EFFEKTIV IT

SYSTEMARVET

RAPPORT NR 8 – JUNI 1994

KUNSKAP FÖR HANTERING AV SYSTEMARVET

– en första systematisering

Lars-Åke Johansson
Mats R Gustafsson
Roland Dahl

SVENSKA INSTITUTET FÖR SYSTEMUTVECKLING
SISU

Innehåll

1 Inledning

1.1 Bakgrund

SISU genomförde under våren 1993 en undersökning bland AU-chefer i ett antal större och medelstora företag och förvaltningar i Sverige [28], där man frågade respondenterna vilka viktiga problem de ser på IT-området och vilka som är viktiga att komma till rätta med den närmaste tiden.

De två viktigaste problem man nämnde i svaren var dels systemutvecklingens alltför långa ledtider och dels att man måste komma tillrätta med det omfattande systemarv som man nu har i organisationerna.

Dessa två problem hänger delvis samman. Långa ledtider och dåliga erfarenheter av nyutveckling gör att man får ökat intresse för att utnyttja befintliga system. Detta trots att nya verksamhetskrav gör att kraven på informationssystemförändringar är förhållandevis starka.

Man indikerade starkt i undersökningen att man måste öka kunskapen för att hantera det systemarv man har idag. Det ligger alltför stor kunskap i de investeringar som gjorts i existerande system, för att det skall vara möjligt att bara kasta bort dem. I praktiken kan man så gott som aldrig utgå ifrån att det inte finns något befintligt informationssystemstöd, som man, på ett eller annat sätt, måste ta hänsyn till. Man måste därför börja bygga kunskap kring dessa frågor och göra det NU.

Det har blivit för dyrt att skapa nya system när man känner att informationsbehoven i verksamheten ändras. Man måste lära sig att ta vara på det existerande.

Intresset för systemarvet har således delvis sin grund i att byggandet av helt nya system är förknippat med risker. Mycket av den kod som framställs idag kommer aldrig i produktion.

Nybyggnation blir dessutom ofta dyrare än vad som beräknats och utvecklingstiderna blir ofta längre än vad som planerats. Dessa orsaker medför att intresset för att ta vara på existerande system, genom att **FÖRNYA** dem, ökar.

1.2 Om denna skrift

Denna skrift är ett första försök att strukturera olika kunskaper och hjälpmedel, som håller på att växa fram som ett svar på kravet att ta vara på existerande system på ett bättre sätt.

Ett viktigt underlag för rapporten är intryck från konferensen 3rd Reverse Engineering Forum, 15 - 17 september 1992, USA [55]. Denna konferens, med temarubriken "Understanding enables Reengineering", bjöd på ett stort antal föredrag med två olika "tutorials". En mängd auktoriteter deltog i konferensen – både från forskningen och från tillämpare i stora företag.

Intryck finns också från två andra resor som SISU gjorde till USA hösten 1993 och påföljande vår, då ett antal företag besöktes, både företag som utvecklar och sådana som använder informationsteknologi.

Skriften bygger dessutom på direkta samtal med svenska företag och förvaltningar om deras situation på området. Många av dessa samtal har genomförts i samband med Effektiv IT-programmets första år.

Vidare bygger skriften på samtal med internationella föreläsare och auktoriteter på området, t ex Michael L. Brodie, som kommit att arbeta med området "legacy systems", bl a tillsammans med amerikansk industri [7], [8].

En del litteratur inom området har också studerats. För en litteraturförteckning, se "Referenser och litteraturförteckning" sid 41.

2 Varför är systemarvet intressant?

2.1 Orsaker till varför intresse har uppstått

Det framförs från många informationsteknologitillämpande organisationer att det är viktigt att arbeta med systemarvet. Några viktiga skäl till detta kan vara följande:

1. Företag och förvaltningar har i mindre utsträckning ansett sig kunna allokera resurser för nyutveckling, vilket medför att man i mindre utsträckning kommer att satsa på helt nya system. De gamla systemen måste vidareutvecklas så gott det går. Man måste kunna använda de system och de systemdelar man har för att skapa det nya, förändrade och moderniserade informationssystemstöd för verksamheterna som behövs. Anledningar till detta är att underhållet är för dyrt idag samt att nyutveckling också är dyr och sällan kan hållas inom den budgeterade kostnadsramen.
2. Mängden och utbredningen av befintliga system i verksamheterna och som stödjer olika verksamhetsfunktioner är idag så omfattande att man i princip inte kan utveckla nya system ifrån "scratch", utan att man måste ta hänsyn till systemarvet på ett eller annat sätt. Detta antingen genom att det finns system som skall ersättas eller förändras, eller genom att det finns gränsytor till intilliggande informationssystem som man måste ta hänsyn till, t ex vad gäller informationskommunikation och tidskrav.
3. Företag och förvaltningar upplever att man måste hitta sätt att ta sig an arvet, men att man inte har instrument för detta. Man vill hitta sätt att komma igång med detta arbete.

I samband med 1) ovan formulerade man det så på 3rd Reverse Engineering Forum [55], att "det ökande antalet system i organisationerna skapar ett växande krav på underhåll till så stora kostnader att man "stjäl" resurser ifrån den nyutveckling som hade behövts för att möta de nya krav på förändringar som ställs allt snabbare i affären och på marknaden."

2.2 Hur har man byggt system?

En av grundorsakerna till att kostnaderna för underhåll av system har varit, och är, mycket stora torde vara att de är dåligt beskrivna.

Hur kommer detta sig? Det verkar ha funnits en tid då man inte använde så mycket "engineering-tänkande" i byggnationen av informationssystem. Man ansågs "produktiv" när man så snabbt som möjligt kom igång med programmering av planerade system samt producerade så stor mängd kod som möjligt.

Att systemen är dåligt beskrivna har resulterat i att endast vissa människor har känt till hur systemen har varit byggda. Det har tagit lång tid för andra personer än för dem som byggt dem att sätta sig in i hur systemen är konstruerade – och än mer att ändra i dem på ett någorlunda säkert och förutsägbart sätt.

3 Begrepp och taxonomi

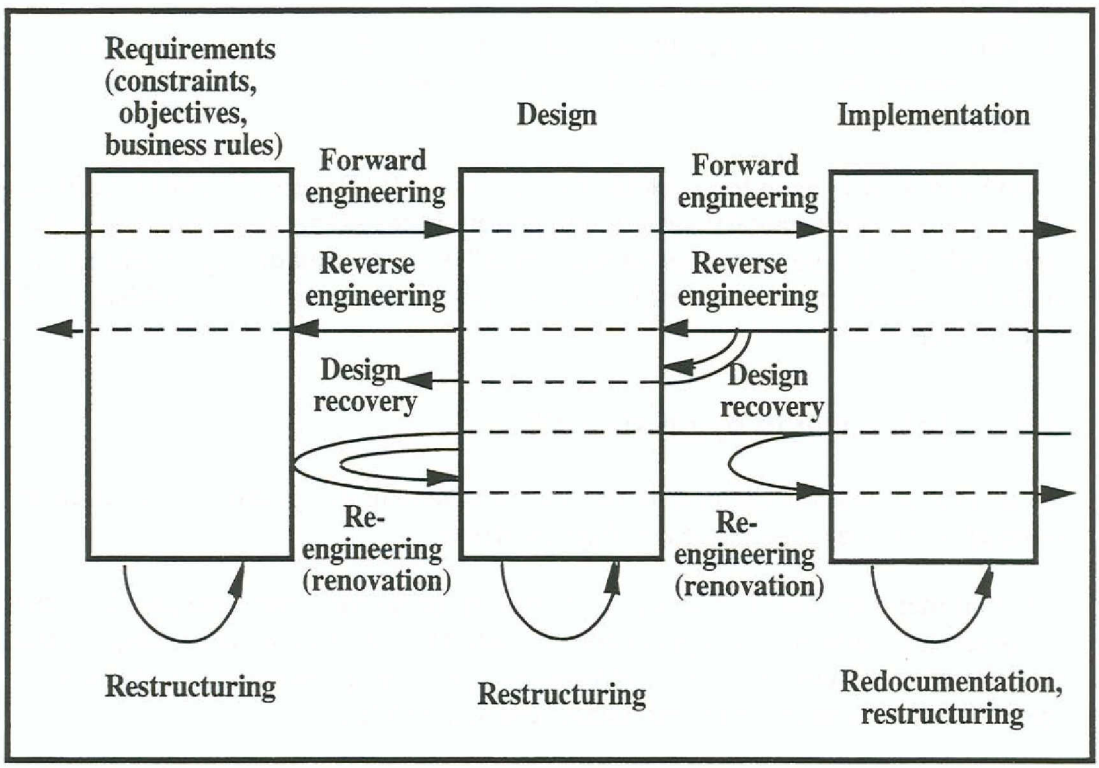
Eftersom området reverse engineering/reengineering är relativt nytt, så är det viktigt att definiera några av de begrepp som förekommer. Terminologin är till stor del hämtad från [13]. Eftersom det skulle vara välkommet med svenska översättningar, har vi även försökt göra sådana där det varit möjligt. Vissa begrepp är mer svåröversatta och vi har inte hittat någon bra svensk motsvarighet.

Software maintenance (Programvaruunderhåll/systemunderhåll)

Modifiering av programvara efter leverans för att rätta fel, för att öka prestanda eller andra egenskaper eller för att anpassa programvara till en förändrad omgiv-ning.

Forward engineering (Systemutveckling)

Den traditionella processen att flytta sig ifrån högnivåabstraktioner och logiska, implementeringsoberoende specifikationer till fysisk implementation av ett system.



(from Chikofsky&CrossII: Reverse Engineering and Design Recovery: A Taxonomy, IEEE Software,Jan 90)

Figur 1. Relationer mellan begrepp

Reverse engineering

är processen att analysera ett system i avsikt att

- identifiera systemets komponenter och deras relationer, samt att
- skapa representationer av systemet i annan form eller på högre abstraktionsnivå.

Reverse engineering betyder alltså, att man *inte* ändrar på det studerade systemet eller skapar något nytt system. Man *analyserar* enbart systemet.

Redocumentation (Omdokumentation)

Skapande eller revision av semantiskt likartad representation på samma relativa abstraktionsnivå.

Den resulterande formen av representation betraktas vanligtvis som alternativa vyer avsedda för människor (t ex transformation ifrån ett program till ett dataflödesdiagram).

Restructuring (Omstrukturering)

Transformation från en representationsform till en annan på samma relativa nivå av abstraktion, med bevarande av systemets externa beteende (funktionalitet och semantik).

Exempel här är olika typer av kod-till-kod-transformeringar, omstrukturering av datamodeller och datanormalisering.

Design recovery (Specifikationsåterskapande)

En delmängd av reverse engineering där domänkunskap, extern information, härledning och resonerande läggs till observationerna av systemet för att hitta högre nivåer av abstraktioner, över de nivåer man får om man bara undersöker systemet självt.

Reengineering

Analys och förändring av systemet för att rekonstituera det i en ny form och den efterföljande implementationen av den nya formen.

Detta innebär, att reengineering innefattar någon form av reverse engineering följd av någon form av forward engineering eller restructuring. Andra namn för reengineering är "renovation" och "reclamation". "Redevelopment engineering" är IBMs term för reengineering.

Normalisering av en databas är ett exempel på reengineering. Det är en strukturell förändring.

Reverse engineering behöver inte i och för sig betyda att man använder sig av ett "repository". Att leverantörerna av CASE säger att reverse engineering innebär att

man endast "letar rätt på saker och stoppar ner dem i ett repository", raljerades det mycket över på 3rd Reverse Engineering Forum [55].

Två viktiga instanser har skapat definitioner som många accepterat: IEEE och Guide Group.

Några andra begrepp som vid konferensen ansågs vara viktiga att snarast få fram skarpa definitioner av:

- Program understanding (programförståelse)
- Application understanding (applikationsförståelse)
- Software visualization (programvisualisering)
- Business rule (verksamhetsregel).

Det är viktigt att inse att reverse engineering är en analysprocess. Medan reengineering/ redevelopment är en förändringsprocess.

3.1 Reverse engineering och reverse modeling

Man kan möjligen skilja på reverse engineering och reverse modeling. Det förstnämnda kan eventuellt innebära ambitionen att skapa en bättre struktur för de existerande systemen och en plattform för att kunna utföra effektivare successiv systemutveckling och förvaltning.

