LUNDS UNIVERSITET
Ekonomihögskolan

Enterprise Application Integration Patterns

En studie kring hur EAI-patterns påverkar en integrationsutvecklare

Kandidatuppsats, 15 högskolepoäng, SYSK02 i informatik

Framlagd: 2014-06-02

Författare: Nikolai Rosengren Jacobsen
Omar Saka

Handledare: Nicklas Holmberg
Examinatorer: Anders Svensson
Magnus Wärja

Abstrakt

Titel: Enterprise Application Integration Patterns -
En studie kring hur EAI-patterns påverkar en integrationsutvecklare

Författare: Nikolai Rosengren Jacobsen
Omar saka

Utgivare: Institutionen för informatik

Handledare: Nicklas Holmberg

Examinator: Anders Svensson
Magnus Wärja

Publiceringsår: 2014

Uppsatstyp: Kandidatuppsats

Språk: Svenska

Nyckelord: Mönster, integrationsutvecklare

Abstrakt
Idag använder sig organisationer av många olika system med många olika preferenser. För att systemen ska kunna vara kompatibla och kommunicera med varandra behöver de integreras. Mjukvaruutvecklare inom integration tillhandahåller den här tjänsten och använder sig av mönster för att integrera de olika systemen. Dessa mönster är abstrakta lösningsförslag som är erkända best practice metoder.

Den här uppsatsen undersöker integrationsutvecklares inställning och uppfattning gentemot mönster. Den empiriska undersökningen genomfördes på Enfo Zystems som är ett företag specialiserat inom integration. Den empiriska datan hämtades från fem integrationsutvecklare via individuella intervjuer.

Studien visar att olika mjukvaruutvecklare inom samma företag har olika definitioner och uppfattningar om vad och hur mönster bör definieras. Däremot erhåller alla utvecklare att mönster bidrar med en kvalitetshöjning av slutresultatet samt att överanvändning utav det ger ett sämre slutresultat. Att bibehålla en balans mellan att utnyttja mönster där det behövs och inte behövs verkar förbli en fråga för den individuella utvecklarens omdöme vid varje enskild integration utan några givna direktiv ifrån den egna organisationen.

Innehållsförteckning

1 Inledning

I det här kapitlet presenteras bakgrunden till uppsatsämnet, problemformuleringen, forskningsfrågorna, rapportens syfte samt avgränsningar.

1.1 Bakgrund

Standardiserade lösningar på återkommande problem inom integration har varit svårt att utveckla. Problematiken har gjort att istället för konkreta lösningar i form av kod så har mönster skapats som ger utvecklarna riktlinjer för hur arbetssättet och metodiken skall gå till för att lösa ett problem (Scheibler & Leymann, 2008). Dessa mönster är experters erfarenheter och kunskap inom ett område där det inte finns standardiserade lösningar som löser alla problem som till exempel objektorienterad design eller integrationslösningar (Hohpe & Woolf, 2011). Ett stort hinder idag för organisationer och utvecklare är fastställandet av normer för hur dessa mönster ska användas. (Scheibler & Leymann, 2008).

Därför har mönster för *"Enterprise Application Integration (EAI)"* utvecklats som beskriver återkommande problem och föreslår lösningar på en abstrakt nivå (Scheibler & Leymann, 2008). EAI-mönster är en dokumentation som arkitekter och utvecklare använder och tar i beaktande när en integrationslösning ska implementeras. Det specificerar inte hur kod ska utvecklas utan hur en implementation kan gå tillväga. Dessa mönster har inte utvecklats i traditionell anda utan snarare upptäcks med hjälp av "trial and error" där erfarna utvecklare och arkitekter har tagit sina erfarenheter, misslyckanden och lösningar på ett återkommande problem och omsatt de i lösningar som har utmynnat i mönster enligt Hohpe & Woolf (2004). Ett antal ramverk för *"Enterprise Architecture"* har presenterats på senare år för att kunna definiera och avgränsa EAI såsom Zachmans ramverk, the Department of Defense Architecture Framework (DoDAF) och The Open Group Architecture Framework (TOGAF) (Scheibler & Leymann, 2008).

De här mönstren ger mer än bara lösningförslag för integrationsutvecklare på återkommande problem. De fungerar även som referensmaterial för att kunna föra kunskap vidare till mindre erfarna medarbetare och tillsammans skapar de en enhetlig terminologi för att kunna diskutera integration med andra kollegor och eliminerar förvirring. (Hohpe & Woolf, 2004). Dessa gemensamma nämnare som mönstren erbjuder, skapar förenkling för en integrationsutvecklare i dess dagliga arbete och inbjuder till diskussioner mellan kollegor. Utmaningarna som en integrationsutvecklare möter varje dag skapar ett behov av en bank med enhetliga lösningar och terminologier för att kunna arbeta effektivt.

1.2 Problemformulering

För att skapa konkurrenskraft använder sig dagens verksamheter av ett flertal system med olika preferenser. Enligt Hohpe (2003) är det fundamentalt att de här systemen kan kommunicera med varandra för att reducera manuellt arbete. För att olika system ska kunna samverka tillsammans krävs det att de integreras. Enterprise Integration innebär att olika system kan köras på olika plattformar som befinner sig på olika geografiska platser enligt Hohpe (2003). Enligt författaren medför det att integration inom storskaliga projekt inte är enkelt att genomföra, då många faktorer behöver beaktas.

Nedan följer några av de utmaningar och problem som Hohpe (2003, 2007) anser uppstå vid integrationsutveckling:

- Integrationer berör i synnerhet legacy-system som inte utvecklats med åtanken att samverka med andra system. Ett hinder vid utveckling av integrationer är att integrationsutvecklaren inte har tillåtelse att ändra i de berörda systemen. Det här medför att utvecklaren måste komma fram till en lösning som kan kompensera bristerna i legacy-systemen.
- För att uppnå en framgångsrik integration krävs det inte bara att kommunikation mellan olika applikationer upprättas, utan även mellan affärsenheter och IT-avdelningar. Det här krävs eftersom ingen grupp har specifik kontroll över en applikation längre, då den enskilda applikationen tillhör en gemensam grupp av applikationer och tjänster.
- Integrationer berör oftast en stor mängd viktiga funktioner i en verksamhet, det är därför viktigt att de fungerar som det är tänkt.
- Drift och underhåll av EAI-lösningar är inte särskilt lätt att hantera då de ofta består av en mix av olika system.
- Det bör finnas en accepterad språkstandard inom mjukvaruindustrin för att undvika missförstånd då individer kan tolka betydelsen av mönster på olika sätt. EAI-mönster kan användas och fungera som ett gemensamt språk för att undvika det här problemet.

Därmed vill vi undersöka på individnivå hur integrationsutvecklare använder sig av mönster, deras uppfattning, terminologi, behov och inställning till mönstren. Detta leder oss till forskningsfrågorna:

På vilket sätt påverkar integrationsmönster en integrationsutvecklare vid systemintegration?
– Vilka för- och nackdelar finns det, sett ur en integrationsutvecklares perspektiv?

Vi ämnar besvara forskningsfrågan och underfrågan genom intervjuer med olika integrationsutvecklare på ett företag för att få en uppfattning om hur varje individ tolkar ett gemensamt mönster som används på arbetsplatsen och inom projekt samt om de känner att mönster behöver användas för att uppfylla syftet med en EAI.

1.3 Syfte

Studien avser att identifiera vilken påverkan och för- respektive nackdelar en integrationsutvecklare finner med ett EAI-mönster vid systemintegration.

1.4 Avgränsning

Vi har valt i vår studie att avgränsa oss till applikationsintegration med hjälp av mellanprogramvaran *"Enterprise Service Bus (ESB)"*, där fokus kommer ligga på integrationsutvecklares tillämpande av mönster vid implementeringen av integrationer. Vi kommer endast undersöka hur arbetssättet och mönstren påverkar en integrationsutvecklarens arbete och inte andra områden som ledning, planering och support.

2 Bakomliggande arkitektur för integrationsmönster

I det här kapitlet ämnar vi att förklara den litteraturen som har använts för att beskriva den bakomliggande arkitekturen för EAI-mönster. Kapitlet påbörjas med beskrivning av arkitekturen rent allmänt och dess respektive för- och nackdelar. Det fortsätter med Service Oriented Architecture för att påvisa hur olika arkitekturer kan appliceras vid användningen av mönster samt skillnader mellan dessa. Slutligen fokuserar kapitlet på förklaring av mönster och även för- och nackdelarna. Kapitlet är indelat i en trädstruktur som går in på arkitektur för att beskriva för läsaren hur det övergripliga ser ut för att sedan smalna av och fokusera på mönster. Slutligen presenteras en sammanfattning av kapitlet för att underlätta och öka förståelsen av ämnet för läsaren samt påvisa kopplingarna mellan arkitekturen, teknologin och mönstren.

2.1 Enterprise architecture

För att kunna skapa en vy över ett företag behöver organisationer kartlägga sin verksamhet, processer, applikationer och den tjänsteorienterade arkitekturen. Relationer mellan dessa måste definieras. För att definiera detta måste strukturen för *"Enterprise Architecture (EA)"* kartläggas vilket är indelat i två delar - verksamhetsarkitektur och tjänsteorienterad arkitektur. Verksamhetsarkitekturen baseras på metadata och hur dess beståndsdelar används i en verksamhet. Vid definition av metadata specificerar IT-experter det som "data om data" enligt Finkelstein (2006) men dessa definitioner är meningslösa för en icke-teknisk verksamhetchef. De beskriver inte i detalj vad som skiljer en verksamhetsintegration från en tjänsteorienterad integration. Det grundar sig i hur olika delar av verksamheten har utvecklats med olika terminologier enligt Finkelstein (2006). Författaren beskriver att behovet av ett enhetligt språk inte har upptäckts tidigare när IT och automatiserade processer infördes och detta orsakar problem när en integration ska genomföras och för att lösa dessa problem måste det införas en enhetlig terminologi.

För att beskriva EA och uppnå transparens mellan en organisations olika delar behövs en fastställd arkitektur. Enligt Lankhorst et al. (s.3, 2012) kan Enterprise Architecture beskrivas som:

> "A coherent whole of principles, methods, and models that are used in the design and realization of an enterprise's organizational structure, business processes, information systems, and infrastructure."

Enterprise architecture underlättar identifieringen av de viktigaste komponenterna inom en organisation. Samtidigt som det underlättar identifieringen så erbjuder EA, flexibilitet och anpassning förklarar Lankhorst (2012). Utan en god arkitektur är det därmed svårt att uppnå framgång. EA erbjuder även användarna en holistisk syn på en organisation, d.v.s. helheten är större än summan av delarna. Exempelvis kan enligt Lankhorst (2012) en optimerad tjänsteorienterad infrastruktur erbjuda en god prestanda men införandet av agila arbetsmetoder och föränderliga arbetsprocesser skapar problem. För att motverka detta berättar Lankhorst (2012) att ett kvalitativt EA balanserar dessa krav mot varandra och

översätter organisationens översiktliga strategi till den operativa nivån.

2.1.1 Enterprise Architecture och verksamheters informationsutbyte

EA har satt en standard för arkitektur och hur olika nyckelfaktorer ska identifieras men organisationer har än idag problem med att distribuera kunskap och information över enheter. Detta problem uppstår hos stora företag och de som har sina enheter spridda över stora geografiska områden. Varje unik enhet har stor självständighet och egenutvecklade lösningar på återkommande problem enligt Bo och Yellin (2007). Det skapar därmed specifika arbetssätt utspritt över olika enheter och det överensstämmer inte med verksamhetens mål och vision. Bo och Yellin (2007) förklarar att det försvårar IT-enhetens arbete för att snabbt kunna svara på förändringar i processer och kunna integrera dessa mellan olika enheter. Författarna poängterar att verksamheter som arbetar på det här sättet har ofta svårt att kontrollera distribueringen av ekonomiska medel, appliceringen av best practices och återanvändningen av teknologier över enheter vilket resulterar i högre kostnader för utveckling och underhåll. Bo och Yellins (2007) summering resulterar i att organisationer därmed möter stora utmaningar och måste utveckla en plattform för att kunna dela teknologier över flera enheter och även kunna återanvända dessa i andra sammanhang.

För att lösa problemen enligt Bo och Yellin (2007) som uppstår kan organisationer fastställa EA standarder för att vägleda IT-avdelningar och affärsavdelningar i deras tekniska val såväl som beslut på projektledningsnivå relaterat till olika applikationer. Författarna dikterar att användningen av EA bestämmer hur underavdelningar ska agera utan att precisera på processnivå. Anpassningen till standarder uppkommer inte automatiskt utan det kräver styrningsmekanismer som underlättar implementationen av EA standarder. När styrningsmekanismerna används som riktlinjer vid en implementationen kan förbättringar i verksamheten mätas och fastställas (Bo & Yellin, 2007).

2.1.2 Enterprise Architecture för- och nackdelar

Användningen av EA standarder erbjuder en rad potentiella fördelar för verksamheter. Enligt Boh & Yellin (2007) påstår ett flertal användare av EA standarder att det erbjuder fördelar kostnadsmässigt och effektivitetsmässigt. Genom att standardisera arbetssättet enligt Bo och Yellin (2007) över flera plattformar, teknologier och arkitekturer kan organisationer reducera komplexiteten i deras operativa verksamhet, minimera kunskapsnivån när det gäller att hantera IT-systemen och erbjuda återanvändning av komponenter och tjänster. EA kan också erbjuda verksamheter en integration över deras olika plattformar och hjälpa de att uppnå målen samtidigt som det går snabbare att utveckla applikationer då EA erbjuder flexibilitet (Boh & Yellin, 2007).

Implementeringen av EA standarder är inte smärtfri. För att effektivisera användningen krävs det att organisationer avsätter tid och resurser för att skapa, implementera och underhålla

standarderna (Bo & Yellin, 2007). De rätta intressenterna måste vara inblandade för att upprätta rätt standarder som är enhetliga med organisationens översiktliga affärsmål (Bo & Yellin, 2007). Enligt Bo och Yellin (2007) måste omfattningen av arkitekturen vara noggrant kontrollerad för att projektet inte ska blir för stort eftersom att det kan resultera i år av utveckling och resultatet kan bli otillräckligt vilket leder till nedläggning. Författarna påpekar även att verksamhetens anställda och övriga individer måste vara insatta i resultatet och ge sitt medgivande för att projektet ska bli lyckat. Detta är faktorer som måste tas hänsyn till vid ett beslut att implementera EA standarder.

2.1.3 Enterprise Architecture - Application Integration

Multinationella organisationer som är spridda över stora geografiska områden med självständiga enheter arbetar oftast med hundratals applikationer som kan vara egenutvecklade, köpta av underleverantörer, ett legacy-system eller en kombination av dessa. Att skapa en gigantisk applikation för hela verksamheten är omöjligt. (Hohpe & Woolf, 2004). För att slippa problemen med flera system som har svårt att kommunicera med varandra eller skilja på arbetsuppgifterna kan de integreras men det är en komplex process. Verksamhetsintegration måste hantera flera applikationer på många plattformar som är spridda över olika geografiska områden. Hohpe och Woolf (2004) förklarar därmed att leverantörer av dessa tjänster har specialiserat sig inom verksamhetsintegration och tillhandahåller *"Enterprise Application Integration (EAI)"* som erbjuder plattformsoberoende, språkoberoende och moduloberoende applikationer som integrerar alla dessa faktorer till ett system.

Därmed erbjuder EA en arkitektur som fokuserar på affärsmålen för verksamheten genom att integrera applikationer som tidigare nämnt. En annan arkitektur som är tätt sammankopplat med EA är "*Service Oriented Architecture (SOA)*" som erbjuder en tjänsteorienterad arkitektur och fokuserar på självständiga tjänster och deras uppbyggnad för att få en verksamhet som kan anpassa sig efter omständigheterna. Dessa två olika arkitekturer stödjer integration, även om det är baserat på olika tillvägagångssätt.

2.2 Service oriented architecture

Service Oriented Architecture (SOA) beskriver enligt Arsanjani et al,. (2009) *vad* som ska göras men inte *hur* det ska göras för att uppnå en flexibel arkitektur för verksamheten som kan anpassas efter omständigheterna. För att uppnå flexibilitet behöver organisationer arbeta agilt. Eftersom SOA inte är någon teknologi utan en konceptuell arkitektur, krävs det en teknologi för att stödja det underliggande konceptet. SOA realiserar den konceptuella arkitekturen genom att förena verksamhetsprocesserna och modellera stora applikationer till tjänster. Definitionen av SOA skiljer sig beroende på vem som tillfrågar och vad som efterfrågas. Enligt Holley och Arsanjani (2010) finns det många definitioner av SOA som inriktar sig specifikt till olika systemintegratörer, arkitekter och företag.

Enligt Arsanjani et al., (refererad i Choi et al,. 2010, s. 3) kan Service-Oriented Architecture definieras som;

> "SOA is the architectural style that supports loosely coupled services to enable business flexibility in an interoperable, technology-agnostic manner. SOA consists of a composite set of business-aligned services that support a flexible and dynamically re-configurable end-to-end business processes realization using interface-based service descriptions."

Dessa definitioner krockar inte med varandra utan snarare kompletterar. Enligt Holley och Arsanjani (2010) kan exempelvis beskrivningen av SOA för en verksamhetschef bestå av reducering av kostnader för att utveckla applikationer, maximera investeringen verksamheten gör i olika teknologier samt reducera ledtider för att leverera värde till verksamheten.

För en IT-arkitekt kan SOA stå för en arkitektonisk stil som framhäver lösningar för att integrera olika system genom lös koppling och återanvändning av moduler, komponenter och tjänster enligt Holley & Arsanjani (2010). Lös koppling i en arkitektur syftar till hur pass beroende moduler, komponenter och tjänster är av varandra. Om det finns få beroenden påvisar arkitekturen en lös koppling medan vice versa syftar till en hård koppling mellan dessa faktorer. Om beroenden existerar, kan en förändring i en tjänst påverka de övriga och detta minskar flexibiliteten i arkitekturen (Karhikeyan & Geetha, 2012).

För en integrationsutvecklare är SOA något som tillhandahåller metoder och verktyg för systemutveckling och integration varvid det fokuserar kring funktionalitet för verksamheten genom skapandet av tjänster. Enligt Holley och Arsanjani (2010) separerar således SOA funktioner till tjänster som utvecklare gör tillgängliga för återanvändning och kombination, oavsett plattform. Författarna påpekar därmed att det skapas plattformsoberoende tjänster som kan användas, återanvändas och kombineras utan att det påverkar användarens upplevelse.

För att uppnå en flexibel arkitektur kan syftet med SOA begränsas till sex riktlinjer som enligt Arsanjani et al,. (2009) förklarar vart SOA lägger tyngdpunkten på när det gäller att utveckla arkitekturen. Arsanjani et al,. (2009, s. 1) anser i deras manifest att följande prioriteringar bör realiseras vid införande av SOA:

- "**Business value** over technical strategy
- **Strategic goals** over project-specific benefits
- **Intrinsic interoperability** over custom integration
- **Shared services** over specific-purpose implementations
- **Flexibility** over optimization
- **Evolutionary refinement** over pursuit of initial perfection"

Det finns ingen konkret, bestämd standard för hur en tjänst ska utformas precis som Arsanjani et al,. (2009) förklarar. Det kan realiseras genom en rad tekniker och olika standarder. Erl (2005) utvecklar detta och påvisar principer för hur en tjänst kan utformas för att få bli kallad "tjänsteorienterad". En tjänst måste först och främst vara designad för att kunna återanvändas och kunna dela information mellan sig och andra tjänster. De tjänster och komponenter som existerar ska inte vara alltför beroende av varandra och kunna vara självgående samtidigt som de ska vara lättförstådda för användare. Dessa designprinciper präglar framtagningen av komponenter och en tjänsteorienterad arkitektur enligt Erl, (2005).

2.2.1 Realiserande teknologier

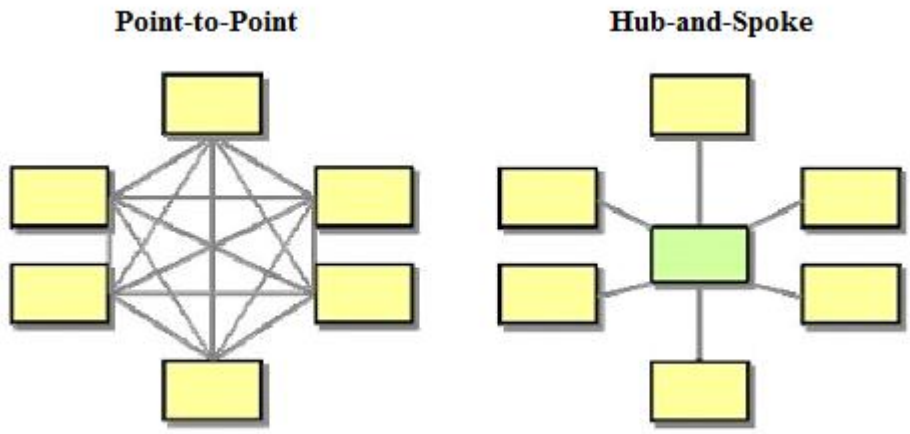
Termen webbtjänster syftar till en grupp tekniker och standarder, och är en av de teknologier som kan användas för att realisera SOA. Webbtjänster är designade för att skapa interoperabilitet mellan olika system över Internet. Interoperabiliteten uppnås genom *"Web Service Definition Language (WSDL)"*, *"Simple Object Access Protocol (SOAP)"* och *"Universal Description, Discovery and Integration (UDDI)"* som är en uppsättning av XML-baserade standarder. (Holley & Arsanjani, 2009)

Utöver webbtjänster kan vanliga programmeringsspråk och mellanprogramvara användas, oavsett plattform för att realisera SOA. Särskilt de teknologier som direkt implementerar tjänstegränssnitt med WSDL och kommunicerar med XML-meddelanden, J2EE, CORBA och IBM´s Websphere MQ är några exempel på mellanprogramvara som stödjer WSDL. (Papazoglou & Van Den Heuvel, 2007)

2.3 Enterprise service bus

Enligt Papazoglou och Van Den Heuvel (2007) är ESB en öppen, standardiserad mellanprogramvara som är designad för att implementera, driftsätta och hantera integrationslösningar. Den förser interoperabilitet mellan applikationer och andra komponenter via standardiserade kopplingar och gränssnitt. ESB:n kan både transportera och omvandla meddelanden som skickas mellan gränssnitt, vilket möjliggör att systemen kan använda olika programspråk och vara fysiskt placerade på olika platser. Papazoglou och Van Den Heuvel (2007) förklarar att mer specifikt så ansvarar ESB:n för kontroll, flöde, och omvandling av alla meddelanden mellan tjänster som använder olika meddelandeprotokoll. Syftet med ESB:n är att verksamheten ska kunna samla applikationer och komponenter som grupper av tjänster för att bilda sammansatta verksamhetsprocesser, och detta medför att verksamhetsfunktioner kan automatiseras (Papazoglou & Van Den Heuvel, 2007).

Figuren (Fig. 2.1) nedan framhäver problematiken med att inte ha en strategi, teknologi eller mönster för att hantera integrationer. Modellen till vänster visar en klassik integration, även kallad ”Point-to-Point”, där systemen är separat integrerade med varandra. Metoden kan användas vid integration av två systemen men blir komplex och svårhanterlig när fler system tillkommer.



Figur 2.1 Point-to-Point integration (till vänster) vs Hub-and-Spoke integration (till höger) (mod. efter Hohpe & Woolf, s.287-288, 2004).

Som modellen till vänster visar så krävs det 15 integrationer för att integrera sex applikationer. Genom att använda mönstret Hub-and-Spoke (till höger) kan integrationerna reduceras till endast antalet system som kommunicerar. Det här påvisar att användningen av mönster kan minimera komplexiteten och antalet integrationer som krävs. Användningen av det här mönstret skapar en lös koppling mellan systemen i jämförelse med Point-to-Point metoden. Vilket resulterar i ett minskat beroende mellan systemen och medför att de kan ändras, uppdateras och bytas ut utan att påverka de övriga. Detta reducerar underhållskostnader och ökar flexibiliteten i verksamheten.

För att tjänster ska kunna kommunicera med varandra så innehåller en ESB ändpunkter som är destinationer där tjänster skickar och tar emot meddelanden (Papazoglou & Van Den Heuvel, 2007). Ändpunkterna definieras med ett destinationsnamn och en URL-adress och tillåter tjänster att kommunicera genom anslutningsnamn. Papazoglou och Van Den Heuvel, (2007) förklarar att under tiden ESB:n körs så kopplas anslutningsnamnen till fysiska nätverksdestinationer och detta medför destinationsoberoende som gör det möjligt att uppgradera, flytta eller ersätta tjänster som är kopplade till ESB:n utan att behöva ändra någon kod eller störa befintliga applikationer. Ändpunkter förser även asynkron och pålitlig kommunikation mellan tjänster. Förenklat kan det förklaras genom att kommunikation garanteras även vid nätverksfel och strömavbrott (Papazoglou & Van Den Heuvel, 2007).

2.4 Skillnader mellan EAI och SOI

"Service-oriented integration (SOI)" använder SOA eller webbtjänster för att integrera applikationer. Konceptet har utvecklats från EAI, med ytterligare tillägget att använda tjänster och kontrakt. Detta gör det möjligt att skapa en uppsättning av löst kopplade gränssnitt som kan kommunicera med varandra för att uppnå syftet med integrationen, vilket resulterar i flexibilitet. SOI kan använda en ESB vid integration, men det finns inget krav för detta då SOI oftast utgår från SOA och webbtjänster. (Holley & Arsanjani, 2010)

EAI integrerar applikationer och system genom att använda en mellanprogramvara (middleware), lösningarna är teknologiskt baserade men oftast inte baserade på standarder som webbtjänster. Det här är den huvudsakliga skillnaden mellan EAI lösningar och SOA. Utöver EAI erbjuder SOA förbättringar i form av användandet av tjänster. SOA introducerar en högre abstraktionsnivå än EAI genom användningen av tjänster och kontrakt. EAI använder vanligtvis *"Application Programming Interfaces (API)"*, vilket motsvarar servicegränssnitt och kontrakt i en SOA.(Holley & Arsanjani, 2010)

EAI har två huvudsakliga integrationsmönster: Hub-and-Spoke, och Publish-Subscribe. I Hub-and-Spoke mönstret informerar en applikation en broker om ett event. Broker:n tar hand om transformeringen av meddelandet och delegering av meddelandet samt utlöser rätt åtgärd i klientapplikationen. Genom intern logik vet broker:n vilka klientapplikationer som ska anropas. I Publish-Subscribe mönstret publicerar en applikation, events till en mellanprogramvara. Applikationen agerar således som en ”provider”. Klientapplikationer kopplas sedan till eventen baserat på regler. Efter att eventen har publicerats, är det mellanprogramvarans uppgift att skicka det transformerade meddelandet till de kopplade applikationerna.(Holley & Arsanjani &, 2010)

En annan skillnad mellan EAI och SOA är enligt Holley och Arsanjani (2010) att det vanligtvis inte finns någon direkt relation mellan requester och provider-applikationer i EAI, medan det existerar i SOA. I en EAI informerar applikationer en mellanprogramvara om något event som har utförts, men de ignorerar vad som händer i nästa led. Ansvaret ligger således på mellanprogramvaran att transformera och delegera meddelandet. I en SOA-lösning

anropar en klientapplikation en tjänst, som sedan förmedlas genom ESB:n till applikationen som förser tjänsten. Dock kan detta beteende enligt författarna även konfigureras i EAI-lösningar.(Holley & Arsanjani, 2010)

Trots skillnaderna kan både EAI och SOI använda en ESB som mellanprogramvara. För att skapa lösningar och kopplingar mellan olika applikationer inom en ESB kan integrationsutvecklare använda mönster. Mönstren fungerar som en riktlinje för hur lösningen kan appliceras men det preciserar inte koden för implementationen. De tjänstgör som abstrakta lösningsförslag.

2.5 Designmönster och historik

Konceptet för designmönster härstammar från Alexander et al., (1977) teori om arkitektoniska mönster för byggnader. Alexander et al., (1977) insåg behovet och fördelarna med att etablera designmönster inom arkitektur och byggindustrin. Alexander et al., (1977) bidrag till industrin, gjorde det möjligt för utbildade och outbildade individer inom arkitektur att hantera och lösa vanligt förekommande problem. Författarna definierade redan år 1977 mönster enligt följande:

> ”Each pattern describes a problem which occurs over and over again in our environment, and then describes the core solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way”(Alexander et al., 1977, s. 10)

Alexander et al., (1977) beskrivning av designmönster stämmer väldigt bra överens med tjänsteorienterade designmönster.

Konceptet med ett mönsterspråk dök senare upp i Donald Normans och Stephen Drapers bok *User Centered System Design*, som publicerades 1986. Boken föreslog tillämpningen av mönsterspråk vid interaktionsdesign, vilket är praxis för att designa interaktiva digitala produkter, miljöer, system och tjänster för människor (Friesen, 2012).

