NYHETSBLAD FÖR SESAM
Försvarssektorns Adaintressenters Användargrupp för Software Engineering

RENDEZVOUS

Nr 3 oktober 2002

Innehåll

Försvarssektorns Adaintressenters Användargrupp för Software Engineering

SESAM

Vad är SESAM?

SESAM är ett samverkansnätverk för projektövergripande och företagsneutral kunskapsuppbyggnad och kunskapsspridning inom området programvaruintensiva försvarssystem.

SESAM skall genom organiserat samarbete mellan dess medlemmar (företag, organisationer, myndigheter och utbildningsinstitutioner) främja tillförlitlighet och effektivitet i utveckling och vidmakthållande av programvaruintensiva försvarssystem.

SESAMs verksamhet att samla, skapa och sprida information och kunskap sker huvudsakligen i arbetsgrupper som verkar inom avgränsade tekniska områden och som efter behov inrättas och avvecklas.

SESAM anpassar, profilerar och förnyar sin verksamhet med hänsyn till ändrade tekniska och andra omständigheter av betydelse för intresseområdet.

När SESAM startade sin verksamhet 1988 var s k inbyggda realtidssystem (i farkoster, sensorer, stridsledningssystem m fl) dominerande inom försvaret och svensk försvarsindustri. Det fanns ett allmänt behov att kunna driva utvecklingen av programvara för dessa alltmer komplexa system på ett mer organiserat sätt, d v s att mer konsekvent praktisera vad som börjat benämnas Software Engineering. SESAMs verksamhet inriktades därför från början på användningen av det speciellt för inbyggda och realtidssystem under början av 80-talet i brett internationellt samarbete framtagna och av ISO 1987 standardiserade programspråket Ada, vilket hade utformats särskilt för att stödja tillämpning av Software Engineering principer. Den fortsatta användningen av Ada, inkl i nya (Ada 95) och framtida versioner, är fortfarande av stort intresse och betydelse för många av medlemmarna i SESAM.

Efterhand har även andra typer av programvarusystem blivit mer förekommande inom försvarssektorn, t ex på informationssystemområdet. Utvecklingen inom kommunikationstekniken, t ex via Internet snabba expansion, har också tillfört nya möjligheter och typer av problemställningar för programvaruutvecklingen. Utbudet av kommersiellt tillgänglig programvara (COTS) som man önskar kunna använda i försvarssystem har också ökat. Inriktningen på uppbyggnad av ett nätverksbaserat försvar, behovet av interoperabilitet vid internationella insatser m m har aktualiserat nya problemställningar i systemutformningen. Denna utveckling gör att nya tekniker, processer och metoder, språk och andra hjälpmedel etc för framtagning och vidmakthållande av programvaruintensiva system av intresse att behandla inom SESAMs verksamhet ständigt tillkommer.

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 Claes Wadsten, AerotechTelub AB , tel 013-231652 .

VU

Andersson Tommy, Ericsson Microwave Systems AB,
tommy.andersson@emw.ericsson.se

Bengtsson Christopher, FMV
christopher.bengtsson@fmv.se

Carlsson Ingemar, adjungerad
ingemar.carlsson@mbox2.swipnet.se

Ekman Mats, Saab Aerospace AB
mats.ekman@saab.se

Gustafsson Bengt, SaabTech Systems AB
bbg@systems.saab.se

Hallén Johan, FMV
johan.hallen@fmv.se

Merkell Curt, Saab Bofors Dynamics AB
curt.merkell@dynamics.saab.se

Wadsten Claes, AerotechTelub AB
claes.wadsten@aerotechtelub.se

Arbetet utförs i arbetsgrupper och följande är f n organiserade:

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

Ag Teknik
Lars Asplund, Mälardalens högskola
lars.asplund@mdh.se

Vilka kan vara med i SESAM?

Medlemmarna i SESAM är företag, organisationer och myndigheter (förvaltningar, utbildningsinstitutioner etc) i Sverige 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- eller arbetsgruppsmedlem krävs status som leverantör till FMV eller FM. Dessutom krävs en skriftlig förbindelse att uppfylla åtagande som ordinarie- resp arbetsgruppsmedlem. För inträde som informationsmedlem (erhåller endast informationsbladet) krävs status som leverantör till FMV eller FM 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
AerotechTelub AB
c/o Kåsjös Kontor
Ytterspåret 14
187 54 TÄBY

Ordföranden har ordet

Som ordförande i SESAM är det ett nöje att se hur verksamheten nu tagit fart och alla pusselbitar kommit på plats.

Vi har nu fått det ”långsiktiga” stöd från försvarsmakten genom FMV som vi tidigare haft. Och VU kan därför ägna sig att verksamhetsfrågor inom SESAM. Jag vill därför tacka de inom FMV och FM Hkv som hjälpt till med att få denna lösning och hoppas att SESAM skall uppfylla alla förväntningar i framtiden, genom att belysa frågor inom programutveckling och datorsystem för svenska försvaret och leverantörer av dessa som är medlemmar.

Arbetsgrupperna Metodik och Teknik har många intressanta projekt vars resultat och redovisningar kommer medlemmarna till godo. Viss redovisning kommer att göras i samband med höstseminariet. Temat för årets höstseminarium ”Modellbaserad utveckling” ligger också detta i tiden med vad som är aktuellt för nya stora projekt inom svenska försvaret.

Den temadag kring Arkitektur som SESAM och FMV tillsammans genomförde blev så populär att vi var tvungna att byta lokal, för att få rum med alla intresserade. Vår Ingmar Carlsson samt FMV:s Johan Bendz skall tillsammans med vår Anna ha all ära att seminariet uppfyllde allas förväntan.

Om fortsättningen på året blir av samma kvalité så får jag som ordförande känna mig mycket nöjd med år 2002. Jag har stora förväntningar på resterade tid av 2002 samt kommande verksamhet under 2003.

Jag vill redan nu nämna att vi kommer att belysa ”programvarusäkerhet” på ett mer aktivt och fokuserat sätt. Redan i samband med höstseminariet kommer vi att starta en intressegrupp för detta under ledning av Inga-Lill Bratteby-Ribbing. Med Inga-Lills bakgrund för dessa typer av frågor så vet jag att mycket ”matnyttigt” kommer att behandlas. Framtida militära system som är uppbyggda som system av system kräver goda kunskaper kring detta.

Claes Wadsten
Ordförande i SESAM

Tfn. dir : 013-23 16 52
Mobil: 070-6000271
E-post: claes.wadsten@aerotechtelub.se

UML för systemarbete

Som har framgått av tidigare artiklar i RV pågår en verksamhet inom OMG för att utvidga UML mot att bli mer anpassat för systems engineering. Ingmar Ögren har haft en del kontakter med detta och redovisar här i tre artiklar dels sin egen syn på ämnet, dels en del av vad som diskuterats i OMGs arbetsgrupp den senaste tiden.

Vidareutveckling av UML för systemarbete

Ingmar Ögren, iog@toolforsystems.com

För information på Internet se http://syseng.omg.org Denna artikel är framför allt baserad på informationen från OMG hemsida där man bl. a. hittar en utmärkt introduktion av ordföranden i gruppen, Sanford Friedenthal från Lockheed Martin (sanford.friedenthal@lmco.com).

Varför utvidga UML

UML (Unified Modeling Language) har nu i praktiken blivit gällande standard för att modellera programvara. Det är då en naturlig tanke att vidareutveckla modelleringsspråket för att hantera, inte bara programvara, utan också kompletta system med maskinvara och personal.

I dagsläget upplever man ett antal problem vad gäller samverkan mellan programvaruarbete och systemarbete:

- Det saknas en gemensam notation/semantik och verktygen stämmer inte med varandra
- Systemmodeller behöver definieras på nytt inför objektorienterad utveckling av programvara
- Krav definieras ofta inte tillräckligt
- Krav förändras
- Ofta införs alltför (onödigt) stora konstruktionsbegränsningar
- Relevant information, med t. ex. scenarios saknas ofta för programvaruarbetet.

Arbetet förutsätter att man utgår från en modellbaserad ansats på grund av flera skäl:

- Förbättrad kommunikation
- Minskade motsägelser
- Färre fel
- Mer komplett representation
- Bättre förutsättningar att fånga befintlig kunskap.

Vad innebär arbetet

Arbetet med utvidgning av UML fokuserar mot följande mål:

- Tillhandahålla ett standardiserat modelleringsspråk för att specificera, konstruera och verifiera komplexa system
- Underlätta integration av disciplinerna systemarbete och programvaruarbete.
- Stödja stabilitet i överföring av information mellan discipliner och verktyg för att utveckla system.

Arbetet innebär bland annat att man går igenom bristerna i nuvarande UML ur systemarbetssynpunkt för att sedan finna åtgärder för att eliminera bristerna. En del av de observerade bristerna avser hantering av

- Kontinuerligt beteende i reell tid
- Icke-datoriserade fysiska företeelser (t. ex. maskiner, energi)
- Fysiska gränssnitt
- Flöden in och ut till/från system
- Prestanda och karakteristika, som inte har med beteende att göra
- Parametriska relationer och matematiska ekvationer
- Hierarkisk modellering av scenarios och beteende
- Krav
- Geometri.

Hur gör man

Arbetet görs dels som en dialog på Internet (http://syseng.omg.org), dels som en serie möten där det senaste var i Orlando i Florida och där nästa möte är i Helsingfors i september. För att komma med på mötena anmäler man sig på hemsidan.

En väsentlig del av arbetet var att begära in information från dem som har erfarenhet av att tillämpa UML för systemarbete. Flertalet svar hade kommit in i maj i år och alla svar skall vara slutgiltigt uppdaterade i augusti. Erfarenheterna, som redovisas i svaren utgör grundläggande underlag för mötena i Orlando och Helsingfors.

Ett möjligt problem med detta arbete är att man förutsätter att resultatet skall bli en utvidgning av UML, något som inger viss oro om man betänker att:

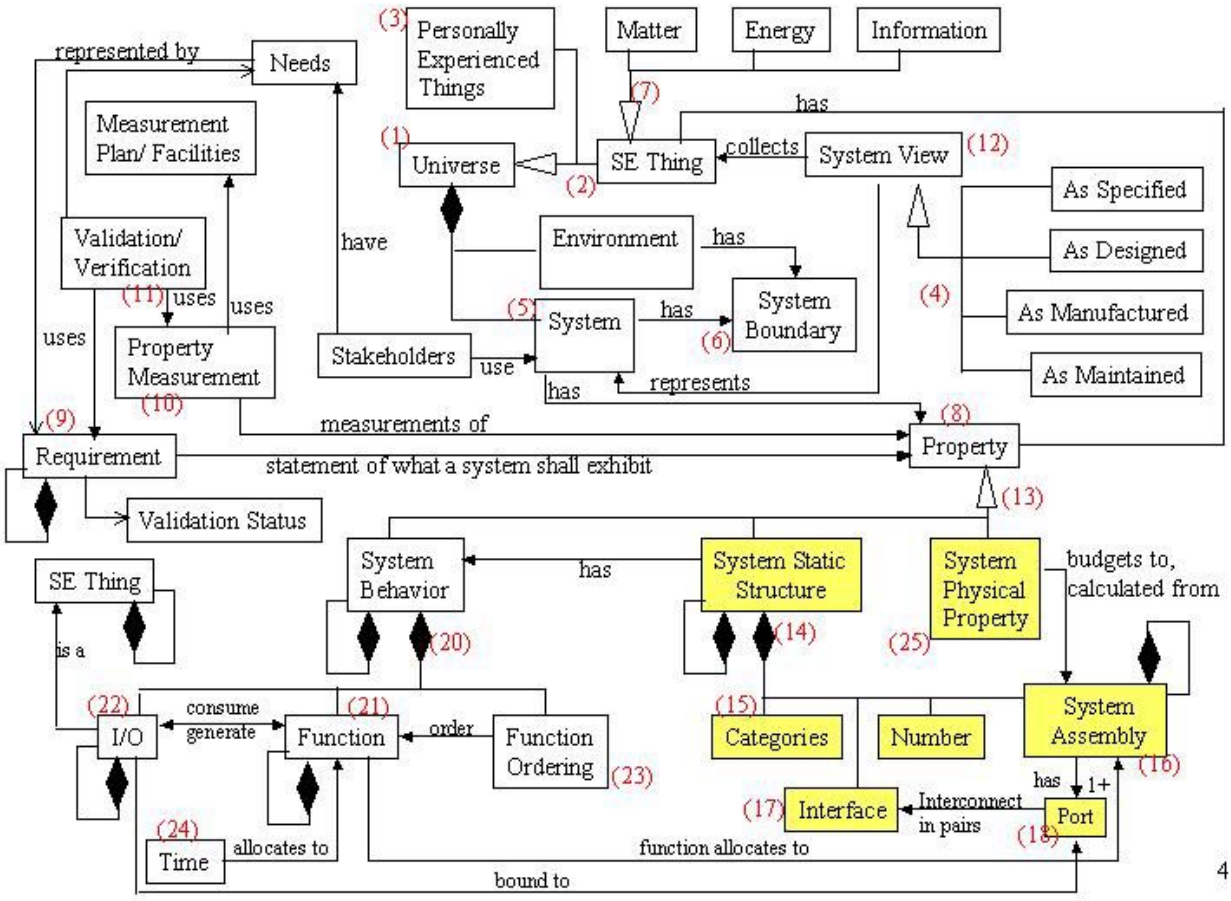
- Många upplever redan nuvarande UML som komplext och svårt att tränga in i
- Ett utvidgat UML riskerar att bli mera komplext
- Vid arbete med komplexa system måste man ofta arbeta med personer, som har olika speciaiteter och litet intresse och föga tid för att sätta sig in i ett komplext modelleringsspråk
- En systemmodell är av litet värde om inte alla intressenter begriper den.

