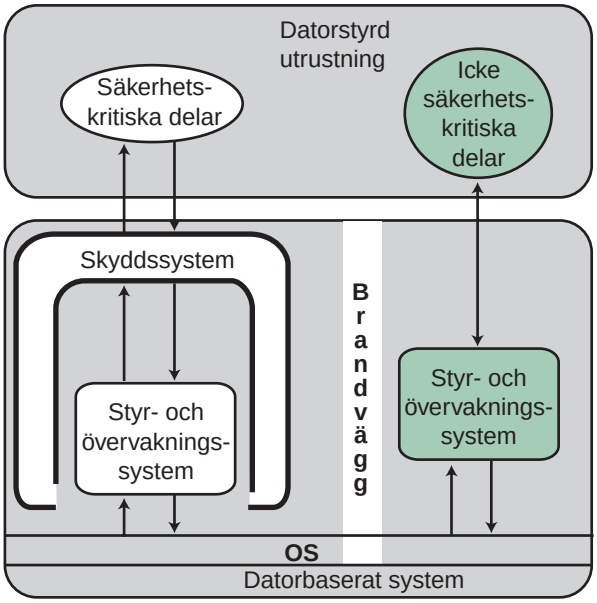
Det är i dessa delsystem som den säkerhetskritiska programvaran återfinns. Kraven för att systemsäkerheten på en övergripande nivå skall kunna upprätthållas är m a o

- Systemsäkerhet — för styrfunktioner till kritiska delar
- Tillförlitlighet — för övervaknings- och skyddsfunktioner

335. Exempel: Det svenska ATC-systemet för tåg, <Lawson_2>.
336. Exempel: Styrsystem för insatsledning, transportsystem, kraftverk, industrirobotar, instrument.
337. Exempel: Motmedel, magnetskydd, aktivt bullerskydd, antikollisionssystem , ABS, krockkuddar, nödstängningssystem, larmsystem.
338. Exempel: **Proaktiva åtgärder** -agera innan osäkert tillstånd övergår till olycka: Motverka riskkälla, hindra farlig kommunikation, bevaka tillgång till kritisk resurs, hantera undantagsfall, överföra systemet till säkert tillstånd eventuellt i degraderad mod.
Reaktiva: Motverka riskkälleexponering , stoppa och starta om systemet, lindra konsekvenser. Jfr arkitektur i <Safety shell>.

- Oberoende för del av högre kritikalitet i förhållande till en del av lägre[339].

Exempel:



339. Dvs en kritisk del isoleras från (eller görs på annat sätt opåverkbar av) övriga delar. Olika typer av isolering finns: fysisk, logisk och temporal. Enklast är att fysiskt isolera kritisk del från del av ingen/lägre kritikalitet. Intresse finns bl a inom flyg för att kunna köra dessa på samma processor (se partitioneringsteknik under 6.1.2.). Detta innebär långtgående krav på logisk isolering av applikationen ända ned till Operativ- och *run-time*system-nivå (4.5.3.1.). Brandväggar med säkerhetsportar är en annan metafor, <Watt>.

6.8 Metoder för systemsäkerhetsanalys

Säkerhetsanalys kan drivas **framåt i tiden** från ett initialt tillstånd eller **bakåt** från ett slutligt (induktiv resp deduktiv sökning). Framåtsökningen leder till en mängd sluttillstånd (både säkerhetsmässigt acceptabla och icke-acceptabla), vilket kan ge en ohanterlig sökmängd, men är intressant då det gäller att utreda effekten av en speciell felyttring. Bakåtsökningen följer kedjan av föregående tillstånd till det utlösande (vilket kan bestå av flera samtidiga händelser) och lämpar sig därför bäst vid haveriundersökningar och där det gäller att finna risker att eliminera eller reducera och hantera.

En analys kan starta från systemnivå och detaljeras ned i underliggande delar (***top-down***) eller utgå från komponentnivå och drivas upp till systemnivå (***bottom-up***). Botten-upp visar en enstaka komponents verkan på systemnivå, men missar samverkande bidrag från flera delar (jfr 6.1.3.). *Top-down* är där att föredra och blir också effektivare i o m att komplexiteten blir lägre och att färre kombinationer behöver beaktas.

Syftet med tidiga säkerhetsanalyser är, att **identifiera ytterligare säkerhetskrav** på ingående delar. Analyserna förfinas allt eftersom konstruktionen tar form och kan i ett senare skede användas, för att **skatta olycksrisken** på närmaste systemnivå och visa, om denna är inom toleransgränsen.

För programvara vars fel är av systematisk typ är det inte meningsfullt, att försöka precisera sannolikheten i siffror[340], annat än för en grov indelning i kritikalitetsklasser. Alternativet är en kvalitativ klassning både m a p rimligheten att en olycka inträffar samt dess konsekvens. Lämpliga åtgärder för kritiska delar över toleransnivån är omkonstruktion, för att främst eliminera (och i andra hand minska) risken för osäkra tillstånd. Detta kan medföra behov av kompletteringar, t ex genom att införa **skyddsfunktioner**, som hindrar spridning av farliga värden, **felåterhämtnings**mekanismer eller **icke programvarubaserade lösningar** (se nedan samt 4.5.2.2.). Målsättningen är att skapa ett system robust mot oförutsedda händelser och förändringar.

De olika metoderna för analys av säkerhet (liksom de för analys av konstruktion, 4.3.2., och av krav), belyser olika aspekter och är därför kompletterande snarare än konkurrerande. Effektivast är därför om tillämpade analysmetoder är integrerade – inte bara med själva programvarutekniken – utan också sinsemellan, så att resultat från en analysmetod kan utnyttjas vid övergång till en annan. Analysverktyg med gemensam databas[341] ger t ex möjlighet till delad information och konsistenskontroll vid uppdateringar. En del verktyg finns redan, som förenar konstruktions- och säkerhetsanalys[342], kombine-

340. Att i stället för ”sannolikhet” tala om ”rimlighet” är **ett** sätt att uttrycka att endast en kvalitativ bedömning av sannolikheten är möjlig. På motsvarande sätt görs i engelskan distinktion mellan *probability* och *likelihood*.
341. <JSSSC> förordar upprättandet av s k *Software Analysis Folders* med samlad information om analysresultat etc.
342. Exempel: *Software Deviation Analysis*, <SDA>.

rar olika analysmetoder[343], integrerar formell teknik med säkerhetsanalys[344] eller tillåter analys av önskat och oönskat beteende hos ett system som en helhet[345].

H SystSäk beskriver ett antal **metoder för säkerhetsanalys** på system- och komponentnivå att användas vid olika **typer av säkerhetsanalyser** (dvs under olika skeden i systemutvecklingen)[346]. I efterföljande avsnitt ges exempel på några analysmetoder användbara på programvarunivå.

6.8.1 Hazards and Operability Analysis – HAZOP/CHAZOP

HAZOP utvecklades på 60-talet inom den kemiska processindustrin (<DS 00-58>, <IEC61882>, <Chudleigh>). Systemets konstruktion och operation granskas av en grupp experter i olika s k **studienoder** m a p vad, som kan avvika eller gå fel[347]. En strukturerad ”*brain-storming*” styrd av frågor baserade på givna **nyckelord** genomförs. Studienoder, nyckelord och **avvikelser** relevanta för programvara och den aktuella applikationen väljs[348].

HAZOPs styrka ligger i att kunna **identifiera** felorsaker och driftsstörningar i gränssnitten (flödets knutpunkter) – en kvalitativ, kreativ teknik med större detaljering än t ex händelseträd och fokus på samspelet mellan komponenter (till skillnad från t ex FMEA, som analyserar individuella komponenter). En preliminär HAZOP utförs på systemnivå, innan programvarukonstruktionen

343. Exempel:(1) *Cause-Consequence Analysis* utför både orsaks- och konsekvensanalys och kombinerar händelseträdets induktiva drag med felträdets deduktiva, <CCA>.
(2) Riskkälleanalys på en hierarki av tillståndsmodeller från systemnivå och nedåt. (3) Analys av säkerhetskrav. (4) Integrering av FFA, FMEA och FTA, <HiP-HOP>.
344. Exempel: Formellt specificerade krav analyserade med PHA, FTA, FMEA, <Scheer> eller med SpecTRM.
345. Exempel: *Dynamic Flowgraph Methodolgy*, <DFM>, för induktiv och deduktiv analys av ett systems logiska och dynamiska beteende, där interaktion mellan människa, maskin- och programvara kan beskrivas i en och samma modell (en riktad digraf med systemets väsentliga komponenter, parameterar, variabler och funktioner). Denna systemmodell, som beskriver både normalt och onormalt beteende, utgör grund för flera olika typer av analyser. Detta kan jämföras med traditionella metoder för säkerhetsanalys, vilka endast beaktar oönskat beteende och är inriktade mot en metod (t ex HAZOP, FMEA eller FTA).
346. Metod för säkerhetsanalys (hur): t ex HAZOP, FTA, FMECA, ETA.
Typ av säkerhetsanalys (när och m a p vad): krav, delsystem, användning, underhåll, miljö: t ex PHA, RHA, SHA, SSHA, O&SHA/ETA. Se även 6.9.3.8.
347. Exempel: Avvikelser: inget flöde, mer, mindre, omvänt.
348. Exempel: **Studienod** för programvara: Helt system, process, kanal/datanät, datalagring, subprogram.

Exempel: Standard**nyckelord** för programvara:
För **studienod** av typ **subprogram** analyseras

Se även <Storey>

Nyckelord	**Avvikelse**
Kommando	Ingen, Fel (oavslutat, avbrutet), Extra
Enkelt datavärde	För högt, För lågt, Noll
Timing	För tidigt, För sent, Fel ordning, Inte alls
Information	Fel, Inkonsistent, Ingen
Lagring	Instabil, Förstörd, Ej åtkomlig, Skräp

fastlagts. Den preliminära riskkällelista, PHL, som erhålls förfinas vid analys av material från programvarukonstruktionen. Tekniken är arbetskrävande och helt manuell. En lämplig strategi kan vara, att överge HAZOP, så fort de komponenter, som är säkerhetskritiska, identifierats och sedan fortsätta med annan teknik (t ex FMEA). **FFA**, funktionell felanalys, bygger på samma idéer. **SDA**, *Software Deviation Analysis*, är en intressant vidareutveckling anpassad till programvara. Konstruktionen analyseras även m a p normal operation. SDA kan automatiseras (se <SDA>).

Hur går man tillväga?

- Klargör förutsättningarna [349].
- Sätt samman en analysgrupp[350].
- Förbered arbetet[351].
- Håll analysmöten[352] : Varje studienod och dess koppling till omvärlden studeras. Standardnyckelord relevanta för viss studienod kompletteras med nyckelord specifika för aktuell applikation.
- Skriv slutrapport[353]. Slutrapporten ger underlag till ytterligare säkerhetskrav.

349. Ex på **frågeställningar**: Vilken är avsikten med analysen? Vad skall inkluderas i systemet? Vilka ekonomiska/ personella/ materiella/ miljömässiga skador är möjliga?
350. **Analysgrupp**: 2-7 personer (**applikations**expert, slutanvändare, HAZOP-kunnig ledare). Deltagarna söker problem, men löser dem ej (konstruktören bör ej ingå: hamnar lätt i försvarsposition).
HAZOP-experten leder mötet (tillser, att alla kommer till tals, men agerar ej i problemsökningen).
351. **Studiematerial** samlas in och tillställs deltagarna före mötet. Kvalitén på kravspecar, systemdesign, omvärldsbeskrivningar, användarmanualer och utkast måste vara god, i synnerhet för objekt, som senare väljs till studienoder.
En **arbets- och mötesplan** upprättas (för mindre system kan 1-2 möten räcka, för identifiering av systemets riskkällor, HAZOP och förslag till problemlösningar). En **analysrapport** planeras.
352. **Möten**: Det första startas i ett tidigt skede (under kravspecificering eller konstruktion). Studiematerialet presenteras.
En grov HAZOP genomförs (kommentarinsamling), vilken senare förfinas.
Systemet delas upp i s**tudienoder** på system-, delsystem- och subprogramnivå.
För varje studienod, ställs frågor baserade på relevanta **nyckelord** beträffande möjliga avvikelser.
353. **Slutrapport**: Analysresultaten sammanfattas ur nyckelordstabeller enl ovan till :

Nyckelord	Avvikelser (*)	Konsekvens	Åtgärd
Nyckelord #1	avvikelse #1	...	...
	avvikelse #2	...	...
	avvikelse #3	...	...
Nyckelord #2	....	...	...

(*) görs spårbar ned till koden.
Obs bra analysresultat kräver kunskap i
– systemet för ”Avvikelser”
– applikationen för ”Konsekvens”.

6.8.2 State Machine Hazard Analysis, SMHA

Tillståndsmaskiner använd ofta under konstruktion av programvarusystem för att beskriva tillstånd och övergångar mellan dessa. Modellen kan också användas för riskkälleanalys.

För att undvika ett ohanterligt antal tillstånd[354] beskrivs övergripande tillstånd som parallella maskiner förfinade i en hierarki av tillståndsmaskiner på närmast lägre nivåer. Modellen bör omfatta hela systemet (såväl program- som maskinvara) och analysen inledas på systemnivå med identifiering av osäkra tillstånd för detaljanalys i underliggande modeller (t ex programvarans). Analysen utgår från varje osäkert tillstånd och söker bakåt tillräckligt långt för att finna var modellen kan ändras för att det osäkra tillståndet ej skall kunna nås.

Modellen inkluderar såväl normala som onormala moder (t ex uppstart, olika degraderingsmoder, partiell nedstängning), tillstånd och övergångar samt operatörskommunikation. Ett minimum av krav är, att varje nödvändigt (specificerat) tillstånd skall vara nåbart från det initiala tillståndet samt att osäkra tillstånd ej skall vara nåbara.

Verktygsstöd finns[355] för specificering (t ex av övergångsvillkor), kontroll och exekvering av modell, riskkälleanalys etc.

