NYHETSBLAD FÖR SESAM
Försvarssektorns Adaintressenters Användargrupp för Software Engineering

RENDEZVOUS

Nr 2 oktober 1999

Innehåll

Försvarssektorns Adaintressenters Användargrupp för Software Engineering

SESAM

Vad är SESAM?

SESAM har tillkommit för att organisera och stimulera samarbete och samverkan inom programvaruområdet mellan försvarsindustrin, FMV och FOA.

Det avtalsfästa syftet med SESAM är ”att genom organiserat samarbete mellan användargruppens medlemmar främja tillförlitlighet och effektivitet i utveckling och vidmakthållande av programvarusystem i Ada inom försvarssektorn”. Inom ramen härför skall SESAM även anpassa, profilera och förnya sin verksamhet med hänsyn till ändrade tekniska och andra omständigheter av betydelse för intresseområdet.

Följande kommer att ske under den närmaste 2-3-årsperioden.

1. SESAM skall allmänt verka för att sprida information om faktorer som påverkar möjligheterna till tillförlitlig och effektiv utveckling och vidmakthållande av programvarusystem. Särskilt skall härvid Adas betydelse i sammanhanget klargöras.

2. SESAM skall i sin verksamhet fortlöpande bevaka möjligheterna att samla, skapa och sprida information om objektiva mät- och andra resultat och erfarenheter vunna vid användning av ”software engineering”-principer och Ada.

3. SESAM behandlar tillvägagångsättet vid utveckling och vidmakthållande av programsystem. Implicit i detta ligger givetvis att använda processer skall tillförsäkra de resulterande produkterna efterfrågade egenskaper. Produktegenskaper som påverkas av processerna är därför av primärt intresse att bevaka i SESAMs verksamhet.

4. SESAM skall i sin verksamhet fästa stor vikt vid att underlätta samexistens mellan Ada-program och programvara skriven i andra språk. Speciellt skall aspekter vid användning av COTS beaktas.

5. SESAM skall där så är möjligt sätta konkretiserade och mätbara mål för sin verksamhet under avgränsade tidsperioder.

SESAM styrs av ett Råd med representanter för gruppens medlemmar. Rådet har till sin hjälp ett Verkställande Utskott (VU) och ett sekretariat.

Rådets ordförande är Björn Källberg, CelsiusTech Systems, tel 08-580 84813.

VU

Bengtsson Christopher, FMV
chben@tranet.fmv.se
Brandt Roger, FMV
robra@fmv.se
Ekfeldt Sune, Enator Telub AB
sune.ekfeldt@enator.se
Johansson Billy, CelsiusTech Electronics AB
bijo@celsiustech.se
Källberg Björn, CelsiusTech Systems AB
bjkae@celsiustech.se

Arbetet utförs i ett två arbetsgrupper:

Ag Metodik
Håkan Edler, CTH/Datorteknik
edler@ce.chalmers.se

Ag Teknik
Torbjörn Andreasson,
Ericsson Microwave Systems AB
torbjorn.andreasson@emw.ericsson.se

Vilka kan vara med i SESAM?

Medlemmarna i SESAM är svenska företag, organisationer och myndigheter (förvaltningar, utbildningsinstitutioner etc) med anknytning till försvarssektorn. Medlemmarna indelas i följande kategorier
- ordinarie medlemmar
- arbetsgruppsmedlemmar
- informationsmedlemmar.

Enskild person kan endast komma ifråga som informationsmedlem.

Inträde i SESAM

För samtliga medlemskategorier gäller att inträde beslutas av Rådet.

För inträde som ordinarie- och arbetsgruppsmedlem krävs status som leverantör till FMV. Dessutom krävs en skriftlig förbindelse att uppfylla åtagande som ordinarie- och arbetsgruppsmedlem.

För inträde som informationsmedlem (erhåller endast informationsbladet) krävs status som leverantör till FMV eller status som myndighet inom totalförsvaret. Rådet kan emellertid anta annan part som informationsmedlem.

För ansökan om medlemskap i SESAM vänd er till sekretariatet.

SESAM-Sekretariatet
Anna Kåsjö, FMV:INFOSYST, 115 88 STOCKHOLM
Tel: 08-782 6745, Fax: 08-66 77 392
Epost: alkas@tranet.fmv.se

Ordföranden har ordet

Ada är säkert säkert!

Nu är det så, att det är vice ordföranden, *v. ordf.,* som har ordet, eftersom ordföranden Björn Källberg är sjukskriven. V. ordf. tänker behandla ett ämne, som alltid är aktuellt, men som fått särskild uppmärksamhet under senare år.

Visst är det förståeligt att folk ofta undviker de goda sakerna i livet och väljer de sämre bara för att de är billigare. Men det är ju egentligen väldigt underligt att man ibland väljer de dåliga sakerna fastän de bättre är billigare! Och det hjälper inte med kunskap, erfarenhet, logiskt tänkande och andra goda förmågor hos folk för att de skall välja rätt. Ja, det vill säga det jag tycker är rätt förstås – eller som jag tror: det *vi* tycker är rätt.

Det är ju inte så, att vi håller på Ada bara för att vi är indoktrinerade, ungefär som en norrköpingsbo gärna håller på IFK Norrköping. Nej, vi tycker att Ada är ett utmärkt programmeringsspråk och vi kan motivera vår uppfattning. Men det räcker inte att säga: ”Det är klart att ni skall använda Ada; det är ett bra och säkert språk!” Det skulle ha samma effekt (kanske inte helt ändå) som om jag till en norrköpingsbo sa: ”Det är klart att du skall hålla på Dju’går’n; det är ett bra och säkert lag!” (AIK:are och övriga må ursäkta valet av lag i detta exempel.)

Nej, vi måste dela med oss av våra värderingar och kanske framför allt av den erfarenhet vi har av den evolution som skett inom programmeringsdisciplinen sedan den kom till. Det är ju faktiskt så, att vi vill ha

säkra program med god kvalitet som gör det vi förväntar oss av dem och inget annat.

Det är kanske det vi skall understryka – inte att språket heter Ada. Den som vill göra bra, säkra program med god kvalitet upptäcker i alla fall snart att Ada är det bästa valet. Detta är ju uppenbart när det gäller att programmera utrustningar som används i starkt begränsat antal som i försvaret. För nyttoprogram som har en mer spridd användning, t ex COTS för PC-världen med miljontals användare, spelar kanske inte språkvalet så stor roll, eftersom programmen utsätts för en jättestor användarskara och (nästan) alla fel upptäcks snabbt. (Men många program skulle helt säkert ha blivit billigare, om Ada hade använts.)

