



Organisation KC Ledsyst	Titel Programvarusäkerhet –en introduktion			Dokumentnummer KC Ledstöd 14910:38346/02	
Namn Inga-Lill Bratteby-Ribbing	Tel 018-12 02 63	Datum 2006-09-21	Utgåva 1.5	Sid 1(30)	

## Innehåll

<b>1</b>	<b>INLEDNING</b>	<b>2</b>
<b>2</b>	<b>GRUNDLÄGGANDE SÄKERHETSBEGREPP</b>	<b>2</b>
<b>3</b>	<b>TEKNIK FÖR PROGRAMVARUSÄKERHET</b>	<b>3</b>
<b>4</b>	<b>RISKMATRISER</b>	<b>5</b>
<b>5</b>	<b>RISKHANTERING</b>	<b>6</b>
<b>6</b>	<b>SYSTEMSÄKERHETSARKITEKTUR</b>	<b>7</b>
<b>7</b>	<b>EXEMPEL PÅ SKYDDSSYSTEM</b>	<b>8</b>
<b>8</b>	<b>H PROGSÄK 2001</b>	<b>9</b>
<b>9</b>	<b>NÅGRA SKILLNADER PROGRAMVARA – MASKINVARA</b>	<b>11</b>
<b>10</b>	<b>VAD ÄR 'FEL' I PROGRAMVARA?</b>	<b>14</b>
<b>11</b>	<b>HUR SKATTA FELFREKVENSN HOS PROGRAMVARA?</b>	<b>15</b>
<b>12</b>	<b>ÅTERANVÄNDNING I SÄKERHETSKRITISKA DELAR</b>	<b>16</b>
12.1	Fallstudie 1: Ett styr- och övervakningssystem	16
12.2	Fallstudie 2: Ett skyddssystem	17
12.3	Fallstudie 3: COTS-realtidskärna	17
12.4	Fallstudie 4: COTS-OS	18
12.5	Referat	20
12.5.1	Tjernobyl 1986-04-26	20
12.5.2	TCAS i flyg över Hongkong 1999-06-28	22
12.5.3	Tågolyckan vid Ladbroke Grove 1999-10-05 kl 08.11	24
12.5.4	Ariane 5 Flight 501 1996-06-04	25
12.5.5	Patriot-missilen vid Dhahran 1991-02-25.	26
12.5.6	Mars Pathfinder 1997-07-04	27
12.5.7	USS Yorktown 1997-09	29
12.5.8	V-22 OSPREY i North Carolina 2000-12-11	30

## Dokumenthistorik

Version	Datum	Kommentar
1.0	02-09-03	Uppdaterar o ersätter tidigare version: FMV-dokumentet "Inspektion 14910:9284/00" (utgåva 1.5 från 2000-11-17) med titel "Programvarusäkerhet –en introduktion".
1.1	03-09-30	Tillägg i fotnot 30.
1.2	05-09-16	Uppdateringar/tillägg: fotnot 11, avsn 4, 5, figur avsn 6, avsn 9.4.
1.3	05-11-09	Ändrad kapitelordning. Nya kapitel: 4-5 (Riskmatriser, Riskhantering).
1.4	06-09-15	Nytt ex under avsn 3 + 12.5.8 (OSPREY) samt 7 (polisbilen). Ref till Swallow i avsn 4.
1.5	06-09-21	Tillägg under 'Underhållsplanering' i avsn 9 (fotnot 36).

Dokumentet finns även på [www.fmv.se](http://www.fmv.se) under 'Publikationer: Handböcker: H ProgSäk 2001'.



Organisation	Titel			Dokumentnummer	
KC Ledstöd	Programvarusäkerhet – en introduktion			KC Ledstöd 14910:38346/02	
Namn	Tel	Datum	Utgåva	Sid	
Inga-Lill Bratteby-Ribbing	018-12 02 63	2006-09-21	1.5	2(30)	

## 1 Inledning

Automatiserade system för styrning, övervakning och beslutsfattande ger möjlighet att kunna ersätta mekaniska och mänskliga inslag i uppgifter, som kan vara komplexa, precisionskrävande och kritiska m a p tid och effekt. Detta ställer extra krav inte bara på de systemdelar, som har inverkan på systemsäkerheten (*safety*) utan också på de processer och personer, som är involverade vid anskaffning, produktion och användning av dessa.

De extra kraven spänner över ett vitt fält av såväl tekniska som psykologiska aspekter och avser att öka förvisningen om, att systemet i avsedd miljö och användning inte kan orsaka allvarliga skador på person, egendom eller miljö. Denna sammanfattning behandlar några aspekter på programvarans roll i säkerhetskritiska system.

## 2 Grundläggande säkerhetsbegrepp

**Säkerhetskritiska system** är system, som i sin användning kan leda till att person, egendom eller miljö skadas. Systemsäkerhet är m a o en viktig egenskap. Detta gäller i synnerhet system, som agerar i verklig tid och miljö (t ex *realtidssystem*<sup>1</sup>).

Talar man enbart om säkerhet associerar gemene man oftast till **informationssäkerhet**<sup>2</sup>, dvs förmågan att kunna motstå otillbörligt intrång, insyn, förlust eller påverkan. IT-säkerhet är därför väsentlig för **informationssystem** –ofta med kopplingar till en öppen omvärld intresserad av dess innehåll.

Idag finner vi allt fler system som både är realtids- och informationsbaserade och som därmed kan vara kritiska m a p såväl system- som IT-säkerhet. Då det nedan talas om **säkerhet** menas dock systemsäkerhet.

Hot mot IT-säkerhet är lättare att beskriva än hot mot systemsäkerhet, vilka kan ha sin grund i en mängd samverkande, oförutsedda omständigheter. Det är därför svårare att gardera sig mot systemsäkerhetshot. Skydden måste i högre grad baserad på en säkerhetsfilosofi för själva konstruktionsarbetet. Det ökade inslaget av programvara i systemkomponenterna leder också till att systemsäkerheten blir allt mer avhängigt egenskaper hos programvaran.

I det följande kommer vi att koncentrera oss på programvarans roll för systemsäkerheten, dvs **programvarusäkerhet** (*Software Safety*). Programvarusäkerhet är ett indirekt begrepp, som avser såväl olika programvaruegenskaper (se 6) som den teknik och verksamhet som erfordras för att ett säkerhetskritiskt programvarusystem skall kunna tillgodose systemsäkerheten (se 3).

Programvara är här en generisk beteckning, som inte enbart avser kod, utan även övrig programvarudokumentation från olika faser framtagna för programvarusystemets utveckling, drift och underhåll<sup>3</sup>.

<sup>1</sup> System vars korrekthet ej enbart beror av beräkningarnas resultat utan också av dess leveranstidpunkt.

<sup>2</sup> **IT-säkerhet** omfattar bl a begreppen konfidentialitet (sekretess), tillgänglighet samt riktighet (integritet –att endast acceptera behöriga ändringar samt värna om informationskvalitet). Se även H ProgSäk 6.1.2.

<sup>3</sup> Programvara representeras bl a av programvarukrav, arkitektur, gränssytespecifikationer, design, implementation, analysresultat, testspecar/-program/-data/-resultat, ändringsspecifikationer, feldatabaser, manualer för användning/drift/handhavande.



Organisation KC Ledstöd	Titel Programvarusäkerhet – en introduktion			Dokumentnummer KC Ledstöd 14910:38346/02	
Namn Inga-Lill Bratteby-Ribbing	Tel 018-12 02 63	Datum 2006-09-21	Utgåva 1.5	Sid 3(30)	

## 3 Teknik för programvarusäkerhet

Programvarusäkerhet som teknik baseras på tre discipliner:

- Programvaruteknik, dvs -själva ramverket för programutvecklingsarbetet
- Säkerhetsmetodik, att
  - utföra säkerhetsanalyser, dels för att
  - kartlägga möjliga hot<sup>4</sup> mot person, egendom, miljö, dels för att kunna
  - identifiera säkerhetskritiska delar och
  - formulera matchande säkerhetskrav som gardering mot dessa hot samt
  - välja konstruktionslösning<sup>5</sup> som reducerar kritikalitet (riskgraden)
  - välja produktionsprocesser i paritet med kritikalitet<sup>6</sup>.
- Tillförlitlighetsteori
  - vars huvudstrategier är, att:
    - o visa att specificerad funktionalitet är uppfylld
    - o avslöja och eliminera/undvika felkällor samt efterföljande feltillstånd
    - o erbjuda korrekt funktionalitet trots förbisedda/oundvikliga felkällor och feltillstånd (s k feltolerans).

– Programvaruteknik omfattar metoder och verktyg för produktion, underhåll och drift av programvarusystem, vilket ställer grundläggande krav<sup>7</sup> på såväl de personer och processer som är involverade i dessa aktiviteter som systemens programvaruprodukter.

– Säkerhetsmetodiken koncentrerar sig på att finna beteenden och situationer, som måste undvikas för att säkerheten ej skall åsidosättas. Olika tekniker för säkerhetsanalys appliceras tidigt och på högsta systemnivå, för att fortsättas ned på underliggande nivåer och slutligen till enskild programvaruenhet. En ”top-down”-teknik där fokus inte inriktas mot fel i största allmänhet, utan på möjliga säkerhetsbrister i aktuellt system. Definition av en säkerhetsinriktad arkitektur genomförs (se 6), där de systemspecifika säkerhetskraven blir styrande för vilka designrestriktioner som måste åläggas ingående systemdelar, för att identifierade säkerhetsshot ej skall utlösas. Partitionering införs för att skydda delar av högre kritikalitet mot påverkan från delar av lägre eller ingen kritikalitet. Realiseringsenheter byggs baserade på inbyggda säkerhetskontroller (t ex m h a ’Design-by-contract’-teknik)<sup>8</sup> för bevakning under drift (*run-time checks*).

– Tillförlitlighet baseras (när det gäller programvara) på principer för felfri konstruktion<sup>9</sup>, defensiv programmering och feltolerans. Uppmärksamheten inriktas mot att realisera och verifiera specificerad funktionalitet. Arbetet utförs ”bottom-up”, dvs från enskilda programvaruenheter upp till komponent, delsystem och slutligen den högsta systemnivån.

<sup>4</sup> **Säkerhetsshot:** Sammanfattande begrepp för de förhållanden i system och omgivning, vilka tillsammans kan leda till olycka, dvs riskkälla, riskkälleexponering, andra omständigheter (t ex person i riskområde).

<sup>5</sup> Huvudprincipen är att så långt möjligt samt i följande prioritetsordning:

- \* eliminera identifierade säkerhetsshot,
- \* reducera resterande, oundvikliga hot,
- \* hantera eventuella kvarstående hot, för att
- \* hålla kvarvarande risker under fastlagd toleransnivå.

<sup>6</sup> **Exempel:** För del av högsta kritikalitet ställs hårdare krav på testtäckning än delar av lägre eller ingen kritikalitet (testtäckning ned på objektkodsnivå i stället för på källkodsnivå).

<sup>7</sup> Grundkrav, se kap 8.

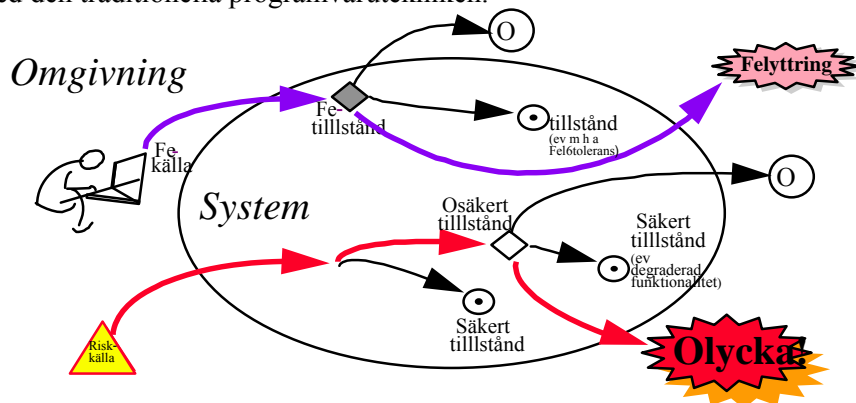
<sup>8</sup> Olika tekniker kan användas för att förhindra att systemet drivs in i osäkra tillstånd: *Pre-/post conditions, invariants, type constraints, constraint requirements*, diversitet (flera olika sätt att utföra viss uppgift) med majoritetsomröstning (för upptäckt o bortsällning av avvikande resultat hos de parallella teknikerna samt ensning av likartade resultat).

<sup>9</sup> Vilket bl a kan understödjas av formella metoder för specificering och verifiering.



Organisation KC Ledsystem	Titel Programvarusäkerhet – en introduktion	Dokumentnummer KC Ledstöd 14910:38346/02			
Namn Inga-Lill Bratteby-Ribbing	Tel 018-12 02 63	Datum 2006-09-21	Utgåva 1.5	Sid 4(30)	

Dessa två synsätt – inriktning på vad system skall resp inte får göra – kompletterar varandra och går att integrera med den traditionella programvarutekniken.



För att tillgodose tillförlitligheten i ett programvarusystem riktas konstruktionsarbetet i första hand in på att så långt möjligt förebygga och eliminera felkällor, i andra hand på att förhindra händelsekedjor från en **felkälla** (*fault*) över till ett **feltillstånd** (*error*) hos systemet och vidare till **felyttring** (*failure*)<sup>10</sup>. I det senare fallet gäller det att identifiera vilka felkällor eller feltillstånd som inte går att undvika vid programvaruimplementeringen samt att från dessa lägga in alternativa exekveringsvägar, vilka inte leder vidare till felyttringar.

På motsvarande sätt går systemsäkerhetsarbetet under konstruktionsfasen ut på att i första hand eliminera riskkällor, i andra hand att omöjliggöra de händelsekedjor som utgående från en **riskkälla** (*hazard*) leder till ett **osäkert tillstånd / risktillstånd** (*unsafe state*) och vidare till en **olycka** (*accident*).

Huruvida ett feltillstånd inom en systemdel leder till en felyttring eller olycka på systemnivå beror på omständigheterna. Samverkande faktorer i komponent, system och systemomgivning – liksom tidsaspekter – spelar in. Ett flygplan, vars höjdbestämning baseras på en eller två olika tekniker hamnar i osäkert tillstånd vid fel i en av dessa. Baserar vi höjden på tre olika tekniker med olika sätt att fallera<sup>11</sup>, klarar systemet fel i en av dessa. Det är därför viktigt att redan under design identifiera möjliga osäkra tillstånd samt att ändra konstruktionen, så att systemet kan övergå till ett säkert tillstånd vid olika typer av fel (eventuellt till priset av degraderad funktionalitet: t ex tillräcklig funktionalitet för säker färd hem). Man behöver dock även analysera säkra men degraderade tillstånd – en analys av komplexa samband mellan komponenter realiserade i vitt skilda tekniker. I det här läget är handlingsfriheten ytterst begränsad. Det gäller att systemet inte kan interagera på ett sådant sätt att detta tillstånd kan lämnas. Olyckligtvis var detta det som hände OSPREY, en flygfarkost, som i låg fart intog helikoptermod och i högre hastigheter vanligt flygplansmod (se 12.5.8)<sup>12</sup>.