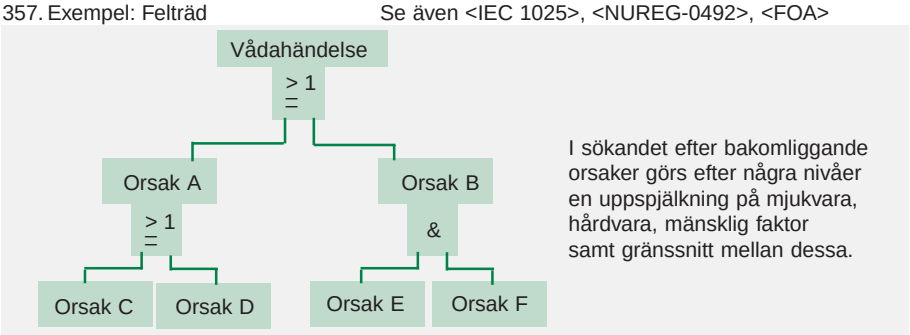
6.8.3 Felträdsanalys – FTA

FTA, framtagen vid Bell Labs[356], utgår från en för systemet oönskad vådahändelse (även kallad topphändelse), vilken undersöks m a p möjliga orsaker – först de omedelbart föregående, därefter de näst föregående osv ned till löven, bas- eller primärhändelserna.

Ett logiskt träd – med start på systemnivå – ritas. Samverkande, oberoende orsaker uttrycks med ”och”-grindar, alternativa med ”eller”-grindar[357]. Orsa-

354. Antikollisionssystemet TCAS II har ca 10^{60} tillstånd.
355. Exempel: SpecTRM-RL
356. Ursprungligen för att kunna analysera en oavsiktlig robotavfyrning.

357. Exempel: Felträd Se även <IEC 1025>, <NUREG-0492>, <FOA>



Om samma händelse dyker upp på mer än ett ställe i samma eller i flera felträd, föreligger beroenden.
”Och” -grindarna förutsätter ej någon särskild orsaksföljd, men det går att ta hänsyn till såväl viss ordning som fördröjningar, exponeringsgrad etc.

ker, som berör programvarudelar, identifieras, och analysen drivs vidare på programvarunivå, där även presumptiva felorsaker i omgivning[358], COTS, underliggande resp extern maskinvara, operatörsinteraktion etc. beaktas. De delar, som i första hand åtgärdas, är ”eller”-grindarna[359], vilka svarar mot adderade sannolikheter för underliggande händelser och därmed ökad risk. Detta gäller i synnerhet vägar genom trädet (från topp- till bashändelse) enbart bestående av ”eller”-noder. I dessa fall utgör bashändelsen ett enkelfel mot en vådahändelse, – en systemlösning, som inte kan tolereras (se 4.5.2.4.2.).

FTAs styrka ligger i ett strukturerat sökande efter orsaker till en viss händelse[360] i syfte att i första hand **eliminera säkerhetshot**, i andra hand att bemästra resterande hot. Analysen erbjuder ett annat betraktelsesätt än det gängse under konstruktion: en koncentration på vad programvaran ej får göra[361], snarare än vad den skall utföra.

FTA:n ger bl a information om (a) **vilka villkor** att kontrollera, (b) **när** under exekvering och (c) **var** i koden. M h a detta kan programvarans **säkerhetskrav detaljeras** och villkor, som verifierar dessa, identifieras. I o m att analysen utförs mot identifierad topphändelse och ej mot kravspecifikation har också **specifikationsfel** kunnat **upptäckas**[362]**. Tekniken** bör därför användas i ett tidigt skede (före konstruktion). Felträden förfinas senare under konstruktionsarbetet med återmatning på kravsättande dokument.

Kvantitativa mått på olyckssannolikhet och exponeringsgrad kan införas i varje nod. Vid analys av programvara görs i stället en rent kvalitativ bedömning av rimlighet för olycka, för att finna var konstruktionsändringar (se 6.8.) är nödvändiga.

Felträd för stora system blir lätt många (ett per topphändelse och systemtillstånd), komplexa och svåröverskådliga – detta gäller i synnerhet interruptdrivna realtidssystem med flera parallella processer. FTA bör därför begränsas

358. Programvarans hantering vid fel från omgivningen kan efter realisering verifieras genom s k felinjicering (se 6.8.7.).
359. En ”eller” -grind bör om möjligt elimineras eller ersättas av ”och” -grind, t ex genom att införa ytterligare utlösande villkor (eller fler steg), före oönskad (del)händelse. Detta svarar mot att ersätta en summa av sannolikheter med en produkt, vilket ger en mindre total sannoklihet:

P(A U B) = P(A) + P(B) - P(A & B)
P(A & B) = P(A) • P(B\A) (givet ober. händelser)

Att införa diversitet (se under 4.5.2.2.) är ett exempel på denna typ av riskreducering.

360. Identifierad t ex under PHA m h a HAZOP.
361. Exempel: Missad funktionalitet (utebliven, felaktig, fel ordning, otid), icke-begärd (oönskad, sidoeffekt).
362. Exempel: Icke angivna förutsättningar för systemet, opreciserade förhållanden/faktorer i omgivningen (s k sekundära felkällor), vilka leder till senare konstruktionsmissar.

till säkerhetskritiska topphändelse och kompletteras med andra analyser, för att utreda tidsaspekternas inverkan på systemsäkerheten[363].

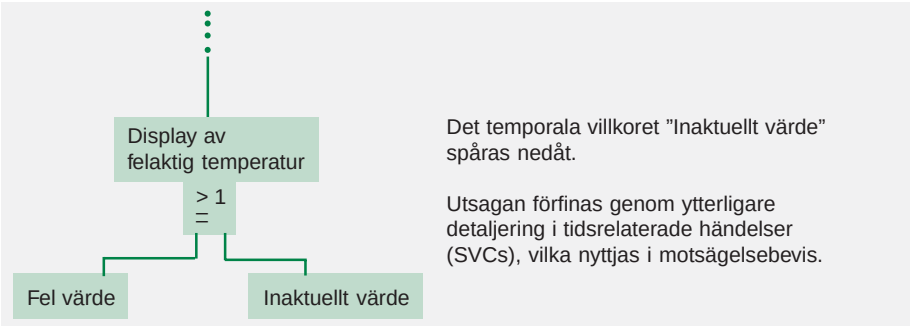
För de noder, där det går att sänka kritikaliteten, genom att i stället ändra konstruktionen, avbryts analysen. Det är inte heller kostnadseffektivt att i trädet införa detaljer ur kodens programvarulogik[364]. Det kan dock visa sig nödvändigt för delar i lågnivåspråk, t ex där kritiskt programflöde styrs av hopp baserat på ett enstaka bittillstånd och där det brister i stöd från kompilatorer och andra statiska analysverktyg.

Resultat från FTA kan användas för att finna testfall till s k negativ testning i syfte att provocera fram oönskad topphändelse. Automatisk testdatagenerering och felinjiceringar kan även kombineras med FTA för att undvika analys av grenar (m h a bevisverktyg), där test snabbare kan finna motexempel[365].

Ett effektivt utnyttjande av FTA på programvara bör inriktas mot att identifiera problem i konstruktion eller specifikation. FTA-verktyg kan användas. De flesta erbjuder trädbeskärning, vilket ger ett minimalt träd byggt på de mest basala villkoren för topphändelsens inträffande (den s k minimala hindermängden, *minimal cut set*). En hel del verktyg finns (t o m för *desktop*miljö) med de vanligaste säkerhetsanalysteknikerna integrerade, så att information i en metod överförs och kan konsistenskontrolleras mot en annan. Med det ökade verktygsstödet kan det därför vara realistiskt att genomföra FTA även på kritiska delar i större programvarusystem. För system, som kan inta olika moder eller konfigurationer, har regeln varit att begränsa analysen till de mest kritiska. Med en modifierad FTA-teknik finns möjlighet att även studera system som genomlöper ett antal mod-övergångar under drift, <Hairston>.

Andra tekniker med samma syfte som FTA är Beroendediagram och Markov analys. Beroende felträdsgrenar kan t ex representeras m h a Markovmodeller

363. Se t ex <formalWARE>, en teknik för härledning av explicita säkerhetsinriktade bevisvillkor (SVC) ur FTA:ns tidsberoende noder och bakomliggande implicita tidsantaganden



364. Detaljanalys av logiken kan i stället plockas fram m h a andra verktyg (statisk analysverktyg) och ur konstruktionsdokumentationens dataflödes- styrfördes-, tillstånds- samt tidsdiagram etc.

365. Ett sådant kan visa, att villkor för topphändelse är uppfyllt eller att exekveringen ej är *exception*-fri, varför målsystemet ej kan strippas från motsvarande *run-time*kontroller, <Tracey>.

och analyseras separat, <Pullen>. Tidsaspekter kan modelleras. Bayesian nätverk (BN) är en mer generell formalism än felträd. Varje felträd kan överföras till ett binärt BN (en riktad, acyklisk graf), vilket möjliggör fler typer av analyser, <Bobbio>.

6.8.4 Failure modes, Effects (and Criticality) Analysis – FME(C)A

FME(C)A, en analysmetod ursprungligen inriktad mot prediktering av tillförlitlighet, syftar till att systematiskt eliminera identifierade felsätt inom en komponent[366]. Tekniken är kvantitativ – varje identifierad felmod tilldelas sannolikhet – och för varianten FMECA – allvarlighetsgrad (jfr 6.1.5.).

FME(C)As styrka ligger i den detaljerade analysen på komponentnivå, vilket kan användas för att identifiera enkelfel eller behov av konstruktionsändringar som införande av redundans och felsäker teknik. Metoden är tillämpbar på programvara (t ex under systemkonstruktion och validering eller vid säkerhetsanalys efter komponentändringar), är bäst för okomplicerade standardkomponenter med ett fåtal, väl kända felsätt, men klarar bara oberoende fel. Den fokuserar tillförlitlighet snarare än säkerhet, är tidskrävande och dyr. Ett bra strategi kan därför vara, att tillämpa FMECA enbart på kritiska komponenter identifierade av en inledande HAZOP[367] och att använda resultaten från FMECA som underlag till FTA.

Hur går man tillväga?

- Utgå från flödesschema, interaktionsdiagram etc. där komponenten är involverad.
- Identifiera felmoder för varje identifierat block [368].
- Fyll i en FMEAtabell: analysera för varje felmod dess orsaker och effekter[369].
- Utför en kritikalitetsanalys[370].

366. C i FME**C**A anger, att kritikalitetsanalysen avslutas med förslag på lämpliga motåtgärder, <IEC 812>.
367. Tillämpa HAZOP på knutpunkter till flera komponenter. Fortsätt med FMECA på en reducerad mängd av kritiska komponenter.
368. Felmod: Utgå t ex från systemnivåns FTA. Spåra systemets riskkällor för alla operativa moder till motsv. komponent och invariabel:
   – Fyll i tabell (*) med huvud: riskkälla, felorsak, kritiska programvaruvärden (variabel, värde).
   – Identifiera felmoder i *input*variablerna (tabell med variabel, variabeltyp, felmod).
   – Identifiera felmoder i programvarulogiken (dvs felaktiga variabelvärden/beräkningar etc).
   – Identifiera feleffekt på andra komponenter och hela systemet från utgångsvärden (tabell med variabel, variabeltyp, felmod).
369. För varje *input*variabel med feleffekt gör en tabell med motsv. *output*variabler och deras felmod, där feleffekten noteras.
370. Kritikalitetsanalys: Mappa FMEA-tabellens effekter på de kritiska värden, som identifierats i tabell enl fotnot ovan, se (*).

- Prioritera identifierat felbeteende[371].
- Kompensera felbeteendet: definiera korrigerande åtgärder [372].
- Upprepa processen vid varje systemförändring.

6.8.5 Händelseträdanalys (ETA)

Analysen utgår från en initial händelse och söker framåt i händelsekedjan för att utreda dess konsekvenser. Den lämpar sig väl för tidsordnade, oberoende händelser, t ex system beskrivna i termer av transaktionsflöden och användningsfall. Som utgångspunkt för analysen väljes en tidigare identifierad riskkälla. I händelsekedjan införs för medverkande komponenter både utfall där dessa fungerar och fallerar. Träden kan bli mycket stora, varför olika typer av trädbeskärningar kan behöva tillgripas. Verktygsstöd finns, som bl a kombinerar händelse- och felträd.

6.8.6 *Common Cause Analysis* (*CCA*)

Analys m a p gemensam felorsak används, där säkerhetskravens uppfyllande förutsätter att oberoende mellan funktioner, händelser eller komponenter kan påvisas, t ex vid förbud mot att ett enstaka fel kan leda till olycka (dvs där gemensam felorsak ej får föreligga) eller där oberoende, redundanta komponenter införts för att minska sannolikheten för vissa kritiska felyttringar.

Ingångsmaterial är detaljerade beskrivningar av krav och arkitektur, underlag till olika säkerhetsanalyser (FMECA, FTA) samt ingående kunskap om den process eller systemomgivning, där oberoende behöver påvisas.

Resultatet är en lista över oberoende resp beroende delar – samt vid felyttringar i de senare – motsvarande effekt på systemnivå. Detta används för att ändra konstruktionen, så att gemensamma felorsaker i första hand elimineras helt eller reduceras till tolerabel kritikalitetsnivå.

För person, som utför *CCA*, kan graden av oberoende i förhållande till utvecklingsgruppen anges (t ex samma grupp, annan grupp, annan organisation). För den som bedömmer analysresultaten gäller att avgöra att fullständighet och korrekthet föreligger samt att vidtagna åtgärder är tillräckliga.

En *CCA* kan belysa olika aspekter. **Zon**analys kan utföras på installerad utrustning, för att utreda interferens med kringliggande system. **Omgivnings**analys används för att finna möjliga gemensamma faktorer (t ex brand, vatten, strålning, elavbrott, nodbortfall). **Process**analys kan tillämpas för att verifiera oberoende under systemkonstruktion (vid t ex val av utrustning och teknik, vid tillverkning eller installation, av procedurer och personal för underhåll samt procedurer för bedömningar).