Vart har man kommit

En viktig del av arbetet är att klargöra begreppen, vilka de är och hur de hänger ihop. Det mesta arbetet här har gjorts av David Oliver som har uttryckt resultatet i ett UML klassdiagram, se nedan.

Till klassdiagrammet finns även en begreppskatalog som förklarar de olika begreppen.

En annan huvudpunkt är att man vid mötet i Orlando har haft presentationer från flertalet av dem som svarat på enkäten i egenskap av användare, verktygsleverantörer, utbildare etc. (Presentationerna finns tillgängliga på hemsidan). Det framgår av presentationerna att man har ganska olika syn på var huvudproblemet ligger. Där finns presentationer från detaljer om de matematiska sambanden för samspel mellan maskinvara och programvara till diskussioner om problem med hur man får in-



Figur 1 Klassdiagram med begrepp UML i systemtillämpning

tressenter utan programmerarbakgrund att begripa UML.

Ytterligare en huvudpunkt är att man gick igenom kravdokumentet och uppdaterade. Även en preliminär version av kravdokumentet finns tillgänglig på hemsidan.

Vad händer härnäst

Detaljerna för det kommande mötet Helsingfors i slutet av september är ännu inte tillgängliga, men kommer att dyka upp hemsidan. Kända punkter är att Volvo och Tofs kommer att redovisa sina erfarenheter.

För egen del kommer jag att koncentrera presentationen på behovet att förenkla UML för att säkerställa intressenternas förståelse och behovet att visualisera ett systems ”djupa beroendestruktur” på ett bättre sätt än vad som kan göras med klassdiagram genom att utveckla komponentdiagrammet. Även Tofs presentation finns tillgänglig på hemsidan eftersom det var meningen att den skulle föredragit på telefon i Orlando, men uppringningen fungerade inte.

. . . . . .

UML för systemarbete, erfarenheter och förslag

I samband med utvecklingen av UML mot användning, inte bara för programvara utan för system i allmänhet, inbjöds intresserade att komma in med erfarenheter och förslag till OMG (Object Management Group). Jag gjorde så och fick tillfälle att presentera mina erfarenheter och förslag vid gruppens möte i Helsingfors i oktober. En första reaktion från gruppen var ”Detta är ett helt annorlunda sätt att hantera UML”.

Denna artikel är ett referat av väsentliga delar av presentationen plus mina synpunkter på hur man bör åtgärda de brister i UML, som observerats i arbetsgruppen.

Erfarenheter att utnyttja

Från arbetet med FM system sedan Stril 60 och flygplan 35, tycker jag mig ha samlat en hel del erfarenheter, som är värt att gå igenom inför uppdatering av UML till ett modelleringsspråk för system:

Intressenternas acceptans

I början av 70-talet var jag projektledare för ett system som skulle förbättra störtåligheten för stridsledningsradion inom Stril 60. Ett prototypsystem byggdes i södra Sverige, provades ut och dokumenterades. Den tekniska dokumentationen visade klart att försöket hade lyckats med förbättrad tålighet mot radiostörning. Systemet blev aldrig utbyggt, troligen beroende på att användarna aldrig satte sig in i den tekniska dokumentationen så långt att de fullt förstod problemet och dess lösning.

Slutsats: System måste modelleras så att användarna förstår både problemet och hur det är löst.

Utgå från systemets uppgift(er) för dess modellering

Under 90-talet var jag inblandad i två projekt, dels en förstudie för ubåtssystem, dels Saabs civila projektförslag ”Baltic Watch”. I bägge systemen hade man börjat med att ansätta ett antal tekniska lösningar och alla berörda förutsattes veta vad uppgiften skalle vara. Mitt viktigaste bidrag (tyckte jag) var att gå igenom, klarlägga och dokumentera vilka uppgifter som systemet skulle lösa och hur de olika delarna i systemet skulle bidra till att lösa uppgifterna.

Slutsats: Det är viktigt för systemförståelsen att system modelleras så att alla blir överens om vilka uppgifter som skall lösas.

Användning av formell engelska för beteendemodellering

Under 80-talet användes det Ada-baserade modelleringsspråket ”Adel” (formell engelska) bl. a. för att modellera simulatorer för kärnkraftverk. En reaktion från kärnkraftsfolket var:

”Var det så här enkelt med programvara”.

Slutsats: Formell engelska är ett bra val när man vill få med både användarförståelse och formalism i beteendemodellering.

Trädgrafer för samverkan i modellering

När Saabs simulator för Presentation och manöver för flygplan JAS 39 Gripen skulle modelleras, användes bl. a. ”Trädgrafer” där de olika systemelementen ordnades efter hur de bidrog till systemets uppgifter. Trädgraferna, presenterade på OH, visade sig ge ett bra underlag för diskussion och optimering av systemets struktur.

Slutsats: Trädgrafer, härledda från UML komponentdiagram är ett bra underlag för gruppdiskussioner för att utveckla och optimera ett systems struktur.

Krav på modelleringsspråk

Bland annat erfarenheterna, som redovisas ovan leder till ett antal krav på ett modelleringsspråk för att arbeta med komplexa system:

- Språket bör göra det möjligt att modellera utgående från ett systems uppgift(er) och täcka hela arbetsrummet för systemarbete, dvs:
  - o stödja olika aktiviteter som kravarbete, konstruktion och verifiering med provning
  - o ge möjlighet att modellera olika objektkategorier som operatörsroller (manuella delar), programvarudelar och maskinella delar (även annat än datorer)
  - o stödja ”djupa strukturella beskrivningar”, som visualiserar hur olika systemelement beror av varandra för att genomföra systemets uppgifter
- Språket skall vara begripligt för icke-tekniker bland systemets intressenter
- Språket skall ge en dokumentation som är användbar för kommunikation, både över tiden och mellan olika intressenter.
- Språket bör ha stöd av programvaruverktyg
- Språket bör vara standardiserat.

För att uppfylla krav finns det (minst) två möjliga huvudvägar:

1. Utvidga nuvarande UML för att täcka kraven. Risken med en sådan utvidgning är att språket växer så att det blir svårhanterligt och att man får systembeskrivningar som blir svåra att förklara för intressenter, som inte har en djupare kunskap om UML
2. Använd befintligt UML, men tillåt ”tolkningar” och komplementära notationer. Risken med denna ansats är arbetssättet upplevs som en ”icke renlärig” användning av UML och att de vanligaste UML-verktygen ger ett dåligt stöd. Detta är ändå, enligt min åsikt, den mest framkomliga vägen.

Några problem med dagens UML

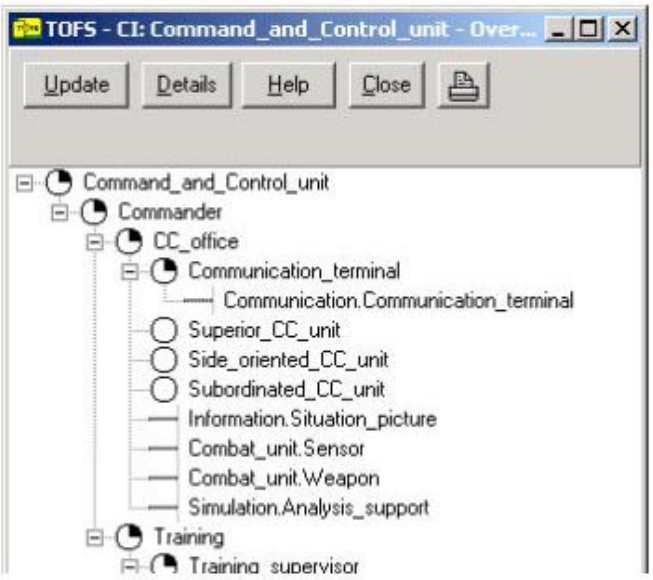
Om man betraktar dagens UML ur systemmodelleringssynpunkt så visar sig en del problem, bland annat:

- Modellering av systemets uppgifter saknas helt liksom av operatörsbeteende medan maskiner är begränsade till datorer i ”deployment diagram”.
- Användningsfallen är ”svaga” eftersom de placerar användaren utanför systemet och saknar information om användarens beteenderymd i relation till systemet.
- Även om det går att modellera djupstrukturer, med flera nivåer i klassdiagram och komponentdiagram så får man ingen tydlig visualisering med dessa diagram.
- Digrammen kan lätt upplevas som komplexa och svårbegripliga för intressenter, som saknar utbildning i UML.

Vidgad användning av komponentdiagrammet

Ett sätt att komma tillrätta med en del av problemen, i nuvarande UML, är att vidga användningen av komponentdiagrammet genom att:

- tillåta objekt av kategorierna Uppgift, Operatör(sroll) och Maskinvara, förutom programvara
- tillåta en alternativ vy, med en ”Trädgraf” där man skiljer mellan eget system och stödjande system.



Bilden visar en trädvy för en del av ett ledningssystem. Trädvyn är härledd från UML komponentdiagram genom att objekten är ordnade efter hur de beror av varandra. På varje rad finns ett objekt och varje objekt beror av underliggande objekt, som är indenterade ett steg. Observera i trädvyn:

- Överst finns ett uppgiftsobjekt (Command and Control Unit), som definierar systemets uppgift. På svenska skulle objektet kallats ”Ledning” eller ”Ledningsenhet” för att definiera den övergripande uppgiften.
- Uppgiftsobjektet stöds av operatörsobjekt (Commander), som i sin tur stöds av programvaruobjekt (CC_office), som i sin tur stöds av maskinvaruobjekt (Communication_terminal).
- Objekt, som ingår i det aktuella systemet har en ”liten klocka” som visar objektets utvecklingsläge.
- Stödjande objekt i externa system saknar ”klocka” och har ett namn av typen Systemnamn.objektnamn (Communication.Communication_terminal).

Genom att tillåta detta alternativa synsätt på komponentdiagrammet åtgärdas flera av problemen med dagens UML för systemarbete utan att man egentligen utvidgar språket.

Synpunkter på de olika diagramtyperna

De många diagramtyperna och den rika syntaxen i UML kan ge förståelseproblem för systemintressenter som saknar utbildning i UML. Man måste alltså begränsa sig. Nedan är en genomgång med en diskussion om vilka diagram som (enligt min erfarenhet) är nödvändiga, mindre nödvändiga och direkt olämpliga.

Användningsfallsdiagram

Om man accepterar att komponentdiagrammet kan innefatta operatörsroller med separat beskrivet beteende så kan man beskriva användningsfall utan något separat diagram och dessutom mera fullständigt. Eftersom användningsfallsdiagrammet är baserat på ett synsätt där användaren ses som utanför systemet och eftersom det ger mycket lite information om användarens avsedda beteenderymd, kan diagrammet bli direkt olämpligt.

Klassdiagram

Med klassdiagram kan man uttrycka det mesta i en mycket rik syntax. Detta gör det lätt att åstadkomma komplexa och svårbegripliga diagram. Klassdiagram bör utnyttjas försiktigt, granskas noga och ha gott om kommentarer.

Meddelandesekvensdiagram

En mycket bra diagramtyp för att visa samspelet mellan parallella processer. Man bör dock se upp med om man har återkoppling till en användare (operatörsroll) och säkerställa att diagrammet får med hela meddelandesekvensen, inklusive återkopplingen.

Samarbetsdiagram

Detta tycker jag egentligen bara är en alternativ syntax till meddelandesekvensdiagrammet. Således inte stor anledning att använda om man inte råkar sitta i en organisation som använder detta diagram av tradition.

Paketdiagram

Egentligen bara en variant av komponentdiagrammet, varför man kan klara sig med komponentdiagrammet.

Tillståndsdiagram och aktivitetsdiagram

Dessa två diagram, som modellerar beteende kan vara svåra att begripa och behöver kompletteras om man vill få in formalism, t. ex. för att definiera typer för variabler. Kan ersättas av ”formell engelska” för beteendebeskrivningar, som då blir både lättare att begripa och mera formella.

Fördelningsdiagram

Om man tillåter att komponentdiagrammet innehåller maskinvaruobjekt, så blir detta diagram onödigt.