<sup>10</sup> Termen *fel* används här som ett generiskt begrepp för *felkälla*, *feltillstånd*, *felyttring*, t ex om det inte är väsentligt att särskilja begreppen eller innan det utretts hur felkedjan ser ut och vad som är själva felkällan.

<sup>11</sup> T ex diversitet i form av en programvarubaserad tröghetsnavigering kompletterad med höjdbestämning via radar resp tryck.

<sup>12</sup> Baserad på a) [http://www.defenselink.mil/news/Apr2001/t040520001\\_t405mv22.html](http://www.defenselink.mil/news/Apr2001/t040520001_t405mv22.html)  
 b) The Risks Digest: 21.33 och 21.38 under <http://catless.ncl.ac.uk/Risks/>  
 c) Report of the Panel to Review the V-22 Program, Apr 2001, <http://www.acq.osd.mil/sts/v22/>  
 d) New Scientist, 5 May 2001, <http://www.newscientist.com>  
 e) GAO-01-369R: Defense Acquisitions: Readiness of the Marine Corps' V-22 Aircraft for Full-Rate Production (sökning efter Osprey o V-22 på <http://www.gao.gov>).



Organisation KC Ledsystem	Titel Programvarusäkerhet – en introduktion	Dokumentnummer KC Ledstöd 14910:38346/02			
Namn Inga-Lill Bratteby-Ribbing	Tel 018-12 02 63	Datum 2006-09-21	Utgåva 1.5	Sid 5(30)	

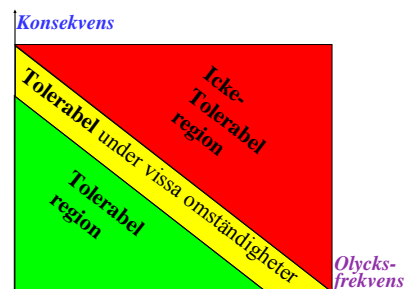
## 4 Riskmatriser

Risk är en term, som används för att beskriva och värdera utfallet vid en vådahändelse / olycksscenario. Traditionellt definieras ett 2-dimensionellt riskbegrepp – sannolikheten för en olycka ( $P$ ) samt dess värsta konsekvens ( $C$ ).

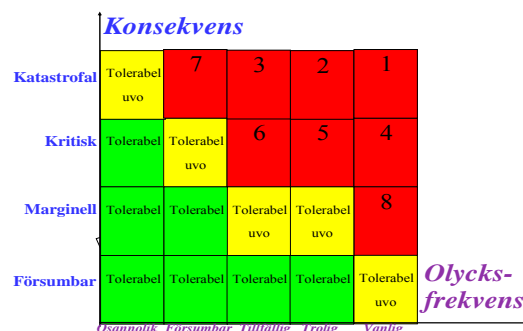
För att ange vilka kombinationer av dessa parametrar, som skall kunna betraktas som tolerabla för ett visst system i dess tänkta användning och omgivning, definieras ett riskdiagram.

Olika zoner får beteckna de kritikalitetsnivåer, som är aktuella för systemet. Ofta används 3 nivåer och färgerna rött för högsta kritikalitet (icke-tolerabla utfall), gult för medelhög kritikalitet (utfall tolerabla under vissa omständigheter)<sup>1</sup> samt grönt för låg kritikalitet (tolerabla risker).

Matematiskt mest intuitivt och praktiskt (se nedan) blir ett diagram med kartesiska koordinater (lägst risk: origo).



Vid diskretisering av riskvariablerna övergår riskdiagrammet i en riskmatris (origo nere till vänster). Riskreduktion inriktas i första hand mot att eliminera / reducera olycksrisker ( $P*C$ ) ovanför toleransnivån. En kostnadseffektiv reduceringsordning i röd zon kan då vara denna:



Kvantifiering av riskparametererna undanröjer problemet att sätta ord på och tolka kvalitativa värden<sup>13</sup>.

En uniform skalning av axlarna blir möjlig. Hela matrisen kan då siffersättas, genom att fastlägga det numeriska värdet ( $P*C$ ) för ett matriselement.

Origoorienteringen gör också, att samma matris kan användas för olika leverantörssystem, vilket underlättar riskbedömning vid integrering till system-av-system.

Ytterligare en fördel är att system av lägre kritikalitet då kan klara sig med en mindre del av matrisen (nederdelen).

Consequence (C)						
100	0.3	1	3	10	30	100
30	0.09	0.3	0.9	3	9	30
10	0.03	0.1	0.3	1	3	10
3	0.009	0.03	0.09	0.3	0.9	3
1	0.003	0.01	0.03	0.1	0.3	1
0.3	0.0009	0.003	0.009	0.03	0.09	0.3
Probability (P)	0.003	0.01	0.03	0.1	0.3	1

Samma skalfaktor på bägge axlarna ger dessutom samma riskvärde över varje matrisdiagonal vinkelrät mot origo<sup>14</sup>.

Med logaritmiskt skalade axlar (steg om 10-potenser) kan samma riskmatris dessutom användas för klassning såväl av mindre allvarliga olyckor som för verkligt globala och universella katastrofer (med jordens eller universums undergång i övre vänstra hörnet)<sup>15</sup>.

<sup>13</sup> Ett vanligt dilemma är att såväl ett enstaka dödsfall som den ultimata naturkatastrofen klassas som 'Katastrofal'.

<sup>14</sup> Skalfaktor 10/3 i riskmatris baserad på 'The RAC Matrix...', Journal of System Safety #2, 2005.

<sup>15</sup> An improved Accident Severity Classification Scheme for United States Department of Defense Systems, D.W. Swallow, 24<sup>th</sup> International System Safety Conference, 2006.

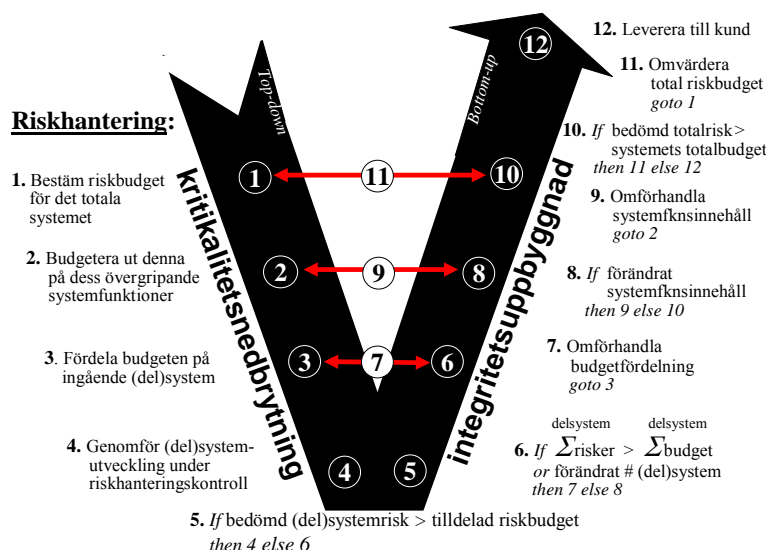


Organisation KC Ledstöd	Titel Programvarusäkerhet – en introduktion	Dokumentnummer KC Ledstöd 14910:38346/02			
Namn Inga-Lill Bratteby-Ribbing	Tel 018-12 02 63	Datum 2006-09-21	Utgåva 1.5	Sid 6(30)	

## 5 Riskhantering

Vid beställning av nytt system fastläggs den totala riskbudgeten på toppnivån, för att – då den inre strukturen börjar ta form – fördelas ut på övergripande systemfunktioner / (del)system samt maskin- och programvarukomponenter. Systemets riskmatris bryts p s s ned till delmatriser för ingående systemdelar. Därmed kommer krav beträffande högsta tolererade riskbidrag även att ställas på motsvarande programvarudelar.

Riskhanteringsprocessen kan – liksom systemutvecklingen – illustreras m h a den traditionella V-modellen<sup>16</sup>: den vänstra skänkeln med top-down-analyser för bestämning av systemdelarnas kritikalitet samt fördelning av riskbudgeten på ingående delar, den högra med en uppbyggnad *bottom-up* från enskilda komponenter (med en integritet i paritet med identifierad kritikalitet) till successivt större systemdelar:



En utvärdering av i vilken mån implementerade delar håller sig inom tilldelad riskbudget utförs för allt större systemdelar. Detta kan leda till kostsamma omtag: Omfördelningar p g a att nya funktioner/subsystem tillkommit eller gamla utgått. Omprövningar, för att utreda huruvida vissa riskfyllda moment i systemfunktionerna verkligen skall realiseras fullt ut för delar, som trots omkonstruktion visat sig medföra alltför stora risker. Även en någorlunda korrekt fördelning på ingående delar/ funktioner kan – p g a samverkan mellan dessa – ge för hög totalrisk. En riskfördelning baserad på systemhieraki är därför sårbar, och en annan modell mer robust mot dessa typer av strukturförändringar kan därför vara önskvärd.

<sup>16</sup> Jfr H ProgSäk avsnitt 4.2.2.



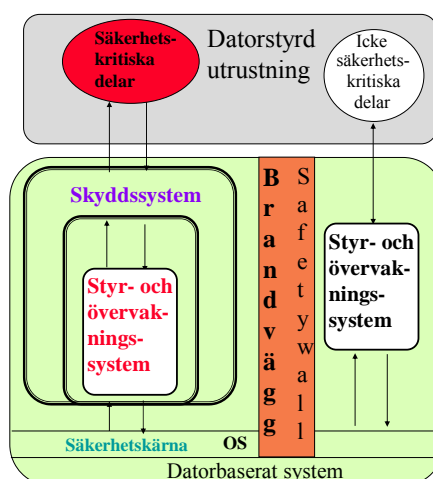
Organisation KC Ledsyst	Titel Programvarusäkerhet –en introduktion	Dokumentnummer KC Ledstöd 14910:38346/02			
Namn Inga-Lill Bratteby-Ribbing	Tel 018-12 02 63	Datum 2006-09-21	Utgåva 1.5	Sid 7(30)	

## 6 Systemsäkerhetsarkitektur

Systemsäkerhet är en egenskap, som måste byggas in i systemet för att denna skall kunna tillgodoses från översta nivån i en systemhierarki och ned till enskilda, kritiska programvarukomponenter samt den grundläggande basprogramvaran (OS) och den hårdvaruinbäddade programvaran. Detta kräver en säkerhetsinriktad arkitektur<sup>17</sup> med en gemensam filosofi för hur den övergripande systemsäkerheten skall bevakas. För system-av-system med olika systemleverantörer är det t ex nödvändigt att bestämma vilka system/delsystem, som skall bemöta olika övergripande säkerhetshot och var extra skyddsfunktioner kan behöva sättas in för att täppa till de säkerhetshål, som kan uppstå, då de olika leverantörssystemen integreras ihop.

För att systemsäkerheten skall kunna upprätthållas på en övergripande nivå fordras bl a att

- Funktioner för styrning av säkerhetskritisk utrustning /aktivitet inte hotar systemsäkerheten,
- Funktioner för övervakning och skydd är tillförlitliga<sup>18</sup> (korrekta under givna förutsättningar),
- Övriga programdelar ej kan påverka säkerhetskritiska delar/funktioner/aktiviteter (medför krav på att kritiska delar skall vara isolerade från övriga delar<sup>19</sup>).



Skyddsfunktionerna kan vara mer eller mindre integrerade med skydds- och övervakningsdelarna. För att bedöma i vilken grad olika programvaruegenskaper är tillgodosedda, räcker det inte att utvärdera dessa hos en enskilda programvarukomponent isolerat; det fordras att programvaran integrerats in i systemet i dess avsedda miljö och användning.

<sup>17</sup> ...och för system-av-system: en arkitekturhierarki.

<sup>18</sup> **Tillförlitlighet** hos en programvara avser dess förmåga att kunna fungera tillfredsställande under givna villkor och specificerad tid (alt per användningstillfälle eller visst antal transaktioner). Egenskapen uttrycker rimligheten för korrekt funktionalitet (dvs inga felyttringar i avsett system, användning och omgivning).

<sup>19</sup> En icke-kritisk del (eller del av lägre kritikalitet), som kan påverka en kritisk (eller del av högre kritikalitet), måste betraktas som kritisk av samma grad och hanteras därefter. Isolering är inte bara ett sätt hålla nere volymer kritisk programvara utan också att hålla nere kostnaderna för utveckling och underhåll av denna.



Organisation KC Ledstöd	Titel Programvarusäkerhet – en introduktion		Dokumentnummer KC Ledstöd 14910:38346/02		
Namn Inga-Lill Bratteby-Ribbing	Tel 018-12 02 63	Datum 2006-09-21	Utgåva 1.5	Sid 8(30)	

## 7 Exempel på skyddssystem

Olika modeller för den övergripande säkerhetsarkitekturen finns. Ofta skiljer dessa mellan olika applikationsområden.

- En del saknar speciella skyddsfunktioner (vilket kan få förödande konsekvenser, se ex nederst).
- Andra modeller inkluderar skydd separerade från styrning och övervakning. Exempel på detta är kärnkraftsverk utrustade med ett eller flera oberoende nödstängningssystem (ett händelseförlopp där dessa kopplades bort refereras i 12.5.1). Då dessa skyddssystem började automatiseras, togs en ny standard fram baserat på de principer och krav, som varit (och är) gällande för maskinvarubaserade realiseringar. Denna standard avser skyddssystem, varför dess krav inriktats mot tillförlitlighet<sup>20</sup> snarare än systemsäkerhet. Styrning- och övervakning vilar fortfarande till stor del på operatörerna<sup>21</sup>.
- Många modeller har skyddsfunktioner integrerade med styrning och övervakning (gäller bl a många inbäddade realtidssystem, t ex vapensystem). I ett fall designades ett torpedsystem med en extra skyddsfunktion, som skulle utlösa en sprängladdning i torpeden, om den hamnade i en bana 180° från utskjutningsriktningen. Under en skjutmanöver till havs fastnade torpeden i tuben. Efter flera misslyckade försök att åtgärda problemet, vände fartyget helt om mot hemmavarvet. Skyddsfunktionen fungerade enligt specifikationerna, och fartyget förstördes<sup>22</sup>. Ett annat fall gällde ett stort antal federala polisbilar, vilka vid uttryckning efter parkering plötsligt hade rusat fram och i ett flertal fall orsakat dödsolyckor. Det visade sig att dessa i efterhand försetts med en extra säkerhetsfunktion (från annan tillverkare), vilken gjorde att bromsljusen vid parkering blinkade i takt med takljusen. Samma krets styrde dock bromsljus och automatväxellås. Effekten blev att säkerhetsfunktionen kom att inhibera växellåset<sup>23</sup>. Dessa exempel illustrerar svårigheterna att tillräckligt väl kunna penetrera under vilka omständigheter tillförlitligheten hos en viss funktion i ett delsystem skall gälla. Analys på hela systemet i avsedd omgivning för alla möjliga situationer hör till nödvändigheterna.

