Vilken version av vad är det som körs?

Lätt att tappa kontrollen över verksamhetslogiken

2012-01-11: Sven-Håkan Olsson

VERSIONSHANTERING När det gäller traditionella program har man oftast bra koll när det gäller vilken version som körs. Men idag manifesteras verksamhetsprocesserna i logik implementerad på många fler sätt.

Många kan nog minnas första gången man som ung programmerare insåg "Aj, aj, exakt vilken programversion är det nu igen som är drift?" – och än värre, ”Hur ska man kunna snabbt rätta driftbuggen när man har satt igång att vidareutveckla koden och den nu överhuvudtaget inte är körbar, eller ens kompilerbar?”. Man lär sig ibland den hårda vägen.

Lösningen är förstås versionshantering och i sin mer avancerade form CM, Configuration Management. Med sådana verktyg går det att hålla reda på vilka olika moduler som finns installerade hos en viss kund, hur modulers olika versioner förhåller sig till varandra, hur man kan slå ihop versioner och mycket annat.

En helhet som kan driftas kallas ofta paket, de finns i en viss version och består i sin tur av en samling moduler av olika versioner. Det är ofta på paketnivå man sluttestar och verifierar att mjukvaran är användbar och håller tillräcklig kvalitet.

Så långt är det ganska enkelt, även om det ger upphov till en del tråkig men nödvändig byråkrati. Programkoden anses representera logiken som ska ge verksamhetsnytta och man måste hålla ordning på koden. Men hur ofta är det egentligen så enkelt inom dagens it?

Verksamhetslogik i en integrationsbuss också?

Säg att vi som exempel tillför en integrationsbuss (ESB, eller ett nav). Inne i en sådan produktsvit kan man utan vanlig programmering definiera övergripande verksamhetsregler, arbetsflöden och datatransformeringar.

Integrationslösningarna säljs ofta in på att man snabbt och flexibelt ska kunna anpassa logiken till förändrade verksamhetskrav, lite i motsats till rutinen kring traditionell programmering där man måste CM-hantera och göra formell systemtestning och som ofta upplevs långsam.

Men vänta nu, om vi menar att det är en helhet av VERKSAMHETSLOGIK som sannerligen borde CM-hanteras och testas, ja då ingår ju den logik som definierats inne i bussen, inte bara i den vanliga programkoden!

Inom bussar och andra integrationsprodukter är variationen stor. Det är inte säkert att det överhuvudtaget går att skilja ut en entydig version av logiken. Definitionerna kanske lagras som en massa fält utspridda i en databas (medan CM vanligen bäst görs på textfiler). I andra fall lagras de om vartannat i jättestora xml-filer eller å andra sidan i alldeles för många små konfigurationsfiler.

Hur ska man på ett bra sätt kunna CM-paketera detta, i sig? Hur ska man kunna överföra definitioner från en testbuss till produktionsbussen? Hur ska man kunna paketera tillsammans med ansluten traditionell programkod? Måste allt detta driftinföras samtidigt?

Det kanske till och med är svårt att komma tillbaka till senaste fungerande versionsläge om man skulle ha klickat bort sig i bussverktyget vid ändring i produktion (naturligtvis olämpligt i sig) - och då riskerar hela företaget att stanna om man utnyttjat reklamens budskap om att centralisera alla viktiga dataströmmar till bussen.

Flexibiliteten har gett sämre ordning och reda. Något som ioch för sig ett vanligt motsatspar.

Ibland känns det faktiskt som att man tar till användningen av regelverk i en buss för att kringgå CM och för att man upplever att ambitiösa testningsprocedurer kostar mycket tid och pengar. Förresten, då kanske man till och med har en ALLTFÖR byråkratisk CM och testning, som därmed övergripande sett motverkar sitt eget syfte!

