EKONOMIHÖGSKOLAN
Lunds universitet
Institutionen för Informatik

Komponenter och generalitet

– Planering för design av generella programvarukomponenter

INF101 – Magisteruppsats
Våren 2005

Handledare:
Erik Persson

Författare:
Anders Olsson
Patrik Palmér

Components and Generality

– Planning the design of general software components

Authors: Anders Olsson & Patrik Palmér
Department of Informatics, Lund University
Master thesis presented: June 2005
Span: 60 pages
Tutor: Erik Persson

Abstract

Background: Development of software components is a way of enabling reuse of common functionality in application programs and consequently brings opportunities for many organisations to lower their development costs. In order for software components to be eligible for reuse, they must be adaptable, general, and functional in different contexts. However, such development takes far longer than that of regular, application-specific components, since planning – amongst other things – requires higher standards. For the development of software components to be profitable, it must be made certain that the use of the component will eventually pay off the longer process of development.

Aim: The aim of this thesis is to investigate how the design of software components is being planned in relation to generality.

Method: The research was done through qualitative interviews with systems developers and a doctoral candidate within the field of components.

Conclusions: It seems as if many organisations are cautious of developing software components, since these can be harder to find use for: Components that have a high level of generality are more complex, and thus, harder to use.

The administration surrounding reusable components is a large investment and most companies seem unwilling to spend the money needed and therefore continue developing specific components, even though long-term benefits may be lost.

Good and pragmatic, albeit limited, analyses that draw from previous experiences are often done and they result in that general components are only developed when there is a clear-cut application of the component. The readiness to develop general components seems to increase the fewer the domains to develop for are and the more product-specific the development is.

Context is a central element when deciding on whether to develop a new component, or simply modify an old one. Compatibility with earlier versions of a component is often crucial, and new versions of a component are usually constructed instead of modifying older versions. The old components are usually left to serve in the systems that still use them.

Keywords: Software component, generality, reusability, planning, software reuse

Komponenter och generalitet

– Planering för design av generella programvarukomponenter

Författare: Anders Olsson & Patrik Palmér
Institutionen för informatik, Lunds universitet
Magisteruppsats framlagd: Juni 2005
Omfång: 60 sidor
Handledare: Erik Persson

Sammanfattning

Bakgrund: Utveckling av programvarukomponenter görs för att möjliggöra återanvändning av gemensam funktionalitet i applikationer och därigenom minska organisationers utvecklingskostnader. För att komponenter ska kunna vara möjliga att återanvända så krävs det att de är utvecklade med viss anpassningsbarhet – generalitet – och på så sätt kan komponenten vara funktionell i olika kontexter. Dock kan det ta mångdubbelt mer tid i anspråk i jämförelse med att skräddarsy en komponent för en viss applikation, då det ställs höga krav på bland annat planeringsarbetet. Om utvecklingen av programvarukomponenter ska vara lönsam måste man försäkra sig om att komponenten kommer att användas till den grad att den längre utvecklingstiden är befogad.

Syfte: Målsättningen med uppsatsen är skapa en förståelse för hur design av programvarukomponenter planeras med avseende på komponenters generalitet.

Metod: Uppsatsen genomfördes genom kvalitativa intervjuer med systemutvecklare och forskare inom komponentutveckling.

Slutsatser: Det verkar som att många organisationer är försiktiga med att utveckla generella komponenter, då dessa anses vara svåra att hitta användningsområden för: Komponenter som har en hög grad av generalitet är mer komplexa och därför svårare att använda.

Organisationen runt återanvändbara komponenter innebär en stor investering och många företag verkar ovilliga att satsa så mycket resurser på en gång och väljer därför att utveckla specifika komponenter istället, trots att långsiktiga vinster kanske går förlorade.

Ofta görs bra men restriktiva analyser som är pragmatiska och till stor del bygger på tidigare erfarenheter och man utvecklar endast generella komponenter i de fall man vet att de kommer att användas. Benägenheten att utveckla generella komponenter verkar öka ju färre domäner utvecklingen sker för samt ju mer produktspecifik utvecklingen är.

Kontexten är en avgörande faktor när man bestämmer om man ska utveckla en ny komponent eller modifiera en gammal. Eftersom bakåtkompatibiliteten ofta är mycket vikig, utvecklas ofta nya komponenter istället för att gamla modifieras. De gamla komponenterna får leva kvar i de system som använder dem.

Nyckelord: Programvarukomponent, generalitet, återanvändbarhet, planering, återanvändning av programkod

Innehållsförteckning

Figur- och tabellförteckning

Introduktion

Introduktionskapitlet ger en kort bakgrund till ämnet och presenterar det övergripande syftet med tillhörande frågeställningar och avgränsningar.

Inledning och problembeskrivning

Ökad återanvändning av programkod har sedan 1960-talet ansetts som en av de mest realistiska förändringarna för att minska programvarukostnader och förbättra effektiviteten vid utveckling av informationssystem (Mili m.fl., 2002, s. xxi). Många informationssystem besitter gemensam funktionalitet och därför är det naturligtvis ytterst intressant att också kunna utnyttja denna. Grundidén för återanvändning av funktionalitet genom programvarukomponenter är enkel – att konstruera komponenter med en rimlig storlek och att återanvända dessa (Jacobson m.fl., 1997, s. 6). Genom utnyttjandet av redan konstruerade komponenter, som i många fall redan varit föremål för systemtest i andra projekt och fått en ökad tillförlitlighet genom användning, skapas stora möjligheter till att minska kostnaderna för programvaruutveckling.

Eventuell vinst från komponentbaserad systemutveckling kommer ur att snabbare och billigare kunna utveckla applikationer. Att göra generella komponenter är dock betydligt mer resurskrävande än att göra specifika komponenter och om utvecklingen ska vara lönsam måste komponenterna användas i så hög grad att kostnaden för den längre utvecklingstiden kompenseras. För att uppnå en hög grad av återanvändbarhet hos komponenten är generaliteten, det vill säga anpassningsbarheten, en av de viktigaste faktorerna att ta hänsyn till. Det är generaliteten som bestämmer vilken möjlighet komponenten har att användas på nytt i andra sammanhang. Motsatsen, den specifika komponenten, är skapad för att verka i en eller ett fåtal kontexter och innehåller därför färre möjligheter till anpassning, samtidigt som den är mindre resurskrävande att utveckla.

Det är svårt att veta vilka krav som kommer att ställas på applikationer framöver, men vid utveckling av komponenter bör sådana analyser göras, då man hela tiden måste ligga steget före och tänka på den programutveckling som kommer att ske bortom de närmaste projekten. Att utveckla en generell komponent för återanvändning istället för att skräddarsy en komponent för endast en applikation kan ta flera gånger så lång tid. Hur kan man då försäkra sig om att en viss komponent verkligen kommer att kunna användas till den grad att den längre utvecklingstiden är befogad?

Syfte

Syftet med magisteruppsatsen är att skapa förståelse för hur design av programvarukomponenter planeras med avseende på generaliteten.

Frågeställningar

- Hur planerar organisationer sin komponentutveckling för att skapa komponenter generella nog för återanvändning?
- Vilka problem kan uppstå beroende på planeringen av komponenters generalitetsgrad och hur hanteras dessa?

Avgränsningar

Planering av komponentutveckling avser de analyser – formella eller informella – som främst föregår konstruktion av komponenter, men även den planering som uppstår då komponenter, av någon anledning, modifieras.

Uppsatsens primära fokus är att undersöka hur det planeras för generalitet – en faktor som till stor del påverkar användbarheten hos en komponent och som ofta indikerar till vilken grad man har tänkt att komponenten ska återanvändas. Dock finns det andra aspekter som också påverkar komponenters möjlighet till återanvändning, men dessa kommer endast att diskuteras översiktligt.

Disposition

Uppsatsen börjar med en genomgång av de teoretiska avsnitt som ligger till grund för analysen av undersökningsresultaten. Detta kapitel inriktas till den största delen på de områden analysen baserades på men förklarar och fördjupar också vissa delar som är intressanta för den allmänna förståelsen för området.

Metodkapitlet presenterar strategier och metoder som användes i samband med undersökningen samt en modell över hur frågeställningarna besvarades.

Resultat och analyskapitlet inleds med en analysmodell, som visar utifrån vilken teoretisk bakgrund resultaten tolkades. Sedan följer en presentation av resultaten och analysen av dessa.

Slutsatserna fungerar som en sammanfattning och en vidare diskussion av de vikigaste analyspunkterna och svarar på respektive frågeställning.

Uppsatsen avslutas med ett kort kapitel med förslag till fortsatt forskning.

Utveckling av programvarukomponenter

Det här kapitlet redogör för olika teoretiska resonemang inom ämnesområdet. Kapitlets syfte är att vara en teoretisk plattform för undersökningen och tolkningen av undersökningsresultaten.

Bakgrund

Utveckling av programvarukomponenter handlar till stor del om att kunna fånga kunskap från verksamheten och införa den i andra sammanhang för att lösa samma typ av problem (Barnes & Bollinger, 1994, s. 14). Utveckling och användning av komponenter ger självklara resursfördelar men ställer samtidigt höga krav på både utvecklare och verksamhetsstrukturer. En central del av utvecklingen rör komponentens möjlighet att vara anpassningsbar och på så sätt möjlig att nyttja i andra applikationer. För att åstadkomma denna benägenhet för förändring hos komponenten krävs en längre utvecklingstid – både på grund av att det är svårare att konstruera denna typ av generella komponent men också på grund av att det går åt resurser att planera för den funktionalitet som man har behov av att återanvända.

Komponenter har varit ett populärt ämne för diskussion inom programvaruforskningen under en lång tid. Ändå menar många forskare att dagens programvaruutvecklare inte praktiserar denna typ av återanvändning i någon större omfattning, och att det finns mycket att lära från andra områden som har lyckats betydligt bättre med att hitta lösningar på samma problematik (Jacobson m.fl., 1997, s. 5).

Den tidiga forskningen koncentrerades mest på hur programvarukomponenter bäst klassificeras i bibliotek och kommer till användning snarare än att fokusera på själva innehållet. På grund av detta avstannade mycket arbete och forskningen styrdes bort från den viktigaste frågan, nämligen vad komponentbiblioteken faktiskt innehöll (Gall m.fl., 1995, s. 225). En del forskare menar att forskningsområdet äntligen börjar närma sig den mognad som krävs för att tankarna om en gemensam komponentindustri ska kunna realiseras i praktiken:

> In recent years the software world has developed the technology and initiated the standards that bring computer-based software engineering from the dream stage to the level of practicality. (Jacobsson, 2001, s. xiii)

Vad är en komponent?

Komponentbegreppet inom programvaruindustrin myntades av McIlroy under ett tal han höll 1968, under en berömd NATO-konferens i Garmisch, Tyskland (McIlroy, 1968, s. 138-150). McIlroy gillade inte det faktum att programvaruindustrin stod lågt på industrialiseringsskalan jämfört med maskinvaruindustrin och han var intresserad av idén att kunna massproducera olika programvaror med hjälp av färdigutvecklade byggblock. McIlroys idé var att istället för att fråga sig vad man bör utveckla, bör man fråga sig vad man bör använda (1968).

McIlroy är tydlig med att ingen bör straffas med oönskat hög grad av generalitet för att denne använder sig av en standardkomponent. Istället tänkte sig McIlroy att man beställer en komponent som passar perfekt till de behov som finns. McIlroy tänkte sig att de olika komponenter man använder passar varandra utan att man som programvaruutvecklare ska behöva modifiera dessa själv:

> He will expect families of routines to be constructed on rational principles so that families fit together as building blocks. In short, he should be able to safely regard components as black boxes. (1968)

Even-Andre Karlsson skriver om ungefär samma visioner långt senare – dock verkar det fortfarande mest röra sig om just visioner: ”Software development will evolve from the composition of simple code statements to the synthesis of large components from smaller ones” (1995, s. 249).

Förutom komponenter finns det flera sätt att återanvända programkod på, som till exempel renodlade källkodsbibliotek och designmönster (Szyperski m.fl., 2002, s. xxi). Vad är då en komponent? Det finns en mängd olika definitioner på vad en komponent är och det finns därför ingen anledning att skapa ytterligare en. Programvarukomponenter anses vara återanvändbara delar av programvara. Detta behöver dock inte medföra att komponenten faktiskt använts eller kommer att återanvändas, men att det finns ett syfte för återanvändning av denna. Vi an-

vänder Szyperskis något snäva definition för att särskilja programvarukomponenter från andra typer av återanvändning:

> Software components are binary units of independent production, acquisition, and deployment that interact to form a functioning system. (2002, s. xxi)

Beroende på hur komponenten är planerad att användas finns flera typer av komponenter: En komponent som återanvänds utan anpassningar kallas populärt för *svart lådkomponent* (black box component) medan en *vit lådkomponent* (white box component) är en komponent som återanvänds med större förändringar. När man använder vita lådkomponenter känner man till källkoden och kan även modifiera denna (Persson, 2002, s. 24).

Komponentstandarder

En grundläggande idé med programvarukomponenter är att de ska kunna användas i flera olika projekt och av flera olika företag. McIlroy talar om hur viktigt det är att komponenter fungerar väl tillsammans (1968), vilket är omöjligt om det inte finns gemensamma standarder. Standarder ska inte enbart beskriva hur gränssnittet ser ut gentemot andra komponenter, utan även vilka avvikelser som får förekomma (Szyperski m.fl., 2002, s. 28).

Det finns flera företag och konsortier som arbetar med olika komponentstandarder och de mest framstående är Microsoft med OLE (Object Linking and Embedding), Active X, COM (Component Object Model), COM+ och, .NET Sun med Java, JavaBeans, EJB (Enterprise JavaBeans) och J2EE (Java 2 Platform, Enterprise Edition) samt OMG (Object Management Group) med CORBA (Common Object Request Broker Architecture) och CCM (CORBA Component Model) (Szyperski m.fl., 2002, s. 28).

Komponentorienterad systemutveckling

Programvarukonstruktion baserad på utveckling och användning av komponenter benämns ofta som komponentorienterad systemutveckling. Denna bygger på planering och utveckling av generella artefakter som sedan kombineras och knyts samman med applikationsspecifik programkod för att bilda kompletta applikationer. Att utveckla komponenter ställer högre krav på planering och konstruktion än vad applikationsspecifik utveckling gör och detta beror på att funktionaliteten i komponenterna måste vara möjlig att kombinera. Programkoden måste vara utvecklad med hänsyn till allmängiltighet så att den är återanvändbar och är möjlig att anpassa till andra komponenter (Karlsson, 1995, s. 257). För att komponenter ska vara återanvändbara krävs att de inte extraherats ur annan programkod, utan att de från början konstruerats som fristående entiteter och för att vara återan-

vändbara. Det har visat sig att dylik extrahering tar så mycket resurser i anspråk att den inte är lönsam (Gall m.fl., 1995, s. 225). Gall, Jazayeri och Rerk menar att man därför inte bör prata om återanvändning av programkod, utan snarare om användning av komponenter (a.a., s. 225).

Vid komponentutveckling skiljer man mellan två typer av funktionaliteter: Dels den *konstanta funktionaliteten* (invariant functionality), och dels *applikationsspecifik funktionalitet* (variant functionality). Per definition kan inte den konstanta funktionaliteten ensam utgöra en applikation, eftersom det alltid krävs vissa anpassningar till kontexten. Oavsett vilken komplexitet eller storlek ett informationssystem som bygger på komponenter har, finns alltid dessa två typer av funktionalitet representerade (Barnes & Bollinger, 1994, s. 20).