Inom transportsektorn förekommer en annan typ av skydd, de s k antikollisionssystemen. Ett exempel är TCAS, som kommunicerar med egen pilot och andra TCAS-utrustade flygplan i närzonen angående aktuell position och nödvändiga positionsändringar. Att tillförlitlighet är en ytterst väsentlig egenskap för skyddssystem illustreras av flera incidenter, där TCAS varit inblandad, bl a den, som beskrivs under avsnitt 12.5.2. Här var det ett fel i TCAS höjdbestämming samt en felaktig installation av den extra höjddkontroll (med uppgift att varsla vid tillförlitlighetsproblem i TCAS och därefter stänga av systemet), som orsakade, att två plan leddes in på kollisionkurs. Än mer ödesdigert kan det system vara, som saknar skyddssystem, eller som enbart är utrustad med enklare varningssystem utan möjlighet till automatiskt stopp vid akut kollisionsrisk<sup>24</sup>. En svår tågolycka inträffade i England 1999 (se 12.5.3)<sup>25</sup>. Ett kvartal senare drabbades Norge av en snarlik olycka.

<sup>20</sup> IAEA:s ”Safety Guide 50-SG-D3” ligger till grund för IEC 880: ”Software for computers in safety systems for nuclear power stations”, 1986.

<sup>21</sup> Viss automatik finns, bl a gällde detta en del av kontrollstavarna i Tjernobyl-anläggningen, se 12.5.1, ett referat från

- a) Safeware, Nancy Leveson, ISBN 0-201-11972-2, 1995,
- b) NE ISBN 91-7024-619,
- c) Artiklar i SvD 1996-04-24,
- d) Physics Today, Sept 1986.

<sup>22</sup> Källa: N Leveson, SOFT-16, Linköping, Safeware -System Safety and Computers, 1996-03-11--12.

<sup>23</sup> Källa: Anna Wilde Mathews, The Wall Street Journal, Nov 1 1999.

<sup>24</sup> Jfr det svenska ATC-systemet med säker och tillförlitlig drift sedan 1980 (’Twenty Years of Safe Train Control in Sweden’, B Lawson, S Wallin, B Bryntse, B Friman, maj 2000).

<sup>25</sup> Baserad på: a) HSE interim report, 8 Oct 1999, [www.hse.gov.uk/railway/paddrail/interim.htm](http://www.hse.gov.uk/railway/paddrail/interim.htm).

b) The Risks Digest, sept-okt 1999

c) *Safety-Critical Mailing List Forum* ([safety-critical@cs.york.ac.uk](mailto:safety-critical@cs.york.ac.uk)), okt-nov 1999.



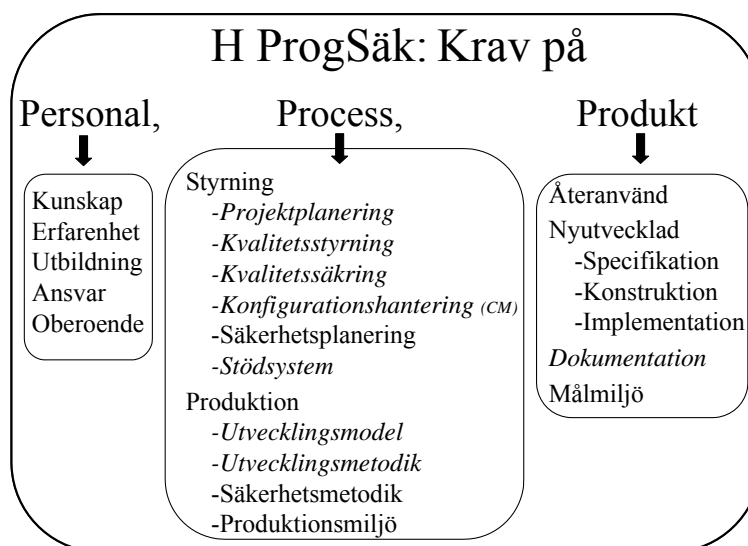
Organisation KC Ledstöd	Titel Programvarusäkerhet – en introduktion	Dokumentnummer KC Ledstöd 14910:38346/02			
Namn Inga-Lill Bratteby-Ribbing	Tel 018-12 02 63	Datum 2006-09-21	Utgåva 1.5	Sid 9(30)	

## 8 H ProgSäk 2001

FMV inledde 1997 arbetet på en handbok i programvarusäkerhet att användas av Försvarmakten och FMV vid anskaffning av programvara i säkerhetskritiska tillämpningar<sup>26</sup>. Handboken, som efter prov och vidareutveckling fastställdes i december 2001, har visat sig användbar även i andra sammanhang (t ex i fall där specifika systemsäkerhetskrav för programvaran saknas) bl a genom att belysa aspekter (utöver krävd funktionalitet) väsentliga att bevaka vid verifiering och certifiering. En engelsk utgåva färdigställdes 2005.

**Generella**<sup>27</sup> *säkerhetskrav* för hela livscykeln har specificerats. Kraven avser programvara som produkt, de processer och produktionsmiljöer som används för att framställa säkerhetskritiska programvarudelar samt nödvändiga kvalifikationer hos personer involverade i dess anskaffning, framtagning och drift<sup>28</sup>.

**Grundkrav**<sup>29</sup>, giltiga även för icke-säkerhetskritisk programvara, finns inkluderade. Samtliga krav har numererats och graderats efter kritikalitet<sup>30</sup>.



Handboken bygger på den systemsäkerhetsverksamhet som fastläggs i H SystSäk. Samma huvudprinciper för riskreducering gäller. De flesta tekniker för säkerhetsanalys är också gemensamma. H ProgSäk visar hur dessa tekniker kan tillämpas på programvara, vilka övriga, speciellt programvaru-inriktade metoder och verktyg som är aktuella, hur koppling mellan traditionell programvaruteknik, systemsäkerhet och tillförlitlighet kan åstadkommas. Vikten av att i ett tidigt skede beskriva en systemsäkerhetsvy som del av den övergripande systemarkitekturen betonas. Möjligheter och problem vid realisering i programvara jämfört med en mer renodlat maskinvarubaserad lösning beskrivs. En översikt av standarder och handledningar inom system- och programvarusäkerhet ges med korta referat,

<sup>26</sup> Försvarmaktens handbok för programvara i säkerhetskritiska tillämpningar, M7762-000531 H ProgSäk (se "Publikationer: Handböcker: H ProgSäk 2001" på [www.fmv.se](http://www.fmv.se)).

<sup>27</sup> Generella säkerhetskrav är oberoende av applikationsdomän och kan ses som en komplettering av Grundkrav.

<sup>28</sup> Uppdragsgivare, beställare, leverantör.

<sup>29</sup> Grundkraven berör främst kursiverade områden i ovanstående figur.

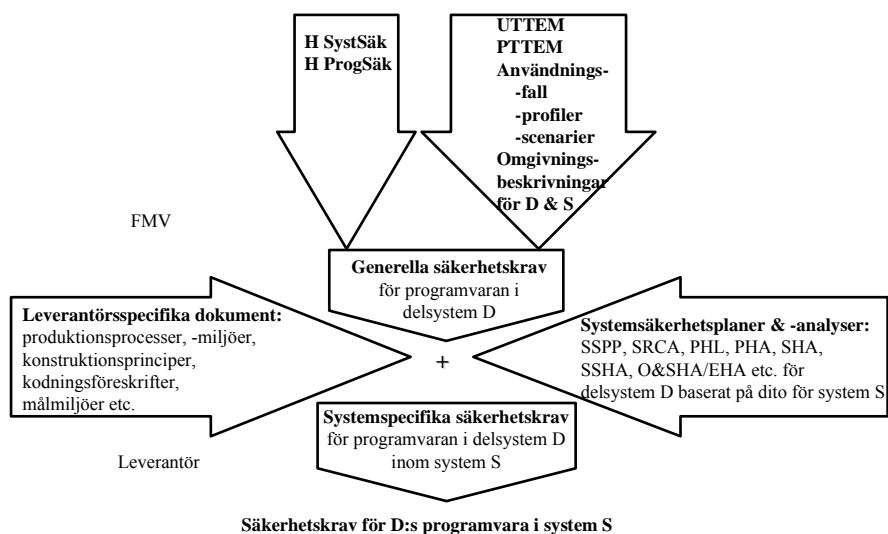
<sup>30</sup> Beteckningen {H} används för hög, {M} för medelhög och {L} för låg kritikalitet. {B} avser grundkrav. Högre kritikalitet medför strängare krav.



Organisation KC Ledstöd	Titel Programvarusäkerhet –en introduktion		Dokumentnummer KC Ledstöd 14910:38346/02		
Namn Inga-Lill Bratteby-Ribbing	Tel 018-12 02 63	Datum 2006-09-21	Utgåva 1.5	Sid 10(30)	

jämförande tabeller, begreppsskiftningar mellan standarderna osv. Checklistor, begrepp, referenser till övrig information inom området ingår. Separat från handboken finns även olika mallar, bl a har den systemsäkerhetsplan, som H SystSäk föreskriver, kompletterats med programvarusäkerhetsaspekter. Korsreferenslistor finns för utvärdering av ett systems kravuppfyllnad samt för jämförelse av H ProgSäk mot annan handbok eller standard. En tillämpning av den senare mallen är en jämförelse mellan H ProgSäk och flygstandarder DO-178B<sup>31</sup>.

H ProgSäk innehåller i princip samtliga generella säkerhetskrav, som kan ställas på programvara vid anskaffning av datorbaserade system. Anpassning till enskilt projekt och system är därför nödvändig. Krav relevanta för aktuell tillämpning väljs ut. Dessa kompletteras med *systems specifika säkerhetskrav* baserade på resultat från säkerhetsanalyser fördjupade ned till programvarunivå. Anvisningar för hur H ProgSäk kan anpassas samt vilka förberedelser en leverantör kan vidtaga inför kommande säkerhetskritiska programvaruprojekt ingår därför.



<sup>31</sup> Se ”Publikationer: Handböcker: H ProgSäk 2001” samt underrubriken ’Hjälpmedel och mallar’ på [www.fmv.se](http://www.fmv.se).



Organisation KC Ledstöd	Titel Programvarusäkerhet – en introduktion			Dokumentnummer KC Ledstöd 14910:38346/02	
Namn Inga-Lill Bratteby-Ribbing	Tel 018-12 02 63	Datum 2006-09-21	Utgåva 1.5	Sid 11(30)	

## 9 Några skillnader programvara – maskinvara

Realiseringar i program- resp maskinvara skiljer sig på många punkter. Här ett urval av aspekter:

Aspekt	Programvara	Maskinvara
Tillverkning	<ul style="list-style-type: none"> <li>• Konceptuella beskrivningar i successivt ökad detaljeringsgrad (från olika specialister)</li> <li>• Konstruktionshjälpmedel realiserade i programvara</li> <li>• Både underlag och hjälpmedel resultat av mänsklig kreativitet ⇒ <i>systematiska fel</i> (logiska fel) möjliga.</li> <li>• Masskopiering ⇒ Identiska kopior (med samma systematiska fel)</li> </ul>	<ul style="list-style-type: none"> <li>• Konstruktionsunderlag och instruktioner för maskinutrustning (från olika specialister)</li> <li>• Arbetsmoment (utförda av maskinoperatörer eller robotar)</li> <li>• Konstruktionshjälpmedel till stor del maskinvarubaserade<sup>32</sup></li> <li>• Automatiserad från givet underlag ⇒ möjlighet både till <i>slumpmässiga</i> och systematiska fel<sup>32</sup>.</li> <li>• Masskopiering ⇒ Kopior kan få varierande egenskaper (och slumpmässiga fel)</li> </ul>
Produkt	<ul style="list-style-type: none"> <li>• Abstrakt, visualiserad enbart m h a olika beskrivningar</li> <li>• Realisering utan fysiska begränsningar<sup>33</sup></li> <li>• Logik enklare, billigare bygga in ⇒ snabbare ändringstakt ⇒ fler tillstånd realiserbara ⇒ ökad <i>komplexitet</i></li> <li>• Möjlighet till självkontroll, automatiserade beslut, snabb respons</li> </ul>	<ul style="list-style-type: none"> <li>• Fysiskt påtaglig och visuell</li> <li>• Realisering med fysiska / mekaniska begränsningar</li> <li>• Logik krångligare bygga in och ändra<sup>32</sup> ⇒ <i>stabila</i> produkter ⇒ lägre komplexitet</li> </ul>
Gränssnitt	<ul style="list-style-type: none"> <li>• Abstrakta: Realisering kan sakna separat specifikation. Förutsättningar kan vara implicita<sup>34</sup>. Komponent kan gå att <i>plugga in</i> i nya sammanhang <i>även om vissa</i> förutsättningar <i>ej</i> är uppfyllda.</li> </ul>	<ul style="list-style-type: none"> <li>• Konkreta: Svårt plugga in komponent, där gränssnitten ej stämmer överens.</li> </ul>
Åldersfenomen	<ul style="list-style-type: none"> <li>• Inget slitage genom upprepat bruk</li> <li>• Ackumulerade avrundningsfel vid lång drift utan omstart.</li> <li>• Successiva förändringar / försämringar av systemtillstånd.</li> </ul>	<ul style="list-style-type: none"> <li>• Slitage kan orsaka ändrade egenskaper (och därmed slumpmässiga fel)</li> </ul>
Underhåll	<ul style="list-style-type: none"> <li>• Genom konstruktionsändringar</li> </ul>	<ul style="list-style-type: none"> <li>• Genom utbyte/reparation av trasiga delar</li> </ul>

<sup>32</sup> VHDL-teknik kommer att medge större ändringstakt samt utrymme för fler systematiska fel.

<sup>33</sup> Fysikaliska begränsningar måste byggas in i logiken.

<sup>34</sup> Implicita förutsättningar: förutsättningar vilka kan ha varit uppenbara vid bygge av det ursprungliga systemet och därför inte explicit uttryckts i dokumentation eller gränssnitt och som eventuellt endast framgår av koden (t ex ur villkor för separering i olika exekveringsfall, ur valet av algoritm, vilken i sig kan vara optimerad m a p viss datastruktur eller att data levereras i viss takt). Detta är villkor lätta att förbise i synnerhet vid smärre förändringar i system eller användningsprofil.



