LUND UNIVERSITY
School of Economics and Management

Krav och arkitektur

Mjukvaruarkitektoniska designbeslut i kravhantering

Nyckelord: Krav, kravhantering, mjukvarukvalitet, arkitektur, mjukvaruarkitektur, kvalitetsattribut

Kandidatuppsats, 15 högskolepoäng, SYSK02 i informatik

Framlagd: 2014-06-03

Författare: Mohamed Ar-rawi 900130
Filip Nilsson 841210

Handledare: Nicklas Holmberg
Examinatorer: Magnus Wärja
Mirella Muhic

Abstrakt

Titel Krav och arkitektur – Mjukvaruarkitektoniska designbeslut i kravhantering

Författare Mohamed Ar-rawi
Filip Nilsson

Utgivare Institutionen för informatik

Handledare Nicklas Holmberg

Examinatorer Magnus Wärja
Mirella Muhic

Publicerad 2014

Uppsatstyp Kandidatuppsats

Språk Svenska

Nyckelord Krav, kravhantering, mjukvarukvalitet, arkitektur, mjukvaruarkitektur, kvalitetsattribut

Abstrakt Uppsatsen berör relationen mellan kravhantering och mjukvaruarkitektur. Att hantera krav inom mjukvara innebär att lösa problem. Kravhantering är komplext då krav kan komma från olika människor från olika nivåer i en organisation samt från den miljö där mjukvaran kommer att verka. Syftet med denna studie är att identifiera vilka konsekvenser kravhantering medför när mjukvaruarkitektoniska designbeslut fattas. Vilket drivs med frågeställningen: *Hur påverkar kravhantering mjukvaruarkitektoniska designbeslut?*. Med hjälp av intervjuer med yrkesverksamma arkitekter har vi undersökt hur de förhåller sig till kravhantering när designbeslut fattas om arkitektur. Slutsatserna för uppsatsen är att kravhantering inte avslutas med att delge utvecklingsteam de krav som ställs på system utan processen fortsätter fram tills dess krav är formaliserade i en mjukvaruarkitektur.

Innehållsförteckning

Figurer

1 Introduktion

När verksamheter ska köpa in, anpassa eller utveckla informationssystem måste behov på något sätt uttryckas och ofta skapas detta i form av krav. Krav kan också kommuniceras på andra sätt så som under möte, intervjuer med intressenter eller under utvecklingsarbetet. Hantering av krav har under många år varit en gren under systemutveckling som går under namnet *Requirements Engineering* (RE) (Hull, Jackson, & Dick, 2011). RE är dock ett generellt uttryck som omfattar kravhantering utan koppling till projektmetod. Det finns således en bred spridning över hur krav ska hanteras. Som ett svar på krav designas system och mjukvaruarkitektur fastställs.

Alla system har en struktur även om dessa inte är uttryckta i modeller eller kommuniceras på något annat sätt (Bass, Clements, & Kazman, 2013). Denna struktur blir arkitektur när den uttrycks på något sätt, det finns dock inga riktlinjer för när en struktur är tillräckligt uttryckt för att kallas arkitektur. Arkitektur är inget mått för hur bra ett system är eller hur väl det fungerar. Den avslöjar endast den underliggande strukturen för ett system samt kan användas som ett medel att uttrycka vilka designbeslut som gjorts. Arkitekturen används även som bedömningsunderlag för intressenter att bedöma hur väl kraven uppfylls av systemet. Precis som mjukvara eller informationssystem har en underliggande struktur som går att uttrycka i arkitektur har också verksamheter en struktur, detta benämns ofta som verksamhetsarkitektur. Denna arkitektur grundar sig i vilka designbeslut som verksamhetsägare har tagit när de har skapat sin verksamhet. I bästa fall ska krav beskriva de behov som verksamhetsarkitekturen ställer.

1.1 Problemdefinition

Krav är i grund och botten kärnan för varje projekt. Detta gör att de krav som ställs från intressenterna skall vara realistiska och behoven från intressenternas sida skall vara väldefinierade för att designbeslut skall kunna tas och för att rätt arkitektur skall kunna skapas. En av de största anledningarna Hull et al. (2011) presenterar som misslyckande i införande av informationssystem är ofullständiga krav. Detta utgör 13.1 % av de anledningar till varför införandet av system misslyckas. Det finns dock flera andra faktorer som påverkar mjukvaruarkitektur. Det kan vara arkitektens erfarenheter av lyckade metoder i tidigare projekt då arkitektur skapas, det kan även vara verksamhetens mål och rutiner samt de resurser som verksamheten har att tillgå (Bass et al., 2013). Arkitekters erfarenhet kan också leda till antagande om outtalade krav eller deras tänkta övergripande arkitektur, alltså samspelet mellan verksamhetsarkitektur och mjukvaruarkitektur. Vilket kan leda till att formaliserade krav täcks in men också mycket mer levereras vilket kan leda till komplicerade system som är svåra att hantera eller blir användarlösa. Ett tänkbart scenario är att systemutvecklingsföretag driver på mjukvaruarkitektur som ska stödja en tänkt verksamhetsarkitektur och verksamhetsägare driver åt ett annat håll. Denna hantering ställer vidare krav på att verksamheten anpassar sig till systemet.