Komponentsdiagram

Som diskuterats ovan ger detta diagram en användbar grund för att utveckla ett diagram som kan beskriva strukturen för komplexa system och även ”system av system”.

Min slutsats blir alltså att man i första hand skall använda meddelandesekvensdiagram och komponentdiagram, kompletterade med formell engelska.

Att hantera krav, strukturer, kommunikation och beteende

Vad gäller hantering av krav kan man ibland få uppfattningen att kravarbete är en ”fas”, som man kan avverka före det egentliga utvecklingsarbetet. Så är det inte, även om det förvisso är en bra idé att arbeta igenom sina krav ett varv innan projektarbetet startar. Med den föreslagna anpassningen av UML för systemarbete kan vi arbeta med krav på flera nivåer:

- Uppgiftsobjekten definierar krav på högsta nivån. Systemet skall kunna genomföra de definierade uppgifterna. Verifiering kan man förbereda genom att koppla scenarion för verifiering till uppgiftsobjekten som attribut.
- Eftersom vi arbetar med en uppdelning av systemet i komponenter så kan vanliga skrivna krav komma in naturligt, som attribut till komponenterna.
- Beteendebeskrivningar, uttryckta i ”formell engelska” ger detaljerade krav för implementering av systemet.

För att krav skall vara meningsfulla krävs att man kan verifiera uppfyllande av krav (prova). Detta kräver provfall, som också kan hanteras som attribut till objekten. För varje objekt får man då ett många-till-många förhållande mellan krav och provfall.

För strukturbeskrivningar är det min erfarenhet att det är bra om man kan frigöra sig från alltför ”datamässigt” tänkande med anrop, parametrar etc. Ett sätt att göra detta är att ta fasta på komponentdiagrammet och ordna komponenterna primärt efter hur de beror av varandra för att leverera sina tjänster. ”Trädvyn” blir då ett bra komplement till standardvyn för komponentdiagrammet föra att visa ”djupa beroendestrukturer”.

Kommunikation mellan systemelement (objekt, klasser) handlar dels om anrop som visas med klass- och komponentdiagram, dels om informationsutbyte via meddelanden, som visas med meddelandesekvensdiagram. Om man utnyttjar ”formell engelska”, som komplement till diagrammen, så får man på ett naturligt sätt med information om både kommunikation och gränssnitt.

Beteende inom systemelement (objekt, klasser) kan, som diskuterats ovan, uttryckas i ”formell engelska”. Det innebär att man arbetar med ett programspråksliknande modelleringsspråk, som innehåller:

- reserverade ord som if-then-else, case, loop etc.
- variabler av definierade typer
- kommentarer.

Slutligen

Åtskilliga års arbete med system och med UML har lett mig till uppfattningen att de två allvarligaste problemen med UML, för systemarbete i allmänhet, är att:

- de många och komplexa diagrammen medför en risk att användare inte begriper systembeskrivningar
- inget av diagrammen klarar att visualisera ”djupa beroendestrukturer”, något som erfarenhetsmässigt är synnerligen användbart vid arbete med komplexa system.

En beprövad lösning är att begränsa användningen av diagram till komponentdiagrammet och meddelandesekvensdiagrammet, men med ett vidgat synsätt på komponentdiagrammet och en kompletterande syntax, som ger möjlighet att visa ”djupa beroendeträd”: Dessutom att komplettera diagrammen med ”formell engelska” för att beskriva beteende. Dessa åtgärder leder en god bit på vägen mot användning av UML för komplexa system i allmänhet.

Ingmar Ögren

Möte i Helsingfors den 30 september till 1 oktober 2002 med OMG SE-DSIG

OMG är Object Management Group
SE DSIG är Systems Engineering Domain Special Interest Group

Arbetet med att utveckla UML för systemarbete fortgår. Protokoll från det senaste mötet, tillsammans med de presentationer som gjordes under mötet finns nu tillgängligt på http://syseng.omg.org

Gruppens ordförande tar gärna emot synpunkter och det går också bra att anmäla intresse om man vill delta i arbetet:

Sanford Friedenthal, Lockheed Martin Corporation, SE DSIG Chair
(703) 293-5557, sanford.friredenthal@lmco.com

Nedan följer ett referat från protokollet, sammanställt av Ingmar Ögren

Mål för mötet

- Granska återstående svar från RFI (Request For Information)
- Fortsätt samordning med dem som lämnat underlag till UML V2.0

Deltagare

Jacob Axelsson (telefon), Volvo
Ed Barkmeyer, NIST
Ian Cornwell, Mott MacDonald
Bruce Douglas, Ilogix
Harald Eisenmann, Astrium GLSH
Phil Eyermann, Raytheon
Sanford Friedenthal, Lockheed Martin Corporation
Eran Gery, Ilogix
Shimamura Masayoshi , Fujitsu
Traci McDonald, NSWCDD
Steve Mellor, Project Technology
Trevor Moore, Holistic Systems Engineering
Ingmar Ogren (telefon), TOFS AB
Tom Rugg, Fujitsu

Introduktion

Sandy Friedenthal (ordförande) öppnade mötet och gick kortfattat igenom målsättningar och status för gruppens aktiviteter.

Tofs AB RFI-svar

Ingmar Ögren från Tofs AB presenterade en sammanfattning av sitt svar. Ingmar har en bred erfarenhet från tillämpning av UML i systemarbete med ledningssystem och andra komplexa system. Ingmar poängterade nödvändigheten för enkelhet i process och språk och att säkerställa att alla intressenter medverkar genom effektiv kommunikation. Hans ansats är att beskriva beteende med en trädstruktur och en version av UML komponentdiagram, som beskriver beroenden från systemets uppgifter och ner till implementationsnivån för konstruktion. Som komplement till denna beskrivning använder han strukturerad engelska och har utvecklat verktygssviten Tofs för att stödja denna ansats.

Under uppföljningsdiskussionen framkom att samverkanstabellen kan ge en liknande ansats som strukturerad engelska för att fånga ”djupberoendet” för detaljerade samverkansfragment.

Holistic Systems Engineering RFI-svar

Trevor Moore från Holistic Systems Enginering presenterade sitt RFI-svar. Trevor har tillämpat UML för specificering och konstruktion av telekomsystem, inklusive två projekt som handlar dels om ersättning av ett Ethernet-kort med en IC, dels om komplex tillämpning med en växel för ATM. Han har använt RUP och Rose Real Time. Han har också använt System C för att utveckla en underliggande exekveringsmaskin för UML-modellerna. System C tillhandahöll portar/kanaler och ett effektivt sätt att hantera parallellitet i reell tid. Några av problemen med UML, som Trevor mötte, var behovet av utvidgning av meddelandesekvensdiagrammen och mera direkt inläggning av användningsfall i dessa diagram. Trevor använde inläggning av användningsfall för representera olika tjänster, som motsvarade OSI-lagren (transport, länk fysiskt). Han poängterade av ett kontextdiagram för

visa en statisk representation av data och händelser, som används av systemet, tillsammans med en logisk konstruktion med klassdiagrammet och en fysisk konstruktion med komponentdiagrammet.

Trevor lämnade också detaljerat underlag för kravanalys till SE UML V0.3. Hans underlag gicks igenom i detalj och det överenskoms att Trevor skall leverera en uppdatering av sitt dokument, baserat på diskussionen.

Volvo RFI-svar

Jacob Axelsson från Volvo presenterade en sammanfattning av sitt RFI-svar. Jacob leder forskningen för elektronik och programvara vid Volvo. Jacob gav en detaljerad sammanfattning av sitt arbete i november-utgåvan av INCOSE Systems Engineering Journal avseende utvidgning av UML för att hantera kontinuerlig tid. Volvo har tillämpat UML för flera projekt, inklusive 2002 SUV-XC90 bil. Bilar innehåller nu upp till tre dussin datorer med några miljoner rader kod. Jacob tryckte på nödvändigheten att kunna beskriva ekvationer, som ger samband mellan objektattribut som flöde, tryck etc. Hans ansats använder UML ”constraints” för att hantera dessa ekvationer. Jacob tryckte också på nödvändigheten att modellera fysiska storheter i reell tid som massflöde och tryck. Bruce Douglas, en UML-expert, indikerade att det nog inte finns några begränsningar i UML vad gäller hantering av kontinuerlig tid, fast det kan vara nödvändigt att göra hanteringen av kontinuerlig tid mera tydlig. Jacob tryckte också på behovet för UML att stödja avvägningar för sådant som bränsleåtgång och säkerhetsanalys.

Sandy presenterade en sammanfattning av Russel Peaks tidigare RFI-svar som presenterades i Orlando i juni, för att utvärdera relationen mellan Jacobs och Russels arbeten. Både Jacob och Russel visar ansatser för att beskriva parametriska relationer mellan attribut, men deras ansatser är olika. Det framkom att bägge ansatserna styrker behovet av och ger möjliga lösningar för att utveckla en parameterisk modell i UML, som kan integreras med de mera detaljerade analytiska modellerna för sådant som prestanda, tillförlitlighet och fysiska modeller.

Project Technology RFI-svar

Steve Mellor, chef för Project Technology och tidigare medverkande i utvecklingen av Ward-Mellor-metoden för programvaruutveckling, granskade kritiska behov för ett systemmodelleringsspråk i sitt RFI-svar. Steve tryckte på behovet att utveckla UML med en tillräcklig semantik så att man kan exekvera en funktionell konstruktion och stödja fördelning på en fysisk konstruktion (exekverbar och översättningsbar UML). Ett av de kritiska behoven är att säkerställa en konsistent semantik mellan de olika beteendediagrammen (t. ex. aktivitet, sekvens och tillståndsdiagram).

Sammanfattning av RFI-svaren

Efter genomgång av dessa fyra RFI-svar, har gruppen gått igenom alla 13 svaren. De kompletta svaren är tillgängliga via RFI-länken från SE-DSIG websida (kräver OMG medlemsaccess). RFI-svaren har uppfyllt de uppsatta målen. Gruppen har fått ett brett spektrum av svar från slutanvändare, verktygsleverantörer och forskare. Baserat på den inkomna informationen, är det tydligt att användningen av UML, för att stödja systemmodellering, har ökat under de senaste åren och att tekniken fortsätter att mogna. Svaren har givit extremt värdefulla indata för att bättre förstå hur UML används i systemarbete och vad som är fördelarna, problemen och möjliga lösningar. Svaren ger utmärkt indata till kravbilden för SE UML och ger ett bra underlag till arbetsläget inom området. Gruppen tackar alla som lämnat bidrag för deras medverkan.

Samordning med UML 2.0

Gruppen granskade de förslag, som inkommit avseende UML 2 ”Superstructure” som del av ett möte tillsammans med Analysis and Design Task Force (ADTF) under september. Superstrukturen avser att tillhandahålla modelleringsdiagrammen, medan infrastrukturen tillhandahåller de underliggande konstruktionerna, som diagrammen sammanställs från. Datum för slutliga bidrag till UML 2 superstruktur är planerat till mötet i januari 2003.

Vid mötet med ADTF, bad Sandy att intresserade, som lämnat underlag till UML 2 att presentera en sammanfattning av sina förslag för superstruktur vid nästkommande UML tekniska möte i Washington DC. UP2-gruppen har bekräftat detta och presentationerna kommer att läggas in i dagordningen för informationsdagen vid mötet i Washington DC.

Samordningsmöten har hållits med UP2-gruppen som uppföljning till workshopen i september (se länk till ”workshop summary” på SE DSIG websida). Mötet avsåg även att diskutera och jämföra formalism för beteendebeskrivningar mellan UP2-ansatsen och systemarbetsansatsen för AP-

233. Vissa nyckelpersoner kunde dock inte medverka varför tiden planeras om för mötet. Man har dock kunnat gå igenom de specifika frågorna vad gäller systemarbete från workshopen så långt att man har en gemensam förståelse. Medverkande inkluderade Sanford Friedenthal och medlemmar av U2P-gruppen med C. Kobryn (Telelogic), B. Selic (Rational), E. Gery (Ilogix), samt med senare diskussioner med Bruce Douglas (Ilogix). Man granskade några av representationerna för systemmodellering, inklusive ett papper från Jim Long som summerar många av de traditionella representationerna för modellering av system (IDEF0, FFBD, Beteendediagram, N2 diagram, etc). Man granskade även UML-relaterade utvidgningar, inklusive ”Elaborated Context Diagram (ECD) från OOSEM och ”Object Constraint Graph” från Russel Peaks arbete.

Övriga ärenden

En utbildning i EDOC (Enterprise Distributed Object Computing) skulle ha gjorts men har utgått p.g.a. planeringsproblem. Presentationen finns inlagd på hemsidan.

Bidrag presenterades som avsåg realtid, kvalitetstjänster och feltoleransprofiler. Några av deltagarna granskade ett av bidragen tillsammans med Alan Moore.

Det fanns flera andra intressanta presentationer tillgängliga bl. a. en genomgång av status för XMI som skall översätta UML-modeller till XML. Dessutom, förslagen till standard för utbyte mellan diagramrepresentationer granskades kortfattat, inklusive layoutinformation så att man enklare kan dela modeller mellan modelleringsverktyg och andra verktyg som MS Word.