Organisation KC Ledstöd	Titel Programvarusäkerhet – en introduktion	Dokumentnummer KC Ledstöd 14910:38346/02			
Namn Inga-Lill Bratteby-Ribbing	Tel 018-12 02 63	Datum 2006-09-21	Utgåva 1.5	Sid 12(30)	

Aspekt	Programvara	Maskinvara
Underhållsplanering	Säkras av <i>leverantör</i> <sup>35</sup> genom <ul style="list-style-type: none"> <li>• systematiska, väletablerade, ensade <i>procedurer</i></li> <li>• <i>stödverktyg</i><sup>36</sup></li> <li>• <i>egenskaper inbyggda i produkt</i>, testprogramvara och dokumentation</li> </ul> Säkras av beställare genom <ul style="list-style-type: none"> <li>• avtal med leverantör</li> </ul>	Säkras vid <i>slumpmässiga fel</i> av <i>beställare</i> : <ul style="list-style-type: none"> <li>• avtal med verkstad eller</li> <li>• eget reservdelslager</li> <li>• egen utbildning i felsökning, utbyte av komponent, enklare reparationer</li> </ul> Säkras vid <i>logiska fel</i> av beställare: <ul style="list-style-type: none"> <li>• avtal med leverantör</li> </ul>
Begrepp	<ul style="list-style-type: none"> <li>• <i>Reliability</i> ↔ <i>Tillförlitlighet</i></li> <li>• <i>Dependability</i> ↔ <i>Pålitlighet</i></li> </ul>	<ul style="list-style-type: none"> <li>• <i>Reliability</i> ↔ <i>Funktionssäkerhet</i><sup>37</sup></li> <li>• <i>Dependability</i> ↔ <i>Driftsäkerhet</i></li> </ul>

Ovanstående lista skulle kunna göras ännu längre<sup>38</sup>. Här nöjer vi oss med att konstatera att

- programvara ger långt fler möjligheter att kombinera egenskaper mellan ingående delar
- programvaruproduktens kvalitet i högre grad beror av produktionsprocesser<sup>39</sup> o personalkvalifikationer
- systematiska fel kräver andra typer av konstruktionslösningar och underhållsåtgärder<sup>40</sup>
- korrekt åtgärdade felyttringar i programvaran återkommer ej, men nya kan ha introducerats
- statistik över programvarans felfrekvenser svåra få fram / underlaget för skattningar är otillräckligt<sup>41</sup>
- återanvända programvarukomponenter<sup>42</sup> samt verktyg för kodgenerering och drift måste genomgå säkerhetsanalyser och verifieringar i likhet med de för nyproducerad programvara
- likartade begrepp hos maskin- resp programvara kan skilja<sup>43</sup>.

<sup>35</sup> Komplexiteten gör det ej kostnadseffektivt för beställare att bygga upp motsvarande kunskap för att på egen hand klara felsökning och underhåll. Även med viss kunskap finns risk att ändringar förstör arkitekturen.

<sup>36</sup> Exempel: Felrapporteringsystem, referensanläggningar, testprogramvara.

<sup>37</sup> *Funktionssäkerhet* är ett uttryck för sannolikheten att begärd funktion eller tjänst kommer att fungera tillfredsställande (dvs utan felyttringar under specificerad tid och givna villkor). Begreppet myntades ursprungligen för maskinvarukomponenter som ett mått på i vilken utsträckning komponenten förväntas förbli fri från slumpmässiga fel, ofta uttryckt i MTBF (medeltid mellan felyttringar). Jfr begreppet tillförlitlighet för programvara (fotnot 18) som i stället uttrycker förmågan att fungera tillfredsställande, dvs i vilken utsträckning programvaran redan från början är fri från systematiska fel i avsett system, användning och omgivning.

<sup>38</sup> Se H ProgSäk avsnitt 6.6.5.

<sup>39</sup> Abstrakta egenskaper kan inte verifieras fullt ut förrän efter nedladdning i avsedd målmiljö (i vissa fall först efter lång drifttid). En kostsam process, som motiverar tidiga, väl specificerade och strukturerade aktiviteter.

<sup>40</sup> Redundans i form av diversifiering (ej duplicering!), se H ProgSäk avsnitt 4.5.2.4.5 samt 6.9.5: Ariane 5.

<sup>41</sup> Se avsnitt 11 nedan samt H ProgSäk avsnitt 4.3.2.2.6.

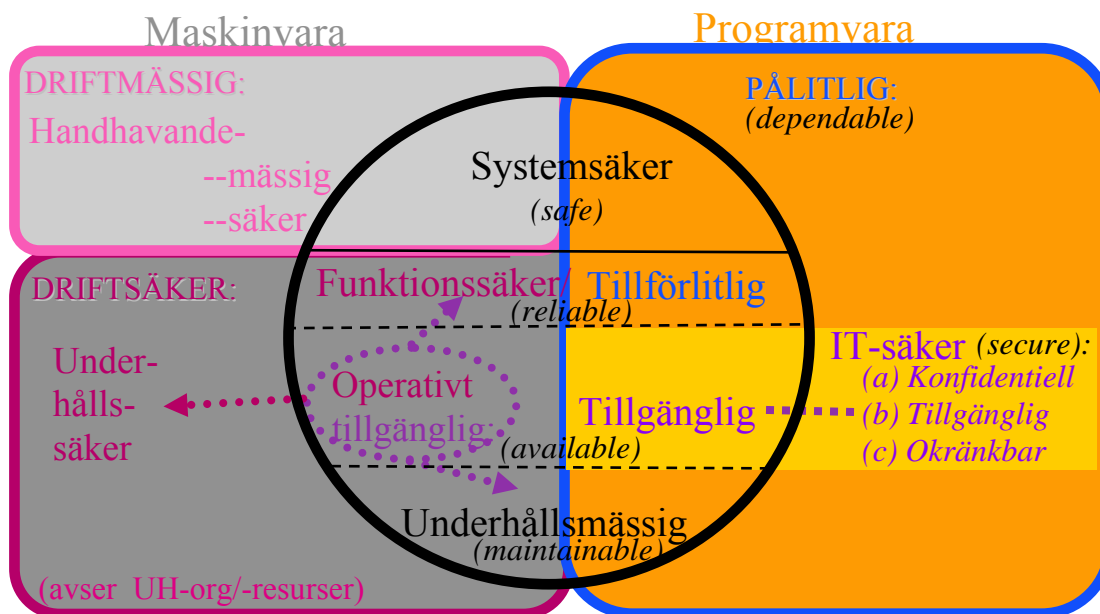
<sup>42</sup> För aspekter på COTS-baserade system se H ProgSäk avsnitt 4.5.1 samt 6.6.5.

<sup>43</sup> För programvara uttrycker *pålitlighet* (eng *dependability*) förmågan att kunna utföra begärd tjänst tillfredsställande, vilket inkluderar funktionssäkerhet/tillförlitlighet, IT-säkerhet, systemsäkerhet, tillgänglighet samt underhållsmässighet (reparer- & utvecklingsbarhet). Se Laprie (*Dependability: Basic concepts and terminology*, ISBN 0-387-88296-8) samt SQUALE *Dependability Assessment Criteria*, www.newcastle.research.ec.org/squale.

För maskinvara talar man i stället om *driftsäkerhet*: sannolikheten, att kunna utföra begärda tjänster tillfredsställande vid viss tidpunkt under givna förhållanden samt m h a erforderliga externa resurser. Driftsäkerhet har gemensamma attribut med pålitlighet, men förutsätter tillgång till en underhållsorganisation: attributet *underhållssäkerhet* (underhållskapacitet) ingår därför, **men ej IT-säkerhet** (frånsett attributet tillgänglighet). Detta är naturligt med tanke på att driftssäkerhetsverksamheten byggdes upp för mekaniserade system, innan programvara kom att bli en viktig systemingrediens. Egenskaper specifika för realiseringar i programvara har därför inte inkluderats. Pålitlighet å sin sida täcker ej in unika maskinvaruattribut: underhållssäkerhet (kapaciteten hos en underhållsorganisation ofta angivet i medelväntetid) samt driftmässighet (se bild nästa sida).



Organisation KC Ledstöd	Titel Programvarusäkerhet – en introduktion			Dokumentnummer KC Ledstöd 14910:38346/02	
Namn Inga-Lill Bratteby-Ribbing	Tel 018-12 02 63	Datum 2006-09-21	Utgåva 1.5	Sid 13(30)	



Tillverkningsprocesser och verktyg för maskin- resp programvara skiljer idag.

I takt med att VHDL-teknik vinner terräng kommer maskinvarukonstruktion mer att likna de tekniker och processer, som nu tillämpas för programvara. Maskinvarans tillverknings- och ändringsprocedurer kommer att snabbas upp. Vi får se mer av systematiska fel samtidigt som utsattheten för slumpmässiga fel kvarstår.

Den gemensamma kärnan av likartade egenskaper på bilden ovan kommer med andra ord att växa. Samtidigt kan nya realiseringstekniker i en framtid innebära, att andra egenskaper och konstruktionsmetoder blir av betydelse<sup>44</sup>. Traditionell elektronik, som styrs enbart av elektriska fält och där miniatyriseringen snart nått sin absoluta gräns<sup>45</sup> kommer att ersättas av nya tekniker: nanometerteknik<sup>46</sup>, amorfa beräkningar<sup>47</sup> och på sikt eventuellt kvantteknik<sup>48</sup>, vilka ger nya möjligheter och utmaningar både m a p realiserings- och lösningsteknik.

Låt oss nu studera några specifika programvaruegenskaper i detalj.

<sup>44</sup> En likartat fall på materialsidan uppkom, då kompositmaterial med andra egenskaper beträffande t ex vridstyvhet började användas i båtskrov och flygplan.

<sup>45</sup> Halvledarteknik: Mikroelektroniken är nu nere på ca 200 nm och anses nå detta tillstånd kring 2015 (35 nm). Ytterligare reduceringar ned till 10-20 nm leder till läckage mellan ledningarna. Därmed skulle slutet på den 20-åriga Moore's lag (beräkningskapacitet /antal transistorer per chip dubblerad var 18:e månad) vara nådd.

<sup>46</sup> Nanoelektronik: Nya material och tekniker (t ex GaAs, aerosol) med möjlighet till trådlös ordning av partiklar tänjer gränsen för Moore's lag än mer: idag tror man sig nå 1 nm om ca 50 år... Avancerad isolering medför att strömmar på 1 eV och mikrovoltspänningar kan räcka för bitsättning – kylbehovet upphör!

Molekylär nanoteknik: Möjligheten till exakt placering av atomer / molekyler för bygge av molekylära maskiner. Realiseringar finns!

<sup>47</sup> Amorfa system realiserade i mikro-, nano- eller bioteknik, det senare med olika enzymer för styrning av grenval i exekveringsträdet. Nya programvarutekniska lösningstekniker för hantering av masskopierade mikrofabrikerade beräknings- och kommunikationselement.

<sup>48</sup> Ett kvantmekaniskt system representerar alla tillstånd samtidigt (t ex var en elektron befinner sig), vilket ger möjlighet till enorm beräkningskraft och miniatyrisering (först i beräkningsögonblicket fastläggs vilket tillstånd som gäller). Helt nya tankebanor, nya lösningsstrategier, algoritmer, språk osv fordras!



Organisation KC Ledstöd	Titel Programvarusäkerhet – en introduktion			Dokumentnummer KC Ledstöd 14910:38346/02	
Namn Inga-Lill Bratteby-Ribbing	Tel 018-12 02 63	Datum 2006-09-21	Utgåva 1.5	Sid 14(30)	

## 10 Vad är 'fel' i programvara?

Det talas ofta om fel i programvaran. Vari består dessa?

Många brukar associera till rena kodningsfel, dvs:

- Krävd funktion utförs ej eller levererar inget resultat
- En oönskad händelse inträffar, t ex
  - \* icke begärd funktion
  - \* felaktig svar / respons / beteende / styrinstruktioner
  - \* avsedd funktionalitet under felaktiga förutsättningar
- Felaktig ordning på begärda händelser
- Rätt händelser vid felaktig tid.

De flesta programutvecklare torde dock vara observanta på den här typen av misstag och lyckas därmed fånga in merparten under olika V&V-faser. Analys av tillbud, som kunnat hänföras till fel i programvara har visat, att konstruktionsfelen endast utgör en bråkdel<sup>49</sup>. Som vi kommer att se i exempel längre fram kan det som betraktas som acceptabelt beteende i ett sammanhang visa sig vara otillförlitligt (i strid mot den funktionella specifikationen) eller t o m osäkert och katastrofalt i ett annat.

Studerar man vad som brukar betecknas som programvarufel finner man att de antingen är *systematiska*

och beror på brister i

- a) specification 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 specifikationer och instruktioner.

Det som till en början uppfattats som fel i en viss programvarukomponent kan visa sig bero på att en ny kombination av komponenter införts eller att användningssituation och miljö förändrats. Sådana fel är naturligtvis mycket svårare att fånga än rena konstruktionsmissar och ställer extra krav på personer inblandade i systemets realisering: ursprunglig systemansvarig<sup>50</sup>, komponentutvecklare<sup>51</sup>, den som står för konstruktion och återanvändning i det nya systemet<sup>52</sup> –och i sista hand– slutanvändaren<sup>53</sup>.

F ö kan 'fel' vara en alltför absolut beteckning vid karaktärisering av programvara. Ofta kan det i stället för 'rätt' eller 'fel' vara mer adekvat att tala om 'graden av tillförlitlighet' i olika sammanhang<sup>54</sup>.

<sup>49</sup> Mindre än 15% orsakas av fel introducerade under konstruktion och implementation, mer än 44% beror på brister i kravspecifikationerna (de senare är f ö de mest kostsamma att åtgärda).

<sup>50</sup> Med ansvar för att brister av feltyp a ovan, ej föreligger.

<sup>51</sup> Ofta omedveten om att en komponent åtskilliga år senare kan komma att återanvändas i ett nytt sammanhang. Ett sätt att gardera sig mot felaktig, framtida återanvändning (och feltyperna b,c,d ovan) är att tillämpa vissa, väl etablerade processer, verktyg och principer vid konstruktion. Se H ProgSäk.

<sup>52</sup> Vilken har att bevaka att feltyp a, e, f ej uppstår.

<sup>53</sup> Som bör vara observant på feltyp f och g.

<sup>54</sup> Hanteringen av Patriot-systemet i Dharahn (feltyp f och g) kom att påverka systemets tillförlitlighet (se 12.5.5).



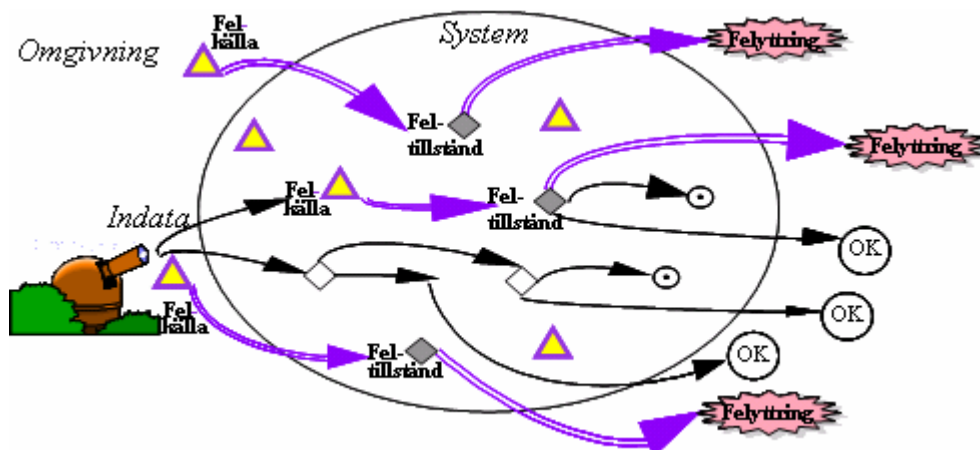
Organisation KC Ledstöd	Titel Programvarusäkerhet – en introduktion		Dokumentnummer KC Ledstöd 14910:38346/02		
Namn Inga-Lill Bratteby-Ribbing	Tel 018-12 02 63	Datum 2006-09-21	Utgåva 1.5	Sid 15(30)	

## 11 Hur skatta felfrekvens hos programvara?

Ett problem med programvara är att felen är systematiska. Det går inte som för maskinvara<sup>55</sup> att för olika komponenter få fram tillfredsställande siffror på sannolikheten att ett fel plötsligt skall inträffa. Från föregående avsnitt vet vi också, att vad som kan betraktas som fel i ett fall inte nödvändigtvis är det i ett annat. Programvaran måste bedömas i sitt sammanhang.