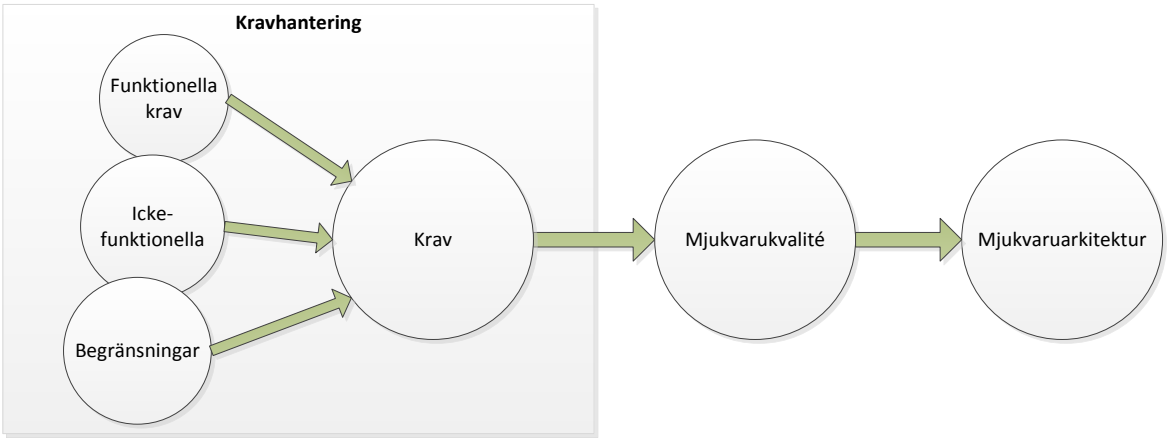
För att exemplifiera problemet har vi valt att här presentera ett projektmisslyckande som ägde rum i USA. Detta hälsovårdsprojekt lanserades i oktober 2013 men misslyckades att uppfylla sitt syfte. Projektet drevs av den Amerikanska regeringen och innebar att utveckla en webbplats som skulle fungera som en central plats att utbyta sjukförsäkringar för de stater som valde att inte skapa sina egna statliga utbyten. Syftet var att hjälpa människor med ekonomiska svårigheter att hitta sjukförsäkringar. Kostnaden för hela projektet uppgick till ungefär 500 miljoner amerikanska dollar. (Wikipedia, 2014)

Det grundläggande hälsovårdssystemet konstruerades med syftet att det skulle vidarebefordra förfrågningar till externa leverantörer. Denna funktion skulle finnas för de människor som sökte sjukförsäkringar. Sammanlänkningen och händelseförloppet skulle i sin tur ske i realtid. Detta ledde till en enorm belastning på de statliga servrarna och gav sedan upphov till att systemet kollapsade (Cleland-Huang, 2014). Ett annat problem med systemet var registreringsprocessen av användare. Cirka 9.5 miljoner användare försökte registrerar sig på webbplatsen och endast 271 000 lyckades. Systemet var således inte konstruerat att kunna hantera så många initiala användare. (Cleland-Huang, 2014)

Arkitekter som var inblandade i projektet lade inte tillräckligt med fokus på att uppfylla renodlade kvalitéer som mjukvaran skulle uppnå, exempelvis prestanda och tillgänglighet. Istället lades fokus på den funktionella nivån och kraven som ställdes var i princip sjukvårdsbestämmelser som konverterades till funktionella krav. Detta är i praktiken ett vanligt misstag i skapande och val av mjukvaruarkitektur. Ofta förlitar sig intressenter på den implicita kunskapen när det kommer till kvalitet och avgörandet om vilken sorts kvalitet som skall uppnås tas inte i beaktning. Det kan vara till exempel frågor som hur snabbt, hur säkert och hur tillförlitligt ett system ska vara, som inte beaktas. Därför hävdar Cleland-Huang (2014) att ignorera krav i form av kvalité kan tyvärr resultera i misslyckande som detta hälsovårdsprojekt. Skulden lades på arkitekten även fast kraven inte var tillräckliga. Således kan inte goda designbeslut fattas om inte krav innefattar kvalitet.

Att samla krav i kravspecifikationer är förknippat med viljan att formalisera och i efterhand kunna rättfärdiga ett informationssystem. Kravspecifikationer är ett vanligt sätt att hantera krav och som uttrycks i vårt exempel från Cleland-Huang (2014) är kraven ofta definierade enligt vilka funktioner som systemet ska utföra under körning. Kvalitetsattribut såsom interoperabilitet, prestanda, tillgänglighet, skalbarhet med flera påverkar designbeslut och bristen av dem i kravspecifikationer innebär ett större ansvar för arkitekter och utvecklare att fylla i gapen.

Problemet är att system i sin enskildhet kan vara väldigt bra och uppnå en rad kvalitetsattribut genom en viss mjukvaruarkitektur, men verksamhetssammanhanget bestämmer om det är rätt kvalitéer. Om krav inte uttrycker vilken kvalitet som systemet skall uppnå blir det således svårt att skapa, välja en mjukvaruarkitektur eller utvärdera om en mjukvaruarkitektur stödjer en specifik verksamhet.



Figur 1.1 Logisk kedja, från krav till mjukvaruarkitektur. Egen illustrerad bild om uppsatsens beröringsområde

För att illustrera logiken har vi skapat modell över problemområdet (Figur 1.1). Krav på mjukvara har tre olika källor (Bass et al., 2013). De hanteras och ligger till grund för att skapa en bild av vilken kvalitet en viss mjukvara ska uppnå. Denna kvalitet kan senare mötas med en teknisk struktur i form av mjukvaruarkitektur. Vidare kan omvänt mjukvaruarkitekturen bestämma vilken kvalitet som mjukvara har uppnått. Det blir således den som bestämmer vilka krav som strukturen uppnår. Kravhanteringsprocessen ansvarar för att de krav som ställs är relevanta och länken till mjukvaruarkitektur går genom de kvalitéer som arkitekturen ska uppnå.

1.2 Forskningsfråga

Hur påverkar kravhantering mjukvaruarkitektoniska designbeslut?

1.3 Syfte

Syftet med studien är att identifiera konsekvenser som kravhantering medför vid design av mjukvaruarkitektur.

1.4 Avgränsning

Denna studie omfattar krav som är relaterade till mjukvara. Utgångspunkten som vi valt är att mjukvaruarkitekter arbetar med att strukturera mjukvarukomponenter genom arkitektur således är mjukvaruarkitektur ett av fokusområdena. Designbeslut i denna kontext handlar därför om att strukturera mjukvarukomponenter eller anpassa krav till befintliga strukturer. Det finns andra arkitekturer inom systemutveckling men dessa behandlas endast för att visa mjukvaruarkitekturens position. Krav på andra delar inom systemutveckling utelämnas. Krav och kravhantering behandlas också eftersom det primärt är denna process som ansvarar för att

renodla krav. Uppsatsen behandlar endast kravhantering inom systemutveckling och vidare är *Requirements Engineering* (RE) den disciplin som hanterar detta teoretiskt. RE kommer därför att behandlas synonymt med kravhantering inom systemutveckling.