Reverse modeling skulle kunna vara att skapa just modeller av olika aspekter av existerande system. Dessa kan finnas på en högre verksamhetsorienterad nivå, så att man förstår vad som hanteras i systemets databaser och program. Dessa modeller kan vara av en mängd olika slag.

Man bör nog egentligen betrakta reverse modeling som en delmängd av reverse engineering.

4 Varför reverse engineering?

4.1 Varför hoppas man på reverse engineering?

Förmodligen är det så att området, när det vuxit fram, har haft en för snäv syn på reverse engineering. Det finns relationer till andra områden som borde fokuseras och utarbetas närmare. Området har dominerats av kompetens som strukturerar program som en isolerad företeelse, vilket inte är realistiskt i ett större informationstillämpande företag. Området reverse engineering måste kopplas ihop med varför och i vilket syfte reverse engineering-insatser utförs. Detta i ett verksamhetsperspektiv. Ett första försök att strukturera sådana verksamhetsmässiga perspektiv har gjorts i rapporten [30].

Vid konferensen 3rd Reverse Engineering Forum framhölls nedanstående orsaker som dem som driver marknaden för reverse engineering och reengineering:

- Underhållet kostar för mycket. 80% av informationssystemkostnaden är underhåll av existerande system (Wall Street Journal).
- Företagen förstår inte sina existerande system.
- Man behöver få sina existerande system att möta nya affärsmål.
- Existerande system bör avkasta bättre genom att utnyttjas på nya sätt. (T ex stora mjukvaruleverantörer som måste utnyttja sina system mot nya kunder och i delvis nya situationer).

I SISU:s undersökning [28] framstod inte första punkten som den viktigaste. Snarare var det andra orsaker, som den tredje punkten, som förekom mer i svaren. Företagen vill förändra sina system i en viss riktning, vilket man finner svårt inte minst därför att existerande system lägger restriktioner.

Punkt 2 är snarare ett följdproblem som man upptäcker och som man måste ge sig i kast med då man vill förändra sina system i en viss riktning.

4.2 Vad kan räknas till reverse engineering?

Att "scanna in" ett dokument och arbeta vidare med det som en beskrivning eller ett bakgrundsmaterial är egentligen en reverse engineering-aktivitet.

Man kan möjligen också hävda att så fort man tittar på och försöker förstå någon annans program bedriver man enligt definitionen också reverse engineering.

Reverse engineering avser att hjälpa oss att

- hantera komplexitet
- generera alternativa vyer - att se saker ifrån olika perspektiv
- återskapa dokumentation (information om systemen)

- upptäcka sidoeffekter (saker som vi inte visste om systemen – fel eller andra diskrepanser mellan uppfattningar i verksamheten. Samt vad systemen har för uppfattningar (se också verksamhetsregler))
- skapa högre abstraktioner av system vi har
- underlätta återanvändning
- förstå verksamheten bättre (vilka verksamhetsregler vi egentligen har och vilka verksamhetsregler systemen arbetar med), eller diskrepanser mellan vad verksamheten är och vilken uppfattning systemet har om detta.

4.3 Kvalitet på beskrivningar

Att skapa dokumentation har hittills ofta upplevts som ett extraarbete som inte varit en naturlig del av skapande- och konstruktionsprocessen. Det har inte funnits en tradition att beskriva det som skapas för att underlätta att kunna göra ändringar och bygga ut. Detta har medfört att beskrivningar av system ofta varit mycket knapphändiga, tagit upp fel saker och varit otydliga.

Om det råkar finnas beskrivningar när systemen är nya, degenererar de snabbt då ändringar utförs utan att dokumentationen samtidigt också ändras. Programmeraren kan ta genvägar, vilket ytterligare reducerar möjligheterna till relevant dokumentation. Detta kan medföra att beskrivningar inte alltid är pålitliga trots att dokumentationsdisciplinen kan ha varit god.

En realistisk avsikt måste vara att dokumentation och beskrivningar hänger bättre samman med vad som skall beskrivas, så att man finner det naturligt att använda beskrivningarna när man gör den substantiella ändringen. Det kan lösas på flera sätt. Ett sätt är att man utför alla ändringar via beskrivningarna och sedan från dessa genererar körbara strukturer som används för att exekvera systemet.

Ett annat sätt är att man aktivt kopplar beskrivningarna till de körbara komponenterna så att man lätt kommer åt dem när ändringar skall göras. Maskinella hjälpmedel kan användas här. Maskinella kontrollstöd kan finnas så att förändrade beskrivningar blir konsistenta. Detta bör i sin tur leda till att beskrivningarna upplevs som meningsfulla när det gäller att hitta fel m m.

4.4 Att flytta system mellan plattformar

Att flytta system mellan plattformar sliter ofta ner systemen. Detta speciellt om inte särskilda åtgärder vidtagits för att undvika plattformsberoende. Plattformsberoende delar finns kvar i koden. Och när man kommer över på den n:te plattformen förstår inte läsaren längre varför viss hantering görs i programmen. Man vågar därför heller inte röra i de delar man inte förstår.

Det blir därför angeläget att bygga upp kunskap om applikationsskapande till en utrustningsoberoende nivå. Detta underlättar flyttning av applikationer mellan plattformar.

En mängd standards växer fram och tar sikte på att man skall kunna bli utrustningsoberoende så att hindren för att flytta applikationer minskar. CORBA[1], CALS[2], Motif, m fl, är exempel på sådana standards. Möjligheterna ökar alltså att skapa applikationer som har en distinkt gränsyta mot plattformen, så att man slipper ändra på för många ställen när man flyttar sina applikationer. Detta förutsätter dock att man skaffar sig god kunskap om vad dessa standards innebär, vilket tänkande som ligger bakom dem och hur man skall använda sig av dem för att skapa de arkitekturer man vill ha.

4.5 Har reengineering sin höjdpunkt nu?

Det förefaller som om man just nu går in i ett skede, där ett ökande intresse uppstår kring reengineering och dess tillämpning.

Detta kan ha sin grund i att många företag känner ett akut behov av att gå igenom och förbättra en mängd system med stora kodmassor skapade för ett antal år sedan.

Man kan naturligtvis ställa frågan om man om 10 år kommer att genomföra lika stora reengineeringsinsatser som nu? Eller får man en annan utveckling där förbättrade beskrivningssätt och hjälpmedel medför att man kommer att utföra ändringar och vidareutveckling genom att endast arbeta med maskinellt hanterade specifikationer? Från vilka man skapar exekverbara program helt automatiskt? Detta skulle möjligen från rationell utgångspunkt kunna innebära att arvsproblematiken blir något mindre besvärande över tiden. En mängd förutsättningar måste dock finnas för denna utveckling. Inte minst på det kunskapsmässiga planet.

På 3rd Reverse Engineering Forum uttryckte man det så att man egentligen inte vill ha 50-60% underhåll som idag, utan 100% underhåll. Med det menade man ju att man förmodligen inte kommer att utveckla system från scratch. Utan att man istället kommer att arbeta med evolutionär utveckling, där man förändrar system steg för steg. Varje steg kan då vara en del i ganska stora omstruktureringar. Underhåll och vidareutveckling blir då också mer sammakopplade än idag.

1. The Common Object Request Broker Architecture, (OMG – Object Management Group)

2. Computer-aided Acquisition and Logistic Support, (US Department of Defense i samarbete med industriföretag)

5 Förväntningar på reverse engineering/ reengineering

5.1 Förväntningar

Det finns många olika förväntningar kring reverse engineering/reengineering. Så är det ofta när det dyker upp någon ny teknologi där det finns stora behov och problembilder. Jämför till exempel med CASE och CASE-teknologin.

Inför en insats kan det vara väsentligt för en organisation att göra en analys av vilka förväntningarna är:

1. Säkerställ och systematisera vilka förväntningarna egentligen är.
2. Analysera hur förväntningarna förhåller sig till varandra.
3. Analysera vilka förväntningar som finns möjligheter att uppfylla.
4. Analysera vilka reella effekter som kan erhållas genom olika insatser.

Problemen bakom intresset för reverse engineering ligger förmodligen i ett akut behov att göra något åt systemarvet. Kostnaderna för att bedriva underhåll och vidareutveckling av existerande informationssystem är alldeles för höga.

Man söker således efter något radikalt som kan ändra på dessa förhållanden. Kanske något som skapar effekt utan alltför stora uppoffringar. Är detta möjligt?

Enligt 1) ovan skall vi försöka sätta upp ett antal förväntningar som vi noterat. Enligt 2) skall vi också analysera hur olika förväntningar hänger ihop.

Hos vissa tillämpare är förväntningarna ibland oklara, eller oklart uttryckta, varför vi nedan också försökt göra dem mer explicita genom viss uttalad förväntan uppdelats i flera punkter. Detta aktualiserar naturligtvis också frågan huruvida olika förväntningar hänger ihop i "nivåer".

- Man väntar sig genom verktygen kunna få goda och uttrycksfulla modeller som hjälper en att bedriva effektiv och väldokumenterad vidareutveckling och justering av existerande system.
- Skapa mig en beskrivning av hur en viss databas ser ut och vad den består av!
- Man vill i en "migreringsprocess", bättre veta vad som finns i den gamla och i den nya systemstrukturen. Så att man på ett kontrollerat och successivt sätt kan plocka över komponenter och data från det ena systemet till det andra.
- Beskriv inte bara hur en databas är strukturerad, utan också vilka begrepp som hanteras i databasen och vad man menar med dem i verksamheten.
- Att man skall få veta vad ett antal program gör, vilka data de hanterar samt hur programmen anropar varandra.

- Man vill beskriva inte bara vad ett program gör och vilka data det hanterar, utan också vilka "verksamhetsregler" som hanteras av programmet. Det innebär att man i verksamhetstermer beskriver vilken regel, riktlinje o s v som ligger bakom en viss operation i ett program.
- Att man skall kunna skapa goda beskrivningar av systemkomponenter så att man kan komma åt dessa för effektivare återanvändning vid vidareutveckling och förändring av system. Detta innebär bl a att man får gå igenom systemet så att man tar bort dubbleringar av systemkomponenter.

I skriften [30] har vi försökt ställa upp olika verksamhetsmässiga perspektiv som kan alternativt bör, motivera olika förväntningar mer än andra.

5.2 När skall man göra reverse engineering?

I ett av föredragen vid 3rd Reverse Engineering Forum [55] försökte någon strukturera olika skäl för att påbörja reverse engineering.

Detta kan gälla, när

- man "får ut mer än man sätter in",
- man kan återskapa förlorad information om system och verksamhet,
- man verkligen skall använda den information man hittar,
- man kan se systemdelar och beskrivningar i nya perspektiv,
- man kan hitta dolda effekter och möjligheter,
- information blir tillgänglig för ny användning,
- man kan skapa återanvändning,
- man kan genomföra åtgärder på basis av det man hittar och
- man inte har något annat val (t ex när konstruktörerna har slutat).

5.3 Vänta inte med arbetet

En föredragshållare vid konferensen [55] uttalade mer skarpt och obetingat att:

> Man bör göra reverse engineering på sina kritiska system nu! Man måste börja att bygga upp sin kunskap.

Det finns en mängd tidsbomber i våra nuvarande system i form av datum för 2000-talet. Man har i stor utsträckning bara använt 2-siffrig representation för årtal. Orsaken till att man inte tog hänsyn till detta redan från början kan vara att systemen inte var avsedda att användas så lång tid som sedan blivit fallet.

Starta emellertid inte reverse engineering blint. Ta utgångspunkt i vad som skall lösas för verksamheten och affären. Skaffa en strategi för att gå fram successivt. Dela upp reverse engineering i delar och ta en del i taget. Strategin skall hålla ihop det hela så att målen kan hållas i sikte.

5.4 Några andra synpunkter på och problem med reverse engineering

Det har rapporterats svårt att göra reverse engineering på 4GL-program. Satsar man på detta skall man således göra klart för sig hur man skall bedriva sin reverse engineering, om sådan behöver tillämpas.

Att gå till objektorientering är ingen enkel konvertering. Det är att gå över till en helt ny teknologi.