371. Prioritet uttrycks i sannolikhet + allvarlighetsgrad (Se fotnot under 6.8. om olämpligheten i att kvantifiera sannolikhet för programvarufel).
372. Eliminera felbeteendet eller inför skydd mot det.

En metod av speciellt intresse för programvara är CMA, som bl a kan användas för att verifiera oberoende mellan villkor i felträdens ”och”-grindar. Detta är av vikt t ex mellan övervakande och övervakad enhet samt för delar med identisk programvara.

6.8.7 Felinjiceringar

En dynamisk teknik (besläktad med FMEA och s k *What-if*-analys), där effekten av felaktig kod eller felaktig inmatning analyseras m a p acceptabelt beteende. Resultat från tidigare utförda riskanalyser på systemet kan användas för att finna, vilka utfall, som inte kan tolereras ur säkerhetssynpunkt (se 6.8.3.).

Injiceringar kan göras i interna delar (t ex mellan ingående komponenter, vid inre tillstånd, vid förgreningspunkter i kritisk händelsekedja) samt i externa gränssnitt (t ex mot komponent eller operatör).

- Genom injiceringar i inre delar kan effektiviteten hos implementerad feltolerans studeras. I stället för att införa en felorsak direkt emuleras denna, genom att introducera det feltillstånd, som följer av felet. Experimenten ligger till grund för skattningar av feldetekteringsmekanismen förmåga att upptäcka och korrigera feltillstånd, vilket kan tas som mått på den kvarvarande risken för den implementerade feltoleransen.
- Injicering i delar klassade som icke säkerhetskritiska (eller av lägre kritikalitet) kan användas för att undersöka om störningar i dessa kan spridas till kritiska delar (eller delar av högre kritikalitet).
- Genom att införa de störningar i ett kontinuerligt system, som kan leda till en säkerhetskritisk situation, och studera hur mycket systemet kan störas och hur länge det därefter kan drivas innan oönskad effekt inträffar, kan begränsningar i driftsvillkoren fastläggas innan systemet tas i bruk (t ex skattningar av den maximala drifttiden utan omstart).
- Olika felsätt i gränssnittet mellan en COTS-produkt och övriga systemdelar kan simuleras för att avgöra om en ny COTS-produkt (eller ny version) skall införas i ett system. Exempelvis kan data inmatas i fel ordning, med felaktiga värden, till full buffert, vid felaktig pekarposition, för snabbt eller för långsamt. Felinjiceringar kan även användas för att testa hur pass väl ett COTS-skal skyddar övriga systemdelar mot oönskad interaktion. Lämpliga data för felinjicering kan härledas ur tidigare utförda test- och säkerhetsanalyser på produkten (4.5.1.).

Inte alla genom felinjektion identifierade brister i en implementation behöver vara relevanta. Ytterligare analys kan visa att en del svarar mot omöjliga situationer och kan sållas bort. På snabbtågssystemet BART injicerades 2000 fel, vilka avslöjade 26 felaktiga tillstånd. Knappt hälften av dessa krävde korrigeringar (resterande 14 hanterades av systemet). Metodstöd finns i form av formella språk för specificering av farlig utmatning eller felaktiga tillstånd samt analysverktyg[373].

373. Exempel: Specifikationsspråk: PRED (<VOAS>). Analysverktyg: <MEFISTO>, MAFALDA (på microkärnor, <Fabre>), *Extended Program Analysis* (<VOAS>).

6.9 Publikationer

Detta är ett informativt kapitel, som behandlar standarder, handledningar och andra publikationer med inriktning mot system- och programvarusäkerhet. Ett urval har gjorts av de, som är mest relevanta vid anskaffning av försvarsmateriel innehållande programvara.

6.9.1 Översikt standarder och handledningar

Antalet standarder har ökat dramatisk under de senaste 25 åren. Enbart inom programvarusektorn har mer än 50 standardiseringsorgan utgivit över 250 standarder. En följd av detta är att vissa branschorganisationer bildat egna standardiseringsorgan för att kunna ge vägledning i den växande floran (ett exempel är den europeiska rymdindustrin, ECCS).

IEEE arrangerade 1997 ett symposium om programvarustandarder. Flera problem uppmärksammades därvid: den stora mängden standarder, den bristande överensstämmelsen mellan dessa [374], den tid det tar att få fram ny standard i förhållande till att utveckla en ny programvaruprodukt[375] samt standardernas blandade karaktär av tvingande föreskrifter och rådgivande tips. Det föreslogs, att IEEE skulle verka för en harmonisering mellan existerande standarder samt ökat samarbete mellan organisationerna. En studiegrupp inom IEEE-SESC bildades med uppgift att studera utgivna standarder och kompatibilitetsproblem mellan IEEE- och IEC-standarder (bl a beträffande kritikalitetsnivåer). Diskussioner har även förts inom fackpressen, <Fenton_1>. Vad som kommer ur dessa initiativ återstår att se.

6.9.1.1 Tillämpningsområden

Inriktningen vid anskaffning och utveckling av svenskt försvarsmateriel och -system är, att så långt möjligt utnyttja civila, internationella standarder. De flesta av dessa behandlar dock inte säkerhetsaspekter. De standarder och handledningar som finns att tillgå skiljer sig också beträffande t ex applikationsdomän, adresserad livscykel, kritikalitetsnivåer, dokumentation och krävd funktionalitet.

Några standarder är generella (tillämpbara på alla typer av applikationer), andra är sektorspecifika. De flesta avser nyutveckling, ett fåtal tar upp aspekter på återanvändning (COTS, GOTS). Somliga beaktar hela livscykeln, andra enbart anskaffningsprocess eller programvaruutveckling (i en del fall endast fram till integrationsfasen)[376]. Vissa behandlar stödsystem och verktyg för utveckling och verifiering. En del handledningar avser enbart ett visst språk.

374. Standarderna skiljer t ex betr terminologi, livscykel, mognadsmodeller, kritikalitet.
375. Ny standard tar i genomsnitt fem år. Ny produkt produceras i snitt på 18 månader.
376. Till de fåtal, som behandlar integrationsfasen (vilket inte enbart handlar om integrering) hör IEC 60880 (planering och dokumentation) samt IEC 61508 (enbart integrationstest).

En ökad användning av COTS innebär en starkare betoning av integrationsfasen. Även om vi i en framtid kan tänkas få tillgång till produkter, som är typcertifierade m a p någon säkerhetsstandard, kommer det att krävas omfattande integrationstest, för att verifiera att produkten uppfyller de säkerhetskrav som ställs på applikationen.

En sammanställning av hur dessa egenskaper fördelar sig över olika system- och programvarusäkerhetstandarder ges nedan[377].

Standard eller Handledning	Säkerhetskritisk Applikationsinriktning	Livscykel	Språkrekommendationer	Övrigt
DO-178B	Programvara i flygburna system	Krav → certifiering	Inga	Avser Process och Produkt.
DS 00-55	Programvara i Försvarsutrustning	Anskaffning, Utv, Certifiering	Inga	Avser Process, Roller/Ansvar, (Produkt). MOD.
DS 00-56	Adm. säkerhetsprogram f försvarssystem	Projektstart → Avveckling	Inga	Avser Process, Roller/Ansvar. MOD.
IEC 60880	Programvara i kärnkraftsanläggningars säkerhetssystem.	Krav → vidmakthållande	HOL etc.	Avser Process och Produkt (språkelement) för skyddssystem. Generella krav på språk.
IEC 61508	Allmän. Säkerhetsrelaterade E/E/PES system. Fristående/bas för sektorspecifika std:er.	Krav → Avveckling	Adasubset	Avser Process och Produkt. Del 3 behandlar programvara.
DEF (AUST) 5679	Säkerhetskritiska försvarssystem	Anskaffn → Modifiering	HOL	Avser Process, Roller/Ansvar.
MISRA_G	Programvara i fordon.	Anskaffn → Bedömning	Ada/ Pascal subset	Process, Produkt, kompetens, mänskliga faktorer.
MISRA_C	-”-	-	C subset	-
STANAG 4404	Säkerhetskrav för vapensystem i armé/marin/flyg	Kontrakt → vidmakthållande. Fokus: utveckling	Inga	Preliminär (v.4). NATO. Process, Produkt. Projektanpassas före kontraktsskrivning.

377. Som jämförelse har även några IT-säkerhetsstandarder listats sist i tabellerna i *kursiv stil.*

Standard eller Hand-ledning	Säkerhets-kritisk Applikations-inriktning	Livscykel	Språk-rekom-menda tioner	Övrigt
STANAG 4452	(Säkerhets-bedömning)	Systemkon-struktion → drift, underhåll	Inga	Preliminär (v.1). NATO. Generiska krav.
NASA-STD -871	Programvara i NASAs rymd-program	Programvarans livscykel	Inga	Process.
MIL-STD-882D	Systemsäkerhets-program	Studier → av-veckling	Inga	Process.
H SystSäk	Systemsäk.verk-samh. för FM:s system	Studier → av-veckling	Inga	Process.
H ProgSäk	Allmän, Program-varusystem (nya, COTS)	Studier → av-veckling	Inga (dock språkkrav)	Process, Produkt, Produktionsmiljö, Personalkvalifika-tioner.
IEC 15942	Allmän, Ada-implementeringar	Programvaru-konstr.	Ada	Produkt-egenskaper.
IEC 15408 (Common Criteria)	*Allmän. IT-säkerhet. Assuranskrav.*	*Hotbild, krav → evaluerad produkt. Utveckling.*	*Inga*	*Ersätter ITSEC Manual: CEM (motsvarar ITSEC:s ITSEM)*

6.9.1.2 *Kritikalitet*

Vid genomgång av säkerhetsinriktade standarder har ca 60 olika klassningar av kritikalitet påträffats. Denna har uttryckts i termer av t ex prioritet, angelägenhetsgrad, riskindex eller som *safety integrity*.

De flesta klassar ett systems kritikalitet i ett antal nivåer beroende på konsekvens (där den allvarligaste innebär förlust av liv) och i vissa fall även frekvens/sannolikhet, eller enbart i felfrekvens[378]. Felfrekvensen kan i vissa standarder avse en individuell kopia av systemet under hela dess drifttid, medan andra avser samtliga exemplar i drift[379].

378. Exempel: IEC 61508, vilken därmed uttrycker kritikalitet snarare i termer av tillförlitlighet än säkerhet.
379. IEC 61508 avser individuellt system/funktion (t ex felfrekvens för bilen med registreringsnr *XXXX*).
DS 00-56 avser samtliga förekomster (t ex felfrekvens m a p alla bilar av samma bilmodell och årgång).

Vissa standarder beaktar hur pass omfattande skador en felyttring kan orsaka egendom, omgivning eller person och får på så sätt ett mått på systemets säkerhetskritikalitet[380].

Andra graderar hur pass hotande en felyttring är inför fullbordandet av de uppgifter systemet har att utföra (s k uppgiftskritiskt system). Ytterligare någon anger hur stor insats, som krävs för att utföra en säkerhetskritisk bedömning av programvarusystemet.

Några handledningar beaktar endast tillämpningar av högsta kritikalitet, medan andra avstår från att göra en explicit indelning. En del behandlar prioriteringar m a p process och val av språkkonstruktioner, men ej kritikalitet hos själva produkten.

Att kritikalitetsbedömningen inte bara gäller systemet som sådant, utan även stödsystem och verktyg för utveckling och verifiering samt COTS, poängteras i vissa standarder.

Det kan m a o vara vanskligt, att försöka översätta kritikalitetsmått från en standard till en annan. Ett exempel på jämförelser ges t ex i <IEC 15942>:

IEC 15942	SIL1	SIL2	SIL3	SIL4
DO-178B	D	C	B	A
IEC 61508	Safety IL 1	Safety IL 2	SafetyIL 3	Safety IL 4
ITSEC	*E0 & E1*	*E2 & E3*	*E4*	*E5 & E6*

380. Ett vanligt krav på säkerhetskritiska system är, att antalet felyttringar per drifttimme skall understiga 10^{-9}. Detta svarar mot en felfrekvens/år i intervallet 10^{-5} - 10^{-4} eller ett MTTF på ca 115 000 år. Att statistiskt säkerställa detta krav genom test (t ex baserade på användningsprofiler) skulle kräva fler än $3*10^{9}$ test för en konfidens på 95%.
En felfrekvens/tim < 10^{-4} skulle för samma konfidensnivå kräva fler än 10^{4} test, vilket måste betraktas som en övre gräns för vad som är praktiskt-ekonomiskt genomförbart (se nedan). Feltolerant teknik uppges kunna förbättra felfrekvensen med en faktor 10. Detta är skäl till åsikten att begreppet **säkerhetskrav** borde reserveras för det som är praktiskt testbart och att termen **säkerhetsmål** borde användas för krav, som inte är möjliga att verifiera.

Om	P	⇔ sannolikhet	P(0 fel på N test) = $(1-F)^N$	
och	N	⇔ antal test	$(1-F)^N < 0.05$	för 95% säkerhet
samt	F	⇔ feltäthet:	$N < \log 0.05 / \log (1-F)$	
			$F = 10^{-4} \Rightarrow$ < log0.05/log 0.9999 ⇒ 29978	

Exempel på litet system med höga krav på tillförlitlighet (10^{-10} per användningstillfälle): ABS. Miljontals bilförare använder ett visst ABS i storleksordningen flera hundra tusen ggr dagligen. (Dock: Felyttringar per drifttimme är egentligen ej samma sak som felstatistik baserad på test).

Nedanstående tabell sammanfattar de huvudsakliga skillnaderna mellan några säkerhetsstandarder.