Under tiden började Kent Beck och Ward Cunningham studera mönster och deras tillämplighet på mjukvarudesign. (Friesen 2012). År 1987, tog de fram fem designmönster som de dokumenterade i rapporten *Using Pattern Languages for Object-Oriented Programs* som de vidarebefordrade till OOPSLA-87(Object-oriented Programming, Systems, Languages & Applications). (Beck & Cunningham, 1987). Enligt Friesen (2012) användes dessa mönster för att hjälpa Tektronix´s Semiconductor Test Systems Group, som hade problem med att avsluta ett designprojekt. Beck och Cunnigham följde Alexanders förslag att låta användarna i projektet bestämma designens utfall och även förse designmönster för att underlätta arbetet (Friesen, 2012).

Erich Gamma insåg också betydelsen av återkommande designmönster under tiden han arbetade med sin doktorsavhandling. Han ansåg att designmönster kunde underlätta uppgiften

att skriva återanvändbar objektorienterad mjukvara, och funderade på hur dem skulle kunna dokumenteras och kommuniceras på ett effektivt sätt (Friesen, 2012). Före 1991 års europeiska konferens om objektorienterad programmering, började Gamma och Richard Helm att utforma en katalog för mönster. På en OOPSLA workshop som hölls 1991, träffade Gamma och Helm, Ralph Johson och John Vlissides. De kom sedan att skriva den populära boken *Design Patterns: Elements of Reusable Object-Oriented Software*, som behandlar 23 designmönster (Friesen, 2012).

2.5.1 Definition av mönster

Designmönster dokumenterar framgångsrika och beprövade metoder för att lösa vanligt förekommande designproblem inom mjukvaruutveckling. (Dong et al,. 2011). Designmönster infördes inom mjukvaruutveckling för att på ett effektivt sätt dela och sprida lösningar på problem som utvecklare vid upprepade tillfällen stött på vid objektorienterad programmering. (Baroni, 2003). Konceptet för designmönster bygger på att man inte ska återuppfinna hjulet igen, utan istället återanvända beprövade och fungerande metoder.

Mönster kan användas som en beskrivning för att lösa ett specifikt designproblem. Ett designmönster innehåller generellt sätt en grupp av deltagande klasser, deras relationer och kollaborativa beteende (Dong et al., 2011). Varje klass definieras i form av vilken roll den har i designmönstret. En generell designprocess av mönsterbaserad utveckling omfattar normalt sätt valet av designmönster baserat på användarens krav och tillämpningen av mönstret i designen (Dong et al., 2011).

Enligt Gamma et al., (1994) har mönster fyra fundamentala element:

- Ett namn som genom ett eller två ord kan användas för att beskriva ett designproblem, dess lösningar och konsekvenser. Genom att namnge ett mönster kan en gemensam terminologi etableras, vilket underlättar kommunikation sinsemellan kollegor.
- Ett problem som beskriver när mönstret bör tillämpas. Det förklarar problemet och dess sammanhang.
- En lösning som beskriver elementen som utgör designen, deras relationer, ansvar och samverkan. Lösningen beskriver inte en specifik design eller implementering, eftersom ett mönster fungerar som mall som kan tillämpas i många olika situationer. Istället ger mönstret en abstrakt beskrivning av ett designproblem och hur det kan lösas genom en grupp av element.
- Konsekvenser, är resultat och avvägningar för att tillämpa mönstret. De är kritiska för att utvärdera designalternativ och för att förstå vilka kostnader och fördelar som tillämpningen av mönstret bidrar med. Eftersom återanvändbarhet är en viktig faktor i objektorienterad design, inkluderar konsekvenser för ett mönster, dess påverkan på systemets flexibilitet, utbyggbarhet och portabilitet.

Mangalaraj et al., (2014) utförde en studie där de bland annat undersökte påverkan av användningen av designmönster för att lösa ett designproblem. Det fanns flera positiva

aspekter med att använda designmönster. I studien kom de fram till att objekten som använde designmönster, slutförde uppgiften fortare än de som inte använde mönster. Slutresultatet var dessutom av högre kvalitet och de var mer tillfredsställda med uppgiften. Författarna förklarar att även utifrån deras resultat så kan mönster effektivisera mjukvaruutvecklingen oavsett utvecklarens erfarenhet (Mangalaraj et al., 2014).

2.5.2 För- och nackdelar med mönster

Olika källor definierar fördelarna med mönster enligt följande;

- Effektiviserar utvecklingsprocessen. (Mangalaraj et al., 2014)
- Bidrar med en gemensam terminologi. (Gamma et al., 1994)
- Mönster inkapslar erfarenhet och best practices för att lösa ett problem. (IBM, 2005)
- Individen som användare ett mönster behöver inte ha någon kunskap om hur ett mönster designas, det måste dock finnas god dokumentation om ett mönster för att användaren ska kunna hitta, välja och applicera mönstret. Användaren måste veta vilket problem mönstret löser och konsekvenser för appliceringen av det. (IBM, 2005)
- Mönster är beprövade best practice metoder. (Dong et al., 2011)
- Skapandet och användningen av mönster främjar återanvändbarhet. Principen för ett mönster är att det skapas en gång men kan återanvändas många gånger. Vilket kan resultera i sänkta utvecklingskostnader och besparing av tid genom att lösning inte behöver designas på nytt. (IBM, 2005)
- Delning av mönster kan ske i projekt, hela verksamheter eller mellan flera verksamheter. Begränsningar för mönster avgörs av mönstrets design och syftet för ett mönster bestäms av grundaren. (IBM, 2005)
- Användningen av mönster bidrar till ett slutresultat med högre kvalitet. (Mangalaraj et al., 2014)

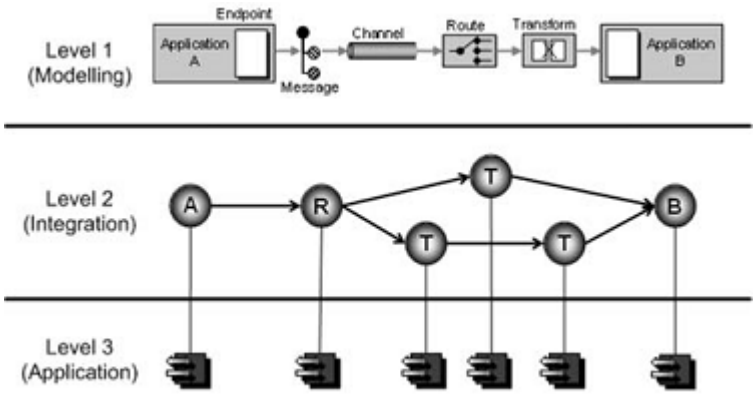
Det förekommer även nackdelar med mönster. Mangalaraj et al., (2014) berättar att designmönster ger inte bara fördelar per automatik utan det finns svårigheter med att implementera och använda de inom en organisation. Eftersom mönstren är abstrakta lösningsförslag måste utvecklare vara noggranna vid implementationen av ett specifikt mönster för att lösningen måste anpassas utefter problemet (Mangalaraj et al., 2014). Om detta inte görs noggrant kan resultatet bli lidande enligt Mangalaraj et al., (2014). Det kan även leda till oönskade funktioner som orsakar många fel i koden och gör det svårt att underhålla applikationen enligt Prechelt et al., (2001).

2.5.3 Enterprise Application Integration Patterns

“Enterprise Application Integration (EAI)” syftar till att integrera stora fristående applikationer med egna artefakter (tjänster och processer). EAI appliceras på stora system

och applikationer som är självständiga och får de att kommunicera med varandra med hjälp av lös koppling (Scheibler & Leymann, 2008). Detta syftar till att modulerna i applikationerna anropar närmaste modul och fungerar även självständigt om närmaste process eller modul skulle haverera. Målet med EAI är därmed inte en stor applikation för hela verksamheten som exempelvis ERP-system, utan det syftar till att få de olika applikationerna att kommunicera och samarbeta med varandra (Scheibler & Leymann, 2008).

Många EAI-scenarion har samma lösningar som används flera gånger för återkommande problem. Dessa problem och de lösningar som finns adresseras oftast som mönster. Förutom de väletablerade mönstren som redan finns inom objektorienterad mjukvarudesign, så finns det även mönster för färdiga EAI-lösningar. Scenariona består av tre lager enligt Scheibler och Leymann (2008); modellering, integration och implementering. Figuren (Fig. 2.2) nedanför beskriver de tre nivåerna. Enligt Scheibler och Leymann (2008) arbetar arkitekterna på den översta nivån med att modellera systemet på en abstrakt nivå, vilket betyder att verktygen som ska användas inte preciseras utan bara själva arkitekturen för systemet modelleras. Den här nivån syftar till att underlätta kommunikationen mellan arkitekten och den mer tekniska personalen som sköter de underliggande nivåerna. Integrationsnivån syftar till att binda ihop tjänsterna eller att integrera applikationer med hjälp av arkitektens ritningar (Scheibler & Leymann, 2008). Det är på kodnivå och den koden representerar och realiserar affärsnivån. Det betyder att koden som produceras ämnar till att uppfylla verksamhetsmålen för projektet. Nivå tre behandlar de applikationer som ska integreras (Scheibler & Leymann, 2008).



Figur 2.2 De tre nivåer av Enterprise Application Integration (Scheibler & Leymann, 2008, s. 2).

Skillnaden mellan modelleringsnivån där arkitekten beskriver integrationen visuellt och integrationsnivån framställer ett mellanrum mellan arkitektur och IT-nivån. Det arkitektoniska och det tekniska vilket syftar till tomrum mellan dessa nivåer (Scheibler & Leymann, 2008). Därmed arbetar integrationsutvecklaren enligt Scheibler och Leymann, (2008), självständigt för att arkitekten inte specificerar vilka verktyg som ska användas. Det gör att utvecklaren kan använda mönster självständigt för de problem som kan uppstå med integrationen (Scheibler & Leymann, 2008).

Enligt Hohpe och Woolf (2004) kan mönster inom EAI som integrationsutvecklaren använder beskrivas enligt många olika strukturer. Författarna har valt en viss struktur som består av elva sektioner. Var och en av dessa sektioner beskriver beståndsdelarna för ett specifikt mönster;

- **Namn**; Det fungerar som en identifikation för mönstret och vad det gör. Författarna använder det i en kontext som kan appliceras för att minska förvirringen bland arkitekter och utvecklare.
- **Ikon**; En ikon används för varje mönster för att underlätta modelleringen för arkitekten. Detta fungerar som ett visuellt komplement till det verbala som används bland arkitekter och utvecklare.
- **Kontext**; Mönstret beskrivs i en kontext där tidigare problem har lösts genom att använda detta specifika mönster. Det refererar även till tidigare mönster som utvecklaren kan ha använt.
- **Problem**; Beskrivningen av mönstret inkluderar även en problemställning som hjälper utvecklaren att förstå om just det specifika mönstret kan appliceras för att lösa problemet.
- **Olika krafter**; Den här beståndsdelen syftar till begränsningarna som gör problemet svårlöst. Det visar alternativa lösningar som inte har fungerat för att visa värdet av den korrekta lösningen.
- **Lösning**; Det är en mall som beskriver lösningen för utvecklaren. Det betyder inte att det är en lösning som passar alla omständigheter men det underlättar och visar lösningen när det har applicerats framgångsrikt i ett tidigare problem.
- **Modell**; Enligt författarna behövs en modell som medföljer lösningen för att underlätta för arkitekten/utvecklaren. Idén betonar att genom granskning av namnet för mönstret och modellen kan lösningen snabbt förstås av alla.
- **Resultat**; Det bygger vidare på lösningen och visar hur det kan appliceras i olika omständigheter. Det påvisar även nya, möjliga utmaningar som kan uppstå vid appliceringen av mönstret.
- **Påbyggande**; Det visar ytterligare mönster som kan appliceras efter den nuvarande lösningen. Oftast visar implementeringen av ett visst mönster, nya problem. Påbyggandet syftar till nya mönster som möjligtvis kan appliceras på de nya problemen. Det resulterar i en katalog av mönster som tillsammans bildar lösningar på återkommande såväl som nya problem.
- **Sidofält**; Varje mönster måste bestå av en viss sektion som beskriver de mer tekniska aspekterna med mönstret. Därmed blir det lättare för användaren att hoppa över dessa delar beroende på om de är tekniskt lagda eller inte.
- **Exempel**; Ett mönster ska även bestå av exempel när eller om det har applicerats. Det kan även vara exempel på hur mönster namnges eller väldigt detaljerat som hur det används rent kodmässigt. (Hohpe & Woolf, 2004)

Strukturen för ett mönster möjliggör för en utvecklare att få en överskådlig blick över vad implementeringen kommer medföra och när det bör appliceras. Mönsterstrukturen är generell

och möjliggör för oss i vår undersökning att konstatera om intervjuobjekten tar hänsyn till det här. Det är en viktig aspekt då det bör finnas ett syfte med implementering av ett mönster.

2.6 Sammanfattning av litteratur

Vid genomgången av den bakomliggande arkitekturen är det viktigt att verksamheter kartlägger sina processer och applikationer. För att definiera den här strukturen kan företag implementera Enterprise Architecture som tar hänsyn till föränderliga arbetsprocesser och agila arbetsmetoder och bibehåller flexibiliteten (Lankhorst, 2012). Fastän EA har satt en standard för hur information ska distribueras över enheter inom en organisation har flertalet verksamheter än idag problem med detta. För att många enheter är självständiga och har egna arbetsrutiner. För att motverka detta måste organisationer fastställa EA standarder för att vägleda avdelningarna inom organisationen (Bo & Yellin, 2007).
Vissa leverantörer har specialiserat sig på att koppla ihop applikationer som är egenutvecklade, köpta eller legacy-system eftersom många verksamheter som har enheter spridda över stora geografiska områden arbetar med olika system. Den integreringen av tjänster och applikationer som tillhandahålls kallas för Enterprise Application Integration (EAI) som erbjuder plattformsoberoende, språkoberoende och moduloberoende applikation som integrerar alla dessa faktorer till ett system (Hohpe & Woolf, 2004).

En annan arkitektur som är tätt sammankopplat med EA är Service Oriented Architecture (SOA). Det är en konceptuell arkitektur som beskriver *vad* som ska göras men inte *hur* det ska göras (Arsanjani et al., 2009). SOA eftersträvar att omvandla verksamhetsprocesser till tjänster som kan återanvändas och tillhandahålla en flexibel arkitektur (Arsanjani et al., 2009). Erl (2005) förklarar att en tjänst ska utvecklas för att kunna dela information mellan sig själv och andra tjänster samt så ska tjänsterna vara självgående samtidigt som de ska vara lättförståda för användare. En teknologi som stödjer EAI och SOA är Enterprise Service Bus som är en mellanprogramvara mellan applikationer. Den förser interoperabilitet mellan applikationer och tillhandahåller kopplingar och gränssnitt (Papazoglou & Van Den Heuvel, 2007). ESB:n möjliggör att system som är placerade på olika fysiska platser och använder olika programspråk, kan kommunicera med varandra. Den omvandlar meddelanden mellan tjänster som använder olika meddelandeprotokoll för att underlätta kommunikationen (Papazoglou & Van Den Heuvel, 2007).

Mönster är välbeprövade best practice metoder som kan användas för att designa och förverkliga de ovannämnda arkitekturerna med stöd av exempelvis en ESB. De användas även av integrationsutvecklare för att lösa vanligt förekommande problem och fungerar som referensmaterial för att föra kunskap vidare. (Hohpe & Woolf, 2004). Integrationsutvecklare ställs ofta inför problem inom integrationsprojekt som har blivit lösta tidigare och dokumenterats i form av mönster vilket gör att mönster framförallt främjar återanvändbarhet enligt Scheibler och Leymann (2008) och effektiviserar utvecklingsprocessen (Mangalaraj et al., 2014).

3 Undersökningsmetod

I det här kapitlet kommer metodvalet att presenteras för att besvara problemformuleringen. Vidare förklaras validiteten och reliabiliteten i själva valet. Dessutom avslutas kapitlet med kritik av metodvalet för att påvisa olika undersökningsmetoder, dess för- och nackdelar samt argumentation för vårt egna val.

3.1 Kvalitativ metod

Syftet med undersökningen var att granska integrationsutvecklares användning, behov samt åsikter av mönster i deras dagliga arbete. Eftersom olika källor har påpekat för- och nackdelar med mönster så ville vi få en nyanserad bild av mönsters betydelse vid integration. Tidigare forskning om mönster har utförts av Mangalaraj et al., (2014) som har granskat effektivitet och betydelse av mönster. Med den forskningen som underlag vill vi specifikt undersöka integrationsutvecklarnas uppfattning av mönster.

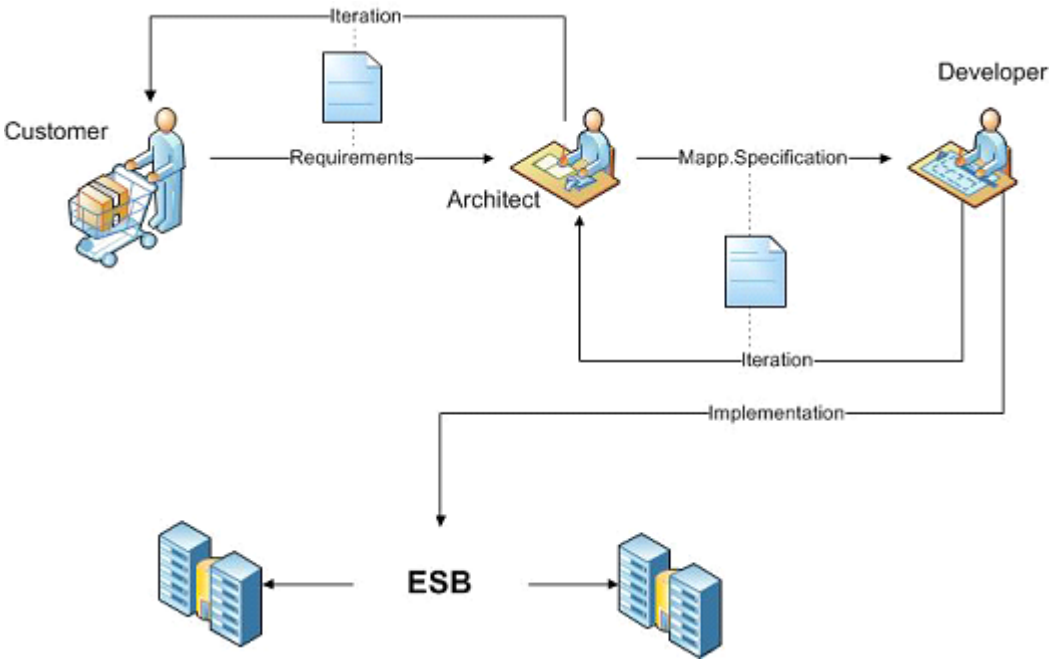
Efter problemformuleringen för området och informationsinsamlingen vid litteraturgenomgången fick vi fram ett antal frågor som vi ansåg bemötte forskningsfrågan. Vi insåg i och med storleken och avgränsningen på rapporten att en kvalitativ ansats till att besvara forskningsfrågan var optimalt. Jacobsen (2002) anser att en kvalitativ metod sätter få begränsningar på vad för svar en informant kan lämna. Därmed valde vi semistrukturerade intervjuer med fast ordningsföljd och tema för att ge informanten möjlighet till öppna svar. Detta förhållningssätt bidrog till en djupare informationsinsamling och enligt Jacobsen (2002) ger det även informantens unika förståelse av ett fenomen eller ett begrepp.

3.2 Val av respondenter och företag

Vi valde att inrikta oss på Enfo Zystems som är ett nischat företag inom integrationsbranschen, dels för att avgränsa antalet informanter och dels för att hitta gemensamma mönster som användes på arbetsplatsen och därigenom få information huruvida mönstren upplevs. Verktyget som främst användes av integrationsutvecklare på företaget var IBM WebSphere och Microsoft BizTalk som är ett integrationsverktyg mellan applikationer. Detta var verktyg som användes av alla utvecklare.

Genom tidigare kontakt med Enfo Zystems och vår närvaro på plats, har vi kunnat bilda oss en uppfattning för hur integrationsprocessen ser ut på företaget. Den börjar oftast med att en kund tar kontakt med företaget, i behov av en integrationslösning eller externa konsulter. Det blir sedan arkitektens uppgift att säkerställa vad kundens behov är. Detta leder till en iteration mellan kunden och arkitekten. Den här iterationen resulterar i att arkitekten kan utforma en mappningsspecifikation med tillhörande dokumentation, vilket är baserat på kundens krav. Mappningsspecifikationen beskriver för utvecklaren vad det är som ska utföras i

integrationen. Utvecklingen av integrationen sker i verktygen IBM Websphere och Microsoft BizTalk. Vilket verktyg som används beror på vilken kunskap som utvecklaren besitter. Utvecklaren har oftast fria tyglar för hur själva koden bakom integrationen ska se ut, det kan dock finnas riktlinjer för hur en specifik lösning, exempelvis för hur ett mönster ska implementeras och när det ska användas. Riktlinjerna är oftast på projektnivå eller i form av rena direktiv från projektets arkitekt.



Figur 3.3 Enfo Zystems integrationsprocess (Andersson et al., 2014).

Kriterier som vi hade var att sprida ut erfarenhetsnivån bland informanterna för att se hur seniora respektive juniora utvecklare upplevde mönstren, dess användning och funktion i deras dagliga arbete. Rollerna bland utvecklarna begränsades inte enbart till dessa positioner utan det fanns arkitekter, lead developers och vanliga integrationsutvecklare. Vissa individer hade flera roller och det var en faktor som togs till hänsyn till för att få så stor spridning som möjligt bland informanterna.

Intervjuformen och metodvalet gjorde att fem individer valdes ut för att erhålla fördjupad information. Roller och erfarenhet vid val av respondenter valdes ut baserat på dessa kriterier för att få en så bred plattform som möjligt samt kunna belysa om det finns bakomliggande faktorer som kan kopplas till roll och erfarenhet vid tolkning av mönster.

3.3 Intervjuguide

För att skapa en struktur för våra frågor och samtidigt säkerställa kopplingen till vår problemformulering och frågeställningen delade vi in frågorna i fem intervjuteman; (1) inställning till SOA vid integrationsutveckling, (2) erfarenhet av mönster, (3) applicering av mönster, (4) åsikter om mönster och kommunikation samt (5) resultat av mönsteranvändning

(Tabell 3.1). Varje tema framställdes utifrån respektive forskningsfråga och kategoriserades därefter. Varje intervjufråga kopplades sedan till de olika temana för att underlätta överskådligheten i förhållandet till forskningsfrågorna. Genom att tematisera intervjufrågorna kunde vi kontrollera att det fanns en stark koppling till vår problemformulering och frågeställning. Enligt Steinar Kvale (1997) bidrar det här även till att underlätta analysarbetet av intervjuerna.

Tabell 3.1 Tematisering av intervjufrågor.

Forskningsfrågor	Intervjutema	Intervjufrågor
På vilket sätt påverkar integrationsmönster en integrationsutvecklare vid systemintegration?	• Inställning till SOA vid integrationsutveckling	- När ni integrerar system, har ni i åtanke att uppnå en Service Oriented Architecture?
	• Erfarenhet av mönster	- Enligt källor är mönster en beprövad best-practice metod för att lösa ett vanligt förekommande problem. Vad är ett mönster för dig? - Vilka är de tre vanligaste mönstren som du använder? - Hur vet du vilka mönster som är mest lämpliga för integrationen?
	• Applicering av mönster	- Hur går du tillväga för att lösa ett integrationsproblem med, respektive utan ett tydligt mönster? - Om ett mönster hittas som har applicerats tidigare, tar ni hänsyn till i vilken kontext det har gjorts? Samt kontrollera om lösningen kan uppenbara problem i ett senare skede och passar det ert specifika problem?
Vilka för- och nackdelar finns det sett ur en integrationsutvecklares perspektiv?	• Åsikter om mönster och kommunikation	- Har du upplevt kommunikationsproblem med kollegor inom integrationsutveckling på grund av skild terminologi? - Vilka positiva och negativa aspekter ser du vid användningen av mönster? - Är mönstren du använder dina egenutvecklade, verksamhetens gemensamt uppsatta mönster eller är det kunddirektiv?
	• Resultat av mönsteranvändning	- Resulterar användingen av mönster i ett mer kvalitativt resultat eller komplicerar det utvecklingsprocessen?

Vi valde att semistrukturera intervjuerna med fast ordningsföljd och tema som vi nämnde tidigare. Det här resulterade i att vi tog fram en intervjuguide baserad på vår teoretiska bakgrund som bestod av tio huvudfrågor och tillhörande underfrågor som användes då respondenten inte täckte in allt i huvudfrågan. Till respektive huvudfråga formulerade vi ett syfte för att säkerställa att intervjufrågan ämnade besvara det vi hade i åtanke, vilket även

medförde att vi kunde kontrollera under intervjuprocessen om respondenten besvarade syftet med frågan. Utöver intervjufrågorna valde vi att inkludera; ålder, utbildning samt antal år inom integrationsbranschen för att kunna undersöka betydelsen av detta. Därtill informerade vi respondenterna att undersökningen kommer att förbli anonym och att deras namn skulle tas bort.

Intervjuerna som genomfördes hade följande struktur;
Ålder:
Utbildning:
Antal år inom integrationsbranschen:

1. **När ni integrerar system, har ni i åtanke att uppnå en Service Oriented Architecture?**

Syfte: Både Service-oriented architecture (SOA) och Enterprise Application Integration (EAI) kan använda sig av en ESB för att integrera applikationer. Vi vill veta om det finns riktlinjer för att uppnå SOA inom integrationsprojekt eller om utvecklaren enbart fokuserar på EAI utan någon SOA.

2. Enligt källor är ett mönster en beprövad best-practice metod för att lösa ett vanligt förekommande problem. Vad är ett mönster för dig?
– Har det hänt att du i efterhand upptäckt att du har använt ett mönster?

Syfte: Efter avgränsningen med antalet mönster ville vi även se om respondenten är medveten om vad mönster är och om de är medvetna att de använder dessa. Detta för att se hur erfarenhet inom branschen påverkar respondentens mönsteringenkänning. Underfrågan syftar till att undersöka om utvecklaren har reflekterat över en tidigare lösning.

3. Vilka är de tre vanligaste mönstren som du använder?
– Hur ofta använder du respektive mönster?

Syfte: Med andra frågan ville vi först och främst avgränsa antalet mönster respondenten ska förhålla sig till gentemot resten av frågorna utan att tydligt påverka objektets medvetenhet.

4. Hur vet du vilket mönster som är mest lämpligt för integrationen?

Syfte: Genom den här frågan vill vi undersöka om utvecklaren utvärderar och jämför möjliga mönster vid en integration.

5. Hur går du tillväga för att lösa ett integrationsproblem med, respektive utan ett tydligt mönster?
– Var tar du hjälp ifrån?
– Hur går du tillväga?

Syfte: Vi vill få fram information om var och hur respondenten hittar en lösning på ett problem utan en tydlig lösning. Därmed ville vi undersöka vilka verktyg som används,

litteratur, en databas av erkända lösningar eller Internet samt om spridning av mönster sker sinsemellan kollegor.

6. Om ett mönster hittas som har applicerats tidigare, tar ni hänsyn till i vilken kontext det har gjorts? Samt kontrollerar om lösningen kan uppenbara andra problem i ett senare skede och passar det ert specifika problem?

Syfte: Enligt litteraturen finns det oftast en struktur för ett mönster, som bland annat framhäver möjliga problem och konsekvenser. Vi vill ta reda på om utvecklaren exempelvis tar hänsyn till de olika faktorerna som kan påverka appliceringen av mönstret.

7. Har du upplevt kommunikationsproblem med kollegor inom integrationsutveckling på grund av skild terminologi?
– Anser du att mönster kan bidra till en gemensam terminologi på arbetsplatsen?
– Anser du att detta bidrar till en mer effektiv kommunikation mellan kollegor?

Syfte: Många artiklar och författare har påpekat att det krävs en enhetlig terminologi på arbetsplatsen för att alla ska definiera och uppfatta mönster på likartade sätt. Vi ville få svar på den här frågan och ifall det existerar, även se om det effektiviserar kommunikationen.

8. Är mönstren du använder dina egenutvecklade, verksamhetens gemensamt uppsatta mönster eller är det kunddirektiv?
– Främjar det återanvändbarhet och en effektivare utvecklingsprocess?

Syfte: Vi vill ta reda på om det finns direktiv i verksamheten eller vid enskilda projekt för användandet av mönster, samt om utvecklarna använder egenutvecklade mönster eller kundens. Vidare vill vi undersöka hur det påverkar utvecklarens arbetssätt i ett senare skede.

9. Vilka positiva och negativa aspekter ser du vid användningen av mönster?

Syfte: Frågan är till för att på ett mer öppet sätt ta reda på hur utvecklaren förhåller sig till mönster utan några givna direktiv.