Många framhöll vid konferensen [55] att det är viktigt att kunna ha tillgång till meta-schemat för det respository man använder (s k open repositories). Detta skapar också möjlighet att bygga bryggor till andra verktyg med speciella gränssnitt. Detta sedan analys gjorts av begrepp i sändande och mottagande repository. Man kan förutsätta att man måste kunna utöka och byta verktyg i sin reverse engineering-miljö.

När man tar strukturer från existerande system och skapar bättre modeller, är det viktigt att man kopplar ihop databeskrivning med programbeskrivningar. Man måste kunna koppla ihop t ex en IMS-databeskrivning med en programbeskrivning av data och dessutom skapa ett "business data name", vilken kanske läggs i en begrepps- alt. datamodell.

När man bedriver reverse engineering är det inte helt triviala mängder beskrivningsdata man får fram. I ett fall rapporterades att 650.000 rader COBOL i 650 program angreps med reverse engineering. Det hela resulterade i 130.000 COBOL data items, som blev 30.000 element i repositoriet.

När man analyserar data i program finns där ofta en stor mängd redundans. Man rapporterade [55] att 75% av datadefinitionerna kunde reduceras genom reverse engineering med tillhörande analys.

6 Metodaspekter på forward och reverse engineering

6.1 Ihopkopplade modeller för reverse och forward engineering

Forward och reverse engineering hänger starkt ihop. Man gör ju ofta reverse engineering för att ta reda på hur systemet ser ut nu, avgör vad systemet bör göra och på vilket sätt, för att sedan gå "forward" för att realisera förändringarna på ett målinriktat sätt.

Mycket av den kunskap om metodik och beskrivningstekniker som finns idag kan användas både i reverse och forward riktning. Det är en potential som utnyttjas och fokuseras i alltför liten utsträckning.

Man kan bli tvungen att backa olika långt för att få med sig de nödvändiga förutsättningarna avseende verksamhet och system, presenterade i beskrivningar och på annat sätt, för att kunna gå framåt igen.

Alan Kortesoja, Ernst &Young [35], presenterar en modell för reverse engineering som går motsatt mot forward engineering och som samspelar med denna genom utpekade delområden.

I en särskild metodrapport planerar vi att i Effektiv IT presentera litet mer av vad som finns tillgängligt i form av metodik på området.

6.2 Deltekniker i reverse engineering

Att arbeta med reverse engineering handlar i mångt och mycket om att **förstå** de system och de data man har. Den egentliga inriktningen på en reverse engineering-insats blir i grunden att förstå dels vad progammen egentligen gör, dels vad de data man innehar egentligen betyder.

För att t ex ta reda på vad ett program gör studerar man vilka data det hanterar samt vilken behandling progammet utsätter data för. En mängd deltekniker kan ingå för att nå dit man vill. Arbetet karakteriseras av att man måste växla mellan att använda datorstödda verktyg, fundera på vad som egentligen skall utföras i verksamheten samt försöka förstå vad kryptiska dataelementnamn egentligen betyder.

Verktygen kan via textsträngar söka i programmassor eller datascheman. Men det kräver mänsklig fantasi för att utröna vad kryptiska namn egentligen står för.

Det svåra är att utföra reverse engineering på ett strukturerat sätt, så att olika åtgärder är adekvata med tanke på vad man vill åstadkomma. Dessa åtgärder måste anpassas beroende på vad den affärsmässiga målbilden ytterst är för reverse engineering-insatsen. Viktiga typer av insatser i reverse engineering är, på ett grovt plan:

- Infångning (detta kan man ofta få hjälp med via verktyg idag)
- Analys

- Rapportering
- Syntetisering
- Lagring (i nya strukturer under nya begrepp och med nya namn m m)

På nästa plan kan man tala om ett antal deltekniker eller aktivitetstyper som ofta blir aktuella vid reverse engineering. De kan vara:

- Dokumentation och diagramgenerering
- Repository-browsing
- Dekomponering (inklusive program-slicing)
- Omstrukturering (med bevarad semantik)
- Identifikation av abstraktioner
- Identifiering av återanvändningsmöjligheter
- Återskapande av design
- Integration med CASE-teknologin
- Strukturella observationer
- Likheter med andra "constructs"
- Undersökning med navigering
- Applicering av domänkunskap

Hur mycket kan vi återskapa? En misstolkning kan hålla i sig och sätta spår i ytterligare tolkningar.

Analys är det viktiga i reverse engineering. Det är grunden för reengineering.

Reverse engineering har starka kopplingar till kognitiv psykologi, t ex vad gäller hur man förstår program. Du kan t ex inte obetingat anta att programmeraren som konstruerat ett program tänker som du.

Det är viktigt att skaffa fram verktyg som kan förstå strukturer i program. Här råder brist idag. Det är också viktigt att skaffa verktyg som kan underlätta de uppgifter som skall utföras. Det kan t ex gälla att finna strukturer i program, men just på denna punkt är det svårare att hitta verktyg.

Enkel keyword-analys kan vara viktig liksom enkla kontroller – det kan ge mycket.

6.3 För stora krav på gamla system

Gamla system var ofta enkla och överblickbara när de skapades. Sedan dess har de blivit alltmer komplexa. Den design som eventuellt fanns har försvunnit – eller har visat sig olämplig. Därför måste man ibland göra reverse engineering för att genom reengineering i olika avseenden förbättra systemen.

6.4 Ett komplext problem

Att ta hand om och förnya stora informationssystem kan vara en mycket komplex uppgift. En väsentlig fråga är hur man kan reducera denna komplexitet och därmed omfattningen av arbetet.

Förvånansvärt nog har det visat sig, att i de flesta system finns ofta relativt stora delar som aldrig utförs. Dessa delar behöver man naturligtvis inte bygga om – det finns ingen anledning att migrera systemdelar som inte längre används.

Det kan dessutom vara så att vissa mindre viktiga delar som körs mycket sällan rent av kan avvaras och att man också på detta sätt kan hålla komplexitet och migreringskostnader under kontroll.

I vissa andra fall behöver man kanske inte göra reverse engineering på alla system. Vissa system kan "ticka och gå" som "black boxes" om man inte behöver göra stora förändringar i dem, även om de har dålig struktur. Dessa kan sedan bytas ut efterhand genom att man skapar helt nya systemdelar under en längre tidsperiod.

Man bör alltså ställa sig dessa frågor, som syftar till att reducera komplexitet och omfattning, innan man sätter igång att förändra och migrera systemdelar.

6.5 Att bedriva redevelopment

Bland föredragshållarna vid 3rd Reverse Enginering Forum framhölls att reverse engineering och redevelopment ofta är viktigare alternativ till nyutveckling än vad man många gånger föreställer sig. Jämförd med reverse engineering är nyutveckling ofta (också) högriskprojekt. Nyutvecklade system blir ofta försenade och dyrare än beräknat och systemen kommer ibland aldrig i produktion. Enligt statistik i USA tas 50 % av all nyutveckling aldrig i bruk.

Exempel på redevelopment kan vara:

- Olika former för applikationsintegrering
- Ersättning/Återanvändning
- Migrering till I-CASE
- Migrering till viss databasteknologi
- Client Server/Downsizing

När man vidtar redevelopment måste man göra klart för sig vilken den existerande arkitekturen är och vilken den planerade arkitekturen är. Den planerade arkitekturen måste definieras väl.

Man talar om flera olika steg när det gäller reverse engineering/reengineering: först en inventerande analys ("Inventory analysis"), därefter skapar man en plan baserad på verksamhetsbehoven. I nästa steg skapas en migreringsplan, baserad både på kortsiktiga och långsiktiga behov. Positionering utförs, vilket innebär att man tar reda på var man är idag. Vilken är den nuvarande arkitekturen? Vilken är den arkitektur som

man vill gå emot? Hur ser nuvarande arkitektur ut i förhållande till planerad? Transformering: hur börjar man bygga upp den planerade arkitekturen? Vilka är delstegen för att nå dit? Man genomför delstegen. Validering: kom man fram till den nya arkitekturen? Uppfyller den planerade strukturen de behov som finns?

Reverse engineering och redevelopment med migrering mot en ny arkitektur kan ibland orsakas av att en viss teknologi blivit föråldrad.

Man bör starta migrering mot ny teknologi innan man blir tvingad till det. Då kan man göra det under bättre planerade och kontrollerade former.

6.6 Vilka program skall man göra reverse engineering på?

Det visar sig vid studier att 20 % av programmen ofta utför 80 % av funktionerna. (Den gamla 80-20 regeln verkar gälla även här.) Andra källor t ex [8] rapporterar att det i många system finns en viss andel program som aldrig används. Om man tvingas göra reverse engineering på delar av systemet kanske man i första hand bör välja de program som används mest och som man dessutom gör många underhållsändringar i.

6.7 Att plocka isär och sätta ihop system

Vissa insatser för reengineering kan genomföras genom att man explicit plockar isär existerande system och sedan strukturerar och sammanfogar dem bättre.

Bell rapporteras ha plockat isär ett system. Men fick dock aldrig ihop det igen. Det finns en mängd verktyg som gör relativt små deluppgifter i detta arbete. Man måste sätta samman dem i ett paket så att man använder delverktygen på rätt sätt. Troligen var det bristen på sådana åtgärder som Bell stupade på.

Reverse engineering kan ibland beskrivas som isärplockande av ett system. Detta måste följas av ihopsättning. Dessutom av att förbättrad struktur snyggas upp. Ihopsättningen kräver mycket manuell kraft. Verktygen gör ganska litet i detta sammanhang.

6.8 Begrepps/datamodellering och reverse engineering

De flesta verktyg för analys av existerande system rör programmen i systemen. För analys av data och hur de hanteras i program och lagras i datalager finns det färre verktyg.

Informationsanalys, datamodellering och datadefinitionsanalys verkar vara högst förbisedda aktiviteter inom reverse engineering. De kan ge stora effekter både vad det gäller att analysera program och att analysera och förstå de datalager man har.

Ofta är det ju lagrade data som representerar det stora värdet vid jämförelse med värdet av program.

I många gamla program har man ofta mycket dåliga datanamn. Detta leder till svårigheter att förstå vad programmen egentligen gör.

Bachmans verktyg kan medverka i arbetet att skapa datamodeller från datadefinitioner. Detta möjliggörs genom att en hel del "AI-stödd" analys görs, vilket uppfattas som en av fördelarna med verktyget.

7 Att skapa beskrivningar av existerande system

Reverse engineering omfattar till stor del att skapa beskrivningar av existerande system. Ibland kan det finnas dokumentation, men den är då felaktig. Missledande dokumentation kan ibland vara värre än ingen alls.

7.1 Systemarkeologi

Reverse engineering jämförs ibland med arkeologi. Man försöker på olika sätt att finna ut vad som döljer sig i program och bland symboler (data).

I detta undersökningsarbete ingår en stor portion varsamhet och eftertanke. Bl a bör man vara mycket noggrann med i vilken kontext man granskar ett program. Man kan t ex fråga sig: I vilken miljö har programmet verkat? Kan omgivningen hjälpa till att förklara vad som egentligen görs i programmet?

Begreppen "program mining" och "data (base) mining" hör också till detta område.

7.2 Olika nivåer på beskrivningar

När man beskriver vad som finns i en fysisk databas och vad som behandlas i ett program, anges data på ett sätt som syftar till att man håller rätt på data för att dessa skall behandlas av en dator.

Att BESKRIVA data så att en människa förstår vad de BETYDER när han eller hon skall läsa dem är en helt annan sak. Man behöver två nivåer av beskrivningar: en verksamhetsnära nivå och en datateknisk nivå. Detta hävdades också av föredragshållarna på 3rd Reverse Engineering Forum.

7.3 Beskrivning av data i program

Program behandlar data. Om man vill ta reda på vad ett program verkligen gör kan man granska de data det behandlar.

I olika programspråk deklarerar (beskriver) man på olika sätt de data som skall användas. (Om man har en bra begrepps/datamodell kan man använda denna för att i ett program definiera de data det skall använda.)

I olika programspråk blir definitionerna olika explicita. För att man skall kunna göra reverse engineering på ett program, eller om man vill återanvända det för nya ändamål, måste man bl a veta vilka data det behandlar. I COBOL beskriver man data i "data definition". Här beskriver man data relativt samlat, men deklarationerna präglas av hur filerna skall behandlas, i vilken ordning de skall behandlas o s v. Datatekniska aspekter som filstrukturer (t ex sekvensiella filer) spelar en stor roll för varför datadefinitionerna ser ut som de gör.