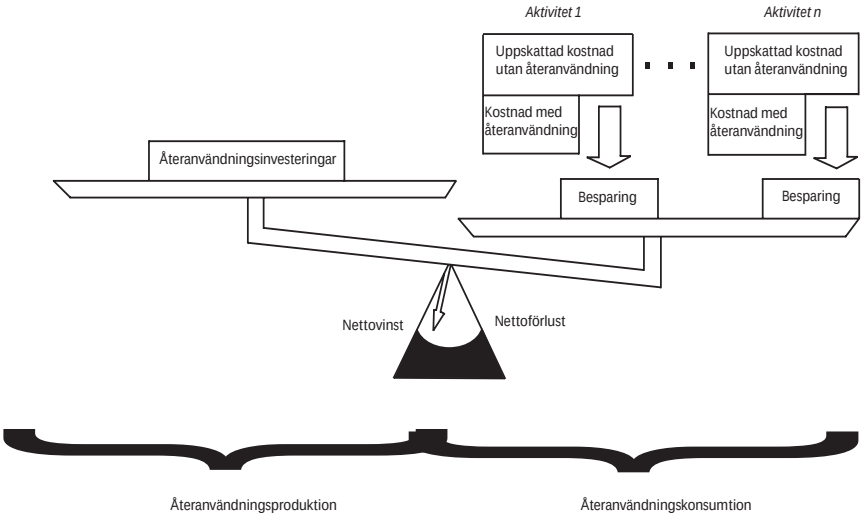
Komponentutvecklingens ekonomiska för- och nackdelar

Möjligheten att kunna använda delar från väl fungerande informationssystem som tidigare utvecklats ger självklara fördelar. Detta förväntas vara en av de främsta källorna till kostnadsreduceringar för många programvaruutecklande verksamheter under en lång tid framöver (Karlsson, 1995, s. 9). De främsta fördelarna som är sammankopplade med framgångsrik komponentbaserad systemutveckling i jämförelse med traditionell utveckling av informationssystem, är framförallt ökad produktivitet, kortare time-to-market samt bättre kvalitet (Mili m.fl., 2002, s. 506-510). Den ökade produktiviteten kommer från att utvecklingstiden blir kortare när programvarukomponenter används (a.a., s. 507f) och genom en förkortad time-to-market kommer projektet att göra vinst snabbare: ”[…] collecting earlier means also collecting more” (a.a., s. 509). Genom att marknaden möts tidigare skapas möjligheter att nå ut till en större marknadsandel, beroende på att man kan knyta förbindelser med konsumenterna tidigare (a.a., s. 509). Genom att utveckla komponenter istället för enbart applikationsspecifik programkod skapas också möjligheter att sälja komponenten till andra företag och på sätt få igen en del av utvecklingskostnaden (Szyperski m.fl., 2002, s. 17). Brad Cox talar lyriskt om hur man kan starta små och specialinriktade företag som utvecklar komponenter, utan att behöva den kompetens som krävs för att kunna bygga stora och komplicerade system (1992). Problemet med det här resonemanget, menar Cox, är att det är mycket svårare att distribuera information än fysiska objekt och det största problemet är att information är alltför reproducerbar, vilket gör att det är svårt att tjäna pengar på den (a.a.). Istället föreslår Cox att system konstrueras för att övervaka hur mycket en komponent används och man betalar för användningen istället för via licenser. Cox kallar det för *superdistribution* och han menar att digital information bör finnas fritt tillgängligt och att man sedan betalar för det man använder (a.a.).

En del av den utvecklingstid som kan besparas genom att utveckling och användning av komponenter även rör testning av programvara. Applikationerna kan

snabbare nå en god kvalitet och hög säkerhet då de bygger på användning av komponenter – både på grund av att komponenten testas i verkliga system och på att kostnaden för kvalitetskontroller delas upp på flera projekt (Mili m.fl., 2002, s. 501-503; Rine & Nada, 2000, s. 52).

I takt med att organisationens komponentbibliotek byggs ut, kommer utvecklingstiden successivt att minska. Beroende på mångfalden av produkter och på storleken på organisationen rör det sig tidsmässigt om mellan ett och fem år för att uppnå den maximala vinsten för en komponent (Karlsson, 1995, s. 4). Dock planar resursbesparingarna ut då biblioteket når en viss storlek, vilket indikerar att biblioteket blir svårare att använda när det når en kritisk massa (Nazareth, 2004, 252f).



Figur 1: Relation mellan återanvändning och investeringar (Barnes & Bollinger, 1994, s. 15)

Figur 1 illustrerar hur de initiala investeringarna för återanvändningen väger tungt på vågen. För varje gång komponenten används väger dock vinsten upp mer och mer och till slut pekar indikatorn mot nettovinst för utvecklingen av komponenten.

De eventuella vinster som uppkommer från komponentorienterad utveckling syns inte i det kortare perspektivet och istället får man en högre utvecklingskostnad. Detta gör att de initiala projekten tenderar att vara mer kostsamma än de projekt som kan dra en större nytta från tidigare utveckling. Rine och Nada talar om *reuse creation cost* för en komponent, vilket innebär ”the costs of purchasing the compo-

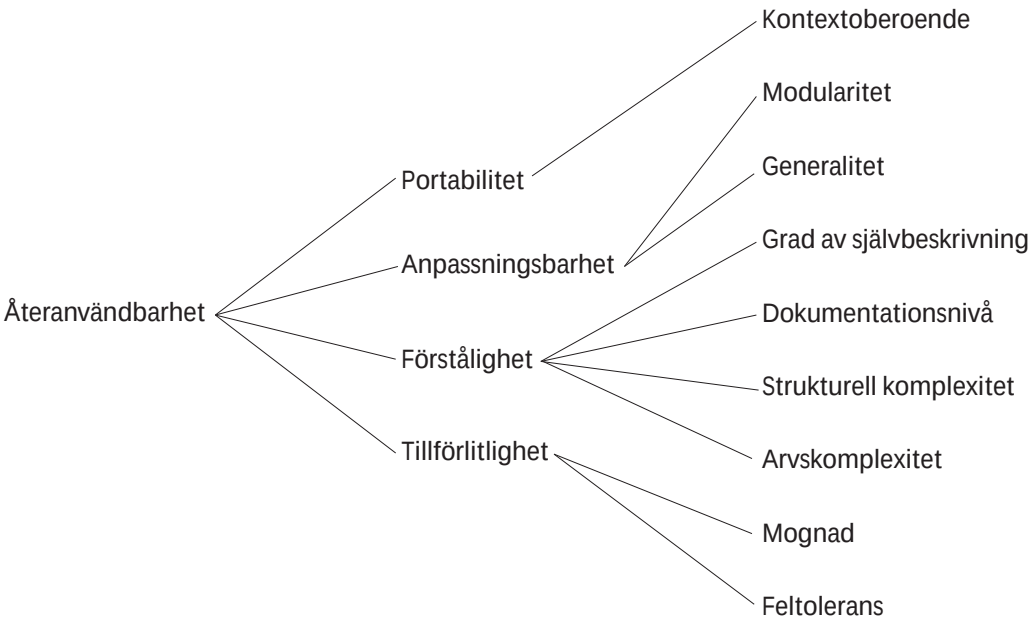
nent, identifying it via domain analysis, and preparing it for reuse (i.e. generalization, standardization, certification, classification, documentation)" (2000, s. 53).

Kostnaden att utveckla och återanvända en komponent beror till stor del på den tillåtna feltoleransen för den komponentstandard som används (Szyperski m.fl., 2002, s. 28). Om feltoleransen är låg, betyder det att mer resurser måste läggas på att uppnå enlighet med komponentstandarden.

Den största delen av den totala kostnaden för utveckling av programvara består i regel av utvecklarnas löner. Dock finns det andra kostnader än de som uppstår vid utveckling av återanvändbara komponenter som istället är associerade med konsumtionen av komponenterna. Dessa kostnader rör bland annat sökning, anpassning, integrering och testning av komponenterna (Barnes & Bollinger, 1994, s. 18).

Komponenters återanvändbarhet

Man värderar ofta nyttan med en komponent utifrån dess grad av *återanvändbarhet* (reusability). Återanvändbarhet är en kombination av två karakteristika – *lämplighet* (usefulnes) och *användbarhet* (usability) (Mili m.fl., 2002, s. 122). Lämplighet indikerar hur stor användning man har för komponenten, det vill säga nyttan med komponentens funktionalitet. Inte ens den bästa komponenten kan komma till användning om den inte har den funktionalitet som efterfrågas (Karlsson, 1994, s. 258). Användbarhet avser hur väl paketerad komponenten är, det vill säga hur enkel den är att använda (a.a., s. 122).



Figur 2: Karlssons återanvändningsmodell (Karlsson, 1995, s. 134)

Even-André Karlsson visar vad en komponents återanvändbarhet består av genom återanvändningsmodellen (1995, s. 134). Denna modell går ut på att tydliggöra de faktorer som direkt och indirekt påverkar komponentens återanvändbarhet. Återanvändningsmodellen utarbetades utifrån en studie som gjordes i fem europeiska länder, där systemutvecklare fick ge sin syn på vilka faktorer de anser viktiga för en återanvändbar komponent. Modellen är konstruerad för att avgöra vilka faktorer som bör prioriteras för en komponent under utvecklingsfasen (a.a., s. 133). Här används modellen emellertid för att på ett tydligt sätt åskådliggöra komponentens viktigaste interna egenskaper.

De olika faktorer som påverkar återanvändbarheten indelas i fyra kategorier, som kortfattat beskrivs nedan: *Portabilitet* syftar till hur lätt det är att flytta komponenten till en ny miljö. Ju mer portabel komponenten är, desto enklare är den att anpassa till nya kontexter. För att det ska vara lätt att implementera komponenten i nya applikationer som har olika funktionalitetskrav, krävs en hög *anpassningsbarhet*. *Förståelighet* för en komponent är viktig för att kunna avgöra när den ska användas och om den uppfyller de krav som ställs på komponenten och *tillförlitlighet* avser hur riskfri komponenten är att implementera (Karlsson, 1995, s. 134).

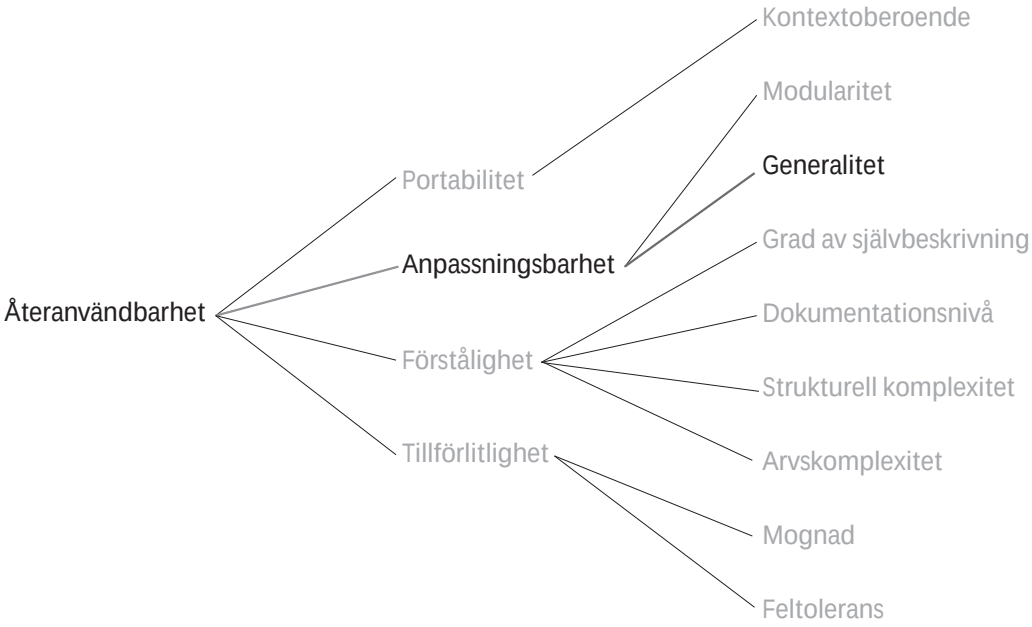
Generella komponenter

Vad skiljer då en återanvändbar komponent från en icke återanvändbar sådan? Ser man till komponenters återanvändbarhet, är den mest centrala funktionella faktorn generalitet; är inte en komponent generell, är funktionaliteten begränsad från att användas i andra sammanhang.

> One of the most important requirements of a reusable asset is generality: the more general the asset, the wider its range of applications, hence its reuse-potential. (Mili m.fl., 2002, s. 28)

De andra faktorerna är till största delen inriktade på användbarhet och gör det mer eller mindre enkelt att återanvända komponenten, men de har mindre inverkan på komponentens grundfunktionalitet. Generalitet definieras i det här fallet som komponentens grad av allmängiltighet samt dess självständighet gentemot den övriga programvaran. Karlsson beskriver generalitet som: ”Criteria that assess a component’s ability to expand the usefulness of a given function beyond the existing module or program and its present scope” (1995, s. 488).

Själva begreppet återanvändbarhet medför att komponenten måste ha en viss grad av generalitet – så fort komponenten kan användas i fler än ett fall är den generell till en viss grad (Karlsson, 1995, s. 255). ”Reusability is useful generality. (a.a., s. 257)”. Frågan under planeringsfasen för komponenter handlar snarare om till vilken grad komponenten ska vara generell.



Figur 3: Karlssons återanvändningsmodell där kopplingen mellan återanvändbarhet och generalitet tydliggörs (Karlsson, 1995, s. 137)

För att använda komponenter i olika applikationer kan man antingen utveckla många små, mer specifika komponenter, eller färre, generella komponenter som kan användas i flera olika applikationer och kontexter (Jacobson, 1997, s. 95f). En komponent bör vara så generell att den kan användas utan att kräva mycket anpassning till de tänkta användningsfallen (Karlsson, 1995, s. 275). Komponenten får dock inte bli så generell att den inte längre är specifik nog för något fall och därför inte används.

Man försöker utveckla komponenter med så stor generalitet som möjligt, så länge utvecklingen ligger inom ramen för organisationens verksamhet och mål. Design av komponenters generalitet, vilken bygger på den analys som komponentutveckling föregås av, kan därför ha stor inverkan på komponentens återanvändbarhet och är en viktig del av utvecklingen. Utveckling av komponenter handlar därför mycket om att identifiera framtida behov och förutse eventuell gemensam funktionalitet, för att sedan konstruera komponenter som möter en rimlig grad av generalitet.

Storleken på komponenten, det vill säga dess funktionella innehåll, påverkar till stor del komponentens återanvändbarhet. Ju mer komplex komponenten är desto svårare är den att använda och förstå (Szyperski, 2002, s. 45). Stora och generella komponenter innebär också ofta problem vad gäller att hitta användningsområden som passar deras funktionalitet, medan komponenter som är mindre och som har en lägre grad av komplexitet är betydligt lättare att finna användningsfall åt (Karlsson, 1995, s. 381). Samtidigt ger mer komplexa komponenter större funk-

tionell nytta vid en eventuell implementering än mindre komponenter (a.a., s. 381):

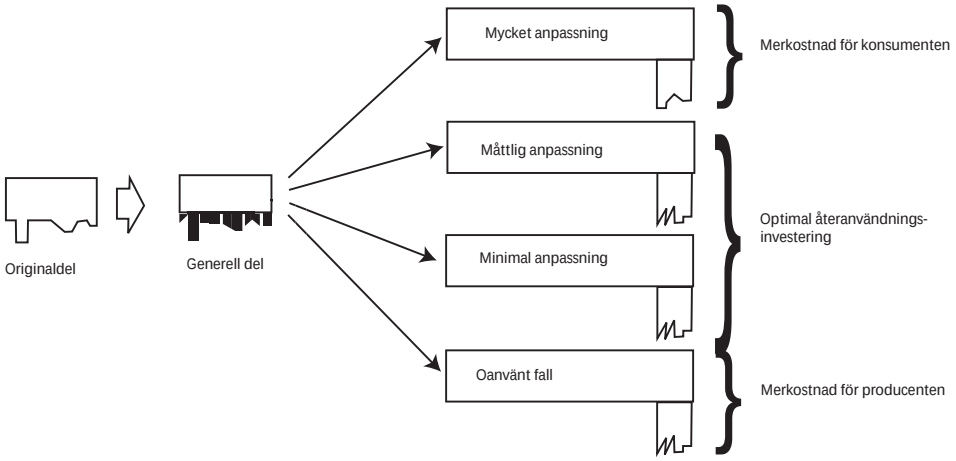
> […] modules that solve difficult or complex problems […] are excellent reuse candidates because they incorporate a high level of problem-solving expertise that is very expensive to replicate. (Barnes & Bollinger, 1991, s. 14)

Överensstämmelsen mellan den domän som en komponent blivit utvecklad för och den domän den ska användas inom inverkar stort på möjligheten till återanvändning. Om domänerna är olika varandra, kommer endast de komponenter som är små och mindre komplexa att kunna återanvändas framgångsrikt (Karlsson, 1995, s. 381).