Det måste ju fortfarande vara väsentligast att *resultatet blir riktigt* och att programvaran inte ger upphov till händelser *som vi inte förväntat oss.* Det övriga som god kvalitet innefattar, nämligen möjlighet till ändringar, återanvändning m.m., upplevs kanske inte av användaren som person men måste betyda mycket för användarorganisationen och den som producerar programvaran. Att användaren inte direkt berörs av sådana kvalitetsegenskaper, som är så karakteriserande för Ada, och därför inte upptäcker dem, kanske är en förklaring till att Ada inte särskilt förespråkas av användarrepresentanter.

En annan sak som har med kvalitet att göra är att de som är ansvariga för IT-användningen i en verksamhet och de som förser ett vapensystem med relevant programvara måste veta vad programsystemet verkligen innehåller. Härvid är jag misstänksam mot COTS: Windows 95 visade sig innehålla ett ”ägg” som inte ens Microsofts försäljningsorganisation kände till! Vad kan månne ligga i de ägg, som förmodligen – oss och de hedervärda försäljarna ovetande – ligger i känslig programvara, som upphandlas i form av COTS?!

Numera sker ju programmering inte alls på samma sätt som för tjugo år sedan, då behovet av ett språk som Ada blev uppenbart. Nu är vi på en högre nivå och Java lär vara ett ’hyggligt’ språk. Men nya funktioner måste ju kunna programmeras, och Ada har därvid fortfarande ett mycket stort användningsområde att betjäna. Det vi bör framhålla är den evolution, som resulterat i Ada, och de logiska och ’rättframma’ motiv, som är kännetecknande för kedjan ALGOL – Pascal –Ada 83 – Ada 95. Den evolutionen beskriver nämligen ett framsynt mänskligt tänkande, inte bara för språket som sådant utan även för de hjälpmedel man behöver för att språket skall kunna användas. I *’Summary’* till rapporten om ALGOL 60 [ALGOL] står i första stycket: “This is a language suitable for expressing a large class of numerical processes in a form sufficiently concise for direct automatic translation into the language of programmed automatic computers.” Här betonas språkets egenskaper för beskrivning av de problem som behöver lösas. Därefter kommer kravet på automatisk översättning till ’maskinspråk’. Observera att redan här för fyrtio år sedan avses ett

språk, som är oberoende av den maskinvara som skall exekvera [hjälp, vilket ord!] programmet.

ALGOL var ett stort steg i riktning mot *Software engineering*, men liksom för efterföljarna stöddes det inte särskilt mycket av datamaskintillverkarna (dator hette 'datamaskin' eller 'matematikmaskin' på den tiden) – redan då hade man nämligen 'lönsamhetskrav' även på något så abstrakt som programmeringsspråk! Men ALGOL var gratis. FORTRAN och COBOL, som inte var gratis, var naturligtvis intressantare för de säljande företagen.

I och med Algol infördes stränga typdeklarationer. Typerna begränsades visserligen till de tre standardtyperna *integer, real* och *Boolean*, men det var ändå en enkel och effektiv åtgärd för att minska felintensiteten i programmen.

Pascal införde flera standardtyper, vidgade möjligheterna till att deklarera de typer man önskade och erbjöd ytterligare förbättringar i förhållande till ALGOL. I rapporten om Pascal [Pascal] står i inledningen till den senare delen "The development of the language Pascal is based on two principal aims. The first is to make available a language suitable to teach programming as a systematic discipline based on certain fundamental concepts clearly and naturally reflected by the language. The second is to develop implementations of this language which are both reliable and efficient on presently available computers." Den andra meningen innehåller just det som är grunden för god *Software engineering*, nämligen tillgången till ett programmeringsspråk för utbildning. Medan Algol kom att bli ett språk för beskrivning av algoritmer och procedurer (ett standardspråk i den akademiska världen för detta ändamål), kom Pascal att användas i praktiken mycket mer än ALGOL och blev just genom möjligheten att göra tillförlitliga program nästan ett standardspråk för processteknik och tekniska beräkningar (den tredje meningen i citatet ovan).

I och med Pascal underströks betydelsen av reserverade ord för att omöjliggöra varianter av språket. (Det är med beklagande jag kan berätta om ett Pascal-symposium i Göteborg 1981eller 1982, då ett jättestort dataföretag med stolthet deklarerade att de infört några extra reserverade ord i sin Pascal-version för att ge programmeraren flera möjligheter!)

Ett mycket stort steg togs vid införandet av Ada 83, som ju alla läsare känner till. Det är ett synnerligen gott exempel på evolutionär utveckling: Man behåller det som är bra (reserverade ord, typkontroll, struktur), byter ut det som kan bli bättre, och bygger på med nya önskade funktioner (*separatkompilering av moduler, avancerad felhantering, hantering av abstrakta datatyper, utnyttjande av generiska moduler* och *multiprogrammering)*. Ett stort steg vad avser kraften i språket, men den största betydelsen för programvaruteknologin var säkert det definierade datorstödet för utveckling, APSE, och kravet på validering av kompilatorerna. Efter en erfarenhetstid av cirka tio år förbättrar man Ada genom införande av arvsmekanismer och dynamisk bindning (begrepp som ingen kände till 1958-60 då Algol kom till), vilket anpassade språket till tidens objektorienterade synsätt och gav oss Ada 95.

Just genom evolutionen

behåll det som är bra, byt ut det som kan bli bättre och tillgodose nya behov

har Ada vuxit fram genom seriöst, framsynt tänkande och utnyttjade erfarenheter till det kraftfulla och säkra språk det är idag, synnerligen väl lämpat för god Software engineering.

Nu frågar sig mången läsare: "Vad var det för ämne han sa' att han skulle behandla?" Ja, det ligger inbäddat i vad jag sagt ovan, men skall kanske belysas litet ytterligare. Jag tänker på möjligheten att göra säker programvara och på det krav som legat latent länge, men som har särskilt betonats under senare år, nämligen kravet på *säker programvara i kritiska system*. Under hösten håller ju SESAM ett seminarium inom detta ämnesområde, och vi får då höra hur viktigt det är att säkerheten byggs in från början och att den genomsyrar hela utvecklingsprocessen.

Under tiden fram till seminariet: Håll tillgodo med detta nummer av Rendezvous, vilket såsom alltid tillhandahåller intressanta artiklar, och där flera av dem just betonar säkerheten i programvarukonstruktion.

Med säker hälsning
Christopher Bengtsson

[ALGOL] Peter Naur (ed.), *Revised Report on the Algorithmic Language ALGOL 60*, Copenhagen 1962, sid. 3.