I programspråket C kan datadefinitionerna i värsta fall vara kraftigt utspridda, om man inte gjort speciella ansträngningar för att hålla samman dem för att få en bättre uppfattning om vad data betyder.

I objektorienterade språk finns det dock en strävan att hålla alla deklarationer av data på ett samlat sätt. De deklareras i form av klassbeskrivningar, som bildar underlag för skapandet av "objekt". DESSUTOM beskrivs vad som kan hända med data – de operationer som kan vara aktuella för objekten. Dessa explicita deklarationer av "händelser" för data kan bidraga till att beskriva vad data BETYDER. Att data "hänger på" objekt bidrar också till ett klargörande.

Vi kan alltså se att programspråken delvis styr hur data deklareras och vilka intentioner som kan upprätthållas när det gäller att beskriva vad data betyder.

Dessa aspekter får speciell betydelse när det gäller att försöka förstå vad ett program egentligen gör. Att förstå de data ett program använder sig av bidrar till förståelsen av programmet.

7.4 Att kunna hitta bland (beskrivningar av) systemkomponenter

En del av förväntningarna på reverse engineering är att kunna återanvända systemdelar. En mängd förutsättningar måste dock vara uppfyllda för att önskvärda effekter skall kunna uppstå.

Det räcker inte med att systemdelar skall vara beskrivna, utan man måste dessutom kunna HITTA bland dem för att kunna återanvända dem. I [26] påpekas att detta inte är trivialt och att försök som har gjorts med återanvändning har medfört få effekter, eftersom man har haft svårt att hitta delar som potentiellt är återanvändbara.

Det verkar således som om man dels måste hitta bra sätt att beskriva programdelar, dels ha tillgång till "navigeringsfaciliteter" som kan användas för att hitta programdelar. Det betyder alltså olika former av datorstöd som har dessa möjligheter.

7.5 Vem kan bedriva analys av program?

Andra egenskaper än vad programmerare normalt har kan vara lämpliga när man granskar program.

Exempel på sådana egenskaper är

- forsknings- och analysintresse
- gestaltförmåga
- att hitta mönster i program (kognitionspsykologin kommer in här).

Det viktiga för en granskare är att hitta strukturer i program. Man blir lätt förblindad av namnen i programmen. (En undersökning refererades under konferensen [55], där personer inte kände igen sina egna program när man hade bytt ut datanamnen i dem.)

8 Informationssystemens "uppfattning" om verksamheten

8.1 Regler i verksamhet och i system

Kan man lära sig om verksamheten genom att studera dess informationssystem? Frågan är om det är intressant och om det är rätt väg att gå.

Om man vill "återanvända" existerande system är man dock intresserad av att ta reda på vilka regler och restriktioner om verksamheten som system egentligen besitter.

- Vilka situationer i verksamheten känner systemet?
- Vilka beslutsregler finns egentligen representerade i systemet?
- Vilka förhållanden i verksamheten känner systemet?
- Vilka situationer, beslutsregler och förhållanden är felaktiga?

Utgångspunkterna för en sådan analys kan vara olika. Man kanske misstänker felaktigheter i systemet i förhållande till den uppfattning man vill att systemet skall ha av verksamheten (som verksamheten ser ut idag eller som man vill att den skall se ut i framtiden).

Studiet av ett system kan ske parallellt med att verksamhetsmodeller tas fram över vad man vill skall gälla i verksamheten. Diskrepanser mellan system och vad verksamheten själv har för uppfattning verkar vara viktiga att hitta, antingen det gäller hur det ser ut idag i verksamheten eller hur man vill att det skall se ut. Det kan dels röra sig om "felaktiga" (oönskade) skillnader och dels kanske verksamheten vill gå åt ett visst håll medan systemet släpar efter.

En parallell analys av befintligt system och en verksamhetsanalys kan vidare i vissa fall vara ett stöd i argumentationen "att som det ser ut idag skall vi i alla fall inte ha det".

Ett exempel på olikheter mellan system och hur en viss verksamhet vill se saker kan vara det lämpliga i att attributet "pris" är en egenskap till objektet "Kontrakt" i en viss verksamhet. Om så är fallet kan det skvallra om en verksamhetsregel, som innebär att man kan skapa rabatter baserade på kvantitet eller på kundlojalitet mot verksamheten, vilket man inte hade fått klart för sig genom att tala med de aktuella beslutsfattarna.

Ibland kan reverse engineering ställa frågor om HUR man bedriver affären. Svaren kan sedan rapporteras till högre nivåer. Är de verksamhetsregler som vi använder oss av idag olämpliga?

9 Projekt- och styrningsaspekter för reverse engineering/reengineering

9.1 Vem skall styra och leda projekt för reengineering?

Det har visat sig i undersökningar från flera håll att en av de viktigaste faktorerna för ett förändringsprojekts framgång är sättet på vilket det leds och styrs. Orsaken till att många projekt har gått snett är att ledningen har varit bristfällig.

Beslut om reverse engineering och återanvändning skall fattas på hög nivå. Inte bland programmerarna. Bland annat betonas från flera håll betydelsen av att reengineering-projekt är förankrade på ledningsnivå i verksamheten och att de styrs med avseende på detta.

Detta är naturligtvis starkt relaterat till att det är viktigt att koppla målen för en reengineeringsinsats till förändringsmålen för affär och verksamhet.

Det är i själva verket farligt att ett reengineeringsprojekt inte är känt och motiverat på ledningsnivå så att man omedelbart kan svara på frågan varför ett visst reengineeringsprojekt är igång och vad det syftar till.

Man måste alltid ha väl formulerade skäl att starta reengineering. Management måste vara med på att insatserna skall utföras. Helst skall en viss person från management direkt kunna svara på frågor om varför insatsen bedrivs. Teamet skall inte behöva sväva i osäkerhet om att det inte finns någon att fråga om inriktning, vad som är väsentligt o s v. Reverse engineering måste ha sin utgångspunkt i verksamheten. Skälen till att man måste göra en insats måste "sitta i verksamheten".

Om man arbetar med reverse engineering och detta leder till att inblandade personer pratar om verksamheten och vad som är viktigt för denna är man på rätt väg. Incitament för deltagarna att göra ett bra jobb i projekten är viktiga.

Under projektledningen kan reengineeringsinsatserna på det operativa planet styras av en specialistgrupp. Denna måste ha befogenheter. Specialistgruppen skall sörja för att avgränsningar görs, rätt analyser görs, lämplig metodansats tillämpas. Den skall vidare känna till olika typer av metodik, samt kunna verktyg.

Det finns ett stort kunskapsbehov när det gäller att kunna kalkylera resurser för att göra olika insatser inom ramen för reverse engineering och reengineering. Hur stora insatser går det åt för att genomföra en viss typ av arbete?

9.2 Hur ser management på reengineering?

Downsizing kan vara ett farligt begrepp, menade man på konferensen [55]. Begreppet är insålt på managementnivå och där tror man att man kan spara en massa pengar på detta. Huruvida effekterna inträder ansågs vara osäkert. Downsizing har en del element av reverse engineering i sig, men det är oklart om dessa tekniker tillämpas

och om man är intresserad av andra effekter som är vanliga att tala om i reverse engineeringsammanhang. Är t ex en annan mer dokumenterad struktur lämplig i underhållssammanhang.

Ibland talar man om att en reverse engineering/reengineering-insats kan ta flera år i anspråk att genomföra. Detta är viktigt att komma ihåg när det gäller att lägga upp planer och kunna leverera delresultat under kortare intervall. Management kan i många företag inte i framtiden garantera 3-4-årsprojekt längre. Man kommer troligen bara att tilllåta 1-årsprojekt. Förändringar i affär och verksamhet är så snabba att det är meningslöst att lova att ett projekt kommer att pågå flera år. Ledningen kan ha avgått under tiden, för att nämna en anledning. Därför måste man ha uppvisbara delresultat som kan produceras med korta tidsintervall. Delresultaten visas enligt en flerårig strategi som löpande revideras.

10 Värdering av insatser för reengineering

Att uppskatta resurser och bedöma effekter av insatser för reengineering av informationssystem är ett svårt men viktigt område. Man kan konstatera att området inte är särskilt utvecklat. Det håller dock på att växa fram en del hjälpmedel för området och man har funderat på vissa metodansatser för hur arbetet skall kunna bedrivas. Kunskap och nyckeltal, t ex för att bedöma vilka insatser som krävs för att genomföra en viss typ av arbete, på en viss typ av system, som har en viss komplexitet, existerar i liten utsträckning.

SISU avser att under första projektåret i Effektiv IT-programmet delvis belysa detta område i en rapport som kommer under senare delen av året.

Det finns däremot, som bekant, en del metoder för att estimera insatser för **forward** engineering och olika insatsers effekter i verksamheterna. Man arbetar nu också på att få fram estimeringsnyckeltal för reverse engineering-insatser. Detta arbete är dock bara i sin linda. Vissa prov har gjorts. Tillämpning av detta måste relateras till arbetssätt i en viss organisation. US Army CECOM har gjort vissa prov. Som grund har man en grov s k reengineering cost model.

11 Verktyg

11.1 Repository för reverse engineering

Ett täckande repository skall också kunna representera de fysiska delarna av det existerande systemet och t ex inte bara innehålla pekare till koddelar. Det är viktigt att ha en genomarbetad semantisk koppling ("semantic coupling") mellan de logiska och de fysiska delarna där begrepp i de två strukturerna kan relateras till varandra genom meningsfulla associationer.

11.1.1 Ny typ av verktygskluster

Det kommer nu på marknaden ett antal verktygsansatser som har ett s k öppet repository och med ett antal verktyg för reverse engineering knutna till detta. Ett sådant verktyg kommer ifrån Advanced Systems Technology Corporation (ASTEC). Metamodellen för repositoriet är duplicerbar. De strukturer som lagras är så utrustningsoberoende som möjligt. Metaschemat är modifierbart. Det finns ett antal bryggor till andra verktyg. Fler bryggor kan byggas på beställning.

När man inför verktyg för underhåll i en verksamhet, är det ibland svårt att få de personer som arbetar med underhåll att ändra sina arbetssätt på kort sikt. Man får istället utgå ifrån hur underhållsprogrammerarna arbetar idag.

Man bör skaffa belöningssystem för att komma igång med reverse engineering och därmed kopplade verktyg. Detta verkar saknas.

11.1.2 IBM och reverse engineering

IBM presenterade vid 3rd Reverse Engineering Forum sitt redevelopmentkoncept [3], [22]. Man uppmuntrar utvecklare som tar fram resultat för reverse engineering. Man vill hjälpa industrin att skapa "redevelopment strategies".

70-80 % av "livscykelkostnaderna" går åt till underhåll. Ca 50 % av dessa resurser använder personer som bedriver underhåll till att lära känna det som man skall underhålla. Mycket av underhållet är alltså "understanding".

"Dålig" kod kan vara mindre intressant att underhålla. Den kan var för dyr att göra reengineering på.

Några frågor som ställdes i inslaget var:

- De så kallade "systemresurserna" finns inte i dagens balansräkningar. Bör de vara där?
- Skall man göra reengineering på program som man inte fått felrapport på?

IBM menar att man skall:

- Skapa planer för reengineering
- Etablera metoder för arbetet
- Sälja in koncept och etablera en kultur som är intresserad
- Utbilda

IBM jobbar med programutvecklingsföretaget Viasoft och tillsammans har man satt upp verktyg i 3 delområden:

- Understanding
- Reverse engineering
- Reengineering

11.1.3 Svårigheter att skapa verktyg

T. Cahill, University of Limerick, Irland, rapporterar [10] att det är svårt att skriva bra verktyg för att analysera program. Program skrivna med vissa programspråk är svårare att analysera automatiskt än andra. Ada är ett av de lättare språken eftersom man där skapar strukturer som gör att automatiska analyser lättare kan göras.