Kritikalitet

Standard / Handledning	Mått	Nivåer	Högsta nivå	Kommentar
DO-178B	Konsekvens	A-E	A:Katastrofal	(FAA:s nivåer: *critical*, *essential, non-ess.*)
DS 00-56, DS 00-55	Konsekvens, Frekvens	S1-S4	S4:Katastrofal, Sällsynt	I vissa fall enbart konsekvensgradering.
IEC 60880	Tillförlitlighet	1-3	1	Ingen kritikalitets-klassning. Avser säker-hetssystem, där tillförlit-lighet är det relevanta begreppet.
IEC 61508	Frekvens	SIL1-SIL4	SIL4: $10^{-5} \leq$SIL4$<10^{-4}$	Målsystemets fel-yttringar/år vid kontinuer-lig styrning eller enstaka användning (skydds-system).
DEF(AUST) 5679	Konfidens, *T* Skydds-åtgärder, *S*	T_0-T_6 S_0-S_6	T_6: Katastrofal, $>10^{-2}$ S_6	T_i = graden av tilltro (LOT) till systemet S_i = integritetsnivå för komponent
MISRA_G	Manövre-ringsbarhet	0-4	4: Okontroller-bar, Extremt osannolik	Integritetsnivåer m a p manövreringsbarhet.
MISRA_C	-"-	0-3		Avser bruket av språket C.
prEN 954-1	Systembete-ende vid fel	B, 1 - 4	B:Skyddsfkns-förlust	Riskkategori = system-uppträdande vid fel.
STANAG 4404, 4452	Bedömnings-insats	Riskin-dex (RI) 1-18	1: Betydande för högriskprogram-vara	RI anger **insats** för en säkerhetskritisk be-dömning av **styr**pro-gramvara snarare än ett mått på risk under systemets exekvering.
IEC 15026	Konfidens: Konsekvens, Frekvens	A - D	A: Hög för Katastrofal & ej trolig: $<10^{-5}$/ år	Integritetsnivån = mått på tilltro över att • skyddsfunktion är tillförlitlig • säkerhetskritisk funktion ej är hotande.
IEEE 1012	Projektvald	–	–	Exempel ges på fyra nivåer, 4: högsta kritika-litet.

Standard / Handledning	Mått	Nivåer	Högsta nivå	Kommentar
NSS 1740.13	Saknas	-	-	Ingen kritikalitets-klassning.
MIL-STD-882D	Konsekvens, Sannolikhet	Projekt-vald	Projektvald	Föreskriver ej någon specifik gradering av allvarlighet och sannolik-het. Exempel ges i bilaga.
H SystSäk	Konsekvens, Frekvens	Projekt-vald	Projektvald	Föreskriver ej någon specifik gradering av konsekvens och inträffandefrekvens. Exempel ges i bild.
H ProgSäk	Enl H SystSäk	Projekt-vald. H,M,L	Projektvald. H:hög.	Givna säkerhetskrav gäller allmänt program-vara av hög, medel, låg kritikalitet och anpassas till projektets valda kritikalitetsnivåer.
IEC 15942	Bedöm-ningsinsats	3 st	Utesluten:Svår-analyserad, – verifierad	Graden analyserbarhet / verifierbarhet m a p säkra konstruktioner är en kostnadsfråga.
ITSEC	*Konfidens*	*E0-E6*	*E6*	*Evalueringsnivå: Grad av tilltro att IT-säkerhets-funktioner uppfyller specade krav. Nivåer ≥E3 kräver granskning av källkod. Funktionali-teten anges i olika funktionalitetsklasser.*
IEC 15408 (CC)	*Konfidens*	*EAL1-EAL7*	*EAL7*	*Evalueringsnivå: Grad av tilltro att IT-säkerhets-funktioner uppfyller specade krav. EAL7 kräver t ex formell test, formellt verifierad konstruktion.*

6.9.2 Standarder

Framtida materielanskaffningar förväntas i högre grad utgå från redan färdig-utvecklade komponenter (eventuellt utvecklade efter andra säkerhetsstandar-der) och genomföras i samarbete med andra nationer (vilka kan ha anammat olika säkerhetsstandarder). Detta kommer att kräva kunskap om vilka övriga standarder, som finns inom systemsäkerhetsområdet.

Korta refererat av ett urval publicerade säkerhetsstandarder ges nedan. De för försvarssystem mest relevanta har placerats först i avsnittet.

6.9.2.1 MIL-STD-882D

MIL-STD-882 beskriver den systemsäkerhetsverksamhet, som skall bedrivas vid anskaffning av system, utrustning och förnödenheter till det amerikanska försvaret. Version 882D från februari 2000 (baserad på anskaffningsförordning <DOD5000.2-R> och amerikansk lag) har väsentligt förändrats i förhållande till föregående utgåva: Riskbegreppet har renodlats[381]. Skattningar av olycka avser ”mest rimligt troliga” konsekvens (i stället för ”värsta, troliga”).

Merparten av listade krav är ej obligatoriska. Anpassning av standarden görs därför genom tillägg till en obligatorisk kärna, som koncentrerats till följande huvudpunkter:

1. Dokumentation av systemsäkerhetsverksamheten.
2. Identifiering av riskkällor.
3. Bedömning av olycksrisker.
4. Identifiering av riskreducerande åtgärder.
5. Reducering av olycksrisk till acceptabel nivå.
6. Verifiering av olycksriskreduceringar.
7. Granskning av riskkällor. Myndighetsgodkännande av kvarvarande olycksrisk.
8. Spårning av riskkällor, avvärjande åtgärder samt kvarvarande olycksrisk.

Detta innebär, att större ansvar läggs på leverantör, när det gäller utformning av systemsäkerhetsverksamheten och belägg för att systemsäkerheten är tillgodosedd. Det finns inte längre några detaljerade kundkrav att hänvisa till, för att få en leverans godkänd [382]. Å andra sidan föreligger ett delat ansvar i o m rapporteringsplikt av kvarvarande risker för formellt godkännande av berörd myndighet.

6.9.2.2 DS 00-55 och 00-56

Engelska MOD utvecklar också försvarsstandarder. Status för dessa rapporteras i <SID News>.

00-56 beskriver säkerhetsprocessen för MODs försvarssystem: säkerhetsansvar, säkerhetskrav, säkerhetsplan, riskkälleidentifiering och -analys[383], risk-

381. 882**D** använder en mer konsekvent beteckning ,*mishap probability/severity,* i stället för 882C:s *hazard probability/severity.*
382. 882**C** ställer t ex krav på upprättande av systemsäkerhetsplaner (SSPP) och -arbetsgrupper (SSWG).
383. Utförd m h a av HAZOP eller annan överenskommen metod.

skattningar, säkerhetsanalys och -bedömningar[384], säkerhetsverifieringar, riskkällelogg[385] och ”*safety case*” [386].

En ny version fastställdes december 1996. I förhållande till 1995 års utkast kan noteras:

1. Oberoende säkerhetsrevisioner (*safety audit*) krävs fortfarande för riskklass A & B.
2. Sannolikhetssiffor bestäms, där praktiskt genomförbart, av det enskilda projektet[387].
3. Allvarlighetskategorier och sannolikheter avser systemets operationella profil[388].
4. *Safety-case*-strukturen[389] är ej fastlagd.

Följande begrepp definieras, vilka även gäller 00-55:

Allvarlighetskategori #6	**Innebörd**
Katastrofal	Flera dödsfall
Kritisk	Ett dödsfall/flera svåra skador
Marginell	En svår skada/flera mindre
Försumbar	Högst en mindre skada

Riskklass	**Innebörd**
A	Ej tolerabel
B	Oönskvärd
C	Tolerabel enligt projektets säkerhetsråd
D	Tolerabel enligt projektets granskare

RISKMATRIS	**Konsekvensens allvarlighetsgrad**#6			
Förekomst#6	Katastrofal	Kritisk	Marginell	Försumbar
Ofta	A	A	A	B
Sannolik	A	A	B	C
Sporadisk	A	B	C	C
Sällsynt	B	C	C	D
Osannolik	C	C	D	D
Ej trolig	C	D	D	D

För varje systemfunktion bestäms den sannolikhetsnivå (SIL), för vilken systemet under givna förutsättningar och i avsedd miljö ej medför att mänskligt

384. För slumpmässiga fel m h a FTA. För systematiska: m h a revisioner av konstruktionsregler och -teknik.
385. En summering av projektets samtliga säkerhetsaktiviteter.
386. Ett strukturerat, argumenterande försvar (baserat bl a på riskkälleloggen) av, att systemet är säkert.
387. Primärt på system-, subsystem- och komponentnivå, men ej för programvarans ingående delar.
388. För systemets **samtliga instanser** – inte enbart ett enstaka system. Jämför fotnot 383 under 6.9.1.2. Kritikalitet.
389. Dvs strukturen vid specificering av säkerhetskrav och sammanställning av säkerhetsargumentationerna.

liv hotas[390]. SIL ärvs av den komponent, som implementerar funktionen. För varje SIL, S1-S4, definieras en motsvarande minsta felfrekvens för en funktion/komponent på denna nivå. Denna minsta felfrekvens baseras på operationella erfarenheter eller väljs enligt nedanst. tabell. Numeriska värden åsätts av projektet på basis av systemets operationella profil.

SIL	Minsta felfrekvens
S4	Sällsynt
S3	Sporadisk
S2	Sannolik
S1	Ofta

00-55 beskriver krav från anskaffning fram till produktion, certifiering och drift m a p säkerhetsrelaterad programvara (SRS) i försvarsutrustning. Distinktion görs mellan SRS på nivå S1-S4 och säkerhetskritisk programvara (SCS på nivå S4).

Några detaljer:
Säkerhetsanalys utförs på såväl SRS som på produktionsprocess[391] och dokumenteras i **Säkerhetsakten** (*Software Safety Case*) – med objektiva belägg för att programvarans formellt representerade säkerhetskrav uppfylls för alla kritikalitetsnivåer. För högre kritikalitet ställs hårdare krav på argumentationen. Bevis samlas i programvarans säkerhetslogg. Argumentationen skall bestå både av formella belägg[392] samt av observationer från olika typer av test. **Roller** och ansvarsfördelning specificeras[393]. Gruppledare namnges i programvarans projektplan. **Formell metodik** pekas ut som den primära tekniken för specifikation och konstruktion. **Formell representation**[394] krävs av alla säkerhetsfunktioner och säkerhetsegenskaper.

SAM (*Safety Argument Manager*) är ett verktyg[395] för uppbyggnad, sökning och konsistenskontroll av säkerhetsargument samt säkerhetsbedömningar enligt 00-56 eller någon annan standard[396]. I denna ingår klassisk säkerhetsanalys (HAZOP, FFA, FMEA, FTA, ETA, RDB), integreringar mellan dessa (<HiP-HOP>), konsistens- och ändringskontroller mellan olika analyser samt stöd för riskkälleloggar.

390. Toleransen kan variera med personkategori, t ex vara olika för tränad operatör, MOD-anställd och allmänhet.
391. Säkerhetsanalys även på utvecklingsprocessen för använda COTS, verktyg samt manuella procedurer.
392. Exempel: Formell specifikation uppfyller säkerhetskraven och satisfieras av objektkoden, analys av både prestanda och resursförbrukning samt av effektiviteten i feldetektion/-tolerans/-återhämtning.
393. Programvaruarkitekt, konstruktionsgrupp, programvaruprojektledare, V&V grupp (oberoende av konstruktionsgrupp), oberoende säkerhetsrevisor (*auditor*), programsäkerhetsingenjör.
394. Klargörande av vad formell representation innebär i jämförelse med formell specifikation saknas.
395. Ett PC baserat verktyg från York Software Engineering, ursprungligen utvecklat vid universitetet i York.
396. En arbetsinsats på 2-3 veckor.

6.9.2.3 DEF (AUST) 5679

En standard för det australiensiska försvaret från 1998 med krav och vägledning för anskaffning av datorbaserade säkerhetskritiska system, där ingående komponenter producerats med metoder (liknande de) för programvaruutveckling (dvs såväl ändringsbar programvara som fasta program, ASIC, PLC, PGA och annan maskinvara).

På ca 70 sidor beskrivs principer och verksamhet för systemsäkerhet samt hur belägg för att systemet är säkert skall åstadkommas (*Safety Case*, jämför 00-55).

Olika roller definieras (kund, leverantör, revisor, granskare, certifierare och slutanvändare).

- **Revisor** (*auditor*) utses av kund för övervakning av säkerhetsverksamheten och standardens efterlevnad, för upprätthållandet av oberoende samt för medling mellan parter.
- **Granskare** (*evaluator*) utses av kund i samråd med leverantör (*developer*) för ingående, oberoende granskning av säkerhetsverksamheten och bedömning av den tekniska relevansen i dokumenterat *Safety Case*.
- **Certifierare** är den organisation, som bedömer systemets säkerhet och utfärdar beslut om användning.

Två begrepp används som mått på kritikalitet.

- **LOT** (*Level of Trust*) uttrycker önskad grad av tilltro[397] över att systemet uppfyller ett visst säkerhetskrav. Sju LOT-nivåer definieras och varje säkerhetskrav[398] åsätts en LOT-nivå. För varje LOT-nivå är konsekvens och sannolikhet[399] definierad.
- **SIL** är ett mått på den försäkran som begärs att en komponent uppfyller sina säkerhetskrav. Sju nivåer definieras och styr den insats[400], som krävs vid utveckling och analys.

Endast i ett begränsat antal fall[401] anses det möjligt att sätta numeriskt värde på sannolikheten för en olycka givet att en riskkälla föreligger. För riskkälla inom ett datorbaserat system (och i synnerhet dess programvara) skall sannolikheten för att riskkällan inträffar överhuvud taget ej kvantifieras.

397. Förenklat uttryckt kan LOT ses som invers till riskbegreppet.
398. För varje identifierad riskkälla specificeras ett matchande säkerhetskrav: att riskkällan ej inträffar.
399. Dock utan att sorten anges. Förmodligen avses per år.
400. Exempel: För de två högsta nivåerna, S_5-S_6, krävs att specification, modell och verifiering är formell.
     För de sex högsta nivåerna krävs högnivåspråk och för de fyra högsta krävs säkert subset av valt högnivåspråk.
401. Exempel: En olyckskedja med slumpmässiga händelser (faktorer i omgivningen, beteende hos maskinvaran).