Det är ofta svårt att utvärdera en generell komponents prestanda, eftersom denna beror på hur komponenten är implementerad i det specifika fallet. I en annan kontext kan komponenten bete sig helt annorlunda (Sitaraman, 2001, s. 3). Generella komponenter blir lätt klumpiga i jämförelse med applikationsspecifika komponenter, vilket kan leda till prestandaproblem.

Komponenter bör inte göras varken mer eller mindre generella än att de innefattar precis sitt tänkta användningsområde. Barnes & Bollinger (1994, s. 20) menar att kostnaderna distribueras mellan olika grupper beroende på om komponenternas allmängiltighet antingen är för låg eller hög:

> Just as a reuse technology that is not general enough can be unduly expensive for reuse consumers, a technology that is either overextended or more general than necessary can result in excessive costs to the reuse producer. (Barnes & Bollinger, 1994, s. 20)



Figur 4: Kostnadsmål när man utvecklar återanvändbara komponenter (Barnes & Bollinger, 1994, s. 19)

Figur 4 visar hur generalitet bör matcha det tänkta behovet av återanvändning. *Merkostnad för konsument* innebär att komponenten är så svår att anpassa att kostnaden överstiger vinsten. I de här fallen beror den uteblivna vinsten på att komponenten inte är generell nog och därför behöver mycket anpassning. I de fall då det krävs en måttlig alternativt minimal anpassning för att använda komponenten, innebär det att generalitetsgraden fungerar väl. Då komponenten inte används innebär framställandet av komponenten naturligtvis en onödig kostnad (Karlsson, 1995, s. 258). Av alla utvecklade komponenter är det en stor del som inte utnyttjas och man bör räkna med ett visst bortfall, vilket inte alltid är av ondo, menar Barnes & Bollinger:

> Many low-cost unused instances are generally acceptable and even desirable as insurance, but many unused instances created with expensive technologies could easily result in reuse-investment losses (Barnes & Bollinger, 1994, s. 20)

Komponenters kontextberoende

När en komponent används mycket betyder det också att dess kontextberoende växer. Detta utgör ett problem när komponenten ska vidareutvecklas, eftersom man då måste ta hänsyn till de olika kontexter komponenten används i. Varje kontextberoende minskar möjligheten att hitta nya sammanhang att använda komponenten i: ”Maximizing reuse minimizes use” (Szyperski m.fl., 2002, s. 45). Det gäller alltså att hitta rätt balans för varje komponent. Samtidigt beror det mycket på vilket slags kontext det handlar om; om samma kontext råder överallt är det riskfritt att göra komponenten beroende av den och tvärtom (Szyperski m.fl., 2002, s. 46).

Det är inte bara utvecklingen av programkoden som tar längre tid, utan på grund av att komponenten konstrueras för att användas i flera olika kontexter måste också bland annat dokumentation och tester göras mer allmängiltiga, vilket tar längre tid (Szyperski m.fl., 2002, s. 17; Mili m.fl., 2002, s. 508): ”Solving a general problem rather than a specific one takes more work” (Szyperski m.fl., 2002, s. 17).

Planering för utveckling av generella komponenter

Utveckling av komponenter med hög grad av generalitet tenderar att kräva stora utvecklingsresurser (Barnes & Bollinger, 1994, s. 18). De höga investeringskostnaderna för att bygga generella komponenter gör att det är aktuellt endast då man förväntar sig att komponenten kommer att användas vid upprepade tillfällen:

> Construction of a generalized component is only reasonable if it is expected that the component will find further applications. (Szyperski, 2002, s. 139)

Det är svårt att planera utformningen av en komponent, eftersom nyttan av denna inte visar sig förrän långt senare (D'Alessandro m.fl., 1993, s. 1). Därför görs ofta någon form av analys av gemensamma användningsområden för komponenter (Barnes & Bollinger, 1994, s. 17). Problemen stöter man inte på förrän i användningsfasen och den här diskrepansen kräver ett stort mått av planering för att överbryggas. Med tanke på att utvecklingen går snabbt inom programvarubranschen kan det vara svårt att sia om även den närmaste framtiden, vilket ger problem när man ska planera komponenterna.

Utvecklingen av generella komponenter påverkas av en rad olika faktorer, som utvecklingskontext, typ av verksamhet och utvecklarnas kompetens (Nazareth m.fl., 2004, s. 251).

För att kunna ta reda på huruvida komponenthanteringen verkligen är lönsam måste utvärderingar göras. Enligt Karlsson kan sådana utvärderingar göras både på komponent-, projekt- och organisationsnivå (1995, s. 118). Man måste väga den extra insatsen att modifiera och analysera en befintlig komponent mot att helt enkelt utveckla en ny från början (Mili m.fl., 2002, s. 18).

Domänanalys

En *domän* är ett kunskaps- eller aktivitetsområde som karaktäriseras av att informationssystemen inom detta är relaterade till varandra på olika sätt (Mili m.fl., 2002, s. 125). En domän har antingen gemensam expertis, gemensam design eller gemensam marknad – eller flera av dessa tre. En optimal domän uppfyller alla tre kriterierna (a.a., s. 125).

> The experience of the past two decades indicates that software reuse techniques are most successful when applied in the context of restricted application domains. (Arango, 1994, s. 42)

Domänanalys innebär att de applikationer som ingår – eller ska ingå – i en domän analyseras för att man ska hitta gemensamma karaktäristika eller vilka variationer det finns och varför (Mili m.fl., 2002, s. 126). Domänanalysen är "[…] a process by which we are able to exploit similarities in applications (products or systems) in the domain, capture experiences, and identify variabilities" (a.a., 2002, s. 126).

Målet med domänanalys är att identifiera arkitekturer och komponenter som kan användas för att implementera de applikationer som kommer att ingå i domänen (Arango, 1994, s. 43; Karlsson, 1995, s. 291). Domänanalys handlar inte enbart om att analysera olika domäner för att hitta gemensam funktionalitet och utnyttja denna, utan även om att identifiera vilka domäner som det överhuvudtaget är ekonomiskt försvarbart att komponentisera (Mili m.fl., 2002, s. 13, Arango, 1994, s. 42). Den gemensamma funktionaliteten som identifieras som potentiellt

vinstgivande att vidareutveckla till generella komponenter kallas ibland *verksamhetsobjekt* (business objects) (Persson, 2002, s. 113).

Under domänanalysen studeras de existerande applikationerna inom domänen, underliggande teorier, kundernas krav och tänkbara framtida krav, samt framtida tekniker inom domänen (Karlsson, 1995, s. 345). Vissa delar återfinns inte i en specifik domän, utan är gemensamma för applikationer inom flera domäner. Dessa kallas lösningsbaserade komponenter (solution-based components) (a.a., s. 294) eller *common business objects* (Persson, 2002, s. 113). Exempel är behållarobjekt och olika konverteringskomponenter.

Under domänanalysen undersöks relationen mellan investeringar och vinster med komponentutvecklingen (Arango, 1994, s. 42) och man bestämmer vilka komponenter som ska utvecklas och inom vilka domäner dessa komponenter ska verka. Komponenterna ska vara så applicerbara som möjligt för alla applikationer inom sin domän (Mili m.fl., 2002, s. 140).

Det finns två vanliga sätt att utföra domänanalysen på – det induktiva och det deduktiva sättet (Mili m.fl., 2002, s. 140). Det induktiva sättet ser till tidigare applikationer inom domänen för att kraven som ställs på komponenterna ska kunna kartläggas. Det deduktiva sättet går snarare ut på att se vilka delar av applikationen som kan göras variabla för att sedan kunna göra en kostnadsanalys för detta så att utvecklingsresurserna används på rätt sätt (a.a., s. 140).

Tekniker för att skapa generella komponenter

När det är bestämt vilka behov som är de mest ekonomiskt lönsamma att bygga in i en viss komponent, måste det bestämmas hur generaliteten ska representeras hos komponenten (Karlsson, 1995, s. 259). Karlsson menar att det finns flera metoder för att åstadkomma generalitet i en komponent (a.a., s. 259-261):

- *Breddning* (widening) innebär att man identifierar de krav som inte är oförenliga och sedan konstrueras en generell komponent innehållande dessa. Fördelen är att komponenten kan användas precis som den är då den passar för de behov den är skapad att hantera, medan nackdelarna är hög utvecklingskostnad då all funktionalitet byggs in från början, samt att komponenten kan bli onödigt komplex och alltså svårare att förstå vid återanvändning.
- Med *avsmalning* (narrowing) menas att man identifierar faktorer som är gemensamma för flera användningsområden och sedan representerar dessa med en abstrakt komponent. På så sätt specialiseras komponenten för ett speciellt användningsområde. Denna typ av återanvändning är vanlig inom objektorienterad programmering där den realiseras med hjälp av arv mellan klasser.

- *Isolering* (isolation) innebär att man avskiljer vissa krav för en mindre del av ett system och utvecklar en komponent utifrån denna. Resterande funktionalitet byggs oberoende av denna komponent.
- *Konfigurerbarhet* (configurability) är en metod för återanvändning där man använder sig av mindre komponenter och på så sätt skapar sig en mängd komponenter som sedan sätts ihop för att bilda själva applikationen: ”This can be compared to the popular Lego toys, which provide everything from generalpurpose kits to very specialized ones” (Karlsson, 1995, s. 260).
- Man kan också använda *generatorer* (generators) som genererar källkod utifrån ett antal val. Detta tillvägagångssätt kräver en komplex komponent alternativt stabil samverkan mellan flera komponenter och kan således kräva stora investeringar.

Vilken metod som man bestämmer sig för bör naturligtvis baseras på det problem man står inför. Det är viktigt att känna till olika generalitetsaspekter för att kunna skapa sig en uppfattning om funktionaliteten kan modifieras för en specifik komponent, och i så fall hur. Metoderna ovan utesluter inte varandra, utan kan användas tillsammans för att representera samma generalitet (Karlsson, 1995, s. 261).

Sammanfattning

Programvarukomponenter är återanvändbara delar av programvara som konstruerats med syfte att återanvändas. Generalitet är komponentens grad av allmängiltighet samt dess självständighet gentemot den övriga programvaran. Generella komponenter bör ha så hög återanvändbarhet som möjligt, vilken är en kombination av komponentens lämplighet samt dess användbarhet.

De främsta fördelarna som anses vara sammankopplade med komponentbaserad systemutveckling i jämförelse med traditionell sådan är ökad produktivitet, kortare time-to-market samt bättre kvalitet i programvaran. Den ökade produktiviteten kommer från att utvecklingstiden blir kortare när komponenter används. Att utveckla generella komponenter ställer högre krav på planering och konstruktion än vad applikationsspecifik programkod gör. Dessutom kostar sökning, integrering och testning av generella komponenter mer. De vinster man kan nå genom komponentorienterad systemutveckling märks först i ett längre perspektiv och i det kortare perspektivet får man istället en högre utvecklingskostnad. En komponent som inte är generell nog kräver utvecklingsresurser för att anpassas till sitt användningsområde.

Alla generella komponenter har både konstant och variabel funktionalitet. Den konstanta delen bör vara så stor som möjligt, eftersom den är gemensam för alla applikationer som använder komponenten. Den variabla delen fungerar som

gränssnitt mellan de olika applikationerna komponenten används i och den konstanta delen.

Kontextberoendet för en komponent växer med varje applikation komponenten används i. Detta utgör ett problem när komponenten ska vidareutvecklas, eftersom man då måste ta hänsyn till de olika kontexter komponenten verkar inom.

Utveckling av komponenter handlar mycket om att identifiera framtida behov och förutse eventuell gemensam funktionalitet för att sedan konstruera komponenter som har rätt grad av generalitet. Analysfasen kallas domänanalys och den går ut på att identifiera komponenter för användning i de applikationer som ingår i domänen.

Komplexa komponenter är svårare att använda och hitta användningsområden för, eftersom de kostar mer att framställa och blir ofta svårare att förstå, men de gör också större nytta när de väl används. Man måste alltid väga den extra kostnad det innebär att modifiera och analysera en befintlig komponent mot kostnaden att utveckla en helt ny. Komponenter som görs väldigt generella för att täcka in många scenarier men som aldrig kommer till användning innebär påfrestande kostnader för verksamheten.

Metod

Kapitlet redovisar undersökningens strategi och metod samt motiverar valet av dessa. Tillvägagångssättet för undersökningen presenteras och slutligen kritiseras undersökningen.

Undersökningsfrågor

Utifrån respektive frågeställning skapades mer specifika undersökningsfrågor för att vi lättare skulle kunna besvara frågeställningarna. Studien utformades för att kunna besvara undersökningsfrågorna. Undersökningsfrågorna definierades innan undersökningen började utifrån litteratur, samtal och provintervju, men omdefinierades och uppdaterades beroende på vad som framkom under studiens gång.

För att illustrera kopplingen upprepas frågeställningarna och de tillhörande undersökningsfrågorna presenteras och motiveras:

Frågeställning	Undersökningsfråga	Motivering
Hur planerar organisationer sin komponentutveckling för att skapa komponenter generella nog för återanvändning?	Finns det strategier för planeringen av komponenters generalitet och hur ser dessa i så fall ut?	*En viktig del för att undersöka hur planeringen ser ut är att undersöka vilka strategier som används.*
	I vilka fall utvecklas generella komponenter?	*Undersöker de faktorer under planeringen som avgör när en komponent ska vara generell eller när den ska vara specifik.*
	Hur bestäms funktionalitetskraven på generella komponenter utifrån framtida behov?	*För att kunna utveckla generella komponenter måste man först ta reda på vilka krav på funktionalitet som kommer att ställas på dessa.*
Vilka problem kan uppstå beroende på planeringen av komponenters generalitetsgrad och hur hanteras dessa?	Vilka problem kan uppstå och hur löser man dessa när komponenter är antingen för specifika eller för generella?	*Denna undersökningsfråga ämnar ge svar till hur man hanterar de problem som uppstår när man utvecklar komponenter med antingen för hög eller låg grad av generalitet.*
	Vilka faktorer avgör om man utvecklar en helt ny komponent eller om man modifierar en redan befintlig sådan?	*Undersökningsfrågan ska besvara hur planeringen ser ut när man har stött på problem med generaliteten och behöver åtgärda detta.*

Tabell 1: Undersökningsfrågorna och hur dessa motiveras utifrån frågeställningarna

Strategi

Vi ansåg att den mest lämpliga informationen kom från personer som dagligen jobbar med komponentutveckling och planering av denna, eftersom undersökningens syfte behandlar hur planering av generalitet praktiseras. Undersökningsfrågorna krävde detaljerade data för att kunna besvaras och därför var det inte rimligt att undersöka ett större antal personer, utan studien inriktades på ett fåtal experter som förväntades känna till undersökningsområdet väl och ha stor erfarenhet av detta. En viktig del av undersökningen var att diskutera undersökningspersonernas svar och resonemang utifrån de kontexter de verkar inom, som

till exempel vilken organisation de arbetar för och vilken position de innehar, eftersom analysen utfördes utifrån undersökningspersonernas förutsättningar.