10. Resulterar användningen av mönster i ett mer kvalitativt resultat eller komplicerar det utvecklingsprocessen?
– Om kvalitativt - på vilket sätt och varför?
– Om komplicerar - på vilket sätt och varför?

Syfte: Frågan ställdes för att särskilja på kvalitativt respektive kvantitativt (tidskrävande) resultat och huruvida integrationsutvecklaren tar ställning gentemot användandet av mönster. Underfrågorna ställdes för att förtydliga svaret och därigenom åsikterna.

3.4 Analys

För att kunna granska och strukturera datan som har tillhandahållits efter intervjuerna har Jacobsen (2002) delat in analysen i tre faser. De består av beskrivning, systematisering och kategorisering samt kombination. Beskrivning utgår ifrån att undersökaren ska få en så detaljerad beskrivning som möjligt av datan de eftersöker samt så ska undersökningen som utförs dokumenteras noggrant. Allt detta för att få en bra bild som möjligt fylld av detaljer och analyser som informanten lämnar. (Jacobsen, 2002).

Intervjuerna spelades in på två olika verktyg vilka var röstmemon på mobiltelefon som placerades framför respondenten och Microsoft Word på en bärbar dator för enligt Jacobsen (2002) finns det risk för tekniska problem som kan uppstå, exempelvis att ett verktyg slutar att fungera. Dessutom ville vi säkerställa att vi fick med allt informanten hade att säga utan att avbryta kontakten som uppstod mellan oss och intervjuobjektet. En annan säkerhetsdetalj vi införde under intervjuerna var att spela in varje fråga var för sig genom att starta en ny inspelning vid varje ny fråga. Detta för att minimera risken med en korrupt fil som skulle kunna påverka hela intervjun i efterhand. Vi utförde fem intervjuer som hölls inom ett relativt kort tidsspann. Intervjuerna transkriberades och sammanställdes strax därefter för att kunna se samband och skillnader i svaren och för att reflektera utan att påverka korttidsminnet för undersökarna.

Den andra delen för analys av datan består av systematisering och kategorisering. Från att tidigare samla in data så fokuseras det på de ämnen som vår frågeställning behandlar. För att kunna få en överblick över informationen och öka överskådligheten så kategoriseras bakgrunden till frågeställningen. Kategorisering belyser vilka teman som liknar varandra och vilka som skiljer sig åt (Tabell 3.1). Övriga syften med kategorisering är att organisera komplicerad data vilket visuellt förenklar datan, och med hjälp av förenklingen, kunna jämföra texterna och informationen. (Jacobsen, 2002).

I den slutgiltiga kombinationsfasen undersöks den insamlade datan med kategoriseringen och litteraturen för att hitta likheter och skillnader enligt Jacobsen (2002). För att uppnå detta så sammanfattade vi intervjuerna och skapade tabeller (se bilaga 8) för att se likheter och skillnader i svaren för att lättare kunna förstå informanterna och deras uppfattning om frågeställningen.

3.5 Undersökningskvalité

Kvaliteten på den insamlade datan kan enligt Jacobsen (2002) mätas i validitet (giltighet) och reliabilitet (tillförlitlighet). Validiten i forskningsansatsen kan definieras på två sätt. Det första är att kontrollera den insamlade datan gentemot annan forskning eller teori. Den andra metoden syftar till att kritiskt granska den insamlade datan som informanter har lämnat (Jacobsen, 2002).

För att uppnå giltighet har båda undersökarna varit med på intervjun och gått igenom inspelningarna från intervjuerna och sammanfattat dessa samt fått transkriberingarna godkända av intervjuobjekten för att öka validiteten. Jacobsen (2002) påpekar att en kritisk genomgång av källorna måste utföras för att kontrollera att de ger rätt information. Han förklarar att källans närhet till ett fenomen är ytterst viktigt för att skillnaderna mellan en förstahandskälla och andrahandskälla kan vara stor. Författaren belyser även att källans kunskapsnivå om ett ämne bör vara relevant till undersökningen för att en mer erfaren källa inom ett ämne eller fenomen kan ge mer korrekt information än en oerfaren. Till sist belyser han att källans vilja att förmedla informationen (Jacobsen, 2002). Vi tycker att vi har uppnått detta genom att endast avgränsa oss till integrationsutvecklare som arbetar med EAI-mönster i deras dagliga arbete och uteslutit övriga delar av den operativa verksamheten som exempelvis support.

Dessa kriterier anser vi är uppnådda för att ett av kriterierna med vår undersökning var tillgången till förstahandskällor som arbetar med dessa mönster dagligen. Intervjuerna skedde på arbetsplatsen för intervjuobjektet eller via telefon där objektet befann sig i en trygg miljö och fick bestämma intervjutid och datum. Detta anser vi ökar validiteten eftersom att intervjuerna utfördes inom ramarna för informantens önskemål. En annan aspekt för validiteten som togs i beaktande var erfarenhetsnivån bland informanterna. Vi har medvetet valt att sprida intervjuerna bland informanter med olika erfarenheter för att få deras individuella tolkning och användning av mönster. Vi anser att detta ger mer värde till undersökningen i form av bredd och vilken roll erfarenhet spelar i användningen av mönster.

Reliabilitet syftar till om undersökningens tillförlitlighet. Jacobsen (2002) delar in reliabiliteten i två delar, undersökareffekten och kontexteffekten samt förklarar att undersökningens upplägg påverkar informanterna i form av stimuli och signaler.

Undersökareffekten påverkar intervjuobjektet oavsett miljö. Det syftar till hur undersökaren beter sig, klär sig eller ser ut och så vidare (Jacobsen, 2002). Därmed försökte vi vara neutrala i vårt förhållningssätt gentemot intervjuobjektet och inte vara för framfusiga eller oengagerade. Vi lät informanten tala oavbrutet men ställde följdfrågor för att reda ut oklarheter. Jacobsen (2002) betonar dock att undersökningseffekten är omöjlig att fullständigt kontrollera men vi har ändå försökt minimera riskerna med att påverka informanten.

Kontexteffekten syftar till sammanhanget som informationen samlas in. En onaturlig miljö för undersökningsobjektet kan vara främmande och göra objektet obekväm. Det resulterar inte i en fullständig, informativ intervju. Därmed föredrar intervjuobjekten en trygg miljö (Jacobsen, 2002). För att minimera den här effekten utfördes intervjuerna på respektive intervjuobjekts arbetsplats.

3.6 Kritik av metodval

De öppna, individuella, semistrukturerade intervjuerna passar bäst när undersökaren och informanten kan träffas ansikte mot ansikte. Informanten upplever det som lättare att prata om känsliga ämnen när en fysisk kontakt upprättas. En annan fördel är till skillnad mot telefonintervju, att informanten inte har lika stor benägenhet att besvara frågorna snabbt för att behaga undersökaren. Dock, kan telefonintervjuer ge stora fördelar i form av undersökareffekten som tidigare nämnt (3.5 Kvalitet på metodval) minskar. (Jacobsen, 2002). Eftersom intervjun innehöll många öppna frågor så valdes individuella intervjuer för att minska nackdelarna med snabba svar. Dock kunde alla intervjuer inte utföras ansikte mot ansikte med intervjuobjektet på grund av brist på resurser och geografiska olikheter. Två av de fem intervjuerna genomfördes över telefon medan de övriga tre var av fysisk karaktär där vi mötte upp intervjuobjekten på deras kontor. Detta kan ha påverkat undersökareffekten både positivt och negativt men att sprida ut intervjuerna på både telefon och fysiska möten tror vi, gav oss en bredare grund. Därmed gav intervjuerna både de positiva och negativa aspekterna vid både fysiska möten och telefonintervjuer.

4 Empiri

Det här kapitlet påbörjas med en presentation av intervjuobjekten och respektives svar. Därefter presenteras resultaten av den kvalitativa undersökningen - kategoriserat efter intervjuns olika teman.

Som grund för undersökningen valdes Enfo Zystems, ett nischat företag inom integrationslösningar. Enfo har fokuserat på integrationslösningar inom affärsprocesser och system för tjänsteorienterad (SOA) arkitektur sedan starten 2001. Företaget består av cirka 150 anställda och deras samarbetsparterns består huvudsakligen av Microsoft och IBM - vilket gör att deras verktyg används av integrationsutvecklarna på Enfo Zystems. Verktygen för integration som utvecklarna använder i sitt dagliga arbete består av IBM Websphere Message Broker och Microsoft BizTalk.

De utvecklare vi har intervjuat på företaget består av fem stycken till antalet och alla är av manligt kön. Alla har olika roller och erfarenhet inom integration samt sitter med utveckling varje dag.

Tabell 4.1 Presentation av respondenter

Respondent	Intervju-form	Tid	Plats
F1: är 24 år gammal och har en kandidatexamen i Systemvetenskap ifrån Lunds Universitet. Han har 1,5 års erfarenhet av integrationsbranschen och arbetar endast som utvecklare.	Möte	32 min	Enfo Zystems kontor
F2: är 30 år gammal och har en kandidatexamen i Design av organisations- och verksamhetsstödjande system från Lunds Universitet. Han har 6 års erfarenhet från integrationsbranschen och jobbar både som arkitekt och utvecklare.	Möte	34 min	Enfo Zystems kontor
F3: är 32 år gammal och har en Magister i Datavetenskap. Han har 8 års erfarenhet från integrationsbranschen och jobbar både som arkitekt och utvecklare.	Telefon	20 min	-
F4: är 28 år gammal och har en kandidatexamen i Systemvetenskap från Lunds Universitet och en Master från Stockholms Universitet. Han har 3 års erfarenhet inom integrationsbranschen och jobbar både som arkitekt och utvecklare.	Möte	30 min	Enfo Zystems kontor
F5: är 39 år gammal och har en kandidatexamen i Systemvetenskap från Lunds Universitet. Han har 10 års erfarenhet från integrationsbranschen och jobbar både som arkitekt och utvecklare.	Telefon	37 min	-

4.1 Inställning till SOA vid integrationsutveckling

Som tidigare nämnt (i kapitel 2.2) står SOA för Service Oriented Architecture som erbjuder en tjänsteorienterad arkitektur. Det är inte en teknologi, utan en konceptuell arkitektur som är i behov av en underliggande teknologi för att kunna implementera en tjänsteorienterad arkitektur. SOA hjälper verksamheter att uppnå en arkitektur som är flexibel och kan förändras efter omständigheterna. Mer specifikt hjälper SOA, organisationer att utveckla tjänster och komponenter som är löst kopplade. Det syftar till återanvändbarhet och kombinering oavsett plattform för att skapa en anpassningsbar arkitektur som i grunden aldrig ska påverka användarnas upplevelse.

De olika utvecklarna på Enfo Zystems har alla olika erfarenheter och små skillnader i sitt tänk när det gäller SOA vid en integration. F1 som är relativt ny på området uppfattar SOA som ett ansvar som ligger på arkitekten i projekten och därmed inte inkluderas i hans arbetsuppgifter. Exempelvis får han idéer från arkitekten om vad som kan återanvändas i en tidigare integration eller få två integrationer att slås ihop för att därmed minimera merarbete och få till SOA-tänket i sitt arbete.

> "Jag har ju gjort en integration och sen så beställer kunden en ny integration då kan ju min arkitekt till exempel säga att: men den här nya integrationen skulle kunna integreras i den här gamla..." (F1, Bilaga 3, s. 44, Rad 13)

F2 och F4 har däremot mer erfarenhet än F1 och de anser att ett SOA-tänk inte är prioriterat men utvecklarna försöker att ha det i baktanke när de integrerar system. När en ny integration genomförs försöker de att gå på djupet och granska integrationen kritiskt för att kontrollera vilka moduler och komponenter som kan skapas som återanvändbara tjänster för att utbringa mer värde i kundernas projekt.

> "Ska vi göra detta som vanlig integration eller ska vi försöka tjänstefiera vissa delar? Så man har det i baktanke men inte i åtanke. Man försöker inte uppnå SOA, utan kan jag på något sätt tjänstefiera detta på ett bra sätt?" (F2, Bilaga 4, s. 52, Rad 6)

F4 tänker i samma banor och försöker att analysera vilka delar som kan återanvändas för att kunna erbjuda kunden, nytta. Däremot är alla respondenter eniga om att tjänstefiera en hel integration är svårt för att det kräver alldeles för mycket tid och det är inte alldeles säkert att kunden kommer att nyttja alla tjänster som skapas. F3 förtydligar detta tänk som alla respondenter är eniga om.

> "Det är ingenting vi sysslar med för det krävs mycket för att uppnå en sådan arkitektur." (F3, Bilaga 5, s. 58, Rad 2)

4.2 Erfarenhet av mönster

Tidigare (i kapitel 2.5.1) beskrevs definitionen av mönster. Det är en beprövad metod för att lösa ett vanligt förekommande designproblem inom mjukvaruutveckling. Det är inte en definitiv lösning i form av kod utan mer ett abstrakt tillvägagångssätt för utvecklaren att implementera en lösning på bästa möjliga vis. För ett problem kan innefatta många lösningar med mönster, men det är upp till utvecklaren att komma fram till den mest optimala lösningen ur ett resursperspektiv.

Respondenterna hade olika definitioner av mönster men svaren var snarlika i syftet. F1, F3 och F4 anser att de är generella nog för att kunna användas och beskriva olika sorters problem med en övergriplig lösning.

> "Jag skulle säga att ett mönster är en konceptuell beskrivning av hur ett problem kan lösas, alltså inte i detaljnivå, hur man löser problemet." (F4, Bilaga 6, s. 64, Rad 11)

Intervjuperson F5 tycker att det är lite svårare att definiera mönster och anser att det är en idé om hur integrationsutvecklaren ska lösa ett problem. I ett eget projekt med flera olika sorters filförflyttningar har F5 tillsammans med övriga inblandande konsulter komprimerat de olika sorterna till ett flöde. Därmed beskriver han att flera olika sätt att flytta filer har konverterats till ett enkelt sätt och ökat återanvändbarheten. Respondenten tycker slutligen att mönster kan definieras på djupare teknisk nivå och inte övergripligt.

Upptäckten av integrationsmönster i efterhand visar en gemensam nämnare i form av erfarenhet. F1 som har minst erfarenhet av integration har upptäckt vissa mönster i efterhand. Han påpekar att det är något som han använder utan att vara medveten om det. Efter en viss tid på arbetsplatsen har han tagit till sig mönster och fått en bild av vad det är. F1 förklarar även att lösningar hade blivit mer komplexa utan mönster, för att han hade kunnat lösa de men inte alls med samma standardiserade sätt som mönster bidrar med. F4 bidrar även med samma åsikter till undersökningen. Han anser att i början av sin karriär hade han svårt att särskilja vad som var mönster och vad som inte var det. Han implementerade bara olika integrationslösningar och fann i efterhand, när han reflekterade, att de lösningarna kan definieras som mönster.

F4 förklarar ytterligare att idag, hade han gjort annorlunda med kunskapen som han besitter för att utveckla en optimal lösning. Däremot tycker F3 att det är väldigt svårt att använda mönster utan att utvecklaren är medveten om det. Han utvecklar resonemanget och påpekar att det är sällan sådant upptäcks i efterhand och att han i sådana fall går tillbaka till tidigare fall med liknande problem där lösningen kan återanvändas. F5 har samma åsikter och förklarar att det är svårt att inte vara medveten om att han använder mönster i sitt arbete. Han utvecklar även argumenten och påpekar att det har med erfarenhet att göra.

> "Jag tror mer att det är något som kommer med erfarenheten och det är det jag gillar med att arbeta på Zystems till exempel, att det finns så himla många kollegor som är specialiserade på samma sak. De det finns alltid någon som har löst ett visst problem, då behöver man inte uppfinna hjulet igen." (F5, Bilaga 7, s. 73, Rad 62)

På frågan om vilka mönster som används oftast på arbetsplatsen besvarar alla respondenter på liknande vis men har olika uppfattningar om vad för information de olika typerna av mönster bidrar med. F1 och F2 har precis samma åsikter om vilka integrationsmönster som används mest.

" Fire-and-Forget, Request-Reply och Publish-Subscribe om vi pratar om integrationsmönster." (F2, Bilaga 4, s. 53, Rad 28)

Medan F4 och F5 kommer i kontakt med nästan exakt samma mönster i deras dagliga arbete förutom små skillnader. Båda utvecklarna har som en gemensam katalysator att de använder Request-Reply och Point-to-Point. Däremot använder F4 Publish-Subscribe mönstret och F5 använder Fire-and-Forget. Detta visar att alla integrationsutvecklare använder samma mönster i stora drag men att det finns skillnader mellan olika integrationsutvecklarna och deras basmönster de använder.

F3:s tankesätt skiljer sig stort från de övriga i både användandet av mönster och uppfattningen om vad som kan definieras som mönster. Han använder sig av mönstret Routing vilket baseras på hans arbetsuppgifter. F3 jobbar inte inom produktutveckling och därmed anser han att de mönster som övriga kollegor använder på arbetsplatsen inte kan definieras som mönster för de är alldeles för generella.

"Fire-and-Forget och Publish-Subscribe säger bara vilken typ av mönster men jag skulle nog inte kalla de för mönster utan snarare vilken typ av integration det är." (F3, Bilaga 5, s. 59, Rad 31)

Nästan alla utvecklare anser att en kombination av erfarenhet och inläsning av kraven visar vilken typ av mönster som passar bäst för en specifik integration. Alla är eniga om att utifrån mappningsspecifikationen kan de se vilket mönster som bör fungera i integrationen men ibland kan det uppstå oklarheter. De löser detta oftast genom att fråga kunden direkt om de har en arkitekt- och utvecklarroll men om de endast har en utvecklarroll i projektet går de till arkitekten för att klargöra förvirringar för att kunna hitta ett lämpligt mönster som kan användas vid integrationen.

"...bygger det väl på att du läser igenom, vad är det man försöker uppnå? Sen försöker förstå. Oftast är det en slags diskussion där behoven växer fram och utifrån behoven så tänker du väl egentligen. För du har ett slags kunskapsbas om ungefär vad EAI-patterns är." (F2, Bilaga 4, s. 53, Rad 49)

"Men generellt så baserar det sig på erfarenhet, om man upptäcker en situation och de beskriver vad de vill lösa, så säger man; har ni tänkt på detta, behöver ni inte veta när meddelandet har gått fram eller behöver ni aldrig veta det, är det kritiskt, tidskritiskt..." (F5, Bilaga 7, s. 75, Rad 119)

Det här understryker tillvägagångssättet vid val av ett mönster som passar integrationen men det finns andra sätt. Enligt F4 kan han även vid vissa fall bygga upp lösningen utan en klar vision om vilket mönster som appliceras för att efteråt förstå att implementationen av integrationen har bildat ett mönster.

"...ibland kan det hända ja, jag ska göra såhär och det blev visst ett sådant mönster." (F4, Bilaga 6, s. 66, Rad 76)

Andra gånger kan utvecklaren även utgå ifrån kraven, både de tekniska- och affärskraven som kunden har. F3 anser att det inte finns ett mönster som är mest lämpligt utan det handlar om vilket mönster som är rätt och vilka andra som blir felaktig i sammanhanget, baserat på integrationskraven och affärskraven.

4.3 Applicering av mönster

Som tidigare nämnt (i kapitel 1.2) finns det en del utmaningar vid integrationsutveckling och (i kapitel 2.5.2) nämner vi att designmönster inte endast bidrar till fördelar per automatik utan att det finns svårigheter med att implementera och använda de inom en organisation. Vi förklarar även (i kapitel 2.5.1 och 2.5.3) att mönster ofta har en struktur, där bland annat kontext, problem och konsekvenser definieras.

Samtliga respondenter använder snarlika tillvägagångssätt för att lösa ett integrationsproblem utan ett tydligt mönster. F1 som knappt arbetat inom integrationsbranschen i ett år och därmed har mindre erfarenhet än övriga respondenter, tar mycket hjälp av omgivningen för att lösa ett problem. Han nämner att han ofta tar hjälp ifrån kollegor, projektets arkitekt, IBM:s hjälpsida, Internet och poängterar att det är upp till honom själv att hitta den informationen han behöver för att lösa problemet.

"Jag arbetar med Websphere Message Broker, och jag vet ju vad jag har att arbeta med, jag har ju mina noder i min palett och sen får man snickare ihop något med hjälpa av det." (F1, Bilaga 3, s. 47, Rad 98)

Även F4 som endast har arbetat tre år med integration tar mycket hjälp från Internet, men nämner att han först och främst analyserar och funderar. Han för ofta en diskussion med kunden för att säkerställa kundens behov och vad det är de efterfrågar för att kunna utveckla en integration. Utöver det här utför han tester och experimenterar.

F2 som har 6 års erfarenhet av integrationsbranschen anser att;

"När det har kommit så långt att utvecklaren får det så är det i princip alltid ett mönster. " (F2, Bilaga 4, s. 54, Rad 59)

F2 vidareförklarar att han går tillväga på olika sätt för hitta och lösa ett integrationsproblem i beroende av om han får en specifikation som säger något eller om det är otydligt. Är det otydligt får han gå tillbaka till beställaren likt F4, för att säkerställa vad det är som efterfrågas. Han kan sedan baserat på sin erfarenhet tyda vad det är för typ av integration, finns inget tydligt mönster för integrationen tittar han på gamla lösningar. Han poängterar även att utan ett mönster kan det bli svårt att lösa ett integrationsproblem.

F3 likt F2 börjar med att titta om det finns färdiga, liknande eller tidigare lösningar. Har han löst det tidigare så försöker han återanvända det. Han anser även att det är väldigt sällan han stöter på något som är helt nytt, han försöker alltid hitta något som liknar integrationen.

F5 testar sig fram likt F4, om han inte har ett tydligt mönster. Han upplever att det ofta inte finns ett tydligt mönster eftersom att kunden inte har kraven klara för sig och kan därmed inte förmedla all nödvändig information.

"Så det är ju svårt och det går ju inte att ställa några frågor till kollegor i det fallet för jag vet inte ens vad jag ska fråga om, för att jag inte känner till problemet som jag ska lösa." (F5, Bilaga 7, s. 76, Rad 146)

F5 nämner att om det finns ett tydligt mönster så blir allt enklare att kommunicera kring integrationen och han kan då lättare ta hjälp av kollegor. F1 föredrar också om det finns ett tydligt mönster och tillägger att han får en guide att utgå ifrån, han anser att det är svårare att själv komma på en lösning och ifrågasätter ofta om det är den mest optimala. F1 lyfter fram att han alltid vill leverera en så bra lösning som möjligt och eftersom mönster är beprövade och en best-practice så behöver han inte tveka över kvalitén på lösningen. F4 anser att tillvägagångssättet inte skiljer särskilt mycket från användingen utan mönster, intervjuobjektet påpekar dock att med ett mönster så får han en grund att utgå ifrån fortare vilket underlättar förståelsen för vad som ska integreras och kommunikationen med intressenterna

På frågan om ett mönster hittas som har applicerats tidigare, tar ni hänsyn till i vilken kontext det har gjorts, samt kontrollerar om lösningen kan uppenbara andra problem i ett senare skede och passar det ert specifika problem? Berättar samtliga respondenter att de tar hänsyn till det här, dock lyfter en av respondenterna fram att det är en ansvarsfråga beroende på vilken roll som man besitter i projektet.

F1 anger att han alltid tar hänsyn till kontexten för att han vill att det ska passa så bra som möjligt. Respondenten berättar även att han hade tänkt på hur lösningen hade fungerat i framtiden och därför tagit hänsyn till vilka problem lösningen kan medföra. F2 har liknande ståndpunkt i den här frågan, han tänker dock även på hur implementeringen av mönstret kan användas i framtiden och hur integrationen kan komma att ändras. På grund av erfarenhet kan han i vissa fall förutspå vad som kan tänkas tillkomma i integrationen och väljer mönster därefter. Han försöker därmed alltid framtidssäkra integrationen. Dock har han även i åtanke hur integrationen kommer påverka en ny utvecklare i projektet eller supporten.

"Är det här klockrent för dem? Eller blir det för komplext?" (F2, Bilaga 4, s. 55, Rad 114)

F3 uppger att han mest tar hänsyn till kontexten kring vilken typ av information det är och poängterar att han syftar på mönster som används i hans nuvarande projekt och inte exempelvis mönster som Fire-and-Forget och Publish-Subscribe. Respondenten berättar att han beaktar de problem som ett mönster kan medföra, han nämner att innan någon kodning utförs i ett projekt så förs en diskussion angående vilken lösning som ska användas.

"...vad är det för risker, vad finns det för problem? Kan vi ens använda den? Kan vi inte använda den?" (F3,

Bilaga 5, s. 60, Rad 64)

F4 anser att det beror på vilket ansvar han har i projektet. Han tycker det är arkitektens ansvar att ta hänsyn till kontexten och anser att det skulle ta alldeles för mycket tid om han skulle ta hänsyn till det utöver sina uppgifter som utvecklare. Intervjuobjektet tar dock alltid hänsyn till om det kan orsaka problem vid integration och när han ansvarar för arkitekturen finns alltid en åtanke på kontexten. F5 anser att han alltid måste ta hänsyn till kontexten eftersom den bestämmer vilket mönster som ska användas. Han tar även hänsyn till konsekvenser och problem som kan uppstå i ett senare skede. Respondenten anser att konfigurerbarhet och skalbarhet finns alltid i åtanke vid utformning av arkitekturen.

4.4 Åsikter om mönster och kommunikation

Gregor Hohpe (2003) och Gamma et al., (1994) poängterar att kommunikation och en icke-enhetlig terminologi på arbetsplatsen är en risk som kan orsaka problem vid implementeringen av en integration. Mangalaraj et al., (2014) och flertalet andra källor beskriver vilka för- och nackdelar mönster kan ha vid integration och detta har lett till en kategorisering av undersökningen till åsikter om mönster och kommunikation som utvecklarna har.

Huruvida kommunikation och en enhetlig terminologi påverkar arbetet är alla respondenter eniga om att en enhetlig terminologi underlättar arbetet men det finns ändå stora skillnader i hur intervjuobjekten uppfattar kommunikationen i verkligheten. F2 och F4 anser att det finns olika termer för olika verktyg som används och detta skapar en förvirring på arbetsplatsen även om alla kollegor arbetar med samma sak.

"Pratar man med BizTalk-folk så heter ju Request-Reply – Request-Response vilket är väldigt irriterande" (F2, Bilaga 4, s. 56, Rad 128)

F1 är verksam inom ett projekt just nu och förklarar att det inte kommuniceras med mönster inom det. Det skapar svårigheter för honom och han anser att mycket slangord och förkortningar används istället för facktermer. En stor skillnad mot de övriga när det gäller kommunikationsproblem står F5 för.

"...bland de som är duktiga så kan jag dra mig för att skicka ett mejl för att jag tycker inte att jag kan förklara mitt problem eftersom att jag inte kan teorin bakom tillräckligt /…/ Det kan jag tycka är ett kommunikationsproblem, att jag ens drar mig för att kommunicera kring det" (F5, Bilaga 7, s. 78, Rad 202)

Han är dock erfaren och har löst många problem tidigare och vet oftast förfarandet för att lösa problemen men har inte teoritiserat kring termer och hur kommunikationen kan ske på enklast möjliga vis. F5 tycker inte att organisationen och kollegorna har diskuterat mönster i större utsträckning. De best practice metoder verksamheten har tagit fram har inte tryckt på just mönster utan bara snuddat vid de enligt honom. En gemensam struktur kan ses för svaren på frågan. De respondenter som har arkitekt- och utvecklarroll tycker inte att de huvudsakliga kommunikationsproblemen uppstår på den övre mönsternivån, utan mer på den djupare,

tekniska nivån.

"...hur man ska lösa det på mer detaljnivå, då försvinner den här fördelen med en gemensam terminologi för då kan man inte riktigt prata i mönster längre utan då måste man prata på en annan nivå." (F4, Bilaga 6, s. 68, Rad 143)

På frågan om vilka aspekter om mönster som är positiva respektive negativa finns det stora likheter. Alla integrationsutvecklare tycker att det effektiviserar kommunikationen, utvecklingsprocessen, återanvändbarhet och sparar framförallt in tid. Rent konkret tycker de att det blir ett mer standardiserat arbetssätt som ger underlag för att arbeta på samma sätt utan att projekten blir svårhanterbara i form av olika lösningar för olika utvecklare om projektet har haft en stor omsättning av konsulter.

"...riskerar det annars att bli spretigt eller i mitt fall med den kunden jag har nu som har bytt resurser ganska ofta från vår sida, då har det inte blivit så mycket kontinuitet, det har istället blivit ganska spretiga lösningar. Det hade man kunnat undvika definitivt." (F5, Bilaga 7, s. 79, Rad 258)

Ett annat konkret exempel på effektivisering av utvecklingsprocessen, berättar F1 mer om. Han skildrar hur de har skapat ett skript i ett av verktygen de använder på arbetsplatsen. Med hjälp av en knapptryckning i verktyget kan ett skelett skapas för en integration till ett enkelt mönster. Det effektiviserar utvecklingsprocessen enligt F1, och föreslår att om det kunde göras för större mönster eller mer avancerade integrationer så kunde det underlätta för utvecklare med mindre erfarenhet.