Numera börjar man för övrigt inom ”agila” sammanhang tala om Continuous Integration och Continuous Delivery *) som anses kunna möjliggöras bland annat tack vare automatisk testning, men då måste man veta vad det verkligen är för andel av hela verksamhetslogiken man egentligen testar, enligt resonemanget ovan.

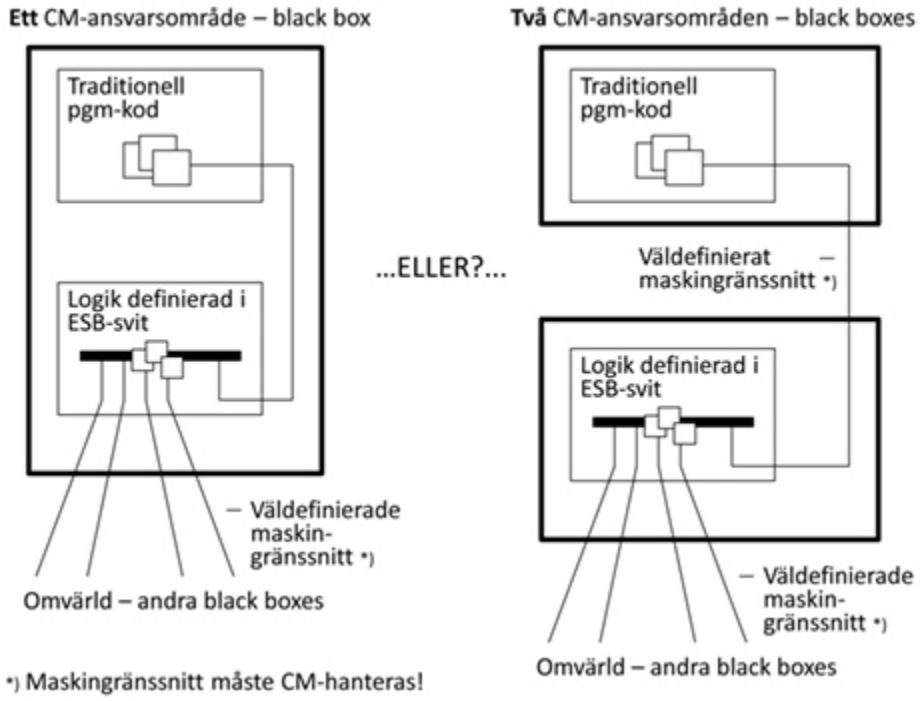
Automatisk testning har också sina begränsningar. Dels formaliseras testfallen ibland fel, dels är användargränssnittskod (ofta en förvånansvärt stor andel av den totala programkoden) också mycket svår att automattesta.

Ansvarsblock

Någon viss version av bussdefinitioner passar vanligen bara ihop med en viss version av programkod som interagerar med bussen. Då blir det en fråga om vilka ansvarsblock man kommer överens om. Om busslogiken och programkoden ska ingå i samma ansvarsblock så ska de CM-hanteras tillsammans. Om man kommer överens om att busslogiken ska vara ett eget ansvarsblock blir det definitionerna av maskingränssnitt som blir heliga, enligt black box-principen *).

Och, mycket viktigt, gränssnittsdefinitionerna måste versioneras och CM-hanteras! Och då menar jag versionshanteras på riktigt, inte bara ha en fasad som utgör tratt för diverse verb man sen kan skicka in – javisst, flexibelt men vanligen alldeles för dålig CM-hantering för att ge ordning, testbarhet och lösningskvalitet.

För övrigt kan nämnas att många större standardapplikationer idag inkluderar en integrationsbuss. Har man många standardapplikationer kanske man med andra ord måste acceptera en ”federation av bussar”!



Verksamhetslogik på ytterligare en mängd ställen?