Publicerade systemsäkerhetsstandarder påpekar det omöjliga i att skatta felfrekvens för programvara<sup>56</sup>. Detta komplicerar uppgiften att visa att säkerhetskrav på övergripande nivå<sup>57</sup> är uppfyllda, baserat på att summerade riskbidrag från underliggande nivåer ej överskrider satta toleransgränser.

Det finns dock teknik, där man ur ett stort antal test (användningsfall) försöker skatta felfrekvensen för programvara i visst system, omgivning och användning. Man bör dock observera att de skattningar man med *statistisk felanalys* kan få fram inte representerar sannolikheten att ett slumpmässig fel inträffar, utan att ett antal slumpade testfall inom användningsprofilen stöter på ett systematiskt fel.



Dessa skattningar gäller ej om användningsprofil, omgivning, någon del i systemet eller i tillverkningsprocessen ändras. Antalet kvarvarande fel som mått på tillförlitlighet är följaktligen ifrågasatt: Programvarans tillförlitlighet kan vara hög – trots ett stort antal kvarvarande fel. Det omvända kan också vara möjligt: en liten andel av de kvarvarande felen kan stå för en majoritet av felyttringarna. Ytterligare en komplikation är den mängd test<sup>58</sup> som krävs. I praktiken är det inte möjligt att tillämpa statistisk felanalys för säkerhetskritiska delar med krav på felsannolikheter under  $10^{-4}$  per användningstillfälle eller drifttimme. För system av högre kritikalitet är tekniken inte användbar. I stället blir man hänvisad till säkerhets- och kvalitetsinriktade konstruktionsprinciper för att gardera mot fel. Ariane-fallet är en bra illustration: bättre kvalitet på tillverkningsprocess och produkt hade kunnat leda till

<sup>55</sup> T ex finns kataloger med uppgifter om felfrekvenser för olika standardkomponenter.

<sup>56</sup> Detta är en anledning till att reliability för maskinvara kallas *funktionsäkerhet* (sannolikheten att fungera tillfredsställande) och för programvara *tillförlitlighet* (förmågan att fungera tillfredsställande).

<sup>57</sup> Ofta uttryckt i termer av *risk*, dvs sannolikhet för olycka och dess värsta konsekvens.

<sup>58</sup> Räkneexempel: För att vi skall vara 95% säkra på att felsannolikheten är mindre än  $10^{-4}$  per användningstillfälle måste vi utföra fler än 30 000 test (en övre gräns för vad som är praktiskt-ekonomiskt genomförbart).

Om $P \leftrightarrow$ sannolikhet	$P(0 \text{ fel på } N \text{ test}) = (1-F)^N$	
och $N \leftrightarrow$ antal test	$(1-F)^N < 0.05$	för 95% säkerhet
samt $F \leftrightarrow$ feltäthet:	$N < \log 0.05 / \log (1-F)$	$F = 10^{-4} \Rightarrow < \log 0.05 / \log 0.9999 \Rightarrow 29978$



Organisation KC Ledstöd	Titel Programvarusäkerhet –en introduktion			Dokumentnummer KC Ledstöd 14910:38346/02	
Namn Inga-Lill Bratteby-Ribbing	Tel 018-12 02 63	Datum 2006-09-21	Utgåva 1.5	Sid 16(30)	

att brister i specifikation, konstruktion och återanvändning upptäckts och att kvarstående fel kunnat hindras från att spridas vidare. Händelsekedjan fram till olycka hade kunnat brytas (jfr 3 ovan).

## 12 Återanvändning i säkerhetskritiska delar

Att i en kritisk systemdel återanvända en för annat system tillverkad programvaruprodukt är problematisk –i synnerhet om vi inte har insyn eller information om dess egenskaper (funktionalitet, beteende, förutsättningar). H ProgSäk föreskriver vid återanvändning i systemdelar av högsta kritikalitet samma säkerhetskrav som vid nyutveckling av programvara. För delar av lägre kritikalitet är kraven något mildare<sup>59</sup>. Har vi redan funnit en återanvändbar produkt av acceptabel kvalitet, gäller det att se upp om en ny version av produkten erbjuds<sup>60</sup>. Problem kan också uppstå i fall där programvarudelarna lämnats intakta, men någon annan del av systemet, omgivningen eller användningsprofilen förändrats.

### 12.1 Fallstudie 1: Ett styr- och övervakningssystem

Flera av de säkerhetskrav för återanvändning av programvara enligt H ProgSäk var ej uppfyllda för det system som ingick i Ariane 5-raketen. En anledning var troligen, att systemet sågs som en smärre vidareutveckling av Ariane 4, vilken varit i drift mer än 10 år utan missöden<sup>61</sup>.

Tidigt misstänktes att ”felet” låg i programvaran. Senare har konstaterats att den ursprungliga felkällan snarare kunde hänföras till brister i gränssnitts-specifikationerna (se avsnitt 10: feltyp a ovan). Denna felkälla hade kunna förhindras<sup>62</sup> att sprida sig vidare, om någon av följande brister i systemsäkerhet och kvalitet ej hade förelegat:

- b) Konstruktion:
  - b1) Redundans i form av duplicerad kod.
  - b2) Ohanterade *exceptions*
  - b3) Exekvering av funktion under inaktuell mod.
- c) Produktionsprocess:
  - c1) Inga säkerhetsanalyser av ändringars effekt.
  - c2) Otillräckliga verifieringsprocedurer.
- e) Återanvändning under andra förutsättningar:
  - e1) Intrimningsfunktion byggd på implicita och i detta fall inaktuella antaganden om systemet (trajektoribanor).

Programvaru-inriktade systemsäkerhetsprinciper liksom hög programvarukvalitet är m a o väsentliga för säkerhetskritiska datorsystem. Detta ger fler möjligheter att bryta en händelsekedja från fel-/riskkälla över till felaktigt/osäkert tillstånd i systemet och vidare till felyttring/olycka på systemnivå (jmf bild under kap 3). Detta gäller även i fall, där grundorsaken inte ligger hos programvaran.

<sup>59</sup> Bland delkraven (Se H ProgSäk 4.5.1 för exakta formuleringar): Tillfredsställande resultat beträffande Användning i större omfattning och i liknande miljöer, Leverantörsbedömning m a p kompetens, processer, produkter, Dokumentation över produkt och produktverifiering, Säkerhetsanalyser, Integrationstest.

<sup>60</sup> Alla ändringar dokumenterade, säkerhetsanalyser uppdaterade, integrationstest upprepade, granskningar/analyser/test omverifierade m a p satisfiering av säkerhetskraven.

<sup>61</sup> En sammanfattning av händelseförloppet ges i avsnitt 12.5.4.

Den fullständiga 10-sidiga rapporten finns på [www.esrin.esa.it/htdocs/tidc/press/Press96/ariane5rep.html](http://www.esrin.esa.it/htdocs/tidc/press/Press96/ariane5rep.html).

<sup>62</sup> Detta är ett annat uttryck för, att felet var samverkande.



Organisation	Titel	Dokumentnummer		
KC Ledsyst	Programvarusäkerhet –en introduktion	KC Ledstöd 14910:38346/02		
Namn	Tel	Datum	Utgåva	Sid
Inga-Lill Bratteby-Ribbing	018-12 02 63	2006-09-21	1.5	17(30)

Ariane 5:s haveri illustrerar vanskligheterna med att återanvända programvara: Det går att plugga in en komponent i ett nytt system och köra detta, även om inte alla förutsättningar är uppfyllda. Dessa kan vara baserade på antaganden eller villkor mer eller mindre självklara för det ursprungliga systemet och därför inte explicit formulerade<sup>63</sup>. I ett nytt system –t o m ett mycket närbesläktat sådant–, där några av dessa villkor ej föreligger, kan exekveringen ta andra vägar än de, som varit möjliga/aktuella för det ursprungliga systemet. Ett ur systemsynpunkt säkert beteende i avsedd omgivning och användning kan med återanvänd programvaruprodukt i nytt system eller systemversion föranleda olycka.

### 12.2 Fallstudie 2: Ett skyddssystem

I detta system var det felaktig användning, som ledde till brott mot systemsäkerheten. Användningsområde och handhavande skilde från de för systemet giltiga (se avsnitt 10, feltyp f, g ovan). I ett tidigare tillfälle, då enbart systemets användningsområde förändrats, hade tillförlitligheten i vissa beräkningar visat sig otillfredsställande. I ett senare sammanhang, då systemet även hanterades i strid mot föreskrifterna, blev konsekvenserna förödande.

Den händelse som avses är Patriotsystemet i Dhahran. Olyckan finns beskriven och kommenterad i ett flertal volymer från åren 91-92 i tidskriften *The Risk Digest*<sup>64</sup>. En sammanfattning ges i avsnitt 12.5.5.

### 12.3 Fallstudie 3: COTS-realtidskärna

Bristande kunskap om säkra mekanismer för realtidssynkronisering var den troliga orsaken till att detta system periodvis ej kunde klara sina uppdrag. Med resursanalyser baserade på kända tekniker, stöd i form av skeduleringsverktyg och detaljinformation från COTS-leverantören, kunde problemet dock ha förhindrats. Tur, pengar och en kvarglömd bakdörr in i systemet gjorde att systemet trots allt kunde fixas och dess uppdrag slutföras.

Felet yttrade sig i att systemet ibland ej levererade begärda data. Felkällan var en konstruktionsmiss. Den kommersiella realtidskärna, på vilket systemet vilade, hade inte parametersatts på korrekt sätt. Därvid kom fel teknik vid fördelning av delad resurs mellan konkurrerande processer att väljas. En teknik, där riskerna för ömsesidiga låsningar (*deadlocks*) och hängande system varit kända i 25 år.

Flera alternativa, säkra modeller har sedan dess utvecklats, publicerats och implementerats. Trots detta händer det fortfarande, att kritiska system konstrueras baserad på olämplig skeduleringssteknik.

Det aktuella fallet är marslandaren Pathfinder. En sammanfattning av de programmeringstekniska detaljerna –baserade bl a på diskussioner och referat i tidskrifter och intressentgrupper– ges i 12.5.6<sup>65</sup>.

<sup>63</sup> Att finna vilka dessa är kan fordra noggrant studium av koden eller intervjuer med programutvecklaren.

<sup>64</sup> *The Risk Digest: Forum on Risks to the Public in Computers and Related System* (ACM) på internetadressen //catless.ncl.ac.uk/Risks/ med drygt 30 notiser om händelserna bakom olyckan.

<sup>65</sup> *The Risks Digest* december 1997 samt september 1999. De omfattande diskussionsinläggen under mars 2000 inom *Safety-Critical Mailing List Forum* har sammanställts under [www.cs.york.ac.uk/hise/selist](http://www.cs.york.ac.uk/hise/selist).



Organisation KC Ledstöd	Titel Programvarusäkerhet – en introduktion			Dokumentnummer KC Ledstöd 14910:38346/02	
Namn Inga-Lill Bratteby-Ribbing	Tel 018-12 02 63	Datum 2006-09-21	Utgåva 1.5	Sid 18(30)	

## 12.4 Fallstudie 4: COTS-OS

Kommersiella produkter är inriktade mot att tillfredsställa en så stor marknad som möjligt. Ofta ingår en mängd faciliteter, där endast en bråkdel är intressant (eller önskvärd) för det aktuella systemet.

Ett av de problem säkerhetskritiska, COTS-baserad applikationer har att tackla är, att identifiera och avskärma överflödigt funktionalitet. Detta kan vara besvärligt även om källkod och dokumentation finns att tillgå. Kostnaderna för att försäkra sig om att fel optioner inte träder in kan också bli stora<sup>66</sup>. Konsekvenserna av att inte till fullo förstå funktionaliteten hos en ingående COTS-produkt framgick i föregående exempel. Det finns risk för fler misstag av denna typ även i framtiden. Speciella analysverktyg från leverantören kan därför vara en nödvändighet, för att i förväg kunna studera beteende och prestanda hos en presumtiv COTS-kandidat till aktuellt system.

Idag baseras allt fler system på Windows-produkter, vilka p g a sin spridning (snarare än inneboende kvalitetsegenskaper) kommit att bli de facto standarder – t o m för säkerhetskritiska tillämpningar. Detta är klart oroväckande, bl a därför att källkoden i regel inte är tillgänglig, beskrivningarna inte är tillräckligt detaljerade eller produktens egenskaper och stabilitet är bristfälliga (för att anknyta till föregående exempel: Windows NT använder sig ej av principen prioritetsarb<sup>67</sup>. Enbart tillgång till källkod löser f ö inte problemen (i synnerhet inte vid kodvolymerna på flera miljoner rader)<sup>68</sup>. Förutom speciella analysverktyg för OS-produkten som sådan samt god kunskap/erfarenhet av de generella, programvarutekniska aspekterna för den aktuella produkttypen (i detta fall: realtids-OS) behövs dokumentation över COTS-produktens struktur med beskrivningar över beroenden och interaktioner mellan ingående delar.

Andra problem kan vara COTS-produkter med dolda ingångar, som medger uppkoppling mot andra datorer genom COTS-användarens brandvägg (vilket öppnar upp för såväl informationsöverföring som möjlighet för utomstående att kunna överta styrningen) vilket tillåter leverantören att införa automatiska uppdateringar över nätet (men därmed även möjlighet att ställa om produktens parametersättningar, att byta ut *password*-filer eller göra andra förändringar, som kan få till följd att tidigare programvaror på den egna datorn ej längre kan användas / installeras) eller där full funktionalitet förutsätter uppkoppling mot nätet. Ytterligare egenheter är operativsystem som tilldelar COTS-leverantören högsta prioritet över användarens dator, som lägger in ursprungsdatorns identitet i varje nyskapad fil eller som använder leverantörsunika protokoll i stället för standardiserade (med onödiga bindningar till leverantörens produkter som följd)<sup>69</sup>. Även om dessa faciliteter på olika sätt möjliggör smidig uppdatering av datorns OS, utgör de – tillsammans med andra mer eller mindre kända hål i säkerhetsnätet – allvarliga hot mot systemets integritet, vilket får bäring både på systemets IT-säkerhet och systemsäkerhet.

<sup>66</sup> En studie bekostad av brittiska MOD visar, att kostnaderna för säkerhetsanalyser, certifieringar och riskindringar (typ skal) överstiger de besparingar man kan göra genom att undvika specialbeställda komponenter (SOUP: Software Of Unknown Pedigree, september 2001).

Software Reuse in Safety-Critical Applications, Summary Report, 19 Sept 2001, ©Crown).

<sup>67</sup> Windows NT tillämpar vad man kallar *priority randomizer*, som försöker angripa problemet prioritetsinversion, genom att ändra prioritet hos en slumpmässigt vald process.