Eftersom undersökningen delvis kan anses vara explorativ, då vi inte kände till vilka svar som kunde tänkas framkomma ur undersökningen, var strategin att inte utföra denna utifrån någon speciellt hypotes eller teori. Det flexibla förhållningssättet erbjöd möjligheter att hantera oväntade skeenden och infall under undersökningen och inriktningen för studien justerades till viss del beroende på de data den gav.

Metodval

För att vi skulle få den typ av kvalitativ information som var syftet med datainsamlingen utfördes intervjuer utformade som personliga samtal; kvalitativa forskningsintervjuer. Även observationer hade kunnat ge liknande data, men undersökningsfrågorna är inte endast kartläggande, utan syftar också till att ta reda på *varför* situationen ser ut som den gör. Intervjuer gör det möjligt att samla in undersökningsmaterial på ett relativt ostrukturerat sätt, vilket gav den önskade flexibiliteten gentemot undersökningsområdet. Ett öppet samtal gör det lättare att säkerställa att respondenten förstår frågorna rätt, vilket kan sägas stärker validiteten för undersökningen.

Genom att använda kvalitativa intervjuer finns det utrymme för att på ett lättbegripligt sätt presentera syftet med undersökningen och även förklara på vilket sätt intervjupersonens kunskaper är intressanta. En sådan beskrivning gör att intervjupersonerna enkelt kan sätta sig in i problemformuleringarna och förhoppningsvis verkar detta som ett incitament för intervjupersonen att svara mer ingående och heltäckande på intervjufrågorna.

Kvalitativa intervjuer ger möjlighet att följa upp intressanta resonemang och ställa följdfrågor – även inom områden som inte ingår i undersökningen. I kvalitativa intervjuer är det just i mötet mellan intervjupersonen och intervjuaren och deras respektive synpunkter som kunskapen uppstår (Kvale, 1997, s. 9, 117). Förhoppningen var att den här öppenheten skulle innebära att intervjupersonerna diskuterade sig fram till de områden som var speciellt intressanta för dem själva och deras organisation.

Den kvalitativa forskningsintervjun handlar inte bara om att ställa rätt frågor, utan även om att kunna lyssna lyhört (Kvale, 1997, s. 13). En stor anledning till valet av intervju som undersökningsmetod var möjligheten att förstå undermeningar och utifrån dessa ställa intressanta följdfrågor.

Intervju

Trots att vi försökte närma oss ämnet utan för mycket teorier var en viss teoretisk bakgrund till undersökningsområdet nödvändig för att vi skulle förstå intervjupersonernas språk samt kunna ställa bra och relevanta frågor som kunde öppna för diskussion. Vi försökte dock inte bli hämmade av våra förkunskaper utan försökte bibehålla ett öppet synsätt.

Under de olika intervjuerna kunde det ibland vara svårt för intervjupersonen att förstå våra frågor korrekt. Vi märkte dock att det var lätt att förklara frågorna ytterligare eller närma oss undersökningsområdet på ett alternativt sätt. Naturligtvis kan det finnas en viss risk med att som undersökningsledare själv vara involverad i undersökningen och möta intervjupersonerna halvvägs, eftersom man genom medverkandet kan påverka resultatet. Man måste vara medveten om att svaren inte enbart beror på frågorna, utan även på vem som ställer dem och på vilket sätt detta görs (Bryman, 1997, s. 94-95). Vi ansåg dock att fördelarna med att kunna förklara, fördjupa och vidareutveckla vägde upp den nackdelen.

Intervjuguide

Intervjuguiden utarbetades för att säkerställa att undersökningsfrågorna skulle diskuteras under intervjuerna. Intervjuguiden uppkom ur en analys över hur undersökningsfrågorna bäst besvaras och den fungerade som en guide till vilka områden som skulle tas upp till diskussion och, till viss mån, i vilken ordning. Vissa områden visade sig vara mer intressanta under vissa intervjuer och då ställdes fler frågor inom det området. Även om intervjuguiden var densamma för samtliga intervjuer, modifierades frågorna på plats för att bättre passa den aktuella intervjupersonen.

Pilotintervju

En pilotintervju genomfördes tidigt i undersökningen för att kontrollera om undersökningsformen kunde generera den efterfrågade typen av information och för att kontrollera frågornas relevans. Intervjupersonen arbetade i ett litet konsultnätverk som utvecklar en smal linje av programvaror och komponenter till dessa. Intervjun följdes av en utvärdering med intervjupersonen, där vi gemensamt diskuterade både innehållet i intervjun och hur intervjuguiden samt utförandet skulle kunna förbättras.

Pilotintervjun användes inte ytterligare i undersökningen.

Urval

Endast kunniga personer som arbetar med komponentutveckling var intressanta för undersökningen, eftersom dessa antogs ha viktiga kunskaper inom området.

För att undersökningen skulle vara enkel att genomföra begränsades företagen till Skåneregionen. Intervjupersonerna kom inte endast från företag, utan perspektivet breddades även med en forskare inom området.

För att bredda bilden ytterligare valdes företag av olika storlekar och med olika inriktningar. Målsättningen var att undersöka något medelstort företag och något stort företag – småföretag antogs ha små möjligheter att investera i generella komponenter. Det var viktigt att också undersöka ett konsultföretag för att se hur utveckling för flera kunder påverkar komponentplaneringen. I kontrast till detta hade vi för avsikt att undersöka något företag som är inriktat på att utveckla en viss produktlinje.

Intervjuer genomfördes tills vi hade tillräckligt med material för att kunna göra en meningsfull analys, samt tills alla undersökningsgrupper var representerade.

Intervjupersoner

Mattias Bryborn på Anoto intervjuades eftersom Anoto är ett mellanstort IT-företag, vars utveckling mest är inriktad på specifika produkter. Mattias är utvecklingsingenjör för programvara och han var väl insatt i Anotos komponentutveckling.

Sogeti i Helsingborg passade studien eftersom de är ett konsultföretag. Intervjupersonen Patrik Eriksson arbetar som systemarkitekt och jobbar mycket med planering av komponentutveckling.

Jon Jarnsäter, systemutvecklare på IKEA IT i Helsingborg, och Mattias Wallinius, programvarumentor på Tetra Pak R & D, intervjuades som representanter för de större företagen i undersökningen.

För att intervjua någon forskare kontaktades Even-Andé Karlsson, som jobbar i Lund. Det visade sig att han var upptagen men han rekommenderade istället Josef Nedstam, doktorand inom komponenthantering vid institutionen för telekommunikation vid Lunds tekniska högskola.

Utförande

Intervjuerna utfördes på en plats som intervjupersonen valde, vilket i samtliga fall innebar dennes arbetsplats. Vår förhoppning var att detta skulle medföra att intervjupersonen kände sig trygg under intervjun. Det var viktigt att bygga upp en atmosfär i vilken intervjupersonerna kände sig bekväma, så att de upplevde att de kunde svara fritt på de frågor som ställdes.

Intervjuerna varade från ungefär 45 minuter till en timme och spelades in i sin helhet. Intervjupersonerna informerades, enligt Kvales rekommendationer, om hur materialet skulle hanteras och vad det skulle användas till, samt vad dennes roll innebar för undersökningen (1997, s. 142).

Intervjuerna var löst strukturerade, vilket innebar att det fanns stort utrymme för intervjupersonen att själv ta upp områden som denne tyckte var intressanta. Ibland leddes intervjuerna in på olika stickspår utanför ämnesområdet, vilket bland annat medförde att konversationsklimatet tinades upp. Den lösa intervjuformen bäddade för fördjupningar inom de ämnen vi ansåg mest intressanta att ta upp med den aktuella intervjupersonen och det var ofta lätt att komma på passande följdfrågor. Det fanns också tillfällen där vi upptäckte att ett visst frågeområde saknade relevans i det aktuella fallet och kunde på så sätt undvikas, vilket Bryman nämner som en fördel med intervjuer och deras flexibilitet (1997, s. 84). Därmed kunde undersökningen fokuseras på de områden som verkligen var givande.

Sammanfattning av intervjusvar

Intervjuns relevanta delar sammanställdes skriftligen och ambitionen var att i högsta möjliga grad spegla den allmänna tonen för intervjun så väl som möjligt. Sammanfattningen skickades sedan till intervjupersonen för granskning och missförstånd kunde på så sätt klaras upp och annat kunde nyanseras eller fördjupas.

Sammanfattningarna bör betraktas som tolkning av rådata om vad som ska finnas med samt i vilken utsträckning och ambitionen var att allt viktigt från intervjuerna skulle finns med i sammanfattningarna. Det är lätt att glömma bort att sammanfattningarna bygger på tolkning och betraktas därför som en stadig grund för vidare analys (Kvale, 1997, s. 154).

Alla intervjusammanfattningar finns med som bilaga 1-5.

Analysförfarande

En analysmodell skapades med syfte att utgöra en grund för vidare tolkning av resultaten inom de olika undersökningsområdena. Modellen visar vilken teoretisk bakgrund som resultaten analyserades utifrån och vilken typ av slutsatser som drogs.

Med hjälp av analysmodellen kodades intervjusammanfattningarna för att bättre kunna överblicka intervjumaterialet. Ibland ledde svaren på en undersökningsfråga samtidigt till diskussioner som hörde hemma i andra delar av analysmodellen. Under kodningen hände det att intressanta resonemang dök upp som inte passade de kategorier som var uppställda och i dessa fall modifierades analysmodellen för att inrymma den nya informationen.

Allt som förekommer i analyskapitlet är att betrakta som analys. Eftersom intervjupersonerna är experter inom undersökningsområdet är deras uttalanden viktiga delar av analysen, även om dessa inte diskuteras utförligt.

Analysmodell

Analysen skedde mot bakgrund av vilken organisation intervjupersonen arbetade inom och vilken position intervjupersonen hade. Analysmodellen är uppbyggd utifrån undersökningsfrågorna och visar utifrån vilken typ av teoribakgrund som dessa diskuteras för att nå fram till slutsatserna.

Finns det strategier för planeringen av komponenters generalitet och hur ser dessa i så fall ut?

Olika strategier och tillhörande motiveringar diskuterades mot bakgrund av olika ekonomiska för- och nackdelar som komponentbaserad systemutveckling kan innebära. Analysen behandlade också svårigheterna med att göra bra domänanalyser för utvecklingen av generella komponenter och detta sattes i perspektiv till olika typer av verksamheter. Analysen diskuterar hur en organisations typ av verksamhet reflekterar dess komponentstrategi.

I vilka fall utvecklas generella komponenter?

Frågan analyserades utifrån när det är lönsamt att utveckla generella komponenter och på vilka grunder beslutet tas. Resonemanget diskuterades till viss del utifrån figur 1 (s. 8), det vill säga relationen mellan återanvändning och investeringar. Diskussionen behandlade frågan om när det är önskvärt att utveckla generella komponenter och hur analysen av detta utförs. Frågan diskuterades också utifrån teoretiska resonemang kring domänanalys och hur domänanalysen påverkas beroende på hur många domäner man har att analysera och hur spridda dessa är. Frågan diskuterades även utifrån hur kontextberoendet påverkas av en komponents generalitet.

Hur bestäms funktionalitetskraven på generella komponenter utifrån framtida behov?

De olika metoder att bestämma funktionalitetskraven och framtida behov som framkom under undersökningen diskuterades och ställdes mot varandra. Frågan analyserades med hjälp av teorier om domänanalyser och hur dessa används när funktionalitetskraven fastställs.

Vilka problem kan uppstå och hur löser man dessa när komponenter är antingen för specifika eller för generella?

Frågan tar upp hur olika kvalitetsaspekter hos komponenten påverkas då den antingen är för specifik eller för generell. Diskussioner utifrån figur 4 (s. 14) fördes också angående olika kostnadsmål och vem som drabbas ekonomiskt vid utveckling av generella komponenter. Frågan mynnade ut i en analys om inställningen till generella komponenter och hur stort problem man anser att komponenter som är för specifika eller för generella utgör.

Vilka faktorer avgör om man utvecklar en helt ny komponent eller om man modifierar en redan befintlig sådan?

Området diskuterades utifrån vilka omständigheter som man tar hänsyn till i valet om att antingen utveckla en ny komponent eller modifiera en som redan används. Frågan analyserades delvis utifrån komponenters kvalitetsaspekter.

Slutsatser

Slutsatsernas syfte är att kort och koncist besvara uppsatsens övergripande frågeställningar och de framkom genom att de viktigaste punkterna från analysen extraherades. I de fall då flera analyspunkter berörde samma område sammanställdes dessa till en slutsats. På grund av detta följer inte slutsatserna, till skillnad från analysen, undersökningsfrågorna, utan de är istället indelade enligt respektive frågeställning.

Metodkritik

Vi upptäckte att det var svårt att hitta rätt person inom de valda organisationerna. Eftersom vi hade liten kunskap om hur företagen var organiserade var det svårt att veta vilken position och person inom företaget som var mest lämplig att undersöka och vi fick förlita oss till att vi blev hänvisade till rätt person. Efter intervjun visste vi betydligt mer om verksamheten och ofta hörde vi om andra personer som också hade varit intressanta att intervjua.

Beroende på vilken organisation intervjupersonen jobbade och vilken position denne hade kunde intervjupersonerna bidra med olika fokus i undersökningen. Effekten blev att tyngdpunkten för de olika intervjuerna hamnade på olika undersökningsfrågor. Under analysfasen saknades synpunkter på vissa frågor och andra hade ett lite annorlunda fokus, vilket gjorde att analysen försvårades.

Man kan inte bortse från det faktum att forskningssituationen påverkade intervjupersonerna, vilket gör det svårt att jämföra resultaten. Genom att ha en intervjuguide som garanterade att vi kom in på samma frågeområden försökte vi motverka denna trend, samtidigt som just skillnaden mellan de olika intervjupersonerna och kontexterna runt dessa var en styrka med studien.

Ett annat problem var att intervjupersonernas definitioner av vad en komponent är skiljde sig från varandra. Här visade sig intervjuformen ha en stor fördel i att vi kunde diskutera ingående med intervjupersonerna och kommunicera vad vi avsåg med en komponent och på så sätt komma fram till ett gemensamt synsätt.

Resultat och analys

I det här kapitlet presenteras de resultat som framkom från undersökningen samt analysen av dessa. Kapitlet inleds med en kort sammanställning av intervjupersonerna och sedan följer resultat och analys indelade efter frågeställning och undersökningsfråga.

Intervjupersoner

Intervjupersonerna presenteras kortfattat som referens vid läsning av resultat och analys.

- Mattias Bryborn (M.B.) – utvecklingsingenjör, Anoto AB
- Patrik Eriksson (P.E.) – systemarkitekt, Sogeti
- Jon Jarnsäter (J.J.) – systemutvecklare, IKEA IT
- Mattias Wallinius (M.W) – utvecklingsmentor, Tetra Pak R & D
- Josef Nedstam (J.N.) – doktorand, Lunds tekniska högskola

Sammanställningar av alla intervjuer finns med som bilagor 1-5.