1.5 Definitioner

1.5.1 Traditionell utveckling

Innebär att utveckling sker fasvis i så kallade trappor. Här hanteras krav och dokumenteras i början av utvecklingen. Kraven ändras således inte efterhand under utvecklingen. (Avison & Fitzgerald, 2006)

1.5.2 Skallkrav

Ordet skallkrav används oftast inom offentliga upphandlingar och skrivs i ett förfrågningsunderlag för en upphandling. Det kan emellertid ses som krav som måste uppfyllas av anbudsgivaren; den som lägger anbud på en upphandling. (Institutet, 2012)

1.5.3 *Oracle och MSSQL*

Databaser som samlar data och bildar detta i en så kallad enhet. Syftet är att lagra och hämta relaterad information (Neeraj Sharma, 2010).

1.5.4 Issue tracking

Ett program som används för att spåra exempelvis, buggar, problem, uppgifter, förfrågningar etc. i ett system (Dane Bertram, 2010).

2 Litteraturgenomgång

I detta kapitel går vi igenom teori för vårt ämne. Vi behandlar problemområdet genom att inledningsvis lyfta teori om krav och kravhantering genom RE. Kravhantering är en uppsättning processer inom systemutveckling och vi behandlar därför teori som är förknippat med agil-utveckling, syftet är att identifiera skillnader i dessa processer i förändringsbenägna verksamheter. Därefter behandlas relationen mellan krav och arkitektur för att binda samman våra fokusområden. Arkitektur inom systemutveckling är inte ett helt renodlat begrepp. Således tas detta upp som utgångspunkt för att vidare beskriva vilka roller som arbetar med designbeslut för arkitektur samt var mjukvaruarkitektur positionerar sig i detta. Arbetet med arkitektur förtydligas därefter med teori om metoder.

2.1 Krav och kravhantering

Krav är behov som uttrycks från intressenter och som ligger till grund för att fatta rätt designbeslut och i sin tur skapa en lämplig mjukvaruarkitektur. Krav från intressenter skiljer sig mycket både i text och struktur. Rago, Marcos, och Diaz-Pace (2011) tar upp hur stor skillnad uttryckta krav har, närmast 80 % av alla kravspecifikationer för mjukvara är skrivna i ett naturligt språk. Vidare är 16 % av kravspecifikationer organiserade och strukturerade i mallar dock utan att anges på ett formellt sätt. Summerar vi dessa siffor är totalt 96 % av krav dokumenterade i ett naturligt språk. Detta kan ses som en fördel då kommunikationen bland intressenter förbättras. Samtidigt kan kravspecifikationer som är skrivna på ett naturligt språk vara negativt. En anledning är enligt Rago et al. (2011) att processen för att analysera och gå igenom specifikationerna kan vara tidskrävande, speciellt i stora implementationer. Det blir även väldigt svårt att omsätta naturligt skrivna krav till tekniska krav och således extrahera de olika typer av krav som presenteras nedan.

2.1.1 Typer av krav

Det finns tre olika typer av krav enligt Bass et al. (2013) dessa är

- Funktionella krav
- Kvalitetskrav
- Begräsningar till exempel infrastruktur med mera

Funktionella krav beskriver hur ett system måste agera utifrån stimuli vid körning. Icke-funktionella krav hanterar hur väl ett system kvalificerar sig mot kvalitetsattribut. Detta kan handla om prestanda, tillgänglighet, skalbarhet med mera. Begränsningar är krav som inte går att förändra till exempel infrastruktur, befintliga system, programspråk eller tjänsteorientering (Bass et al., 2013).

Vogel, Arnold, Chughtai, och Kehrer (2011) visar endast krav som funktionella och icke-funktionella. Begränsningar hanteras som indirekta icke-funktionella krav och kvalitetsattribut som direkt icke-funktionella krav.

2.1.2 Kravhantering

För att beskriva vad en verksamhet behöver uttrycks detta i krav. Krav är inte bara ett löst önskemål, utan kopplade till måsten, till exempel: *för att få leverera måste ni komma med lastbil*. När krav är uttalade och överenskomna är det kontrakt för hur en specifik situation ska hanteras (Hull et al., 2011). De är också statiska fram till dess att de upplöses eller blir inaktuella menar Hull et al. (2011).

Krav är från intressenter ofta uttryckta i naturligt språk (ett språk som skiljer sig från konstruerade och formella språk), vilket skapar problem när detta språk ska omsättas till mjukvara eftersom naturligt språk sällan beskriver en teknisk implementation (Hull et al., 2011). Aurum och Wohlin (2005) beskriver att naturligt språk kan vara svårtolkat och leda till att utvecklare designar och implementerar lösningar som inte når upp till krav. Trots detta behandlas krav i detta språk och lägger också grunden till hur projekt utformas. Krav kan dessutom komma från många olika intressenter och vara i konflikt med varandra (Hull et al., 2011). Som ett svar på denna problematik skapades disciplinen *Requirements Engineering* (RE). Denna disciplin ämnar till att hantera krav och definieras som:

> ”*Requirements engineering*: *the subset of systems engineering concerned with discovering, developing, tracing, analyzing, qualifying, communicating and managing requirements that define the system at successive levels of abstraction.”*(Hull et al., 2011).

Således är RE en del av systemutveckling med klart definierade aktiviteter och ska definiera efterföljande system. Aurum och Wohlin (2005) förklarar att den mesta forskning om RE handlar om insamling, dokumentering, och styrning av krav inom projekt. Författarna belyser också att RE innefattar processer som går att applicera i produktionsmiljö. Vidare kan alltså RE ses som kravhantering inom specifikt systemutveckling men är inte begränsad till kravhantering inom projektföring.

Enligt Aurum och Wohlin (2005) finns ett antal risker som karakteriserar RE. Det finns en tendens att förbise krav som anses vara kritiska, hänsyn tas oftast till funktionella krav men även försök till att göra kraven perfekta och förstådda innan utvecklingsarbetet har påbörjat. Problemet är att kraven inte är helt kända förrän det används i praktiken. Dessa risker kan adresseras i kravhanteringsarbetet då organisering, kategorisering samt prioritering av krav sker. För att lyckas extrahera de riktiga och väsentliga kraven kan exempelvis *user stories* användas (se kapitel 2.3.7), workshops med mera.