6.9.2.4 NASA-STD-871

Denna NASA-standard, <NASA-871> ersätter tidigare version, <NASA-1740>, och gäller all inköpt eller utvecklad programvara avsedd att ingå i säkerhetskritiska delar. Metodik och aktiviteter för att uppnå programvarusäkerhet i NASA:s rymdprogram specificeras. Säkerhetsanalys, säkerhetsplanering, kravdetaljering, konstruktion, integration, test, drift, underhåll, fel- samt ändringshantering för programvara beskrivs.

Standarden anpassas till projekt m a p tolererade risknivåer. Vid anpassning anges, vilken programvara som skall anses säkerhetskritisk samt huruvida en säkerhetsrisknivå skall kräva formell säkerhetscertifiering. Ingen vägledning betr kritikalitetsklassning ges.

6.9.2.5 STANAG 4404, STANAG 4452

Dessa standardavtal i förslagsform från NATO avser säkerhetskritiska datorsystem för vapen och ammunition.

4404 innehåller **krav och råd vid konstruktion**, t ex betr produktionsprocess, CPU, säkerhetsanalys, identifiering av säkerhetskritiska funktioner (exempel ges), gränssnitt, konstruktionsprinciper, systemstart, självtest, skydd, underhåll, test. 4404 är avsedd att användas med 4452 och skall anpassas till aktuellt system innan den införs i motsvarande kontraktsdokument.

4452 specificerar ett **ramverk för bedömning av säkerheten**. I systemsäkerhetsprogrammet definieras de säkerhets-/riskkälleanalyser, identifieringsmetoder för säkerhetskritiska delar, teknik för riskkällespårning (mellan krav, konstruktion, implementation, test och dokumentation), som säkerhetskritiska programvarudelar och gränssnittet mot dessa skall genomgå. Vad som är tolerabel risk specificeras under systemkonstruktion eller av nationellt säkerhetsorgan.

Analys- och testaktiviteter under olika utvecklingsfaser samt **riskindex** beskrivs i bilagor. Systemets riskindex definieras som ett mått på **den insats, som krävs för att bedöma säkerheten** i ett datorsystem (i stället för den risk det datorbaserade systemet innebär). Motiveringen för att inte använda sig av konventionell riskbedömning med bestämning av sannolikhet och allvarlighetsgrad är, att detta inte anses praktiskt möjlig för datorbaserade system. Tillförlitlighetsskattningar anses ej tillräckligt tillförlitliga för säkerhetskritiska system. Teknik att mäta tillförlitlighet för en enstaka funktion i förhållande till hela programvarusystemet finns ej heller. Att mäta tillförlitligheten för hela programmet anses drabba programvaran orättvist och leder till en riskbedömning med artificiellt höga riskindex.

Riskindex	Insats för säkerhetsbedömning av programvaran	
1 - 3	Hög risk:	Väsentliga insatser krävs.
4 - 7	Måttlig risk:	Måttliga insatser.
8 - 12	Låg risk:	Låg insats.
13 - 18	Programvaran påverkar ej bedömningsinsatsen.	

6.9.2.6 *DO-178B*

RTCA/DO-178B från 1992 ger vägledning vid utveckling av programvara för luftburna system och utrustningar, vars funktioner skall uppfylla existerande luftvärdighetskrav[402]. Dokumentet täcker programvarans livscykel – ej övriga systemfaser[403]. DO-178B har en europeisk motsvarighet i EUROCAE ED-12B. En gemensam kommitté för arbetsgrupperna EUROCAE WG52 och RTCA SC190 avser att komplettera DO-178B med separata anvisningar[404], vilka kommer att publiceras på nätet. FAA räknar med att behov av en ny version, DO-178C, inte kommer att föreligga förrän kring år 2003.

Systemaspekter behandlade i samband med programvaruutveckling

- Datautbyte mellan systemets och programvarans livscykelprocesser.
- Kategorisering av feltillstånd, definition och bestämning av programvarunivåer (5 st).
- Överväganden beträffande systemarkitektur.
- Synpunkter på olika programvaror, modifierbar av användare, valbara eller COTS.
- Konstruktionssynpunkter på programvara, som inladdas ”fältmässigt”.
- Synpunkter på systemkrav vid verifiering av programvara[405].
- Synpunkter på programvara vid verifiering av systemet.

402. **RTCA**: *Requirements and Technical Concepts for Aviation*, en omarbetning av 1985-års DO-178A (se www.rtca.org).
Exempel **Luftvärdighetskrav**: FAR i USA, JAR i Europa.
Säkerhetskraven för civilt flyg är en faktor 10^2 ggr högre än motsvarande krav för militärt flyg, t ex gäller, att:
-Ingen enstaka händelse -oavsett dess sannolikhet- skall kunna leda till ett katastrofalt felbeteende.
-Ingen enstaka, slumpmässig felkälla skall kunna orsaka ett katastrofalt felbeteende.
En orsak till skillnaden är, att militärt flyg kan lita till en diversitet vid nödläge som inte är möjlig för civilflyg: raketstolen.
Kraven ovan medför bl a att det fordras två piloter i kabinen.
403. Operativa synpunkter på den resulterande programvaran ingår ej (t ex ej certifiering av användarmodifierbara data). Inte heller synpunkter på organisation, kund-leverantörssamband, ansvarsfördelning, kompetens.
404. Inkl även markbaserade system, områdena CNS & ATM och klarlägganden betr utveckling, verifiering. Se t ex <DO-248>.
405. T ex MC/DC test vid högsta kritikalitetsnivå (A).

Programvaruprocesser:

- Planeringsprocess för kooordinering av programvarans utveckling och integrering.
- Utvecklingsprocess inklusive konstruktion och kodning.
- Integreringsprocess (verifiering, konfigurationsstyrning, kvalitetssäkring och certifiering).

Kritikalitetsnivåer:

Felyttrings-konsekvens[406]	Katastrofal *Catastrophic*	Svårartad *Hazardous/ Severe-Major*	Omfattande *Major*	Lindrig *Minor*	Verkningslös *No Effect*
Programvaru-nivå[407]	A	B	C	D	E

DO-178B fastslår, att tillförlitlighetsmått (felfrekvens/-sannolikhet) ej kan tilldelas ovanstående programvarunivåer, vilka därför ej kan användas vid bedömning av systemets säkerhet på samma sätt som maskinvarans felfrekvenser. I övrigt är utgångspunkten, att programvaran åsätts samma kritikalitet som det subsystem, där det ingår, samt att övervakningsfunktion tilldelas samma kritikalitet som övervakad funktion. Olika konstruktionsprinciper kan dock minska allvarlighetsgraden[408].

Tabeller över de krav, som skall uppfyllas under de olika programvaruprocesserna för kritikalitetsnivå A-D och de dokument, där detta kan verifieras, ges i ett appendix.

Mappning mellan DO-178B och t ex DS 00-55 har utförts.

6.9.2.7 *ISO/IEC 15026*

En standard, som beskriver hur **integritetsnivåer** kan bestämmas för programvara utifrån de nivåer, som fastlagts för system och icke-programvarubaserade komponenter. I o m att felyttringar från programvara har sin grund i systematiska fel, är dess integritetsnivå en indikation på den grad av tilltro, som krävs för att en **skyddsfunktion** skall betraktas som tillförlitlig resp en **säkerhetskritisk funktion** ej skall utgöra ett säkerhetshot. Exempel ges på integritetsnivåer (tabell 1) samt strategier för hur förtroende kan vinnas. Fastställandet av integritetsnivåer och integritetskrav överlåts dock åt aktuellt projekt, specifikt applikationsdomän eller enskild nation.

406. Konsekvens är m a p flygplanet samt dess personal och passagerare. Se FAAs AC 25-1309-1A för detaljer.
407. Nivån uttrycker programvarans bidrag till felyttringar i systemet: Programvara på nivå A kan orsaka eller bidraga till en katastrofal felyttring för flygplanet.
408. Detta är i överensstämmelse med t ex ISO/IEC15026.

15026 använder begreppen hot (*threat*) och skadeeffekt (*adverse effect*) i stället för riskkälla (*hazard*) respektive skada (*harm*).

En oberoende instans förutsätts inrättad, som intygar överensstämmelse med upprättade integritetskrav. Första steget är att utföra en riskanalys på systemet (hotidentifiering, frekvens- och konsekvensanalys). En riskmatris upprättas, som tilldelar en riskklass (hög, medium, låg, trivial) för varje kombination frekvens-konsekvens (tabell 1). Varje riskklass svarar mot systemets olika integritetsnivåer (tabell 2). Toleransnivåerna för tabellerna sätts gemensamt av konstruktörs- och certifieringspart. Initialt tilldelas programvaran samma integritetsnivå som systemet. Analys utförs på systemarkitekturen för att utröna, om nivån kan sänkas för programvaran (t ex p g a av att en kombination av villkor från flera subsystem/komponenter krävs, för att hotet skall utlösas). Denna process upprepas tills integritetsnivån för ett subsystem, som enbart innehåller programvara, kan fastställas eller tills nivån för ett subsystem med programvara är acceptabel för varje ingående programvarukomponent. För att uppnå detta resultat kan modifiering av systemarkitekturen bli nödvändig (t ex genom att införa en felhanteringsmekanism, redundans etc).

Tabell 1 – Exempel på riskmatris

Förekomst	Frekvens/år	Konsekvensens allvarlighetsgrad[409]			
		Katastrofal	*Svårartad*	*Allvarlig*	*Lindrig*
Ofta	> 1	Hög	Hög	Hög	Medium
Sannolik	$1\text{-}10^{-1}$	Hög	Hög	Medium	Låg
Sporadisk	10^{-1} - 10^{-2}	Hög	Hög	Låg	Låg
Sällsynt	10^{-2} - 10^{-4}	Hög	Hög	Låg	Låg
Osannolik	10^{-4} - 10^{-6}	Hög	Medium	Låg	Trivial
Ej trolig	< 10^{-6}	Medium	Medium	Trivial	Trivial

Kritik har riktats mot standarden, bl a därför att exemplet ovan kopplar integritetsnivåer till fixa värden. Detta anses inte meningsfullt annat än på projektnivå, eftersom bedömning av vad som kan betraktas som acceptabla felfrekvenser varierar med applikationsdomän.

Tabell 2 – Associering riskklass-systemintegritet

Riskklass	Systemintegritetsnivå
Hög	A
Medium	B
Låg	C
Trivial	D

409. Gradering: *Catastrophic, Major, Severe, Minor.* En jämförelse med DO-178Bs klasser blir lätt förvirrande.

6.9.2.8 IEEE 1012

En programvarustandard som definierar ramarna för processerna verifiering och validering (V&V)[410] från anskaffning till avveckling. Processernas aktiviteter, uppgifter samt in- och utflöden anges. För varje aktivitet och kritikalitetsnivå definieras den minimala uppsättningen V&V-uppgifter, som skall utföras på programvaran, bl a listas säkerhetsanalyser m a p kritikalitet, riskkällor samt risk. Kritikalitetsnivån beror av den specifika applikationen, avsedd användning och de risker, som kan tolereras vid dess användning. Standarden ger exempel på en kritikalitetsklassning i fyra nivåer (SIL 1 – SIL 4: *Low, Moderate, Major, High*), men är oberoende av vilken typ av kritikalitetsklassning ett projekt väljer. Anpassningar av standarden skall specificeras i projektets V&V-plan (SVVP).

6.9.2.9 IEEE 1228

Denna standard definierar de minimikrav, som skall gälla för programvarans säkerhetsplaner från anskaffning till avveckling. Programvaruplanen skall framställas inom ramen för systemsäkerhetsverksamheten och avse programvaran som del av systemet i sin omgivning och med sina operatörer. De enda anpassningar av standarden som accepteras är tillägg av ytterligare, mer stringenta krav. Varje säkerhetsplan skall ha det innehåll, som listats i standarden. Bl a skall beskrivas organisation, ansvarig för programvarusäkerheten, personalkompetens samt dokumentation över projektledning, konfigurationshantering, kvalitetssäkring, säkerhetskrav, säkerhetskonstruktion, utvecklingsmetodik och metriker, test, V&V, analyser m a p krav, konstruktion, implementation, test och ändringar.

6.9.2.10 IEC 61508

Denna omfångsrika, generiska standard (350 sidor)[411] över funktionell säkerhet för elektriska/elektroniska och programmerbara elektroniksystem, (E/E/PES), är resultatet av drygt 15 års arbete. Funktionell säkerhet (men ej övergripande systemsäkerhet) m a p process och produkt samt kompetens (del av den mänskliga faktorn) behandlas.

Följande delar ingår:

1. *General Requirements*
2. *Requirements for E/E/PES*
3. *Software Requirements.*
4. *Definitions and abbreviations.*

410. **Validering enl 1012**: Bekräftelse genom undersökning och objektiva belägg att kraven specificerade för den avsedda användningen är uppfyllda.
411. Generisk – dvs ej inriktad mot ett speciellt applikationsområde.

5. *Examples of methods for determination of SIL*.
6. *Guidelines on 2 & 3*.
7. *Overview of techniques and measures.*

Initialt var standarden inriktad mot industriell styrning efter begäran (s k *demand mode*)[412] av ett system med tre separata delar: maskinutrustning, styr- resp skyddssystem (en klassisk arkitektur för processindustrin). Sannolikheten för felfunktion i maskinutrustningen kunde därmed fördelas ut på styr- och skyddssystemen. Denna modell passar bra för t ex kärnkraftsanläggningar, vars skyddssystem måste uppfylla hårda krav på tillförlitlighet (när de vid behov kopplas in). Med tiden har standarden utvidgats för att även kunna tillämpas på realtidssystem i kontinuerlig drift, där säkerhetsfunktioner även kan ingå i styrsystemet. Systemsäkerhet- och riskbegrepp har också tillförts. Standardens fokus ligger dock fortfarande på tillförlitlighet hos separata skyddssystem (se t ex SIL nedan), varför den kan vara svår att tillämpa för andra arkitekturer. Hur system, som kombinerar kontinuerlig drift med avstängbara system skall hanteras, anges t ex ej.