Planering av generella komponenter

Frågeställningen som behandlas är: *Hur planerar organisationer sin komponentutveckling för att skapa komponenter generella nog för återanvändning?*

Finns det strategier för planeringen av komponenters generalitet och hur ser dessa i så fall ut?

I alla organisationer som undersökts planerar man generaliteten hos komponenterna. Detta görs dock på olika sätt och på olika nivåer inom organisationerna. I vissa fall sker det intuitivt av systemutvecklarna själva och i andra fall mer formellt och högre upp i organisationen.

Sogeti har ingen uttalad strategi för utvecklingen av generella komponenter. Man har, enligt P.E., ett lokalt programkodsbibliotek som mestadels består av källkod i form av klasser och alltså inte färdiga komponenter. Innehållet i biblioteket har blivit allt mer likt komponenter med tiden. På Sogeti används annars e-postlistor för att kommunicera med andra kontor inom organisationen om vad som är möjligt att utnyttja från tidigare utvecklade applikationer. Det finns alltså inga gemensamma bibliotek av programkod som kan användas, utan man får ta kontakt med personer som är ansvariga för ett visst kompetensområde och ställa frågor om vad som tidigare blivit utvecklat inom organisationen och hur man bäst kan dra nytta av detta. P.E. menar att det är en nödvändighet att organisationer får till stånd en viss interaktivitet – som Sogetis e-postlistor – för att återanvändningen ska fungera tillfredsställande.

En strategisk begränsning som P.E. måste ta hänsyn till under utveckling av komponenters generalitet är att utvecklingen är knuten till en specifik plattform, i hans fall Microsoft. Detta leder till att P.E. ofta använder Microsofts designförslag.

P.E. menar att trenden rör sig bort från att ha många generella komponenter som är svåra att använda – dels på grund av att kraven är luddiga och dels för att dessa modifieras kontinuerligt, men ändå aldrig passar riktigt bra när de används. Han menar vidare att det ofta är alltför kostsamt att underhålla stora komponentbibliotek, och känner till flera företag som har misslyckats med just detta. Vid komponentutveckling på Sogeti utvecklar man initialt en komponent och då kraven sedan förändras bygger man vidare på samma komponent. På så sätt försöker de utveckla komponenter som ska vara enkla att uppdatera: ”Det är en mer lättrörlig process och det viktigaste är att komponenten är lätt att förändra.” Istället för att utveckla generella komponenter från grunden med fokus på återanvändbarhet, är strategin alltså att utveckla komponenter med fokus på modifierbarhet.

J.N. tar upp ett övergripande problem när det gäller komponentanvändning, vilket är att många systemutvecklare kan ha svårt att trivas med och känna tillförlitlighet till komponenter som andra har utvecklat. J.N. menar vidare att många organisationer inser vilka fördelar det skulle ge organisationen om man kunde återanvända delar av det som tidigare utvecklats, men att det ofta saknas resurser för detta i projekten. J.N. tycker att det verkar som att komponentutvecklingen och organisationen av komponenter blir lidande genom det faktum att företagen inte ser dessa som en del av den primära verksamheten.

Det är svårt att veta hur marknaden ser ut när komponenterna är färdigkonstruerade och produkterna kan levereras. Senarelägger man produktutvecklingen finns det en risk att man missar viktiga möjligheter på marknaden, menar J.N. Det är viktigt att man överväger vilka risker det finns med utvecklingen av generella komponenter, samtidigt som man bör sätta dessa i relation till de vinster man eventuellt kan göra genom framgångsrik komponentutveckling. Dessutom kan man göra direkta vinster genom att distribuera de generella komponenterna vidare till andra företag.

På IKEA IT har man planeringsmöten för programvaruutvecklingen, men det förs inte direkta diskussioner angående komponenters generalitet. Den grundläggande komponentstrategin är att en speciell grupp utvecklar ramverkskomponenter, som sedan används av andra systemutvecklare. Ramverkskomponenterna fungerar som grundbultar för den vidare utvecklingen av applikationer. Om förändringar av dessa komponenter anses vara nödvändiga, föreslås dessa till den avdelning som ansvarar för utvecklingen, vilken sedan skapar nya versioner av komponenterna. J.J. menar att det också finns informell återanvändning inom organisationen genom att man delar lösningar med andra systemutvecklare och genom att distribuera lösningar via komponentbibliotek.

M.W. menar att man på Tetra Pak försöker bygga upp sina applikationer av småskaliga komponenter för att de ska vara lätt utbytbara. Delar av funktionaliteten köps in från andra leverantörer och andra delar utvecklas av Tetra Pak, eftersom det inte alltid går att finna komponenter som tillhandahåller den önskade funktionaliteten hos tredjepartsleverantörer. Genom att använda sig av klassiska designmönster försöker man, enligt M.W., skapa förutsättningar för att utveckling ska kunna ske gentemot olika plattformar. All utveckling sker under Microsofts .NET-plattform, men utvecklingen skulle kunna migrera till en Java-baserad plattform.

Enligt M.W är en del av strategin på Tetra Pak R & D att köpa in komponenter vars funktionalitet är modern, så att komponenterna ska kunna användas länge utan att funktionaliteten blir föråldrad. Utvecklingstiden är begränsad på Tetra Pak och M.W. menar att utvecklingen inte får ta längre tid än sex månader. Utvecklingen sker i små steg för att man snabbt och enkelt ska kunna förändra funktionalitet när så behövs. En viktig del av strategin är att inte betrakta systemen som avslutade bara för att utvecklingsprojektet är det: Varje projekt leder till ett nytt projekt och utvecklingen ses därför som konstant pågående.

Det verkar som att många företag är försiktiga med att binda upp resurser i generella komponenter och organisationen som hanterar dessa. Man väljer istället det säkra före det osäkra och utvecklar förändringsbara komponenter istället för anpassningsbara. Man räknar med att många komponenter behöver uppdateras ofta men tar hellre den kostnaden som är lättare att beräkna än osäkerheten generella komponenter innebär. Några av intervjupersonerna hade både egna erfarenheter

och kände till andra företag som inte har fått tillbaka de investeringar som gjorts i generella komponenter, komponentbibliotek och så vidare. Generella komponenter används till stor del inom de flesta organisationer, men man försöker hålla organisationen av dessa till ett minimum. Företagen vill vara säkra på att de generella komponenterna verkligen kommer att användas inom en överblickbar framtid så att investeringarna säkert betalar sig. Det verkar som om företagen gör bra analyser i de fall generella komponenter används, men det är möjligt att organisationerna är för försiktiga med att utveckla komponenter och därför förlorar i intäkter och konkurrenskraft.

Enligt M.B. görs ingen formell utvärdering av enskilda komponenter på Anoto, förutom i de fall komponenten också är en produkt i sig. Inte heller görs någon formell utvärdering på IKEA IT, menar J.J. På Tetra Pak utgår man från feedback från tidigare produkter när en ny utvecklingscykel startar. Utifrån feedbacken skapas användarfall (use cases), som man utgår från när man fastställer funktionalitetskraven samt generalitetsgraden hos de nya komponenterna. Varje ny utvecklingscykel på Tetra Pak föregås av en genomgång av komponenter från tidigare projekt, där man diskuterar vad som varit problematiskt och korrigerar detta. Dock, erkänner M.W., är inte Tetra Pak speciellt bra på utvärderingar och menar att mängden administration som är förknippat med utvärderingar ofta tar död på systemutvecklarnas kreativitet. Komponentnyttan utvärderas dock i varje projekt och man tittar på hur värdefull komponenten har varit för verksamheten, vilket på så sätt visar vilka komponenter som är värda att använda i framtida projekt.

Det verkar vara en medveten strategi att inte besvära systemutvecklarna med för mycket utvärdering på Tetra Pak och möjligtvis kan detta tänkesätt också råda i många företag, även om det inte uttrycks öppet. Som tidigare nämnts är systemutvecklarnas tid ofta dyrbar för företagen, vilket också kan vara en anledning till att inte ägna för mycket resurser på utvärdering. Dessutom kan det vara svårt att se vilken nytta som utvärderingen bidrar med.

I vilka fall utvecklas generella komponenter?

M.B. menar att man bör utveckla generella komponenter när det finns ett flertal olika produkter som har samma behov av grundfunktionalitet. Däremot kan det utöver grundfunktionaliteten finnas olika krav. På Anoto används inte komponenter i produkter vars funktionalitet är fundamentalt annorlunda från varandra, utan snarare i produkter som baseras på modifikationer i samma kravställning. Samma sak gäller när maskinvaran i de olika produkter komponenten ska användas i skiljer sig alltför mycket från varandra. I dessa fall försöker man på Anoto att abstrahera funktionaliteten och hitta en gemensam plattform med komponenter som täcker alla funktionalitetskrav. På Anoto används till stor del en induktiv domänanalys för att bestämma vilka generella komponenter som ska utvecklas.

Detta ger goda resultat, vilket förmodligen beror på att produkterna som utvecklas av Anoto liknar varandra. Detta gör i sin tur att det är lätt att se vilka delar av systemen som ingår i samma domän, eftersom de har liknande design och marknad, och utifrån detta dra paralleller till nuvarande system.

Tidsperspektiv, alltså hur långt fram i tiden man räknar med att den generella komponenten bör kunna användas, är ”en produkt som fungerar ungefär som den här”, enligt M.B. Tidsperspektivet är enormt viktigt för möjligheten att återanvända komponenter och är ofta en direkt avgörande faktor vid valet att utveckla generella komponenter.

P.E. menar att det ofta utvecklas för många generella komponenter där de egentligen borde vara mer specifika. Han menar att det brister i analysen och man utvecklar funktionalitet som utvecklaren tror kommer att behövas – ”nice to have-grejer”. I slutändan visar det sig dock att det byggs upp stora komponentbibliotek som inte används. M.W. menar att Tetra Pak har utvecklat många generella komponenter tidigare, som varit tänkta att kunna användas för flera olika applikationer inom verksamheten, men som oftast har slutat som misslyckanden – komponenterna har inte har använts i den utsträckning det från början var tänkt. J.N. resonerar på samma sätt och menar att generella komponenter ofta inte används till den grad man avsåg. Problemen beror många gånger på svaga domänanalyser, som snarare inriktas på vilka delar som *kan* göras generella, och inte på vilka som *bör* vara det. Varje utvecklare gör dessutom sina egna analyser under utvecklingsfasen, och lägger ibland resurser på fel område. Detta betyder att komponenterna kan bli undermåliga, även om domänanalysen var bra. Man kan också tänka sig att systemutvecklarnas personliga erfarenheter och kreativitet leder till att nya möjligheter tas tillvara, som inte upptäcktes under domänanalysen.

Kostnaden avgör i de flesta fall om man ska utveckla generella komponenter, menar M.W. Det är dock svårt att granska hur mycket komponenten kommer att användas, eftersom kraven på denna ändras över tiden då förutsättningarna förändras. Man behöver för det första veta den extra kostnaden för själva utvecklandet av den generella komponenten, för det andra hur många gånger den behöver användas för att motivera den högre utvecklingskostnaden, och för det tredje hur många gånger den verkligen kommer att användas.

P.E. menar att man är mer benägen att utveckla generalitet för applikationer som används i samband med en specifik produkt, vilket stämmer väl överens med tillvägagångssättet på Anoto. På grund av att P.E. arbetar inom ett konsultföretag, som utvecklar många produkter till flera olika kunder, är det svårare att hitta användningsområden för generella komponenter överlag. Grafiska gränssnitt och webbapplikationer kan dock ofta göras väldigt generella, även om de utvecklas till helt olika produkter och kunder, säger P.E.

Vi trodde före undersökningen att konsultbolag, som Sogeti, gör liknande applikationer till flera kunder och därför utvecklar många generella komponenter. Vår

undersökning visar dock att mångfalden av domäner som komponenter utvecklas för inom ett konsultbolag gör hanteringen av generella komponenter tungrodd. Generella komponenter är en mer integrerad del av utvecklingen på Anoto, som är ett företag som utvecklar en mer specifik produktlinje. P.E. menade att Sogeti säkerligen hade haft en väldokumenterad utvecklingsplan för generella komponenter om de hade sålt mer specifika produkter. Ju fler domäner man arbetar inom, desto högre krav ställs på organisationen kring komponentutvecklingen. Komponentbibliotek måste nå en större kritisk massa för att man ska få användning av komponenterna, vilket också gör detta betydligt svårare att underhålla och att hitta i. I Sogetis fall måste de också ta hänsyn till att deras kunder använder sig av olika plattformar, vilket också försvårar och gör organisationen kring komponenterna mer komplex. Utvecklar man komponenter som ska vara väldigt generella och användbara, måste man vara medveten om att kontextberoendet för komponenten kan öka (Szyperski m.fl., 2002, s. 45ff). Komponenter utvecklade för en viss plattform och som är skapade med dess speciella programspråk och förutsättningar, kan visa sig vara väldigt resurskrävande att implementera i en annan miljö.

M.W. menar att komponenterna kan vara mer generella för applikationer nära slutanvändaren, eftersom förutsättningarna inte ändras lika ofta som de gör nära maskinvaran, vilken ständigt är i utveckling: ”Ju specifikare applikationen är för Tetra Pak, desto specifikare komponenter efterfrågas.” På Anoto är det i princip samma situation med maskinvara i ständig utveckling. För att hantera detta letar man efter en gemensam nivå som kan använda samma generella komponenter. Det verkar som om det i många fall handlar om alltför stora förändringar nära maskinvaran för att det ska vara rimligt att utveckla generella komponenter för detta. På komponentnivå påverkar komponentens fysiska omgivning den konstanta och den variabla delen av komponenten på så sätt att när kontexterna är fundamentalt annorlunda mellan de olika användningsfallen, blir den variabla delen så komplex att man ofta tjänar i utvecklingstid på att göra komponenten specifik.,

I Tetra Pak R & D:s fall är det väldigt svårt att återanvända generella komponenter eftersom funktionaliteten hos komponenterna måste vara modern, vilket beror på att produkterna utvecklas långt innan de når marknaden. Detta ställer krav på att man har kännedom och erfarenhet av verksamhetsområdet för att kunna göra denna typ av estimeringar. Det verkar som att större krav ställs på organisationer som har fler produktlinjer – som till exempel vissa konsultbolag. Det är lättare att identifiera gemensam grundfunktionalitet då man specialiserar sig mot ett mindre antal likartade domäner.

Hur bestäms funktionalitetskraven på generella komponenter utifrån framtida behov?

För att bestämma funktionalitetskraven på generella komponenter jobbar M.B. tätt tillsammans med utvecklare från maskinvarusidan på Anoto. Genom det täta samarbetet skapas förutsättningar för att komponenterna ska kunna klara framtida förändringar. Man försöker få alla kundernas krav att sammanfalla i en gemensam plattform och för att planera detta gör man design reviews som försöker identifiera de minsta gemensamma nämnarna. Utifrån dessa bestäms sedan hur generell en komponent bör vara och vilka för- och nackdelar valet innebär. P.E. menar att de arbetar ungefär på samma sätt då de försöker finna subsystem som kan utvecklas till komponenter: ”Man letar alltid efter delar att bryta ut under designprocessen.” Det är en tydlig fördel att kunna arbeta nära maskinvaran som produkterna används till vid utvecklandet av generella komponenter. Det innebär att man kan ta del av maskinvaruutvecklarnas analyser om hur utvecklingen kommer att se ut i framtiden och på så sätt kartlägga vilka krav som kommer att ställas på de generella komponenterna.