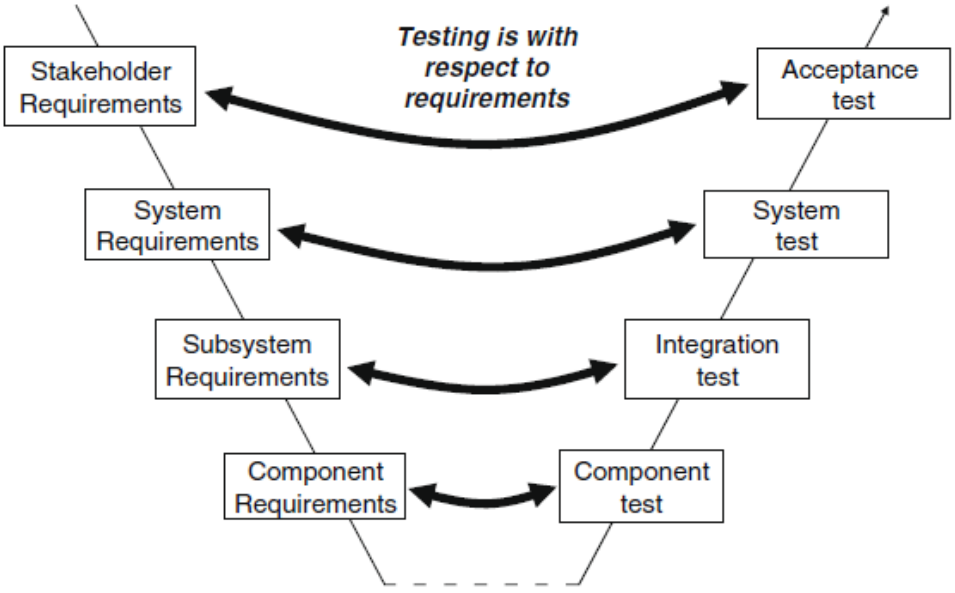
Enlig Bass et al. (2013) är ofullständiga krav en av de största anledningarna till misslyckade projekt. Dock finner Carrera, Iglesias, och Garijo (2013) att om en systemimplementation skall lyckas krävs det en ökad förståelse för de krav som intressenter ställer, det krävs också att systemet uppfyller önskad funktionalitet. Dessa aspekter är de viktigaste enligt Carrera et al., (2013) och ses som en central del av kravhantering enligt Bass et al. (2013). Kommunikation

blir då en viktig aspekt att hantera inom kravhantering på grund av att ökad kommunikation mellan intressenter och utvecklingsteam ses som den största utmaningen, men även att konkret förstå vilka problem som finns (Carrera et al., 2013).

Kommunikationsgapet mellan intressenter och utvecklingsteam orsakas främst av att krav missförstås och speciellt de krav som är ytterst nödvändiga. Då krav ställs är de ofta överspecificerade och anledningen till detta är för att de är uttryckta som lösningar till de problem som finns. Problemet blir att mycket fokus läggs på vad som skall göras och inte varför det skall göras. På detta sätt skapas inte så stort utrymme för utvecklingsteam att överväga om kraven är lösningen till att uppfylla intressenternas behov. (Carrera et al., 2013)

2.1.3 Livscykeln

Inom teorin råder det delade meningar om en enhetlig RE modell som beskriver kravens livscykel. Hull et al. (2011) beskriver vilka olika moment som ingår i systemutvecklingsprojekt och menar att den sista aktiviteten består av acceptanstest. Denna fas består i att säkerställa att intressentkrav fortfarande består i sista skedet. Detta exemplifierar Hull et al. (2011) genom den klassiska V-modellen. Testaktiviteter delas då in i olika nivåer beskrivet i figur 2.1 och inleds med komponenttest. När denna testfas är över och säkerställd tar nästa fas över och integrationstest utförs. Slutligen utförs systemtest och livscykeln avslutas med acceptanstest.



Figur 2.1 V-modellen (Hull et al., 2011, s 10)

Denna modell liknar *Systems Development Life Cycle* (SDLC) även kallad vattenfallsmetoden (Avison & Fitzgerald, 2006). Men i de fall vattenfallsmetoden används saknas trappan tillbaka för att säkerställa krav. Det som likställer dessa metoder är att krav hanteras som statiska från projektinledning.

Aurum och Wohlin (2005) finner att processen inte kan beskrivas som en fastställd process och menar att RE ser olika ut i olika projekt. Vidare argumenterar författarna att RE inleds linjärt för att ske iterativt under en prototypfas. De menar också att även om kravhantering är en fundamental process för att lyckas med projekt, kan inte förbättring av denna process medföra förhöjda chanser att lyckas med projekt.

2.1.4 Spårbarhet och testning

Spårbarhetens syfte är att införa mer förtroende för intressenter att möta mål, förmågan att utvärdera inverkan av förändring, bättre motivering av subsystem, förmågan att spåra framsteg samt förmågan att balansera kostnad mot nyttan (Hull et al., 2011). Detta stärks också av Aurum och Wohlin (2005) som belyser att spårbarhet leder till mer kvalitativa produkter.

Enligt Hull et al. (2011) bygger RE på spårbarhet och testbarhet. Test ska relatera till de krav de ämnar att testa (Hull et al., 2011). Intressenters krav ska vara spårbara från systemkrav likaså subsystem ska vara spårbara av komponenter och så vidare. En grundtanke för denna kedja är att ny information ska länkas tillbaka till äldre information. På så sätt kan förändring spåras uppifrån och ner genom att besvara frågan ”om detta förändras vad kommer då att hända?”. Samtidigt kan också krav motiveras genom att ställa frågan ”varför är detta här?” och på så sätt spåras till sitt grundläggande svar. Spårbarhet inför också frågan om omfattning, om alla krav går att spåras uppifrån och ner och vice versa (Hull et al., 2011).

Som beskrivet tidigare ska tester utföras i olika lager. Hull et al. (2011) anser att testning handlar om kvalificering i sitt perspektiv. De hävdar också att denna uppgift ska påbörjas tidigt eftersom de annars kan leda till dyr och tidskrävande omdesign. Hull et al. (2011) använder samma V-modell som illustreras i figur 2.1 för att visa en tänkbar strategi för att utföra tester i olika lager.