Tanken har varit, att 61508 skall kunna användas både fristående och som bas för kommande sektorspecifika standarder och handledningar. De första exemplen på detta är järnvägsindustrins EN 50128 och processindustrins IEC 61511 (vilken i USA förväntas ersätta ISA S84). Diskussioner pågår även betr en sektoranpassad variant för medicinsk utrustning. Inom flyg med en redan etablerad sektorstandard kan en möjlig utveckling i stället bli, att nästa version av DO-178B utses till flygindustrins variant av 61508. Vid IEEEs workshop i juni 1997 om säkerhetsstandarder för programvara föreslogs, att 61508 skall utgöra bas för framtida IEEE-standarder i området.

Ramverket för all säkerhetsverksamhet från systemets planering till dess avveckling är säkerhetscykeln (***Safety Life Cycle***), där säkerhetsanalys ligger till grund för specificering av säkerhetskraven och säkerhetsvalidering utförs före driftöverlämnandet.

Säkerhetscykelns huvudprinciper kan sammanfattas i följande steg:

1. Identifiera systemets riskkällor, dess orsaker och konsekvenser (t ex m h a HAZOP).
2. Bestäm kategori för varje riskkällas konsekvens (från Katastrofal till Försumbar).
3. Definiera systemets sannolikhetskategorier (t ex Frekvent motsvarande > 10^{-3}).
4. Definiera riskklasser (Klass I-IV svarande mot Oacceptabel – Försumbar).
5. Upprätta en riskmatris (Riskklass för given konsekvens och sannolikhet).

412. Exempel: Avstängningssystem – ett typiskt exempel från den kemiska processindustrin på skyddssystem i *demand mode*.

6. Placera varje riskkälla i riskmatrisen (dvs bedöm dess riskklass).
7. Specificera säkerhetskraven (krav som reducerar de icke-tolerabla riskerna).
8. Realisera dessa i en säker konstruktion.
9. Härled SIL-värden för ingående funktioner och hela systemet.

Säkerhetskraven på systemets säkerhetsfunktioner specificeras i termer av **SIL 1-4**, en diskret skala, som anger inom vilket intervall felfrekvensen skall ligga, för att en säkerhetsfunktion skall kunna fullfölja sina skyddsuppgifter och det **individuella** systemet skall kunna betraktas som säkert [413].

SILi svarar mot intervallet $10^{-i-1} < SIL\ i < 10^{-i}$ och representerar för styrsystem i kontinuerlig drift felfrekvens **per år**[414]. För skyddssystem avser felfrekvensen **per användningstillfälle** (*demand mode*). Bestämning av systemets SIL och fördelning på ingående komponenter görs *bottom-up*: Högsta SIL för en funktion bestämmer systemets SIL. Tilldelat SIL-värde för enskilt system eller funktion är inte ett mått på dess risk (riskbegreppet inbegriper sannolikhet och konsekvens), utan grundas indirekt på den risk som associeras med denna. Standarden anger ej hur man bedömmer att realiserade funktioner uppnått de SIL-värden som krävs (och därmed den felfrekvens motsvarande SIL-nivå representerar), <Fenton>.

ALARP-modellen ges som exempel på hur riskreducering kan tillämpas. Oacceptabla risker (riskklass I) reduceras minst ned till vad som fastlagts som tolerabel nivå för systemet (riskklass II-III). Hur långt ned i underliggande nivåer riskreduceringen skall drivas bestäms av vad som bedömts rimligt genomförbart och avbryts då kostnaderna för dessa överstiger de för den resterande risken eller då risken kan betraktas som försumbar (riskklass IV)[415]. Proceduren upprepas för varje identifierad riskkälla. Eftersom 61508 bygger på SIL-begreppet avser reduceringen enbart felfrekvens.

413. SIL fastställs vid kravställning för enskild funktion (jfr ALARP: bestäms under konstruktion för hela systemet och kan överrida SIL).
SIL enl 61508: Diskret nivå för specificering av krav på säkerhetsintegritet (SI) för varje säkerhetsfunktion, som tilldelats det säkerhetsrelaterade systemet (fyra SI-nivåper, där SIL 4 är det högsta).
SI enl 61508: Sannolikheten för att ett säkerhetsrelaterat system tillfredsställande utför krävda säkerhetsfunktioner inom specificerad tidsrymd och under samtliga givna villkor (dvs ett tillförlitlighetskrav avseende förmågan att kunna bidra till riskreduceringen).
414. Exempel: Årlig felfrekvens för SIL 4: $10^{-5} < SIL\ 4 < 10^{-4}$ svarar mot en felfrekvens/tim: $10^{-9} < SIL\ 4 < 10^{-8}$.
415. Bättre och billigare teknik kan m a o innebära att ALARP-nivån sänks.
61508 skiljer mellan vad som kan vara **acceptabelt** rent allmänt och vad som i det enskilda fallet kan vara **tolerabelt** givet faktiska omständigheter (säkerhetsmässiga fördelar kontra kostnader för införandet av ytterligare säkerhetshöjande åtgärder).
Tolerabel risk fastläggs för varje vådahändelse. ALARP avser den totala risken från alla källor.

6.9.2.11 MISRA, MIRA

MISRA_G

Ett konsortium från bilindustri och universitet i England tog 1994 fram en handledning, *Development Guidelines for Vehicle Based Software*, <MISRA_G> med följande delrapporter:

1. *Diagnostics and Integrated Vehicle Systems*
2. *Integrity*
3. *Noise, EMC and Real-Time*
4. *Software in Control Systems*
5. *Software Metrics*
6. *Verification and Validation*
7. *Subcontracting of Automtive Software*
8. *Human Factors in Information*
9. *Sources of Information*

Rapporterna täcker in Produkt, Process, Personalkvalifikationer samt mänskliga faktorer. Rekommenderade språk för säkerhetskritiska applikationer är subset av Pascal, Ada eller Modula 2 (rapport 1), typade och strukturerade språk som Ada och Pascal (rapport 2). Starkt avstånd tas från C/C++, vilka pga ospecificerade, implementationsberoende element eller komplexa kompilatorlösningar anses svagare än assembler och kräver omfattande extrakontroller. Kritikalitet uttrycks i termer av kontrollerbarhet, dvs sannolikheten att en förare skall kunna klara en felyttring, utan att en olycka inträffar. Bra vägledning ges hur kontrollerbarhet skall bestämmas. Andra standarder för bilindustrin finns eller planeras, (<JA1003>,<JA1004>).

MISRA_C

En handledning över otillåtna språkelement i C samt restriktioner över hur resterande tillåtna element får användas i bilindustrins säkerhetsrelaterade system har sammanställts av MIRA, den engelska bilindustrins forskningsorgan. Detta har gett en lista på 120 regler, vilka gäller för MISRA_G:s säkerhetsnivåer 0-3. För högsta kritikalitetsnivå (4) har fastslagits att C ej skall användas.

Förekomsten av en regelsamling i C för säkerhetskritiska realtidssystem har skapat oro inom kretsar med lång erfarenhet av språk och systemsäkerhet (se 4.5.2.6.).

6.9.2.12 IEC 60880

En standard från 1986 för programvara i säkerhetssystem för kärnkraftverk. Fokus är på hög tillförlitlighet, vilket är det krav som skall ställas på skyddssystem[416].

416. Jfr 6.1.3., 6.6.2. samt föregående avsnitt om ISO/IEC 15026 resp 61508 ang tillförlitlighet/ korrekt funktionalitet för skyddssystem

Önskvärda språkegenskaper behandlas, men inga specifika rekommendationer betr språkval ges:

- Översättare, länkare, laddare, stödprogram skall vara vältestade
- Språk skall vara fullständigt och otvetydigt definierade.
- Högnivåspråk, företrädesvis problemorienterat väljes.
- Språket skall understödja felbegränsande konstruktioner, typkontroll under kompilering samt *run-time*kontroll av typ, parametrar och matrisgränser.

6.9.2.13 *ISO/IEC 1226*

En standard för klassificering inom kärnkraftsindustrin av felkonsekvenser. Tre kategorier definieras (I, II, III).

6.9.2.14 *UL 1998*

Underwriters Laboratories har utvecklat en standard för mikrodatorbaserad styrning, <UL1998>. En europeisk motsvarighet till denna standard finns f n inte.

6.9.3 Handledningar

Detta avsnitt avser handledningar inom system- och programsäkerhet.

6.9.3.1 *ISO/IEC 15942 (Ada in high integrity systems)*

En handbok för utveckling av säkra system i Ada95. Både systemsäkerhet (*safety*) och IT-säkerhet (*security*) behandlas. Man har medvetet avstått från att mappa språkelement på någon av de kritikalitetsklasser, som de säkerhetsinriktade standarderna definierar (se 6.9.1.2.). I stället har en klassning baserad på de insatser och den teknik, som krävs för att analysera och verifiera ett språkelement m a p säkerhet, införts:

Inkluderad	Analys av detta språkelement utgör inga problem: motsvarande verifieringsteknik är både lätthanterlig och väl känd.
Tillåten	Analys av detta språkelement bjuder på kända, väl utredda problem, vilka kan kräva extra verifieringssteg och därmed kostnader (problemen listas i handboken och åtgärder föreslås).
Utesluten	Detta språkelement är i huvudsak oförenligt med vald analys- och verifieringsteknik. Den enda effektiva lösningen är, att avstå från det.

De tekniker för analys- och verifiering, som behandlas är

- Korrekt funktionalitet (m h a formell kodverifiering, symbolisk exekvering)

- Flödesanalyser (m a p styr-, data-, informationsflöde),
- Resursanalyser (m a p stack, övriga minnen, tidsåtgång),
- Intervallkontroll (m a p variablers definitionsområde),
- Strukturella test (minimikrav på täckning: alla grenar, för högsta kritikalitet: MC/DC),
- Kravtest (ekvivalenstest, randvärdestest),
- Granskningar och spårbarhetsanalyser.

Unchecked_Conversion är exempel på ett användbart språkelement, som är svårt att verifiera. Handboken har klassat denna som tillåten. Det extra verifieringssteg, som krävs i detta fall är kontroll av motsvarande kompilatorgenererade objektkod. Ur denna klassning framgår att ett s k *alias*-objekt[417] ej kan tillåtas, om endast formella verifieringsmetoder är tillåtna.

Handboken föreslår ej definition av ett standardiserat Ada-*subset*, à la SPARK, p g a att,

- ett *subset* är optimalt endast för på sin höjd **en** kritikalitetsnivå,
- användarkontroll över tillåtna språkelement ges redan via Ada 95:s Annex H,
- ett optimalt *subset* kommer att behöva utvidgas vid förbättrad verifieringsteknik,
- några av Adas lågnivåelement behövs i många tillämpningar på ett fåtal ställen, trots att motsvarande verifiering är besvärlig.

6.9.3.2 *Safety Critical Software Handbook*

En koncis handbok om programvarusäkerhet, vilken bl a avhandlar begrepp, standarder, organisationer, språk (Ada), (typ)certifierade verktyg (C-SMART, T-SMART, AdaCover, SPARK), certifieringskrav och testteknik . Den baseras på företaget Aonix:s erfarenheter från system- och verktygskonstruktion samt certifiering till bl a Boeing 777, 737 samt Lockheed C130J.

6.9.3.3 *Software System Safety Handbook*

En tämligen omfattande handbok (ca 250 sid) för anskaffning och utveckling av säkra programvarusystem, <JSSSC>, har utvecklats gemensamt av bl a *US Navy, Army, Air Force, Coast Guard Safety Centers* främst m h a FAA, NASA, försvarsindustri och universitet. Den skall ses som ett referensdokument och har bl a baserats på MIL-STD-498, 882B/C/D, STANAG 4404 samt föregånga-

417. Ett Ada-objekt deklarerad som ”*aliased*” kan refereras av en pekartyp. Den tidigare versionen av DS 00-55 krävde formella specificeringsmetoder. Detta krav medför, att språkelement som *alias*-objekt ej kan tillåtas.

ren till <NASA-871>. Huvuddokumentet beskriver ansvar, roller och aktiviteter inom systemsäkerhetsområdet. För programvara konstateras, att det är svårt att kvantifiera dess felförekomster. Dess inverkan på systemsäkerheten beskrivs i stället enbart kvalitativt, t ex efter vilken utsträckning den autonomt styr potentiellt farliga delar (I, IIa-b, IIIa-b, IV enligt MIL-STD-882C) eller vilken allvarlighetsgrad dess felyttringar har för säkerheten (A-E enligt DO-178B). Förutom en arbetsgrupp för systemsäkerhet (SSWG) föreskrivs även en grupp för programvarusäkerhet (SwSWG).

Bilaga A-C ger definitioner[418], referenser samt beskrivningar över säkerhetsdokument, kontrakt, planer, mötesformer och arbetsgrupper. All säkerhetsinformation samlas i en säkerhetsdatabank. Bilaga D tar upp problem med säkerhetsanalys av tidigare utvecklad programvara utan tillräcklig insyn (bl a COTS och NDI). Bilaga E innehåller allmänna säkerhetskrav, rekommendationer och kontrollpunkter, vilka anpassas till aktuellt system (en blankett föreslås för varje krav/kontrollpunkt med motiveringar till varför det ej är tillämpligt eller uppgifter hur det skall verifieras). Bilaga F refererar olyckor, där programvara varit involverad. Bilaga G innehåller processkartor.