Samverkan mellan en integrationsbuss och traditionell programlogik är bara ett av en mängd tänkbara fall där verklig och sammanhållen CM (och testbarhet) kan bli svår. Några andra exempel på "utspridd logik":

- Konfigurationsfiler kan ofta styra mängder av logik
- Användningen av en separat regelmotor
- Dokument- och ärendesystem som innehåller workflow-definitioner
- Regler som sätts upp i ett ERP-standardsystem. Ett stort system som SAP har många, många tusen parametrar som styr verksamhetslogiken. Sådana lagras ofta direkt i databasen.
- Excelfiler, kan innehålla helt enorma logikregelverk, ofta utan någon som helst CM
- Kontosträngar som sätts upp i ett ekonomisystem påverkar kraftfullt hur logiken blir och motsvarande gäller löneartsdefinitioner.
- Logik som styrs av definitioner i Windows registry eller motsvarande
- Saker som definieras i en LDAP/AD-katalog eller i smartcard och som leder till logikregler
- E-blanketter och PDF-formulär kan innehålla script och andra logikdefinitioner
- Data warehousing och olika statistiksystem, som brukar både innehålla logik för datahämtning, ha en styrande datamodell och förstås ha logik för sammanställning.

Samtidigt finns det en intressant gränsdragning mellan olika sorters definitioner av regler/verksamhetslogik och vanligt dynamiskt verksamhetsdata.

Ett första påhittat exempel: Vad som ska göras när ett fakturanummer börjar på 12 kanske definieras i regler i en integrationsbuss (de fakturorna tänks hanteras i en separat internationell faktureringsrutin). Bör troligen anses vara en verksamhetsregel (som visserligen kan ändras längre fram, och då borde CM-hanteras).

Ett exempel i andra änden av skalan: Hur många sängar finns i ett visst patientrum på ett sjukhus (styr logiken för utplacering av patienter), är det verkligen konfigurationsdata eller är det verksamhetsdata? Det kanske snabbt ska gå att stoppa in en extrasäng och en sådan ändring kan inte praktiskt CM-hanteras och testas, utan logiken ska i sig kunna hantera antal sänger per rum som verksamhetsdata. (Att man däremot bör ha historik i sin databas är en helt annan, men också viktig fråga.)

Om en hel lösning sammansatt av många sorters logik-komponenter samt traditionell programkod ska kunna CM-hanteras och testas så behöver alltså komponenterna ovan vanligen kunna använda eller exportera/importera definitionsfiler.

Ovan nämnda black box-princip kan vara rimlig att använda för att undvika en överkomplex CM, men då gäller sannerligen att både komponentens logikdefinitioner och maskingränssnitten tydliggörs och versionshanteras i sig. Så är ofta inte fallet idag (tänk bara på alla gigantiska Excelkalkyler som inte är kvalitetshanterade).

Så min slutsats blir att man ofta har jättebra kvalitetskontroll på den traditionella programkoden, men dålig på all den verksamhetslogik som finns definierad i alla de andra komponenterna.

*) Lästips:

- Black box-principen tas bl.a. upp i min tidigare trendspaning Hur den lösa kopplingen ändå blir hård
- Sök på Internet om Continuous Integration (CI) och Continuous Delivery (CD), exempelvis på Wikipedia. Som jag nämner ovan är det dock inte alltid helt lätt att enligt CI/CD- principerna tackla problemen jag tagit upp i artikeln.

Sven-Håkan Olsson sysslar just nu med en iPad-tjänst för bolagsstyrelser och nämnder. I övrigt är han oberoende konsult som särskilt arbetar med att kombinera verksamhetsnytta med teknikhöjd. Han har en lång karriär sedan 70-talet som it-konsult (it-arkitektur, systemdesign, programmering, reviewer, utredningar, kursledning). Sven-Håkan är också medgrundare av Know IT och var dess teknikchef 1990-2003. Han utsågs till en av "Sveriges topputvecklare" av Computer Sweden 2008 och 2010.

Sven-Håkan håller regelbundet kurser åt Dataföreningen Kompetens samt kommer att vara moderator på vårens Patterns-konferenser. Läs gärna mer på hans blogg www.definitivus.se.

Sven-Håkan Olsson