2.1.5 Modellering

Modellering och kravhantering går hand i hand. Krav skapas som sedan omsätts till modeller för att representera dessa krav. Detta är en stor del av systemutvecklingsprocessen, modellen står som grund för att skapa krav i nästa lager. En systemdesign består av flera olika relaterade modeller. I sin enskildhet beskriver aldrig en modell ett helt system, i så fall skulle de inte vara en modell. De delar av designen som inte täcks in av modeller lämnas till krav i textform att uttrycka (Hull et al., 2011).

En modell är en abstraktion av ett system som beskriver specifika delar och ignorerar andra. Modeller ger fokus till att arbeta med en liten del åt gången när helhetsperspektivet blir för komplext.

I RE används modeller enligt Hull et al. (2011) för att:

- Kommunicera med kunder samt skapa större förståelse för systemet som ska utvecklas.
- Analysera systemet för att säkerställa önskade egenskaper.

- Bestämma hur systemet ska tillfredsställa krav genom att härleda krav till nedre lager.

2.1.6 Kravhantering i en dynamisk miljö

Enligt Ramesh, Lan och Baskerville (2010) är krav och kravhantering (RE) svåra att hantera när det kommer till förändringar i verksamheter. Utvecklare får ständigt hantera krav som förändras och utvecklas. Risken med detta är att krav hinner bli inaktuella innan projekt når sitt slut. De förändringar som Ramesh, Lan och Baskerville (2010) tar upp är plötsliga förändringar av konkurrenshot från utomstående verksamheter, intressenternas egenskaper, teknologi inom mjukvara och ständigt tryck från marknaden. Förskrivna kravspecifikationer blir då mycket svåra att hantera på grund av dessa förändringar (Ramesh, Lan, & Baskerville, 2010).

Agila-utvecklingsmetoder spelar således en viktig roll i att hantera dessa förändringar. Dessa metoder innehåller olika delar såsom små och korta iterationer, ständig leverans till produktion, enkel design, utvärdering av leveranser samt att låta kunder delta i arbetet (Ramesh et al., 2010).

Kravhantering i agil-utveckling skiljer sig från kravhantering i traditionell utveckling eftersom traditionell-utveckling främst använder dokument som källa till krav och kunskapsdelning. Agil-utveckling förlitar sig på verbal-kommunikation för att skapa förutsättningar för att bedriva arbetet. Skillnaden mellan kravhantering i agil-utveckling och traditionell-utveckling menar De Lucia och Qusef (2010) handlar om när krav skall hanteras, snarare än om krav skall hanteras. (De Lucia & Qusef, 2010)

Att arbeta agilt innebär att ständigt leverera till kunden genom snabb och konstant förändring. Detta tillvägagångssätt har blivit mycket uppmärksammat och uppskattat bland verksamheter och utvecklingsteam. (De Lucia & Qusef, 2010)

I agil-utveckling medföljer en rad olika principer som De Lucia och Qusef (2010) belyser:

- Ständigt leverera färdig mjukvara till kunden.
- Mjukvaruarbetet ses som en viktig faktor och princip i att mäta framgång.
- Hela tiden göra kunder nöjda med snabb och ständig leverans av mjukvara.
- Hela tiden skapa förutsättningar för att göra förändringar sent i utvecklingsarbetet.
- Ha ett nära samarbete mellan utvecklare och verksamhetsägare.
- Ständigt kommunicera ansikte mot ansikte.
- Bedriva utvecklingsarbetet runt individer med hög motivation och pålitlighet.
- Enkelhet.
- Självorganiserade team.
- Ständigt anpassa arbetet utifrån förändringar som sker.

Utifrån dessa principer förklarar De Lucia och Qusef (2010) att agil-utveckling bär med sig en stor fördel i att möjliggöra processer för utvecklingsteam att hantera förändringar i kravspecifikationer och detta även sent i utvecklingen.

Enligt Ramesh et al. (2010) är aktiviteter i kravhantering identifiering, analysering, dokumentering samt validering av krav i syfte att utveckla rätt mjukvara. Agila-metoder handlar mer om att bedriva hanteringsarbetet av krav i små och informella steg. Att låta kunden vara delaktig i arbetet ger upphov till att krav blir mer lätta att förstå då de genast skapar en klarare bild om vad som efterfrågas. En nackdel här är enligt Ramesh et al. (2010) att det skapar ett beroende av kommunikation med kunder i att ta fram krav och validera dessa. Detta på grund av att i agil-utveckling läggs tillförlitligheten oftast på verbal kommunikation med kunderna (Ramesh et al., 2010).

User stories är ett annat välanvänt sätt i att hantera krav, speciellt i agil-utveckling. Cohn (2004) hävdar att det bästa sättet att skapa mjukvara som kan möta användarnas behov är att börja utvecklingsarbetet med att skriva ner *user stories*. *User stories* kan förklaras som användarberättelser som innehåller funktionella eller icke-funktionella beskrivningar som är skrivna för hand på exempelvis papper. Dessa funktionella beskrivningar är behov som användaren eller kunden behöver för ett system. Det finns inga direkt riktlinjer om hur detaljerad en *user story* ska vara. Den kan vara så detaljerad att den omfattar mycket funktionalitet eller bara en funktion. Både utvecklingsteam och kunder har ett nära samarbete under hela utvecklingsarbetet då *user stories* framställs (Cohn, 2004).

User stories kan fungera som kravspecifikationer men de är ofullständiga tills både utvecklingsteam och kunder/användare skapar en diskussion om berättelserna genom kollaboration. Enligt Cohn (2004) ger ett agilt-arbetssätt upphov till detta samarbete vilket möjliggör att *user stories* fungerar som krav som går att hantera. Cohn (2004) förklarar att *user stories* är ett sätt att presentera användarnas krav och inte ett sätt att dokumentera kraven. Detta tankesätt är enligt Cohn (2004) ett perfekt sätt att tänka vid skapandet av *user stories*.

Arbetet i stort innebär att efter användarberättelserna är nedskrivna börjar en process med att prioritera dem utifrån vilket värde de har för verksamheten. Vidare placeras berättelserna i iterationer och sedan sker ett acceptanstest som syftar till att validera om en berättelse innehåller de funktioner som den var tänkt att ha då den skapades (Cohn, 2004).