COBOL är ett av de svårare språken. Detta eftersom man har konstruktioner som MOVE och REPLACE. Där går man från en datastruktur till en annan och från analys-synpunkt är detta mycket svårt att följa. Data betyder olika på olika ställen, vilket är svårt att maskinellt deducera. Olika sätt att se data på dyker upp i programmet.

Strävan i det aktuella forskningsprojektet var att lagra saker i ett repository på ett platt-formsoberoende sätt. Det finns en hel del svårigheter med detta. Arbetena är gjorda i ett ESPRIT-projekt (REDO).

11.2 CASE och reverse engineering

Vid 3rd Reverse Engineering Forum [55] hävdades att CASE-teknologin är mycket lämplig att användas tillsammans med reverse engineering, men också att denna teknologi saknar en del viktiga möjligheter för att kraftfullt kunna användas för reverse engineering.

Till CASE-teknologins möjligheter i sammanhanget påpekades att man kan

- se saker i nya projektioner och perspektiv
- göra analyser av olika slag.

11.2.1 Bryggor mellan verktyg

När man skall arbeta med reverse engineering arbetar man ofta med en grupp verktyg som måste kunna kommunicera med varandra.

Ansatsen som finns till att en viss uppsättning verktyg kan kommunicera med varandra är viktig. Dåliga kopplingar mellan verktyg kan göra att mycket av det som skall överföras går förlorat.

11.2.2 Passar I-CASE-verktygen för reverse engineering?

En intressant observation gjordes vid 3rd Reverse Engineering Forum [55] avseende CASE-verktygens lämplighet att stödja arbete med reverse engineering genom påpekandet att I-CASE-verktygen inte representerar fysiska begrepp. De är inriktade på att generera strukturer som inte lagras explicit. Man regenererar i stället. JCL-, IDMS- och IMS-begrepp kan således inte lagras.

I något fall hade man gjort reverse engineering på ett gammal system, gjort reengineering på det och lagt det i I-CASE-miljö. Detta gjorde att det blev ännu svårare att göra underhåll på systemet än vad som var fallet förut. Strukturerna var inte tillgängliga som man var van vid.

Namngivning är viktig i reverse engineeringssammanhang. Hålkorten gjorde tidigt att namn blev korta och meningslösa. Många CASE-verktyg konserverar detta förhållande. Har man goda och semantiska namn har man bättre chanser för reverse engineering. Att förstå datahantering i program kan vara mycket svårt om man inte har bra namn. Att använda en semantiskt kraftfull modellering är naturligtvis en mycket bra teknik i detta sammanhang.

Att lagra resultat från en reverse engineeringsinsats i ett repository resulterar ofta i en stor mängd "instanser". Om ett mellanstort oljebolag, t ex, skulle lagra samtliga logiska och fysiska begrepp i ett repository skulle man få ca 50 milj instanser.

Boeing berättade att man gjorde ett mycket stort reengineeringsarbete där man gör reverse engineering på COBOL-program och går över till en objektorienterad miljö. Man har specialbyggda verktyg i något som man kallar COBOL Reuse Workbench. En hel del kunnigt folk verkar ligga bakom insatsen och det borde vara intressant att följa upp detta arbete efterhand.

11.3 Typer av verktyg

I [19] sammanfattas den rådande situationen på verktygsområdet på följande sätt:

> "At present there is no single comprehensive reverse engineering tool. Rather there is a wide spectrum of tools that provide different capabilities addressing various pieces of the reverse engineering process."

Verktyg för reverse engineering/reengineering kan, i enlighet med syfte och funktionalitet, grovt indelas i tre kategorier med stigande ambitionsnivå [18], nämligen:

1. Verktyg för programanalys
2. Verktyg för programförståelse
3. Verktygsmiljöer som integrerar reverse och forward engineering.

I den första kategorin, verktyg för programanalys, finner vi verktyg som ger en hel del detaljinformation om existerande programkod. Typiskt för dessa verktyg är att programkoden beskrivs med olika alternativa vyer. Det kan t ex vara strukturdiagram, som dels visar anrop mellan olika subrutiner, dels visar den interna programstrukturen. Andra uppgifter som lämnas kan vara listor över korsreferenser mellan programmoduler och dataelement eller olika storheter från komplexitetsberäkningar av moduler.

I kategorin verktyg för programförståelse går man ett steg längre genom att t ex i en fönstermiljö förse programmeraren med en snabb navigeringsmöjlighet till olika delar av programkoden. Genom att exempelvis klicka på en modul i ett strukturdiagram får man upp den aktuella koden i ett annat fönster. Man kan också få tillgång till programkod från filtrerade listningar, mönstersökningar, m m. I många fall klarar också verktygen av att hantera inkrementella ändringar av programkoden. För att klara detta, lagras programkoden efter en första genomläsning i ett repository, varefter ändringar i programkoden endast uppdaterar detta repository.

I tredje kategorin har man integrerat verktyg för reverse engineering med verktyg för forward engineering, dvs CASE-verktyg. Detta kan göras antingen genom att CASE-verktyget direkt läser i det repository som verktyget för reverse engineering har skapat, eller genom en mer lös koppling, där kommunikationen mellan verktygen sker via filer. Denna typ av verktyg kan vara effektiv för att förbättra en obefintlig eller undermålig dokumentation och därigenom görs det också lättare att underhålla och förbättra äldre system. Om existerande system också kan representeras i en modelldatabas i ett Upper CASE-verktyg, så kan också förändringar i verksamheten lättare underhållas. Denna typ av verktyg kan inte bara med hjälp av reverse engineering bakvägen hjälpa till i arbetet med att ta fram designinformation och programdokumentation, utan kan även i fortsättningen hålla dokumentationen synkroniserad med programkoden.

För att fördjupa ovanstående indelning och gå litet närmare in på vilka olika typer av analyser av program som kan vara aktuella, och vilka typer av förståelsebefrämjande assistans som olika verktyg kan ge, återges här en kategorisering och beskrivning av egenskaper/faciliteter hos verktyg för reverse engineering/reengineering från [19]. I denna artikel tas följande områden upp:

1. Statisk analys
2. Programförståelse
3. Reverse engineering
4. Portföljanalys
5. Datastabilisering

6. Omstrukturering
7. Ompaketering
8. Transformation

Statisk analys

Statisk analys avser analys av element, strukturer och relationer i ett program. Resultatet är vanligen en detaljerad fysisk modell av programmet i någon form.

Strukturell analys avbildar programstrukturen i form av ett anropsträd eller ett strukturdiagram ("structure chart"). Beskrivningar av denna typ visar modulerna i en applikation och deras anropsrelationer. Resultatet kan också bestå av en korsreferenstabell med mer detaljerad information om ingående data- och programelement.

Vid analys av styrflöden avbildas alla vägar genom programmet i syfte att identifiera alla beslutspunkter och beslutskriterier, att upptäcka redundans, onåbar eller rekursiv kod och att uppskatta testbarhet och underhållsvänlighet av ett program.

Resulterande information visas vanligen i form av flödesplaner som visar beslutsnoder, ibland med vidhängande beslutskriterier.

Statisk analys kan ge värdefulla insikter om ett program och en uppfattning om konsekvenserna av en föreslagen ändring, dvs hur en ändring fortplantas och vilka andra delar som påverkas. Dock ger en sådan analys inte tillräcklig hjälp till att förstå beteendet hos och syftet med ett program.

Programförståelse

Verktyg för programförståelse är avsedda att hjälpa till med att få en klarare uppfattning om ett programs beteende och att upptäcka olika typer av defekter på kodnivån. De innehåller ofta funktioner som brukar finnas i s k "debuggers" och innehåller i tillägg olika typer av bläddrings-, söknings- och filtreringsfunktionalitet.

Typiska egenskaper är exekveringskontroll (exekveringen kan startas och stoppas på godtyckliga punkter och utföras stegvis), "spårning" (trace) framåt eller bakåt med utgångspunkt från en viss kodrad, att utföra komplexa sökningar (inkluderande "what-if"-sökningar) baserade på programlogik och aktuella eller inmatade värden på dataelement, villkorliga brytpunkter och watch-punkter, omedelbar exekvering av programkod genom interpretativ exekvering eller inkrementell kompilering och länkning, filtrerade kod-views, strukturerade data-views och automatisk inkorporering av kodlagningar.

Allt detta avser att hjälpa programmeraren att förstå koden genom att verifiera eller motbevisa hypoteser och uppfattningar som kan finnas om ett programs syfte och operativa beteende.

Ett annat syfte kan vara att härleda verksamhetsregler ("business rules") som ligger invävda i koden.

Reasoning systems Refine, t ex, skapar beslutstabeller från programlogik, vilket underlättar identifiering av inbäddade verksamhetsregler.

En del verktyg ger exekveringshistorik i form av "traces" av programsatser, datavärden och andra parametrar, eller grafiskt som en dynamisk anropsgraf som visar moduler som verkligen exekveras och anrop som faktiskt görs i syfte att klarlägga programmets beteende vid speciella transaktioner eller för speciella värden på dataelement. En del verktyg producerar och sammanställer exekveringsstatistik, som t ex antalet exekveringar av en procedur, en subrutin, ger total exekveringstid, max- och min-antal gånger, tidsåtgången för vissa programsatser o s v. Detta ger information som kan vara användbar som underlag för åtgärder som syftar till att förbättra programs och programdelars effektivitet m m.

Reverse engineering

I vid mening innebär reverse engineering, som vi tidigare sett, att genom analys inhämta och sammanställa information om existerande program och applikationer och representera denna i en form – ofta en mer abstrakt form – som är lättare att förstå för människor.

Existerande databasscheman, filstrukturer, skärmdefinitioner o s v, kan representeras på ett strukturerat sätt i ett CASE-repository. Genom att använda informationen i ett sådant repository kan andra verktyg sedan hjälpa till med att härleda olika fysiska och abstrakta representationer av typ korsreferenslistor, dataflödesdiagram, datastrukturdiagram, flödesplaner, logiska datamodeller, abstrakta syntaxträd, statiska anropsgrafer, "action diagrams", meny/dialog-strukturer. Verktyg finns som kan hjälpa till att normalisera logiska datamodeller.

Naturligtvis gäller att härledningar, utifrån beskrivningar/modeller på fysisk nivå, av olika representationer på mer abstrakt, logisk eller konceptuell nivå, inte kan göras automatiskt i sin helhet av ett verktyg. Verktygen kan ge assistans genom att producera underlag, som sedan kan bearbetas, förfinas och kompletteras av människor.

Information i repositoriet kan användas för att förbättra systemet och sedan omgenerera systemet med hjälp av "forward engineering"-verktyg.

Portföljanalys

Verktyg inom detta område hjälper till med att utvärdera existerande program eller applikationer för att möjliggöra "cost/benefit"-analyser avseende rätt nivå på underhållet, reengineering eller avveckling/ersättning. De kan bero av mer specialiserade verktyg för sina indata, t ex verktyg som utför analys av komplexitet och struktur. De kan också länkas till ändringshanterings- ("change management") och riskbedömningsverktyg och till verktyg som spårar upp olika defekter i program.

Datastabilisering

Datastabilisering innebär att identifiera alla ingående dataelement i databaser och samla deras definitioner – deras synonymer och homonymer, deras typer och domäner och var de definieras, skapas, används och modifieras.

Stabiliseringsverktyg stödjer också användningsanalys, inkluderande frekvenser och storlekar på dataåtkomst, genererar statistik avseende responstider, användningen av relationer mellan data m m.

Omstrukturering

Analyserar och omgenererar existerande kod på ett sådant sätt att funktionaliteten bevaras, men där den resulterande koden är strukturerad i så mening att den (t ex) följer principerna för strukturerad programmering.

En omstrukturering resulterar vanligen i att programmets komplexitet minskar, vilket potentiellt gör det lättare att underhålla och mindre känsligt för sidoeffekter.

Verktyg för omstrukturering av program är mest användbara när det saknas förståelse eller dokumentation för ett existerande system, och används ofta som ett försteg till reverse engineering. Om förståelse och dokumentation finns, kan en omstrukturering dock ge motsatt resultat. I så fall är ompaketering (se nedan) ett bättre sätt.

Ompaketering

Omstrukturering, som termen kommit att användas, har att göra med att ändra det interna flödet i ett program på satsnivå. Ompaketering är väsensskilt från detta och har att göra med den funktionella strukturen på ett program. Det är en generell term för flera relaterade aktiviteter som innebär att bryta isär ett existerande program och reorganisera det eller att extrahera speciell delfunktionalitet från det.