<sup>68</sup> Enl uppgift består Windows XP av ca 45 miljoner rader kod.

<sup>69</sup> För fler aspekter: se artiklar/översikter på SESAM:s hemsida (<http://sesam.smart-lab.se/> under 'Arbetsgrupper: Programvarusäkerhet: Produkter'.



Organisation	Titel			Dokumentnummer	
KC Ledsystem	Programvarusäkerhet –en introduktion			KC Ledstöd 14910:38346/02	
Namn	Tel	Datum	Utgåva	Sid	
Inga-Lill Bratteby-Ribbing	018-12 02 63	2006-09-21	1.5	19(30)	

I programvarukretsar finns stark oro för Windows-baserade system i kritiska tillämpningar. Det väckte därför stor uppmärksamhet, då det blev bekant att det nya systemkonceptet *Smart Ship* i sin första installation drabbats av problem.

Publicerad information ger dock ej belägg för att skulden ensidigt kan läggas på operativsystemet. Brister i applikationsprogramvaran torde också ha varit bidragande.

Läs mer om incidenten med USS Yorktown i 12.5.7. Uppgifterna är bl a hämtade från Government News samt The Risk Digest<sup>70</sup>.

---

<sup>70</sup> Government News: //gcn.com/gcn/1998/ (juli, oktober), The Risks Digest (juli 1998-maj 1999), Scientific American (nov -98, mars-99).



Organisation KC Ledstöd	Titel Programvarusäkerhet –en introduktion			Dokumentnummer KC Ledstöd 14910:38346/02	
Namn Inga-Lill Bratteby-Ribbing	Tel 018-12 02 63	Datum 2006-09-21	Utgåva 1.5	Sid 20(30)	

## 12.5 Referat

### 12.5.1 Tjernobyl 1986-04-26

Klockan 01 en fredagnatt –i samband med den årliga avställningen av reaktor 4 i Tjernobyl– inledde några tekniker ett enkelt experiment. Man ville testa i vilken utsträckning en turbogenerator, då dess ångflöde strypts, kunde ge tillräckligt med el till några reservmotorer, som ingick i reaktorns nödkylningssystem. Testansvarig var av denna anledning en elkraftsingenjör –inte en kärnkraftsspecialist.

Anläggningen drevs med el från andra kraftstationer, frånsett några ångdrivna pumpar för matarvatten. Minst 4 kylvattenpumpar måste vara igång hela tiden, för att hålla reaktivitet och klyvningshastighet på säker nivå. Av de drygt 200 kontrollstavarna kunde vissa föras ned i reaktorn från toppen och andra (med automatik) från botten. Säker drift krävde minst 30 stavar nere i härden.

Reaktorn var av typ RBMK , en grafitmodererad modell utvecklad i Sovjet och installerad på ett 30-tal anläggningar. Bränslet utgörs av låganrikat uran och kylning åstadkommes med kokande vatten i trycksatta rör. Den termiska effekten kunde nå ca 3200 MW<sub>th</sub>.

Nedstängningen utfördes långsamt -12 timmar senare var reaktorns kraftproduktion nere i 50% och en av de två turbinerna stängdes av, för att simulera brott i el-försörningen. Nödkylningssystemet kopplades ur 1 timme senare, för att inte dess el-förbrukning skulle kunna påverka testresultaten. Enligt planen skulle produktionen strypas ytterligare ned till 30%, men en oväntad begäran från nätcentralen i Kiev om fortsatt distribution, ledde till att anläggningen kom att köras ytterligare 9 timmar med avstängd nödkylning (ett brott mot förordningarna, som dock inte skulle få någon verkan på olycksförloppet). Testförberedelserna återupptogs därefter. Avsikten var att få ner den termiska effekten till 700-1000 MW<sub>th</sub>. Detta krävde, att det automatiska, finreglerande lokala styrsystemet stängdes av. Enbart det globala, mer grovkorniga systemet, som normalt utgör back-up, hölls igång.

I detta läge blev det svårt att hålla uppe kedjereaktionen. Effekten sjönk under 30 MW<sub>th</sub>. Operatörerna lyckades få upp den, genom att helt dra ut i stort sett alla kontrollstavar (frånsett 6-8 st av totalt 211) ovanför härden. Två timmar senare nådde man upp till 200MW<sub>th</sub>, men inte längre. Man beslöt –trots att detta underskred målet– att fortsätta testet. I detta ingick att mot slutet stänga av 4 kylvattenpumpar. Eftersom 6 redan var igång, startades nu ytterligare två, för att ha stipulerade 4 kvar för eventuell nödkylning. En hel del manuella justeringar krävdes (p g a den låga effekten), för att få till stånd en säker balans mellan ånga och vatten.

Vad AECL (Atomic Energy of Canada Ltd) tror sedan hände –ingen av de inblandade finns kvar att berätta– är att vattenpumparna ökade inflödet av kallvatten i reaktorn, vilket minskade ångbildningen och ledde till att vattennivån i ångseparatorn sjönk under säkerhetsmarginalerna. Under vanliga omständigheter skulle nödstängningen då trätt in.

Manuell reglering av vattenflödet från turbinen krävdes nu, för att kunna fortsätta experimentet. Den automatiska styrningen kunde inte klara detta på ett tillfredsställande sätt, eftersom den var konstruerad för högre effekter. Det var svårt att få rätt flöde, och det var ytterst nära att nödstavarna skulle stänga reaktorn p g a dess instabilitet. Operatörerna kortslöt därför signalerna från skyddssystemet. Inför avstängningen av den enda återstående turbinen, som skulle leda till att reaktorn automatiskt stängdes av, kortslöt man även signalen för nedstängning av reaktorn, eftersom man snabbt ville kunna upprepa testet utan alla förberedelser, ifall det misslyckades denna första gång.



Organisation KC Ledstöd	Titel Programvarusäkerhet – en introduktion		Dokumentnummer KC Ledstöd 14910:38346/02		
Namn Inga-Lill Bratteby-Ribbing	Tel 018-12 02 63	Datum 2006-09-21	Utgåva 1.5	Sid 21(30)	

Den andra turbinen stängdes nu av. Reaktorn körde på 200 MW<sub>th</sub>. Ångan, som normalt skulle gå till generatorerna, hade ingenstans att ta vägen. Samtidigt sjönk vattenflödet, då 4 av de 8 pumparna jobbade mot den avstängda generatoren. Mer ånga producerades, och med dess sämre förmåga att absorbera neutroner, steg inom 3 sek effekten mer än 530MW<sub>th</sub>. I detta läge tryckte operatörerna på nödknappen, som släpper ned samtliga kontroll- och nödstavar. Då så gott som samtliga var uppdragna ovanför härden, fastnade de på vägen ned, varför kopplingarna kapades, för att de skulle falla ned av egen tyngd. Efteranalyser antyder, att grafiten i de hastigt införda nödstavarna på 4 sek kan ha ökat effekten i botten av reaktorn en faktor 100 av dess nominella effekt på 3200 Mw<sub>th</sub>. Två explosioner i tät följd hördes. Detta var 17 minuter efter start av pump nr 2, och 1 dygn och 24 minuter efter det att experimentet inleddes.

Utströmmande ånga vräkte undan det 1000 ton tunga reaktortanklocket och byggnaden rasade samman. Nya explosioner slungade ut radioaktivt bränsle, och grafithärden började brinna. Ett radioaktivt moln steg upp på nästan en km:s höjd och nådde under måndagsnatten Skandinavien. Någon vecka senare drev ytterligare ett moln in över Sverige.

Omvärlden hölls ovetande om olyckan i flera dygn – trots larm om förhöjd radioaktivitet på personal in till måndagens första pass i Forsmark. Frampå eftermiddagen konstaterade kärnteknikerna, att läckan måste härröra från Kievområdet, men först vid 9-tiden på kvällen gjordes ett första tillkännagivande i sovjetisk TV.

31 personer dog omedelbart. Minst 400 000 ukrainare, vitryssar och ryssar tvångsförflyttades. Drygt 3,5 miljoner ukrainare och totalt nio miljoner människor påverkades direkt eller indirekt av katastrofen. Över 15 000 anses fram till april år 2000 ha dött i sviterna av denna och fler offer kommer att skördas. Mellan år 2005 och 2010 väntas sköldkörtelcancerens härjningar nå sin topp. Kostnaderna har beräknats till 140 miljarder dollar.

Med internationellt bistånd göts en betongsarkofag runt reaktor 4. En fond på 768 miljoner dollar har skapats för en miljömässig ombyggnad av denna. Den 29 mars år 2000 godkände den ukrainska regeringen en tidtabell för stängning av den tredje och sista av de fyra reaktorerna. Vid president Clintons besök i Kiev den 5 juni meddelades det officiella slutdatumet: 2000-12-15.

Vilka var då de bidragande faktorerna till olyckan?

Mest uppenbart och det officiella representanter i första skedet helt skyllde på var den mänskliga faktorn. Även sedan mer fakta kommit fram, måste det gravt felaktiga handhavandet fortfarande betraktas som den utlösande riskkällan.

Starkt bidragande till att förloppet kunde få så katastrofala konsekvenser var även brister – väl kända sedan 10 år – i reaktorns lösningsmodell, RBMK. Denna reaktortyp har, i o m att den baseras på s k positiv void-koefficient<sup>71</sup>, en viss inbyggd instabilitet. Minskat vattenflöde i härden ger ökad effekt. Detta förutsätter ett väl intrimmat vattenflöde och leder till ett komplicerat och därmed ytterligt svårkontrollerat styrsystem, som i manuell mod och kritiskt läge kan kräva sekundsnabb reaktion från operatören. Samma press uppkommer ej vid en reaktor med negativ void<sup>72</sup>, vilken i stället strävar mot att stänga av sig själv vid stigande void.

<sup>71</sup> Void: det tomrum, som åstadkommes av ångblåsor i det kokande kylmedlet (vanligen vatten) i en kärnreaktor.  
Void-koefficient: Storhet som anger hur voiden i kylmedlet påverkar reaktorns reaktivitet.

<sup>72</sup> Exempel: Kokvattenreaktorer av typ BWR.

Reaktorer av denna typ strävar m a o mot att stänga av sig själv vid stigande void.



Organisation KC Ledstöd	Titel Programvarusäkerhet – en introduktion		Dokumentnummer KC Ledstöd 14910:38346/02		
Namn Inga-Lill Bratteby-Ribbing	Tel 018-12 02 63	Datum 2006-09-21	Utgåva 1.5	Sid 22(30)	

Ytterligare faktorer, som förvärrade olycksförloppet, var att moderatorn utgjordes av lättantändligt material (grafit) samt att skyddsystemen var otillräckliga (bl a hade anläggningen ingen inneslutningsbyggnad). Ett eftersatt område var också förebyggande träning och satsning på simulatoranläggningar. Det senare hade kunna minska sannolikheten för att riskkällan uppkom. Övriga säkerhets- och kvalitetsaspekter på reaktorkonstruktionen samt styr-, övervaknings och skyddssystem hade kunnat bryta olyckskedjan från riskkälla till haveri.

## 12.5.2 TCAS i flyg över Hongkong 1999-06-28

Ett brittiskt B747-400 med 419 passagerare och ett äldre, koreanskt B747-fraktplan närmar sig Hongkong från varsitt håll efter en lugn flygning över de ödsliga delarna av nordvästra Kina<sup>73</sup>. I dessa områden saknas radar, varför flygtrafiken i stället leds via speciella procedurer och trafikorder per radio. Det koreanska planet ligger på 31 500 fot inne i en molnbank. Det brittiska befinner sig på 33 500 fot ovan molnen. Båda planen har en besättning på 3 man och är utrustade med antikollisionssystemet TCAS<sup>74</sup>.

Plötsligt går TCAS-systemet i den koreanska kabinen ut med varningen ”*Traffic, traffic*”.

Skärmen visar att planet ligger på 33 900 fot, dvs 400 fot ovanför det brittiska.

Samtidigt noterar det brittiska TCAS en diskontinuitet i höjdgivelsen från det koreanska systemet – en stigning från 2000 fot under till 400 fot över eget plan. P g a sin orimlighet ignoreras detta (helt korrekt) av det brittiska systemet, som därför inte går ut med någon varning till sin besättning.

Det koreanska systemet reagerar på att nivåskillnaden mellan planen ligger under minimum och sänder ut varningen ”*Climb! Climb!*” till den koreanska piloten. Denne tvingar upp sitt plan och ser till sin förvåning att avståndet enligt skärmen krymper mot noll. Varningarna blir starkare ”*Increase climb!*”.

Det brittiska TCAS upptäcker nu fraktplanets stigning och beordrar följdriktigt den brittiske piloten att dyka. Planet förs på så vis in på verklig kollisionskurs. Turligt nog verkar enda skadan bli en chockad andrepilot, som vid passage ur molnen oväntat ser sin halva av rutan uppfyllas av fraktplanet, när det brakar förbi i 575 miles per timme endast 600 fot från det egna planets vingspetsar. Hela episoden är över på 34 sekunder<sup>75</sup>.

Omfattande undersökningar visade att British Air:s TCAS (implementerad av Allied Signal) fungerade korrekt. Då det framkom, att en trafikledare påpekat för piloterna vid en tidigare flygning av det kore-

<sup>73</sup> Sammanfattningen är baserad på uppgifter ur

a) The Risk Digest, sept 1999,

b) A Flawed Safety Device in Jets Has Airlines Seeking Answers, William Carley, Wall Street Journal, 12 okt 1999 (i sig en sammanfattning av en 75-sidig, intern brittisk rapport).

<sup>74</sup> *Traffic Alert and Collision Avoidance System* kommunicerar under flygning med pilot i eget och angränsande plan. TCAS utvecklades efter en serie flygkollisioner på 80-talet. Höjdinformation visas på kabinernas skärmar. Vid kollisionsrisk ljuder dessutom varning av typ ”*Climb*” resp ”*Descend*” (endast vertikala korrigeringsanvisningar ges).

TCAS har räddat många liv. I USA skattas att antalet kollisioner i luften minskat från 20/år till ca 4/år.

<sup>75</sup> Slump / tur gjorde att planen på kollisionskurs lateralt råkade passera varandra på 200 m avstånd. Det har framförts åsikten, att om de två jetplanen använt ett GPS-baserat och därmed något mer exakt navigeringssystem, hade kollisionen varit ett faktum. Inget finns som underbygger detta. Ett tillförlitligt GPS-baserat system hade för det första inte lett in planen på kollisionskurs, för det andra vid lateral invisning följt de konventioner som gäller idag, nämligen att hålla på viss sida om centrumlinjen.



Organisation	Titel		Dokumentnummer		
KC Ledstöd	Programvarusäkerhet –en introduktion		KC Ledstöd 14910:38346/02		
Namn	Tel	Datum	Utgåva	Sid	
Inga-Lill Bratteby-Ribbing	018-12 02 63	2006-09-21	1.5	23(30)	

anska planet, att detta sände felaktig höjdsposition, koncentrerades utredningen på dess elektroniksystem.