[Pascal] K. Jensen, N. Wirth, *Pascal User Maual and report*, Springer 1975, sid. 133.

Ada i Gripens Styrsystem

Bo Frisberg, Saab AB, SE-581 88 Linköping
Tel: 013-18 32 18, Bo.Frisberg@saab.se

Ada har använts vid utveckling av ett uppgraderat styrsystem till Gripen. Erfarenheterna är positiva, och visar att preemptive scheduling implementerad med tasking kan kombineras med höga krav på tillförlitlighet och ett deterministiskt beteende.

1. INLEDNING

Styrsystemet till JAS 39 Gripen har uppgraderats och är nu programmerat i Ada 83. Även tasking och exception-hantering har använts på ett begränsat sätt (tidigare beskrivet i ref. 1). Traditionellt har flygande styrsystem programmerats i assembler med en enkel cyklisk exekutiv.

Utvecklingen startade 1992, och provflygningar med det uppgraderade styrsystemet skedde i början av 1996 (enligt tidplan!). Leveranser i serieflygplan till flygvapnet startade i slutet av samma år.

Styrsystemet är det första i Gripens datorsystem som programmerats i Ada. Numera har Ada valts för ytterligare delsystem, där utvecklingen pågår för fullt.

2. SYSTEMBESKRIVNING

Gripen är ett flerrollsflygplan (Jakt, Attack, Spaning), och produceras i både en- och två-sits versioner. Skillnaderna mellan dessa versioner hanteras av samma mjukvara.

Flygplanet har ett elektriskt styrsystem ("fly-by-wire"), som utför styrlagsberäkningar, funktionsövervakning, redundans-hantering, dataregistrering, etc. Även autopilot-funktioner tillhandahålls.

Insignaler kommer bl a från sensorer för styrspak- och pedallägen, vinkelhastighetsgivare, accelerometrar och pilotens knappar för modval, etc. Utsignaler skickas till servon som styr sju primära styrytor på vingar och sidroder. Dessutom styrs framkantsklaffar, noshjul, luftbroms och varningslampor. Kommunikation med andra delsystem i flygplanet sker via en 1553 databuss.

Styrsystemet har tre redundanta kanaler med identisk programvara. I varje kanal finns två processorer:

- Primär-processor (MC68040).
- I/O- och backup-processor (TMS320C30)

.

Förenklade backup-styrlagar finns i I/O processorn för att kunna flyga hem och landa säkert i händelse av fel på primärsidan. I/O och backup-processorn är programmerad i C, medan primär-processorn programmeras i Ada. Olika språk valdes för att uppnå diversifiering. Storleken på mjukvaran skriven i C är väsentligt mindre, och förmodas inte behöva ändras i någon större omfattning. Framtida utveckling och uppdatering av systemet förväntas ske på primärsidan där vi har Ada.

De tre redundanta kanalerna utbyter data via en Cross Channel Data Link (CCDL), och de två processorena inom en kanal kommunicerar via ett Dual-Port RAM.

3. ANVÄNDNING AV ADA

Primär-processorn är programmerad i Ada 83. Använd kompilator är EDS XDAda version 1.3 med ordinarie run-time system och med VAX/VMS som värdmiljö. Ada används både för den flygande delen av programmet och för den inbyggda testen (BIT) som utförs på marken före flygning. Erfarenheterna visar att Ada är tillförlitligt och tillräckligt, även för rutiner med nära samverkan med hårdvara. Det har varit möjligt att minimera användningen av assemblerkod (assembler har inte behövts av prestandaskäl).

Den starka typningen och kontrollerade interface mellan paket är kännemärken i Ada som stödjer utvecklingen av tillförlitlig programvara. Abstrakta Data Typer (ADT) har varit lämpligt för implementering av bl a filter och integratorer, som är typiska byggblock i den här tillämpningen.

3.1 Tasking

Styrsystemet är implementerat i Ada med fem periodiska tasks som körs i 120, 60, 30, 15 och 7.5 Hz. Dessa harmoniska frekvenser är gemensamma i hela avionikdatorsystemet (se även ref. 2). Funktioner allokeras till respektive task beroende på vilken frekvens för datauppdatering som krävs.

Dessa periodiska tasks är tilldelade fasta prioriteter i enlighet med algoritmen för rate monotonic scheduling, dvs högre prioritet till task med kortare periodtid.

Scheduleringen är implementerad med ett särskilt task med den högsta prioriteten, som aktiveras i 120 Hz med ett avbrott från en timer (standard Ada interrupt entry). Detta task aktiverar övriga periodiska tasks i rätt intervall med rendezvous. Vidare kontrolleras att ett periodiskt task inte har missat sin deadline. I denna tillämpning innebär en missad deadline att backupstyrlagarna (förenklade med endast en periodisk tråd i 60 Hz) i den andra processorn aktiveras. Det kan noteras att med denna design så är det möjligt att upptäcka en missad deadline innan nästa periodiska aktivering av ett task, dvs tillåten exekveringstid kan vara kortare än periodtiden (men båda måste vara multipler av periodtiden för det överordnade interrupt-tasket).

Ett bakgrundstask enbart mäter och lagrar exekverings-belastningen för de periodiska tasksen (en realtids-timer läses av när bakgrundstasket återupptar sin exekvering efter att ha blivit avbruten).

Vidare finns två tasks med lägre prioritet som endast körs under den inbyggda testen (BIT).

Även om tasking tillåts i denna säkerhetskritiska tillämpning, så gäller följande restriktioner:

- Alla tasks är deklarerade på biblioteksnivå och skapas vid programmets start.
- Inget task tillåts att terminera, vilket innebär att varje task innehåller en yttre loop. Abort är givetvis inte heller tillåtet.
- Alla tasks har unika och fasta prioriteter (inga dynamiska prioriteter tillåts).
- Varken Calendar-paketet eller Delay-satsen används (behövs inte i den här tillämpningen).

3.2 Konsistenta data

I ett säkerhetskritiskt system där preemptive scheduling tillåts, är det väsentligt att kunna garantera att data som utbyts mellan tasksen är konsistenta. De harmoniska periodtiderna gör det möjligt att utforma ett fast schema för dataöverföringar via buffertar (beskrivet mer i detalj i ref. 1). Dessa databuffertar kräver en del extra utrymme i RAM, medan priset i form av extra exekveringstid är minimalt.

Med denna design kan också timingen för dataöverföringar göras oberoende av den aktuella CPU-belastningen, vilket innebär att preemptive scheduling kan kombineras med ett deterministiskt beteende.