De negativa aspekterna som integrationsutvecklarna är överens om behandlar överanvändning av mönster. Samtliga respondenter beskriver att ibland kan mönster användas bara för att de finns och att detta inte gynnar projektet. Det blir för komplicerat och spretigt eftersom att det inte finns uppsatta mönster som utvecklarna inom projektet ska förhålla sig till.

"Negativa aspekterna är kanske att man kanske försöker skapa mönster där det inte finns mönster utan man skapar det bara för sakens skull. För man använder en mönsterfälla. Allt ska handla om mönster så man måste liksom använda sunt förnuft" (F3, Bilaga 5, s. 62, Rad 115)

I sådana fall anser F1 att en speciallösning är bättre. Det gynnar den specifika integrationen mer eftersom att istället för att leta efter mönster i flera dagar kan en specifik lösning tas fram som är specialtillverkad för en integration. Han anser att på så sätt kan både tid och pengar sparas in för kunderna. F2 utvecklar den här tanken och förklarar att det är viktigt med tydlighet när det gäller vilka mönster som löser vilka problem. Han förklarar att de har stött på olika mönster som har använts inom projekt för att lösa problemen men att mönstret inte tillät skalbarhet. När systemet och meddelandena som skickades mellan applikationerna mångdubblades var de tvungna att byta ut det existerande mönstret mot ett mer simpelt men det gav resultat. Därmed påpekar han att den mest komplicerade lösningen inte alltid är den bästa och att det återigen kräver tydlighet när det gäller att förklara vilka mönster som löser vilka problem.

För den sista frågan inom intervjutemat huruvida utvecklarna använder egenutvecklade

mönster, verksamhetens gemensamt uppsatta eller kunddirektiv samt om det främjar återanvändbarhet, så är tre av fem intervjuobjekt överens om att egenutvecklade används väldigt ofta inom projekt. Ofta kommer kunderna till Enfo Zystems och köper en tjänst för att integrera de olika system och applikationer som finns hos dem. Detta gör att integrationsutvecklarna får fria tyglar för att implementera integrationen på det sättet de tycker är optimalast. För att det finns många olika sätt att lösa problem med hjälp av många olika mönster men framgångsfaktorn ligger i hur pass effektiv den specifika lösningen är. De får inte vara för komplexa så det blir svårt för andra utvecklare att förstå men inte heller för lätt för att det kan påverka den framtida skalbarheten. De tre intervjuobjekt som är eniga om egenutvecklade mönster berättar att de jobbar inom team på specifika projekt. Det ger de frihet att sätta upp gemensamma mönster inom projektet utan att några utomstående kan påverka. En viss styrning finns dock på projektnivå. F4 förklarar att på det projekt som han sitter och utvecklar på så finns det utvecklarriktlinjer för hur de ska gå tillväga.

"...BizTalk-projektet som jag håller på med, där har vi utvecklarriktlinjer som beskriver implementationen av olika mönster." (F4, Bilaga 6, s. 69, Rad 164)

Intervjuperson F3 utvecklar detta ytterligare och berättar att det dels kan vara egenutvecklade och dels kunddirektiv. Han förespråkar att vissa mönster måste användas beroende på kundens önskemål och anser att det är bra om kunden har några grundläggande mönster som kan användas. Därutöver är det en fördel om kunden är flexibel och tillåter att egenutvecklade mönster kan läggas till för mönster måste kunde utvecklas för att hänga med i den teknologiska utvecklingen enligt F3.

Intervjuperson F2 har dock en stor skillnad i åsikter gentemot de övriga respondenterna när det gäller utvecklingen av mönster. Han anser att Enfo Zystems har satt upp tre mönster som är övergripliga för hela verksamheten och som ska användas. Han utvecklar detta och förklarar att tidigare har de varit 20 utvecklare på företaget men numera är de färre. När de väl återigen tar in nya utvecklare har de förstått att det blir alldeles för svårt för de att sätta sig in i arbetssättet hos Enfo Zystems och detta har resulterat i tre övergripliga mönster som har satts för hela verksamheten.

"Det innebär egentligen att du bara har strömlinjeformat din utveckling till att bli mer generisk så att när utvecklare nummer sex sen kommer, så kan han titta på en integration och fatta att det här är en Request-Reply för att vi har de här två artefakterna." (F2, Bilaga 4, s. 56, Rad 158)

På underfrågan om detta arbetssätt främjar återanvändbarhet och effektiviserar arbetsprocessen är alla utvecklare överens om definitionen. De motiverar detta genom att mönstren ger de en gemensam terminologi på arbetsplatsen och ger även vägledning för att hitta tidigare lösningar som kan appliceras. De påpekar även att det ger en grund för att återanvända mönster och dess koncept. F2 understryker att detta inte främjar SOA-tänket med återanvändning av tjänster, utan mer konceptet för mönster som exempelvis skelettet för integrationen som finns i verktyget de använder på arbetsplatsen. Som tidigare nämnt kan de generera ett skelett för en integration som visar upp kopplingarna mellan applikationerna, endast genom en knapptryckning.

"Vi har mer återanvändbarheten i utvecklingsdelen. Alltså har vi ett Request-Reply scenario så har vi ett färdigt skelett för att just producera en Request-Reply integration. Så på det sättet har vi ju återanvänt konceptet men vi har inte återanvändbarhet i själva tjänsterna." (F2, Bilaga 4, s. 57, Rad 169)

4.5 Resultat av mönsteranvändning

Enligt Mangalaraj et al.(2014) finns det flera positiva aspekter med att använda designmönster när en individ ställs inför ett problem. Deras studie visade att användandet resulterade i att hastigheten ökade och att resultat blev av högre kvalité.

Intervjuperson F2 berättar att han tycker att det inte komplicerar utvecklingsprocessen utan snarare ger ett bättre resultat. Han poängterar dock att det är viktigt att tänka på vad det är de vill uppnå med integrationen och vad det är för problem som ska lösas. Han ser fördelen i att kunna plocka ut ett mönster baserat på vilket problem som ska lösas.

"Oftast om du sitter i kammaren och ritar på lösningar så är det ju lätt att komma på coola, komplexa lösningar. Men jag tycker det är viktigt att hela tiden titta på vad är det vi försöker uppnå? Vilket problem är det vi vill lösa? Om du har en mönsterbas så är det ju väldigt lätt att läsa igenom ditt problem, titta på dina mönster och tänka 'Den här, den fungerar klockrent'. " (F2, Bilaga 4, s. 58, Rad 200)

F1 anser att mönsteranvändning besparar tid och poängterar att tid är pengar. Rent kvalitetsmässigt förlitar han sig på att det är en best practice och anser att användandet är lite som att kvalitetssäkra lösningen. Han berättar att överanvändandet av mönster kan komplicera utvecklingsprocessen. Intervjuperson F3 anser att första gången han skapar ett mönster så bidrar det till ett kvalitativt resultat men komplicerar utvecklingsprocessen.

"Men andra, tredje och fjärde gången så är förhoppningsvis den komplexiteten i utvecklingsprocessen mindre och det kvalitativa resultaten ska förhoppningsvis vara densamma. " (F3, Bilaga 5, s. 62, Rad 125)

F4 berättar att det är kvalitativt för att det finns en riktlinje för hur mönstret ska implementeras vilket bidrar till att det görs likadant genom hela integrationen. Intervjupersonen anser att användandet av samma mönster i en integration kan bidra till att det går att se hur den kommer se ut i ett senare skede. Det här bidrar även till att integrationen blir lättare rent underhållsmässigt. Intervjupersonen berättar även om hans nuvarande projekt, där de använder mönster för tidsuppskattning och förklarar att de har en lista med mönster och respektive komplexitetsgrad, vilket gör att de kan estimera tiden för implementering av ett visst mönster. Han anser att det här en positivt men medger att han tycker det är farligt att bryta ner integrationen i små beståndsdelar för att man tappar känslan för integrationen i sig. Han poängterar också att estimeringen av en hel integration kan skilja sig avsevärt från den totala estimeringen av mindre beståndsdelar.

Intervjuobjektet F5 håller med om att det ökar kvalitén och underlättar underhållningen av integrationen. Han lyfter fram att användandet av mönster gör så att utvecklare inte behöver återuppfinna hjulet vilket reducerar potentiella fallgropar. Respondenten anser även att när

han använder mönster som begrepp för att beskriva en integration så underlättas förståelsen för beställaren, vilket skapar en trygghet för båda parter. Komplexitet kan uppstå om en utvecklare försöker applicera ett mönster på varenda litet scenario och motiverar valet genom att han använder sin erfarenhet för att bedöma när det är lämpligt.

4.6 Sammanfattning av Empiri

Intervjuerna har kategoriserats i fem olika teman och underkategorier med tillhörande underfrågor till varje tema (se Tabell 3.1). Det ställdes tio frågor totalt till respondenterna och inställningen till SOA var respondenterna eniga om att det inte var något som de sysslade med eftersom det krävdes för mycket för att uppnå en sådan arkitektur.

Beroende på integrationsutvecklarens erfarenheter så definierar de mönster olika. Vid vissa fall är samtliga utvecklare ense i stora drag om vad som är ett mönster men de uppfattar nyttan av det olika och definierar den även annorlunda. På frågan om de vanligaste mönstren som de använder är de även eniga i stora drag men styrker även sina egna arbetssätt. På sista frågan inom intervjutemat, hur de vet vilket mönster som är mest lämpligt för en viss integration, framhåller alla utvecklare erfarenhet som en nyckelfaktor för att hitta det optimala mönstret.

Samtliga respondenter har något hjälpmedel för att lösa ett integrationsproblem utan ett tydligt mönster. De två respondenterna med lite mindre kunskaper kan inte förlita sig på sin erfarenhet i samma utsträckning som de övriga och nämner aldrig i intervjuerna att de tittar på tidigare lösningar som de själva har utvecklat. Alla är dock eniga om att ett mönster underlättar utvecklingsprocessen.

Samtliga respondenter tar även hänsyn till vilken kontext ett mönster har applicerats i tidigare samt kontrollerar om lösning kan uppenbara andra problem i ett senare skede och att det passar deras specifika problem. För samtliga integrationsutvecklare ses detta som en självklarhet och nödvändighet eftersom de vill optimera sitt arbete.

Vid kommunikation och terminologi på arbetsplatsen är alla integrationsutvecklare ense om att det effektiviserar kommunikationen inom verksamheten men alla är inte insatta i terminologin, dels för att olika verktyg har olika termer och dels för att vissa inte är insatta i teorin kring integration. Vid vilka positiva och negativa aspekter som utvecklarna upplever är det de positiva aspekterna som har ett genomgående tema i alla svar. Det effektiviserar kommunikationen, utvecklingsprocessen, återanvändbarhet och sparar framförallt in tid. De negativa aspekterna i svaren domineras av överanvändningen av mönster bara för att de finns samt att förklara problem på en djupare teknisk nivå går inte med hjälp av mönster. På den sista frågan om vilken sorts mönster de använder är de flesta integrationsutvecklare överens om att egenutvecklade i projekt används mest och kundens önskemål spelar in vid val av dessa mönster. Däremot anser en respondent att tre övergripliga mönster har satts för hela verksamheten som ska följas. Detta exemplifierade han genom att berätta att detta appliceras

på nya utvecklare framförallt. Slutligen anser alla utvecklare att detta främjar återanvändbarhet och effektiviserar utvecklingsprocessen.

Avslutningsvis så anser samtliga respondenter att användningen av mönster resulterar i ett mer kvalitativt resultat och framhäver även några ytterligare positiva aspekter. I frågan om det komplicerar, nämner fyra av de att det kan komplicera om mönster endast används för sakens skull, det bör endast appliceras där det är möjligt och verkligen passar. Det ska finnas ett syfte med mönstret annars resulterar det bara i onödig administration och förvirring.

5 Diskussion

Följande kapitel presenterar en jämförelse mellan våra resultat och empirin. Även tillhörande analys och åsikter inkluderas i det här kapitlet för att visa läsaren våra egna tankar rörande hur mönster påverkar en integrationsutvecklare.

5.1 Inställning till SOA vid integrationsutveckling

När ni integrerar system, har ni i åtanke att uppnå en Service Oriented Architecture?

Att uppnå en SOA tar alldeles för mycket tid och resurser enligt F2 som påpekar att han istället fokuserar på att försöka tjänstefiera vissa delar av en integration så de kan återanvändas för att uppnå nytta med detta i framtiden istället.

F1 förklarar ur sin synpunkt att ett SOA-tänk är ett ansvar som ligger hos arkitekten vilket styrks av Holley och Arsanjani (2010) som berättar att SOA för en IT-arkitekt står för en arkitektonisk stil som framhäver lösningar för att integrera olika system genom lös koppling och återanvändning av moduler. Det här synsättet som F1 besitter kan spåras tillbaka till hans erfarenhet och roll som endast utvecklare. När respondenterna inte utvecklar enligt SOA så sker integrationen genom EAI-lösningar, vilket medför att utvecklare inte behöver tjänstefiera de olika delarna enligt Holley och Arsanjani (2010).

Alla integrationsutvecklare har både arkitekt- och utvecklarroller förutom F1. Alla dessa informanter visar kunskaper i SOA och det kan kopplas till deras erfarenhet. De berättar att även om kunden vill implementera SOA i sin arkitektur kan de med hjälp av sina kunskaper exemplifiera för kunden vilka delar av en integration som kan utvecklas som tjänster för återanvändning och vilka andra komponenter som är onödiga att utveckla enligt SOA-tänk. Detta kan tolkas som en individuell rådgivning till kunder för att enligt Holley och Arsanjani (2010) skiljer sig definitionen av SOA beroende på vem som tillfrågar och vad som efterfrågas. Det styrks ytterligare från det faktum att de olika utvecklarna jobbar inom projekt med separata kunder i team eller ibland, ensamma.

Vi har märkt att alla utvecklare ser en Service Oriented Architecture som något för omfattande och komplext för att implementera vid integrationer. Vi tror att detta begränsas på grund av att Enfo är nischade på integrationer och har inte SOA som sin grundläggande specialitet men detta kan bero på faktorer som kundens mogenhet, kunskaper och även integrationsutvecklarens erfarenhet. Just erfarenhet är något som alla integrationsutvecklare påpekar för användningen och påverkan mönster har för deras projekt och arbete. Som namnet antyder och som vi tror står arkitekturen för något alldeles för komplext, även för en utvecklare som har ett större ansvar inom organisationer ses detta som något för stort att sträva efter.

5.2 Erfarenhet av mönster

Vad är ett mönster för dig?

Definitionen av mönster särskiljer sig mellan undersökningspersonerna. Vissa ser det som en standardiserad lösning som främjar återanvändbarhet medan andra definierar mönster på en mer tekniskt nivå. Dock avgränsar de syftet med mönster på ett snarlikt sätt. De påtalar att de är generella nog för att kunna användas som en övergriplig lösning vilket Baroni (2003) nämner. Författaren förklarar att designmönster infördes inom mjukvaruutveckling för att på ett effektivt sätt dela och sprida lösningar på problem som utvecklare vid upprepade tillfällen stött på. Vi anser att även om vissa integrationsutvecklare inte är särskilt insatta inom teorin kring mönster så har de förstått definieringen utav de. Det är lösningar som ska användas på återkommande problem vilket utvecklarna påtalar med återanvändbarheten som en av de faktorer som mönster bidrar med. Vi tror att kommunikation och informationsspridningen inom verksamheter för hur mönster ska användas är två viktiga faktorer för att främja mönsteranvändning.

Vissa intervjupersoner berättar att de har upptäckt att de har använt mönster efter en implementation och detta kan tyda på bristen av erfarenhet. Dessa intervjupersoner har sedan tittat tillbaka och reflekterat om det var den bästa lösningen för just det tillfället. En stor skillnad huruvida mönster har upptäckts i efterhand står F3 och F5 för. Det är intervjupersoner med 8 respektive 10 års erfarenhet från integrationsbranschen. I fallet med dessa respondenter upptäcktes inte mönster efter en implementation. Utan de gick tillbaka för att hitta tidigare lösningar med mönster för att kunna lösa sina problem. Detta tyder på att en bas med mönsterlösningar hjälper till att lösa problem och erbjuder oerfarna integrationsutvecklare en plattform för att kunna reflektera över sina beslut gällande vilka mönster som är optimala. Eftersom mönster inkapslar erfarenhet och best practices för att lösa dessa problem (IBM, 2005).

Vilka är de tre vanligaste mönstren som du använder?

På frågan om vilka tre mönster som de använder är de flesta utvecklare överens om vilka som används. F2 beskriver i sin intervju att verksamheten har satt upp tre övergripande mönster som ska användas men så är det inte i alla fall, vilket Scheibler och Leymann (2008) beskriver som ett stort hinder för organisationer. Enligt författarna måste organisationer fastställa hur mönstren ska användas. Med utgång från intervjusvaren anser till exempel inte F3 att de mönster som övriga använder på arbetsplatsen kan definieras som mönster. Han anser att de visar mer vilken typ av integration det är snarare än att visa hur man implementerar en integration. Det här kan tolkas som om normerna för hur mönster ska användas inte är fastställda inom verksamheten vilket kan skapa förvirring enligt Hohpe och Woolf (2011).

Hur vet du vilket mönster som är mest lämpligt för integrationen?

När utvecklarna tillfrågades om vilka mönster de anser är mest lämpligt för integrationen besvarar alla intervjuobjekt frågan med stora likheter. De anser att det är upp till utvecklaren att själv försöka förstå vad som ska uppnås med integrationen. När oklarheter uppstår i projekten går de flesta utvecklare till arkitekten eller kunden för att kunna bestämma sig för vilket mönster som ska användas. F3 anser däremot att det inte finns ett mönster som är mest lämpligt, utan det handlar om att välja rätt mönster för annars anser han att det blir en felaktig lösning. F4 har en helt skild utvecklingsmetodik. Han utvecklar ibland lösningen och inser i efterhand att integrationen bildade ett visst mönster. Det här visar att dokumentationen för mönster måste vara komplett för att utvecklare ska kunna se och jämföra mönster för att se vilka som passar bäst eftersom mönster inkapslar best practice-metoder. (IBM, 2005).

5.3 Applicering av mönster

Hur löser du integrationsproblem med, respektive utan ett tydligt mönster?

Utan ett tydligt mönster tar de flesta utvecklare hjälp utav Internet och Google. Ofta har varje utvecklare en bas av mönster som har använts tidigare för att lösa problemen. De återgår till dessa lösningar och kontrollerar om det finns något som kan återanvändas. Vilket Dong et al., (2011) styrker genom att förklara att mönster är allmänt accepterade, best practice-metoder. Om det fortfarande skulle finnas oklarheter tar de hjälp av arkitekten eller kunderna. Oftast är kunderna inte tillräckligt mogna för att kunna veta sina exakta krav från början. Detta resulterar i att utvecklarna måste återgå till kunden för att fastställa vad som behövs så de kan veta vilka mönster som fungerar för integrationen. Både F4 och F5 betonar dessutom att det går att implementera en integration men att om kundens krav är tydliga så underlättar och effektiviserar det utvecklingsprocessen. Ibland går de även till kollegor för att få hjälp med problemen. Det här leder till att användningen av mönster bidrar till ett slutresultat med högre kvalitet (Mangalaraj et al., 2014). Därmed kan det tolkas att när det uppstår förvirring inom projekt så kommuniceras det inte tillräckligt bra. Ibland tas det upp med arkitekten, ibland med kunden och andra intressenter. För att stävja kommunikationsproblemen kan en enhetlig språkstandard införas (Hohpe, 2007). Mer om detta nämns senare (se 5.4).

Om ett mönster hittas som har applicerats tidigare, tar ni hänsyn till i vilken kontext det har gjorts? Samt kontrollera om lösningen kan uppenbara problem i ett senare skede och passar det ert specifika problem?

Den här frågan valde vi att inkludera i intervjuguiden för att se om respondenterna tog hänsyn till teorin bakom ett mönster. Det här kan exemplifieras i form av ett köp av en tjänst eller produkt där det inkluderas ett användaravtal och manual, sker ingen kontroll av användaravtalet kan konsumenten gått med på något som inte fanns i åtanke, läses inte användarmanualen kan konsumenten oftast inte nyttja tjänsten/produktens fulla funktion vilket leder till att användandet inte optimeras. Likt den här exemplifieringen nämner Gamma et al.(1994) att mönster har fyra fundamentala element; namn, problem, lösning och

konsekvenser. Hohpe & Woolf (2004) nämner att mönster kan beskrivas enligt många olika strukturer och väljer i sin bok *Enterprise Integration Patterns; Designing, Building, and deploying messaging solutions* att strukturera ett mönster i 11 sektioner, vilket beskriver beståndsdelarna för ett specifikt mönster. Samtliga respondenter svarar att de tar hänsyn till vilken kontext ett mönster har applicerats i tidigare samt kontrollerar om lösningen kan uppenbara andra problem i ett senare skede och att det passar deras specifika problem. Dock poängterar respondent F4 att det här beror på vilket ansvar han besitter i projektet eftersom det är tidskrävande och skulle innebära att mindre tid skulle kunna läggas på hans huvudsakliga uppgift. Han anser att det är arkitektens uppgift att säkerställa detta. F5 anser dock att han alltid måste ta hänsyn till det här eftersom kontexten bestämmer vilket mönster som ska användas. Respondent F5:s resonemang stödjer vad F4 berättar eftersom han huvudsakligen har arkitektrollen i ett projekt. Det här tyder på att rollen och ansvaret i ett projekt har betydelse för i vilken omfattning respondenterna tar hänsyn till det här. Ju mer ansvar integrationsutvecklaren har, desto mer tänker de på kontexten och efterverkningarna av besluten.

5.4 Åsikter om mönster och kommunikation

Har du upplevt kommunikationsproblem med kollegor inom integrationsutveckling på grund av skild terminologi?

Alla utvecklare tycker att det effektiviserar kommunikationen med en enhetlig terminologi men alla är inte insatta i definitionerna. Detta styrks genom Gamma et al., (1994) som påtalar att mönster bidrar till en gemensam terminologi. Hohpe (2007) påtalar återigen, för att stävja kommunikationsproblemen bör en enhetlig språkstandard införas. Detta verkar ha implementerats i organisationen i form av tre övergripliga mönster som har satts som gemensamma mönster som alla på arbetsplatsen ska använda. Något som är anmärkningsvärt är att endast en respondent påtalar detta medan övriga refererar till att användningen av olika verktyg skapar förvirring även om mönstren som används är i grund och botten samma, oavsett verktyg. Samtidigt som vissa överhuvudtaget inte har reflekterat kring teorin och definitionerna av mönster. Osäkerheten i att kommunicera med övriga på grund av avsaknad av teoretiska kunskaper kan stävjas genom en dokumentation för mönster så att användare ska kunna hitta, välja och applicera rätt mönster (IBM, 2005). På detta sätt skulle förvirringen kunna minska i organisationen och samtidigt skulle anställda kunna lära sig teorin runt de olika termerna utan att bli förvirrade eller dra sig för att kommunicera på grund av bristande kunskap.

Vilka positiva och negativa aspekter ser du vid användningen av mönster?

På den här väldigt öppna frågan hade alla respondenter förvånansvärt liknande svar. Alla integrationsutvecklare föreslog att de positiva aspekterna bestod av effektivisering av kommunikationen, utvecklingsprocessen, återanvändbarheten och besparingen i form av tid.

Dessa åsikter styrks i litteraturen (Mangalaraj, 2014. IBM, 2005) som påtalar den effektiviserade utvecklingsprocessen, återanvändbarheten och sänkningen av utvecklingskostnaderna tillsammans med tidsbesparingen för att en ny lösning behöver inte skapas varje gång. Vi antar med utgång ifrån deras svar och litteraturen att de kan kopplas ihop och att verksamhetens alla integrationsutvecklare är insatta med de positiva aspekterna.

På de negativa sidorna av mönster påpekade utvecklarna, överanvändningen. De tycker att mönster överanvänds i vissa situationer bara för att de finns. Vid användningen av mönster som är abstrakta lösningsförslag så måste utvecklare vara noggranna vid implementeringen av ett specifikt mönster för att lösningen måste anpassas efter problemet (Mangalaraj et al., 2014). Detta tror vi beror på att dokumenthanteringen och riktlinjer saknas inom verksamheten som både nya och gamla utvecklare kan läsa och utbilda sig inom. För att empirin inte påvisar någon databas av information som utvecklarna kan vända sig till när det gäller *hur, var* och *när* vissa mönster ska appliceras.

Är mönstren du använder dina egenutvecklade, verksamhetens gemensamt uppsatta mönster eller är det kunddirektiv?

Mönstren används mestadels i projekt enligt respondenterna och de kan vara utvecklade specifikt för projekten. Detta medför att det blir enklare att sprida lösningar på problem som utvecklare stöter på i mjukvaruutvecklingen (Baroni, 2003). För att projekten kan bestå av flera gruppmedlemmar eller enbart en person. En respondent uttrycker att kundens önskemål måste tas i beaktande vid valet av mönster och förklarar att kunden gärna får lämna flexibilitet för egenutvecklade mönster inom projektet för att mönster är något som ständigt måste utvecklas för att hänga med i den teknologiska utvecklingen. Det här kan tolkas som en medvetenhet om att kundens önskemål är en prioritet vilket är underförstått men att utvecklaren tänker på hur integrationen ska implementeras för att kunna använda egenutvecklade mönster påvisar ett kontexttänkt. Gamma et al., (1994) nämner att utvecklaren måste tänka på konsekvenserna av implementeringen för ett mönster för att främja flexibiliteten, utbyggbarheten och portabiliteten. Vi tror att om alla utvecklare genomgick samma utbildning eller hade en diskussion för hur mönster ska användas i grunden så skulle det kunna hjälpa verksamheter att hålla ett enhetligt kundbemötande och dessutom främja ett enhetligt arbetssätt.

En annan aspekt ses hos en respondent som påtalar att tre mönster har satts för hela verksamheten och att de ska användas. De övriga respondenterna nämner inte att tre mönster har bestämts för hela verksamheten utan påtalar att det är baserat på riktlinjerna inom projektet och kundens önskemål. Detta kan tolkas som att en dokumentation inte finns vilket det bör göra (IBM, 2005) för att underlätta att hitta mönstren.

5.5 Resultat av mönsteranvändning

Resulterar användingen av mönster i ett mer kvalitativt resultat eller komplicerar det utvecklingsprocessen?

Samtliga respondenter är överens om att användning av mönster höjer kvaliteten på slutresultatet. Somliga berättar även att det ökar underhållbarheten De pekar bland annat på att det finns riktlinjer för hur det ska användas och att det är en form av kvalitetssäkring av resultatet. Utvecklarna berättar även att det sparar tid och pengar. Gamma et al., (1994) nämner att mönster har en problembeskrivning som visar när det bör tillämpas och i vilket sammanhang. Vi tror att Gammas et al., (1994) beskrivning kan appliceras i kontexten eftersom att det visar riktlinjer för hur det bör användas och för vilket problem. Det här bekräftar besparingen i form av tid och pengar.

Respondenterna är överens om att det komplicerar utvecklingsprocessen när mönster överanvänds och appliceras utan ett tydligt syfte. Mangalaraj et al., (2014) poängterar att mönster är abstrakta lösningsförslag som måste anpassas utefter problemet, annars kan det leda till oönskade funktioner och orsaka fel i koden. En respondent förtydligar detta och påtalar att han har stött på utvecklare som har applicerat mönster på varenda litet scenario vilket endast har komplicerat integrationen och ökat potentiella fallgropar. Respondenten förklarar att erfarenhet är en viktig faktor för att bedöma när och var det är lämpligt att applicera ett mönster. Eftersom utvecklarna oftast har fria tyglar för hur integrationen ska utformas kodmässigt tror vi att det är upp till individens sunda förnuft att betrakta lösningen kritiskt och subjektivt för att applicera mönster där det är lämpligt. Det finns inga givna direktiv på var och när mönster bör appliceras eftersom det är, som tidigare påtalat, abstrakta lösningsförslag.

6 Slutsats

Syftet med uppsatsen var att besvara forskningsfrågan som ställdes i inledningen:

På vilket sätt påverkar integrationsmönster en utvecklare vid systemintegration?
– Vilka för- och nackdelar finns det, sett ur en integrationsutvecklares perspektiv?