Förberedelser planerades för en SE informationsdag.

Nästa möte

Nästa tekniska OMG-möte är planerat i Washington DC, tillsammans med SE informationsdag den 18-22 november. Informationsdagen innebär att resultaten från SE DSIG och relaterade UML-aktiviteter, som berör systemarbete, presenteras. Man kan registrera sig, antingen för hela mötet eller enbart för informationsdagen på SE DSIG hemsida.

Måndagen den 18 november börjar det med en fyra timmars utbildning i grundläggande UML, följt av en översikt över realtidsprofilen och en presentation av UML-2-gruppen UP2 av deras förslag till superstruktur. På tisdagen den 19e blir det en serie presentationer med en översikt över arbetet i SE DSIG, presentation av några av RFI-svaren, en paneldiskussion om SE UML och en demonstrationsarea där verktygstillverkare kan visa hur de stödjer UML för systemarbete.

Arbetsgruppen planerar att mötas på onsdag och/eller torsdag för ett arbetsmöte med uppdatering av krav och planer för en SE UML profil.

Veckan innan planeras en granskning av den konceptuella modellen för systemarbete. Närmare information om detta kommer.

Aktioner efter mötet

Följande aktioner beslöts och fördelades:

- Fortsatt samordning och uppföljning med dem som lämnat underlag till UML V2
- Distribution av papper om beteendeformalism till Steve Mellor (klart)
- Slutför samordning av granskning och genomgång av den konceptuella modellen för SE
- Uppdatera kravanalysen för SE UML V0.3
- Förbered kraven för SE UML som underlag till anbudsinfordran för SE UML-profil för granskning vid mötet i november.
- Slutför planer och logistik för informationsdagen.

Referentens kommentarer

Jag gick in i detta arbete utan större förväntningar om att kunna påverka. Som framgår av separat artikel var mitt huvudbudskap att övergång från arbete med programvarusystem till ”riktiga system” framför allt kräver:

- förenkling av UML syntax för att säkerställa att alla intressenter förstår modellerna utan någon längre utbildning
- att modellerna byggs så att man utgår från systemets uppgifter.

Jag kan nu konstatera gruppen inte tog till sig motiveringarna för att arbeta från systemets uppgifter och att det finns en tydlig risk att UML för SE får en minst lika komplex syntax som UML för programvara. Som framgår av detta protokoll krävs det en hel del arbete om man vill engagera sig i standardiseringsprocessen med många möten, framför allt i USA.

Ingmar Ögren
iog@romet.se

Upprop till ny intressegrupp Programvarusäkerhet

IG Programvarusäkerhet

SESAM:s rådsmöte har beslutat inrätta en intressegrupp för programvarusäkerhet. Sammanhållande för gruppen är FMV:s strategiska specialist inom programvarusäkerhet, Inga-Lill Bratteby-Ribbing. Verksamheten startar med ett första möte under SESAM:s seminariedagar 23-24 oktober.

Inriktning

Intressegruppens inriktning är programvarusäkerhet, dvs programvarans roll för systemsäkerheten. Detta område är av speciell betydelse för säkerhetskritiska programvarutillämpningar, för att förhindra att dessa kan leda till skada på person, egendom eller miljö. Programvarusäkerhet berör inte bara de programvarukomponenter som ingår i systemet, utan även användare, beställare och leverantörer samt de programvaruprocesser och de verktyg dessa använder sig av för utveckling, underhåll och drift av systemet.

Målsättning

Till målsättningarna med gruppens verksamhet hör att

- belysa och diskutera frågeställningar av betydelse för programvarusäkerheten (t ex delområden som arkitektur, design, återanvändning, operativsystem, underhåll).
- ta del av verksamhet inom Ag Metodik resp Teknik som berör programvarusäkerhet.
- bjuda in personer med speciell erfarenhet inom visst delområde till möten och seminarier.
- prova/demonstrera lämpliga verktyg för analys av programvara m a p systemsäkerhet.
- sprida information om programvarusäkerhet (metoder, verktyg, utbildning, litteratur) dels genom att
  - göra informationen tillgänglig via SESAM:s hemsida.
  - genomföra utbildning betr. detaljaspekter inom programvarusäkerhet (ett komplement till de introduktioner FMV redan bedriver).

Deltagare

Berättigade att ingå i intressegruppen är personer från företag med medlemskap i SESAM (företag, som först behöver ansöka om inträde, kan få vidare information via SESAM:s sekretariat). En förutsättning för att ingå i gruppen är att deltagare bidrar till gruppens verksamhet.

Första möte

Ett första möte hålls den 24.10 som avslutning på SESAM:s seminariedagar. Vid detta möte fastläggs gruppens målsättningar, en första planering av den fortsatta verksamhet utförs och några specialinbjudna talare presenterar sina erfarenheter av OS i säkerhetskritiska system.

Anmälan

Den som är intresserad av att deltaga och kan bidra till gruppen kan anmäla sig till Anna Kåsjö på SESAM:s sekretariat via epost-adress kasjos.kontor@telia.com. Ange namn, företag, adress, tel, email samt intresseinriktning m.a.p. delområde ovan.

Förberedelser

Synpunkter, förslag på ytterligare delområden etc inför det första mötet är välkomna och kan lämnas till Inga-Lill Bratteby-Ribbing, FMV: KC Ledstöd per tel: 018-12 02 63 eller epost: inga-lill.bratteby-ribbing@fmv.se.

Referat med kommentarer av Lloyd Mosemanns key-note vid STC 02

Förlorade vi vår religion

Lloyd K. Mosemann II var "key-note speaker" på årets Software Technology Conference i Salt Lake City i slutet på april. Mosemann, som var "deputy assistant secretary of the Air Force" i 25 år var bl a en av de stora förespråkarna för Software Engineering och Ada i det amerikanska försvarsdepartementet (DoD) och har tilldelats ett antal förtjänstmedaljer för sina insatser (bl a startade han just STC). Han avgick 1996 och är numera "senior vice president" på Science Applications International Corporation, SAIC, ett stort konsultföretag, här hemma kanske mest känt för att de utförde den första studien för Försvarsmakten om det nätverkade försvaret, för några år sedan. Föredraget höll Mosemann dock som privatperson.

Mosemann angav i sitt föredrag att orsaken till att han var en tidig och högljudd förespråkare för Ada, var primärt att till skillnad från andra språk så tvingar Ada fram tillämpning av grundläggande "software engineering". (Ungefär samma resonemang som låg bakom bildandet av SESAM.)

Mosemann var en av dem som förde Ada till den position och det mandat betr användning det fick, i varje fall formellt, inom DoD, men han var också den som innan han slutade där, fick vara med om att upphäva detta mandat.

I sitt föredrag gick Mosemann hårt åt den nu rådande "kulturen", eller rättare sagt bristen på kultur, inom DoD på programvaruområdet.

Mosemann ställde frågan "har vi förlorat vår religion"?

Att han använde religionsmetaforen, sa han berodde på att den är det traditionella exemplet på "tro"-baserat uppförande, d v s uppförande baserat på tro, och inte på utifrån påtvingade regler, som t ex tyngdlagen.

Kom ihåg hur de känslofyllda diskussionerna om huruvida Ada eller C++ skulle föredras, ofta beskrevs som "religiösa" argument baserade på tro snarare än på objektiva fakta.

RELIGION STYR FORTFARANDE PROGRAMVARUANSKAFFNINGEN

Sorgligt att säga styrs programvaruvärlden fortfarande av religionsliknande tro snarare än av objektiva utvärderingar, ansåg Mosemann.

Han sa när han lämnade DoD för sex år sedan, att den inriktning på Software Engineering och Ada som gällt inom DoD, ersatts av vad han kallade "bumper stick management", d v s att följa, snarare än leda konsumenten, och situationen inom DoD i dag har inte ändrats till det bättre.

[Som exempel använde han då striden mellan Beta och VHS på videomarknaden. Beta var bättre, men VHS vann. Han liknade Beta/VHS-situationen med Ada gentemot andra språk.]

Mosemann sa betr dagens situation inom DoD, att han ibland har känslan att det är "en blind som leder en blind" och att ledarskapet är lyckligt ovetande om i vilken riktning de är på väg.

Det enda meningsfulla direktivet om programvaruteknikfrågor från vare sig Office of Secretary of Defense eller försvarsgrenarna de senaste åren, är det s k Gansler-direktivet, att DoD för mycket stora system (sk ACAT 1 nivån) bara skall anlita leverantörer som har certifierats på CMM Level 3 . Så gott som alla informations- och ledningssystem faller dock nedanför den nivån och omfattas alltså inte av direktivet.

De senaste två åren har kostnadsökningarna för ACAT 1 program årligen varit 5.5 % beroende på felaktiga kostnads- och tidsuppskattningar samt kravändringar (låter som programvara). Ändå har dessa stora program de mest erfarna projektledarna inom DoD och industrin, och de har kravet att leverantörerna skall hålla CMM Level 3.

2/3 av DoDs anskaffningskostnader är för program under ACAT 1 nivån, för vilka det inte finns något CMM-krav. Mosemann gissar att dessa icke-ACAT 1 program är minst dubbelt så dåliga som ACAT 1 programmen. Det innebär att det skulle vara omkring 9 miljarder dollar årligen i kostnadsökningar beroende på estimerings- och tekniska problem - av vilka många säkert är programvarurelaterade. Mosemann tyckte att dessa program förtjänade större uppmärksamhet och styrning på programvarusidan än bara att utlämnas till den s k "best commercial practice", som är DoD enda kompetens- eller metodkrav på leverantörerna av denna mycket omfattande kategori system.

"BEST COMMERCIAL PRACTICE" EXISTERAR EJ HOS DoDs LEVERANTÖRER

Vad är fel med "best commercial practice"? Faktum är enligt Mosemann, att den helt enkelt inte förekommer hos DoDs leverantörer. Det är en fantasi skapad av dem som vill strömlinjeforma anskaffningsprocessen, minska specificering, uppföljning och granskning genom att ta bort tillfällena till sådana aktiviteter. Bästa sättet att rättfärdiga en "hands off" attityd är att fastslå att leverantörerna alltid gör allting rätt.

Det finns mer mogna programvaruorganisationer i dag säger Mosemann. Praktiskt taget varje stor leverantör till DoD kan skryta med minst en organisation på CMM nivå 4 eller högre, och flera på nivå 3. Å andra sidan utvecklas den mesta programvaran för DoD fortfarande i mindre mogna organisationer, huvudsakligen därför att anskaffningsansvariga inte kräver att den del av företaget som verkligen skall utveckla programvaran, skall vara minst nivå 3.

Mosemann berättade vidare att många brukade säga till honom att DoD måste hoppa på dotcomtåget, därför att dom människorna utvecklar och levererar programvara snabbt. Det är sant att Internet och tillkomsten av webb-bläddrare fundamentalt har ändrat den miljö i vilken t o m uppdragskritiska system utvecklas. Men i stället för att använda beprövade programvaruutvecklingsmetoder, har programvaruforskningen övergett sina omsorger om formella metoder, "peer reviews", "cleanroom"-processer och andra tillförlitlighetstekniker, inkl språket känt som Ada, vilket konstruerades för att främja tillförlitlighet. Och utom min vän Barry Boehm, säger Mosemann, har mycket av den akademiska världen mer eller mindre upphört med att undersöka bättre sätt att estimera systemkomplexitet och mäta programvarutillväxt. I stället har sådant som uppfinning av nya användargränssnitt, av nya distribuerade programvaruarkitekturer, av nya (mer flexibla och mindre pålitliga) programspråk, fått högsta prioritet. Målen med pålitliga prestanda och predikterbara utvecklingskostnader har i det hela taget ignorerats.

BERODDE DOT-COM KRASCHEN PÅ UNDERMÅLIG PROGRAMUTVECKLING

Mosemann återgav en konversation han nyligen haft med Wayt Gibbs, känd för sin artikel i Scientific American 1994 om "den kroniska programvarukrisen". Gibbs menade att det var lockande att argumentera att bristen på disciplinerad programutveckling var det huvudsakliga skälet till att så många nya dot-com företag kraschade och på vägen dit gjorde av med så otroligt stora penningmängder. Kanske det är att gå väl långt, ansåg Mosemann, det fanns förstås flera orsaker. Men man kan i alla fall konstatera att, med få undantag, praktiserade dessa företag modellen med "programmeraren som mästarhantverkare" i sin utveckling. Denna mycket gamla modell anses fortfarande vara "avant garde" i Open Source kretsar. Hur många programvaru-"start-ups" har under de senaste åtta åren underkastat sig en SEI-utvärdering eller försökt nå CMM level 3. Man behöver inte många fingrar för att räkna dem, ansåg Mosemann.