3.3 Andra designegenskaper

Eftersom det inte förekommer någon direkt kommunikation eller synkronisering mellan de periodiska tasksen, betyder det också att det inte finns någon risk för prioritetsinversion, deadlock eller annan oönskad blockering.

All hantering av tasking och databuffertar hanteras på en exekutivnivå som är skild från funktionerna i själva applikationen. Det innebär att de sekventiella delarna av programmet kan analyseras och testas separat. Detta är också en väsentlig fördel ur underhållssynpunkt. När gamla funktioner ändras eller nya läggs till, så påverkas normalt inte ramverket (exekutivnivån med tasking och dataöverföringar).

3.4 Exception-hantering

Exception-hantering är en annan mekanism i Ada som har diskuterats och ifrågasatts ur säkerhetssynpunkt. I styrsystemet i Gripen så hanteras exceptions, men det sker ingen avancerad återställning efter ett exception. Exceptions i programmet bör givetvis undvikas så långt det är möjligt. Men det kan vara svårt att förutse alla situationer där ett exception kan uppstå, t ex kan ett exception utlösas av ett hårdvarufel.

Om ett exception uppstår, så registreras alltid typ av fel och var i programmet det inträffade. Exception-information och andra felindikeringar lagras i ett icke-flyktigt minne för utläsning efter landning.

Den allmänna åtgärden efter ett exception

är att aktivera backup-styrlagarna (på samma sätt som efter en missad deadline). Undantagsvis kan också fel fångas i isolerade procedurer som inte är kritiska för styrningen. Ett exception hanteras då lokalt, och en korrekt normalfunktion kan fortfarande garanteras. Ytterligare en speciell användning av exception-hantering är för urgång ur BIT-exekveringen på marken, om flygplanet sätter igång att rulla med en viss hastighet och är på väg att lätta (detta är inte det normala sättet att gå ur BIT).

Hittills så har inget exception inträffat under flygning.

4. FRAMTIDEN FÖR ADA I GRIPEN

Ada har nu valts för flera uppgraderingsprogram i Gripen:

GECU (General Electronic Computer Unit) som innehåller bränsle-, hydraul- och luftsystemfunktioner, programmeras nu i Ada.

I de nya PowerPC-baserade D96-datorerna (plattformen kallas även MACS, Modular Airborne Computer System) blir det möjligt ha program skrivna i både det tidigare använda Pascal/D80 och Ada (se artikel i Rendezvous nr 1 1998). Just nu sker utveckling i Ada av ett nytt navigerings- och landningssystem, som skall finnas i en av CPU-erna i systemdatorn.

Ada var inte med i Gripen-projektet från början, men nu kommer hon att vara i luften under lång tid framöver.

5. REFERENSER

[1] B. Frisberg. Ada in the JAS 39 Gripen Flight Control, System, In Lars Asplund (Ed.) Ada-Europe'98 Conference Proceedings, LNCS 1411, Springer Verlag, 1998.

[2] D. Folkesson, Principer för realtidsexekvering i svenska militära avioniksystem, SNART-konferensen 1993.

Reserapport från Ada Europe konferensen i Santander, Spanien 7-11 Juni 1999

Håkan Wissman, Celsius Aerotech

Detta var min första Ada Europé konferensen. Universitet och högskolor utgjorde den största delen av deltagarna och en ännu större del av föreläsarna. Jag tyckte att de amerikanska föreläsarna var de bästa föreläsarna med lite ytligare föredrag, medan de europeiska föreläsarna hade djupare och mer specialiserade föredrag, men var mindre slipade föredragshållare. Kanske skulle jag behövt ett par extra års erfarenhet för att uppskatta de europeiska föredragen som till största del gavs av doktorander, med en långt större erfarenhet än mig.

På konferensen kunde man märka en klar övervikt av GNAT förespråkare. En ny version, 3.12, ska komma någon gång efter sommaren, som väl tyvärr nu har nått sitt slut.

Då Kristina Lundkvist i senaste AiS-Brevet sammanfattat större delen av konferensen, ser jag ingen anledning att göra detsamma. Jag tänker istället belysa två andra föredrag som gavs under veckan.

Realtids arkitekturer

Efter tisdagens inledningstal inleddes konferensen med ett ”Invited talk” av Douglass Locke från Lockheed Martin Corporation i U.S.A. Hans föreläsning handlade om realtids applikationer ur ett arkitektoriskt perspektiv.

Även om det kan tyckas finnas ett oändligt antal sätt att designa ett realtidssystem på, så kan dessa delas in i fyra familjer: tidsstyrt, händelsestyrt, ”Pipeline”, och ”Client-Server”-styrt.

Föredraget byggde på egna erfarenheter som inhämtats under många års arbete.

Den tidsstyrda arkitekturen är den äldsta

och enklaste av de fyra. Detta är det vanligaste valet för säkerhetskritiska system. Tidsstyrda system är det dominerande valet för flygplan-, missil- och industriell produktionsstyrning samt andra relativt enkla applikationer.

Händelsestyrda system tar nu mer och mer över från de tidsstyrda systemen, förutom inom de säkerhetskritiska systemen där certifierare fortfarande förespråkar tidsstyrning. Fördelen med händelsestyrning ligger i en lägre underhållskostnad och större robusthet vad avser att nå tidskraven för större system. Med verktyg som Ada Revenscar Profile kanske andelar även kan tas inom säkerhetskritiska system.

I ”Pipeline”-systemet propagerar händelser genom systemet och delprocesser kan behandlas av separata processorer. Detta innebär ett komplext system som är svårt att förutbestämma. Systemet används inom flygtrafikledning och ibland inom fordonssimulerings system.

”Client-Server”-styrda realtidssystem har inte varit vanliga i rena Ada applikationer. Med CORBA tros denna arkitektur växa i popularitet och troligtvis användas i system som flygledning, industriautomation och bevakningssystem.

Med de erfarenheter som C. Douglass Locke har skulle han för ett distribuerat system välja ett ”Pipeline”-system, men om tillgång till ”real-time”-CORBA finns skulle detta utgöra en stark motkandidat. För andra system skulle tids- eller händelsestyrning väljas. Troligtvis händelsestyrning då det är det bästa i avseende på underhåll, tillförlitlighet och kostnadsbesparing. Dock skulle valet bli tidsstyrning för ett säkerhetskritiskt system.

Testning

Bruno Hémeury från Marconi Research Centre i England höll ett föredrag om programvarutestning. Långt ifrån det roligaste ledet i utvecklingskedjan, men ett av de viktigaste.