Genom att intervjua integrationsutvecklare på företaget Enfo Zystems har vi kunnat konstatera hur mönster påverkar deras arbete. Svaren skiljer sig mycket vid vissa frågor och liknar vid andra. De kvalitativa samt positiva och negativa faktorerna som har identifierats är följande;

- Definieringen av mönster är liknande i alla svar men vissa tycker att mönster inte hjälper att förklara problemen på en djupare, teknisk nivå.
- Kommunikationen förbättras och effektiviseras eftersom en gemensam terminologi kan fastställas. Alla integrationsutvecklare är däremot inte insatta i terminologin och det skapar kommunikationsproblem på arbetsplatsen. Den här problematiken har inte diskuterats öppet på arbetsplatsen och det kan bero på att individer inte vill bli stämplade som okunniga.
- Alla integrationsutvecklare är överens om att det effektiviserar utvecklingsprocessen för att det besparar tid, pengar och ökar förståelsen för integrationen samtidigt som det underlättar underhållbarheten för integrationer eftersom att nästkommande utvecklare vet om vilka mönster som har använts och kan snabbt sätta sig in i arbetet.
- Främjar återanvändbarhet då de inte behöver uppfinna hjulet igen utan kan använda tidigare mönster som de har applicerat på liknande problem.
- Överanvändning av mönster är en negativ aspekt som alla integrationsutvecklare påpekar. De tycker att det finns en viss tendens att applicera mönster överallt och det hjälper inte integrationen som helhet, det komplicerar istället för nästa individ som ska ta över arbetet.

Avslutningsvis har forskningsfrågan undersökts men ett definitivt svar som kan dras som en generell slutsats är svår att definiera. Alla integrationsutvecklare definierar mönster på olika sätt, de arbetar på olika sätt och med olika verktyg samt har olika rutiner och arbetsprocesser. Däremot har bakomliggande faktorer för *hur, vad* och på *vilket sätt* som mönster påverkar integrationsutvecklare, identifierats. En djupare undersökning och varför just dessa faktorer har likställts skulle kunna erbjuda ett definitivt svar men på grund av avgränsningar på studien har dessa inte kunnat fastställas.

Bilaga 3 – Transkribering av intervju 1

F1 = Ålder: 24 år
Utbildning: Kandidatexamen i Systemvetenskap på Lunds universitet
Antal år inom integrationsbranschen: 1,5 år

I = Intervjuperson

I: När ni integrerar system, har ni i åtanke att uppnå en Service-Oriented Architecture?

F1: Ja, jag vet inte om jag har det i åtanke så mycket, faktiskt. Det är mer att jag inte har kommit så långt än, man borde ha det, man borde alltid tänka på det. Sen är det också lite vad som menas med en Service-Oriented Architecture. Nä, jag tänker nog inte så mycket på det när jag utvecklar, utan jag tycker att det…

I: Det finns direktiv högre upp eller?

F1: Ja, jag får ju liksom oftast specifikationer, de säger ju inte hur jag ska utveckla saker. Ibland kan man ju få arkitektoniska direktiv för hur en integration ska utvecklas men det är ju väldigt olika och då får man ju utgå ifrån att arkitekten eller någon som sitter ovanför har tänkt på det här.

I: Typ, låg koppling?

F1: Ja, till exempel återanvändbara objekt och sådant här. Ibland är det till exempel så att, låt säga att jag får. Jag har ju gjort en integration och sen så beställer kunden en ny integration då kan ju min arkitekt till exempel säga att: men den här nya integrationen skulle kunna integreras i den här gamla för att vi kan använda en gammal integration för att inkludera en ny, istället för att ha två stycken integrationer då som måste underhållas på olika sätt så kan man inkludera de i en och samma. Då tänker han ju lite på SOA-tänket. Men jag hade nog inte tänkt det om jag bara hade fått en specifikation, att den här integrationen kan gå in i den här integrationen.

I: Du behöver inte tänka på det heller?

F1: Alltså inte i den rollen som jag har. För jag har inget större ansvar hos de kunderna jag sitter på just nu, alltså den arkitektoniska biten. Så jag tänker nog inte på det, men man borde göra det.

I: Vet du om det finns direktiv högre upp för det?

F1, Det kan jag inte svara på, det är nog från person till person och vem som sitter som arkitekt. Jag vet att en kollega är sjukt inne på det och även min projektledare sjukt duktig på

det. Han sköter mest det här arkitekturella tänket. Han säger inte hur jag ska utveckla integrationerna men oftast hur jag kan gå till väga. Så det är han som sköter det.

I: Så om du får två olika specifikationer som du tycker kan göras som en?

F1: Då hade man kunnat gå tillbaka till arkitekten och frågat om man kunde göra på det här sättet istället eftersom att det hade gynnat oss på det här viset i framtiden. Jag har det lite åtanke, men ja jag går inte och tänker jättemycket på det. Man litar på att det kommer högre uppifrån.

I: Enligt källor är ett mönster en beprövad best-practice metod för att lösa ett vanligt förekommande problem. Vad är ett mönster för dig?

I: Du kan hålla med om den definitionen om du vill det.

F1: Men ja, det är det väl liksom. Ett mönster är en standardiserad lösning, jag ser det mer som ett sätt att utveckla någonting, alltså man har en uppgift som ska lösas och det bästa sättet att lösa denna är att utveckla den på det här sättet. Det är inte bara lösningar på problem, så har jag uppfattat det. Men ja, det stämmer väl, det tycker jag. Just det här, det ska ju vara en best-practice, sen är det ju från fall till fall också. Det är inte alltid den bästa lösningen. Men visst om man ser ett vanligt förekommande problem eller om man säger en vanligt förekommande uppgift. Liksom har man ett sådant här Fire-and-Forget mönster. Du har ett system A som bara ska skicka till system B och vi bryr oss inte om det går fram eller inte. Då har man en input, sen vad man gör med meddelandet i mitten, det kvittar ju. Mönstret är att man har en input och en output utan något svar tillbaka. Det sändande systemet ska ju inte få något tillbaka, utan det ska bara skicka saker. Går det fram så går det fram, går det inte fram så går det inte fram. Det är ju ett Fire-and-Forget mönster. Det använder jag ofta och det är ju en best-practice.

I: Men till exempel första gången du träffa på nya mönster, tänkte du på att det här var ett mönster eller har du upptäckt det i efterhand?

F1: Nej, jag har upptäckt det i efterhand. Jag hade ingen aning när jag började. Utan det var mer att här har du en integration, se till att göra den klar. Man använder det utan att tänka på det. Exempelvis det här Fire-and-Forget mönstret, det är så lätt liksom. Men om man har ett Request-Reply mönster, där man då skickar iväg ett meddelande och sen förväntar man sig ett svar tillbaka. Det är väl någonting som jag inte hade gjort om jag inte hade vetat att det var ett mönster. De lösningarna hade nog blivit mer omständliga om jag inte hade vetat att det fanns ett mönster för det, ett standardiserat sätt att lösa den här integrationen på. Så där har jag haft nytta av mönster. Det talar ju om för mig hur jag ska göra för att det ska fungera.

I: Vilka är de tre vanligaste mönstren som du använder?

F1: Fire-and-Forget, Request-Reply, Publish-Subscribe

I: Hur vet du vilket mönster som är mest lämpligt för integrationen?

F1: Det får man utreda från integrationsspecifikationen, man får ju en specifikation av sin arkitekt, så skriver han kanske att vi har ett direkt-system som ska till ett SAP-system sen vill även det sändande systemet ha en slags bekräftelse på att meddelandet har gått fram, då är det ju lite Request-Reply. Man skickar in någonting och så vill man ha ett svar tillbaka, då kan man läsa ut det från integrationsspecifikationen. Skulle det vara Fire-and-Forget så är det i princip bara att vi ska skicka iväg ett meddelande och det behöver inte returnera någonting, det ska exempelvis bara in i en databas. Då vet jag att det är ett Fire-and-Forget.

I: Det ser du direkt?

F1: Ja, men det har nog med erfarenhet att göra hur snabbt man kan se och även hur många mönster man kan. Ibland kan man ju heller inte applicera något mönster, då får man göra en speciallösning. Det beror på specifikationen och hur väldefinierad den är om man kan använda sig av mönster eller inte.

I: Den här speciallösningen du gör själv för just den här integrationen, sparar ni den någonstans?

F1: Nä.

I: Delar ni den mellan varandra?

F1: Ja, det är klart att man delar med sig av erfarenheter och saker man har gjort. Men det är inte så att vi kallar det något särskilt mönster, att vi säger att det här en standard eller standardlösning på det här problemet. Man kan kanske göra det, men jag har aldrig gjort det. Man ska ju inte överanvända mönster, det kan ju lätt bli så. Vi hade tidigare en kollega som utvecklade och använde mönster hela tiden, i princip allt skulle bestå av mönster och det blev tyvärr inte så bra. Det resulterade ofta i väldigt komplicerade lösningar och det var svårt att applicera hans mönster på olika integrationer. Så man ska helt enkelt inte överanvända mönster, man ska endast applicera dem där det går. Det finns ju en anledning till varför det är en best-practice. Men för att svara på frågan, hur jag vet att det är mest lämpligt. Det vet jag genom att jag tittar på specifikationen och talar med arkitekten och han kanske har lite input och jag frågar honom om det här verkar lämpligt.

I: Du tittar alltså inte på olika mönster och jämför de?

F1: Jo, det gör jag ibland och det sker genom att jag söker efter det på internet för att se om det finns en standard. Men det är svårt eftersom man inte kan specifikt söka på sin specifikation men man har kanske en idé om någonting som man kan söka efter. Så man kan absolut kolla upp det själv också. Men grund idén kommer nog från specifikationen från början.

I: Hur går du tillväga för att lösa ett integrationsproblem med, respektive utan ett tydligt mönster?

F1: Jag vet ju vad jag har att arbeta med, vi har färdiga produkter. Jag arbetar med Websphere Message Broker, och jag vet ju vad jag har att arbeta med, jag har ju mina noder i min palett och sen får man snickare ihop något med hjälpa av det. Ibland blir det kanske lite omständliga lösningar men i första hand gör jag det själv, men sen finns det väldigt bra hjälp på IBM:s hemsida, där de ger förslag på olika mönster. Websphere Message Broker, är ju IBM:s produkt och de har en stor hjälp sida där man kan gå in och söka efter saker, där de föreslår mönster och lösningar. Så där tar jag hjälp ifrån först. Sen pratar jag kanske med min arkitekt eller kollegor, man får ju höra sig för, det är upp till mig själv att skapa lösningen och ta reda på informationen som jag behöver. Men hjälpmedlen är väl allt ifrån Google, IBM:s hjälpsida, kollegor och projektets arkitekt.

I: Föredrar du att använda ett mönster om det finns?

F1: Ja, det är klart. Då har man ju en guide, det är svårare att sitta själv och komma på lösningar också ifrågasätter man om det här verkligen är den bästa lösningen. Man vill ju alltid leverera en så bra lösning som möjligt, oftast lösningar som är lätta att underhålla och de ska framförallt vara skalbara. Det har ju hänt att folk har gjort hemma snickrande lösningar och när de väl gjorde den här integrationen så kanske det endast skickades fem meddelanden per dag, och sen har den vuxit så helt plötsligt finns där 40 mappningar och 60 meddelanden var 20 minut. Då måste man ibland bygga om hela integrationen just för att man inte har tänkt till arkitektoniskt, hur det här ska fungera. I det här fallet hade ett mönster varit bra eftersom att jag antar att mönster tar in allt det här med skalbarhet.

I: Ja, det är beprövat.

F1: Ja precis, det är en beprövad best-practice. Det är ju att föredra och om man inte gör det så ska man verkligen tänka till.

I: Om ett mönster hittas som har applicerats tidigare, tar ni hänsyn till i vilken kontext det har gjorts? Samt kontrollerar om lösningen kan uppenbara andra problem i ett senare skede och passar det ert specifika problem?

F1: Det hade jag nog inte gjort, men man borde ju göra det. Men ska jag vara helt ärlig så hade jag nog inte tänkt så långt, men det har nog med erfarenhet att göra. Men det spelar egentligen ingen roll, alla som är integrationsutvecklare är inte erfarna. Men nä, jag hade inte tänkt på det. Och det här kontrollera om lösningen kan uppenbara andra problem i ett senare skede. Det hade jag nog funderat på, för man vill ändå tänka lite längre fram och ändå utveckla integrationen så den ska kunna byggas på i framtiden. Man vill inte göra något som är helt statiskt utan det ska ju gå att bygga på och modifiera, så det hade jag nog funderat på. Om jag hade funderat på det första, om jag först hade tänkt på "om ett mönster hittas som har applicerats tidigare".

I: Tar ni hänsyn till vilken kontext det har gjorts? om du till exempel hittar det genom Google, om du kollar på när det har använts och går det att applicera på mitt specifika problem?

F1: Ja, det hade jag tänkt på, man vill ju att det ska passa så bra som möjligt.

I: den andra delfrågan: Samt kontrollera om lösningen kan uppenbara andra problem i ett senare skede?

F1: Det är detta jag menar att jag inte tänker på. Att jag hade kontrollerat att lösningen kan uppenbara andra problem i ett senare skede, eller jo det hade man ju gjort. Jag måste ha läst fel. Man hade ju försökt hitta något som passa så bra som möjligt och sen när man hade implementerat det så hade man försökt tänka hur det här fungerar i framtiden, så ja det hade jag absolut gjort.

I: Har du upplevt kommunikationsproblem med kollegor inom integrationsutveckling på grund av skild terminologi?
F1: Ja, det har jag. Men det beror ofta på att folk använder slangord och förkortningar. Ni tänker mer på mönster?

I: Ja, du till exempel om du säger Fire-and-Forget så vet…

F1: Nä, det hade nog inte alla vetat.

I: Vi tänker om det blir lättare att kommunicera på det viset?

F1: Ja, det är klart. Alla borde använda, om man får kalla det facktermer. Man borde göra, men sen är det inte alla som gör det.

I: Har det framkommit när du till exempel pratar med arkitekten att ni pratar om samma sak, och när du sen applicerar det upptäcker du att ni har uppfattat det på fel sätt. Till exempel angående frågorna om integrationen.

F1: Aldrig när vi har diskuterat om ett mönster, då har det inte uppkommit. Eller du menar om han har menat ett mönster som jag inte har förstått?

I: Ja, precis.

F1: Nä, jag har aldrig upplevt det. Men jag kan tänka mig att det förekommer, att folk som inte är så insatta eller använder sig av mönster, eller de tror inte de gör det. Att de inte hade förstått det. Det hade kunnat bli fel eftersom att man inte förstår vad den andra menar. Eller har jag upplevt det..?

I: Så det kan bli åt båda håll egentligen, att det inte finns någon terminologi plus att någon

uttrycker sig i from av mönster och någon inte förstår det.

F1: Ja, precis. Jag kan säga att jag inte har stött på det än. De pratar nämligen inte så mycket i mönster, i alla fall inte i det projektet jag arbetar i nu. Min kollega gör det dock, han är glad för mönster.

I: Vi tänker här på arbetsplatsen om han säger till dig att du kan lösa det här genom att göra det här också nämner han ett specifikt mönster, exempelvis Fire-and-Forget.

F1: Ja, det gör han. Och då är det väldigt mycket klarare än att förklara att du ska ha en MQ input nod där, du ska ha en MQ output där och den ska inte returnera någonting.

I: Underfrågan till denna fråga är om du anser att här bidrar till en mer effektiv kommunikation?

F1: Ja, det gör det ju. Ja men det är som du säger, jag frågar ju en del eftersom jag fortfarande håller på att lära mig och då är det mycket lättare för mina kollegor att nämna ett mönster som jag ska titta på. Då vet jag hur jag ska göra liksom. Så absolut, det tycker jag definitivt, det är sjukt viktigt också som ni säger när man ska prata om det. Till exempel på en arbetsplats att man har en gemensam terminologi. Så ja, det tycker jag men jag kan inte säga att jag har upplevt kommunikationsproblem än.

I: Är mönstren du använder dina egenutvecklade, verksamhetens gemensamt uppsatta mönster eller är det kunddirektiv?

F1: Vad menar du med egenutvecklade?

N: Vi syftar på om du tar fram egna.

F1: Nä, det gör jag inte. Det finns ju standarder, typ de som vi har tagit upp, det är sådana man använder. Sen om man vill kalla sina egna lösningar ett mönster, ja då får man väl göra det, men det är ju inte mönster. Men nä, jag har inga egenutvecklade mönster och vi har inga gemensamt uppsatta i verksamheten. Däremot har vårt utvecklingsteam i mitt projekt, de har satt upp mönster som vi ska använda. Så det är inte på verksamhetsnivå utan på projektnivå. Kunddirektiv nä, de har ingen aning om vad vi gör egentligen så det är svårt för dem. Men ibland så har ju faktiskt företag en integrationsavdelning och då skulle de det kanske kunna komma ett kunddirektiv men det är väldigt ovanligt, utan det är vi som står för att utveckla en så bra integration som möjligt och leverera till kunden. De köper ju den här tjänsten av oss. Sen om de vill gå in och påverka så absolut då lyssnar vi på vad kunden säger men jag tror mer att det antingen är verksamheten som sätter upp det eller som vi har det på projektnivå.

I: Och vi tänkte också på att kunden ibland äger mappningsspecifikationen.

F1: Jaja.

I: Om det påverkar vilka mönster som används?

F1: Alltså för mappningsspecifikationerna så spelar det ingen roll, för att en mappningsspecifikation aldrig kommer vara, nu ska jag inte säga för mycket här… De tre mönstren som jag har nämnt nu, där spelar det ingen roll hur mappningsspecifikationen ser ut eller vem som äger dem för att det är fortfarande mönstret, ja hur ska man förklara det här. Oavsett om det är ett Fire-and-Forget eller om det är Publish-Subscribe så spelar det ingen roll vem det är som äger mappningsspecifikationen för att mappningsspecifikationen kommer alltid se likadan ut oavsett vilket mönster du använder. En mappning ingår inte i ett mönster utan det är såhär, ett mönster om man tar Publish-Subscribe, det är att man har input en publisher också har man subscribers, vad som händer här i mellan är routing, routing är ju på Publish-Subscribe. Om det är en mappning här, kanske en mappning till här, man kanske arkiverar ett meddelande och sådant, vad som händer här i mellan spelar ingen roll för mönstret kommer fortfarande se likadant ut.

I: Så länge som ni utför det som är i mappningsspecifikationen så kvittar det hur ni gör det?

F1: Ja och det kvittar vart den är och vilket mönster den ligger i den kommer alltid se likadan ut.

I: Sen har vi en underfråga här, främjar det återanvändbart och effektivare...

F1: Ja, absolut. Det är ju det som vi har pratat om hela tiden här i princip, så ja det gör det. På alla sätt och vis. Om vi går tillbaka till den förra frågan så en gemensam terminologi får man ju, som vi har ju, alla är införstådda med att de mönstren ska vi använda och när någon då säger ett mönster till någon annan så vet man precis vad man menar och ja det är precis som ni skriver här; effektivare utvecklingsprocess, återanvändbarhet…

I: Vi tänker om det nu finns på projektnivå, så kanske man kan återanvända mönstren?

F1: Ja, men det gör man ju. Ja men då pratar vi mer om egenutvecklade?

I: Ja, om det finns så kanske det är lättare?

F1: Ja, det finns ju också, att någon kanske har en riktigt schyst lösning på något, inom mitt projekt så kan det här problemet kanske återuppstå igen. Oftast är det ju så att många gånger när jag får in en uppgift, så säger kanske person X, men kolla på den integrationen. Det är samma lösning alltså, man kanske inte kallar det ett mönster men det blir ju ändå återanvändbarhet, låt oss kalla det mönster, det är det ju faktiskt. Det är ju ett sätt att lösa någonting på, så det är ju ett mönster.

I: Vilka positiva och negativa aspekter ser du vid användningen av mönster?

F1: Ja, det är som vi har nämnt här innan ju. Det positiva är effektivare utvecklingsprocess på

grund ut av en gemensam terminologi, sen har man ju återanvändbarhet, det är ju också bra. Ja, vad finns det mer… det är ju att man har en standard. Det underlättar arbetet i allmänhet när du får in någonting, du vet hur du ska lösa det, du kan lösa det snabbare. När du har gjort de här många gånger så går det på ren rutin, man bara slänger in allting. Vi har faktiskt gjort en grym grej, jag kan ta det kort bara. Enligt våra mönster så skapas det upp ett skelett i toolkitet, man har ett skript som bygger delar av integrationen till dig, så när du har ett Fire-and-Forget mönster, då har du alltid en input och alltid en output och kanske någon loggning där i mellan. Då har vi ett skript, man klickar bara på en knapp så öppnas toolkitet också skapas skelettet till integrationen. Så man slipper själv sitta och dra och pilla och det är ju jättebra för någon som aldrig har gjort det innan. För låt oss säga att vi får en helt ny kille som sitter här också så säger vi att du ska göra den här integrationen, du ska använda det här Fire-and-Forget mönstret, du kan bara använda det här skriptet, köra det också kommer du får upp skelettet till integrationen. Då har ju inte halva arbetet gjorts men en stor del och hade man kunnat applicera det här på lite större mönster som är lite mer avancerade så har man sjukt mycket gratis. Det var bara en litet sidospår. Men ja, där hade man verkligen sett hur det kan effektivisera arbetsprocessen och underlätta för alla involverande. Men ja, vart var vi. Negativa aspekter är väl det här, överanvändandet av det, man blir kanske lite insnöad på det här med mönster och det kanske till och med blir så att lösningen inte blir så bra just i den här specifika integrationen för att man är såhär insnöad på det här att vi måste använda mönster. Det blir kanske inte en särskilt effektiv integration då, ibland måste man göra speciallösningar och om man då sitter och ältar mönster för länge så kanske det tar längre tid att utveckla. Man kanske sitter och letar flera dagar efter ett lämpligt mönster också är det ju tidsaspekter och det kostar ju pengar när vi är inhyrda och sådant. Så överanvändning tror jag inte är bra eller att man snöar in totalt på att man ska använda mönster. Annars ser jag inte så många negativa aspekter med att använda mönster, det är ju de bästa lösningarna, det är så klart att man ska använda de i så stor utsträckning som möjligt.

I: Resulterar användningen av mönster i ett mer kvalitativt resultat eller komplicerar det utvecklingsprocessen?

F1: Om vi tar kvalitativt då, man sparar ju tid om man hittar rätt mönster eller om man har ett mönster, ett färdigt mönster som man kan återanvända. Då tycker jag att man sparar tid och tid innebär ju att man sparar pengar till kunden. Det är ju den bästa lösningen, det är ju en beprövad lösning så det här ska ju vara det bästa. Man får ju lita på att det är kvalitativt det man gör liksom. Det är väl lite som att kvalitetssäkra, nu har jag utvecklat det här enligt ett mönster eller en lösning som är beprövad, det här funkar och är effektivt. Så bara användandet av det gör väl att det blir kvalitativt får man ju hoppas.

Ja, komplicerar. Det är samma sak där igen, med överanvändandet och att det medför att det kan ta längre tid.

Bilaga 4 – Transkribering av intervju 2

F2 = Ålder: 30 år
Utbildning: Informatik - Design av organisations- och verksamhetsstödjande system.
Antal år inom integrationsbranschen: 6 år

I = Intervjuperson

I: När ni integrerar system, har ni i åtanke att uppnå en Service-Oriented Architecture?

F2: Nej inte i åtanke att uppnå för att beroende på storlek på kund och vad man vill uppnå så är det bra att ha det i baktanke. Men att tjänstefiera allt är väldigt komplext för att det bygger på att du har väldigt mycket kunskap om vad som finns och hur du ska nyttja de tjänsterna osv.

Så för de kunderna jag jobbar för, har man oftast en fördiskussion innan man implementerar något, t.ex. ett nytt system. Ska vi göra detta som vanlig integration eller ska vi försöka tjänstefiera vissa delar? Så man har det i baktanke men inte i åtanke. Man försöker inte uppnå SOA, utan kan jag på något sätt tjänstefiera detta på ett bra sätt? Som vi kan ha nytta av i framtiden.

Problemet med nya, mindre kunder som precis har börjat med integration är att man vill tjänstefiera allt. Det innebär att du lägger ner väldigt mycket jobb på att tjänstefiera saker som du sen inte kommer att nyttja, om ni är med på vad jag menar. Så det är lite tidsperspektiv och nytta av det. För några år sedan så tjänstefiera man allt och det gör man inte riktigt idag där jag jobbar i alla fall.

I: Enligt källor är ett mönster en beprövad best-practice metod för att lösa ett vanligt förekommande problem. Vad är ett mönster för dig?

F2: Det är precis samma sak. Dels bygger det ju på att om jag ritar en integration så kan jag säga att den bygger på Request-Reply eller Fire-and-Forget, det gör att den jag ger det till - redan förstår det själva logiska flödet. Sen mönster finns ju på flera olika sätt, mönster är t.o.m. toolkit där du egentligen kan använda mönster så att du kan producera dina artefakter. Det är något man vill jobba med men är väldigt komplext.

T.ex. en kund jag jobbar för så har jag skriptat så du har en standardintegration (ett skelett) som kan ses som ett mönster som du hela tiden genererar, så du har din basintegration. Därutöver har du ett mönster, ett logiskt flöde som du har ritat upp som du kan ge till utvecklaren och säga - jag vill att du utvecklar det här.

I: Det här skelettet, mönstret, är det du som har skapat det åt kunden?

F2: Ja, det är jag som har skapat det.

I: Vilka är det tre vanligaste mönstren som du använder?

F2: Publish-Subscribe, Fire-and-Forget och Request-Reply om vi pratar om integrationsmönster. Request-Reply är ju fråga-svar. Du ställer en fråga till ett system och får en fråga tillbaka. Fire-and-Forget, du skickar från ett system till ett annat. Du vill inte ha en bekräftelse eller något tillbaka. Du skulle kunna ha en Fire-and-Forget med en dum bekräftelse som egentligen säger 'vi skickar filen, varsågod'.

ESB:n har ju ett oftast ett problem eftersom det är mannen i mitten. Så även om jag har skickat filen så vet jag ju inte om mottagande systemet har processerat den. Så en riktig bekräftelse är när systemet processerar meddelandet och skickar en bekräftelse tillbaka och säger att vi har tagit emot filen och processerat den. Ibland implementerar man en dum bekräftelse som säger 'vi har tagit emot filen'. I bankvärlden är det supervanligt och det finns ju t.o.m. kommunikationsprotokoll som har inbyggd bekräftelse. T.ex. AS2, AS3, AS4, du kan t.o.m. bygga in det med SFTP till exempel.

Du har egentligen ett åtagande som säger att - vi har skickat filen, ja, vi har mottagit den. Så det aldrig kan vara ett objekt för diskussion. För du kan hela tiden vara säker på att just den här filen vi skickade säger de att dem har tagit emot på teknisk nivå. Med detta så kan du göra en verifiering på att de faktiskt har tagit emot och datan har inte ändrats på vägen. Det finns ju flera sätt att uppnå det.

Publish-Subscribe är egentligen att du publicerar ett meddelande på ett specifikt ämne. Sen har du mängder av subscribers (prenumeranter) en till flera eller inga. De tar emot just det meddelandet.

I: Hur vet du vilka mönster som är mest lämpliga för integrationen?

F2: Det är en väldigt bra fråga. Oftast bygger det väl på att du läser igenom, vad är det man försöker uppnå? Sen försöker förstå. Oftast är det en slags diskussion där behoven växer fram och utifrån behoven så tänker du väl egentligen. För du har ett slags kunskapsbas om ungefär vad EAI-patterns är. Om du har läst den boken så har du ändå en viss kunskapsbas att hämta information från.

Det är svårt att säga hur jag vet. Det bygger snarare på erfarenhet och har gjort liknande lösningar tidigare och på samma sätt kan man då veta att det kanske inte alltid är bra att göra en sån här lösning för det här problemet osv.

I: Hur går du tillväga för att lösa ett integrationsproblem med, respektive utan ett

tydligt mönster?

F2: När det har kommit så långt att utvecklaren får det så är det i princip alltid ett mönster. Vi har egentligen tre standardmönster och det är de tre jag nämnde.
Sen har vi förkortningar från det. Såna saker som skiljer sig. Fire-and-Forget kan ju ha en databas som den använder också för att routa meddelanden rätt. Att lösa ett integrationsproblem utan ett mönster kan bli svårt. För då bygger du egentligen något jättekomplext. Om vi går tillbaka till integration så pratar vi om spagetti och att ESB:n löser ihop spagettin men du har lika stor möjlighet att bygga spagetti i ESB:n. Har du inte tydliga mönster finns det ju risk för att du gör t.ex. samma sak om och om igen i flera olika versioner istället för att kanske använda gemensamma bibliotek. Använder du mönster så har du mycket större potential redan i din logiska ritning säga att - det här är ju en Fire-and-Forget och då kan jag använda den här bibliotekstjänsten för att lösa detta.

På kunder oftast, så har man ju gemensamma komponenter som återanvänds i flödena. T.ex. en 'error handler' det är inte något du bygger för varje unikt flöde för det är ju som ett bibliotek eller en gemensam komponent. Det är sådant man kan återanvända. På en kund som jag jobbar för så skapade vi en sak som vi kallar för MBC (Micro Build Component) där vi egentligen på ritningsnivån drog in subkomponenter som kan likställas med EAI-mönster. Skillnaden var att det var komprimerat så en ikon stod för error handler, loggning och en MQ-input. En annan ikon stod för transformering och en MQ-output t.ex.