En annan sak som Internet, enligt Mosemann, har gjort omodern, är "fästningsmodellen" för säkerhet och föreställningen att en skarpsinnig administratör kan upprätthålla central kontroll över all komponenter i ett system. Det är inte längre realistiskt att drömma om att varje programvarukomponent med matematisk säkerhet är korrekt. I en värld där standarder av vilka nätverkskommunikation och användargränssnitt beror, ändras var 12- 18 månad, är det inte mycket mening med att stadfästa detaljerade krav, tillbringa ett år med att bevisa att de är internt konsistenta och så spendera två år ytterligare på att bygga dem enligt specifikation.

Resultatet av detta är enligt Mosemann att det finns en ny rörelse mot "autonomous computing", en vag term som innefattar flera forskningsinriktningar med målet att tämja komplexiteten för att uppnå tillförlitlighet. Ansatsen är inte "formell" utan "organisk"; hitta sätt att bygga system som kan bota sig själva, som kan genomföra sina uppdrag trots buggar och hårdvarufel och t o m sabotage. En del säger att sådana mål kan uppnås på 10-20 års sikt, men satsa inte Era lönecheckar på dem, rådde Mosemann.

Ironiskt är det problem som nu en stor del av industrin måste ta ställning till det motsatta. Enligt Mosemann påminde Gibbs om att många VD:ar vaknade upp 1997 och upptäckte en tidsinställd bomb i form av Y2K-problem tickande i sina system och tvingades till mycket kostsamma åtgärder. En ironisk konsekvens av detta är att när de 1999 "uppgraderade" sina system för att fungera mot Internet och olika dagsländor till standarder, som diverse "start-ups" byggde "lösningar" till, är de i dag är lika föråldrade som Algol. Som resultat är dessa "uppgraderade" system tickande bomber.

Även om de inte kommer att krascha samtidigt, kommer deras felyttringar att bli lika opredikterbara och mycket svårare att rätta till.

VART TOG SOFTWARE ENGINEERING ETC VÄGEN

Mosemann tog sedan upp ett antal ämnen som han hade märkt varit mycket tydligt frånvarande på Software Technology Conference de senaste åren: Software Engineering, Product Line Development, Formal Methods Programming och Predicatbility.

Han citerade Paul Strassman, en känd IT-profil även i civila sammanhang, som 1991 när han var DoDs Director of Defense Information, hade sagt att DoDs prioritet nr 1 var att omvandla sin programvarteknologiförmåga från en "cottage industry" till en modern industriell produktionsmetod. Detta har inte skett enligt Mosemann. Varför inte? Därför att det fordrar Software Engineering. Software Engineering innehåller tre nyckelelement: metoder, verktyg och procedurer (processer), som gör det möjligt för en chef (manager) att styra programutvecklingsprocessen och tillhandahålla dem som praktiserar programutveckling en grund för att bygga högkvalitativ programvara med på ett effektivt sätt.

Den grundläggande ingrediensen, enligt Mosemann, i en Software Engineering ansats, är utformningen av en robust programvaruarkitektur. Med arkitektur menas då inte gruppering och hopkoppling av servar, routrar och PC:ar, utan det grundläggande konstruktionsupplägget av själva programvaran, identiteten hos modulerna och deras relationer, inkluderande särskilt infrastrukturen, styr- och datagränssnitt som tillåter programkomponenterna att fungera tillsammans som ett system.

Mosemann berättade att han för en tid sedan hade hört av en närvarande vid en DoD Systems Design Rewiew, att leverantören beskrivit sitt föreslagna system som "modulärt". Det är en fin "arkitekturterm" och accepterades utan diskussion som riktigt vid granskningen. Den oberoende närvarande fann sedan när han tittade närmare på systemet, att det bestod av endast två moduler. När detta påpekades för DoDs projektledare, sa denna bara "Leverantören säger att det är modulärt. Han är klyftigare än vi är". Den här lilla incidenten visar bl a att varken leverantören eller DoDs projektledare begrep vad arkitektur är och att utsikterna till att det aktuella projektet skall bli framgångsrikt är små.

Alltför ofta är DoDs ursäkt för att inte kräva en utvärdering av arkitekturegenskaper att "kraven ändras så ofta - vi kan inte riskera att bli inlåsta i en viss arkitektur". Detta är ju totalt fel menar Mosemann; det är särskilt när kraven är föränderliga, som man behöver en arkitektur så att kravändringar lättare kan tas om hand.

Software Engineering Institute (SEI) kallas ofta in för att göra oberoende analyser av varför ett anskaffningsprojekt inte har producerat de önskade resultaten. Varför vänta tills det skall göras en efterhandsanalys (post mortem) och få reda på vad som misskötts; varför inte kalla in dem från början i projektet och låta dem granska anbudsförfrågan och åtagandedefinitionen, sedan vara med i anbudsutvärderingen av vilka Software Engineering metoder som föreslås tillämpas, av utvecklingsmiljön som man anser använda och av den föreslagna arkitekturen. och när projektet är igång, för att bedöma kvaliteten i software engineering och utvecklingsprocesserna. SEI är inte billiga, men att terminera ett $100 M projekt p g a otillfredsställande resultat är inte heller billigt.

TVÅ OLIKA VÄRLDAR AV "BEST COMMERCIAL PRACTICE"

För att återgå till "best commercial practices", menade Mosemann att det finns två mycket olika världar där ute. Det finns myndighets-leverantörernas värld och det finns den verkliga kommersiella världen - banker, försäkringsbolag, UPS och FedEx, Eckert Drug och Disney World o s v. Dessa senare företag utvecklar sin programvara själva och utnyttjar de bästa tillgängliga verktygen; de väljer inte de billigaste verktygen. De gör sig inte beroende av COTS eller utomstående utvecklare - deras programvara är deras "business" och de anser att den ger dem ett konkurrensmässigt övertag. De vill ha kontrollen över den och använder de bästa tillgängliga verktygen, oberoende av vad de kostar.

PRODUKTLINJEUTVECKLING - MED CELSIUSTECH (!) SOM PIONJÄR

Produktlinjeutveckling blir också allt vanligare i den verkliga kommersiella världen påpekade Mosemann. Förutom CelsiusTech, det svenska företag som var det första i världen som exploaterade fördelarna med en produktlinjeansats till utveckling av programvarutillämpningar redan i senare delen av 80-talet, finns det nu ett antal företag Du känner igen namnen på, som använder en arkitekturcentrerad ansats. T ex Nokia, Motorola, Hewlett-

Packard, Philips, Cummins Engine och, tro det eller ej, en myndighet, the National Reconnaissance Office (NRO). NRO rapporterar 50 % reduktioner i kostnader och utvecklingstid och nära en tiofaldig minskning av fel och utvecklingspersonal.

Mosemann listade de vanligaste och även för DoD relevanta skälen för att välja en arkitekturcentrerad produktlinjeansats för programvaruutveckling:

- Stora produktivitetsökningar
- Förbättrad "time-to-market" = få ut vapnen i fält snabbare
- Förbättrad kvalitet
- Ökad kundtillfredsställelse = "Warfighter satisfaction"
- Möjliggör masskundanpassning
- Kompenserar för problem med att få tag på programvaruingenjörer

De nämnda företagen och NRO har verkligen "best practices", men dessa är ännu inte brett erkända att vara just "best practices". Uppriktigt sagt, enligt Mosemann, krävs det beslutsfattare på höga nivåer (ovanför projektledare) inkl t ex DoDs controllers, för att få produktlinjeansatsen för programvaruutveckling och anskaffning erkänd och föreskriven.

De största hindren för produktlinjeansatsen, bortsett från okunnigheten om dess fördelar, är kulturella, organisatoriska och i synnerhet DoDs skorstensorienterade sätt att budgetera och ge anslag. DoD har enligt Mosemann många potentiella produktlinjer för programvara, men ingen av dem har hittills byggts. Produktlinjer fordrar strategiska investeringar, vilket verkar ligga utanför DoDs controller- och anskaffningskretsars referensramar. Ändå är det DoD som oftast använder uttrycket "strategiskt".

Mosemann rekommenderade läsning av en ny bok - Software Product Lines - som i år getts ut av Addison-Wesley. Författarna är från SEI. Sorgligt nog verkar det som om de flesta läsarna hittills är från kommersiella företag. Det fanns en tidigare version av boken - Software Architecture in Practice - som såldes i mer än 14000 ex, men jag slår vad, sade Mosemann, om att ingen av Er har hört talas om den.

Även om leverantörer till myndigheterna är kommersiella organisationer, och jag arbetar för en så jag vet det, så har de ingen identifierbar "best commercial practice". I grunden erbjuder de det som deras kunder, myndigheterna, frågar efter. Enda orsaken till att många av dem kan skryta med att ha åtminstone några CMM-organisationer, är därför att, med början för 10 år sedan. många av myndigheternas anbudsförfrågningar fordrade en s k Software Capability Evaluation som ett villkor för att överhuvudtaget få offerera. I dag är det, sorgligt att behöva säga, så att många leverantörer i sina anbud anger att de har CMM-godkännande, men sedan i verkligheten genomför utvecklingen med en organisation som inte ens kan stava till CMM.

MYNDIGHETERNA BRYR SIG INTE

Varför låter myndigheterna detta fortgå? Varför görs det inte Software Capability Evaluations längre. Svaret man kan höra, är att de tar för lång tid att göra och kostar för mycket. Bättre kanske att göra ett snabbt leverantörsval och längre fram acceptera en undermålig produkt eller terminera kontraktet. Alltför ofta har myndigheterna bett om COTS. Hur många fiaskon med COTS-baserade anskaffningar har det förekommit det senaste decenniet. För många!

Det säger jag - "best commercial practice" innebär inte att göra sig av med alla programvarukunniga hos myndigheterna och till 100% lita på att leverantörerna levererar bra programvara.

"Best commercial practice" är det som verkliga kommersiella företag gör. De har egen programvaruexpertis, de använder en robust programutvecklingsmiljö, och de baserar sina programutveckling på en sund programarkitektur. Det är ingen hemlighet eller överraskning att stora erkända verktygs- och metodföretag har en växande marknad bland de kommersiella företagen och en minskade marknad hos myndigheterna och deras leverantörer.

Det betyder inte, framhöll Mosemann, att jag föreslår att myndigheterna kan eller bör utveckla programvara internt. Men jag föreslår mycket bestämt att myndigheterna behöver tillräckligt mycket programvaruexpertis för att veta vad de köper. Det stämmer fortfarande att man får vad man betalar för och att äpplen och apelsiner inte är samma sak.

Mosemann citerade en ny bok av Watts Humphrey - *Winning with Software - An Executive Strategy*, vilken föga förvånande riktar sig primärt till ledare för kommersiella företag. Men även varje DoD anskaffningsansvarig, programledare och projektledare behöver läsa och förstå budskapet: Programvaruprojekt misslyckas sällan av tekniska orsaker; problemet är dålig ledning (management). Humph-

rey ställer två intressanta frågor och belyser dem med många exempel*: "Varför går kompetenta programvarumänniskor med på leveranstider som de inte har någon aning om hur de skall uppnås"* och "*Varför accepterar beslutsfattare leveranstidsåtaganden när ingenjörerna inte visar några bevis för att de kan hålla dem*"?

Humphrey framhåller att ledningens odisciplinerade attityd till åtaganden bidrar till alla de fem främsta orsakerna till att projekt misslyckas:

- Orealistiska tidplaner
- Olämplig bemanning
- Ändrade krav
- Dåligt kvalitet på arbetet
- Tron på magik

FORMELLA METODER OCH CMMI

Mosemann hade en del positivt att säga om formella metoder, speciellt om SPARK ("säkert" subset av Ada) och om CMMIs tillämpning vid utveckling av inneslutna system, där Systems Engineering aspekterna är viktiga. Däremot delade han tydligen en uppfattning som finns på sina håll, om att CMMI är "overkill" för Informationssystem och de flesta ledningssystem och han var starkt emot att det talas om att upphöra med stödet för CMM for Software. (Se artikel om CMMI i RV nr 1 i år.)

PREDIKTERBARHET

Mot slutet av sitt anförande tog Mosemann upp frågan om betydelsen av predikterbarhet vid programvaruutveckling. Predikterbarhet är den enda metrik som beslutsfattarna på högre nivå frågar efter. Hur kan vi åstadkomma att dessa beslutsfattare inser att de inte kan köpa programvara direkt från leverantörernas "show-rooms".

De måste fås att förstå den software engineering paradigm som producerar programvara. Mer än så, de måste förstå att för att vara säkra på att få programvara som fungerar enligt en predikterbar tidplan och till predikterbara kostnader, måste det finnas någon inom myndigheten som är kunnig nog att formulera de grundläggande processer som skall utnyttjas av leverantören för att producera programvaran.