I det försök som låg till grund för föredraget hade man studerat fyra olika sätt att kontrollera mjukvaran: kodgranskning, ”Fagan inspection”, Rational Ada Analyzer och TestMate. I testet ingick tre personer alla med programmeringserfarenhet, men bara en med Ada erfarenhet. Det man granskade var kod som genomgått kompilering. För varje testalternativ fanns en begränsad tid. Bruno var dock medveten om att testgruppen varit liten, men den skulle utvidgas i kommande experiment.

”Fagan inspection” bygger på sju steg:

1. **Planering:** Välja ut testpersonal, ta fram nödvändig dokumentation och planera testerna.
2. **Inledning:** Författaren gör en kort presentation som beskriver innehållet och målet med dokumenteringen/programvaran.
3. **Förberedelse:** Varje deltagare läser dokumentet och förbereder sin roll. Rollerna är granskningsledare, läsare (tolkar några kodrader i taget), författare (förklarar sin intensioner och rättar feltolkningar) och testare (ger förslag på hur tester sak utformas).
4. **Granskningsmöte:** Granskningsledaren tillser att alla fyller sin roll. Gruppen koncentrerar sig på att hitta fel, inte åtgärda dem.
5. **Förbättringsprocess:** Granskningsledaren kallar till möte och förbättringsåtgärder i utvecklingen diskuteras.
6. **Omarbetning:** Granskningsledaren och författaren går igenom felen och bestämmer vilka åtgärder som skall vidtas och vilka som kräver ytterligare utredning.
7. **Uppföljning:** Felrättningarna granskas för att undvika felaktiga rättningar.

Enligt testet hittar man två och en halv gånger fler allvarliga fel med ”Fagan inspection” än med kodgranskning. Med hjälp av Ada Analyzer hittades något fler fel än med kodgranskning, men väldigt få av felen hittades av båda metoderna, de kompletterar alltså varandra. TestMate befanns vara krångligt och den största delen av tiden gick åt för att få verktyget att fungera, vilket gjorde att väldigt lite tid blev över till testning och att inte speciellt många fel kunde hittas.

Som slutreslutat av experimentet fann man att ”Fagan inspection” i kombination med Ada Analyzer var det effektivaste sättet att testa kod.

Requirements management as a matter of communication

Ingmar Ogren
Tofs corporation, Fridhem 2, SE-76040 Veddoe, Sweden
URL: http://www.toolforsystems.com, E-mail: iog@toolforsystems.com

Abstract. Requirements work all started as a dialogue between a vendor and an end user. Nowadays this dialogue has been complicated through introduction of marketing and acquisition personnel. This has led to introduction of requirements specifications, but the need for a basic dialogue is still there since a specification cannot capture the increase of knowledge that will always take place during development.

An efficient dialogue must not only include requirements, but also stated missions, problem management and understandable formalism for the system's structure and behavior.

Further study of the dialogue shows that requirements management must be managed as a process in parallel with processes for development and verification.

INTRODUCTION

Once there was a buyer (end-user) who asked a producer: "Can you sell me this and that and what is your price?"

Alternatively the producer could ask a buyer: "Do you want to buy this and that at this price?"

This was a long time ago, before the days of complex systems, when both buyers and producers understood the meaning and the use of "this and that".

To-day, with complex systems, the situation has become somewhat more complicated:

- The buyer is not always the end-user since the end-user needs to take help from acquirers and contract people, who may understand more about bargaining and contract issues than about the end-user's real needs
- The seller is not always the producer since the producer needs to take help from marketing and sales people, who may understand more about marketing than about the product
- Nobody involved really understands "this and that", because the product sold is a new complex system, which was never seen until it is delivered.

Still the basic questions, cited above, are still asked but the complicated situation necessarily makes the answers somewhat misty, leading to a situation where we need requirements specifications and requirements management.

TODAY'S SITUATION

Today we have gotten into a situation where all too often a system project may run like this:

- The end-user and the acquirer put together a requirement specification. They may take help from one or several consultants to make it really complete and correct. The result is a specification, which requires considerable work just to be read and understood.
- The acquirer asks for proposals, based on the specification.
- Several vendors look into the specifications, with various reactions, such as:
  - We can do it, but we do not have the time to analyze these requirements until we get the contract, propose 40 M$.
  - This looks interesting, but these requirements need to be analyzed, propose 3 M$ to analyze the requirements and to build a first model of the system.
  - We do not understand these requirements, but we desperately need business, propose 20M$, with 30% in advance.
- The acquirer interprets the acquisition regulations and gives the contract to the third vendor as this was the one with the best price.
- A couple of years pass and the vendor tries to understand the requirements and to build a system in accordance with this understanding. During the process of building the understanding, the vendor will find a number of

inconsistencies, contradictions and gaps in the requirements.

- The vendor runs out of money and comes back to the acquirer saying: There are some problems with this contract, which need to be sorted out. We need an additional 20M$ or we cannot manage to complete the contract.
- The acquirer is then left with two alternatives:
  - Not paying extra, not getting any system and possibly causing the vendor to go bankrupt.
  - Paying extra and getting a system, which is probably late and not up to the "real needs" since the end-user's understanding of needs has grown during system development.

THE NEED FOR DIALOGUE

As discussed above there are obviously problems and if you talk to people involved for example in building military software systems, you will get some clear opinions as to the causes of the problems:

The military end-user: "These keyboard monkeys do not understand a thing about military needs. They do not even understand that you cannot put an M317 backwards upon a A32!"

The programmer: "These brass-hats do not understand a thing about their own systems, look here where they try to express accuracy percentage in inches!"

You can not have a dialogue without mutual understanding

This may be somewhat funny, but as long as the most important players in system production, the end-user and the producer talk about each other instead of to each other, the prognosis for the system to be produced is not very good.

What is needed is dialogue and still more:

- understanding in a language, which is common among everyone involved in a development
- respect for each other's professional competence, meaning that you stop calling other participants in the development funny names and start respecting their professional competence
- Courage in the sense that you dare to speak up whenever someone says or writes something you do not understand.

TAKE CARE OF THE KNOWLEDGE GROWTH

Provided you get the dialogue going between end-users and producers in a complex system development project, it is inevitable that knowledge will grow among those involved. The knowledge growth may reveal fundamental glitches in the original specification and/or in the original proposed solution.