Ompaketering kan ses som ett specialfall av statisk analys och är inriktad på att identifiera samtliga komponenter i ett program som är relaterade till ett visst programelement, som t ex en variabel, en subrutin eller en Perform-enhet i COBOL.

Det kan finnas många orsaker till att man vill utföra en ompaketering. Motivet kan t ex vara:

- att identifiera redundanta funktioner eller procedurer som kan konsolideras till en modul. Storleken på en applikation kan därmed minskas och dess underhållsvänlighet kan förbättras.
- att isolera en funktion eller ett delsystem som har hög förändringsgrad eller felbenägenhet. Genom att extrahera funktionen, som kan vara utspridd över flera moduler, och lägga in den i en enda inkapslad modul, kan man bättre kontrollera påverkan av en ändring. Genom att arbeta om den och förbättra dess struktur och dokumentation kan man förbättra dess tillförlitlighet och göra den lättare att underhålla.

- att identifiera alla de delar i ett program som är relaterade till en funktion som kommer att flyttas från en monolitisk mainframe-applikation till en klient eller till en server-del i en klient/server-applikation.
- att isolera all kod relaterad till en speciell verksamhetsregel och identifiera alla aspekter av regeln. Genom att ompaketera den till en enda anropbar rutin, kan man försäkra sig om att den utförs konsistent genom hela applikationen. Genom att lägga in den i ett gemensamt bibliotek eller genom att knyta den till datadefinitionen, kan man försäkra sig om konsistens genom applikationerna. Detta är speciellt viktigt för tvärfunktionella applikationer som utnyttjar gemensamma databaser och verksamhetsregler.
- genom att identifiera, isolera och extrahera vanligt använda funktioner från existerande kod kan man börja att bygga en bas av återanvändbara komponenter. Nya applikationer kan sedan sättas samman från dessa moduler. Tom McCabe kallar detta "just-in-time"-återanvändning, dvs att återanvända existerande kod istället för att bygga omfattande återanvändningsbibliotek för hypotetiska framtida applikationer.
- stora applikationer kan brytas ner/dekomponeras för ökad överblickbarhet eller kan partitioneras så att medlemmarna i ett utvecklingsteam kan arbeta med olika delar mer oberoende av varandra.

De olika aspekterna på ompaketering kallas vanligen modularisering, konsolidering och dekomponering eller "slicing".

Modularisering innebär att bryta ut programsegment, som t ex en funktion, en paragraf eller en Perform-enhet, från ett existerande program samt alla programelement som beror på detta. Den kod som skall brytas ut måste identifieras manuellt.

Reverse engineering kan underlätta att ta hand om sidoeffekter och lösa ändar som kan uppstå som konsekvens av ett sådant ingrepp.

Konsolidering innebär att identifiera redundant kod och göra en enda anropbar enhet. Redundans är relativt vanlig. Den uppstår vanligen när en ändring görs i en del av ett program och en extra funktion eller parameter behövs. Hellre än att generalisera den nya funktionaliteten så duplicerar man komponenten och gör en lokal variant. Över tiden kan procentandelen redundans bli ganska stor – så mycket som 80 % redundans har uppmätts i en del system.

Dekomponering eller "slicing" innebär identifiering och extraktion av en delmängd av ett program med en speciell given startpunkt, som t ex ett dataelement, en rapport, ett logikbeslut eller en källkodssats.

Transformation

Många organisationer vill föra över/portera viktiga applikationer till nya, mer kostnadseffektiva plattformar. Detta innebär att behovet att kunna transformera till nya språk, nya användargränssnitt och nya och förändrade operativsystemmiljöer blir alltmer viktigt. Det finns ett stort antal specialiserade verktyg för att översätta från ett språk, ett användargränssnitt eller ett databasssystem till ett annat.

De flesta av dessa har en ganska direkt och enkel rak ansats som transformerar en stor del av systemet. Och som lämnar en hel del lösa ändar som programmeraren själv måste ta hand om.

Det finns dock några få mer sofistikerade översättningsverktyg som använder objekt-orienterade tekniker och expertsystemtekniker för att förse verktyget med visst mått av intelligens, vilket gör transformationerna mer tillförlitliga och robusta.

För företag som har mycket stora applikationsmängder kan det löna sig att investera i verktyg som Reasoning Systems Refinary (se "Exempel på verktyg med integration av reverse och forward engineering" sid 34) och i den anpassning av detta verktyg som behövs för att göra det effektivt i en viss given omgivning.

11.3.1 "Leading-Edge Tools"

I [19] presenteras vidare ett urval av verktyg inom reverse engineering-området som av författaren anses ligga i framkanten av utvecklingen. Följande leverantörer/verktyg nämns:

1. Bachman Information Systems
2. Burl Software Laboratories
3. Cadre Technologies
4. Hypersoft
5. Interport Software
6. Intersolv
7. Price Waterhouse
8. Reasoning Systems
9. SEEC
10. Visaoft

För närmare information om dessa verktyg hänvisas till artikeln och till resp leverantör. Några av verktygen presenteras litet närmare i nästa avsnitt.

11.4 Några exempel på verktyg

SISU har under det första året av Effektiv-IT-programmet tagit del av ett antal verktyg för reverse engineering och reengineering. Några av dessa skall vi redovisa här. Vissa av dem finns ännu inte att köpa i Sverige. Några är dock av så pass stort intresse att de borde finnas tillgängliga och vara kända på den svenska marknaden.

11.4.1 Några exempel på verktyg för programanalys

Cadre

Cadres Teamwork/FORTRAN Rev för FORTRAN och Teamwork/C Rev skapar strukturdiagram från FORTRAN- respektive C-kod. Därefter kan användaren enkelt navigera i programkoden med hjälp av dessa diagram. Det finns även versioner för Pascal, Basic och PL/1. Verktygen körs på olika typer av arbetsstationer.

McCabe & Associates

McCabe & Associates har för analys av program tagit fram ett antal verktyg, vilka hjälper till med underhåll och uttestning. Man kan bl a bestämma komplexiteten hos program och systemlösningar. Om vi exempelvis får höga komplexitetstal för vissa program i ett programsystem, så kan dessa ligga till grund för beslutet att antingen dela upp programmen i mindre delar eller att skriva om programmen helt och hållet i stället för att gå in i programmen och ändra dem. Program som är mycket komplexa är också mycket svåra att underhålla. Vid ändringar av sådana program tenderar man att göra dem ännu mer komplexa och därmed ännu svårare att underhålla. Därför kan det vara en god idé att skriva om eller strukturera om program med höga komplexitetstal, innan man inför nya ändringar.

Verktyget Analysis of Complexity Tool (ACT) mäter den s k "cyklomatiska komplexiteten" ("cyclomatic complexity") för ett program. Detta mätetal anger, om programmet har många interna vägar och därför kan tänkas delas upp. Förutom dessa komplexitetstal producerar ACT grafiska programstrukturer, genererar alla testvägar och testvillkor. Genom att hålla reda på dessa testvägar vid uttestningen av programmet kan ACT även ange vilka delar av koden som ännu inte är uttestade.

Verktyget Battlemap Analysis Tool (BAT) analyserar källkod på systemnivå och beräknar komplexitetstalet för "essentiell komplexitet" ("essential complexity"), som anger, om ett program är välstrukturerat. Program med höga sådana mätetal kan behöva skrivas om. I ett strukturdiagram markeras modulerna som testbara, underhållsbara eller icke underhållsbara.

Med verktyget McCabe Slice Tool kan man avgöra vilka delar av ett program som genomlöps för exempelvis vissa transaktioner. Detta kan vara till stor hjälp för att antingen förstå ett system utan att känna till dess struktur och kod eller för att kunna bryta ned ett komplext system i delar. "Slicing" kan användas i ett antal olika situationer:

1. Programförståelse
2. Programavlusning – att identifiera programvägar för "buggen"
3. Nedbrytning – olika funktioner som finns i samma program delas upp på olika program, exempelvis lagerhanterings- och redovisningsfunktioner splittras
4. "Downsizing" – att omkonstruera stora satsvisa system till ett distribuerat system
5. Redundant kod – att avlägsna kod som inte längre används

6. Återanvändbar kod – att identifiera kod för någon funktion som skall användas på ett annat ställe.

Med CaseBridge kan man överföra strukturer från McCabe-verktyg till andra företags "lower CASE"-verktyg. Hittills stöds Cadre's Teamwork och IDE's Software through Pictures.

Dessa verktyg stödjer de flesta vanliga programspråk och körs på olika typer av arbetsstationer.

11.4.2 Några exempel på verktyg för programförståelse

Inom denna kategori finns det flera likartade verktyg som erbjuder god utvecklingsmiljö för program. Efter det att ett verktyg har "scannat" ett existerande programsystem och lagrat information om systemet i en egen databas, ger verktyget bl a stöd för

- generering av olika strukturdiagram över system och program,
- "browsing" mellan strukturdiagram och kod,
- sökning av kod efter fördefinierade och användardefinierade frågor och sökmönster samt
- syntaktisk och semantisk kontroll av program.

Verktygen kan eventuellt även ge stöd för

- avlusning av program med en välutvecklad "debugger"
- inkrementell uppdatering av databasen vid ändring av kod.

Exempel på sådana här verktyg är CenterLine's CodeCenter för utveckling av C-program och ObjectCenter för C++, ProCASE's SMARTsystem för C-program och Advanced Software Automation's (ASA) Hindsight för C eller FORTRAN-program. Hindsight ger i nuvarande version inget stöd för inkrementell uppdatering av databasen. För COBOL-applikationer finns Interport Software Corporation's InterCycle. Detta verktyg ger förmodligen inte stöd för de två sista punkterna i uppräkningen ovan.

11.4.3 Exempel på verktyg med integration av reverse och forward engineering

IDE's The C Development Environment

Många verktyg inriktar sig på en liten del av reengineering-processen. Man behöver därför använda flera verktyg för olika delar av processen. Det problem man då ofta stöter på är att de olika verktygen har svårt att kommunicera med varandra.

Företaget Interactive Development Environments (IDE) har skapat en intressant miljö för utveckling av C-program i verktyget The C Development Environment. I detta verktyg har man integrerat sitt eget verktyg Software through Pictures med CenterLine's CodeCenter, som innehåller en programutvecklingsmiljö för C, och FrameMaker eller Interleaf för automatisk uppdatering av dokumentationen av programmoduler.

Det integrerade verktyget The C Development Environment har ett antal intressanta egenskaper:

- i olika X-fönster kan man samtidigt titta på flödesplaner, programkod och dokumentation. Genom att klicka på ett begrepp i ett av fönstren får man upp motsvarande begrepp i de andra fönstren.
- frågemöjlighet till ett gemensamt repository, som används för att lagra alla systemkomponenter. Denna möjlighet används bl a för att undersöka vilken påverkan på systemet som helhet en viss förändring för med sig.
- reverse engineering av ett existerande system med C-program för att skapa ett repository, nya flödesplaner och ny dokumentationsmall.
- skapa eller ändra i program genom att antingen ändra i flödesplanerna eller i programkoden. Om man ändrar i koden uppdateras flödesplanerna automatiskt och vice versa.
- dokumentationen uppdateras automatiskt.

IDE har också en motsvarande miljö för Ada-programutveckling som heter The Ada Design Environment och som integrerar ObjectOriented Structure Design/Ada, The Code Generator for Ada, the Verdix Ada Development System och FrameMaker eller Interleaf.

Software Refinery

Många verktyg inom reverse engineering och reengineering kan analysera programkod för att producera olika flödesplaner och rapporter. Men detta är inte alltid tillräckligt eller just det man behöver. Så t ex vill man kanske kunna arbeta med någon speciell språkdialekt eller göra någon analys som inte finns i verktyget. Då har man ingen möjlighet att modifiera verktygen för dessa speciella krav. Reasoning Systems, Palo Alto, har tagit fram ett programsystem under samlingsnamnet Software Refinery, som man kan anpassa efter önskemål.