6.9.3.4 *System Safety Analysis Handbook*

En omfattande handbok, <SSS>, på ca 650 sidor framtagen inom System Safety Society med bl a översikt av drygt 100 tekniker och metoder för säkerhetsanalys, programvaruverktyg för riskhantering samt systemsäkerhetsverksamhet och -utbildning inom amerikanska myndigheter resp universitet. Den ger också exempel på hur oskarp (*fuzzy*) logik och matematik kan tillämpas för säkerhetsanalys, där osäkerheter föreligger i modell eller data (t ex p g a stokastiska variationer, beroenden eller värden baserade på subjektiva bedömningar), dvs fall, där säkerhetsanalysernas traditionellla sannolikhetsberäkningarna inte är tillämpbara.

Kapitel 6, som behandlar programvarusäkerhet, konstaterar bl a att programvara kan analyseras m a p systemsäkerhet enligt samma principer som gäller för övriga systemdelar. Handboken uppdateras fortlöpande.

6.9.3.5 *BCAG*

Boeing Commercial Aircraft Group har tagit fram standarder och krav för flygprogramvara till bl a Boeing 777, <BCAG_1, BCAG_2>. Dessa kräver Ada 83 eller Ada 95 för programvara på DO 178-B:s nivå A eller B och beskriver de Ada-element, som kan användas. För de delar av *run-time*systemet, som laddas med applikationen, ställs samma krav, som för denna samt belägg för att determinism och certifieringskrav är uppfyllda.

418. Riskbegreppet är annorlunda i o m att dess sannolikhet avser riskkällans inträffande och ej olyckans. (Risk svarar därmed mot vad som vanligvis brukar betecknas riskkällenivå).

6.9.3.6 ARP 4754, ARP 4761

Några rekommendationer, som utgivits av *The Society of Automotive Engineers* är:

ARP 4754: certifieringsaspekter för tätt integrerade eller komplexa flygsystem med syfte att demonstrera överensstämmelse med luftvärdighetskrav, främst FAR/JAR 25.1309.

ARP 4761: en handledning för säkerhetsbedömning av civila flygsystem och utrustningar främst mot FAR/JAR 25.1309 att användas tillsammans med ARP 4754, DO178 m fl. Olika metoder för säkerhetsanalys beskrivs, t ex de tre alternativa teknikerna FTA, Beroendediagram, Markovanalys samt FMEA, FMES och CCA. Under den tidiga funktionella riskkälleanalysen identifieras felsituationer för systemfunktionerna. Felvillkoren klassas efter allvarlighetsgrad (konsekvens) och högsta accepterade felfrekvens per timme (t ex för kritikalitetsnivå A: 10^{-9}). För programvara konstateras, att dess felfrekvenser ej kan kvantifieras och följdaktligen ej heller användas för sannolikhetsberäkningar (t ex i felträd). Enbart kvalitativa bedömningar kan göras baserade på hur utvecklingsprocessen försäkrat sig mot programvarufel. Exempel på olika säkerhetsanalyser ges i appendix A-L.

6.9.3.7 NUREG

Denna handledning i programmeringsspråk för säkerhetssystem i kärnkraftsanläggningar från 1997 studerar språkegenskaper väsentliga för säkerhetskritiska system (predikterbarhet, robusthet, spårbarhet, underhållbarhet osv) för språken Ada83 och 95, C/C++, Pascal, PLC och PL/M.

Anvisningarna grundar sig beträffande Ada 95 bl a på <IEC 15942> och <ORA>. För att undvika eventuella missförstånd beträffande Ada, rekommenderas dock studium av dessa referenser direkt.

6.9.3.8 H SystSäk

H SystSäk definierar grundläggande begrepp och beskriver ett antal säkerhetsinriktade aktiviteter under ett systems livstid fördelade över olika aktörer (**FM, FMV, Leverantör**), bl a

1. Säkerhetskrav specificeras i TTEM (HKV) och i offertförfrågan, RFP (FSC/**FMV** eventuellt med **leverantörsmedverkan**),
2. Systemsäkerhetsplan, SSPP, tas fram (**FSC/FMV eller Leverantör eventuellt under FMV-medverkan**) och godkänns (FSC/FMV). **SSPP reglerar i detalj vem som utför vad av punkter enligt nedan**,
3. Arbetsgrupper för systemsäkerhet, SSWG, bildas för uppföljning av systemsäkerheten (av **FSC/FMV eller Leverantör med medverkan av övriga parter**),

4. Säkerhetsgenomgångar planeras och utförs: SSPR (av **Leverantör under FM, FSC/FMV-medverkan, av FSC/FMV vid dessas interna genomgångar**),
5. Säkerhetskrav för produktutvecklingen, SRP, formuleras och identifieras (**Leverantör**). (Kraven uppdateras efter utförd SRCA och verifieras via SV, se dessa nedan).
6. Preliminär riskkällelista, PHL, tas fram (**Leverantör eller SC/FFMV under Leverantörs medverkan**),
7. Säkerhetskrav och riskkällor analyseras: SRCA, PHA (baserad på PHL), SHA, SSHA, O&SHA/EHA (**SSPP reglerar vad som görs av FSC/FMV resp Leverantör**),
8. FMV Rådgivningsgrupper för Systemsäkerhet resp Säkerhetskritisk Elektronik och programvara (**FMV**) anlitas för råd och stöd i säkerhetsfrågor (**av FMV-projekt**),
9. Ett Felrapporteringssystem för olika typer av fel, olyckor och tillbud (FRACAS) installeras, hanteras och uppföljs (**HKV, FSC, FMV, Leverantör**),
10. Provningsvärdighet, TES, utarbetas (**Leverantör**) under vissa villkor (bl a då SCA, SAR ej föreligger vid prov hos Leverantör/FSC/FMV men ej FM) och granskas (**behörig instans**),
11. Användningsrestriktioner, SRS, tas fram (**Leverantör/FSC/FMV**) och granskas (**FMV**),
12. Säkerhetskravverifiering, SV, utförs efter avslutad utvecklingen inför serieproduktion (**av FMV efter Leverantörs dokumentation av verifieringsmetod och utfall samt anmälan till FMV**).
13. Säkerhetsutlåtande med säkerhetsrapport, SCA/SAR, tas fram (**Leverantör/FSC/FMV**),
14. Risknummerblanketter med samtliga risker identifierade och preliminärt stängda tas fram före SCA/SR (**Leverantör**). Slutlig stängning av risk eller återremittering för riskeliminering/-minimering (**FMV**).
15. Hanterings- och förvaringsbestämmelser, PHST, utarbetas (**FSC/FMV**),
16. Användarmanualer och utbildning, TSR, fastställs (**HKV**),
17. Riskanalys inför systemavveckling, RADS, utarbetas (**Leverantör/FSC/FMV**),
18. Säkerhetsgodkännande, SS, formuleras (**FSC/FMV**) samt
19. Beslut om användning, SR, tas fram (**HKV**).

H SystSäk skall anpassas till aktuellt projekt. Akronymer och engelska termer ovan är hämtade från <MIL-STD-882C>.

Notera att PHA, RHA, SHA, SSHA, O&SHA/ETA betecknar den **typ av analys** som skall utföras, dvs när och m a p vad (krav, (del)system, användning, underhåll, miljö), men anger **ej hur**, dvs vilka metoder som skall användas (t ex HAZOP, FTA, FMECA). Vid en SSHA kan flera metoder vara aktuella. En och samma metod kan användas på olika systemnivåer och systemrepresentationer.

Exempel på punkt 9 är flygvapnets DIDAS, TRAB och DA.

6.9.3.9 *RML*

Ett regelverk för svensk militär luftfart, <RML>, är under utveckling. Dess olika delar kommer att fastställas efter hand (fyra delar föreligger f n).

RML-G innehåller grundläggande föreskrifter och allmänna råd, främst för de som leder och ansvarar för verksamhet vid förband, staber, verk, underhållsinstanser och industriföretag inom det militära luftfartssystemet. Ur denna framgår t ex, att begreppet luftfartsprodukt inbegriper programvaruprodukter, grunddata samt uppdragsdata. Vidare att flygsäkerhetsarbetet skall anpassas till Försvarsmaktens uppgifter i fred, kris och krig.

RML-V -1: Ledning,

-2: Flygdrift,

-3: Flygplatser och baser,

-4: Flygledning, stril och samband,

-5: Utveckling, certifiering och produktion av luftfartsprodukter samt

-6: Flygunderhållstjänst,

är olika detaljbeskrivningar av verksamheten inom ovanstående delområden.

Verksamheten klassas i ett antal nivåer.
De högsta nivåerna är:

0: Försvarsmakt
1: Försvarssystem
2: Flygmaterielsystem
3: Produkt
4: Delsystem

En programvaruprodukt kan komma in på nivå 7.

Nedan ges en stark sammanfattning av regler med viss anknytning till kritisk programvara:

- Försvarssystemunderlaget skall vara spårbart, inkludera dokumentation och visas överensstämma med funktionella och fysiska krav för en viss baskonfiguration (**1.1.12**),
- Specifikationer för materielsystem med tillhörande gränssnitt och samverkansunderlag mot yttre system skall upprättas och omfattas av konfigurationsledning (**1.22.1**).

- Flygmaterielsystemintyg resp militära flygcertifikat svarar mot H SystSäk:s säkerhetsgodkännande (**1.22.4, V-5-vi**),
- Flygsäkerhetskritiska förändringar analyseras och beslut om riskminimerande åtgärder dokumenteras (**1.23.3**),
- Korsreferenser upprättas mellan RML-krav och verksamhetsbeskrivning (**1.31.1**),
- En dokumenterad procedur skall finnas för att omsätta externa krav till interna. Internt krav skall vara spårbart till den process som omsätter kravet till handling (**1.31.3**),
- Huvudprocesser, som erfordras för att leverera tjänster inklusive produkter, skall vara beskrivna (**1.36.1**),
- Kritiska procedurer, som påverkar flygsäkerhetsnivån, skall upprättas, dokumenteras, implementeras, underhållas och kontinuerligt förbättras. Logiken skall bl a ta hänsyn till behovet av kompetens och träning, arbetets komplexitet, arbetsmetod, hjälpmedel (**1.36.2**).
- Dokumenterade procedurer för upphandling av tjänster och produkter skall finnas (**1.39.5**).
- FSI får föreskriva tilläggskrav påkallade för att etablera säkerhetsnivå i överensstämmelse med RML (**5.16.1**),
- Bland sökandes utförda flygutprovningar/flygningar ingår att förvissa sig om en rimlig sannolikhet för att programvaruprodukter är pålitliga och fungerar tillfredsställande (**5.35.2.2**) i sin taktiska miljö (**5.35.8.2, 5.191.2.3**),

Övriga RML-delar, **RML-P** , avser Personal
RML-M, Materielsystem och förnödenheter
RML-F , Mark, anläggningar och lokaler
RML-D , Specifika driftskrav på verksamheter.

6.9.4 Övriga dokument

Detta avsnitt avser standarder och handledningar ej specifikt inriktade mot system- och programvarusäkerhet.

6.9.4.1 H Säk IT

Handbok för Försvarsmaktens Säkerhetstjänst, Informationsteknik ger underlag för beslut om säkerhetsskydd för IT-system inom FM. Boken ger även råd och riktlinjer bl a beträffande tillämpning av FM:s föreskrifter om säkerhetsskydd (FIB 1999:4), samt FM:s interna bestämmelser om IT-säkerhet (FIB 1999:5 samt ändringstryck 2001:1).

6.9.4.2 ITSEC (ITSEM)

ITSEC (*Information Technology Security Evaluation Criteria*) utgör grunden för hur evaluering (IT-säkerhetsgranskning) av IT-system skall utföras. Evalueringen syftar till att skapa en viss grad av tilltro av att säkerhetsfunktionerna i en produkt eller ett system uppfyller de ställda kraven samt fungerar på avsett sätt. Granskningen omfattar bl a säkerhetsmålsättning, arkitektur, design, implementering samt penetrationsförsök. ITSEC har nu en efterföljare i *Common Criteria*, <IEC 15408>.

ITSEC skiljer tydligt mellan krav på assurans resp krav på funktionalitet. Assuranskraven ställs för att antal evalueringsnivåer (E0-E6). Nivåerna kan kortfattat beskrivas enligt följande, där E6 är högsta nivån:

E0 inga krav uppfylls (eller krav enligt högre nivå ej uppfyllda)

E1 kräver viss testning, vissa grundkrav på dokumentation

E2 mer testning, informell säkerhetsmodell

E3 källkod granskas

E4 formell säkerhetsmodell

E5 strikt koppling mellan säkerhetsmodell och källkod (spårbarhet m a o)

E6 matematisk modell för säkerhet, specificering/dokumentation samt verifiering av säkerhetsmodellen

ITSEC ger också exempel på ett antal olika funktionalitetsklasser, vilka alltså bara anger, vilken funktionalitet en produkt eller ett system skall ha. Metodiken i ITSEC beskrivs i manualen ITSEM (*Information Technology Security Evaluation Manual*).

6.9.4.3 ISO/IEC 15408 (Common Criteria, CC)

Common Criteria är en samling kriteria för utvärdering av IT-säkerhet vidareutvecklad från den europeiska ITSEC kompletterad med erfarenheter från bl a USA och Canada. IT-säkerhetskraven är grupperade i en hierarki av klasser, familjer och komponenter, där de senare kan kombineras till paket. CC blev antagen som internationell standard under 1999, <IEC 15408>, och inkluderar en utvärderingsmetodik, CEM (*Common Evaluation Methodology* – en motsvarighet till ITSEC:s ITSEM). CC kommer att ersätta ITSEC.