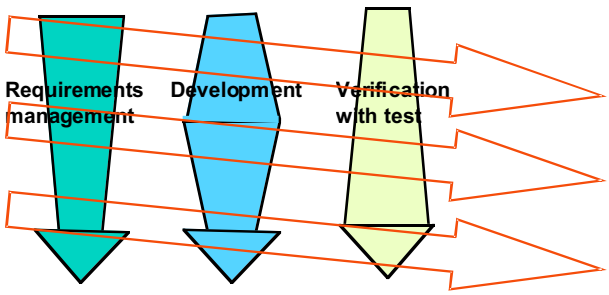
If you then have a fixed contract with negotiated deliveries and payments, the best that can be said about the situation is that it is a real challenge to the contract people, who may not even be able to understand that there is a problem.

What you need is a work principle, which anticipates that knowledge will grow during system development and allows this new knowledge to change the direction and content of the development effort. To find a contract form, which allows this consideration for growth of knowledge is the real challenge for the contract people.

REQUIREMENTS ON THE REQUIREMENTS DIALOGUE

Besides the obvious need for mutual respect, indicated above, some requirements on the dialogue and its language are:

- The dialogue must use a language, which is readily understandable by both end-users and developers after no more than a few hours' training.
- The language used must be sufficiently formal to avoid misunderstandings.
- The language must be able to describe objects to be managed by the system under development.
- The language must be able to describe system performance precisely.
- For critical systems, the dialogue must support discussion of fault-tolerance issues including possible unexpected operator behavior.
- The dialogue must include definition, discussion and decision on the problems, which will surface during development of any non-trivial system.
- The dialogue must anticipate that knowledge will grow during system development and allow this knowledge to influence requirements and design solutions.
- The dialogue must accept as a fact that requirements management, development and verification are three processes which must be run in parallel during a systems engineering effort.



Successive deliveries with parallel processes

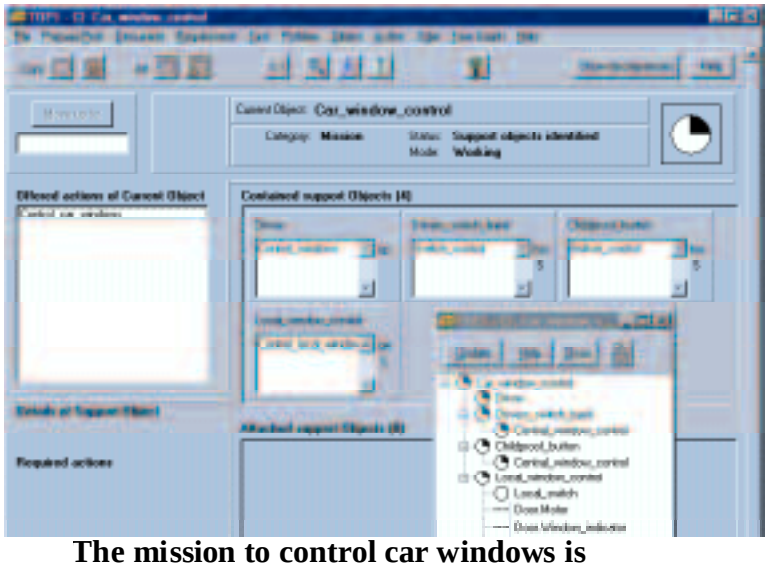
ELEMENTS OF A SOLUTION ALTERNATIVE

As understood from the reasoning above, there is more to requirements management than writing and accepting a requirements specification. In the following some aspects, which should be considered as a basis for the necessary dialogue, are discussed

Missions and scenarios To the end-user the missions of a system are often so obvious that he does not even state them. Instead he defines a set of scenarios, which may or may not cover the complete mission space. On the other hand the developer will interpret the specification and create a number of "Use cases", which also may or may not cover the mission space.

To create understanding it is necessary that the end-user states the missions clearly as these constitute the foundation of both requirements and design work. Scenarios can then be added to clarify the meaning of the missions, but the missions are fundamental.

One way to express missions is as "Mission objects", which are supported by other objects used to complete the mission at hand. See the figure below for the mission to control the windows of an automobile.



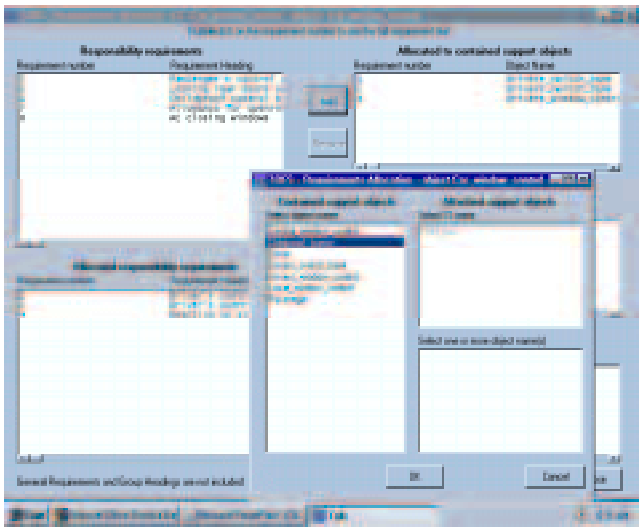
The mission to control car windows is supported by

Original requirements are often stated in a requirements specification. They may also be found in published standards and regulations. For example railway systems are heavily regulated.

To make it possible to work with the original requirements, you need a database. Many system engineering tools offer such a database where you can insert the requirements through copy/paste from a document or even have the

original document automatically parsed for requirements. You can also use common office tools like MS Word or MS Excel to get a low cost requirements database.

It is however strongly recommended to do the requirements insertion manually as you will need to check uniqueness, testability, contradictions etc. These checks will give valuable understanding of any problems, which may pertain the original requirements.



Requirements allocation in a compositive

Derived requirements. Since knowledge grows during development and since design decisions are taken, new requirements will surface. These are called "derived requirements" and they should be input into the requirements database, marked as "derived".

Allocation through a compositive structure. Provided that the system under development is modeled as a compositive structure allocation of requirements is simple. A compositive structure is a structure where the system is composed from objects, connected through their interfaces in such a way that it is always obvious which objects depend on which other objects to complete their action. The compositive structure makes it possible to "float" requirements downwards through the structure until you find the object, which the requirement should be tested with. The requirement then becomes "fulfillment requirement" of that object.

One tool, that supports such compositive structures is Tofs (Tool For Systems). The figure above shows the screen used for requirements allocation.

Problem management. Management of problems or issues is a very important part of the dialogue during system evolution, since problems will inevitably surface and since most of these require a combined effort from end-users and developers to find a feasible solution.

This means that the dialogue must include problem management, including:

- problem statements with category and priority
- problem headings and numbers
- solution alternative
- solution decisions.