Fraktplanet hade uppgraderats med TCAS under tidigt 90-tal. Komponenterna var sammankopplade med en elektronikkrets benämnd *Gilhams interface*. Denna nyttjar 11 trådar, som beroende på vilken som är på/av (dvs 1/0), ger en kod för höjden. 31 500 fot motsvarar för 3 av de 11 trådarna 001. Fel i en tråd skulle dock ge 101, dvs 33 900 fot, dvs exakt 2 400 fot högre. Här fanns felorsaken!

Nästa fråga var: Varför hade det kontrollsystem, som bevakar TCAS tillförlitlighet, inte reagerat? TCAS använder en separat dator för insamling av höjddata, vilka kontinuerligt jämförs med primärdatorns höjddata. Vid avvikelser stängs TCAS av och piloten uppmärksammas via ett instrument i kabinen.

Vid British Air:s förfrågningar till mekanikerna på Korean Air framkom efter deras granskning av tråddragningen, att en tråd saknades. I o m detta kom höjdd kontrollen att elimineras, och TCAS kunde fortsätta exekvera även i händelse av höjdfel. Inga varningsmekanismer fanns för felyttringar i själva kontrollsystemet. Inga procedurer fanns heller definierade för mekanikernas översyner t ex för kontroll av att själva kontrollsystemet för TCAS inte var satt ur funktion.

En dold felyttring hade därmed identifierats –aktuell för tusentals andra trafikplan<sup>76</sup>. Det blev aldrig klarlagt om felet funnits med redan vid installationen av TCAS eller vem som utfört denna. Nästa oklarhet gällde felkällans hemvist. Fanns den i *Gillham*-koden för flygdatorn, transpondern eller i tråddragningen mellan dessa<sup>77</sup>? Utrustningarna sändes därför över till leverantörernas respektive lab i USA för test. Transpondern uppgavs därvid fungera klanderfritt. Samma sak konstaterades efter upprepade test av flygdatorn, men då den lämnades på under en längre tid och med ”extra elektrisk kraft på” dök höjdfelet på 2400 fot upp. (Huruvida transpondern också testades på detta sätt framgår ej).

En snarlik incident inträffade nära Hawaii i januari 1998 mellan en B747 och en DC8. TCAS i flygplanet på den lägre kursen rapporterade då en höjd 1500 fot över den verkliga och varslade om att en undanmanöver skulle bli nödvändig.

Även i detta fall saknas utrustning för kontroll av TCAS tillförlitlighet. Den flygledning, som ansvarade för luftrummet upptäckte dock, att uppgifterna från transponder respektive besättning skilde, och felet kunde klaras ut innan piloterna hann följa TCAS:s anvisningar.

<sup>76</sup> Denna installation finns på ca 3000 trafikplan över hela världen (Boeing 747, 737, 727, några Airbus A-300).

<sup>77</sup> Flygdatorn var köpt av *Allied Signal* från *Bendix Corp.*, transpondern kom från *Honeywell Inc.*



Organisation KC Ledstöd	Titel Programvarusäkerhet –en introduktion		Dokumentnummer KC Ledstöd 14910:38346/02		
Namn Inga-Lill Bratteby-Ribbing	Tel 018-12 02 63	Datum 2006-09-21	Utgåva 1.5	Sid 24(30)	

### 12.5.3 Tågolyckan vid Ladbroke Grove 1999-10-05 kl 08.11

Ett snabbtåg på väg österut mot Paddington Station och ett motorvagnståg i motsatt riktning kolliderade vid Ladbroke Grove en höstmorgon 1999. Antalet döda uppgavs strax efter olyckan till minst 30 och antalet skadade 160 –en del mycket allvarligt (till stor del p g a häftig brand från antänd diesel).

Det brittiska järnvägsnätet, som tidigare administrerades av British Rail, är indelat i regioner, där olika privata operatörer via Railtrack är ansvariga för driften. Den reläbaserade förregleringen (*interlocking*) är på många avsnitt föråldrad och i dåligt skick, vilket försvårar underhåll. Haveriutredarna fann dock inga belägg för att signalsystemet inte skulle ha fungerat som avsett under olycksdagen.

Samtliga tåg är försedda med någon typ av säkerhetssystem: alla har AWS några dessutom TPWS eller ATP (se nedan). Längs spåren finns signalljus, som stödjer AWS. Långt ifrån alla är anpassade till TPWS eller ATP. Signalerna kan visa grönt (fritt spår), dubbelgult (nästa signal är enkelgul), gult (nästa ljus är röd), och rött (stoppa!).

**AWS** (*Automatic Warning System*) är ett varningssystem (med rötter i signalteknik från 20-talet), som uppmärksammar föraren på de signaler, som passeras: vid grönt ringer en klocka i hytten, vid dubbelgult, enkeltgult samt rött hörs en och samma obehagliga summerton. Vid gula och röda signaler måste föraren snabbt kvittera summertonen, för att inte bromsarna automatiskt skall slå till. Gula signaler får passeras med försiktighet<sup>78</sup>. Vid rött gäller stopp<sup>79</sup>. Det är alltså möjligt att passera successivt klargjorda spåravsnitt (och faktiskt en hel resa) genom att hela tiden kvittera olika gul-signaler.

Trafiken på spåren mot London är tät –vid högtrafik 1 tåg/min. Köer uppstår. Förarna försöker undvika onödiga småstopp, vilka i slutänden kan resultera i stora förseningar (och böter). Samma signalsekvenser upprepas längs bansträckan för det tåg, som köar bakom ett annat: ljusserien ”röd-enkelgul-dubbelgul-grön” akustiskt förstärkt med ”summer-summer-summer-pling”. Ett snabbt och monotont kvitteringsförfarande blir följd. Med samma summerton för gula och röda ljus kan en stressad förare, som inte uppfattat att en ljussignal visat rött, missledas och tro, att även denna var gul.

**TPWS** (*Train Protection Warning System*) är ett modernare, mer automatiserat system vidareutvecklat ur AWS med täckning på ca 25% av Railtracks spårområden (de mest riskbenägna).

**ATP** (*Automatic Train Protection*) är det nyaste automatiserade systemet.

I det aktuella fallet var expreståget utrustat med ATP. Systemet var visserligen avstängt<sup>80</sup>, men eftersom tåget haft grönt på sista bansträckan, anses detta ej ha påverkat olycksförloppet. AWS visade sig dessutom ha varit påslagen.

Motorvagnståget var försett med TPWS. Det hade passerat ett antal dubbel- och enkelgula försignaler samt 700 m före kollisionspunkten signal SN109, som visat rött. Signalen är illa placerad (svårtolkad vid motsol eller dimma) med minst 8 rödljusspassager registrerade för 1999. I planerna ingick enligt Railway Safety Regulations 1999 därför att SN109 skall anpassas till TPWS före årsskiftet 2004. Hade dessa planer varit verkställda redan i oktober 1999, skulle olyckan ha förhindrats.

<sup>78</sup> S k SPADs, *Signals Passed At Danger*. Bromsning måste inledas här, om någon av de efterföljande är röd.

<sup>79</sup> Får dock passeras efter specialtillstånd från tågcentralen. Vanligt för sträckor ofta drabbade av signalfel.

<sup>80</sup> Anledningen enl huvudkällan: (a) bristande tillförlitlighet ( ”not operational” )

Vid tidigare, liknande incidenter: (b) tidsbrist: start av ATP tar 4 min och kräver att tåget ej är igång



Organisation KC Ledstöd	Titel Programvarusäkerhet –en introduktion		Dokumentnummer KC Ledstöd 14910:38346/02		
Namn Inga-Lill Bratteby-Ribbing	Tel 018-12 02 63	Datum 2006-09-21	Utgåva 1.5	Sid 25(30)	

## 12.5.4 Ariane 5 Flight 501 1996-06-04

Bärraketen förstördes på 3700 m höjd 39 sek efter avfyrning p g a en felyttring i den duplicerade attitydkontrollen. Felorsaken låg i en intrimningsfunktion och dess konverteringar från 64-bitars flyttal till 16-bitars heltal (*signed*), där ett värde större än vad som gick att representera ledde till en felsituation, som inte hanterades av koden. Intrimningen behövdes enbart, för att justera plattformen före avskjutning, men funktionen var operativ 50 sek efter inträde i så kallad flygmod<sup>81</sup>. P g a att Ariane 5 försetts med starkare raketmotorer erhöles under dessa sekunder 5 gånger större värde på den horisontella hastigheten jämfört med Ariane 4 och *overflow* i motsvarande programvariabel. Eftersom ingen *exception*-hantering fanns inlagd i koden, la exekveringen av. Flygdatorn växlade då över till attityddator nr 2 (med identisk maskin- och programvara i hot stand-by), vilken dock hade upphört att fungera av samma anledning. Diagnostisk information kom därvid att tolkas som flygdata med stora, felaktiga korrigeringar och kapselbrott som följd, vilket utlöste den automatiska destruktionsen (på höjd och avstånd från avskjutningsrampen enligt specifikationerna).

I det felande kodavsnittet fanns flera liknande konverteringar: fyra av dessa hade skyddats mot *overflow/underflow* (Operand Error), tre lämnades oskyddade, då man ville undvika risk för att attityddatornas maximilast skulle överskridas. En specialanalys hade utförts av oskyddad kod och i synnerhet dessa flyttalskonverteringar. Varför tre av dessa lämnats oskyddade framgick ej klart av dokumentation eller kod, men referat av förda resonemang indikerade, att man ansett dessa antingen vara begränsade fysiskt eller att marginalerna varit väl tilltagna.

Den bakomliggande faktorn till haveriet var alltså den ändrade hårdvaran, men felkällan kan hänföras till programvaran (närmare bestämt till felaktiga och ofullständiga specifikationer över dess gränssnitt mot hårdvaran). De banvärden (*trajectory*) programvaran utgick ifrån gällde inte för Ariane 5. Det förelåg också en del andra brister hos programvaran. Verifieringarna var baserade på inaktuella banvärden. Inga test hade utförts av attityddatorn i nedräknings- eller flygmod (däremot hade ingående test med övrig utrustning genomförts, dock med simulerad attityddator). Vid granskningar hade attitydprogramvarans gränser ej analyserats i tillräcklig omfattning (med bristfällig testtäckningen m a p gränserna som följd). Trots ändrade nedräkningsprocedurer hade vissa krav förknippade med Ariane 4:s återstartrutiner behållits, utan att effekten av intrimningsfunktionens fortsatta exekvering efter start hade undersökts. Vidare var den redundans som införts i koden anpassad till slumpmässiga maskinvarufel (där övergång till back-up är en vettig strategi), snarare än till systematiska programvarufel (där en bättre lösning är att fortsätta exekveringen, efter det att *exception*-hanteraren lagrat tillbaka en bästa skattning inom variabelns gränser).

Konsekvenserna för haveriet blev ”enbart” ekonomiska. Det verkliga beloppet är okänt<sup>82</sup>, men värdet av bärraketens last (de fyra clustersatelliterna) har uppgivits till 3 miljarder kronor.

<sup>81</sup> Den förlängda exekveringen möjliggjorde, att Ariane 4 snabbare skulle kunna återuppta nedräkningsmod efter ett tillfälligt, kortare stopp, något som inte behövdes för Ariane 5 med dess förbättrade nedräkningsprocedurer.

<sup>82</sup> En siffra runt 500 miljoner ECU nämndes strax efter haveriet.



Organisation KC Ledstöd	Titel Programvarusäkerhet –en introduktion			Dokumentnummer KC Ledstöd 14910:38346/02	
Namn Inga-Lill Bratteby-Ribbing	Tel 018-12 02 63	Datum 2006-09-21	Utgåva 1.5	Sid 26(30)	

### 12.5.5 Patriot-missilen vid Dhahran 1991-02-25.

Det amerikanska Patriot-systemet var avsett för identifiering och automatisk bekämpning av anfallande flyg<sup>83</sup> samt omgrupperingar flera gånger per dag. Specificerad drifttid hade därför satts till 14 tim, och systemet konstruerades för kontinuerlig drift av högst denna tid.

I Dhahran kom systemet att användas i 100 tim utan omstart och mot Scud-missiler med en hastighet på Mach 6. Ledningen kände till att användningsområdet gällde flyg. Tanken var att erfaren personal m h a information från andra anläggningar och med försvarssystemet i manuell mod skulle kunna kompensera att det ej konstruerats för Scud-missilernas hastigheter.

Den förlängda drifttiden gav upphov till avrundningsfel vid vissa konverteringar i systemet (från 24-bitars heltal till flyttal), vilket försakade tidsfördröjningar på 0,3433 sek och avståndsfel på nära 700 m vid målsparning<sup>84</sup>. Systemet kom därvid att tolka eko nr 2 från en identifierad Scud-missil som ett nytt mål<sup>85</sup> och hade därför inte tillräcklig information för att kunna sätta in en bekämpning. Först 30 sekunder före nedslag uppfattades hotet. Missilen träffade en barack, och 29 soldater dog.

Israelerna hade vid tidigare användning av systemet rapporterat en missvisning på 55 m redan efter 8 timmars drift. Amerikanska armén uppges ha fått vetskap om programvaruproblemen ca 2 veckor före attack. Raython fick i uppgift att ta fram en förbättrad version. Motsägande uppgifter har lämnats beträffande när denna levererades, vissa källor anger före –andra efter– attacken. Båda kan dock vara riktiga: en av källorna uppges att den nya versionen anlände till basen dagen före. Där bedömde man att det var mer angeläget att först installera systemet i några anläggningar närmare Irak. Därifrån kom banden i retur dagen efter attacken.

<sup>83</sup> Detta torde motsvara en hastighet på Mach 1 (1200km/h) för flyg och för attackplan på låg höjd Mach 2 (7200 km/h). Tillgängliga uppgifter preciserar dock ej vilka hastigheter systemet byggts för, enbart att det avsåg flyg.

<sup>84</sup> Ett enkelt räknexempel av avståndfelet vid målföljning av företag i Mach 6:

$$7200\text{km/h} * 1000\text{ m/km} * 0,3433\text{ sek} / 3600\text{ sek/h} = 686,6\text{ m.}$$

Refererade källor anger 678 m (förmodligen skrivfel av 687 eller också var hastigheten något under Mach 6).

<sup>85</sup> Ett flygplan skulle inte hinna denna sträcka på motsvarande tid.



Organisation KC Ledstöd	Titel Programvarusäkerhet –en introduktion		Dokumentnummer KC Ledstöd 14910:38346/02		
Namn Inga-Lill Bratteby-Ribbing	Tel 018-12 02 63	Datum 2006-09-21	Utgåva 1.5	Sid 27(30)	

## 12.5.6 Mars Pathfinder 1997-07-04

Efter Pathfinder's lyckade landning på mars inledde Sojourner-farkosten uppdraget att samla in och överföra data (inklusive bilder för *web*-en) via landaren till jorden. Några dagar senare (kort efter att insamling av meteorologisk information påbörjats) upptäcktes att stora mängder data sporadiskt gick förlorade –en följd av att Pathfinder-systemet då och då gjorde total omstart. Senare framkom att samma beteende uppträtt ett par gånger före uppskjutning. Man lyckades då ej reproducera händelsen och avstod från vidare analyser, då orsaken antogs ligga på maskinvarusidan.