På Sogeti gör man analyser tidigt i ett projekt över de krav som kan komma att ställas på produkten i framtiden. P.E. är dock ödmjuk inför att man inte kan täcka in alla behov och menar att man inte kan ta hänsyn till alla eventualiteter vid utvecklingen, eftersom man då får komponenter som han menar är oanvändbara. På IKEA gör man också analyser och man försöker sätta sig in i de olika situationer komponenten kan komma till användning i, enligt J.J. På Tetra Pak utgår man ofta från use cases för att se vilka situationer komponenten bör klara av och på så sätt bestäms deras generalitet.

På Anoto utvecklar man gränssnitt utifrån planeringen för att klara framtida behov av funktionalitet. M.W. anser också att en av de absolut viktigaste delarna i utvecklingen är begränsandet av komponentens funktionalitet, vilket sker genom att konstruera bra och tydliga gränssnitt. En komponent med ett precist utformat gränssnitt blir betydligt lättare att återanvända. Genom att begränsa funktionaliteten för komponenten, begränsar men även komponentens generalitet. Med tydliga gränssnitt tvingas komponenten till att vara okomplex och lättanvänd, vilket verkar prioriteras snarare än återanvändning av funktionalitet.

J.N. pekar på att man kan låta marknaden avgöra vilka funktioner komponenterna ska klara av, genom att låta kunderna bestämma komponenternas generalitet. Detta görs genom att man över tiden ser vilka komponenter kunderna har behov att förändra och hur förändringarna bör se ut och sedan implementera dessa förändringar i en gemensam lösning. Möjligtvis blir den här modellen lättare att organisera i framtiden, genom att komponenterna lättare kan distribueras digitalt. Om idén med superdistribution slår igenom behöver inte företagen vara oroliga för olaglig kopiering av komponenterna, vilket skulle kunna göra den här metoden populär.

Problem med komponenters generalitetsgrad

Frågeställningen som behandlas är: *Vilka problem kan uppstå beroende på planeringen av komponenters generalitetsgrad och hur hanteras dessa?*

Vilka problem kan uppstå och hur löser man dessa när komponenter är antingen för specifika eller för generella?

P.E. tycker oftast inte att det är några problem att komponenter är väldigt specifika, eftersom han tycker att det ofta ”överkomponentiseras”, med vilket han menar att det läggs ner för mycket tid och resurser på att göra komponenter generella. Detta leder i sin tur till bland annat sämre prestanda hos komponenterna, anser P.E.

M.W. menar att man helst undviker alltför generella komponenter vid inköp, eftersom dessa ofta inte har den efterfrågade funktionaliteten, utan de har istället generell funktionalitet. De är också svåranvända och M.W. anser att konfigureringen av generella komponenter ofta är väl så svår att förstå och sätta sig in i som vanlig programkod. Dessutom undviks väldigt generella komponenter, eftersom de ofta är fulla av buggar, fortsätter M.W. Generella komponenter bör ha högre tillförlitlighet än specifika komponenter, eftersom de ofta har testats i flera olika sammanhang (Mili m.fl., 2002, s. 501-503; Rine & Nada, 2002, s. 52). Diskrepansen kan bero på tillfälligheter men buggproblemen indikerar också att det är betydligt svårare att utveckla och testa generella komponenter.

Den ökade utvecklingstiden då man skapar en generell komponent gör att man måste få tillbaka kostnaden i form av snabbare utveckling. Alldeles för generella komponenter innebär därför en stor påfrestning på utvecklingsresurserna (Barnes & Bollinger, 1994, s. 18). Dessutom får användaren av en alltför generell komponent lägga onödigt stora resurser på att förstå komponenten, eftersom denna tenderar att bli mer komplex. Det är viktigt att tänka på att det inte endast är själva utvecklingen av generella komponenter som är kostsam, utan att även testningen och dokumentation med mera tar mer resurser i anspråk (a.a., s. 18). Precis som M.W. påpekar är även konfigureringen av de generella komponenterna kostsam och blir mer komplicerad desto komplexare komponenter som används är. Vid utveckling av generella komponenter bör man också tänka på vad som krävs för att implementera komponenterna när de väl ska användas – om komponenten är väldigt svår att implementera kan kostnaden överstiga vinsten. Kostnaden får i det här fallet bäras av dem som använder komponenten (a.a., s. 19f). I relation till detta bör man nämna att vinstpotentialen växer med komponentens innehåll, eftersom återanvändbarheten ökar. Detta förutsätter dock att man hittar användningsområden för komponenten, vilket är svårare ju större komponenten är och ju mer funktionalitet den innehåller.

Ibland har man problem med komponenter som inte är generella nog, säger P.E. Man löser oftast dessa problem genom refactoring, det vill säga att man hela tiden uppdaterar komponenten i små steg när kraven förändras.

J.J. stöter ofta på problem med att de ramverkskomponenter som används inom IKEA IT inte är generella nog för att kunna hantera en viss efterfrågad funktionalitet. I dessa fall kan man efterfråga att stöd för att den funktionalitet som saknas ska finnas med i nästa version. För att lösa problemet med funktionaliteten läggs denna istället i applikationen. Komponenterna inom IKEA IT planeras på en nivå och används på en annan. J.J., och de som jobbar på samma nivå som honom, kan inte göra så mycket när en ramverkskomponent inte är generell nog eller när den är alltför specifik, utan ansvaret finns istället hos kärngruppen.

Vilka faktorer avgör om man utvecklar en helt ny komponent eller om man modifierar en redan befintlig sådan?

J.J. menar att det sällan görs modifikationer av ramverkskomponenter. Anledningen till detta är att bakåtkompabiliteten är ett av de viktigaste elementen att ta hänsyn till, då komponenterna fortfarande kan finnas i bruk i olika applikationer. Istället utvecklas nya versioner av komponenter där så behövs. P.E. resonerar på samma sätt och låter gamla komponenter leva vidare i de system där de används och bygger snarare nya komponenter baserade på de gamla, även om uppdateringen är liten. Annars riskerar man att bygga vidare på komponenter som hela tiden är lite felaktiga och till slut har man en komponent som egentligen inte kan användas till någonting. När man bygger nya komponenter som baseras på äldre återanvänder man ofta på källkodsbasis. Vissa forskare menar att dylik extrahering inte är lönsam (Gall m.fl., 1995, s. 225) och det är därför intressant att notera att det är precis så P.E. beskriver utvecklingen på Sogeti.

M.W. resonerar att komponenternas gränssnitt också är komponenternas kontrakt för användning varför man aldrig modifierar gränssnittet. Eftersom de komponenter som utvecklas vid Tetra Pak också är så pass små, menar M.W. att det är oftast är enklare att utveckla nya komponenter än att modifiera de gamla.

En viktig faktor när man bestämmer om man ska utveckla en ny komponent eller modifiera en gammal verkar vara storleken på komponenten: Om man har lagt ner mycket utvecklingsresurser på en generell komponent är man mer benägen att modifiera denna istället för att konstruera en ny. Det betyder i så fall att benägenheten är starkt korrelerad till den allmänna komponentstrategin, det vill säga hur man planerar sin komponentutveckling i stort och alltså också vilken typ av verksamhet man bedriver. M.B. ser flera andra faktorer som också spelar in, som tid, risk och testbarhet. Tiden är skillnaden mellan att utveckla en ny komponent och att modifiera en redan existerande sådan. Testbarheten har med säkerheten att göra och om det är svårt att testa en komponent kan det vara bättre att behålla den gamla som man vet fungerar. Det finns även andra, mindre mät-

bara faktorer som spelar in, som till exempel kompetens man har till sitt förfogande under projekten: ”Om alla faktorer talar för att återanvända den gamla, men du har en kille som sitter och rullar tummarna som skulle vara perfekt till att skriva om komponenten, är det klart man utvecklar en ny.” M.B. påpekar att längden på den tidsperiod som komponenten kommer att användas inom också spelar in på valet. Det verkar som om man är mer benägen att utveckla en ny komponent om denna är viktig för verksamheten och man vet att den kommer att användas en tid framöver.

Det finns oftast brytpunkter för komponenter, menar M.B. En komponent som skrevs för en annan kravställning och som senare modifierats till den nuvarande kommer ofta till en punkt när underhållet och förändringarna kostar mer än vad det kostar att utveckla en helt ny komponent. Återigen ser man hur avvägningen mellan utvecklingstid och användning av en komponent är den mest centrala punkten för beslutsfattandet. Samtidigt verkar det som om erfarenhet och sunt förnuft spelar in i valet en hel del; M.B. visar till exempel att valet ofta avgörs av omständigheter, som att rätt personer är lediga vid rätt tidpunkt.

Slutsatser

De viktigaste slutsatserna som studien ledde fram till sammanställs och presenteras i detta kapitel.

Strategi, utveckling och funktionalitetskrav

Slutsatserna nedan strävar efter att besvara frågeställningen om hur organisationer planerar sin komponentutveckling för att skapa komponenter generella nog för återanvändning.

- En komponent som inrymmer mycket funktionalitet och som är generell för att kunna användas i flera olika fall ger ofta stora vinster när den används. Det är dock svårare att hitta användningsområden för komponenten ju större och ju mer generell denna är. Det gäller därför att finna en bra balans mellan storlek och generalitet å ena sidan och användbarhet å andra sidan.
- Tidsperspektivet verkar ofta inte vara långsiktigt, utan man ser istället till ett projekt i taget och utvecklar inte generella komponenter till framtida applikationer i någon större utsträckning. Undantaget verkar vara när utvecklingen sker mer produktspecifikt och för färre domäner.
- När utvecklingen sker långt innan produkterna kommer att nå marknaden blir tidsperspektivet speciellt viktigt, eftersom komponenterna redan vid lanseringen kommer att vara gamla. I de här fallen gäller det att använda sig av nya och moderna komponenter som fortfarande håller hög klass vid produktlanseringen.
- Ofta görs det bra analyser utifrån vilka möjligheter man har för återanvändning av programkod i form av komponenter. Komponenter utvecklas

på ett väldigt pragmatiskt sätt och endast då organisationerna verkligen är säkra på att komponenterna kommer att användas. Detta kan bero på tidigare erfarenheter om svårigheterna med att få till stånd en framgångsrik organisation kring generella komponenter. Vissa intäkter går säkerligen många organisationer förbi men också många kostnader. Det verkar finnas en viss oro för alltför hög koncentration på utvecklingen av generella komponenter.

- Det verkar vara vanligt att man inser fördelarna med generella komponenter, men ändå inte satsar på sådan utveckling. Ofta verkar det som man tar det säkra före det osäkra och därför utvecklar mer specifika komponenter. Anledningen till osäkerheten är ofta att man vet för lite om hur marknaden kommer att se ut i ett längre perspektiv och vilka krav som kommer att ställas på komponenterna. Det betyder också att man gör specifika komponenter som enkelt kan modifieras, vilket innebär att man får vidareutveckla komponenten i varje användningsfall. Fördelen är att inte hela kostnaden kommer på en gång och att man ungefär vet hur mycket resurser som går åt. Samtidigt är man försäkrad om att komponenten verkligen fungerar som den ska i det specifika fallet.
- Det verkar som att man är mer benägen att utveckla generella komponenter när man driver en produktspecifik utveckling. Är utvecklingen inriktad mot specifika produkter verkar det vara lättare att överblicka framtiden för programvaran och på så sätt göra investeringar i generella komponenter mindre osäker. Detta beror möjligen på att marknaden är lättare att överblicka för endast ett fåtal produkter samt i vissa fall även på att man kan arbeta tillsammans med maskinvaruutvecklarna och ta del av deras prognoser för framtida krav.
- Då organisationer har en större mångfald av domäner ökar också komplexiteten med att göra tillräckligt omfattande analyser. Detta ställer högre krav på verksamhetens organisation och gör hanteringen av generella komponenter mer tungrodd. Det ställs högre krav på en bredare kompetens inom verksamheten eftersom det utvecklas mer bredd i funktionaliteten.
- Det verkar som om många organisationer försöker hålla utvärderingar av sin komponentutveckling och av komponentnyttan till ett minimum. Möjligen är en orsak att man inte vill hämma programmerarnas kreativitet genom att införa tråkiga, administrativa uppgifter.

Generalitetsproblem

Slutsatserna nedan strävar efter att besvara frågeställningen om vilka problem som kan uppstå beroende på planeringen av komponenters generalitetsgrad och hur dessa hanteras.

- Om en komponent innehar en hög generalitetsgrad kan denna ge upphov till olika försämringar av komponentens kvalitet i form av att komponenten blir svårare att förstå och hitta användningsområden för. Dessutom kan komponenten lida av sämre prestanda och tillförlitlighet. Generella komponenter är inte alltid anpassningsbara, eftersom funktionaliteten är för generell och därför inte passar in väl någonstans. Detta går till viss del mot många komponentforskares uppfattning att generella komponenter snarare skulle vara av bättre kvalitet än specifika sådana, bland annat beroende på att de har testats i olika sammanhang.
- När komponenter är för specifika utvecklas ofta en helt ny komponent som är baserad på den gamla. Den förra komponenten fortsätter att användas av de system den används av, eftersom man ofta sätter bakåtkompatibiliteten först.
- Kontexten är en avgörande faktor när man avgör om man ska modifiera en äldre komponent eller utveckla en helt ny. När kontexten är statisk behövs ingen generalitet och när kontexten är fundamentalt annorlunda mellan de olika fallen är det slöseri med resurser att utveckla komponenter med hög grad av generalitet. I övriga fall, det vill säga då det finns mindre skillnader mellan användningskontexterna, utvecklas ofta generella komponenter.
- Storleken på en komponent avgör ofta om man ska modifiera en befintlig komponent eller om man ska utveckla en helt ny: Har man lagt ner mycket utvecklingsresurser på den befintliga komponenten är man mer benägen att behålla denna och endast uppdatera den.
- I vissa fall är det omständigheter som avgör om man ska utveckla en ny komponent eller endast modifiera en gammal, som till exempel hur arbetsuppgifterna för utvecklarna ser ut vid det specifika tillfället.

Förslag till fortsatt forskning

Ett intressant angreppssätt vore att undersöka en specifik organisation betydligt djupare än vad som var möjligt i den här undersökningen. Man kan tänka sig flera intervjuer med personer på olika positioner och inom olika avdelningar för att undersöka hur olika led inom en organisation uppfattar utvecklingen av generella komponenter och vilka som är för- och nackdelarna i olika fall. Vår undersökning visade att man ofta betraktar generella komponenter med viss skepsis och det vore intressant att undersöka det här fenomenet djupare, genom att till exempel bredda perspektivet med rena komponentutvecklare – det är möjligt att synen på generella komponenter skiljer sig markant på olika nivåer inom en organisation.

Det var flera intervjupersoner som talade om tjänster, som till exempel web services, och om hur deras verksamheter blivit mer och mer tjänstebaserade snarare än komponentbaserade. Grundtanken är liknande, men tjänster och komponenter implementeras väldigt annorlunda. Det är inte otänkbart att det är just tjänster som i vårt nätverkssamhälle uppfyller byggblocksvisionen inom programvaruutvecklingen snarare än generella komponenter och det vore spännande att undersöka detta.