It is possible to manage this with an ordinary word processor, but a tool, which supports at least numbering of the problems, is preferable.

Understandable formalism. An end-user, who has expressed himself in clear English, with some explaining diagrams would normally believe ha has made himself completely clear. That will also often be the case, but is amazing how such clear requirements can be transformed into software and hardware, which does quite unanticipated things.

It is doubtful if you should attempt to make the end-user write formal specifications since the necessary knowledge to do so is normally not available at specification writing time.

What could however be done is to provide a formal representation as part of the design effort with two objectives:

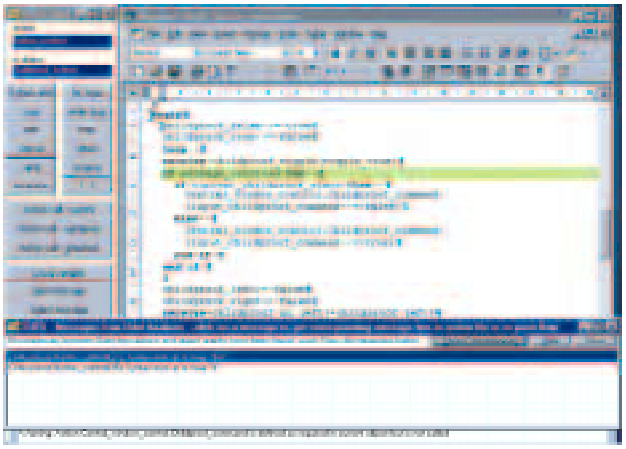
- to get a reconfirmation from the end-user that the specification is understood in an acceptable way
- to give a detailed and formal basis for the implementation work, be it hardware, software or an operator role.

One way to get this formal description is to use "formalized English", a simplified language containing:

- variables of defined types

- control structures
- comments.

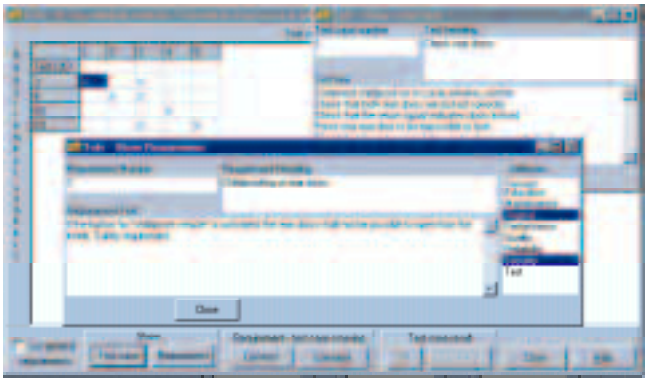
As an example is shown above a description of part of the software in “Car window control”.



Analysis of formal English

Test specifications. Requirements are not much good unless they can be tested. Provided that requirements are distributed to design objects as “fulfillment requirements”, you can design test cases for each object to test that these fulfillment requirements are met.

To ensure that a correct set of test cases has been written, they should be reviewed by the end-user. It will then help if the test cases are presented together with the requirements, which they are intended to verify. One way to do this is to produce a requirements/test case matrix. As an example is shown the matrix for the software object “central window control” in “Car Window control” example:



The test matrix connects

CONCLUSIONS

Some aspects of requirements management have been discussed finding that the single most important issue is to establish a dialogue between the end-users and the producers of a system.

It has also been found that the original requirements specification is only a part of the information required for successful requirements management.

The issue of understanding has been discussed, stressing the necessity to make the requirements documentation understandable both to the end-user and to the developers concerned.

Since the dialogue will result in more information than can be comfortably managed manually, computer-based support tools will be helpful.

It has further been discussed the necessity to create a work situation, where everyone involved communicates and respects the professional competence of others in the process.

Another important conclusion is that you cannot “do the requirements” in the beginning of a project, but that you must establish a requirements management process in parallel with processes for development and verification.

THE AUTHOR

Ingmar Ogren was graduated with an M SC in Electronics from the Royal University of Technology in Stockholm in 1966.

He then worked with the Swedish Defense Material Administration and various consulting companies until 1989 with systems engineering tasks in areas such as Communications, Aircraft and Command & Control.

He now chairs the board and is owning partner in two companies:

- Tofs which produces and markets the Tofs (Tool For Systems) software.
- Romet, which consults in the area of systems engineering methods with the method O4S (Objects For Systems) as its main product.

Further information about Ingmar Ogren can be found on the web page www.toolforsystems.com

Forskning och Teknikutveckling för Anpassning (FoTA) – aktuellt läge

Enligt Bror-Arne Ersson, FMV sammanhållande för FoT-programmet för anpassning, är det aktuella läget för programvaruteknikprojekten följande.

Beställningar till industrin (motsv) är lagda för

P. 4 Patterns och komponentåteranvändning

FMV handläggare: Christopher Bengtsson, FMV:INFOSYST, chben@fmv.se
Industrins kontaktperson: Barbro Sjöland, Sjöland och Thyselius Datakonsulter, barbro.sjöland@st.se

P. 7. COTS-produkter i militära lednings- och informationssystem

FMV handläggare: Jan Flodi,n FMV:ELEKTRO, jaflo@fmv.
Industrins kontaktperson: Bengt Gustafsson, begu@celsiustech.se

P. 10. Projekt MANA - Ett run-time system för säkerhetskritiska komplexa system
(Kompletteringsprojekt till NUTEK Komplexa Tekniska Systems projekt "A Run-Time System for Safety Critical Complex Systems")

FMV handläggare: Gunnar Fredriksson, FMV:FlygE, gufre@fmv.se
Industrins kontaktperson: Lars Asplund, asplund@Minsk.DoCS.UU.SE

Följande projekt är fortfarande i anbudsfas, men bör kunna gå i beställning i närtid:

P. 3. COTS och objektorientering som bas för konstruktion av realtidsapplikationer

FMV handläggare: Gustaf Myhrman, FMV:UVsystS, gumyh@fmv.se
Industrins kontaktperson: Lennart.Bie@emw.ericsson.se

P. 9. Experimentell verifiering av feltolerans

FMV handläggare: Gunnar Fredriksson, FMV:FlygE, gufre@fmv.se
Industrins kontaktperson: Håkan Edler, edler@hisafe.se

P. 11. Formalisering, analys och hantering av krav på säkerhetskritiska system

FMV handläggare: Per-Henrik Persson, VapenP, phper@fmv.se
Industrins kontaktperson: Göran Anger, anger@L4i.se

Betr det återstående projektet

P. 12. Överföring till industrin av programvaruteknik för säkerhetskritiska system

FMV handläggare: Inag-Lill Bratteby-Ribbing, FMV:InfosystU
Industrins kontaktperson: Dag.Folkesson@saab.se

som är ett slags uppsamlings- och kunskapsspridningsprojekt, avvaktas överenskommelse med deltagarna och med berörda andra projekt innan beställning kan läggas.