Detta är viktigt, därför att eljest kommer anbudsgivarna att offerera orealistiskt låga priser och orealistiskt korta leveranstider, och få kontraktet. För att leverera till detta låga pris, det betyder att man inte har råd med någon robust utvecklingsmiljö, att ingen tid och inga insatser kan ägnas åt en vettig programarkitektur och förmodligen betyder det också att man bara har råd att sätta in "billig" personal. Om myndigheterna vill få programvaran rätt, måste tillräcklig processanvisningar ges för att säkerställa att alla anbudsgivarna offererar på huvudsakligen samma kapabilitetsnivå. Mosemann framhöll att han anser att myndigheterna skall vara explicita betr behovet av en arkitektur, en robust utvecklingsmiljö och kanske även betr ett programspråk (som SPARK för säkerhetskritiska tillämpningar), men som minimum måste man kräva att den organisation som skall genomföra utvecklingen är på minst CMM Level 3.

Det är verkligen sant att **"you get what you pay for"**. Om Du vill ha det billigt, så får Du det billigt, men det kanske inte fungerar som förutsett, om det fungerar alls.

* * *

I en intervju ungefär samtidigt med framträdandet på STC, med en representant för Ada Resource Association (en slags företagssponsrad lobbyorganisation för Ada), gjorde Mosemann bl a följande uttalanden.

De flesta beslutsfattare och seniora människor nu för tiden har ingen aning om vad Ada är för något, eller varför Ada borde väljas före någonting annat. Mosemann berättade att han hade problem att "sälja" Ada till sina kollegor och överordnade i DoD, när C++ hade kommit in på scenen. Folk hade hört att C++ var nyare och bättre, och tänkte inte på, eller hade ingen aning om, svårigheter med multipelt arv och annat som gjorde C++ svårt att underhålla och vilket hade stor inverkan på livscykelkostnader.

Programvaruansvariga använder fortfarande Ada för robust och portabel programvara. För övrigt använder de flesta företag sådant som är nytt och populärt. Ada sorteras ofta bort av programvaruansvariga därför att "inte tillräckligt många utvecklare är erfarna programmerare och så är Ada inte "sexigt" så att folk vill inte använda det".

* * *

Är det en stenåldersinställning som Mosemann har fastnat i, eller ligger det något i hans resonemang?

OM VI JÄMFÖR MED SVERIGE

Det ligger naturligtvis nära till hands att jämföra

situationen i USA och Sverige.

Följande är bara referentens spekulationer. Är det kanske någon av medlemmarna som skulle vilja kommentera detta?

Efter det att FMVs i slutet på 1994 uppdaterade anvisningar för programvaruanskaffning, efter ett fåtal år automatiskt upphävdes av ett generellt påbud om utsortering/upphävande av gamla föreskrifter, verkar det inte längre finnas några generella anvisningar betr programvaruanskaffning vid FMV. Undantaget är säkerhetskritiska system där det finns vissa krav och betr programvaran i dessa finns också "Inga-Lills handbok", som ju väckt glädjande stort intresse och säkert gett många något att tänka på; dock är handboken inte föreskrivande.

Kanske är FMV och DoD ungefär lika "laglösa" på programvaruområdet. DoD har dock kravet om CMM Level 3 för leverantörer av sina stora system, vilket kanske inte finns hos oss.

Det skall ha förekommit en omfattande intern processutveckling på FMV de senaste åren. Kanske har man i samband med dem konstaterat behov av riktlinjer även för programvaruhantering och vidtagit åtgärder. Den nya programvarustandarden ISO/IEC 12207 och även den kommande ISO/IEC 15288 "Systems Engineering – System Life Cycle Processes" skulle annars kunna vara utgångspunkter för att ta fram nya anvisningar både som stöd för den egna personalen och som riktmärke för industrin. Kanske används den gamla på beprövade erfarenheter byggda MIL-STD-498, som hade börjat få fotfäste både inom FMV och hos vissa leverantörer, fortfarande som "likare" i vissa projekt. (498 var en av de standarder som DoD förbjöd när man under andra halvan av 90-talet gick över till "best commercial practice".)

Förhoppningsvis är i alla fall inte attityden till programvarufrågor lika "laissez faire" på FMV, som den av Mosemanns beskrivning verkar vara hos DoDs anskaffningsansvariga.

Det vore också intressant att veta hur många försvarsleverantörer i Sverige som på eget initiativ är utvärderade på eller fyller krav motsvarande CMM på Level 3.

Betr användningen av Ada förefaller det, trots (eller tack vare?) bristen numera på generella krav eller anvisningar om programspråk, som om Ada fortfarande kan hävda sig i den typ av stora realtidsorienterade försvarskritiska system som det i första hand var avsett för. Vi har i Rendezvous de senaste åren kunnat läsa om flera nya system som, helt frivilligt tycks det, utvecklats i Ada med mycket goda resultat. Där Ada används borde man automatiskt kunna räkna med att seriös Software Engineering också praktiseras, vilket kanske fortfarande är det övergripande målet.

F d CelsiusTechs världspionjärinsats betr produktlinjebyggnad har av naturliga skäl inte fått så stor efterföljd hos våra andra försvarssystemleverantörer, men det finns säkert en viss potential för att tillämpa liknande koncept på andra håll och även i det nätverkade försvaret.

Det vore intressant att få till stånd en diskussion i RV om dessa frågor, som är rätt centrala för SESAM.

Vilken är den rättvisande bilden av dagens situation och vart är vi ur programvaruteknisk synpunkt på väg med det nätverkade försvaret?

Inlägg välkomnas!

I Carlsson hade läst och tolkat Mosemanns anförande, tittat på Ada Resource Associations hemsida samt spekulerat vidare.

. . . . . .

15288 antagen

Vid en omröstning i somras inom ISO antogs slutgiltigt ISO/IEC 15288 Systems engineering - System life cycle processes. Därmed har ett långvarigt arbete med denna standard krönts med framgång. Vi har i RV flera gånger uppmärksammat detta arbete, som särskilt genom Bud Lawsons medverkan, bl a som "arkitekt" för standarden, haft betydande svenskt inflytande.

Vi gratulerar Bud och andra som varit inblandade i arbetet till det lyckliga slutet.

Den nya standarden har redan börjat prövas på flera håll här hemma. Här finns kanske en chans att utvinna komparativa fördelar för svenska både myndigheter och industri!

SNART industridag med Parnas

Professor David Parnas, en av de mest kända profilerna och debattörerna i världen inom programvarutekniken och en del andra teknik- och samhällsfrågor, var huvudtalare vid det heldagsseminarium som SNART arrangerade på KTH den 19 augusti med fokus på säkerhetskritiska realtidssystem.

Parnas genomgång visade sig ha titeln "Design through Documentation: The Path to Software Quality".

Genomgången följdes av en paneldebatt på temat "Safety of future embedded systems".

Några intryck från Parnas genomgång redovisas här följt av en sammanfattning av paneldebatten, vilken vänligen ställts till förfogande av Martin Törngren på KTH och SNART.

Design through Documentation: The Path to Software Quality

Förutsägelsen i förra nr av RV att Parnas skulle dra stor publik, visade sig slå in. Det var fullsatt i Kollegiesalen där luftkonditioneringen denna heta sommar för dagen (?) var ur funktion, vilket bidrog till den förtätade stämningen. (Efter lunch fick vi flytta upp till en lokal i nya biblioteksbyggnaden där det var något mer luftväxling.)

Parnas började med att berätta att han, som de senaste åren arbetat vid McMaster University i Kanada, just accepterat en post vid Universitetet i Limerick, där han skall fortsätta sin forskning om dokumentationens betydelse vid programutveckling och -underhåll.

Parnas är ursprungligen ingenjör (elektro), vilket haft stor betydelse för hur han ser på programvarutekniken och dess tillämpning. På Elektro, sade Parnas, lärde man först ut principer. Det blev en kulturchock för honom när han först kom i kontakt med datavetenskapen, där man började med att lära ut helt godtyckliga regler för programmering i något för tillfället populärt språk. Man visade exempel på program, men sade inget om principerna för att konstruera dem.

I traditionell ingenjörskonst utarbetas en följd av dokument innan den egentliga konstruktionen påbörjas. Alla dokumenten används för analys och granskning och efter att de reviderats tjänar de som input till nästa fas. När de (oundvikliga) felen upptäcks och ändringar måste göras, finns konstruktionsdokumenten redan arkiverade och tas fram, uppdateras och granskas igen. Varje ny förfining granskas mot de föregående dokumenten.

Vid programvarukonstruktion påpekade Parnas, tillämpas nästan aldrig denna "vattenfallsmetod". Fast det borde vara attraktivt, kan eller vill inte programkonstruktörer skriva precis dokumentation. I stället skriver de vaga pratbubblor som inte kan bli föremål för rigorös analys och är värdelösa för dem som skall utföra nästa steg i processen.

Parnas fortsatte sedan med att beskriva hur man kan skriva precisa, kompletta och testbara dokument som kan bidra till en förbättrad programutvecklingsprocess.

Det skulle föra för långt att här mera i detalj referera Parnas genomgång, så det får bli några spridda citat, som kan illustrera karaktören på Parnas budskap. Som Ni redan har förstått är han inte rädd för att utmana rådande konventioner.

Parnas grundläggande tes var: *det kostar inte mera att göra bra programvara!*

Varför är programvara så ofta ett problem

* Utvecklare underskattar systematiskt svårigheten att bygga program som skall fungera under lång tid
* De *skriver* snarare än *konstruerar* (designar) dem
* De gör inte
    * systematisk identifiering och nedskrivning av kraven

* granskningar av kravdokument
* explicit konstruktion, dokumentation och granskning av programstrukturen
* noggrann inspektion av alla konstruktioner och program

Dessa steg är standardförfarande för all annan produktutveckling an programvara.

Dessa steg genomförs inte för programvara därför att

* "Programvara är enkelt"
* "Koden är självdokumenterande"
* "Program är bara en följd av instruktioner"
* "Vem som helst som kan språket kan programmera"

Famous last words!

Om vi hade bättre dokumentation, skulle vi bättre förstå vår programvara därför att

* Bra dokumentation hjälper oss att förstå vår programvara
* Du kan inte producera sådan dokumentation utan att förstå programvaran

I programvara betyder "nästan rätt": "fel"

* små saker betyder mycket
* säkerhetsöverträdelser sker genom att små misstag exploateras

I programvara betyder "vag": "lätt att missförstå"

När något är "nästan rätt" kan det fungera för det mesta, men

* gå fel vid oväntade tillfällen
* utnyttjas av någon med kriminella avsikter

För många programvarumänniskor är "abstrakt" = "till ingen nytta", men vi behöver abstrakt dokumentation. Utan den kan vi inte förstå vår programvara. Den blir outgrundlig, oförklarlig.

Dokumentationens relevans

"Vi har viktigare saker än att dokumentera"
"Vi säljer kod, ej dokument"
"Om jag hade velat skriva dokument, hade jag valt språklinjen"

men,

* vi kan inte samla, granska eller kontrollera om krav är kompletta om vi inte dokumenterat dem
* vi kan inte göra, granska eller leva upp till strukturbeslut utan att de är dokumenterade
* vi kan inte inspektera konstruktioner utan konstruktionsdokumentation
* vi kan bäst inspektera program med hjälp av dokumentation

Design genom dokumentation är nyckeln till bättre programvara

De fyra viktigaste dokumenten programvaruberoende organisationer behöver

* Ett **kravdokument** som exakt talar om för användarna vad de kommer att få och programmerarna vad de skall bygga
* En **modul-guide** som leder granskare och reparationsprogrammerare till exakt de program som kommer att påverkas av en föreslagen ändring
* En sats **modul-gränssnittsspecifikationer** som talar om för programmerarna vad de skall bygga och andra programmerare vad de kan vänta sig av en modul
* En sats **modul-interna design dokument** som (1) noterar viktigare interna designbeslut för granskare, (2) leder och stödjer programmerarna, (3) leder och stödjer underhållsfolket

En modul enligt Parnas: ett bra arbetspaket är en bra modul; d v s ett arbetspaket som tillåter att man arbetar självständigt så långt möjligt är en bra modul

Om det är så bra att dokumentera, varför gör vi inte det?

Det enkla svaret: Programmerarna vet inte hur det skall göras!

* Föreställ Dig att Du skall utveckla en bil utan att Du vet hur man specificerar gängorna på skruvar och muttrar
* Föreställ Dig att Du skall utveckla elektronik utan att Du vet hur man specificerar impedans, spänning, ström, induktans, frekvensrespons
* Föreställ Dig att Du skall producera för oljeindustrin om Du inte vet hur man specificerar viskositet, oktantal, etc

* Föreställ Dig att Du skall utveckla elektriska kretsar utan kunskaper om kretsteori

Du **behöver inte föreställa Dig** att försöka producera programvarusystem utan kunskap om komponentegenskaper: **Det görs varje dag**!
(Och resultatet blir därefter.)

Bygg aldrig på sand

Programvara hör till de mest komplexa produkter som finns

Vi kan inte hoppas på att hålla sådana produkter under intellektuell kontroll, om vi inte bygger på solida, sunda, väl förstådda grunder.