Källförteckning

Arango, Guillermo, 1994, *A Brief Introduction to Domain Analysis*, Proceedings of the 1994 ACM symposium on Applied computing, Phoenix, USA, s. 42-46, http://portal.acm.org/ft_gateway.cfm?id=326656&type=pdf, 2005-04-29

Barnes, Bruce H. & Bollinger, Terry B., 1991, *Making Reuse Cost-Effective*, IEEE Software, nr. 1, volym 8, s. 13-24, http://www.idi.ntnu.no/grupper/su/courses/dif8901/papers2003/P-r12-barnes91.pdf, 2005-04-29

Bryman, Allan, 1997, *Kvantitet och kvalitet i samhällsvetenskaplig forskning*, Studentlitteratur, Lund

Cox, Brad, 1992, *What if there is a Silver Bullet and the competition gets it first*, Journal of Object-Oriented Programming, Juni, http://virtualschool.edu/cox/pub/92JOOPWhatIfSilverBullet/index.html, 2005-05-25

D'Alessandro, Massimo; Iachini, Pier Luigi & Martelli, Alessandra, 1993, *The Generic Reusable Component: an Approach to Reuse Hierarchical OO Designs*, Proceedings of 2nd International Workshop on Software Reusability, Los Alamitos, USA, s. 39-46, http://ieeexplore.ieee.org/Xplore/loginconcurrency.jsp?url=/iel2/897/7225/00291719.pdf, 2005-01-25

Gall, Harald; Jazayeri, Mehdi & Rerk, Klosch, 1995, *Research Directions in Software Reuse: Where to go from here?*, Proceedings of the 1995 Symposium on Software reusability, Seattle, USA, s. 225- 228, http://delivery.acm.org/10.1145/220000/211850/p225-gall.pdf?key1=211850&key2=7608872011&coll=GUIDE&dl=GUIDE&CFID=33660763&CFTOKEN=54075767, 2004-12-11

Jacobson, Ivar, 2001, *Foreword* från Heineman, George T. & Councill, William T., 2001, *Component-Based Software Engineering: Putting the Pieces Together*, Addison-Wesley, Boston, USA

Jacobson, Ivar; Griss, Martin & Jonsson, Patrik, 1997, *Software Reuse – Architecture, Process and Organization for Business success*, ACM press, USA

Karlsson, Evan-André, 1995, *Software Reuse – A Holistic Approach*, John Wiley & Sons Ltd., Chichester, England

Kvale, Steinar, 1997, *Den kvalitativa forskningsintervjun*, Studentlitteratur, Lund

McIlroy, M. Douglas, 1968, *Mass Produced Software Components*, Software Engineering, NATO Science Committee, Januari 1969, s. 138-150, http://cm.bell-labs.com/cm/cs/who/doug/components.txt, 2005-04-28

Miles, Matthew B. & Huberman, A. Michael, 1994, *Qualitative Data Analysis*, SAGE Publications Inc., USA

Mili, Hafedh; Mili, Ali; Sherif, Yacoub & Addy, Edward, 2002, *Reuse-Based Software Engineering – Techniques, Organization & Controls*, John Wiley & Sons, New York, USA

Nazareth, Derek L. & Rothenberger, Marcus A., 2004, *Assessing the cost-effectiveness of software reuse: A model for planned reuse*, Journal of Systems and Software, nr. 2, volym 73, s. 245-255, http://www.sciencedirect.com/science?_ob=MImg&_imagekey=B6V0N-4B9K8J6-6-1P&_cdi=5651&_orig=search&_coverDate=10%2F31%2F2004&_qd=1&_sk=999269997&view=c&wchp=dGLbVzb-zSkWW&_acct=C000041498&_version=1&_userid=745831&md5=51ee0630a96c6c05fe5fb95f0b00af0a&ie=f.pdf, 2004-12-15

Persson, Erik, 2002, *Shadows of Cavernous Shades – Charting the Chiaroscuro of Realistic Computing*, Ph.D. thesis, Dissertation 20, 2003, LU-CS-DISS:2003-1, Department of Computer Science, Lund University

Rine, David C. & Nada, Nader, 2000, *An empirical study of a software reuse reference model*, Information and Software Technology, nr. 42, s. 47-65, http://www.sciencedirect.com/science?_ob=MImg&_imagekey=B6V0B-3Y21WCF-5-5&_cdi=5642&_user=745831&_orig=search&_coverDate=01%2F01%2F2000&_sk=999579998&view=c&wchp=dGLbVzz-zSkWz&md5=8a2324c524c97de3ce63caf91109e5e9&ie=/sdarticle.pdf, 2005-05-12

Sitaraman, Murali; Kulczycki, Greg; Krone, Joan; Ogden, William F. & Reddy, A. L. N., *Performance Specification of Software Components*, Symposium on Software Reusability, Proceedings of the 2001 symposium on Software reusability: putting software reuse in context, s. 3-10, http://delivery.acm.org/10.1145/380000/375223/p3-sitaraman.pdf?key1=375223&key2=6203357111&coll=GUIDE&dl=GUIDE&CFID=46327022&CFTOKEN=78063473

Szyperski, Clemens; Gruntz, Dominik & Murer, Stephan, 2002, *Component Software – Beyond Object-Oriented Programming*, andra upplagan, Pearson Education Limited, Great Britain

Bilagor

Bilaga 1: Sammanfattning av intervju med Mattias Bryborn, Anoto AB

Mattas Bryborn är utvecklingsingenjör för programvara på *Anoto AB* vars utveckling är baserad i Lund. Mattias utbildade sig till civilingenjör i teknisk fysik i Lund med speciellt fokus på programvara med tekniska beräkningar. Mattias arbetar med att framställa den inbyggda programvaran i de optiska pennor som Anoto utvecklar, samt PC-program och serverprogram som används i det systemet. Han arbetar mest med moduleringen av programvara för de olika pennorna och är ansvarig för komponentutveckling för den inbyggda programvaran. Cirka 100 personer jobbar inom koncernen och ungefär 20 arbetar på Mattias avdelning.

De relativt nyutvecklade pennorna använder sig av absolut positionering vilket resulterar i vetskap om när något har skrivits, vilket papper som använts och var på pappret anteckningar skett. Detta ger en väldigt smidig automatisering av formulärhantering. Samarbete sker bland annat med Nokia, Logitech, Maxell, Hitachi och Hewlett Packard. Då tillverkning sker för flera aktörer som har lite olika krav på produkten men som ändå har samma behov av grundfunktionalitet, skapas komponenter som grund för detta. Ofta används olika chip eller kamerasystem i pennorna och då skrivs hela det lagret av programvara specifikt för produkten. ”På en viss nivå finns det ett konstant interface, men man får variera lite grand beroende på vilken teknik de stoppar in.” Resterande funktionalitet försöker man ha som generella komponenter. Om hårdvaran fungerar fundamentalt annorlunda kan det dock vara svårt att abstrahera denna och använda samma komponenter även i de andra lagren: ”Man får skära ett steg upp”, menar Mattias.

Den största delen av koden utvecklas ”på plats” för pennan för att användas i just den specifika produkt och koden som utvecklas till denna är sällan ämnad att användas någon annanstans. Däremot finns det ett gränssnitt som man har definierat för att klara framtida krav på funktionalitet eller hårdvara. För att klara olika förändringar inom ett funktionalitetsområde görs en komponent som är konstant, oavsett vilka kravändringar som kommer att göras, och en som man kan förändra. Tidsperspektivet är ”en produkt som fungerar ungefär som den här”. Komponenterna används med andra ord inte i andra fundamentalt annorlunda produkter, men däremot i liknande produkter som baseras på ändringar i nuvarande kravställning. För att förbereda komponenterna för kommande ändringar av maskinvaran samarbetar man med utvecklare från maskinvarusidan.

Det finns även väl definierade API:er (Application Programming Interface) för till exempel de data som pennan producerar eller olika papperslayouter för att lättare kunna arbeta med de program som finns runt pennan. Dessa system är mer rigorösa än pennans inbyggda system, eftersom även externa utvecklare jobbar mot dessa. Det finns också andra krav på portabilitet. Det finns alltså stora skillnader i de olika system Anoto tillverkar: de inbyggda systemen är flexibla för att klara framtida förändringar, men har ingen generalitet för olika plattformar, medan applikationerna som används utåt kan användas till olika plattformar, men inte förändras, eftersom de används av olika externa grupper.

De generella komponenterna i pennan, den variabla delen, tar mer tid att utveckla än den specifika delen. Det är dock inga *svarta lådkomponenter* (black box components) som utvecklas, utan de modifieras i princip alltid lite för en ny produkt. Mattias uppskattar att 90 % av arbetet läggs på att få de olika kundernas krav att sammanfalla i en gemensam plattform. Då gäller det att stödja både de krav och de maskinvaruförändringar som kommer att komma. Den delen som är specifik för varje kund försöker man göra så liten som möjligt.

Ibland uppstår det problem huruvida man ska utveckla en helt ny komponent eller man bara ska modifiera en redan befintlig. Till exempel i de fall det finns en intern komponent som ska börja användas externt. Hur man hanterar detta beror på hur det specifika fallet ser ut, men de faktorer som spelar in är tid, risk och testbarhet. Tiden är skillnaden mellan att modifiera en gammal gentemot att utveckla en ny, risken är att den nya inte kommer att fungera (”den gamla komponenten fungerar ju faktiskt, oavsett hur den ser ut”). Testbarheten avser hur lätt det är att göra heltäckande testning av en ny komponent. Är det svårt att testa en ny komponent väl, är det bättre att behålla den gamla: ”Är det många realtidsegenskaper inblandade så vill man gärna att komponenten ska ha körts ett tag.” Mattias understryker att det inte bara handlar om tiden i sig, utan även om vilka resurser man har i projektet: Finns det någon med rätt kompetens som kan göra det? ”Om alla faktorer talar för att återanvända den gamla, men du har en kille som sitter och rullar tummarna som skulle vara perfekt till att skriva om komponenten, är det klart man utvecklar en ny.” Den som är bäst lämpad att avgöra va-

let är oftast den som hittills har utvecklat den aktuella komponenten. Det spelar också stor roll hur länge produkten kommer att användas.

Det finns oftast brytpunkter för komponenter, menar Mattias. En komponent som skrevs för en annan kravställning och som senare modifierats till den nuvarande kommer ofta till en punkt när underhållet och förändringarna kostar mer än vad det kostar att utveckla en helt ny komponent. Vad gäller portabilitet mellan olika plattformar finns det alltid begränsningar i vilka funktioner man får använda i de olika språken beroende på vilken plattform det handlar om, vilket gör att man letar efter den minsta gemensamma nämnaren. Det betyder också att någon lösning får ta smällen och inte bli lika enkel som den skulle kunna ha blivit.

Det görs ingen formell utvärdering av komponenterna, men däremot görs design reviews innan utvecklingen börjar för att bolla för- och nackdelar. När komponenten också är en produkt görs dock formella utvärderingar, eftersom det då handlar om supportavtal och produktstrategier med mera.

Bilaga 2: Sammanfattning av intervju med Patrik Eriksson, Sogeti

Patrik Eriksson utbildade sig vid Människa-dator-programmet, som han beskriver som systemvetenskap med mer psykologi och fokus på användarvänlighet, i Ronneby och tog sin examen 1993.

Sogeti ingår i CapGemini-koncernen och meningen är att Sogeti är en mer lokal spelare med mer fokus på tekniken än CapGemini. Sogeti är oftast ute hos kunden och utvecklar. De är runt 70 stycken anställda på kontoret i Helsingborg.

Patrik har titeln systemarkitekt och systemutvecklare inom Microsoftprodukter.

Vad gäller återanvändning av kod har varje region specialiserat sig på olika områden. Behöver något av de andra kontoren en lösning inom det området hör de av sig. Det finns inget direkt kodbibliotek inom organisationen som alla kan vända sig till och det finns heller ingen instans som samordnar återanvändningen. Istället är olika personer inom ett visst kompetensområde inom Sverige men också i övriga Europa knutna till e-postlistor där man kan ställa frågor och få hjälp i mån av tid. Det är frivilligt att vara med i e-postlistorna och man behöver inte svara på frågor för att undvika stress.

Patrik tror att enkelheten i e-postlistorna är en stor del av framgången och menar att det är lättare att verkligen använda dem eftersom förfrågningarna automatiskt hamnar i personens e-postlåda och man inte behöver göra en aktiv insats: ”får man inte den flaggan så blir det inte av”. Patrik tror att det är likadant med stora system för återanvändning: ”Är det inte extremt enkelt och inte bygger på något slags automatisk interaktivitet är det mycket svårt att få det att fungera väl.” Ibland är det flera personer som löser samma problem på e-postlistan, men det löses genom att de som tror sig ha en lösning hör av sig till personen med förfrågan direkt och denne kan sedan välja att korrespondera vidare med en speciell person.

Det finns också ett källkodsbibliotek i Helsingborg som hålls uppdaterat och som faktiskt används. Patrik påpekar speciellt att det handlar om just källkod, till exempel klasser, och inte färdiga *svarta lådkomponenter* (black box components). Patrik säger dock att det finns en viss tendens att källkoden blir mer och mer färdig och liknar komponenter allt mer.

Patrik är skeptisk till den klassiska bilden av komponenthantering där en organisation först bygger komponenter med ett strikt gränssnitt som sedan kan antingen säljas vidare eller användas internt. Han tror att det möjligen kan fungera internt, men utåt tror han snarare på olika typer av tjänster, som till exempel *web services*, vilket beror på att det blir allt enklare att kommunicera säkert. Fördelen är att man slipper att underhålla komponenterna, vilket många kunder ofta ställer sig skeptiska till, bland annat eftersom det mer eller mindre innebär att organisa-

tionen blir knuten till en specifik organisation som tillhandahåller komponenten. En service kan man köpa från vem som helst om man har ett väl definierat gränssnitt. Patrik ser trenden tydligt även hos en del av de kunder han jobbar med, som börjar använda tjänster också internt mellan sina olika interna system. Patrik menar att det mellan organisation och också mellan avdelningar inom en organisation kommer att finnas ett tjänstelager som kommunicerar utåt och att dessa internt sen mycket väl kan vara uppbyggda av komponenter; ”Komponenterna har kanske flyttats in lite grand och används inte lika mycket utåt”. Däremot tror han att det fortfarande kommer att hända att man köper in en komponent som löser ett visst problem.

Hittills använder Sogeti i Helsingborg mest tjänster internt för att exponera olika delar av ett system utåt och säkert komma runt brandväggen. Det görs genom att endast gränssnittet – i form av en ”dum webbapplikation” – finns utanför brandväggen och den kritiska delen av applikationen innanför och dessa kommunicerar genom web services. Än så länge är det alltså inte web services som kommunicerar externt, utan en vanlig webbapplikation, men Patrik tror att det också kommer i större skala: ”Jag tvekar inte över att det kommer att explodera. Det är bara en tidsfråga.”

Vad gäller utvecklingen av komponenter, beror metoderna på vilken arkitektur det gäller. Patrik som jobbar med Microsoftkomponenter använder de designer som Microsoft föreslår. Genom erfarenhet utifrån detta har det skapats insikt i hur man bäst gör olika typer av projekt. Problemet är dock, menar Patrik, att ”vad som är rätt i ett läge är nästan rätt i ett annat läge och så bygger man vidare på det”. Det är stor skillnad mot konsultbolag och ett produktbaserat företag, menar Patrik vidare, eftersom det inte byggs samma typ av system runt en specifik produkt: ”För oss är det mer en kompetensfråga”, menar Patrik och tillägger att om de hade utvecklat en produkt så hade det varit värdefullt att ha Sogetis arbetssätt för att göra komponenter väl manifesterat och sedan använda detta gång på gång.