I Software Refinery finns programmen:

1. DIALECT, med vilket man kan
   - bygga en "parser" för godtyckligt programspråk
   - läsa program i detta programspråk och skapa objekt i en AST-representation (AST = abstrakt syntaxträd) i en databas (ej språkneutral AST)
   - generera programkod i samma eller annat programspråk från objektträdet
2. REFINE, med vilket man kan
   - analysera och modifiera objekten i databasen (ex.vis överföra AST- representationen från ett språk till ett annat)
   - skriva egen analys i REFINEs specifikationsspråk
3. INTERVISTA, med vilket man har byggstenar för att bygga ett modernt X-fönstergränssnitt.

Färdiga modifierbara reverse engineering-verktyg finns för vanliga programspråk i produkterna REFINE/FORTRAN, REFINE/C, REFINE/Ada och REFINE/COBOL.

Som exempel på användning av REFINE/COBOL anger man erfarenheter från ett modulariseringsprojekt där man delade upp COBOL-program, som hade i genomsnitt 15000 rader kod/program, till program med i genomsnitt 1000 rader kod/program. Med den automatiserade uppdelningen med REFINE/COBOL tog det 4 persontimmar att åstadkomma detta. Medan man har uppskattat att det skulle ta 20 personveckor utan REFINE/COBOL. Eftersom konstruktionen av konverteringsprogrammet tog 1 man-månad, var det en lönsam affär. Kostnaden för verktyget?

Ett annat fall med användning av REFINE/FORTRAN beskrevs i ett föredrag vid 3rd Reverse Engineering Forum [17] av Phil Eschallier, Paramax Systems Corp. I projektet konverterade man flera miljoner rader gammal FORTRAN-kod tillhörande NASA till VS FORTRAN-77. Den gamla koden var mestadels IBM-370 FORTRAN-IV med PREFOR-utvidgningar, som man inte längre ville understödja.

Med DIALECT och REFINE skapade man

- en "parser" för att analysera de gamla FORTRAN-programmen och skapa abstrakta syntaxträd i en databas
- en mönsterigenkännare för att manipulera programmen i databasen, bl a att konvertera från PREFOR till VS FORTRAN
- en kodgenererare för att generera VS FORTRAN-77-kod från databasen.

Att skapa dessa tre program tog 2,5 månader i anspråk för 2 personer.

Ett annat skäl för projektet var att förbättra den dåliga dokumentationen. Till detta använde man IDEs CASE-verktyg Software through Pictures.

Reasoning Systems har, vad vi känner till, tyvärr ingen representant i Sverige, inte ens i Europa. Trots det finns företag i Europa som har köpt Software Refinery, bl a norska Telecom.

Cadre's Ensemble

Om man redan har Cadre's CASE-verktyg Teamwork för konstruktion av C-program, så kan reverse engineering-verktyget Ensemble vara ett naturligt komplement för att förbättra underhållet av dessa program. Ensemble innehåller två moduler – System Understanding och Function Understanding. System Understanding levererar systemflödesplaner, medan Function Understanding ger programflödesplaner och beräknar komplexitetsstorheter från existerande kod.

Cadre's DB Designer

För att konvertera system med VSAM, IMS eller IDMS till relationsdatabaser, gå från en Entity Relationship(ER)-modell till en relationsdatabasimplementering, restrukturera en

relationsdatabas eller utföra någon kombination av dessa uppgifter, har Cadre Technologies utvecklat verktyget Teamwork/DB Designer.

Verktyget DB Designer hämtar och analyserar data från tre olika källor – ER-diagram, relationsdatabasbeskrivningar och aktuella data från existerande filer. Dessa aktuella data kan härröra från relationsdatabaser och andra filer från system med VSAM, IMS eller IDMS. Från dessa data tar verktyget fram relationsdatabasbegrepp som funktionella beroenden, domäner, primärnycklar, främmande nycklar, attribut, index, tabeller och databasinstanser. Samtidigt noterar DB Designer inkonsistenser och redundanta data, exempelvis redundanta attributnamn.

Vid varje fas i konstruktionsarbetet är det sedan möjligt att generera och visa exempelvärden i de aktuella tabellerna. Detta gör det lättare att i samspråk med slutanvändarna upptäcka felaktiga eller icke fullständiga relationer. Eftersom DB Designer läser verkliga data visas också om möjligt verkliga data i exempeltabellerna.

Som utdata från verktyget genereras exekverbar SQL-kod för ett antal relationsdatabaser som DB2, Oracle, Rdb, Ingres, Informix och SYBASE.

DB Designer kan köras på PC med OS/2 under Presentation Manager eller på arbetsstationer med Open Look eller Motif som fönsterhanteringssystem. Verktyget har länk till Teamwork.

11.5 Forskningsorienterade verktyg inom reverse engineering

På forskningssidan kommer det fram en del verktyg för reverse engineering. Bl a ett verktyg från University of Victoria. Scott Tilley [53] menade vid 3rd Reverse Engineering Forum att man nu får mycket mer uppmärksamhet av att utveckla för reverse engineering än att utveckla en ny metod eller ett nytt verktyg för forward engineering. Det presenterade verktyget hade en parser för vissa programmeringsspråk (C och COBOL), ett repository och en grafikeditor som ritar upp strukturerna. Denna typ av verktyg verkar bli vanlig inom området. Man väljer ofta att ta bort olika typer av "noise" innan man ritar upp strukturer mellan programdelar. Det kan t ex vara att en viss modul anropas från en mängd ställen. Man funderade på att koppla på ett multimediagränssnitt på det aktuella verktyget. Presentatören menade också att parsers för olika programspråk borde spridas lite mer bland verktygsutvecklarna, så att inte varje utvecklare behöver uppfinna hjulet varje gång.

David Sharon, CASE Associates Inc, presenterade vid samma konferens en uppsättning kriterier för evaluering av verktyg för reverse engineering och reengineering [45]. Det fanns både karakteriseringskriterier och mera av värderingskriterier. De aktuella behoven måste styra vikterna för värderingskriterierna i varje enskilt fall.

11.6 Några speciella frågeställningar

11.6.1 Löpande analys av ständigt ny kod

Reverse engineering har starka kopplingar till underhåll. När Hewlett-Packard exempelvis underhåller sina 1,2 miljoner rader C-kod (1000 directories) enligt OSF, måste man hålla reda på olika versioner. Vissa delar av koden kommer ifrån flera olika parter inom OSF – t ex IBM eller DEC. Speciella egenheter kan finnas hos de olika delleverantörerna. Versioner av mjukvaran kommer från OSF när denna organisation släpper en ny version. HP måste då så snabbt som möjligt komma ut med sin version för sina maskiner. Man måste därför hålla reda på de delar som är utrustningsberoende och om de nya delarna har någon effekt på de delar som är utrustningsberoende o s v. Man vill kunna "komma ihåg" att vissa delar var speciellt utrustningsberoende fast man utifrån får ny kod vid varje tillfälle. Man vill från HP ha bättre verktyg för att vid reverse engineering kunna göra ett bättre löpande underhåll. Man uppmuntrar verktygsutvecklarna i denna riktning. Mer analysstöd i verktygen. Inte minst önskar man bidrag som möjliggör tidsbesparingar i arbetet, eftersom detta är ett överlevnadsvillkor.

11.6.2 Tekniker för extrahering av verksamhetsregler ur programkod

Phil Glasier, Hewlett-Packard, m fl, gör i [23] en översikt av metoder som tar fram verksamhetsregler ("business rules") ur programkod. Dessa verksamhetsregler är unika för ett visst företag och är därför mycket viktiga för företagets fortsatta investerings- och konkurrensfördelar. Man antar att dessa verksamhetsregler finns inbyggda i programkod som det inte finns dokumentation för. Eller som har skrivits av programmerare som inte längre finns kvar i företaget eller som har glömt programkoden.

När översikten presenterades vid 3rd Reverse Engineering Forum var auditoriet oenigt om vad man menade med verksamhetsregler. Föredragshållaren exemplifierade med två typer av verksamhetsregler. Den första var av typen restriktions/avbildningsregel: en order måste innehålla en eller flera orderrader. Den andra var av typen behandlingsregel: en vara måste ha anlänt, innan man betalar leverantören.

I översikten noteras tre metoder med vilka man kan ta fram verksamhetsregler ur programkod:

- Kodskivning ("code slicing"), med vilken man kan isolera de delar av programkoden som utnyttjas för en viss verksamhetsregel. Michael A. Wolf, Viasoft, uppges att ha utnyttjat denna teknik. Kodskivning är lätt att förstå men svår att automatisera. En annan nackdel är att man får kvar en massa kod i en skiva, kod som inte har med verksamhetsregeln att göra.
- Logikmodellering ("Logic Modelling"), med vilken man grafiskt eller textuellt beskriver interaktionen mellan entiteter och processer. Charles Bachman, Bachman Information Systems, refereras. Utöver data- och processmodeller beskrivs interaktionen mellan entiteter och processer i en matris.

- Processnormalisering ("Task Normalization"), där man identifierar små odelbara programdelar (processer). Projektet ROSADE med Aaron Hanks t ex. Verksamhetsreglerna utgörs då av processerna tillsammans med villkor för att processen skall utföras. Några andra karakteristika är att processerna kan byggas upp i en hierarki med funktionsklasser, processklasser och processer, integrerar textuell beskrivning med processerna, är extremt lågnivåbetonad och metoden existerar endast i teorin.

11.6.3 Layout för grafer för programförståelse

J. Smart, Lawrence Livermore National Laboratory [46], har gjort ett verktyg för att generera "optimala" grafer för komplexa programvarusystem. Exempelvis en graf med moduler som anropar varandra. Grafen är "optimal" i den meningen att en energifunktion, som beror av den genererade grafen, har så litet värde som möjligt. Energifunktionen är summan av hela 14 viktade komponenter. Komponenterna anger storheter som närhet mellan moduler, hierarki mellan moduler, antalet linjekorsningar, avstånd mellan moduler m fl.

Eftersom man inte kan beräkna energifunktionen för alla tänkbara grafer, så används en iterativ strategi för att successivt söka sig mot allt lägre värden på energifunktionen. Med parametrar kan man ställa in hur snabbt man vill komma till resultat och hur noggrant resultatet skall vara. Ett antal olika algoritmer för sökningen har provats ut.

Att med verktyget konstruera en anropsgraf av verktyget självt med sina enbart 18 moduler tog ungefär 20 sekunder på en VAX. Den uppritade grafen såg helt perfekt ut. Det vore naturligtvis intressant att få ta del av tidsåtgången och resultatet för en mycket stor och komplicerad graf.

11.6.4 Ett exempel på ett kodtransformationsverktyg (JOVIAL - Ada)

Michael Olsem, USAF Software Technology Support Center [39], beskrev vid 3rd Reverse Engineering Forum, ett pågående treårigt projekt som innebar att transformera JOVIAL-kod till Ada-kod. Projektet är uppdelat i två faser. Den första fasen, som skall vara avslutad i oktober 1992, går ut på att transformera JOVIAL-kod till en AST-representation (AST = abstrakt syntaxträd) i en databas. Under andra fasen transformeras AST-representationen till Ada-kod. Föredraget redogjorde till största delen den första fasen, medan den andra tyvärr beskrevs mycket ytligt.

I fas ett ingår följande delar:

- att transformera JOVIAL-kod till en AST-representation samtidigt som koden kontrolleras vara korrekt JOVIAL-kod enligt en viss standard
- att analysera AST-representationen, varvid bl a oanvänd kod identifieras
- att generera indata till Cadres verktyg Teamwork i form av dataflödesdiagram, strukturflödesplaner (Structure Charts), datakatalogposter (Data Dictionary Entries) och processpecifikationer.

Det kan vara intressant att notera att dessa indata beskrivs i en i CDIF-fil (CDIF = CASE Data Interchange Format). Teamwork används endast för att grafiskt åskådliggöra programsystemet. Det går inte att modifiera med CASE-verktyget och sedan återföra ändringarna till AST-representationen. Ändringar måste göras i JOVIAL-koden och senare i Ada-koden!

12 Slutsatser och sammanfattning

Reengineering av informationssystem kan betraktas som en kunskaps- och förändringsprocess. Verktygen är bara en del. All reverse engineering måste ha en riktning – ett mål. Målen kan vara högst olika. Det lämpliga i och sättet att bedriva reverse engineering och reengineering måste starkt anpassas till varje företag och dess verksamhetsmässiga situation. Sådana verksamhetsmässiga situationer har systematiserats i [30].