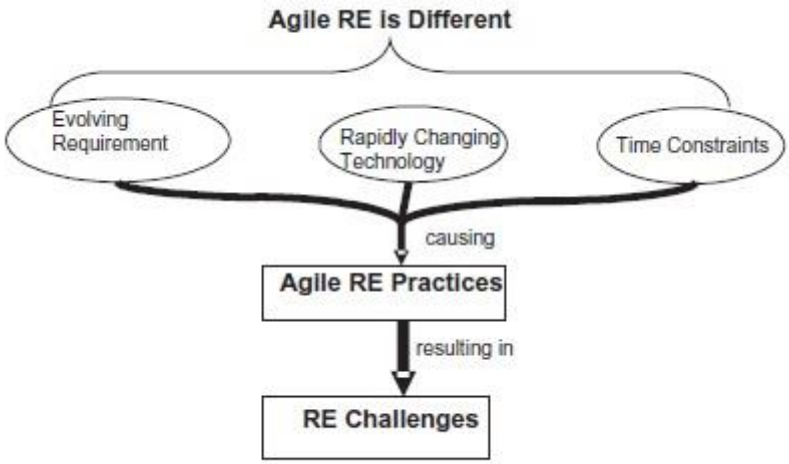
För att förenkla hur en användarberättelse skrivs, visar Cohn (2004) en mall som används i processen. Denna mall ser ut på följande sätt: Som en <*roll*>, vill jag <*funktion*> så att <*syfte*>. Ett scenario kan då vara: Som en *användare* vill jag *söka upp anställda* för att kunna *skicka ut mejl* till dem.

Det finns olika faktorer som påverkar det agila arbetet och dessa måste få den uppmärksamhet de behöver menar Ramesh et al. (2010). Ständig utveckling av krav, snabba teknologiska förändringar samt strikta tidsbegränsningar och andra resurser är några av de faktorer som har störst inflytande i mjukvaruarbetet enligt Ramesh et al. (2010). På senare tid har marknaden, systemkrav och teknologi förändrats i ökande takt och de fortsätter att förändras menar författarna. Dagens verksamheter vill implementera innovativa produkter och tjänster i olika marknader vilket kräver att system stödjer deras mål. Ramesh et al. (2010) belyser att kunder ofta ställer krav utan att förankra dem till verksamhetens mål. Författarna menar också att kraven är alltför vaga och obestämda i början av utvecklingsarbetet. Det ger upphov till kraven ständigt får ändras och göras mer konkreta under utvecklingen. Denna problematik skapar

svårigheter att följa de principer och metoder som RE bär med sig. Därför blir det svårt att utveckla kravspecifikationer som är tydliga, konsekventa och helt färdiga innan utvecklingsarbetet och utformningen påbörjas.

Ramesh et al. (2010) hävdar även att det krävs stora förändringar i kraven dels för att inte ständigt förlita sig på formella specifikationer då de ständigt förändras och dels för att inte ha ett beroende av informella dialoger mellan användare och utvecklare. Det har visat sig mycket framgångsrikt för verksamheter eftersom de får möjlighet att tidigt lansera sina produkter till marknaden (Ramesh et al., 2012).

För att förenkla denna utmaning presenteras en figur nedan:

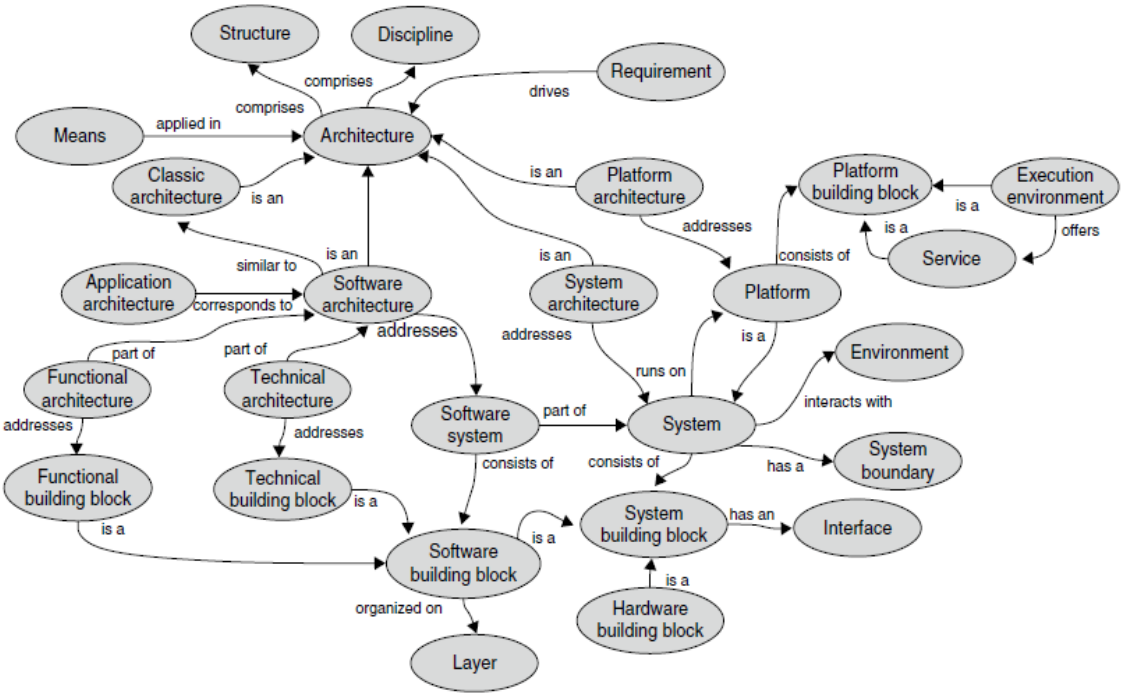


Figur 2.2 Agile RE (Ramesh et al., 2010, s 453)

Figur 2.2 visar tre faktorer såsom teknologiska förändringar, krav och tid som påverkar sättet att arbeta i agil RE. Detta kan resultera i utmaningar som är svåra att hantera för att uppnå en lyckad mjukvaruutveckling (Ramesh et al., 2010).

2.2 Krav och dess relation till arkitektur

Oavsett var krav kommer från måste dessa fångas i en mjukvaruarkitektur (Avgeriou, Grundy, Hall, Lago, & Mistrík, 2011). På så sätt är krav och arkitektur alltid relaterade till varandra, den ena kan inte existera utan den andra (Avgeriou et al., 2011). Detta hävdar även de Boer (2008) genom att belysa att kravhanterare och arkitekter behövs och att en arkitektur inte medvetet kan formas utan bådas medverkan.