Drog vi in de två i ett flöde så hade vi två stycken MBC. De var väldokumenterade och gjorde att en arkitekt skulle kunna dra in de och veta exakt hur lösningen skulle se ut och en utvecklare visste precis att MBC 1 var error handler, loggning och MQ-input och den andra var transformering och MQ-output. På så sätt har du egentligen stöpt utvecklaren i en form att den här integrationen utvecklar vi på det här sättet och du har tydliga riktlinjer inom de här MBC-komponenterna.

Medan EAI-mönster så har du en nod för ett ämne, en nod för MQ-kö osv. Här komprimerar vi ihop det till klossar som t.ex. lego-klossar. Nu jobbar vi inte så mycket med det, för nu finns det ett problem. Tar du ifrån en utvecklare dess utvecklingsmöjligheter för ett problem kan du lösa på tusen olika sätt så blir det tråkigt att sitta och göra samma sak i princip. Så nu kör vi de logiska flödena för på en ritning så göms flödena i en flödeskloss. Där har utvecklaren vilken frihet som helst att utveckla precis hur den vill. Men själva logiska flödet d.v.s. Request-Reply så är det två flöden. Ett för Request och ett för Reply.

Hur jag går tillväga för att hitta ett mönster eller för att lösa ett integrationsproblem bygger mycket på att om jag får en specifikation som säger något och det är otydligt. Då frågar jag tillbaka så att jag kan säkerställa ett Request-Reply t.ex. om det är synkront eller asynkront. För det är sådana saker som är viktiga rent tekniskt. Är det en fråga och det kommer ett svar på den

specifika frågan eller är det bara en fråga där de inte bryr sig om svaret är kopplat till frågan. Då är den en lösning.

Men ska det vara kopplat så är det en annan lösning osv. Det bygger lite på Best Practice sen tidigare och att säkerställa vad det är vi vill uppnå, vad det är vi ska utveckla.

I: Om ett mönster hittas som har applicerats tidigare, tar ni hänsyn till i vilken kontext det har gjorts? Samt kontrollerar om lösningen kan uppenbara andra problem i ett senare skede och passar det ert specifika problem?

F2: Absolut, så är det alltid. Efter att du har gjort en implementation av t.ex. en Publish-Subscribe-lösning så utvärderar du - behöver vi en standardisering här, i vår interna ESB för att detta ska fungera?

I den värld jag kommer ifrån skiljer man på Subscriber-objekt, Ämnesobjekt och flöden. Flöden ägs av en teknisk lösning medan Subscriber och Ämnen ägs av en annan teknisk lösning. Det kan ur supporthänsyn bli väldigt jobbigt att hålla koll på. Är den hanterbar, kan den supportas eller kan vi lösa det på ett snyggare sätt för att göra plattformen mer underhållbar. Vi försöker alltid göra en utvärdering. Kommer det in ett problem vi hade två år tidigare som vi löste med Publish-Subscribe tittar vi först om vi kan återanvända den vilket är ett SOA tänk men man återanvänder själva komponenterna och inte själva tjänsten. Eller ska vi bygga nytt? Om vi ska bygga nytt - vad behöver vi ändra? Behöver vi uppdatera något? Fungerar detta bra? Uppfyller den det vi försöker uppnå? Eller har den massa funktionalitet vi inte behöver?

I ett t.ex. Publish-Subscribe blir konsekvenserna oftast när nästa utvecklare kommer eller supporten kommer. Är det här klockrent för dem? Eller blir det för komplext? Exempelvis en Publish-Subscribe med två prenumeranter kan du lika väl lösa med en Fire-and-Forget, där du i princip gör två hårda linjer till dina mottagare. Det innebär att till nästa mottagare måste du dra ett streck till medan i en Publish-Subscribe behöver du bara skapa en prenumerant så har du din mottagare färdig. Det innebär också - genererar vi mer utrymme för en kommande mottagare än vad det kostar att underhålla lösningen. Man behöver hela tiden utvärdera.

Det kan ju t.ex. innebära att en kund jag har jobbat länge hos. Där känner jag igen avsändande system och tänker att det här meddelandet kommer flera förmodligen vara intresserade utav. Det innebär att istället för att göra en simpel Fire-and-Forget så gör jag en komplex lösning med Publish-Subscribe. Just för att jag egentligen ponerar att det här meddelandet kommer flera vara intresserade av. Alltså är det supersmidigt att i framtiden bara lägga till flera prenumeranter istället för att göra fler och fler flöden som skickar igenom samma meddelanden.

I: Har du upplevt kommunikationsproblem med kollegor inom integrationsutveckling på grund av skild terminologi?

F2: Pratar man med BizTalk-folk så heter ju Request-Reply – Request-Response vilket är väldigt irriterande. Sen just standardmönster är vedertagna. Nästan alla vet vad detta innebär och det är även de tre vanligaste man använder. Men går man ner på nod-nivå på EAI-mönster t.ex. där blir det svårt. Det skiljer sig också hur Enfo Zystems utvecklar, t.ex. avsändare, mottagare och här är ett flöde samt namnsättning. Det gör man nästan på alla kunder men på vissa kunder gör man också implementationsbilden. En översiktsbild med EAI-mönster och ibland med något annat osv. Problemet vi har där är att en lösning som görs och sedan uppdateras med ny funktionalitet ett år senare, man glömmer oftast att uppdatera dokumentationen för lösningsdesignen. För att vara helt ärligt, noderna i EAI-mönster är extremt gammalmodiga. Jag tycker att det är svårt att titta på de och veta vad man ska göra liksom.

Mönster i allmänhet ger ju egentligen ett scenario som är jobbigt att beskriva i flera ord. Att enbart kunna säga Publish-Subscribe så är alla med på terminologin, vad det innebär. Men om jag ska förklara hur vi jobbar med Publish-Subscribe så blir det ju genast mer komplext. För det finns ju...du kan t.ex. i ESB:n göra ett hårt ämne där du knyter ett helt objekt annars kan du göra den dynamisk. Alltså att du använder 'payload metadata' för att skapa ditt ämne. Då har vi genast två skilda terminologier, antingen knyter du ett ämne till ett objekt eller gör den dynamisk. Så jag menar att istället för att gå in på tekniska detaljer så kan man göra säga det här är en Publish-Subscribe lösning så är alla med på själva konceptet. Vad är det vi gör? Vad är det vi försöker uppnå? Hur fungerar det? Så absolut, det hjälper ju att förklara en komplex värld med ett ord egentligen. Det hjälper absolut till om alla är med på vad det innebär. Men jag har aldrig varit i en diskussion där någon har en skild uppfattning av vad Publish-Subscribe är gentemot mig. För konceptet är ju densamma på vilken plattform som helst även om teknologin bakom det är helt annorlunda implementerat.

I: Är mönstren du använder dina egenutvecklade, verksamhetens gemensamt uppsatta mönster eller är det kunddirektiv?

F2: Nej, mönstren vi använder är ju mer best practices. Tidigare så hade man inte så mycket dokumentation om hur man implementerade integrationer på det sättet vi gör. Men för att...vi gick från att vara 20 utvecklare till ett fåtal och upptäckte att för varje ny utvecklare som kom in igen så var det svårt att få de att arbeta såsom vi gjorde hos kunden. För att egentligen...återigen linjeforma utvecklingen så satte vi egentligen tre mönster. Dessa är mönstren vi jobbar med, så här implementerar vi mönster ett, två och tre samt de här förkortningarna kan du använda. Det innebär egentligen att du bara har strömlinjeformat din utveckling till att bli mer generisk så att när utvecklare nummer sex sen kommer, så kan han titta på en integration och fatta att det här är en Request-Reply för att vi har de här två artefakterna. Alltså att sammankoppla allt.

Jag skulle inte säga att de är verksamhetens eller kunddirektiv. På de flesta kunder så anlitar man

ju oss för att vi ska lösa deras huvudvärk. Det innebär ju att vi får jobba fritt. Då försöker man ju oftast att gå back-to-basics på något sätt. Hela vår kund X:s ESB är byggd från början till slut av oss. Det är inte en enda annan utvecklare som har varit därinne förutom en.

Det främjar återanvändbarhet och effektiviserar men vi ska ha i åtanke att detta inte främjar ett SOA-tänk utan vi har SOA i bakhuvudet. Därmed återanvänder man inte tjänster på det sättet som i SOA. I SOA innebär det egentligen du tjänstefierar säg ett legacy-system och så har du tjänster mot legacy-systemet och så kan du använda dessa tjänster för att utföra operationer. Vi har mer återanvändbarheten i utvecklingsdelen. Alltså har vi ett Request-Reply scenario så har vi ett färdigt skelett för att just producera en Request-Reply integration. Så på det sättet har vi ju återanvänt konceptet men vi har inte återanvändbarhet i själva tjänsterna. Vi har inte så många tjänster på de kunderna, som sagt.

I: Vilka positiva och negativa aspekter ser du vid användningen av mönster?

F2: Alltså på rak arm kan jag inte säga några negativa. Det du egentligen gör är att du namnsätter någon slags standard för det du jobbar med. Så länge alla förstår det man pratar om så är det ju bara positivt. Tidsbesparande, huvudvärksbesparande, man fattar direkt vad man vill uppnå.

Vi har haft problem med att man har använt mönster men med fel teknologi. Som i ett Request-Reply scenario använder man ett aggregation scenario istället. Så istället för att lösa ett simpelt problem så gjorde man en väldigt komplex lösning. Sen kom nästa utvecklare och tyckte att det mönstret fungerade ju. Då återanvände man det mönstret. Så man löste ett problem med fel mönster så att säga. Istället för att använda ett Request-Reply mönster använde man ett aggregationsmönster vilket är något helt annat. Men det uppfyllde ju samma krav bara det att det blir väldigt, väldigt komplext.

Man kan säga att oftast i ett Request-Reply har du ett tidspann, vilket betyder att det är tidskritiskt. Någon ställer en fråga oftast i ett gränssnitt och förväntar sig ett svar. Då pratar vi om någon sekund högst. I ett aggregeringsförfarande så aggregerar du ihop information vilket betyder att det tar lite längre tid och när du har mindre integrationer med inte så många användare så fungerar det lika bra. Men kund X:s ESB har växt explosionsartat, vi pratar 40, 50 miljoner meddelanden varje månad. Så det är väldigt hög trafikmängd och det innebär att går den två sekunder långsammare än den brukar så är det superdåligt. Så vi bytte ju t.ex. den här aggregeringslösningen och fick ner svarstiden med 400 % bara genom att byta mönster. Så det är kanske det negativa i så fall. Om man inte är tydlig med vilka mönster som löser vilka problem så kan problemet bli det här. Någon kommer och ser vad man har gjort och återanvänder det om och om igen istället för att lösa problemet på riktigt, egentligen. Det är väl också ett bra exempel på någon, någon gång hade för mycket tid och ville inte göra det enkla utan ville göra det svåra. För det är en cool lösning men den var dålig ju.

I: Resulterar användningen av mönster i ett mer kvalitativt resultat eller komplicerar det utvecklingsprocessen?

F2: Jag skulle aldrig säga att det komplicerar utvecklingsprocessen, snarare tvärtom. Oftast om du sitter i kammaren och ritar på lösningar så är det ju lätt att komma på coola, komplexa lösningar. Men jag tycker det är viktigt att hela tiden titta på vad är det vi försöker uppnå? Vilket problem är det vi vill lösa? Om du har en mönsterbas så är det ju väldigt lätt att läsa igenom ditt problem, titta på dina mönster och tänka "Den här, den fungerar klockrent".

Sen kan det vara så att veckan efter så har det kommit ett nytt krav som gör att du byter mönster. Men det är ju fortfarande rent tekniskt lätt att hoppa mellan mönster. Alltså att det ger ju en självrannsakan på något sätt att göra rätt redan från början. Så länge man utgår ifrån att, antingen kunden använder de här standardmönstren eller "om man bortser från de så kan du använda de här också men då ska du tänka på detta och detta". Så jag tycker inte det komplicerar utan snarare ger ett bättre resultat i slutändan.

Bilaga 5 – Transkribering av intervju 3

F3 = Ålder: 32
Utbildning: Magister i Datavetenskap
Antal år inom integrationsbranschen: 8år

I = Intervjuperson

I: När ni integrerar system, har ni i åtanke att uppnå en Service-Oriented Architecture?

F3: Det är ingenting vi sysslar med för det krävs mycket för att uppnå en sådan arkitektur. På de kunderna jag har varit på så har det inte varit aktuellt på integrationsnivå.

I: Enligt källor är ett mönster en beprövad best-practice metod för att lösa ett vanligt förekommande problem. Vad är ett mönster för dig?
F3: Ett mönster för mig är ganska generellt nog för att kunna användas och återskapas för att lösa flera sorters problem. Det är liksom ett mönster för mig – jag kan se, ok den här typen av problem återkommer flera gånger om, då kan jag använda den lösningen för det problemet och det problemet istället för att återuppfinna hjulet. Mönster kan vi använda i vissa fall t.ex. vid Routing. Du får in viss information som ska skickas till flera olika håll och istället för att hårdkoda något så kanske du gör det på ett annat sätt. Mönster använder vi och det tycker jag är bra.

I: Har det hänt att du har upptäckt i efterhand att du har använt ett mönster?

F3: Nej, det är väl sällan man upptäcker det i efterhand. För man försöker nog förstå det du vill ta fram redan från början för det är så svårt att göra sådant här i efterhand. Väldigt ofta vet du redan från början ok den här typen av problem det har vi löst innan så vi kan använda det igen eller det här är helt nytt.

I: Är det erfarenheten som avgör då?

F3: Ja, det spelar roll på erfarenhet, det spelar roll på - om man jobbar i team så de andra känner igen det, om man jobbar ensam, vad det är för kund osv, så det är väldigt många faktorer som spelar roll där.

I: Vilka är de tre vanligaste mönstren som du använder?

Jag har ju inga specifika namn men vi kan väl säga att ett av de vanligaste är ju så kallad Routing. Att man försöker routa information från en plats till en annan. Då försöker man t.ex. inte hårdkoda saker utan du kanske använder konfigurationsfiler eller liknande för att byter du ut ett system och lägger till ett annat så ska det gå smidigt att koppla in det. För olika typer av Routing-mönster är väldigt vanligt. Det är svårt att svara på för vi tar ju inte fram någon produkt så det blir svårt att använda de typerna av mönster som finns inom produktutveckling

I: Vi tänker oss de vanligaste som används på arbetsplatsen, här på Enfo?

F3: Jag har svårt att kalla de för mönster, för att de är såpass generella. Fire and Forget säger bara vilken typ av integration det är. Fire-and-Forget och Publish-Subscribe säger bara vilken typ av mönster men jag skulle nog inte kalla de för mönster utan snarare vilken typ av integration det är. Om man skulle kalla de för mönster så har vi använt Fire-and-Forget, Publish-Subscribe och vi har använt Web Services också.

I: Det är på den nivån och typen av mönster vi är på.

F3: Ja, ok. I så fall har vi använt alla tre i alla fall.

I: Hur vet du vilka mönster som är mest lämpliga för integrationen?

F3: Väldigt ofta har du ju krav som kommer till oss så till exempel om ett system säger att vi kräver ett svar tillbaka, då vet man kanske att när vi skickar något att vi vill vi ha ett direkt svar tillbaka och då blir det kanske ett Fire-and-Forget.

Eller så säger kunden att vi vill skicka ett meddelande men vi vill att det skickas till tio olika personer, då blir det Publish-and-Subscribe istället. Så det beror mycket på kraven, både på de tekniska- och affärskraven.

I: Hur går du tillväga för att lösa ett integrationsproblem med, respektive utan ett tydligt mönster?

F3: Väldigt ofta börjar man med att titta om det finns färdiga, liknande eller tidigare lösningar. Har du löst det tidigare då försöker man återanvända det. Har du inte löst det tidigare då försöker man se vad det är för typ av problem, finns det något som liknar den för då kanske man kan återanvända den. För det är ju väldigt sällan du stöter på något som är helt nytt. Man försöker alltid hitta något som liknar och går på den vägen.

I: Om ett mönster hittas som har applicerats tidigare, tar ni hänsyn till i vilken kontext det har gjorts? Samt kontrollerar om lösningen kan uppenbara andra problem i ett senare skede och passar det ert specifika problem?

F3: Den kontexten som tas mest hänsyn till det är nog vilken typ av information det är. Nu menar jag mönster här. Jag menar inte om det är Fire-and-Forget, Publish-Subscribe, utan mer hos den kunden jag sitter hos. Den kunden jag sitter på så är mönstren mer vilken typ av information som skickas och från vem det är. Då försöker man göra så att den typen av information och den typen av system går igenom samma sorts flöde. Då blir det ett mönster för det systemet, plus den typen av information.

Så att nästa gång man vill berika den typen av information då använder man samma flöde igen. Kontexten här är system och informationstyp.

I: Tar ni i beaktande problem som kan förekomma ifall man skulle använda ett visst mönster?

F3: Ja, det brukar vi göra. Väldigt ofta innan man börjar koda någonting så brukar man ha en diskussion om vi ska använda den här lösningen, vad är det för risker, vad finns det för problem? Kan vi ens använda den? Kan vi inte använda den? Det är först då när vi tänker ok det skulle kunna gå, då använder vi den.

I: Den här diskussionen, vem har du den med?

F3: Är det ganska komplicerat brukar man ta det med arkitekten. Är det något utvecklaren kan fixa själv så går vi inte igenom arkitekten.

I: Med problem i ett senare skede menar vi om en ny utvecklare skulle komma in i projektet och

kolla på en lösning. Om du tänker på nästa person som ska komma in i projektet?

F3: Det är ganska sällan det inträffar. Jag har nog inte varit med om det. För att mönstren inte ska vara enkla men tillräckligt enkla för att när nästa person tar över så ska det vara lättförståeligt. Det är ju hela poängen med ett mönster. För om du gör den för komplicerad så slutar den ju vara ett mönster.

I: Har du upplevt kommunikationsproblem med kollegor inom integrationsutveckling på grund av skild terminologi?

F3: Inte inom integration om jag t.ex. pratar med en annan integratör. Men kommunikationsproblem kan ju uppstå om du pratar med kollegor som är utanför integrationsvärlden då kanske man använder samma ord men menar olika saker.

I: Kan det vara supporten då?

F3: Nej, inte supporten men det kanske är från de som ställer kraven eller någon som kommer från en annan del av IT osv. de är inte insatta inom integration. Då kanske de ser det på ett annat sätt och vi ser det på ett annat sätt.

I: Anser du att den här förståelsen som användandet av mönster bidrar till en bättre kommunikation?

F3: Såklart, det allra bästa vore ju om en kund som behöver hjälp med en integration känner till dem. Då blir det plötsligt mycket enklare att kommunicera men det är väldigt, väldigt sällan att det är så.

I: Sinsemellan kollegor då?

F3: Jag har nog inte varit med om kommunikationsproblem mellan en integratör. Vi skulle nog inte ha problem på det sättet att kommunicera.

Jag skulle nog tro att det svåra med att förklara vad en integration är kanske inte vilket mönster den tillhör som t.ex. Fire-and-Forget, Publish-Subscribe eller något mer. Det svåra med att förklara en integration det är, vad för typ av information som kommer in och vad det är för typ av information som kommer ut ur ett informationsperspektiv och vad är relationen mellan input och output. Det är nog komplicerat att förklara när det kommer en helt ny utvecklare. Jag tror inte det är på Fire-and-Forget eller Publish-Subscribe det svåra sitter.

I: Är mönstren du använder dina egenutvecklade, verksamhetens gemensamt uppsatta mönster

eller är det kunddirektiv?

F3: Jag skulle säga att det är dels egenutvecklade och dels kunddirektiv. För vissa mönster måste du använda enligt kundens önskemål. För dels har du ju önskemål hos kunden och dels tekniskt så det är de kraven de har och då får man följa de hänvisningarna.

Sen när det kommer till projektet så finns det inga och då är det nya saker. Men då får du utveckla de själv så det är liksom en blandning av egenutvecklade och kunddirektiv. Jag skulle säga att det är alltid bra om kunden har några mönster som är grundläggande. Sen är tillräckligt flexibel för att låta egenutvecklade mönster att läggas till. Ett mönster måste liksom på nått sätt ändå kunna utvecklas för att kunna hänga med.

I: Vilka positiva och negativa aspekter ser du vid användningen av mönster?

F3: Jag skulle kunna säga att positiva aspekter är att om det används på rätt sätt så kan vi spara väldigt mycket tid. Det gör saker och ting mindre komplext, kommunikation gällande vissa typer av problem blir enklare och man kan återanvända mycket av koden som man skriver. Så kan man dela in problem i beståndsdelar som är enklare att förklara.

Negativa aspekterna är kanske att man kanske försöker skapa mönster där det inte finns mönster utan man skapar det bara för sakens skull. För man använder en mönsterfälla. Allt ska handla om mönster så man måste liksom använda sunt förnuft. Var är det lämpligt att använda mönster och ser man i framtiden att vi kommer att stöta på det här problemet igen, ok, då kanske vi ska använda mönster här. Eller är det här bara en engångsföreteelse då kanske man bara ska göra det för den här gången och inte liksom hålla på och skapa mönster för det kanske blir mer arbete för ingenting.

I: Resulterar användningen av mönster i ett mer kvalitativt resultat eller komplicerar det utvecklingsprocessen?

F3: Jag skulle nog säga att det beror på i vilket stadie det är. Första gången du ska skapa ett mönster så bidrar det ju till ett kvalitativt resultat och komplicerar utvecklingsprocessen. Men andra, tredje och fjärde gången så är förhoppningsvis den komplexiteten i utvecklingsprocessen mindre och det kvalitativa resultaten ska förhoppningsvis vara densamma. Så komplexiteten är nog mest i början, mindre senare medan den kvalitativa ska ju vara samma hela tiden.

I: Några andra frågor eller funderingar?

F3: Jag tänkte bara nämna att väldigt ofta när det gäller mönsterintegration har det att göra med väldigt specifika system som du använder.

Till exempel om du använder SAP eller om du använder Movix, ett annat system eller om du använder Amazon eller… osv. Väldigt ofta är de här systemen såpass stora och att folk använder de väldigt ofta då kanske du gör ett specifikt mönster för de systemen. Så det blir väldigt systemanpassat snarare än om det är Fire-and-Forget, Publish-Subscribe. Publish-Subscribe är på väldigt hög nivå så oftast när det gäller integration, så brukar man oftast beskriva mer specifikt vilken typ av mönster det är. Bara man har det i åtanke när man pratar mönster.

I: Är dessa egenutvecklade då?

F3: Precis, det blir ju väldigt egenutvecklat. För skillnaden är ju t.ex. inom programmering om du har ett mönster då kan du oftast uppnå det mönster sen om du skriver det i Java, C# eller PHP det är bara språket man använder men du vill ju få till samma typ av mönster. Men inom integration så beror det väldigt ofta på vilken typ av produkt du använder för att få fram den lösningen till det mönstret.

Bilaga 6 – Transkribering av intervju 4

F4 = Ålder: 28 år
Utbildning: Kandidatexamen i Systemvetenskap på Lunds universitet och master i Stockholm. Antal år inom integrationsbranschen: 3 år

I = Intervjuperson

I: När ni integrerar system, har ni i åtanke att uppnå en Service-Oriented Architecture?

F4: Mm, det är liknande som jag har svarat på tidigare när ni har frågat, men det jag sa då var ju att inte till punkt och pricka att försöka följa några SOA-regler men ja till att försöka bygga löst sammankopplande komponenter som går att återanvända. Känns det som delvis.

I: Vi tänker även på tjänsteorienterat, att allting kanske måste tjänstefieras.

F4: Ja, alltså jag har väl det åtanke fast det går inte alltid. Det beror på vilken arkitektur som finns från början och när. Ibland försöker jag väl att föra in det där jag kan också, lite mer tjänstetänk. Men det är inte alltid möjligt.
I: Enligt källor är ett mönster en beprövad best-practice metod för att lösa ett vanligt förekommande problem. Vad är ett mönster för dig?

F4: Jag skulle säga att ett mönster är en konceptuell beskrivning av hur ett problem kan lösas, alltså inte i detaljnivå, hur man löser problemet. Men på vilket sätt man ska tänka för att lösa det. Så i detaljnivå är det olika beroende på vilken produkt som används, det kan ingen säga som inte kan produkten till punkt och pricka. Men säger man ett mönster så förstår man på det stora hela hur integrationen är tänkt att fungera. Det är så jag skulle beskriva ett mönster.

I: Vi har även en följdfråga, har det hänt i efterhand att du har upptäckt att du har använt ett mönster?

F4: Inte i direkt följd utan jag har nog blivit ganska medveten om jag ser eller bygger någonting att ja, det här är ett sådant mönster och det här är ett sådant mönster, om man nu är lite inläst så tror jag att det blir så automatiskt. Men jag kan såhär i efterhand se att när jag började integrera att jag implementerade vissa mönster då som jag inte var medveten om men det kan jag ju tänka på nu att då gjorde jag det och det.

I: Så det var mer i början på grund ut av att man inte riktigt hade erfarenhet?

F4: Ja, jag hade inte riktigt koll då. Men det är inget konstigt med det. Jag visste bara inte riktigt

hur mönster fungerande och hur man applicerade de inom integration. Men nu känner jag att jag har koll på det. Så nu finns det ju i åtanke men jag kan också se tillbaka, här gjorde jag det, samma sak som jag vill göra nu och på så sätt så kan jag också få lite idéer om hur jag ska lösa den.

I: Vilka är de tre vanligaste mönstren som du använder?

F4: Den är lite svår att besvara med tanke på att det finns väldigt många olika typer av mönster. Jag håller ju på med två stycken integrationsprodukter, BizTalk och IBM-Sterling, rent arkitektoniskt i de här så är ju BizTalk... hela BizTalk är byggd på en Publish-Subscribe arkitektur som i säg är ett mönster och det går inte att komma ifrån. Så fort du gör någonting i BizTalk så implementerar du Publish-Subscribe, för det är det den gör. Sen inom Sterling, nästan hela Sterling-Sverige, alla företag i Sverige som använder Sterling har haft några få integrationsarkitekter från IBM eller Sterling-Commerce hos sig och där har man i de flesta fall implementerat någon Point-to-Point arkitektur. Alltså publicera eller ta emot ett meddelande och skicka till en mottagare, bara en rak linje igenom, så där är också ett mönster som man inte riktigt kan komma ifrån om man inte bygger om hela lösningen. Så de två är jättestora för mig, Publish Subscribe och Point-to-Point, sen där nedanför finns det ju massa saker som jag använder hela tiden. Till exempel Content-Enrichment gör jag åtminstone någon gång i veckan, Message Transformation jätteofta, alltså att sitta och mappa. Request-Reply använder jag ofta när jag bygger webbtjänster. Så det blir fler än tre, men de två stora är rent på arkitektonisk nivå, Point-to-Point och Publish-Subscribe. Det är de jag kommer i kontakt med dagligen.

I: Begränsas man av det på något sätt?

F4: Jag upplever det inte så för att man gräver in sig i lösningarna, man blir väldigt familjär med lösningen på något sätt och då kan man hitta vägar att kringgå det. Jag kan tycka att när jag började med BizTalk att det var lite begränsande för den har inte alls samma öppna arkitektur och samma möjligheter out of the box som Sterling har utan det är ofta att man behöver fundera en gång extra och kanske skriva lite egen kod för att lösa problem. Men det går att komma runt det och sen att man bara använder Publish-Subscribe behöver inte begränsa, man kan ju. Det begränsar inte att andra mönster används, för att i grunden om man har ett meddelande som tas emot så är det någon som sitter och lyssnar efter det här meddelandet som prenumererar det alltså. Men det funkar på samma sätt tillbaka om man vill ha ett svar, en Request-Reply, då blir det egentligen en Publish-Subscribe och båda håll. Så man kan ha både och, det är bara att just Public-Subscribe kommer man inte ifrån. Man kan inte skicka ett meddelande och tvinga de att ta emot det.

I: Hur vet du vilket mönster som är mest lämpligt för integrationen?

F4: Jag tror inte att det är något mönster som är mest lämpligt egentligen utan det finns ett alternativ och använder man inte det så har man gjort fel. Det finns alltid en integrationsspecifikation om det nu är på papper eller om någon berättar, men om någon säger att de bara vill skicka en fil från en yta till en annan, man kan ju inte implementera ett Request-Reply flöde då. Det blir bara fel, det är inte det som efterfrågas utan det är en sak som är rätt eller fel. Det är inte så mycket överlappande där egentligen. Sen hur jag vet vad som är mest lämpligt är väl vad som är rätt då.

I: Vi tänker om det är så att du jämför olika mönster kanske, om det är något som ser liknande ut?

F4: Jag kan inte säga att jag jämför olika mönster utan det är nog mest bara egen analys i huvudet utan att tänka så mycket, funkar det här mönstret för det här eller funkar det här mönstret på beskrivningen. Det är inte riktigt så jag tänker utan det är nog mer eftertänksamhet och analys.

I: Beror detta på erfarenhet, att vet du vilken lösning som kan appliceras eller vilken som är mest lämplig?