Om

* vi inte förstår innebörden av vår notation
* vi inte förstår de tillhörande inferensreglerna, eller om
* dessa regler är komplexa och villkorliga

så hjälper denna notation inte oss att förstå vad vi gör

För att bygga komplexa produkter måste alla designnotationer ha en sund och enkel matematisk bas.

Idag är det alltför många, **inkl svenskar**, som framhåller "Undefined Modelling Languages" (UMLs) som en lösning. Utan en precis definition av notationen blir UML:er del av problemet, inte lösningen.

(Indikerar kanske att Parnas inte är riktigt överens med Ivar Jacobsson, men referentens intryck är att de båda predikar ungefär samma "ordningens" evangelium)

Ändringar i efterhand av notationer fungerar aldrig. Så snart folk börjar använda vagt definierade notationer, sprider de sig som ogräs.

Betr den pågående revisionen av UML: hur kan man skriva om UML, när det inte är definierat?!

Ett systematiskt sätt att upptäcka och dokumentera krav

Steg 1

* identifiera övervakade variabler
* identifiera styrda variabler

De primära *övervakade variablerna* är saker utanför systemet, vilkas värden skall påverka systemets output

De primära s*tyrda variablerna* är saker utanför systemet, vilkas värden skall bestämmas av systemet

Detta är endast början, men för många system kan man inte ens få tag på en fullständig lista över dessa variabler och det finns ingen enighet om vad de är.

Parnas berörde sedan hur man kan gå till väga för att lägga till sådana variabler under designprocessen, vilket alltid är nödvändigt i verkligheten och beskrev sedan hur man tillämpar "Divide and Conquer". Han förordade användning av tabeller framför regler.

Betr att få med tiden i sammanhanget, visade Parnas hur man kan beskriva alla dessa variablers tidsvariation med tidsfunktioner. Och att hantera realtid, var inget problem, i motsats till vad vissa datavetenskapare tror. Temporal logik gillades ej; den behövs inte enligt Parnas.

Matematiken måste tas med i verktygslådan och Parnas visade hur man kan få bort den "svåra, skrämmande" matten ur dokumentationen.

Att dokumentera den interna designen

Detta är vad som behöver dokumenteras och det skall göras innan man släpper in programmerarna:

(1) Den kompletta datastrukturen

(2) Tolkningen av den datastrukturen (känd som abstraktionsfunktionen)

(3) Resultatet av varje program

Parnas diskuterade sedan hur man bör beskriva program, hur man skall beskriva startvärden och slutvärden hos variabler, baserat på Harlan Mills "Clean Room", men med vissa skillnader, som att använda tabellformat, hur hantera determinism och hur man skiljer mellan relationer som specifikationer och relationer som beskrivningar.

Parnas fick en del mothugg från åhörare betr att använda tabeller, vilka faktiskt såg rätt komplicerade ut för en lekman, jämfört med den rätt enkla kod som behövdes för att realisera funktionen.

Han behandlade också hur man kan arbeta med vissa imperfektioner i dokumenten. Användning av matematik i ingenjörssammanhang betyder inte tro på perfektion hos program eller matematik.

Kan folk lära sig detta

Han ställde sedan frågan, kan vanliga människor lära sig sånt här? Det kan finnas problem menade Parnas, men gav exempel på hur mer formella

metoder använts med gott resultat i många sammanhang och av “vanliga människor”, ej doktorer. Matten som används är inte mer “formell” än koden. Men även utan matematiken kan man få stora förbättringar i dokumentationskvaliteten.

Parnas hade givetvis synpunkter på hur granskningar både av dokument och av kod bör gå till; det skall t ex vara aktiva granskningar för att få effektiva granskningar och han anvisade diverse vägar att uppnå detta; mycket matnyttigt.

Slutligen behandlade Parnas rätt utförligt test av programvarusystem för att underbygga sin “bottom line” där:

Testning kan bli mycket bättre om man har dokumentation!

Sammantaget en mycket lärorik dag, där det enda referenten kan beklaga att den inträffade först när ens karriär var över. Hade man kunnat insupa detta för 25-30 år sedan, skulle kanske mycket gått annorlunda till, i varje fall i flygelektroniksystemens anskaffningsverksamhet.

I Carlsson hade svettats på KTH

Summary of the panel debate and discussion on the theme,

Safety of future embedded systems

that took place at the Snart/Encress seminar, August 19, 2002 at KTH

Scope of the seminar and the panel

The Snart/Encress seminar on August 19, 2002, had a focus on safety critical real-time systems. At the seminar the invited speaker, Prof Dave Parnas, talked about the importance of software documentation to describe a design. He then presented one approach that emphasizes readability at the same time as formality to enhance the software development. About 80 persons attended the seminar that ended with a panel discussing ”safety of future embedded systems” with the idea to address key challenges and how they can be addressed.

The topic for the panel was formulated as follows by the panel moderator: In the coming era of ubiquitous and embedded computing, electronics, software, sensors and novel actuators will provide completely new functionalities promising to enhance our everyday lifes. Examples of such functionalities include active collision avoidance, convoying and drive by wire in automobiles, robotic assistants in houses, and robot assisted surgery. While such functionalities can be beneficial and even save lives given that they work properly, their realisation requires complex embedded systems that may jeopardize the safety (dependability in a broader sense) of the new products. The embedded systems provide many new failure modes both at component and at the system level. While it probably can be argued that such new products can be built to be safe enough it is questionable whether current technologies and methodologies allows this in a sufficiently cost-efficient manner. One open question for the panel is how such safety critical systems can be built to be safe enough in a cost-efficient manner?

The panelists were asked to initially present their view on the following questions:

What is the most important problem that needs to be addressed to enable the development of dependable products such as the above mentioned ones?

Is there a key technique or method that drastically could improve the current state of practice?

The panel participants included

Simin Nadjm Tehrani, Ass. Prof, Linköping Univ. of Technology

Rolf Johansson, PhD- Senior Safety specialist, CRT (www.crt.se)

Peter Eriksson, Senior Specialist, ABB Robotics

Harold Bud Lawson, Prof.

David Lorge Parnas, Prof.

Martin Törngren, Prof, panel moderator.

Summary of problems/issues raised by the panelists and the audience

* Trade-offs between different requirements and system qualities is becoming very difficult. As an example the obvious conflict between time to market, cost and safety was mentioned.

* Multidisciplinarity is required for safety critical systems that typically require skills from different disciplines, meaning among other things that different models and techniques for analysis and design are required and moreover need to be properly combined. In addition people with different background needs to understand each other.
* The combinatorial explosion inherent in digital and state based systems reflecting the system complexity and making for example verification difficult.
* System complexity and systems integration, involving many stakeholders, companies and subsystems in producing an end product.
* The evolving nature and long life time of many products, meaning new/changed requirements and features which affect the whole system and pose major challenges to an architecture and system maintainence.
* The difficulty of finding major contributions to unsafety where one problem is that people (e.g. designers) do not like to find weaknessess of their own designs (on a broader scope this relates to companies uninterest to reveal incidents etc.).
* Components and their reuse is a key problem since safety is a systems property and thus always has to be analysed and ensured given the relevant system context making component reuse in safety critical systems a difficult problem.
* The relevant education in software engineering is missing, as is a common understanding of what the core body of knowledge that should be taught.
* There is a need for licensing of engineers working with safety critical systems.
* The importance of human factors was emphasized where users of safety critical systems are part of a safety critical system and where there many incidents and accidents have been due to poor human - machine interaction/ understanding.
* Available techniques and methods for safety critical systems need to be modifed (or new ones invented) to handle software (Inga-Lill)

Summary of key solutions/approaches towards the above mentioned problems as raised by the panelists and the audience

* Compositional reasoning can be a key by providing the ability to reason about properties of combinations of components. The availability of such techniques will be a step towards enabling faster development and verification.
* Selection and deployment of the ”right” system architecture for the problem. An example with the ATC system architecture that remained stable for many years was mentioned.
* Holistic view of the roles and responsibilities of the stakeholders involved as encapsulate in life cycle processes (e.g. ISO/IEC 15288)
* Support for the processes in the form of minimal but sufficient methods and tools.
* To be able to pinpoint the really critical parts of a system/design and then to be able to isolate that part as far as possible to facilitate and simplify system development and maintainance.
* One key is to learn from other branches with a well established safety culture. The establishment of a safety culture is perhaps the most important aspect.
* A key is to mimise complexity so that the system can be properly developed and maintained. Complexity reduction means to simplify the failure modes and this can be achieved by proper system decomposition. Also, from an analysis point of view, system complexity can be reduced by identifying behavioural modes.
* Documentation is key to proper understanding/ analysis/design and maintainence, thus one of the most central issues.
* There is a need for licensing of engineers working with safety critical systems.

Martin Törngren
/Panel moderator

SESAM CD 2002

Det blir en SESAM CD även för 2002. Efter höstseminariet planeras en CD som kommer att innehålla presentationerna från de senaste fyra SESAM-seminarierna, d v s fr o m höstseminariet 2000 (där vi dock ej har alla föredragen elektroniskt). Dessutom avses CDn innehålla en del intressant bakgrunds och referensmaterial. Dock ej lika mycket som SESAM CD 2001, då vi ej har rätt att framställa ytterligare kopior av en del av det materialet. Men det kommer en del mycket intressant nytt bl a föredragen från FM och FMV om Ledsystprojekten, vilka hölls vid ett AFCEA-seminarium alldeles nyligen. Distribution kommer att göras i början på november till SESAMs medlemmar och till deltagare vid årets två seminarier.

Mer Ada på SESAM CD 2002

I förra årets CD hade vi förmånen att kunna ta med ett par turorials i viktiga Ada-ämnen av Matthew Heaney. Dessa återkommer i årets CD men nu kompletterade med ytterligare en tutorial presentation av Matt, "Charles: A Data Structure Library for Ada95". En översikt över innehållet i den framgår av artikel på annan plats. Vi rekommenderar verkligen alla som är intresserade av Ada att studera Matts tre tutorial på SESAM CD 2002, där finns mycken lärdom och erfarenheter att insupa.

Vi är mycket glada över att ha kunnat bygga upp kontakterna med Matt och har bett honom att presentera litet om sin bakgrund och syn på hur man kan göra god objekt-orienterad programmering med Ada95; följer här med Matts egna ord.

"I've been developing software for over 15 years, and have designed a few large, real-time systems in Ada. I've always been interested in object-oriented programming, especially as it relates to programming-in-the-large.

For large software systems, you have be to very concerned about minimizing dependencies among modules. This is especially true in static languages as C++ and Ada95, which allocate objects on the stack. Even though the representation of an abstract data type is hidden from the user (in the sense that it's not accessible), it's not hidden from the compiler, and the compiler has to recompile any module that depends on that type when it is changed. Forced recompiles can really wreak havoc with development cycles.

That's why you have to be smart about how you design your system, and always remain aware of how modules interact. You have to organize the modules and define the types in such a way that you're able to limit the scope of a recompilation. Here Ada95 provides a lot of language support, with its subsystems and child units.

There are also various design patterns for this, such as Factory Function and Flyweight. A factory function declared as a public child subprogram, for example, turns out to be an extremely useful idiom. You can often reduce module dependencies by playing Chapter 13 tricks to bypass the type system, but in a way that doesn't allow the abstraction to be violated. Yet another technique is move a dependency from a spec to the body (say, by initializing a record component with the value of a function call, instead of a compile-time constant), which defers a dependency from compile-time to elaboration time. Indeed, class-wide programming using tagged types and dynamic binding is simply a way of delaying another kind of choice until run-time.

In general, you want to try to defer commitment as long as possible, but without incurring excessive run-time penalties. It's the static, compile-time dependencies that you have to watch out for when building large systems. And that's really what motivates me, trying to discover clever techniques for programming-in-the-large. If you don't understand all the issues, development of large software systems is difficult, if not impossible."

Matthew Heany

Charles: A Data Structure Library for Ada95

Matthew Heaney
April 30, 2002

Charles is a data structure library for Ada95, modelled principally on the C++ STL. It features both ordered (lists and arrays) and unordered (sets and maps) collections.

Associated with each data structure type is a separate iterator type, which allows you to visit each item in the container. In particular, an iterator abstracts away differences in specific container types, allowing you to view the collection simply as a sequence of items. A generic algorithm (for sorting, say) can be written in terms of an iterator, so that you can use the algorithm over any data structure having an it-erator with the requisite operations.

Data structures are categorized by their time and space semantics. A list has constant time insertion and removal of items, but searching for an item requires linear time. An array (”vector”) has constant time searching (”random access”), but insertions and removal require time proportional the length of the array. (A ”deque” is another STL data structure that supports constant time insertion of items at the either end of the sequence, and allows random access of elements. This data structure is planned for a future version of Charles.)

In addition to lists and vectors, the Charles library has set, multi-set, map, and multi-map data structure types. A map is an ”associative container” that allows you to index an item by some other arbitrary type. These data structures are extremely useful for testing membership of an item in a collection. In Charles all of these types are are implemented using a red-black tree, so finding an item requires only logarithmic time. (Note that even though sets and maps are traditionally ”unordered” data structures, for pragmatic reasons the item type used to instantiate the component must have a relational operator.)