En relativt stor del av den totala utvecklingen är komponentutveckling, beroende lite på hur man definierar det, menar Patrik. Om man menar att en väl definierad klass också är en komponent om den är byggd som en komponent med tydliga gränssnitt men inte förpackad som en komponent, är det mycket av utvecklingen som rör just komponenter. Det är samma problem som man löser, enligt Patrik: Återanvändningen och inkapsling. Så fort utveckling sker, letar man efter subsystem, som man kan komponentisera, enligt Patrik: ”Man letar alltid efter delar att bryta ut under designprocessen.” Patrik tror att själva återanvändningen är lite överskattad, eftersom ”varje projekt är unikt hur man är gör det” och enligt tidigare erfarenheter behöver man ändå sätta sig in i koden igen och göra modifieringar och på det sättet lever komponenten vidare i nya projekt. Detta kan bero antingen på att komponenten utvecklades för specifikt, men Patrik menar också att teknologierna och tänkesätten förändras med tiden och man vill kanske inte dela upp det på samma sätt som man en gång gjorde. Patrik trycker särskilt på

att det inte behöver vara en nackdel att komponenten är specifik, eftersom det är ofta man ”överkomponentiserar”, det vill säga man gör för många och för specifika komponenter, med bland annat dåliga prestanda som följd. Han jämför detta med att dra objektorienterad systemutveckling för långt och förlorar prestanda: ”Objekttanken kanske inte är lika vacker när den kommer upp i skala.” Dessutom, poängterar Patrik, måste man nå resultat och leverera i tid och man kan inte lägga ner all tid på designen. Patrik tror att trenden går från att designa väldigt mycket generella komponenter med luddiga krav. I dag handlar det snarare om refactoring, vilket innebär att en version av komponenten byggs och sedan bygger man vidare på den när kraven ändras. ”Det är en mer lättrörlig process och det viktigaste är att komponenten är lätt att förändra.” Patrik menar att det är lätt att hamna i fällan att ”designa för mycket i början av projektet och bygger för många nice to have-grejer och till slut har man ett helt kodbibliotek med komponenter som man kanske kommer att ha. Sen använder man kanske tio procent eller tjugo procent och det blir ändå inte vad kunden vill ha”. På samma sätt vet Patrik många kunders om har bränt sig på att bygga upp en organisation runt återanvändning som sedan inte fungerar som de hade tänkt sig.

Ibland blir det problem med generaliteten för de komponenter som utvecklas internt inom organisationen. Ibland har man en bas som man tror ska fungera, men så visar det sig att den inte är generell nog. Detta kan ha flera orsaker, som till exempel att folk byts ut under projektet och inte fullföljer de initiala tankegångarna. Om en komponent inte längre kan användas i nya projekt av olika anledningar, får den oftast leva vidare i de system som den redan är implementerad i, och nya, uppgraderade versioner byggs för framtida bruk. I de fallen återanvänder man ofta på kodbasis från den gamla komponenten. Då sker också en analys över vilka krav som kommer att ställas på komponenten men Patrik poängterar att man inte kan täcka in alla behov: ”Det är ingen idé att göra det så finkornigt som möjligt, för då är det helt plötsligt inte användbart.”

Det är stor skillnad på återanvändbarheten mellan olika typer av applikationer. Exempelvis grafiska gränssnitt till webbapplikationer är generella och används väldigt ofta med endast små variationer.

Bilaga 3: Sammanfattning av intervju med Josef Nedstam, Lunds tekniska högskola

Josef Nedstam är doktorand vid institutionen för telekommunikationssystem vid *Lunds tekniska högskola*. Josef har i sitt forskningsarbete undersökt en rad företag som har gjort strukturella förändringar för att hantera återanvändbar programkod inom organisationen. Han har undersökt programvaruarkitekturen, det vill säga det som finns runt komponenterna och som gör avgör hur dessa passar varandra, på en rad företag. Han har även undersökt hur programvaruarkitekturen har vidareutvecklats i takt med att företaget och marknaden har förändrats. Det är dock svårt att avgöra hur lyckade projekten har varit, eftersom vinsterna i de flesta fall först kommer långt senare.

Josef har speciellt forskat på produktlinjetänkandet, där man istället för att fokusera på en produkt snarare ser en hel produktlinje och där generella komponenter utvecklas för denna. De generella komponenterna modifieras sedan för att passa en viss produkt inom produktlinjen.

Det finns en rad problem med att ha en organisation som hanterar generella komponenter i ett företag, enligt Josef. Problemet är att komponenterna – även om de är funktionella – inte används i den utsträckningen som det var tänkt. Anledningarna kan vara många, men Josef pekar på att programmerarna inte trivs med komponenter som någon annan har utvecklat, och menar att det här till största delen beror på makten och utrymmet man har när man själv har skapat komponenten: ”Man har makt över det man har skapat själv; man kan ändra det hur man vill.” Det är också problem med tillförlitligheten, eftersom man inte är säker på att man får det man egentligen vill ha.

Josef menar dessutom att ”att makt och resurser hamnar i de projekt som är direkt inriktade på att utveckla en produkt”. Som exempel nämner Josef ett företag som han har arbetat med i sin forskning som har svårt att ge tillräckliga resurser till de generella komponenter, eftersom det mesta går till att utveckla själva produkten. Komponentutvecklingen och underhållet av dessa blir lidande, eftersom det inte tillhör den primära verksamheten för organisationen. Speciellt i slutskedet av en produktutveckling, när alla krafter är inriktade på att få produkten färdig i tid, finns det lite utrymme för design för återanvändning, och utvecklarna är istället inriktade på att få produkten färdig. Det verkar också som att det är mer status att vara delaktig i ett projekt som utvecklar en produkt, enligt Josef. En vanlig attityd är att det vore bra att utveckla återanvändbara komponenter, men att man inte har tid, även om man långsiktigt skulle vinna tid.

Josef ser hur många företag vet för lite om marknaden för att följa produktlinjemetoden fullt ut; det är svårt att veta hur marknaden ser ut när investeringen är

färdig och produkterna kan börja utvecklas och investeringen börjar ge avkastning. Senarelägger man produktutvecklingen, finns det också en risk att man missar viktiga möjligheter på marknaden.

Det finns inga uppenbara skillnader på hur lätt det är att införa en organisation för återanvändning inom ett företag beroende på storleken, menar Josef. Däremot tror han att det i viss mån är lättare att få de anställda att göra som man säger åt dem i större företag.

Ett sätt att kringgå ovissheten om vilka komponenter som man behöver göra generella och som kommer att användas är att låta marknaden styra komponenternas generalitet. Josef talar om ett företag som han har undersökt som utvecklar en komponentplattform, som företagets kunder sedan kan modifiera fritt på filnivå. När en ny version av plattformen ska utvecklas, tas hänsyn till vilka förändringar de olika kunderna har gjort och man försöker hitta en gemensam lösning som inbegriper alla kunders krav på funktionalitet. På det här sättet ser man inte bara hur generella de olika komponenterna bör vara, utan också vilka komponenter det finns ett behov av att göra anpassningsbara. Josef säger dock att problemet med att ha en stor organisation bakom återanvändningen med mycket administration kvarstår.

Bilaga 4: Sammanfattning av intervju med Jon Jarnsäter, IKEA IT

Jon Jarnsäter jobbar som systemutvecklare vid *IKEA IT* i Helsingborg vars huvudsakliga uppgift är att stödja IKEA med olika informationssystem och applikationer. Av de 1000 personer som jobbar för IKEA IT runt om i världen sitter ungefär 400 i Helsingborg. Jon har ingen högskoleutbildning, utan har läst tekniskt program på gymnasiet och tidigare haft en del tillfälliga arbeten som programutvecklare. Utöver sin gymnasieutbildning läste även Jon påbyggnadskurser under en tidigare anställning.

Jon jobbar med både programmering för server- och klientsidan; *Java* för serversidan och *Visual Basic 6* och *Visual Basic .NET* för klientsidan. Kommunikationen mellan dessa sker med *XML* genom olika tjänster (services). All utveckling görs för tjänster, som är en betydligt viktigare del av återanvändningsstänkandet än komponenter inom IKEA IT.

En relativt stor del av programkoden består av komponenter, men dessa utgör endast ett ramverk till applikationerna och definieras och utvecklas av en annan utvecklingsavdelning, kallad kärngruppen, på IKEA. Dessa är sedan färdiga att använda och modifieras sällan ytterligare. Däremot kan man skicka en förfrågan till kärngruppen om man saknar något i en viss ramverkskomponent och detta kan uppdateras i nästa version.

Det utvecklas även komponenter direkt av applikationsutvecklarna, men den största delen består av ramverkskomponenter. Återanvändningen av de egenutvecklade komponenterna sker oftast informellt genom att man frågar sina arbetskamrater, men det finns också mer formella kodbibliotek. Jon tycker dock att biblioteket inte fungerar speciellt bra och kan tycka att det är lite svårt att hitta den funktionalitet som eftersöks.

Om ramverkskomponenterna inte är generella nog, kan man inte ändra direkt i komponenten, utan istället får man programmera den extra funktionaliteten applikationsspecifikt, alternativt konstruera en extra komponent, även om den borde tillhöra ramverkskomponenterna. Det är relativt ofta Jon stöter på problem med att de tjänster som används inte är anpassningsbara nog att hantera alla de situationer han vill använda dem i. Som exempel nämner han att storleken på informationsutbytet mellan servern och klienten kan vara större än vad utvecklarna har dimensionerat tjänsten för. Det är aldrig så illa att problemen hindrar utvecklingen nämnvärt.

Jon strävar att göra så stor del som möjligt av programkoden återanvändbar, men det sker i princip alltid på tjänstenivå, det vill säga att han implementerar en tjänst av en funktion som han tror sig kunna använda fler gånger. Tjänsterna ut-

görs av *EJB* (Enterprise JavaBeans). När Jon utvecklar en tjänst så försöker han fundera på de olika situationer tjänsten kan komma att användas i, för att den ska kunna anpassas till framtida krav. Visar det sig att tjänsten inte var generell nog är det sällan man modifierar den, utan man utvecklar snarare en ny version av komponenten, eftersom bakåtkompatibiliteten är extremt viktigt när flera applikationer använder samma tjänst. Planeringen för hur tjänsterna ska se ut och vilka situationer de ska kunna användas i sköts av Jon själv och ibland efter överläggningar med den grupp av systemutvecklare han själv ingår i. Däremot finns inga formella möten som rör just design av generalitet.

En del av återanvändningen sker också genom generatorer som skapar programkod. Man matar in specifikationer för hur det ska fungera och sedan automatgenereras kod både för serversidan i Java och för klientsidan i Visual Basic .NET.

Eftersom planeringen av generalitet och återanvändning i stort sker inofficiellt på Jons avdelning, sker det heller ingen officiell utvärdering.

Bilaga 5: Sammanfattning av intervju med Mattias Wallinius, Tetra Pak

Mattias jobbar på *Tetra Paks* Research & Development-avdelning som mentor för programvaruutveckling. Han är utbildad till maskiningenjör vid Chalmers och har sedan glidit över till att arbeta mer med programvaruutveckling. Som mentor är han inblandad i de flesta systemutvecklingsprojekt inom R & D och dessa projekt består huvudsakligen av utveckling för styrsystem till Tetra Paks maskiner samt en del hjälpsystem som används internt.

En stor del av utvecklingen är komponentbaserad, vilket beror på att man inte kan utveckla allt själv, utan måste köpa in vissa delar. Dessutom ”kan man inte utveckla stora monoliter”, utan systemen byggs hellre upp av mindre, utbytbara delar. Det är ofta kostnaden som avgör när man köper in en komponent eller när man utvecklar den själv. Det är dock svårt att utvärdera vilken användning man kommer att få av komponenten, eftersom man inte kan veta hur länge den är duglig eftersom kontexten för den ändras ofta.

R & D köper oftast in nya komponenter med modern funktionalitet för att dessa inte ska vara omoderna när produkterna väl når marknaden, vilket är viktigt eftersom utvecklingstiden för en produkt är väldigt lång.

Komponentstrategin går ut på att man utvecklar systemen utifrån *use cases* och *user stories*. Ur dessa kan identifieras olika designmönster och utifrån dessa bestäms vad som ska implementeras som komponenter, vad som kan köpas in och vad som utvecklas specifikt för applikationen. Mattias understryker att komponenterna inte bör vara generella, utan snarare små, specifika komponenter. Detta görs för att utvecklarna gärna lägger till finesser i komponenterna, vilket tar mycket tid och som dessutom sällan används. Tidsaspekten är viktig och Mattias menar att utvecklingen som mest får ta ett halvt år och man måste kunna byta delarna i systemet. För att systemen inte ska bli omoderna snabbt bygger man dessa i så små delar som möjligt för att kunna byta ut endast de delar som är omoderna. Man tar hela tiden små steg framåt i utvecklingen och ändrar något när man märker att det behövs. Mattias menar att Tetra Pak har utvecklat en hel del generella komponenter tidigare, som varit tänkt att kunna användas för flera olika applikationer inom verksamheten, men som oftast har slutat som misslyckande.

En viktig del av strategin är att inte betrakta systemen som avslutade bara för att utvecklingsprojektet är det. Mattias menar istället att så fort ett projekt slutar, tar ett nytt vid. Utvecklingen går i cykler och när en produkt är levererad kommer det snart feedback, vilken fungerar som *use cases* för nästa utvecklingscykel.

Det viktigaste i utvecklingen är enligt Mattias att designa gränssnitt och på så sätt begränsa komponenterna. Han menar att ju tydligare definierat en kompo-

nents gränssnitt är, desto mer användbar är den. Både under den egna utvecklingen och när komponenter köps in är det viktigt att komponenterna inte är för generella, vilket beror på att de är svåra att använda och att de dessutom ofta saknar den funktionalitet som efterfrågas; de har alltför ”generell” funktionalitet, enligt Mattias. Generella komponenter är ofta väldigt konfigurerbara, vilket Mattias betraktar mer eller mindre som programkod som kan vara väldigt svår att förstå och man behöver versionshantering med mera. Man brukar också undvika komponenter som har alltför många finesser, eftersom dessa också brukar vara fulla av buggar, enligt Mattias. Ofta används öppen källkodskomponenter, eftersom dessa, enligt Mattias, ofta gör precis det de ska göra. Mattias menar att komponenter utvecklade ur öppen källkodsprojekt ofta har fått en större funktionalitetskontroll genom sina många utvecklare. Det kan ofta vara svårt att hitta komponenter med den eftersökta funktionaliteten hos tredje part och speciellt svårt att finna det man söker är det på de lägre nivåerna i utvecklingen. Här blir det så att man oftast får förlita sig på mer egenutvecklad programkod.

När man börjar en ny utvecklingscykel går man igenom komponenterna och uppdaterar det som man hade problem med under tidigare utvecklingscykler eller det som man tvingats undvika. Ofta gör man helt enkelt en ny komponent snarare än att modifiera en gammal, vilket fungerar bra eftersom komponenterna vanligen är ganska små. Dessutom får man aldrig ändra i komponentens gränssnitt, eftersom det är ”dess kontrakt med omvärlden”, menar Mattias.

Man försöker planera för generalitet gentemot olika plattformar, även om all utveckling sker med *Microsofts .NET*-plattform, genom att försöka använda mönster och implementeringar som i princip också fungerar på en *Java*-baserad plattform. De flesta designmönster fungerar till båda plattformarna, vilka är de enda plattformarna man bryr sig om i dagsläget, enligt Mattias.

Det är stor skillnad på synen på generalitet beroende på vilken typ av system det gäller. ”Ju mer specifik applikationen är för Tetra Pak, desto mer specifika komponenter är det”, menar Mattias. Ju längre från maskinerna och ju närmare slutanvändaren man kommer, desto mer generella komponenter kan användas, eftersom dessa inte behöver kunna omdefinieras och bytas ut lika snabbt som de maskinnära komponenterna: ”Det är behoven som styr.”

Tetra Paks R & D är inte speciellt bra på utvärdering, erkänner Mattias, utan menar att synen på utvärderingen är till stora delar pragmatisk. Mattias tycker att man ska akta sig för alltför mycket tungrodd administration, eftersom det lätt tar död på kreativiteten. Man utvärderar dock komponentnyttan i varje projekt, genom att se vilken nytta man har haft av komponenten i tidigare projekt. På så sätt ges en bra överblick över vilka komponenter som är lönsamma att satsa på i framtiden.