F4: Ja, jag ser ju vilket mönster som det hela blir. Det behöver inte betyda att jag utgår från att jag ska bygga ett visst mönster, ibland kan det hända att ja, jag ska göra såhär och det blev visst ett sådant mönster. Man går andra vägen. Det behöver inte alltid vara så att man utgår från mönstret i sig utan det kan...

I: Det kommer efterhand?

F4: Ja, precis.

I: Hur går du tillväga för att lösa ett integrationsproblem med, respektive utan ett tydligt mönster?

F4: Ja, utan blir väl lite som jag beskrev i föregående fråga, ren eftertänksamhet och analys. Jag tittar efter vad beställaren vill och frågar, vad vill du ska hända. Det är inte så mycket mer med det egentligen utan det är bara att fundera och analysera. Med ett mönster så är det väl egentligen samma sätt bara att jag har en grund att gå från snabbare, att jag kan förstå lite snabbare vad de är ute efter men det är samma tankeverksamhet. Att de till exempel säger att jag vill ha att det här ska hända och det är ett Fire-and-Forget, då vet jag på en gång att de ska ta emot något och bara skicka iväg det, sen bryr de sig inte så mycket. Då behöver jag inte fundera om de vill ha någon bekräftelse på vad som hänt, då är det vad som har hänt, då vet jag det.

I: underfrågan där är, var du tar hjälp ifrån, är det via nätet eller är det kollegor eller är det bara du själv som tänker på det?

F4: Var jag tar hjälp ifrån i vilket sammanhang?

I: Vi tänker när du får den här integrationsspecifikationen, om du inte kan det själv, vart du tar hjälp ifrån. Du sa tidigare att om det inte fanns ett mönster så bedriver du en analys och tänker efter. Men vi undrar om du tar hjälp av något annat ut över det, du kanske kollar någonstans?

F4: Alltid Google, jättemycket. Frågor till beställare och intressenter, det behöver inte alltid bara vara beställaren som vet hur det ska vara, det kan vara användare som är involverade och verkligen ska använda integrationen. Testande och experimenterande. Men verkligen jättemycket Google.
I: Om ett mönster hittas som har applicerats tidigare, tar ni hänsyn till i vilken kontext det har gjorts? Samt kontrollerar om lösningen kan uppenbara andra problem i ett senare skede och passar det ert specifika problem?

F4: Vad menar ni med vilken kontext?

I: I vilken kontext det har gjorts, alltså enligt litteratur och källor som vi har gått igenom så finns det en struktur för ett mönster och det har till exempel implementerats i den här kontexten och har fungerat i den här miljön, kommer det fungera i min miljö. Kollar ni på det eller kollar ni bara på lösningen eller kollar ni på kontexten runt omkring? Som vi har förstått så finns det oftast olika konsekvenser för att implementera ett visst mönster och vi undrar om du kollar på den stora bilden? Vi antar att ni kollar på det här med problem och tar hänsyn till det.

F4: Det här tror jag beror lite på vilken ansvarsnivå jag har, mitt uppdrag med BizTalk just nu, där sitter jag bara som utvecklare och vi har arkitekter som tar fram specifikationer och funderar på det stora hela. Så där lägger jag inte så mycket tid på det, för att det är utanför mitt bord helt enkelt. Skulle jag börja göra det, det skulle ta för mycket tid. Jag skulle inte hinna göra mina vanliga arbetsuppgifter då, jag får bara förlita mig på att de gör det. Där jag sitter med Sterling nu, där har jag betydligt mer ansvar och då funderar jag absolut på konsekvenser av användandet, kanske inte av mönster men av lösningar vilket ju är en implementation av mönster i och för sig. Men det är inte så att jag gör någon större analys av det hela, utan jag tänker vad händer om jag kör, kommer det bli något fel någonstans.

I: Men du tar hänsyn till det i alla fall?

F4: Ja, det gör jag. Men jag ska inte säga att jag gör det i ett jättemycket, men tanken finns där.

I: Om lösningen skulle kunna uppenbara problem i ett senare skede och då menar vi om någon annan skulle komma in eller om det är fler mappningar som ska göras och fler prenumeranter eller något liknande. Att det här mönstret passar i den här lösningen men i framtiden om systemet ska expandera eller minimera kommer det här fortfarande fungera då om en ny utvecklare kommer in, kommer han förstå det här. Sådana småsaker som kan uppstå som problem senare. Alltså om man tänker på det här när man utformar lösningen?

F4: Ja, absolut det måste man ta hänsyn till. Just det där med annan utvecklare, då får man nog snarare förlita sig på bra standarder och bra dokumentation för att få de att förstå. Men skalbarheten som ni nämner där om systemet växer eller om antalet transaktioner går upp eller ner. Det finns i åtanke, ibland har jag varit tvungen att göra något som jag inte vet riktigt kommer fungera om det växer för mycket och det har känts lite illa.

I: Har du upplevt kommunikationsproblem med kollegor inom integrationsutveckling på grund av skild terminologi?

F4: Ja det har jag nog, fast mer när jag var ny inom området. Eftersom att jag inte hade riktigt koll på hur terminologin var. Sen ja det finns ju även senare, jag sitter ju ensam med båda de integrationsplattformarna jag håller på med i Malmö och det är då ofta broker specifik terminologi som kommer upp eftersom jag tror nästan alla andra kan det till lite mer i alla fall. Men inte jag så då är det lite svårt för mig att förstå.

I: Underfrågan är: anser du att mönster kan bidra till en gemensam terminologi på arbetsplatsen?

F4: Ja, men det kan inte lösa det helt och hållet. Men det kan bidra till en gemensam terminologi när man pratar koncept, större perspektiv. Men som jag sa tidigare så fort man går in i minsta bit, hur man ska lösa det på mer detaljnivå, då försvinner den här fördelen med en gemensam terminologi för då kan man inte riktigt prata i mönster längre utan då måste man prata på en annan nivå.

I: Vi tänker att det kanske är lättare att kommunicera genom att säga Fire-and-Forget än att förklara hela biten, det täcker allt med ett ord istället.

F4: Absolut, det håller jag med om.

I: Du sa det här med att det uppstod problem med termer inom broker:n. Hur gör du för att lösa det? Hur lång tid tar det?

F4: Ja, alltså det är inte så ofta jag behöver förstå. Det uppkommer väl oftast när jag pratar med någon annan som har problem eller under möten, men jag behöver inte förstå det egentligen så

jag lyssnar lite och försöker förstå så gott jag kan sen låter jag andra lösa det hela. Så det kanske inte är ett jättestort problem egentligen, utan det är snarare besvärande att inte hänga med alltid. Men arbetsmässigt kanske det inte är något problem eftersom jag inte behöver lösa problemet.

I: Sen har vi ju en underfråga till det också: anser du att detta bidrar till en mer effektiv kommunikation mellan kollegor, att man kan kommunicera det på samma vis.

F4: Ja, det tycker jag definitivt.

I: Är mönstren du använder dina egenutvecklade, verksamhetens gemensamt uppsatta mönster eller är det kunddirektiv?

F4: Mönstren i sig är ju inte utvecklade på något sätt, det är ju generiska mönster som kommer från ja... jag tror nog de beskrivs EAI-patterns. Sen själva integrationen av de, det varierar ut ifrån projekt till projekt som jag har sagt tidigare, om man nu går till BizTalk projektet som jag håller på med, där har vi utvecklarriktlinjer som beskriver implementation av olika mönster. Hur vi går tillväga om vi ska aggregera flat-filer, hur gör vi då ett aggregat mönster helt enkelt.

I: Så då har du riktlinjer?

F4: Ja, då har jag riktlinjer. Sen är det inte en komplett lösning men jag har riktlinjer för hur vi ska göra det här. Det finns nog riktlinjer för de flesta i alla fall och det här är ju på projektnivå då.

I: Som vi har förstått så har ni inga i verksamheten, det är oftast på projektnivå.

F4: Ja, det är oftast på projektnivå och projektet kan ju bestå av en själv eller en större grupp. Består det av en själv så är det ju individuellt, hur man löser det.

I: Men där med egenutvecklade så tänkte vi om du tar fram egna mönster, vi har hört att någon har gjort det, att personen har tagit fram egna mönster och använt de och försökt sprida de.

F4: Jag är rätt nyfiken på det faktiskt, jag hörde det där också och undrar lite vad de menar egentligen. För jag vet inte hur man kan ta fram ett eget mönster riktigt och vad man försöker täcka in som inte finns redan. Men det är inget jag gör i alla fall.

I: Sen den här underfrågan, främjar det återanvändbarhet och en effektivare utvecklingsprocess? att det finns liksom.

F4: Att vad finns?

I: Att ni till exempel har riktlinjer i projekt för de här mönstren?

F4: Ja, det tycker jag definitivt det gör. Det är ju lättare för en ny utvecklare att komma in i samma projekt. Man behöver inte börja om från början utan det finns en standard för hur vi löser olika... hur vi implementerar olika mönster för den kunden i det projektet. Så ja, det främjar återanvändbarhet.

I: Vilka positiva och negativa aspekter ser du vid användningen av mönster?

F4: Ja, positiva är flera som vi har nämnt tidigare, gemensam terminologi, återanvändbarhet och som jag sa tidigare att man har en liten startbräda vilket medför att man snabbare greppar vad det är som ska utvecklas. Det är väl egentligen det som jag kan se som positivt.
Tittar man negativt så är det väl inte lika stora saker. Men jag kan väl ibland uppleva att man pratar mönster för mönstrets skull och då går man i fel riktning tycker jag, det är ju inte det man vill göra, man vill ju inte prata mönster för att de finns utan man vill ha hjälp av dem.

I: Man ska inte bara använda de för att de finns.

F4: Nä precis, det ska finnas ett syfte med det. Annars är det bara onödig administration och förvirring faktiskt. Jag kan också säga att i ett av projekten jag arbetar i nu så använder vi mönster för tidsuppskattning. Så att man egentligen har en lista med olika mönster och deras komplexitetsgrad så att en Message Transformation av en simpel komplexitetsgrad bör ta så många timmar och en aggregator av den komplexitetsgraden tar så många timmar. Det är ju också en positiv sak. Samtidigt som man kanske bör se en integration i sin helhet också ibland. Jag kan tycka att det är lite farligt att bryta upp det i så små beståndsdelar hela tiden för man tappar lite känslan för integrationen i sig. Ser man helheten så kan man kanske komma med estimat om en tid som skiljer sig ganska ordentligt från om att man bryter upp varje del för sig. Så det kan vara lite begränsande på det viset men positivt för att det går att göra, men risk för att det är begränsande

I: Resulterar användningen av mönster i ett mer kvalitativt resultat eller komplicerar det utvecklingsprocessen?

F4: Det är också några saker som vi rört vid lite tidigare. Ja kvalitativt, jag anser väl att just det vi nämnde om återanvändbarhet, inte återanvändbarheten i sig utan att man har, om man pratar implementation av mönster, inte konceptet mönster utan en implementation av mönster. Så anser jag det är kvalitativt för att man har en riktlinje för hur man ska implementera mönstret och då görs det på samma sätt hela tiden och då blir det en viss kvalité, för den här riktlinjen är förhoppningsvis bra och man kan se om en integration ser ut på ett sätt. Använder någon annan

samma mönster så vet man ungefär hur den ser ut även i ett senare stadium. De som underhåller lösningen helt enkelt får en bättre bild av det hela. Det tycker jag är en stor kvalitets höjning med användningen av mönster. Komplicerar, vet jag inte riktigt om det gör så mycket om man använder det på rätt sätt. Jag nämnde ju tidigare att man kan använda det för användningens skull men då gör man väl egentligen på fel sätt. Så jag tror inte det kan vara så komplicerande egentligen

Bilaga 7 – Transkribering av intervju 5

F5 = Ålder: 39
Utbildning: Kandidat i Systemvetenskap
Antal år inom integrationsbranschen: 10 år

I = Intervjuperson

I: När ni integrerar system, har ni i åtanke att uppnå en Service-Oriented Architecture?

F5: Det jag först reflekterar över är ju att SOA fortfarande mer ett säljbegrepp och inte en utopi, absolut inte. Men det är få som når dit om renodlar det att man tänker att man ska ha upptäckbara tjänster som är väl definierade och mer eller mindre automatiskt hittar andra tjänster som... hela den floran är ett jätte projekt att ta sig an och kunder når oftast inte ända fram. Sen kan man plocka russinen ur det och försöka designa så att man identifierar återanvändbarhet och det är väl där jag känner att vi kommer in, Zystems. Dels har vi en erfarenhet av hur man brukar göra och dels har vi... och då går det ganska mycket fortare och se vart man kan lägga krutet på åtminstone på återanvändbarhet eller vad man inte behöver lägga så mycket tid på.

F5: Förstår ni vad jag menar där?

I: Vi förstår vad du menar där, det är kanske väldigt komplext att införa i en verksamhet...

F5: Ja.

I: Hela det här men man kanske väljer att tjänstefiera vissa delar om det är möjligt osv.

F5: Sen finns det ju de som har lyckats och som verkligen... men då är det ofta erfarna organisationer eller att de har väldigt stark fokus eller väldigt starkt incitament för att göra detta, jag vet... vilka är det som har det här stora vårdsystemet. Det finns ju ett backbone i Sverige mellan alla landsting och regioner som ska utbyta vårddata, hela det projektet... jag kommer inte ihåg vad organisationen heter men Zystems är med på ett hörn och tar fram backbonet till den integrationsplattformen och där finns ju väldigt tydliga. Det ska ju funka väldigt generellt och där finns ju väldigt avgränsat och vad det ska kunna göra, det är väldigt viktigt att alla kan förstå det och vara med på tåget. Så där är det uppe på ett toppstyrt projekt som verkligen har incitamenten för att det här måste funka och det måste vara tjänsteorienterat. Det är inte bara som några normala kunder som har ett litet företag som köpt ett annat litet företag och så har de gamla system som måste kunna kommunicera med varandra också löser man det bästa man kan.

I: Enligt källor är ett mönster en beprövad best-practice metod för att lösa ett vanligt förekommande problem. Vad är ett mönster för dig?

F5: Nu sitter jag med rätt många integrationer som enbart är filförflyttningar och de löstes, alltså hos min nuvarande kund och de löstes på lite olika sätt. En filflytt kan man... alltså i en integrationsplattform kan du lösa samma problem på massa sätt men det blev lite spretigt och det var lite många konsulter inblandade. Så det som har gjorts där är att nästan alla filförflyttningar, de allra enklaste som ser likadana ut i princip, har kokats ner till att bli konfigurerbara tjänster i en ända integrations som egentligen bara består av två flöden som kan ligga och snurra lite någonstans i lösningen. Det är ju konfigurerbart, då är det ett slags mönster, men när jag tänker mönster så tänker jag mer att man har en idé om hur man löser ett visst problem. Så i det här fallet med filflytt så tycker jag nog inte att det är ett mönster, det är mer bara en återanvändbar tjänst för att den är så himla enkel. Det är bara en fil adapter egentligen, men i fallet med kund X för flera år sen så satt vi och... jag och en kollega... vi höll på med migreringen också fanns det en massa tjänster som var mellan ett system som anropades via en webbtjänst också gjorde den en massa hokuspokus för att hålla reda på en massa data innan man fick ett svar tillbaka från andra änden. Det var ett mönster som var alldeles för komplicerat, men då hittade vi egentligen på ett nytt mönster och där var det väl också då...och då visste vi på grund av erfarenhet att här finns 10-15 integrationer som gör precis samma sak på ett alldeles för komplicerat sätt, vi hittar på ett nytt mönster och delar av mönstret innefattade också sub-flöden så att man verkligen fysiskt återanvänder delar av det man har byggt, alltså delar av integrationen. Men mönstret där var ju just att identifiera...i det här fallet var mönstret ända ner till vilket system som pratade med vilket annat system, för att vi visste att det fanns en massa integrationer mellan de här två systemen och de pratade på väldigt olika sätt. Så då bestämde vi mer eller mindre... för att det skall ska gå snabbare att utveckla såklart, då bestämde vi att det ska se ut såhär, tjänsten ska gå via Http och det ska finnas en sån här sub i mellan och det ska loggas på det här viset. Då visste vi att i efterhand att det fanns en massa integrationer som gjorde samma sak men som var ganska spretiga och för komplexa. Det här kom i efterhand, av erfarenheten och vi hade fördelen att vi satt i ett uppgraderingsprojekt, alltså migreringsprojekt där vi bytte version på plattformen och då fanns det avsatt tid i projektet för kvalitets förbättringar av det här slaget. Så där upptäckte vi att det behövdes ett mönster egentligen. Så i efterhand, vi har inte upptäckt i just det här fallet, för fråga två på mitt papper står om det har hänt i efterhand.

I: Ja, i efterhand, att du har upptäckt att du har använt mönster. Vi tänker om det hände i början när man inte var så erfaren, att man då använde mönster utan att veta om det.

F5: Det tror jag är... så som jag ser på ett mönster, tror jag att det är väldigt svårt att använda ett mönster utan att man vet om det. För jag tror mer att...jo, man kanske löser likartade problem på samma sätt men då känns det som att man har mer tur i sådana fall. Jag tror mer att det är något som kommer med erfarenheten och det är det jag gillar med att arbeta på Zystems till exempel,

att det finns så himla många kollegor som är specialiserade på samma sak. De det finns alltid någon som har löst ett visst problem, då behöver man inte uppfinna hjulet igen. Där finns ju mönster som någon slags, inte officiellt dokumenterade men det finns en massa de-facto mönster som inofficiellt sprid kunskap och ju mer man lär känna personerna och kollegor, ju mer man arbetar tillsammans, desto mer vet man vem som kan ungefär vad. Jag tror att det är svårt att upptäcka att man använder mönster... jag bara tänker på hur det skulle, det känns som att mönster är något medvetet val när man hittar ett samband och börjar bygga efter det. Ofta krävs det någon slags erfarenhet och någon slags tanke om design i början.

I: Vilka är de tre vanligaste mönstren som du använder?

F5: Nuvarande kund är inte så mogen inom integration och de kringliggande systemen kring integrationsplattformen, så det vanligaste mönstret är väldigt grundläggande filförflyttningar och det är ganska rudimentärt. Det får vi kalla ett mönster även om det inte igår i den ända integrationen så är det mycket filflytt och FTP. Det är väl ett slags mönster och det är ingen transformering utan bara, ta den här, arkivera den, logga att vi har tagit och flyttat en viss fil och leverera den till den andra änden som också bara blir en mapp. Det händer inte så jätte ofta.

I: Point-to-Point då?

F5: Ja, verkligen. Point-to-Point kan ju vara i och för sig... Point-to-Point är väl skällsordet man använder när någonting inte är så återanvändbart eller generellt. I det här fallet är det både Point-to-Point och enbart filförflyttningar. Något annat mönster jag använder, jag försöker bara tänka här... nej, inte som jag har varit så duktig och identifierat som ett mönster.

I: Men använder standardmönster som tillexempel Fire-and-Forget och Request-Reply?

F5: Ja, okej om man ser de som mönster, så ja.

I: Det är den typen av mönster vi tänker på.

F5: Ja, okej. Jag förstår. Men de känns... när man säger Fire-and-Forget eller Request-Reply, det är mönster men de känns väldigt generella, det känns bara som ett sätt att...det känns inte som ett mönster för mig. Fire-and-Forget eller Request-Reply det känns bara som ett sätt att benämna eller att berätta vad en integration är för typ. Ett mönster för mig känns mer som att, det är mer tekniska detaljer som ingår i ett mönster när jag vill implementera något och jag känner att jag kommer göra 15 likadana saker. Då vill jag specificera ner till utvecklaren att han ska tänka på detta, gör detta, logga på detta viset, gärna... det är rätt mycket mer i ett mönster för min del för att det ska vara återanvändbart och för att man ska ha nytta av det. Så är det rätt mycket mer som ska specificeras så att man slipper tänka ut det varenda gång, Fire-and-Forget är ett mönster men

det är så himla hög nivå och det är ett jättebra sätt att kommunicera på när man använder de termerna, om du säger till någon Fire-and-Forget för den sändaren, det är inte så kritiskt att han behöver ha ett kvitto, han behöver inte veta hur det gick osv. Så jag tycker det mer är en term för att beskriva vilken sorts integration det är, för definitionen av mönster för mig är som jag nämnde mer tekniskt, alltså djupgående tekniskt, exakt hur man gör en viss... och det är kanske därför jag inte tycker att jag återanvänder några mönster för att det är svårt att hitta så identiska integrationer som jag menar är mönster. Det är väl därför jag inte tycker som sagt att... utom i det här specifika fallet med kund X där vi hade massa precis likadana integrationer som var mellan en web-shop som skulle göra en massa lager saldo på varor och reservdelssystem också var det lite olika integrationer. Det var liksom HTTP, Request-Reply över MQ och tillbaka till HTTP, och det var rätt mycket pyssel inblandat i de tekniska detaljerna, hur exakt det skulle se ut. Så då förstår jag vad ni menar.

I: Det de mönstren, inte just Fire-and-Forget eller Request-Reply, men det är på den nivån som vi har riktat in oss på.

F5: Ja, jag förstår. Jag har inte jättebra koll på EAI-patterns.

I: Hur vet du vilket mönster som är mest lämpligt för integrationen?

F5: Det är väl erfarenhet i sådana fall. Dels erfarenhet, för om kunden inte är så mogen så att den kan ställa kraven, att de som har behovet, om de inte vet att de till exempel behöver en Request-Reply. Då får man försöka... det är ju det som är arkitektens roll att fråga de efter deras behov och försöka kartlägga vad de ska ha det till också får man välja patterns, dels som båda systemen inblandande kan hantera för kan de inte lämna ett svar i andra änden då blir det svårt att ha en Request-Reply kanske. Dock kan vi i mitten säga att vi åtminstone har levererat någonting, vi vet inte vad resultatet blev. Så det kan ju halta beroende... det är ju väldigt mycket beroende på omständigheterna. Men generellt så baserar det sig på erfarenhet, om man upptäcker en situation och de beskriver vad de vill lösa, så säger man; har ni tänkt på detta, behöver ni inte veta när meddelandet har gått fram eller behöver ni aldrig veta det, är det kritiskt, tidskritiskt eller ja, det finns ju rätt många sådana saker. Ofta de mönster, om jag går ner till den tekniska nivån som jag pratade om tidigare, ofta de mönstren som... det är mycket sådana aspekter som spelar in angående vilket mönster man väljer. Även om det ska vara asynkront, synkront eller om man vill att det ska flöda jättemycket meddelanden men de får inte ta någon prestanda av integrationsplattformen, då får man ju välja ett visst mönster så att man ser till att man inte stoppar in 10 000 meddelanden som suger upp allting när det finns viktigare och högprioriterade meddelanden. Då finns det mönster rent tekniskt sätt för hur man implementerar för att den ska bromsas lite. Det är mycket sådant jag tänker på, när jag tänker på mönster.

I: Hur går du tillväga för att lösa ett integrationsproblem med, respektive utan ett tydligt mönster?

F5: Ja...

I: Du nämnde tidigare att det var kanske kollegor..

F5: Exakt.

I: Vart du tar hjälp ifrån och hur du går tillväga.

F5: Om jag inte har ett tydligt mönster... då hoppas jag ju... ja, bra fråga. Då får man testa sig fram oftast, och tillsammans med kunden om man inte har ett tydligt mönster och man inte vet. Det är ofta så det blir hos en kund för de inte har kraven klara för sig och man kan inte riktigt få all information. Så då bygger man någonting som hjälper lite löst, så att första beskrivningen av problemet också itererar man sig. Har kunden tid och råd med det så är det väl så det går till när man inte har tydliga krav, och det beror mycket på mognadsgraden hos kunden, men det är mycket trial-and-error och iteration för att..."oj nu behövde vi visst detta", så märker de när man testar första gången att i sitt system, "Jaha, nu vet vi ju inte hur det här blev", för det är ingen som har tänkt på hela processen och i mitten oftast... det är inte alltid som vi har koll på affärsprocessen. Vi är oftast ganska tekniska. Så det är inte alltid att det finns någon inblandad som har koll på helheten på processnivå och vet vad det kommer att kräva. Så det är ju svårt och det går ju inte att ställa några frågor till kollegor i det fallet för jag vet inte ens vad jag ska fråga om, för att jag inte känner ens till problemet som jag ska lösa. Är det tydligt med mönster så är det mycket lättare, då har man något att kommunicera kring, jag ska göra såhär och då kan man ju ställa frågan till exempel på vår mejllista internt. Men det förutsätter att man vet helheten och då har jag mycket mer nytta, för då hjälper mönstren mig att förklara vad det är för typ av grej. Men då måste jag veta vad det är och där hade jag nog haft mer nytta om alla... vi har aldrig pratat om det på Zystems men jag tror att om vi hade pratar mer EAI-patterns, så att de termerna också blev mer spridda, då hade det varit lättare att kommunicera kring. Alltså vissa lever ju och andas integration här, men jag som inte har det som hobby, jag kan inte det av bara pratet. Jag lägger inte jättemycket fritid på att läsa böcker kring integration, men vissa göra ju det i form av att läsa bloggar testar och implementerar på fritiden också. Det märker man när de kommunicerar så kan de teorin bakom på ett annat sätt. Så de har ju lättare att kommunicera sina frågor eller fynd, jag tycker det är svårt att ställa frågan till och med.

I: Om ett mönster hittas som har applicerats tidigare, tar ni hänsyn till i vilken kontext det har gjorts? Samt kontrollerar om lösningen kan uppenbara andra problem i ett senare skede och passar det ert specifika problem?

I: Just de här EAI-patterns som vi har fördjupat oss i har oftast element, namnet, vilket problem de är till för att lösa och vilka konsekvenser det medför osv. Så det vi undrar med den här frågan är om man tar hänsyn till detta.

F5: Ja, jag förstår. Nej, det tror jag inte om man pratar de specifika patterns som ni menar, för de har jag inte förhållit mig till och det var det jag sa var lite synd också. Man borde känna till de bättre.

I: Men ifall vi leker med tanken och vi säger att du hade haft erfarenhet av de, hade du tagit hänsyn till detta, alltså tittat på detta innan du valde att implementera ett mönster.

F5: Ja, är inte kontexten en del av parametern som bestämmer vilket mönster man ska använda. Så det måste man ju ta hänsyn till.

Vad menar ni med att kontrollera om lösningen kan uppenbara andra problem?

I: Där syftar vi på konsekvenserna för att använda ett mönster, då tänker vi om en ny utvecklare skulle komma in till ett projekt eller en kund, att de ska förstå det här mönstret ganska snabbt, bli införstådda med vad det betyder och vad det har löst för problem, även till exempel om systemet skulle expandera eller minimera, kommer det här mönstret orsaka problem då. Skulle något ändras i systemet, skulle det orsaka problem i ett senare skede.

F5: Sådant är erfarenhetsbaserat om inte annat. Det är det som sitter i ryggraden alltid, skalbarhet och konfigurerbarhet, hur generellt man kan göra det. Det är en av de största delarna i designen, att det går att skala. Det är givetvis en kostnadsfråga också, man kan lägga mycket tid på att skapa en jättegenerisk lösning, men gör man den för generisk, för generell och konfigurerbar, då blir det svårare att förstå för en ny utvecklare. Jag försöker... det är lite det jag kommer tillbaka till med SOA, den här stora planen. En nackdel med den stora planen är att det blir oftast så generellt att det blir en länge startsträcka och har vi mycket små problem som vi bara försöker hjälpa kunden med att bygga inte bara Point-to-Point kanske, men det blir ganska enkla integrationer. Då skjuter man sig lite foten om man dels lägger för mycket tid på att det ska bli generellt återanvändbart när man inte ens kommer behöva det. Så det är en avvägningsfråga, men det är där mycket av jobbet ligger i att, bedöma och avväga hur mycket det att behövas och hur mycket kraft ska jag egentligen lägga på detta. Vi sitter med det nu i alla fall, med folk som har uppfunnit världens bästa integration som används på ett ställe. Så ska man gå in på en liten detalj... det växte aldrig som man hoppades eller trodde. Så när vi ska gå in och ändra en liten sak, så tar det en halv dag att bara förstå vad den gör och vad det är jag ska förväntas ändra. De kan bli jättesnygga och jättestora men det är svårt för en ny person att förhålla sig till dem för att man har tänkt för generellt och för bra. Det kan ju vara bra om det behövs, men vi får inte bli för

duktiga där. Det är viktigare att de går att förstå det och att det är tydligt än att det är supersnyggt och rätt.

I: Har du upplevt kommunikationsproblem med kollegor inom integrationsutveckling på grund av skild terminologi?

F5: Ja, vi var inne på det innan. Jag tycker att problemen är väl att kommunikationen uteblir helt och hållit i och med att jag inte ens vet vad jag ska... hur jag ska... beskriva det. Jag kan dra mig för att... bland de som är duktiga så kan jag dra mig för att skicka ett mejl för att jag tycker inte att jag kan förklara mitt problem eftersom att jag inte kan teorin bakom tillräckligt. Jag är ganska erfaren rent hur man löser problemen för att jag har suttit mycket i det men jag har aldrig teoretiserat så mycket kring det så att jag kan termerna liksom. Det kan jag tycka är ett kommunikationsproblem, att jag ens drar mig för att kommunicera kring det.