CC bygger i stort på samma grundprinciper som ITSEC. Den har säkerhetsfunktioner indelade i 11 och assuranskrav i 8 klasser – även här med 7 evalueringsnivåer, EAL1-EAL7, men någon exakt överensstämmelse föreligger ej mellan evalueringsmetoderna:

IEC 15408 (*Common Criteria*)	US TCSEC (*Orange Book*)	ITSEC
–	D: *Minimal Protection*	E0
EAL1: Funktionellt testad	–	-
EAL2: Strukturellt testad	C1: *Discretionary Security Protection*	E1
EAL3: Metodiskt testad o kontrollerad	C2: *Controlled Access Protection*	E2
EAL4: Metodiskt konstruerad o testad	B1: *Labeled Security Protection*	E3
EAL5: Semiformellt konstruerad o testad	B2: *Structured Protection*	E4
EAL6: Semiformellt verifierad konstruktion o testad	B3: *Security Domains*	E5
EAL7: Formellt verifierad konstruktion och testad	A1: *Verified Design*	E6

6.9.4.4 *KRAVDOK*

En handbok för kravspecificering inför anbudsinfordran och till senare kravunderlag i avtal med vald leverantör vid anskaffning av försvarsmateriel. KRAVDOK har bearbetats under åren1982-1996. KRAVDOK avser täcka all typ av programvara, dock saknas referens till hantering av säkerhetskritisk programvara.

FMV:s uppdragsgivare är främst FM, som bl a tar fram kravdokumenten TOEM, TTEM och TEMU. FMV omformulerar de taktiska kraven i TTEM och TEMU till tekniska krav. Är systemupphandlingen av större omfattning, utarbetas normalt en systemspecifikation. Denna baseras på ingående system- och funktionsanalyser och tjänar som underlag för kravspecifikationer på större systemdelar. Vid upphandling av måttligt stora system översätts ofta TTEM direkt till en AnbudsInfordransSpecifikation (AIS).

Anbudsgivarnas svar, Anbudsspecifikationerna, är en tolkning och fördjupning av FMV Teknisk Specifikation (TS) i anbudsförfrågan. FMV TS och anbudsgivarens Anbudsspecifikation används som grunddokument vid förhandlingar med anbudsgivarna.

KRAVDOK består av fem delar:
1. Allmän del [419]
2. Teknisk specifikation (TS)[420]

419. Tre modeller för anskaffning nämns: Vattenfalls-, Steganskaffning (med utveckling, prototypframtagning och serieanskaffning) samt Evolutionär utveckling och anskaffning.
420. En tidigare TjF refereras för krav på främst programvara för ”inneslutna system” av realtidskaraktär.
Övriga programvarukrav att specificera är funktionalitet, övergripande krav betr prestanda för t ex tidskritiska funktioner, mekanismer för prioritering av trafik, belastning/utjämning, etc., samt krav på konstruktion, programspråksval (ref. till DOD-STD-2167A), förordande av standardprodukter samt av återanvänd programvara.
Övriga krav gäller leverans, underhåll och drift, dokumentation samt resurser för programunderhåll.

3. Verksamhetsåtagande (VÅ)[421]
4. Tidplaner (Verksamhetsplaner) (TP)
5. Vidmakthållandestöd (VS)[422]

6.9.4.5 *DIT 01*

Direktiv för Försvarsmaktens IS/IT-verksamhet berör verksamhets- och insatsledningssystem (i tillämpliga delar även stridsledningssystem) och övriga IS/IT-produkter för användning inom FM och avser att inrikta och samordna dess IS/IT-verksamhet samt att organisera och definiera ansvar och roller för denna. En livscykelmodell baserad på ISO/IEC 12207 samt ISO/IEC 15288 och anpassad för FM har definierats. DIT 01 fastställdes 2001-03-15 och ersätter (tillsammans med bl a ITK 01 samt H Säk IT) därmed handbok FM HIT 97:3. Direktivet kommer att omarbetas till två dokument, för att med övriga regelverk för FM:s IS/IT-verksamhet i ett senare skede vidareutvecklas mot tre huvuddokument för ledningssystem.

6.9.5 Lästips

Detta avsnitt ger ytterligare referenser till läsvärda böcker, rapporter, tidsskrifter och hemsidor.

System- och programvarusäkerhet:

→ ***Safeware: System Safety and Computers*, <Leveson>.**
Systemsäkerhet och datorer: Historik, begrepp, analysmetoder, konstruktion och verifiering. Texten försedd med rikliga fallbeskrivningar, vilka även sammanfattats i fyra bilagor. Exempel på refererade olyckor är *Therac-25, Apollo 13, Challenger, Seveso, Bhopal, Three Mile Island* samt Chernobyl. 680 sidor.

→ ***Safety-Critical Computer Systems*, <Storey>.**
Olika systemsäkerhetsaspekter vid konstruktion av datorbaserade system. Begrepp, metoder för säkerhetsanalys, feltolerans, tillförlitlighet, V&V, kvalitet och certifiering ingår. Säkerhetskritisk maskin- och programvara samt PLC:er behandlas. 450 sidor.

→ ***Software Safety and Reliability*, <Hermann>.**
Grundläggande begrepp, analys och verifieringstekniker inom programvarusäkerhet och -tillförlitlighet. Sektorspecifika (transport, flyg, försvar, kärnkraft, biomedicin) och sektorgemensamma tekniker, angreppssätt och standarder.

421. Avser krav på leverantörens verksamhet: Systemarbete, Utvecklingsplan/ -modell, Konfigurationsledning, Programvaruutveckling, -konstruktion, -verifiering, -leverans och -skyddsarbete, -underhåll, -provning, V&V (ref. <MIL-STD-1521>), Produktsäkerhetsverksamhet, Dokumentframtagning, Utbildning, Tillverkning, Installation och driftsättning.
422. Avser det systemstöd FMV planerar att upphandla.

→ ***Software Engineering*, <Sommerville>.**
System och programvaruprocesserna från anskaffning till underhåll och vidareutveckling. Tillförlitlighet, systemsäkerhet. Fri tillgång till undervisningsmaterial i form av läraranvisningar, källmaterial, OH-bilder, Adaintroduktion, källkodslösningar finns på nätet (se hemsida för *Lancaster University*). 740 sidor.

→ ***Safe and Reliable Computer Control Systems, Concepts and Methods*, <Thane>.**
Begrepp och metoder vid säker och tillförlitlig (*reliable*) konstruktion av programvara för säkerhetskritiska styrsystem (39 sidor). Följer nära Levesons bok.

→ **Ariane 5, *Flight 501 Failure*, <Ariane5>, <FMV_2>.**
Undersökningskommissionens 10-sidiga rapport från kraschen den 4 juni 1996. Bakomliggande orsaker rörde bl a:

- **återanvändning under delvis förändrade villkor**: inaktuella ärvda krav (t ex betr återstart), missade nya krav (bärraket med starkare motor och därmed andra trajektorybanor), implicita antaganden i återanvänd kod (marginalerna för *overflow* ansågs betryggande), olämpliga granskningsbeslut (av sju granskade variabler fann man det onödigt att skydda tre mot *overflow*),
- **konstruktionsmissar**: redundant lösning (duplicerad maskinvara med identisk programvara), aktiv deaktiverad kod (intrimningsfunktion tilläts forsatt exekvering efter start och drabbades p g a av annorlunda trajektorybanor av overflow, vilket inte hanterades och som den redundanta programvaran ej heller klarade).
- **ofullständiga verifieringar inför återanvändning** (ej test i nedräknings- eller flygmod, test endast med simulerad attityddator och ofullständig testtäckning, ofullständiga granskningar).

Minst 3 miljarder kronor hade kunnat sparas, om någon av dessa omständigheter ej hade förelegat.

→ **Mars *Pathfinder*, 1997-07-04, <RisksDigest>, <FMV_2>.**
Sporadiska bortfall av insamlade data hos Marslandaren beroende på ett system baserat på en skeduleringsmodell med prioritetsinversion (i sin tur orsak till upprepade låsningar och återstarter.

→ **Patriot-missilen, 1991-02-25, <RisksDigest>, <FMV_2>.**
Systemet för målspårning och ledning av Patriotmissilerna var konstruerad för max 14 h kontinuerlig drift samt för detektering av flyg (dvs Mach 1–2). I Dharan kom systemet att användas utan omstart i 100h och mot SCUD-missiler (Mach 6). Identifiering och bekämpning av en anfallande irakisk SCUD-missil missades därmed. Förlust: 29 soldater.

→ ***The RisksDigest: Forum on Risks to the Public in Computers and Related Systems, ACM,*<RisksDigest>.**
En elektronisk tidskrift om risker i datorbaserade system sammanställd av P.G. Neumann. Ca 20 volymer om 50-100 notiser har publicerats sedan 1986.

→ ***Safety Engineering Bulletine, The Electronic Industries Association, G-48 System Safety Committee (No 6B: System Safety Engineering in Software Development).***

Tillförlitlighetsteknik[423]:

→ ***Handbook for Software Reliability Engineering,* <Lyu>.**
Boken har förordats av framstående tillförlitlighetsexperter (t ex Musa, Schneidewind) och inkluderar en diskett med flera estimeringsprogram (bl a AT&T SRE, SMERFS, CASRE, SoftRel). CASRE (en av de bättre) kombinerar olika predikteringsmodeller, vilket ger användaren möjlighet att välja den för applikationen bäst lämpade.

→ ***Recommended Practice for Software Reliability,* <AIAA>.**
Definierar en process för bestämning av programvarutillförlitlighet för amerikanska rymdtillämpningar.

Adalitteratur:

→ ***Programming in Ada 95,* <Barnes>.**

→ ***Concurrency with Ada,* <Burns>.**

6.9.6 Hemsidor

Formella metoder:

DoD Software Information Clearing House	www.dacs.dtic.mil/
European Workshop on Industrial Computer Systems, TC7: Reliability, Safety and Security.	www.ewics.org
Formal Methods Europe hemsida	www.fmeurope.org
NASA Langley Formal Methods Team	http://shemesh.larc.nasa.gov/fm/
Wetstone Technology: Formal Methods	www.wetstonetech.com
World Wide Web Virtual Library: Formal Methods	www.afm.sbu.ac.uk/fm/

423. En IEEE-standard är under utveckling.

Språk:

Annex H Rapporteur Group	www.npl.co.uk/npl/cise/systems/hrg.htm
ASIS	www.acm.org/sigada/wg/asiswg
ISO/IEC JCT1/SC22/WG9 (ISOs Adastandarder)	http://anubis.dkuug.dk/JTC1/SC22/WG9/
Safer C subsets & tools (*Oakwood Computing*)	www.oakcomp.co.uk/
SESAM, Försvarssektorns Adaintressenters Användargrupp för Software Engineering	http://sesam.tranet.fmv.se/

System- och programvarusäkerhet:

Adelard´s hemsida	www.adelard.co.uk
AFMC:s System Safety Division	www.afmc.wpafb.af.mil/HQ-AFMC/SE/ssd.htm
Agena Ltd Bayesian Networks (N. Fenton)	www.agena.co.uk
Center for Safety Research (KTH)	www.ce.kth.se/aom/ASAK/ASAK.htm
Centre for Software Reliability	www.csr.ncl.ac.uk/links.html
Defense Acquisition Deskbook	www.deskbook.osd.mil
Department of Defense, Safety links	http://rota-www.med.navy.mil/safety/safety_links.htm
DSTO och DEF (Aust) 5679	www.dsto.defence.gov.au/esrl/itd/safety/
European Workshop on Industrial Computer Systems, TC7: Reliability, Safety and Security.	www.ewics.org
High Integrity Systems Assurance	http://hissa.ncsl.nist.gov/pubs/high_int.html
IEC standarder, rapporter, arbetsgrupper	www.iec.ch
IEEE Computer Society Technical Committee on Software Reliability Engineering (Newsletter).	www.tcse.org/tcseform
Incidents with Commercial Aircrafts (P.B.Ladkin)	www.rvs.uni-bielefeld.de/publications/Incidnets/main.html

Institutet för Riskhantering och Säkerhetsanalys	www.irisk.se
ISO, International Organization for Standardization	www.iso.ch
Johnson Chris, *Accident Analysis & Safety Links*	www.dcs.gla.ac.uk/~johnson/teaching/safety/links.html
Lancaster University, Software Engineering	www.comp.lancs.ac.uk/computing/resources/ser/
MIT (Leveson)	http://sunnyday.mit.edu/papers.html
MOD Directorate of Standards, UK	www.dstan.mod.uk
Musa´s hemsida	http://members.aol.com/JohnDMusa/
Naval Surface Warfare Center	www.nswc.navy.mil/safety/
Navy, FAA & AF Lessons Learned Record	www.nawcad.navy.mil/call
Nuclear Regulatory Commission	www.nrc.gov
North Texas System Safety Society	www.flash.net/~rcade
Oxford Univ Computing Lab: *Safety-Critical Systems*	http://archive.comlab.ox.ac.uk/safety.html
Reliability Analysis Center	http://rome.iitri.com/RAC
RISKS-FORUM Digest (P.G.Neumann)	http://catless.ncl.ac.uk/Risks/
RTCA/EUROCAE Sc190/WG52	http://forum.pr.erau.edu/
Safety & Mission Critical Systems (DTI/EPSRC)	www.aber.ac.uk/~dcswww/SCSP/index.htm
Scandinavian Reliability Engineers (SRE)	www.iae.dtu.dk/sre/
SINTEF Industrial Management Safety and Reliability	www.sintef.no/units/indman/sipaa/
Society for Risk Analysis (SRA)	www.sra.org/
Software Assurance Technology Center	http://satc.gsfc.nasa.gov/
Software Engineering Institute	www.sei.cmu.edu/publications/publications.html
Software Engineering Standards Committee	http://computer.org/standards/sesc/
Software in Safety-Critical Systems Club	http://sunnyday.mit.edu/safety-club

Svenskt Nätverk för Systemsäkerhet, SNSS	www.snss.nu
System Safety Society	www.system-safety.org
Svenska Encressklubben (*European Network of Clubs for Reliability and Safety of Software*)	www.sp.se/encress
Univ of Sheffield, Real-Time Safety-Critical Special Interest Group	www.dcs.shef.ac.uk/~colinc/rtsc
Univ of Washington	www.cs.washington.edu/research/projects/safety/www
Univ of York, High Integrity Systems Engineering Group	www.cs.york.ac.uk/hise