Figur 2.3 Relation mellan krav och arkitektur (Vogel et al., 2011, s 40)

För att system ska vara användbara måste arkitekturen leverera på både funktionella och icke-funktionella krav. En stor utmaning har varit att hålla krav och arkitektur konsekventa under förändring. Förändring i krav medför att arkitektur och underliggande implementation också måste förändras. Nya trender inom systemutveckling såsom *agile, open source* och *out sourcing* och nya paradigmer såsom *service orientation* och *self organizing* medför en mer dubbelriktad hantering av krav och arkitektur samt snabb utveckling av hårdvarukomponenter medför ramar kring vad som är genomförbart. Detta komplicerar förändringshantering och spårbarhet från krav till arkitektur och vidare till kod (Avgeriou et al., 2011).

Detta leder oss fram till ett gap som uppstår mellan krav och mjukvaruarkitektur. Avgeriou et al. (2011) menar att det är svårt att uppnå fördelarna med spårbarhet i syfte att förstå arkitektoniska representationer. Anledningen till detta är enligt författarna att arkitektoniska kunskaper vanligtvis förblir implicita hos arkitekter.

Traditionellt har spårbarhet hanterats statiskt med korsreferenser mellan verksamhetskrav och arkitektoniska element (Avgeriou et al., 2011). Hanteringen av dessa kan stödjas med diverse verktyg. Ett krav kan då härledas till kod som utför kravet. Spårbarhet ökar då i den mening att intressenter kan se vilka delar som är färdiga under utvecklingsarbetet samt designers och arkitekter kan se vilken del av mjukvaruarkitekturen som löser ett specifikt krav. I agila-projekt med många intressenter kan kraven dock förändras under arbetets gång och att behålla denna typ av spårbarhet medför att komponenter och mjukvaruelement ständigt måste omvärderas. Intressenter kan ha svårt att kommunicera och krav kan gå emot varandra samtidigt som arkitekturen har förändrat förutsättning för att uppnå kraven (Avgeriou et al., 2011).

2.3 Arkitektur

Ämnet arkitektur förknippas det ofta med ritningar av exempelvis byggnader. Men arkitektur är inte enbart ritningar utan det är även hur saker och ting förhåller sig till varandra i ett datoriserat system enligt Bass et al. (2013).

Den arkitektur som ligger som fokus i denna uppsats är som tidigare nämnt mjukvaruarkitektur; Enligt Bass et al. (2013) en konstruktion utifrån en sammanställning av verksamhets och tekniska beslut som tagits fram. Med andra ord påverkas arkitekturen av den tekniska och sociala miljön den verkar i, samt ger upphov till att verksamheter kan uppnå sina mål. Författarna menar att bedömningen av vad som är bra eller dålig arkitektur är väl förankrat med hur väl anpassad den är till den miljö den ska verka i. Dock finns det en spridning i åsikter om exakt vad arkitektur innebär. Detta belyses av Vogel et al. (2011) som menar att ett medvetet arkitektoniskt tänkande inom mjukvaruutveckling har endast funnits i några decennier. Byggnader har en uttalad arkitektur med exempelvis planlösningar. En sådan uttalad arkitektur menar Vogel et al. (2011) inte alltid är självklar inom programvarusystem.

2.3.1 Arkitektur och struktur

Allt från neurologer till dermatologer ser människokroppens struktur på olika sätt. Vissa koncentrerar sig på subsystemen i kroppen till exempel beteende eller blodcirkulationen. Alla dessa små strukturer och subsystem är länkande till varandra och bidrar till att beskriva arkitekturen för den mänskliga kroppen. Så är det även med mjukvara och moderna system menar Bass et al. (2013). Dagens moderna system är komplicerade vilket gör det svårt att förstå alla komponenter på en gång (Vogel et al., 2011). För att öka förståelse för komplexa arkitekturer i dagens system är det viktigt att koncentrera sig på en viss mjukvarustruktur i arkitekturen. Förståelsen för en arkitektur ökar då en meningsfull kommunikation sker. Bass et al. (2013) menar att kommunikation om en arkitektur blir meningsfull när arkitekter och intressenter pratar om samma struktur inom arkitekturen. Detta leder till något som Vogel et al. (2011) nämner som arkitektoniska vyer.

Bass et al. (2013) förklarar att en struktur representeras med en så kallad vy. En vy i det här fallet är en representation av arkitektoniska element som är länkade till varandra och relationerna mellan dem. Dessa är framtagna av systemets intressenter. Vidare förklarar Bass et al. (2013) att en struktur är en uppsättning element som existerar i både mjukvara och hårdvara. Vyn är således en representation av en viss struktur (Bass et al., 2013). En specifik arkitektonisk vy skall innehålla en specifik abstraktion av systemets arkitektur i syfte att skapa möjlighet för intressenter att förstå arkitekturen i mjukvaran (Vogel et al., 2011).

Arkitektoniska strukturer kan delas in i tre olika grupper *Module structures* (modulstrukturer), *Component-och-connector structures* (komponentstrukturer) och *Allocation structures* (allokeringsstrukturer) (Bass et al., 2013). Bass et al. (2013) förklarar att en modulstruktur innehåller element som fungerar som moduler och kan betraktas som enheter av en implementation. Moduler beskriver och representerar ett system på ett kodbaserat sätt. Dessa moduler är kopplade till systemets funktionalitet och de har således ett funktionellt ansvar. När

arkitekter pratar om modulstrukturer besvaras frågor som vilket funktionellt ansvar varje modul har, vilka andra mjukvaruelement kan en modul använda, vilka andra mjukvaror som en modul använder samt arvsperspektiv - det vill säga vilka relationer moduler har till varandra. (Bass et al., 2013)

Komponent och kopplingsstrukturer innehåller element som är körningskomponenter (programexekvering) vars syfte är att agera som enheter vid exekvering/beräkning. Kopplingsstrukturer fungerar som länk och kommunikationsmedel mellan komponenterna. Dessa strukturer ger upphov till att få en övergripande bild över vilka komponenter som är viktiga vid exekvering och hur interaktionen sker mellan dem. De ger även upphov till att förstå vilka datalager som är delade, vilka delar av mjukvaran som svarar och delar information samt vilka delar av systemet som kan köras parallellt. (Bass et al., 2013)