Följande meddelande från Inga-Lill beskriver status för P. 12 och planer för att komma vidare.

Status FoTA P12: Överföring till industrin av programvaruteknik för säkerhetskritiska system

Ett upprop om anmälan för företag som önskar deltaga i FoTA P12 gick före semestern ut bl a via R-V och mail. Projektets uppgift är att interagera dels med de huvudprojekt inom FoTA, som behandlar frågor av intresse vid utveckling av säkerhetskritisk programvara (dvs P3, P4, P7, P9, P10, P11), dels med det projekt som avser framställa en handbok för anskaffning av programvara i säkerhetskritiska tillämpningar (H ProgSäk). Syftet är ge P12:s deltagare möjlighet att påverka och ta del av resultat från huvudprojekten.

Bland anmälda till P12 finns nu (förutom sammanhållande från berörda huvudprojekt) Bofors Missiles, Celsius Aerotech, CelsiusTech Electronics, CelsiusTech Systems, Enator, FOA, Kockums, Romet, Saab, S&T.

För att underlätta samarbete och utbyte mellan projekten har gemensamt datum reserverats för träff under hösten, genom att 6 rum bokats på FMV, Tre Vapen, Banérg 62, Sthlm i B-husets konferenscenter. Tanken är att under en dag (kl 08-17) avsätta f.m. för enskilda projektmöten (där motsv FMV-handläggare är värd) och e.m. för gemensamma projektredovisningar (där P12 värd). Enskilt projektmöte kan även läggas in under e.m. på tider som ej avsatts för interaktion med P12. För varje e.m.-genomgång räcker det med en föredragande representant från huvudprojektet, men övriga är givetvis också välkomna. Kallelse, dagordning, anmälan, starttidpunkt etc. för f.m. sköts av resp projekt och för e.m. av P12.

Preliminär dagordning ons 20 Oktober**:**

09.00-11.30:	Enskilt projektmöte,		FMV/Tr/B/ rum: se nedan
11.30-12.30:	Lunch		
12.30-13.00:	H ProgSäk redovisning		FMV/Tr/B/ rum: ”Drottningholm”
13.00-13.30:	P3	-”-	-”-
13.30-14.00:	P4	-”-	-”-
14.00-14.30:	P7	-”-	-”-
14.30-15.00:	P9	-”-	-”-
15.30-16.00:	P10	-”-	-”-
16.00-16.30:	P11	-”-	-”-
16.30-17.00:	Sammanfattning, avslut		-”-

Förslag till rumsfördelning**:**

P3	(COTS-OO-Realtid)	FMV/Tr/B/ rum: ”Skånelaholm”	(12 platser)
P4	(Patterns & Åa)	FMV/Tr/B/ rum: ”Tullgarn”	(16)
P7	(COTS i militära system)	FMV/Tr/B/ rum: ”Steninge”	(12)
P9	(Feltolerans)	FMV/Tr/B/ rum: ”Skokloster”	(32)
P10	(Run-timesystemet MANA)	FMV/Tr/B/ rum: ”Ulriksdal”	(12)
P11	(Formaliserade krav)	FMV/Tr/B/ rum: ”Tyresö”	(12)
P12	(Kunskapsspridn SäkTeknik)	FMV/Tr/B/ rum: ”Drottningholm”	(50)

Notera att ovanstående är förslag, där ändringar kan bli aktuella. Även om inte alla projekt har kommit igång eller har behov av eget möte på f.m. förutsätts att varje projekt deltar med en föredragande under e.m.-programmet, eftersom denna del i alla händelser kommer att genomföras.

Förutom 99-10-20 har rum bokats tis 99-12-07. Vid avslut av det första mötet kan det vara lämpligt att besluta om behov föreligger av samordnade möten även det senare datumet.

Inga-Lill Bratteby-Ribbing
(FMV-handläggare P12), ilbra@fmv.se, tel 018-12 02 63, fax 018-12 02 72.

Guide to Software Engineering Body of Knowledge

En gemensam kommittee inom IEEE Computer Society och ACM, SWECC - Software Engineering Coordinating Committee håller på att ta fram en "Guide to Software Engineering Body of Knowledge", ett kompendium om de kunskapsområden en professionell programvarungenjör bör förstå. Handboken skall vara klar under nästa år, men ett utkast finns till påseende på http://www.swebok.org/.

Samma SWECC har f ö tagit fram ett nytt utkast till "Software Engineering Code of Ethics and Professional Practice" .

SWECC har också andra projekt i gång vilka kan studeras på http://computer.org/tab/swecc/.

IT-arkitekt 99; nästa kursstart i januari

Dataföreningen i Sverige driver en utbildning till certifierad IT-arkitekt. Det är en terminskurs med 6 avsnitt över 100 timmar som har visat sig mycket efterfrågad.

Den första kursen gick under våren och var fulltecknad. Den andra kursomgången startade 15 september, också fulltecknad. Den tredje omgången startar den 4 oktober, även den fulltecknad. Nästa kursstart blir planeras till januari.

Kursen behandlar ett antal högaktuella ämnen som torde vara av stort intresse även för många av SESAMs medlemmar. Vidare information och anmälningsblankett finns på Dataföreningens hemsida www.sto.dfs.se.

Du besöker väl vår websajt?

http://sesam.tranet.fmv.se

Välkomna till SESAM-seminariet 21-22 oktober

SESAM-seminariet i höst har temat "Säkerhetskritiska system", men inrymmer också projektredovisningar från arbetsgruppernas miniprojekt.

Tid: 21-22 okt 1999

Plats: Gällöfsta Konferens & Herrgård, Kungsängen

Bidrag till Rendezvous och SESAM hemsida efterlyses

Varken Rendezvous eller hemsidan blir bättre än vad medlemmarna gör den till. Håll ögon och öron öppna för intressanta händelser i eller nära SESAM-världen och skicka in en notis till sekretariatet, alkas@tranet.fmv.se. När det är aktuellt att kunna publicera rapporter eller beskrivningar från projekt som Ni är inblandade i, glöm då inte våra egna medier.

SESAM-Sekretariatet: Anna Kåsjö, FMV:INFOSYST, 115 88 STOCKHOLM
Tel: 08-782 6745, Fax: 08-66 77 392
Epost: alkas@tranet.fmv.se