I enlighet med målen för verksamheten och målen för reengineering-insatsen måste man dessutom ha en plan – och i många fall en strategisk plan. Planen bör innehålla ett antal delresultat. Man måste se till så att olika delresultat måste kunna produceras om insatsen drivs över flera år.

Analys är en viktig del av reverse engineering. Det handlar mycket om att förstå de informationssystem man har. Verktygen kan ge vissa resultat i denna process, men det måste finnas kunskap om hur sådana resultat skall tolkas och sättas i ett sammanhang för att formulera lämpliga åtgärder. Det finns inga enkla och självklara lösningar i detta område.

Reverse och forward engineering hänger starkt ihop. Det förefaller dock saknas metodik för att arbeta på ett systematiskt sätt i de två riktningarna.

Hur verksamheten ser ut kan man inte få reda på med hjälp av reverse engineering-resultat, men man kan få fram en uppfattning som systemet har om verksamheten. Skillnader i vad systemet känner till och bör känna till är naturligtvis viktiga att attackera.

Utveckling och underhåll av informationssystem kostar i USA ca 100 miljarder dollar om året. 70% av detta är underhåll. Man antar att dessa kostnader väsentligt kan minskas genom arbete på ett systematiskt sätt med reverse engineering/reengineering.

Det rapporteras att man i vissa fall har minskat underhållskostnaderna med 40 - 50% genom olika former av reverse engineering. I andra fall 35%. Tillförlitligheten i uppgifterna, som kommer från flera håll, är svår att avgöra.

Det verkar dock av SISU:s undersökning [28] bland svenska företag som om det inte är kostnadsnivån i sig som är det man mest reagerar på när det gäller systemarvet, utan restriktionerna i de gamla systemen när man vill göra större ändringar. Dessa upplevs speciellt då man vill ändra systemen i väsentlig omfattning, orsakad av att verksamheten ändras på ett markant sätt.

13 Referenser och litteraturförteckning

[1] Arnold, Robert S (Ed). "Software Reengineering", IEEE Computer Society Press, 1993

[2] Barnes, B. and Bollinger, T. "Making Reuse Cost-Effective", IEEE Software, January 1991, pp 13-24

[3] Benson, K. "AD/Cycle ReDevelopment Tool Integration", in [55]

[4] Berndtson, O. och Welander, T. "Fungerande systemförvaltning", Studentlitteratur, 1991

[5] Biggerstaff, T.J. "Design Recovery for Maintenance and Reuse", IEEE Computer, July 1989

[6] Brandt, P. "Hur bedriver man systemförvaltning", DSV, Report No. 92-001-DSV, 1992

[7] Brodie, Michael L. and Stonebraker, M. "DARWIN: On the Incremental Migration of Legacy Information Systems", GTE Laboratories Technical Report TR-0222-10-92-165, March 1993

[8] Brodie, Michael L. "Interoperable Information Systems: Motivations, Challenges, Approaches, and Status", Documentation at the Nordic Symposium on Interoperability and Legacy Systems, April 20-21, 1994, SISU, Stockholm

[9] Bush, E. "Reverse Engineering", Proc. of Fourth Software Maintenance Workshop, Sept. 1990, Centre for Software Maintenance Ltd., University of Durham

[10] Cahill, T. "Practical Difficulties in Developing Automated Tools for Analysis of Large Application Systems", in [55]

[11] Calow, H. "Adding a Reverse Engineering Flavour to Today's Methodologies", Proc. of From Software Maintenance Towards Reverse Engineering and Beyond, June 1990, Blenheim Online Ltd

[12] "CASE Tools for Reverse Engineering", CASE Outlook, February 1988, Vol 2, No 2

[13] Chikofsky, E, and Cross II, J. "Reverse Engineering and Design Recovery: A Taxonomy", IEEE Software, Jan 1990

[14] Chikofsky, Elliot J: "Reverse Engineering", Videotape & Video Notes Booklet, IEEE Computer Society Press, 1991

[15] Dahl, R & Johansson, L-Å: "Referat och reflektioner från konferensen 3rd Reverse Engineering Forum", SISU, 1992

[16] David, M. and Marietta, M."Upper CASE Reverse Engineering: Applicable Applications", CASE Outlook, Vol 6, No 1, 1992

[17] Eschallier, P. et al. "Reverse Engineering of NASA Fortran Applications", in [55]

[18] Forte, G. "Reverse Engineering Tools. For work station, PC and technical applications", CASE Outlook, Vol 6, No 2, 1992

[19] Forte, G. "Re-engineering Tools: A Spectrum of Objectives and Capabilities", CASE Outlook, Vol 6, No 3, 1992

[20] Forte, G. "Software Maintenance Under the CASE Umbrella", CASE Outlook, Vol 6, No 1, 1992

[21] Frazer, J.A. "Reverse Engineering. Hype, Hope or Here?", The Institute of Software Engineering, Belfast, 1991

[22] Gardner, B. "AD/Cycle Maintenance ReDevelopment Strategy", in [55]

[23] Glasier, P et al. "Survey and Analysis of Business Rule Extraction Techniques", in [55]

[24] Hagwall, H. "Återanvändning av programvara – utvecklingen i USA", Utlands-rapport, Sveriges Tekniska Attachéer, 1992

[25] Hanna, M.A. "Defining The "R" Words for Automated Maintenance", Software Magazine, May 1990

[26] Hefedh, M., Fatma, M., Ali M. "Reusing Software: Issues and Research Directions", Departement de Mathématique et d'Informatique, Université de Quebec a Montréal, School of Engineering and Computer Science, Oakland University, University of Tunis II, Tunisia and University of Ottawa, 1992

[27] IEEE Software (ISSN 0740-7459), January 1990. Temanummer om Software Maintenance, Reverse Engineering & Design Recovery.

[28] IT 2000, Effektiv IT, Förutsättningar för ett nytt utvecklingsprogram inom informationsteknologins tillämpningsområden, En förstudie, Ds 1993:43, Näringsdepartementet, Regeringskansliets offsetcentral, 1993.

[29] Johannesson, P. et al. "En kokbok i remodellering – utkast", TRIAD Arbetsrapport N 5, 1991

[30] Johansson, L-Å och Gustafsson, M.R. "Affärsmässiga scenarier som bakgrund till reengineering av informationssystem", Effektiv-IT, Systemarvet, Rapport Nr 3, SISU, 1994

[31] Jones, R. "Business Software Review", Jan/Feb 1988

[32] Kalman, K. "Reverse modelling from Relational Schemata to Entity-Relationship Schemata", SISU Rapport nr 11, 1991

[33] Kerr, J. and McGovern, T. "The Three R's of IS", Database Programming and Design, October 1991.

[34] Khakhar, D (Ed). "Systemförvaltning", Studentlitteratur, Lund 1987

[35] Kortesoja, A. "Redevelopment Engineering – A CASE Strategy for Existing Systems", in [55]

[36] McCabe, T.J., Williamson, E.S., "An Engineering Approach to Software Maintenance", CASE Outlook, Vol 6, No 1, 1992

[37] McClure, C. "The Three R's of Software Automation. Reengineering, Repositories, Reusability", Extended Intelligence, Inc., 5000 South East End Avenue, Chicago, IL 60615

[38] Munro, M. "The Software Maintenance Context", Proc of From Software Maintenance Towards Reverse Engineering and Beyond, June 1990, Blenheim Online Ltd

[39] Olsem, M.R. "Developing a JOVIAL Reverse Engineering Toolset", in [55]

[40] Pfrenzinger, S.J. "Reengineering: How High and Why?", Database Programming and Design, July 1992

[41] Proceedings of ERCIM Workshop on Methods and Tools for Software Reuse, Institute of Computer Science, Heraklion, Crete, Greece, October 29-30, 1992

[42] Riksdataförbundet, Systemförvaltning, Rapport 26:1-26:6, (1. Generell modell, 2. Organisation och styrning, 3. Metoder och hjälpmedel, 4. Beslutsunderlag och införande, 5. Systemdiagnos, 6. Anvisningar för handbok), 1987

[43] Rock-Evans, R. och Hales, K. "Reverse Engineering: Markets, Methods and Tools", Ovum Report, 1990

[44] Rouve, C. "Using Program Analysis Tools to Understand Existing Software", CASE Outlook, Vol 6, No 2, 1992

[45] Sharon, D. "Developing and Applying a Tool Classification Scheme and Evaluation Criteria for Reverse Engineering and Re-Engineering Tools", in [55]

[46] Smart, J.C. and Rao Vemuri, V. "Aesthetic Graph Layout for Program Understanding", in [55]

[47] Sneed, H. "Software Renewal – A Case Study", IEEE Software, July 1984

[48] Sneed, H. "The Economics of re-Engineering", Proc. of Fourth Software Maintenance Workshop, Sept. 1990, Centre for Software Maintenance Ltd., University of Durham, England

[49] Sneed, H. and Kaposi, A. "A Study on the Effect of Reengineering Upon Software Maintainability", Proceedings Conference on Software Maintenance, 1990

[50] Samuelson, P. "Reverse Engineering Someone Else's Software: Is It Legal?", IEEE Software, Jan. 1990

[51] "Software Maintenance Technology Reference Guide", Nicholas Zvegintzov Tech. ed., Software Maintenance News Inc., B10-Suite 237, 4546 El Camino Real, Los Altos, CA 94022, USA

[52] "Software Reuse: Emerging Technology", Will Tracz ed., IEEE Computer Society Press, Los Alamitos, CA

[53] Tilley, S.R. and Müller, H.A. "Spatial and Visual Representations of Software Structures. A Model for Reverse Engineering", in [55]

[54] Ulrich, W. "Re-development Engineering: Formulating and Information Blueprint for the 1990's", CASE Outlook, No 2, 1990, pp 15-23

[55] "Understanding enables Reengineering", Proceedings of the 3rd Reverse Engineering Forum, Sept. 15-17, 1992, Northeastern University, Burlington, Massachusetts, USA, 1992

[56] Walston, Felix. "A Method of Programming, Measurement and Estimation", IBM Systems Journal, Vol 17, No. 1, 1977

[57] Weizman, D. "Integration by Re-engineering", CASE Outlook, Vol 6, No 1, 1992

[58] Zvegintzov, N. "Re-engineering", Software Maintenance News, February 1991, Vol 9, No 2, pp 12-19

Effektiv IT-rapporter

Nr 1 Att Mäta Informationsteknologi – Data om IT i Sverige och utomlands,
Mattias Hällström, december 1993. *IT:s Ekonomi & Management*

Nr 2 Mätning för Effektiv Systemutveckling,
Tapani Kinnula, mars 1994. *Systemutvecklingens Ledtider & Kvalitet*

Nr 3 Affärsmässiga Scenarier som bakgrund till Reengineering av Informationssystem,
Lars-Åke Johansson, Mats R Gustafsson, mars 1994. *Systemarvet*

Nr 4 Concepts and Notations for Open-edi Scenarios,
Matts Ahlsén, mars 1994. *Affärskommunikation*

Nr 5 Business Process Reengineering – vad är det?
Mattias Hällström, april 1994. *Verktyg för Verksamhetsutveckling*

Nr 6 Managing Information Technology: The Capital Budgeting Process,
Thomas Falk, Nils-Göran Olve, maj 1994. *IT:s Ekonomi & Management*

Nr 7 Integrerad Systemutveckling – lärdomar från industrin tillämpade på
systemutveckling, Sten-Erik Öhlund, Lars Bergman, maj 1994.
Systemutvecklingens Ledtider & Kvalitet

Nr 8 Kunskap för hantering av systemarvet – en första systematisering,
Lars-Åke Johansson, Mats R Gustafsson, Roland Dahl, juni 1994. *Systemarvet*

Svenska Institutet för Systemutveckling, SISU, bedriver forskning, följer utvecklingen och förmedlar kunskap om informationsteknologins tillämpning på informationsanvändning och informationsförsörjning i företag, myndigheter och andra organisationer. Institutet verkar inom detta område som ett opartiskt nationellt kompetenscentrum.

SISU

Electrum 212, 164 40 Kista
Isafjordsgatan 26
Telefon 08-752 16 00 Telefax 08-752 68 00