JPL-laboratoriet körde en exakt replik av systemet i *debug*-mod för att kunna återskapa feltillståndet och i detalj spåra systemhändelser (kontextväxlingar, synkroniserande objekt, avbrott etc). Först framåt morgonkvisten lyckades den ingenjör, som ensam jobbat vidare, reproducera händelsen. De loggade spårningarna analyserades och visade att återstarterna berodde på *prioritetsinversion*<sup>86</sup>. Detta är för professionella realtidsprogrammerare ett välkänt fenomen, som kan uppstå för vissa *skeduleringsmodeller*<sup>87</sup> baserade på fixa prioriteter. Problemet ligger i, att vald skeduleringsteknik inte kan garantera att processernas väntetid på delad resurs är begränsad eller fri från s k *deadlock*<sup>88</sup>. Det är också sedan tidigt 70-tal känt hur man kan komma runt denna typ av synkroniseringsproblem<sup>89</sup>.

Det aktuella systemet var baserat på realtidskärnan VxVorks från Wind River Systems. Denna ger möjlighet till skedulering dels enligt principen *preemptive priority*<sup>90</sup> dels *prioritetsarv*<sup>91</sup>. En parameter finns för att ange om prioritetsarv skall tillämpas. *Default*-mässigt är denna satt till *FALSE*.

Den begränsade resursen de olika processerna konkurrerade om i detta system var en informationsbus för överföring av data mellan olika komponenter. En administrativ process med uppgift att flytta data ut och in på *busen* fick exekvera frekvent och med hög prioritet. Tillgång till denna synkroniserades via principen ömsesidig uteslutning (*mutex*, *mutual exclusion lock*).

Valet av skeduleringsteknik föll på *preemption*: processprioriteter i systemet tilldelades efter processernas angelägenhetsgrad. Uppfattningen var, att den frekvent exekverande och tidskritiska databusen inte tillät tid för att tillämpa prioritetsarv (en i och för sig felaktig bedömning).

<sup>86</sup> En resurs är låst av en lägre prioriterad process, trots att en högre prioriterad process kräver den.

<sup>87</sup> Algoritmer som definierar principer för i vilken ordning systemets processer behöver få tillgång till begränsade systemresurser (s k skyddade, delade objekt), för att varje process skall hinna utföra sina uppgifter inom föreskriven tid.

<sup>88</sup> Exekveringen hänger sig p g a att en enhet / process väntar på resurs reserverad för en annan, vilken i sin tur är låst (direkt eller indirekt) i väntan på resurs tilldelad den första.

<sup>89</sup> Exempel på fungerande, säkra varianter: *Basic Priority Inheritance*, *Priority Ceiling*, *Immediate Priority Ceiling*. En tidig tillämpning var OS enl a. Bra beskrivningar av problemet prioritetsinversion finns i b-d.

(a) Borroughs MCD (*Master Control Program*).

(b) Experiences with processes and monitors in Mesa, B Lampson, D Redell, CACM vol 23, no 2, Feb 1980.

(c) Priority Inheritance Protocols: An Approach to Real-Time Synchronization, L Sha, R Rajkumar, J Lehoczky på URL-adress //data.uta.edu/~ramesh/cse5326/papers/sha90.html, Sept 1990.

(d) Fixed Priority Scheduling: A Historical Perspective, Audsley, Burns, Davis, Tindell, Weeling, Real-Time Systems Journal, March 1995.

<sup>90</sup> En process av högre prioritet får omedelbart ersätta den exekverande.

<sup>91</sup> Om en process av högre prioritet blockeras, kan resursfördelaren (schemaläggaren) dynamiskt höja prioriteten hos den exekverande processen till samma (fler varianter på denna modell av s k *priority inheritance* finns).



Organisation	Titel			Dokumentnummer	
KC Ledsystem	Programvarusäkerhet –en introduktion			KC Ledstöd 14910:38346/02	
Namn	Tel	Datum	Utgåva	Sid	
Inga-Lill Bratteby-Ribbing	018-12 02 63	2006-09-21	1.5	28(30)	

Den process som samlade meteorologiska data exekverade sällan och med låg prioritet. Publicering av data gjordes via informationsbusen genom begäran om tillgång till resursen via *mutex*, skrivning och därefter frisläppning av resursen. Tanken var att på så vis blockera (temporärt stoppa) övriga processer i behov av resursen tills låset öppnats. Detta fungerade i regel bra.

Emellanåt inträffade dock att en kommunikationsprocess av prioritet medium skedulerades under den korta tid då den högprioriterade informationsbusen var låst i väntan på lågprioriterad meteorologdata. Kommunikationsprocessen kom på så vis att hindra meteorologprocessen från att exekvera och blockerade därmed även informationsbusens process (ett klassiskt exempel på prioritetinversion). Efter en tid av inaktivitet på busen reagerade bevakningsrutinen (*watchdog timer*) med att återställa hela systemet. Insamlad information gick därmed förlorad.

VxWorks innehåller också en C-interpretator, som medger att ett system i *debug*-mod kan uppdateras med C-kod för omedelbar exekvering. Denna option var inte frånslagen i det ivägsända systemet. Likaså var diverse initieringsparametrar (bl a den för val av skeduleringsmodell) lagrade som globala variabler och möjliga att sätta om via C-interpretatorn. Ironiskt nog var det dessa programmeringstekniska brister<sup>92</sup>, som gjorde att det enda som behövdes för att åtgärda återstartsproblemet (när orsaken väl identifierats) var, att ladda upp ett kort C-program via radiolänk och ändra parametervärdet i marsystemet från *FALSE* till *TRUE*.

---

<sup>92</sup> Skarpt system i *debug*mod utgör ett hot mot IT-säkerheten. Globala parametrar ger möjlighet att manipulera data från alla delar av systemet (i stället för under kontrollerade former av det objekt, som äger attributet).



Organisation KC Ledstöd	Titel Programvarusäkerhet –en introduktion			Dokumentnummer KC Ledstöd 14910:38346/02	
Namn Inga-Lill Bratteby-Ribbing	Tel 018-12 02 63	Datum 2006-09-21	Utgåva 1.5	Sid 29(30)	

## 12.5.7 USS Yorktown 1997-09

En uttalad policy i den amerikanska flottans IT-21 initiativ var att nya system skulle baseras på civila COTS-produkter<sup>93</sup>. Bl a förordades Windows NT som OS framför t ex Unix. Utveckling av ett nytt koncept, *Smart Ship*, påbörjades 1995. En bärande idé i detta var att COTS-baserade system och högre automatisering skulle kapa kostnaderna för utveckling, drift och underhåll. En årlig besparing på \$2 miljoner förväntades enbart genom möjligheten till en 10%-ig nedskärning av besättningen.

Den första kryssaren (typ Aegis Missile Cruiser), som utrustades med systemet, var USS Yorktown. Under en manöver i september 1997 drabbades denna av ett systemfel, som slog ut framdrivningssystemet och gjorde fartyget manöverodugligt. Efter 2 tim och 45 min uppges fartyget ha bogserats tillbaka till hemmahamnen för underhåll. Två dagar senare var problemet åtgärdat.

I ett memo från viceadmiral Giffins i oktober -97 anges, att incidenten berodde på en operatörsinmattning av noll i ett datafält till ett distribuerat databashanteringsprogram. En division med detta tal orsakade *overflow* i databasen, vilket ledde till att LAN-konsoler, avlägsna terminalenheter och slutligen själva framdrivningssystemet kraschade. Ur denna knapphändiga och tekniskt bristfälliga beskrivning är det dock svårt att dra slutsatser om den verkliga felkällan. Inga detaljerade analyser har presenterats, vilket gett utrymme för spekulationer och gissningar. Många vill hävda att orsaken låg i Windows NT: dess oförmåga att hindra fel från att spridas över det distribuerade nätet, dess tendens att lägga av i stället för att föregripa eller hantera felen. I detta fall bestod felhanteringen enbart i en för operatören kryptisk diagnos –*Division by zero*<sup>94</sup>. Andra har pekat på möjligheten till interoperabilitetsproblem mellan olika COTS-produkter och systemet. Ytterliga några anser att skulden ligger i applikationen, som borde ha konstruerats så, att enkelfel ej kan föreligga och leda till systemkrasch. Samtliga dessa förklaringar kan vara giltiga. Om t ex systemet utvecklats på en plattform vars flyttalsenhet (FPU) maskar alla flyttals-*exceptions* (däribland division med noll), medger detta fortsatt programexekvering<sup>95</sup>. En programutvecklare omedveten om denna maskning kan missa att garderar applikationen mot denna typ av *overflow*. Flyttas systemet efter utvecklingen till en omgivning<sup>96</sup>, där FPU:n inte *default*-mässigt maskar *exceptions*, leder detta till att systemet lägger av vid *overflow*.

Även om ett Windows NT kan betraktas som en de facto standard innebär inte detta, att de olika plattformar för vilka Windows NT finns implementerad är lika. Just flyttalshanteringen är ett klassiskt exempel, där flytt av en applikation till ny miljö skapar problem. Eftersom flyttal inte kan representeras exakt blir inte beräkningarna identiska vid olika flyttalsrepresentationer. Detta är något ett operativsystem inte kan överbrygga.

<sup>93</sup> Se ITSG på [www.doncio.navy.mil/links/IPTs/Information\\_Technology\\_Standards\\_Guidance/](http://www.doncio.navy.mil/links/IPTs/Information_Technology_Standards_Guidance/).

<sup>94</sup> Systemets beteende bör vara så pass väl konstruerat och verifierat, att inga meddelanden av programvaruteknisk natur från underliggande systemlager släpps utan vidare hantering direkt ut till operatör /användare. Dessa bör ha fångats upp under utvecklingskedet och åtgärdats eller formulerats om till mer användaranpassade formuleringar.

För att ta ett exempel: Vad förväntas den kärnkraftsoperatör göra, som vid nödstängning nås av Windows-meddelandet "Slut på virtuellt minne"?

(En något mindre drastisk tillämpning –i detta fall ett bankautomatsystem– kan studeras på [//home.studit.com/com00120/sparbanken1.jpg](http://home.studit.com/com00120/sparbanken1.jpg)).

<sup>95</sup> Exempel: Intel.

<sup>96</sup> Exempel: Alpha (ITSG<sup>93</sup> anger, att applikationerna skall kunna köra på heterogena plattformar).



Organisation KC Ledsystem	Titel Programvarusäkerhet – en introduktion	Dokumentnummer KC Ledstöd 14910:38346/02			
Namn Inga-Lill Bratteby-Ribbing	Tel 018-12 02 63	Datum 2006-09-21	Utgåva 1.5	Sid 30(30)	

## 12.5.8 V-22 OSPREY i North Carolina 2000-12-11

Denna typ av luftfarkost – en hybrid mellan helikopter och propellerplan – är försedd med tippbara rotor, vilka kan vinklas från ett uppåtläge till ett framåtläge. Övergången från vertikalt flygläge till horisontellt sker automatiskt vid ca 160 knop och kombinerar helikopterns fördelar vid start och landing med propellerplanets snabba marschfart. Detta gör flygfarkosten extremt användbar, bl a genom att den inte kräver landningsbanor och kan hovra som en traditionell helikopter, samtidigt som den har tystare gång samt kan nå dubbel marschfart och flyghöjd. Det hela sker dock till priset av en halverad nyttolast och ett mer komplext framdrivningssystem – inte minst p g a de stora, vridbara gondolerna (*nacelle*), den anordning vid vingspetsarna som håller motor, rotor-propeller. För höjdstyrning kan vinkeln hos rotorbladen ändras i förhållande till rotationsplanet (*pitch*) antingen likartat för samtliga (*collective pitch*) eller i viss orientering i förhållande till flygplanskroppen (*cyclic pitch*). Denna typ av rörelser överförs via en fast anslutning från bladen till en fästplatta – en stor lös rondell runt rotorns axeltapp. De titanlinor, som används, är dimensionerade att klara tryck på 5000 psi (mot 3000 psi för traditionella helikoptrar). Fästplattan kan röras upp och ned längs tappen för 'kollektiv vinkling' av bladen. Vid 'cyklisk propellerstigning' vinklas i stället fästplattan på tappen i en viss riktning. Detta medger 360<sup>0</sup> riktningssändring – att jämföra med flygplanets två (skevroder upp/ned).

Denna teknik hade testats i ett antal prototyper (t ex NASA's XV-15) under mer än 20 år. Flera haverier hade också inträffat – för Osprey 4 gånger under en tioårsperiod<sup>97</sup>. Ospreys MV-22 var den första ansatsen mot en storskalig produktion av hybridplan för militära ändamål (i detta fall för *US Marine*).

Den 11 december 2000 efter normal flygning i *North Carolina* inleddes en hastig nedstigning. Vid 160 knop startade gondolerna övergången mot helikoptermod. I detta skede brast en hydraulledning i vänstra gondolen och övergången avbröts (helt enligt avsedd design). Brottet uppstod då titanlinan utsattes för belastning vid vinkling av axeltappen och i ett försvagat parti (delad mellan hydraulsystem 1 och 3) skavde mot kabelknippet. Den här typen av nötningssskador hade fö noterats sedan 1999 och återfanns efter kraschen på samtliga, återstående Osprey-farkoster.

Styrsystemet var m a o försett med tre, delvis oberoende hydrauliksystem. Förlust av styrförmåga klassas som katastrofal: såväl designprinciper som regelverk föreskriver att möjlighet till enkelfel ej får föreligga. Då kabelbrott noterats tidigare hos hydrauliksystem nr 1 och 3 samt förlusten av vätska sker på några sekunder, designades ett felande system (t ex nr 1) att omedelbart sättas *off-line* medan nr 3 isoleras m h a en avstängningsventil (nr 3 blir p s s inaktivt på vänster sida, men förblir aktivt på den högra). Hydrauliksystem nr 2, däremot, kan fortsätta normalt på båda sidor. En partiell redundans föreligger i och med att det krävs två oberoende felyttringar för att orsaka en katastrof (ett oberoende kabelbrott även på lina nr 2 skulle t ex ha omöjliggjort all manövrering av vänstersidans fästplatta). Vid den aktuella flygningen hade Osprey-farkosten m a o en fungerande hydraulisk styrmekanism för fästplattan på vänster sida och två på höger, vilket är tillräckligt för fortsatt säker flygning.

Nu fanns emellertid hos det primära styrsystemet en *reset*-knapp. Den lyser upp (och blir därmed valbar) under vissa omständigheter. Tanken är, att piloten skall trycka på knappen och därmed återställa systemdatorerna till ett känt, säkert tillstånd. Dock var systemet designat så, att knappen i och med kabelbrottet aktiverades. När den sedan valdes, växlade motorerna mellan negativ och positiv acceleration. Så fort detta skett, aktiverades dessutom *reset*-knappen igen. Men i det läge som förelåg skulle programlogiken över huvud taget ej ha aktiverat denna knapp. Under de sista 20 sekunderna hade *reset*-knappen pressats 10 ggr. De kraftiga växlingarna i dragkraft och stigning ledde till att flygfarkosten stegrade sig och kraschade – fö 5 dagar efter att modellen var avsedd gå i full produktion.

<sup>97</sup> 1992: 7 dödsfall, april 2000:4 och i december samma år:19.