I: Men som du sa tidigare, så hade du tyckt det var positivt att använda EAI-patterns för att kommunicera med andra?

F5: Ja, precis. Vi har rätt mycket kunskapsspridning på företaget kring saker och vi pratar mycket Baseline och där har vi tjänster, kontrakt och integrationer. Vi är rätt eniga om hur vi designar saker och vad vi menar när vi säger kontrakt osv, för det bankas in från början. Det skulle vara bra om antingen till exempel Baseline, det är kanske till och med så att det innehåller valda delar av EAI-patterns eller inte, men antingen får man berätta då vilka delar som är det. Jag tycker inte vi har pratat så särskilt mycket patterns, nej. Vi har snuddat vid det. Sen vet jag inte om det är för att man inte upplever att man behöver det, men det vet jag inte, alltså uppifrån.

I: Är mönstren du använder dina egenutvecklade, verksamhetens gemensamt uppsatta mönster eller är det kunddirektiv?

F5: I den mån att jag begränsar egenutvecklade till mig och min närmsta krets, typ skrivborden omkring eller de jag jobbar med hos kunden, så är det egenutvecklade. Kunddirektiv känns det inte som jag har haft några i mina uppdrag och verksamhetens var det jag nämnde nyss, jag tycker inte att vi pratar särskilt mycket patterns på företaget. Så egenutvecklade men inte bara jag utan egenutvecklade i projektet eller...

I: Då är det lite mer på teknisk nivå som du sa innan.

F5: Ja, precis. Men det är oftast hos kunden och tillsammans med en kollega som man pratar om det. Det har inte funnits någon större tanke, det är bara vi som försöker vara duktiga och bygga återanvändbara saker.

I: Har det funnits några direktiv på projektnivå?

F5: Jag har jobbat på ganska små kunder som inte har så mycket... i visa fall har det inte ens varit projekt. Det är bara löpande att försöka bygga integrationer. Så nej, inga sådana projektnivåer i mina fall.

I: Och underfrågan där, du sa också att ni har egenutvecklade med dina närmsta kollegor och då undrar vi såklart om det främjar återanvändbarhet och en effektivare utvecklingsprocess, eftersom att ni själva utvecklar det.

F5: Jaja, absolut. Det tycker jag verkligen och det är den nivån jag tycker när jag tänker mönster, just att effektivisera utvecklingsprocessen och återanvända tankarna helt enkelt då, det är hela poängen med de mönster vi tycker att vi bygger. Man har en ryggradsreflex när man kopierar och klistrar in för mycket så tänker man, nä vänta nu, nu får jag nog börja tänka efter om jag ska återanvända något eller försöka bygga någon modul eller någon funktion som kapslar in det här, för jag har redan kopierat och klistrat in det här fyra gånger och det är jag allergisk mot.

I: Vilka positiva och negativa aspekter ser du vid användningen av mönster?

F5: Positiva är typiskt som jag sa innan, jag tycker att... och i mitt fall... pratar vi era mönster, alltså den höga nivån, då tycker jag att det främjar framför allt kommunikationen som vi var inne på. Jag tycker att det hade varit lättare. Pratar vi mina mer tekniska mönster så främjar det hastigheten i utvecklingen och förståelsen, för att har man mönster som är uttalade och dokumenterade så kan man mycket lättare inom projektet säga det här är ett sånt eller det här är en sådan integration, det är samma typer. Då kan man förstå mycket snabbare, när man väl har förstått vilka typer det är så har man... då kan man mycket lättare förklara för en ny person och minska spretigheten i integrationen. En ytterligare positiv sak är hela förvaltningsprocessen och underhåll, för ju mer mönster man använder desto mindre spretig blir lösningen, alltså som plattform. Ju mer mönster man försöker tänka och förhålla sig till, man använder ett av ett visst antal givna, då blir det mycket enklare att hålla ihop det, det blir lättare att känna igen sig även för de som ska driftas, leta fel och supporten osv. Det blir mer strömlinjeformat eftersom att det inte blir så spretigt. Sitter man utan mönster och uttalade design patterns på den nivån så är det rätt lätt att duktiga utvecklare uppfinner hjulet igen. Då blir det så mycket komponenter, det blir så mycket olika sorters komponenter. En integrationsplattform är fortfarande i grunden bara en verktygslåda, du kan göra precis vad du vill, så det är ganska mycket upp till oss och vår erfarenhet att bygga rätt liksom. Då är mönstren, i alla fall när man är tillräckligt med många i ett projekt så riskerar det annars att bli spretigt eller i mitt fall med den kunden jag har nu som har bytt resurser ganska ofta från vår sida, då har det inte blivit så mycket kontinuitet, det har istället blivit ganska spretiga lösningar. Det hade man kunnat undvika definitivt.

I: Negativa aspekter?

F5: Jag tror inte det finns någon utom ja, det jag nämnde innan var om man inte har erfarenheten och se... när det är något så litet och så enkelt så att man inte behöver lägga krut på det, jag menar om mönster ska... om mönstren i säg blir förstora så att man försöker applicera de bara för att man har ett vackert mönster, då kan det börja ta tid att bara försöka bygga något generellt. Man behöver ju inte uppfinna ett mönster för varenda... bara för att man har ett visst scenario så kanske man inte behöver dokumentera ner det som ett mönster. Det kan ju ta tid om det är den ända integrationen man någonsin gör. Däremot kanske man ska tänka till andre eller tredje gången man gör samma sak. Det är väl den ändå då, men jag tror att det inte är många som är så himla pretentiösa att de sitter och försöker att hitta på ett mönster, det skulle i alla fall kunna vara en nackdel. Ibland är det kanske bara ett jätteenkelt problem som ska lösas.

I: Du menar att man ska inte använde bara för att det finns?

F5: Nej, precis. Man får inte haka upp sig på att det måste finnas ett mönster som den här integrationen ska passa i och finns det inget mönster så måste jag hitta på ett nytt mönster och definiera det. Det tar ju väldigt mycket tid om det är någon pytteliten integration som bara ska göra något minimalt.

I: Resulterar användningen av mönster i ett mer kvalitativt resultat eller komplicerar det utvecklingsprocessen?

F5: Jag bara funderar på... jag tror att det blir mer kvalité, framförallt som jag sa i underhållbarhet och dessutom har du ett mönster så slipper du uppfinna hjulet igen, för uppfinner du hjulet igen så har du nya fallgropar och nya möjligheter till buggar. Det finns nya fallgropar hela tiden, om det är ett mönster så förbättrar du förhoppningsvis alla integrationerna som bygger på samma mönster när du upptäcker en svaghet i ditt mönster om vi nu pratar tekniska mönster och återanvändbara sub-funktioner osv. Rent tekniskt så kan de ärva samma sub-funktioner och återanvända de helt enkelt. Så jag tror bara det är bättre rent kvalitativt och som sagt underhållbarheten och förståelsen blir bättre. Det komplicerar...

I: Då tänker vi din nya kund som är ganska ny inom integration, där kanske det kan finnas någonting?

F5: Nä, jag tror bara om man lyckas hålla det till en nivå att man åtminstone använder det som begrepp, att man använder patterns som begrepp för att beskriva en integration så tror jag att… då märker kunden att vi har lätt att veta vad vi prata om. Vi klassificerar... det kan ha sin funktion i att man bara låter... vi känner oss tryggare med att vi vet vad det är vi prata om på något sätt och kunden sen märker att vi känner oss trygga med vad vi pratar om när vi

identifierar något. Då sätter man ord på det, ibland när man sitter och försöker beskriva något så låter det lite rörigt, man vet inte riktigt hur man... har man något att förhålla sig till så är det... då kan man väl peka de mot en lathund mot patterns och om någon är intresserad, det finns ju massa sådana, typ en A4 av hopkok av såklart sammanfattningar. Jag tror det finns en fördel, jag tror inte det komplicerar.

I: Det kan det används som, inte säljargument men för att göra kunden tryggare?

F5: Ja, precis. Jag kände lite nu när jag tänkte på det, just som säljargument kanske, men jag tror att det förenklar liksom. Man har vant sig vid det så har man något... när man ska prata om saker... för det är väldigt viktigt i min uppfattning att man förstår vad det är man vill åstadkomma. Om man pratar samma språk... fördelen med ett pattern utifrån definitionen och boken är att det är rätt mycket vettiga människor bakom som har jobbat rätt mycket som har tänkt ut hur saker brukar se ut och vad man kallar saker. Sen skulle väl hela fallgropen vara att man försöker applicera det på varenda litet scenario, då får man väl försöka ha någon slags fingertoppkänsla och baserat på sin erfarenhet, man kanske inte behöver klämma in varenda liten integration i ett pattern. Det är väl den ända fallgropen, men det är mer en personlighetsfråga. Jag tror inte det finns några direkta risker.

Bilaga 8 – Sammanfattning av intervjusvar

Forskningsfråga	Intervjutema och underfrågor	Sammanfattning av intervjusvar
På vilket sätt påverkar integrationsmönster en integrationsutvecklare vid systemintegration?	**Inställning till SOA vid integrationsutveckling**	
	När ni integrerar system, har ni i åtanke att uppnå en Service Oriented Architecture?	F1: Har inte det i åtanke särskilt mycket men anser att man bör ha detta. Har inte större arkitektoniskt ansvar och erfarenhet. F2: Har det inte i åtanke men i baktanke. Anser att det är komplicerat att tjänstefiera allt som i SOA-tänket utan försöker hitta delar att implementera som tjänster. F3: Sysslar inte med det, han anser att det krävs för mycket för att uppnå en sådan arkitektur. F4: Har i åtanke att försöka bygga löst sammankopplade komponenter som går att återanvända. Han menar att det inte alltid är möjligt att tjänstefiera allting då det oftast finns en befintlig arkitektur redan från början. F5: Anser att SOA är ett jätteprojekt att implementera och därmed svårt att implementera i de projekt han är involverad i eftersom kunderna ofta inte är så stora och tillräckligt mogna. Han strävar dock efter att designa så att man kan identifiera återanvändbarhet.
	Erfarenhet av mönster	
	Enligt källor är mönster en beprövad best-practice metod för att lösa ett vanligt förekommande problem. Vad är ett mönster för dig?	F1: Ett sätt att lösa en uppgift på bästa möjliga vis. Hade ingen erfarenhet sedan tidigare av mönster och har upptäckt at han har använt de i efterhand.

		F2: Anser att det är en best-practice metod. F3: Generellt nog för att kunna användas och återskapas för att lösa flera sorters problem. Har sällan upptäckt det i efterhand. F4: En ”konceptuell beskrivning av hur ett problem ska lösas” vilket inte är på detaljnivå. Anser att det ger en överblick över hur integrationen är tänkt att fungera och medger att i början av sin karriär visste han inte om vad för mönster han implementerade men nu i efterhand förstår han vad det var. F5: Anser att mönster som Fire-and-Forget och Request-Reply är väldigt generella och känns inte som mönster för honom, han anser även att de är på en väldigt hög nivå. Ett mönster för honom innehåller mer tekniska detaljer.
	Vilka är de tre vanligaste mönstren som du använder?	F1: Fire-and-Forget, Request-Reply och Publish-Subscribe F2: Fire-and-Forget, Request-Reply och Publish-Subscribe F3: Har inga specifika namn men anser att olika sätt att förmedla information så kallad ”Routing” är mönster han använder. Anser inte att Fire-and-Forget, Request-Reply och Publish-Subscribe är mönster utan mer vilken typ av integration det är. F4: Eftersom respondenten arbetar med specifika integrationsverktyg som MS BizTalk och IBM-Sterling stöter han på två, arkitektoniska mönster, huvudsakligen. De är Publish-Subscribe och Point-to-Point och även Request-Reply som han kommer i kontakt med dagligen. F5: Point-to-Point**,** filförflyttningar, Fire-and-Forget och Request-Reply. Dock har han inte så stor erfarenhet av EAI-patterns.

Hur vet du vilka mönster som är mest lämpliga för integrationen?	F1: Utreder beroende på kravspecifikationen och bestämmer sig utefter den. Uttalar sig om att det beror på erfarenhet och ibland resulterar detta i speciallösningar. Frågar även arkitekten om oklarheter uppstår när det inte finns ett specifikt mönster. F2: En kombination av att försöka förstå vad det är man försöker uppnå och erfarenhet. F3: Anser att det beror på vilken typ av krav det är. Avläser på kraven vilken typ av mönster som ska användas och tillsammans med de tekniska- och affärskraven så beräknar han vilket mönster som passar den specifika integrationen. F4: Tror inte att det finns ett mönster som är mest lämpligt utan det finns alternativ. Beroende på kravspecifikationen anser han att erfarenhet avgör. Ibland kan han gå andra vägen och skapa en lösning och se att det skapar ett mönster. F5: Vet det här utifrån erfarenhet och tar reda på fakta om integrationen genom analys och diskussion med kunden för att säkerställa att mönstret passar den specifika integrationen.
Applicering av mönster Hur går du tillväga för att lösa ett integrationsproblem med, respektive utan ett tydligt mönster?	F1: Arbetar med verktyget IBM:s produkt Websphere Message Broker och använder sig utav verktygets hjälpmedel i första hand. Därefter tar han hjälp av IBM:s hemsida, Google, kollegor eller projektets arkitekt. F2: Anser att när kravspecifikationen har kommit till utvecklaren så är det ett integrationsmönster. Om några oklarheter finns gällande vilka mönster som ska användas så frågar han arkitekten eller kollegor för att säkerställa det tekniska i integrationen. Anser att det bygger ”lite på Best Practice sen tidigare”.

		F3: Tittar ofta på om det finns färdiga, tidigare eller liknande lösningar. Stöter sällan på något problem som är helt nytt. F4: Eftertänksamhet och analys enligt respondenten om det inte finns tydliga mönster. Fråga intressenterna vilket kan vara alltifrån beställaren till användarna. Om det finns mönster anser han att det bara bidrar till en ökad förståelse, snabbare när det gäller vilken typ av integration det är. Annars använder han Google väldigt mycket. F5: Finns inget mönster så testar han sig fram och för en diskussion med kunden. Har han ett tydligt mönster anser han att det är mycket lättare, för då kan han kommunicera kring integrationen och ta hjälp av kollegor.
	Om ett mönster hittas som har applicerats tidigare, tar ni hänsyn till i vilken kontext det har gjorts? Samt kontrollera om lösningen kan uppenbara problem i ett senare skede och passar det ert specifika problem?	F1: Tänker inte på kontexten men anser att utvecklare bör göra det och att det har med erfarenhet att göra. Uttalar sig även om att han tänker till om en lösning kan uppenbara problem i ett senare skede och om det passar hans specifika problem. F2: Respondenten uttrycker att han absolut tänker på detta. Kontrollerar om lösningen är konfigurerbar med andra komponenter och tänker på skalbarhet. Han tänker även på senare skeden med nya utvecklare och förändringar i systemen. F3: Kontexten enligt respondenten är vilken typ av system och information som integrationen innefattar. Respondenten tänker även i nästa steg i form av diskussion med arkiteken om en oklarhet uppstår annars löser han det själv. Anser även att mönstren ska vara tillräckligt komplicerade för att skapa en bra lösning men tillräckligt enkla för att nästkommande kollega ska förstå den.

		F4: Anser att det beror på ansvarsnivån. Som ren utvecklare är kontexten utanför ansvarsområdet men dock tänker han ändå om det kan orsaka problem vid implementation. Tar också hänsyn till att problem i senare skeden kan uppstå när en ny kollega tillkommer. Tycker att då bör man förlita sig mer på bra dokumentation och standards i arbetssättet. F5: Anser att man måste ta hänsyn till kontexten eftersom det bestämmer vilket mönster man ska använda. Han tar även hänsyn till konsekvenser och problem som kan uppstå i ett senare skede. Han anser att skalbarhet är en av de största delarna i designen och skalbarhet och konfigurerbarhet finns alltid i åtanke vid utformning av arkitekturen.
	Åsikter om mönster och kommunikation Har du upplevt kommunikationsproblem med kollegor inom integrationsutveckling på grund av skild terminologi?	F1: Ja, enligt respondenten uppstår kommunikationsproblem på grund av att kollegor använder slangord och förkortningar vilket även kan benämnas facktermer. Han anser att vissa kollegor inte använder mönster i kommunikationen på det projekt han befinner sig i. Han tycker att det effektiviserar kommunikationen väldigt mycket på en arbetsplats om det finns en gemensam terminologi. F2: Ja, det har han upplevt. Kollegor som använder andra verktyg har andra benämningar på samma term. Förklarar även att det är svårt med terminologin på djupare, teknisk nivå. Dock anser han att det hjälper att förklara en komplex värld med ett ord för att konceptet bakom är densamma även om teknologin bakom är helt annorlunda implementerat.

		F3: Har aldrig stött på kommunikationsproblem med kollegor inom integration men anser att det kan uppstå med människor från ”de som ställer kraven eller någon som kommer från en annan del av IT”. Tycker även att kommunikationsproblem inte uppstår på den övergripliga mönsternivån utan på den djupare, tekniska. Dock förespråkar han att det skulle effektivisera om alla var införstådda med definitionen av termer inom integrationsmönster. F4: Ja, när respondenten var ny inom området. Föreslår även att termer inom olika verktyg skiljer sig åt och det skapar förvirring för honom även om alla jobbar med integrationsutveckling. Men beskriver att på lägre teknisk, detaljnivå så går det inte att kommunicera med mönster utan krävs specifika förklaringar. F5: Drar sig från att kommunicera eftersom att han inte har så stor kunskap om just EAI-mönster. Han är väldigt erfaren, rent hur man löser problem men har inte teoretiserat kring det särskilt mycket, och kan därför inte alla termer. Detta leder till kommunikationsproblem i form av att han drar sig från att skicka ett mejl till någon som han vet är duktig eftersom att han inte kan förklara problemet på grund av bristande teoretisk kunskap.

Vilka för- och nackdelar finns det sett ur en integrationsutvecklares perspektiv?	Vilka positiva och negativa aspekter ser du vid användningen av mönster?	F1: Det positiva enligt respondenten är en effektivare utvecklingsprocess på grund av gemensam terminologi, en bas av standardslösningar och det blir en rutinmässig arbetsprocess. Nämner hur de har utvecklat ett skelett till en integration. Genom en knappklickning så dyker grundkopplingarna för ett mönster upp och det underlättar arbetet. Det negativa är överanvändandet av mönster enligt respondenten. Det blir inte effektivt och en speciallösning i sådana fall anser han är bättre. F2: Det positiva är enligt respondenten tidsbesparingen, enhetlig förståelse för termer och en standard för arbetssättet. Medan det negativa kan vara omedvetenheten om vilket mönster som är den mest optimala. För han anser att tidigare integrationsutvecklare har gjort en korrekt lösning men med fel mönster. Det kan påverka integrationens framtida skalbarhet enligt respondenten. F3: Om det används på rätt sätt sparar det tid, mindre komplext, förbättrar kommunikationen och återanvändbarhet i kod samt enklare att förklara problemen eftersom det är indelade i mindre beståndsdelar. Negativa aspekter är skapandet av egna mönster för sakens skull. Det genererar merarbete för ingenting enligt respondenten. F4: Positivt är gemensam terminologi, återanvändbarhet och snabbare förståelse för integrationen. Det negativa anser respondenten är användandet av mönster bara för att de finns, som exempelvis mönster för tidsuppskattning om hur lång tid delar av projektet bör ta. Nedbrytning av projektet i så pass små delar kan skapa förvirring och det påverkar projektet negativt enligt honom.

		F5: Anser att EAI-mönster främjar kommunikation, mönster på en mer teknisk nivå främjar hastigheten i utvecklingen och förståelsen. Han anser även att det underlättar förvaltningsarbetet och underhållsprocessen. Det blir även mer strömlinjeformat istället för spretigt. Respondenten tycker att en negativ aspekt kan vara att man använder ett mönster bara för sakens skull, vilket kan göra integrationen för komplex och vara tidskrävande.
	Är mönstren du använder dina egenutvecklade, verksamhetens gemensamt uppsatta mönster eller är det kunddirektiv?	F1: Respondenten använder en kombination av mönster som är egenutvecklade och utvecklade på projektnivå. Anser även att det främjar återanvändbarhet och effektivitet på ”alla sätt och vis”. Pekar på att en gemensam terminologi underlättar kommunikationen mellan kollegor och att kunna peka ut tidigare lösningar som skulle kunna appliceras vilket effektiviserar och främjar återanvändbarhet. F2: Anser att de mönstren som används är Best Practices. De har enligt respondenten satt tre mönster som är övergripliga för verksamheten. Det främjar enligt honom, en återanvändbarhet i mönstren men påpekar att det inte är ett SOA-tänk. I SOA återanvänder man tjänster men inom organisationen återanvänds endast skeletten för mönstren. F3: Om det används på rätt sätt sparar det tid, mindre komplext, förbättrar kommunikationen och återanvändbarhet i kod samt enklare att förklara problemen eftersom det är indelade i mindre beståndsdelar. Negativa aspekter är skapandet av egna mönster för sakens skull. Det genererar merarbete för ingenting enligt respondenten.

		F4: Anser att mönstren inte är utvecklade utan snare upptäckta. Respondenten har endast träffat på de i projektform där det har funnit riktlinjer för användning av dem. Han anser att riktlinjer och mönster effektiviserar utvecklingsprocessen och främjar återanvändbarhet, särskilt när det kommer in en ny utvecklare eftersom det finns en standard för man löser problem. F5: Egenutvecklade i den mån att han begränsar de till sig själv och sin närmaste krets. Han anser att detta främjar återanvändbarhet och effektiviserar utvecklingsprocessen.
	Resultat av mönsteranvändning Resulterar användingen av mönster i ett mer kvalitativt resultat eller komplicerar det utvecklingsprocessen?	F1: Rent kvalitativt sparar man tid och pengar till kunden. Kompliceringen pekar han återigen på överanvändandet och tiden det tar i sådana fall. F2: Anser att det ger mer kvalité och komplicerar inte utvecklingsprocessen. F3: Första gången ett mönster skapas bidrar det till kvaliteten men komplicerar utvecklingsprocessen. Övriga gånger minskar komplexiteten och kvaliteten förblir densamma enligt respondenten. F4: Anser att det bidrar till kvaliteten för att mönster ger riktlinjer för integrationen. Anser inte att det komplicerar utvecklingsprocessen förutom att det kanske används i onödan.

		F5: Tror att det bidrar till ett mer kvalitativt resultat och att förståelsen blir bättre, framförallt underhållbarhet och har man ett mönster så slipper man uppfinna hjulet igen, vilket reducerar antalet fallgropar som kan påträffas. Respondenten anser även att kommunikationen med kunden förbättras genom att han kan skapa en trygghet eftersom att han kan förmedla kommunicera integrationen på ett bättre sätt. Han tycker det endast komplicerar om man använder det för sakens skull.

Referenser

Andersson, F., Chidiac, T., Jacobsen, R., Olsson, K., Saka, O,. 2014. *The Art of Data*. Lunds universitet

Alexander, Cristopher. Ishikawa, Sara. Silverstein, Murray. 1977. *A Pattern Language - Towns, Buildings, Construction.* New York: Oxford U.P. ISBN *0195019199, 1171s.*

Arsanjani, A., Booch, G., Boubez, T., Brown, P. C., Chappell, D., deVadoss, J., Erl, T., Josuttis, N., Krafzig, D., Little, M., Loesgen, B., Thomas Manes, A., McKendrick, J., Ross-Talbot, S., Tilkov, S., Utschig-Utschig, C., and Wilhelmsen, H. (2009). *SOA Manifesto*.. http://www.soa-manifesto.org/ (Hämtad 2014-04-14).

Baroni, A., Gu\'eh\'eneuc, Y. and Albin-Amiot, H., 2003. *Design patterns formalization*. Rapport de recherche, D\'epartement d'informatique, \'Ecole des Mines de Nantes, (03/03).

Beck, Kent. Cunningham, Ward. *Using pattern languages for Object-Oriented programs*. Technical Report CR-87-43, Apple Computer, Inc. and Tektronix, Inc., 1987.

Boh, Wai Fong; Yellin, Daniel. 2007. *Using Enterprise Architecture Standards in Managing Information Technology.* In: Journal of Management Information Systems. Vol. 23, 3 uppl. s163-207, 45s

Choi, Jae; Nazareth, Derek L.; Jain, Hemant K. 2010. *Implementing Service-oriented Architecture in Organizatios.* In: Journal of Management Information Systems. Vol. 26 uppl 4, s253-286. 34s

Dong, J., Peng, T. and Zhao, Y., 2011. *On instantiation and integration commutability of design pattern*. The Computer Journal, 54(1), pp.164--184.

Erl, T. 2005. *Service-oriented architecture*. Upper Saddle River, NJ: Prentice Hall Professional Technical Reference. ISBN *0-13-185858-0, 760s*.

Finkelstein, C. (2006). *Enterprise architecture for integration*. 1st ed. Boston: Artech House. E-book. ISBN *9781580537131, 500s*.

Friesen, Jeff. 2012. *Design patterns, the big picture, Part1: Design pattern history and classification*. Javaworld.
http://www.javaworld.com/article/2078665/core-java/design-patterns--the-big-picture--part-1--design-pattern-history-and-classification.html (Hämtad 2014-04-08)

Gamma, H., Helm, R., Johnson, R., Vlissides, J. (1995):
Design patterns : elements of reusable object-oriented software
ISBN*: 0201633612, 395s*

Hohpe, G., 2003. *Enterprise Integration Patterns - Solving Integration Problems using Patterns*. [online] Eaipatterns.com.
http://www.eaipatterns.com/Chapter1.html
(Hämtad 2014-04-09).

Hohpe, G., 2007. *SOA Patterns--New Insights or Recycled Knowledge. Enterpise Integration Patterns*.[Retrieved March 4, 2012] http://www. eaipatterns. com/docs/SoaPatterns. pdf.

Hohpe, G., Wirfs-Brock, R., Yoder, J. W. and Zimmermann, O. 2013. *Twenty Years of Patterns' Impact*. In: IEEE Software, Vol 30 Uppl 6, s84-87, 4s

Hohpe, Gregor. Woolf, Bobby. 2004. *Enterprise Integration Patterns; Designing, Building, and deploying messaging solutions.* 1 uppl. Addison-Wesley Professional. ISBN *978-0-321-20068-6, 736s.*

Holley, K. and Arsanjani, A. 2010. *100 SOA Questions Asked and Answered*. E-book Pearson Education Inc. www.zenisoft.cn/calibre/get/pdf/772. ISBN *978-0-137-08020-5, 242s.* (Hämtad 2014-04-14)

IBM. 2005. [online] Publib.boulder.ibm.com.
http://publib.boulder.ibm.com/infocenter/rsdvhelp/v6r0m1/index.jsp?topic=%2Fcom.ibm.xtools.pttrn.author.doc%2Ftopics%2Fc_benefits_ptrns.html
(Hämtad 2014-04-09).

Jacobsen, D. I. (2002): *Vad, hur och varför? Om metodval i företagsekonomi och andra samhällsvetenskapliga ämnen*. Lund: Studentlitteratur.

Karhikeyan, T. and Geetha, J. 2012. *A metrics suite and fuzzy model for measuring coupling in Service Oriented Architecture*. In: Proceedings of the 2012 International Conference on Recent Advances in Computing and Software Systems (RACSS), pp. 254--259. ISBN: 9781-467302524

Kvale, S. (1997): Den kvalitativa forskningsintervjun. Lund: Studentlitteratur

Lankhorst, Marc. 2012. *Enterprise Architecture at Work.* 3 uppl. Springer Heidelberg New York Dordrecht London. E-book. ISBN *978-3-642-29651-2, 356s.*

Mangalaraj, George; Nerur, Sridhar; Mahapatra, RadhaKanta; Price, Kenneth H. 2014. *Distributed Cognition in Software Design an Experimental Investigation of The Role of Design Patterns and Collaboration.* In: MIS Quarterly. Vol. 38, Uppl 1, s249-A5, 30s

Nofal, Muhmmad I. Yusof, Zawiyah M. 2013. *Integration of Business Intelligence and Enterprise Resource Planning within Organisations.* In: Procedia Technology , Vol. 11, s658-665, 8s

Papazoglou, M. P. and Van Den Heuvel, W. 2007. *Service oriented architectures: approaches, technologies and research issues*. The VLDB journal, 16 (3), pp. 389--415.

Prechelt, Lutz; Unger, Barbara; Tichy, Walter F.; Brössler, Peter; Votta, Lawrence G. 2001. *A Controlled Experiment in Maintenance Comparing Design Patterns to Simpler Solutions* . In: IEEE Transactions on Software Engineering. Vol. 27 Uppl 12, s1134-1144, 12s

Scheibler, T. and Leymann, F. 2008. *A framework for executable enterprise application integration patterns*. In: Springer, s. 485-497.