Lists and arrays are ordered sequences, and you’re allowed to insert an item at a specific position in the container. There are special operations for inserting an item in the front or back of the container. Insertion of an item into a set or map is performed according to its order relative to the items already in the container.

Charles doesn’t have Stack or Queue container types specifically, because the ordered container types allow you to achieve the same effect by simply pushing and popping an item from the appropriate end. One data structure that is missing, however, is a priority queue; this is planned for a future ver-sion of the library. (You can use a set or map as a priority queue, but this is not as efficient, because a height-balanced tree requires more maintenance during insertions and removals.)

One of my design goals for Charles was that instantiation of components should be as painless as possible. My philosophy is that common things should be easy to do (and consequently that less common things should be less easy to do). The library has therefore been optimized for container items whose type is non-limited and definite. In general, creating a container type from a Charles component requires only a single instantiation, passing only the item type as a generic actual parameter.

Container types are declared as traditional abstract data types, not as tagged types. A theme of the library is that you create a complex abstraction by composing instantiations of generic components, not by extending a tagged type. (If polymorphism is required, then the client can use the Adapter pattern to provide that himself. For that reason I specifically rejected the approach used by the Booch components, which implements containers as a hierarchy of tagged types, requiring two separate instantiations.)

It is useful to have set elements and map indices that are indefinite (a symbol table, for example, is a map whose index type is a string). One solution is to simply require that clients use a definite type directly (type Unbounded String, say) when instantiating the data structure. However, in the absence of automatic type conversion (as you have in C++),

manipulation of such a data structure would be awkward. Therefore, given the importance of unconstrained array types in general, and type String in particular, in Charles I have provided additional set and map components that accept an indefinite generic formal type.

The container types in Charles have both bounded (stack-based) and unbounded (heap-based) forms. For the bounded forms I pass the size as a discriminant of the type. (I prefer to do it this way rather than passing the size as a generic formal constant.) The unbounded forms have a generic formal pool object (of type Root Storage Pool'Class), which is used for all internal allocation. Charles provides a pool object to simplify instantiation for those clients that don't have specific storage pool needs.

There is the orthogonal issue of whether items in the collection can be limited. On the one hand, we desire the library be as general as possible, and therefore we should support collections of limited items (to implement a queue of File Type, for example). On the other hand, this would require passing in a generic formal subprogram to assign items (because data structure types are themselves non-limited), which is in con ict with our goal to make instantiation as painless as possible. (A queue of integer items should require only that the integer type be passed as the generic actual type.)

My solution in Charles was therefore to have different versions of the data structure types, one for non-limited items and another for limited items. For limited items, the data structure type is itself limited. A operation for adding an item to the container doesn't actually accept an item argument; rather, it just creates a new "slot" for the item. A separate selector function returns a pointer to the item, so that the client can manipulate the actual item (not just a copy). This allows the component to be agnostic about whether the item is "really" limited. (Indeed, even for a type that has assignment, if the items are "large" it may be more efficient to store them as "limited" items in a limited container, as this prevents undesirable copying of items.)

When I began writing the library, I was unsure about whether to implement unbounded data structure types as controlled. One argument against is that you can add controlled-ness as necessary, at the point of declaration of the object. For example, you can implement a generic controlled helper type that calls a generic formal subprogram during its own finalization. You can use an access discriminant to bind the controlled helper object to the data structure object, and have it call an appropriate finalization operation. Another technique is that if the uncontrolled data structure is a component of some higher-level abstraction that is itself controlled, then the data structure can be manually finalized during finalization of the enclosing record.

However, in practice it's too easy to forget to finalize the data structure. It's one more thing to have to think about, and more likely than not you won't think about it. So it turns out to be easier to simply have the component automatically finalize itself. Another reason is that if data structures weren't controlled, then they'd have to be limited, in order to prevent structure sharing. But that would make it harder to create complex data structures comprising elements that are themselves simpler data structures.

The data structure types in the STL require separate iterator types for forward and backward traversal. This is because templates in C++ have the sense of type-safe macros: the text of the template is simply expanded inline, so the operations of the type have to be identically named. (For example, the increment operator ("++iter") moves forward for a forward iterator type, but backwards for a reverse iterator type.) In Ada, however, there is less coupling between the generic formal type and the generic actual type, and therefore only one iterator type is necessary. The only requirement is that the signature of an operation match, not the name. (That being said, it's awfully convenient that the name does match, because this greatly simplifies instantiation when formal operations are marked having a default. Charles always defaults all generic formal subprograms.)

My goal was to make iterators as simple, general, and efficient as possible, and that generic algorithms be easy to instantiate. This means passing only an iterator type during the instantiation, and taking advantage of "default" generic formal subprograms to implicitly pass operations directly visible at the point of instantiation.

There is lots of debate about how type-safe iterators should be. Should a container keep track of the iterators designating items in the container? What should happen if you try to remove an item from the container while it's being designated by an iterator? However, worrying about this would have complicated the design, and made everything else much less efficient. (In general, in the design of

Charles I have been willing to trade type-safety for exibility and efficiency.)

For unbounded forms, iterator implementation is relatively simple, and usually it's just a thin wrapper around an access type that designates a node of internal storage. I implemented iterator types for bounded forms exactly the same as for the unbounded forms, but unfortunately this means I lost the accessability checks that Ada provides when manipulating pointers to locally declared objects. I am not entirely satisfied with that choice, and it may very likely change in a future version of the library.

(For example, an iterator for a bounded form could be implemented as an integer array index, and iterator operations could accept both the data structure and the iterator. The operation would be implemented by using the integer index to dereference an array component. This means the signature for iterator operations of a bounded form would be different from an unbounded form, which means you wouldn't be able to use a generic algorithm directly. However, you could work around that by declaring local subprograms, that are implemented by calling the operations of the locally declared bounded data structure.)

It was also my goal that generic algorithms (such as a sort) also work directly on Ada array types, as is the case in C++. This can be effected in Charles by declaring local subprograms that look like iterator operations, and which operate on the locally declared array object. These subprograms are then implicitly passed as default actual subprograms to the instantiation of the algorithm, which is instantiated with the array index type as the "iterator" type.

In order to iterate over a range of items of a sequence, it is necessary to have a nonce item that acts as a sentinel to terminate the iteration. We refer to this as a "half-open" range, meaning that it includes the first element but not the last element. Unfortunately, this idiom con icts with Ada's higher-level approach, which uses a "closed range" that includes both the first and last elements. (In an earlier version the library I tried an iterator design that used a closed range, but there were certain cases that caused dangling iterator references. I have therefore concluded that the current half-open design is superior.) One consequence of the current design is that different sentinels are needed for forward versus backwards iteration over a half-open range, so the data structures in Charles have selectors that return "first" and "back" iterators for forward iteration, and "last" and "front" iterators for backward iteration.

Charles also includes an "access control" type, which is exactly analogous the the auto_ptr type in C++. This is extremely useful for those times when it's necessary to manipulate access objects directly (polymorphic programming, for example). The abstraction reifies the notion of the "owner" of an access object who can transfer ownership to another control object, or release ownership entirely. If the access control object still owns an access object at the time of its finalization, then it automatically frees the designated object.

Slutseminarium för FoTA-projekt den 17 dec -02

De system- och programvaruteknikprojekt som definierades 1998 inom FoTA (ett Forsknings- och Teknikutvecklingprogram avsett att stimulera svensk industris förmåga att stödja ett anpassningsförsvar) börjar nu närma sig sin fullbordan. Ett slutseminarium kommer att hållas den 17/12 2002 på FMV, Stockholm i Filmsal C. Program för seminariedagen kommer i november.

Inga-Lill Bratteby-Ribbing
AOL FoTA P12, ilbra@fmv.se, tel 018-12 02 63

Programvarusäkerhet och H ProgSäk – inbjudan till en introduktion, onsdagen den 4 december 2002

Bakgrund

H ProgSäk, Handbok för programvara i säkerhetskritiska tillämpningar, utgör FM och FMV: s rekommendationer vid anskaffning av programvara i säkerhetskritiska system. Handboken fastställdes i december 2001 att tillämpas från och med 2002-03-01.

Seminariet belyser olika aspekter på programvarusäkerhet (programvarans roll för systemsäkerheten) och ger en introduktion till H ProgSäk. Några korta diskussionsuppgifter ingår. Presentationen är en upprepning av tidigare seminariedagar detta år.

Målgrupp

Seminariedagen är ett led i KC Ledstöd:s kompetensutveckling av medarbetare inom FMV. Deltagare från FM, FOI, industri och högskolor är i mån av plats också välkomna.

Tid och plats

Seminariet genomförs 2002-12-04 kl 08.30 – 17.00 på FMV, Banérg 62, Stockholm i Filmsal C.

Anmälan

Anmälan lämnas senast 2002-11-29 till Christina Gunnarsson på KC Ledstöd (för externa deltagare: namn, företag samt personnummer) via mail, christina.gunnarsson@fmv.se , alternativt fax 08-782 5788. Deltagandet är kostnadsfritt.

Program:

Tider	Aktivitet	Ansvarig
08.30	Kaffe	
09.00	Inledning	*K-G Lövstrand, Chief Scientist*
09.15	Programvarusäkerhet – vad är det?	*Inga-Lill Bratteby-Ribbing, Strategisk specialist*
10.00	Diskussionsuppgift 1-2:Programvarusäkerhet, Återanvändning	
10.45	Tillförlitlighet och fel i programvara	*I-L B-R*
11.15	Diskussionsuppgift 3: Programvaruspecifika egenskaper	
12.00	Lunch	*Enskilt*
13.00	Programvarans roll för säkerhetskritiska system	*I-L B-R*
13.15	Diskussionsuppgift 4: Arkitektur	
13.30	Vad innehåller H ProgSäk? -Kravaspekter	*I-L B-R*
14.15	Diskussionsuppgift 5: Språk - OS	
14.30	H ProgSäk: Informativa avsnitt -Standarder	*I-L B-R*
15.10	Kaffe + Diskussionsuppgift 6: Felträd - Riskreduktion	
15.50	Hur skall H ProgSäk användas?	*I-L B-R*
16.15	Diskussionsuppgift 7 : Inför säkerhetskritiska projekt	
16.30	Framtid, frågor och diskussion	*I-L B-R*
17.00	Avslutning	

FÖRSVARETS MATERIELVERK
Per Magnus Wicén, C KC Ledstöd

Det extra sommar-seminariet satte deltagarrekord

Det extra sommarseminariet den 28 augusti med den högtflygande rubriken “Architecture Framework Evolution in the United States and Sweden” blev en stor publik framgång, med ca 150 deltagare, vilket t o m krävde lokalbyte.

Intresset var kanske inte helt oväntat då huvudtalaren Kathie Sowell från the MITRE Corporation är en central person i de amerikanska försvarsmyndigheternas strävan att få fram ett för DoD gemensamt arkitekturramverk och varit med om de flesta turerna i detta inte helt okontroversiella arbete. Att ett motsvarande arbete, fast i litet mindre skala, pågår här hemma för det nya nätverksförsvaret var naturligtvis grunden för den stora uppslutningen.

Sowell, som har en enorm erfarenhet i ämnet, stack inte under stol med att det fanns många olika åsikter om det överhuvudtaget var lämpligt att ha ett gemensamt ramverk och ännu fler om hur det skulle utformas och tillämpas. Hon tyckte dock att de inledande erfarenheterna av tillämpning av det hittillsvarande C4ISR Architecture Framework i flera sammanhang varit positiva och motiverade fortsatta insatser på en bredare front.

Sowell är också inblandad i och berättade om några av de civila amerikanska myndigheternas ansträngningar att få fram arkitekturramverk, vilka till del bygger på erfarenheterna från DoDs arbete. Därför fanns det en del deltagare på seminariet som vi vanligen inte har nöjet att se på SESAMs övningar.

Den andre föredragshållaren Johan Bendz från FMV beskrev mycket målande det mycket ambitiösa arbetet med att ta fram ett Försvarsmaktens Arkitekturramverk, FMA. Detta ramverk visade sig i många intressanta avseenden skilja sig från den amerikanska motsvarigheten. Här finns ett rikt fält för intressanta framtida diskussioner.

Även SESAM-iter som till äventyrs inte kunde vara med på seminariet kan studera presentationerna i lugn och ro på den nya SESAM 2002 CD som kommer ut efter höstseminariet om några veckor.

Ta gärna en titt på vår
websajt

http://sesam.tranet.fmv.se

där finns konferenser för hela året uppräknade

SESAM-Sekretariatet:
AerotechTelub AB
c/o Kåsjös Kontor
Ytterspåret 14
187 54 TÄBY

Telefon: 08-510 51866
Telefax: 08-510 51932
GSM: 070-716 9702
E-post: alkas@tranet.fmv.se
kasjos.kontor@telia.com