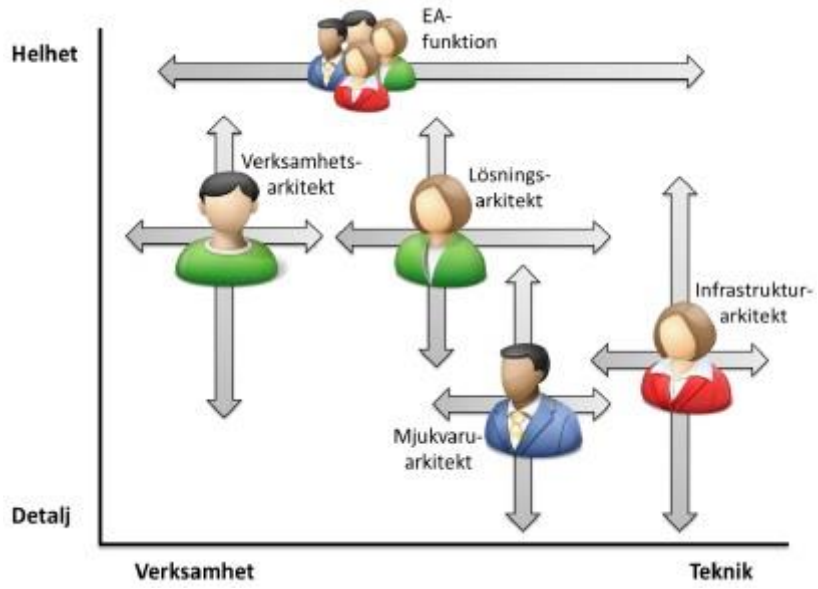
Allokeringsstrukturer visar på förhållandet mellan element av mjukvara och element i externa miljöer där mjukvaran skapas och exekveras. Denna struktur ökar förståelsen över vilka mjukvaruelement exekveras på vilken process. Den besvarar även frågor om vilka filer ett element lagras i både under utveckling och testning. (Bass et al., 2013)

Dessa strukturer ger upphov till design av arkitektur och ger en övergripande bild över hur arkitekturen skall struktureras utifrån moduler, komponenter, kopplingar och externa miljöer såsom filsystem, nätverk, processorer med mera. (Bass et al., 2013)

Bass et al. (2013) förklarar att dessa strukturer kan ses ha stort primärt inflytande vid val och skapandet av arkitektur. Varje individuell struktur kan på ett eller annat sätt påverka olika kvalitetsattribut och ger samtidigt en effektiv separation för att skapa en arkitektur för att senare kunna presentera det till intressenterna. På detta sätt kan förståelsen öka mellan arkitekter och intressenter och underlättar samtidigt kommunikationen. Dock är det viktigt att rätt struktur väljs som kan möta de kvalitetsattribut (Se kapitel 2.4.1) som krävs för att uppnå god arkitektur (Bass et al., 2013).

2.3.2 Arkitektroller

Det finns en rad olika definierade roller i en arkitektur. IASA (2012) tar upp de olika rollerna för att skapa en enad bild om vilka arkitekturroller som finns inom IT. Dessa är verksamhetsarkitekt, lösningsarkitekt, mjukvaruarkitekt, infrastrukturarkitekt samt enterprisearkitekt vilken benämns som EA-funktion (IASA, 2012). Figur 2.4 illustrerar arkitekters position i ett helhetsperspektiv:



Figur 2.4 Arkitektoniska roller (IASA, 2012, s 8)

En mjukvaruarkitekt befinner sig på en mer detaljerad teknisk nivå jämfört med exempelvis en verksamhetsarkitekt som istället har en större helhetsbild över en arkitektur.

Enterprise architecture (EA) används i många verksamheter och är en sorts planbeskrivning över en verksamhet och dess delar (Taleb & Cherkaoui, 2012). Det finns åtskilda uppfattningar om vad EA är och står för. Taleb och Cherkaoui (2012) förklarar arkitektur som:

> *"Enterprise Architecture is a complete expression of the enterprise; a master plan which acts as a collaboration force between aspects of business planning such as goals, visions, strategies and governance principles; aspects of business operations such as business terms, organization structures, processes and data; aspects of automation such as information systems and databases; and the enabling technological infrastructure of the business such as computers, operating systems and networks"* (Taleb & Cherkaoui, 2012, s 45).

Alltså är EA en sorts helhetsbeskrivning av en verksamhet där mål, visioner, principer, strategier, aspekter av automatisering av information och system. ingår. Op 't Land, Proper, Waage, Cloo, and Steghuis (2009) förklarar även EA som:

> "*Enterprise architecture (EA) is the process of translating business vision and strategy into effective enterprise change by creating, communicating, and improving the key principles and models that describe the enterprise's future state and enable its evolution*." (Op´t Land et al., 2009, s 33.)

Alltså menar även Op't Land et al. (2009) att EA är en sorts beskrivning av en verksamhets vision och strategier. För att ytterligare förenkla förståelsen för EA har Op't Land et al. (2009) dragit en slutsats om vad de anser EA är för något genom att ta del av en rad olika samlade definitioner:

> *”A coherent set of descriptions, covering a regulations-oriented, design oriented and patterns-oriented perspective on an enterprise, which provides indicators and controls that enable the informed governance of the enterprise’s evolution and success”.* (Op’t Land et al., 2009, s 34.)

Mjukvaruarkitektur är i detta perspektiv är en del av en planbeskrivande arkitektur. Zachmans ramverk och TOGAF (The Open Group Architecture Framework) är två stora ramverk som ämnar att förklara och definiera EA. TOGAF används i syfte att utveckla en arkitektur och används ofta av verksamheter då det anses vara ett moget ramverk, den ämnar alltså inte att specificera hur en arkitektur ser ut rent formmässigt (Taleb & Cherkaoui, 2011). Nedan illustreras en bild över TOGAF ramverket:

Figur 2.5 TOGAF Framework (Taleb & Cherkaoui, 2011, s 47)

Figur 2.5 visar olika sorts arkitekturer och deras relation till varandra. Dessa arkitekturer bildar en helhetsarkitektur i en verksamhet via så kallade mönsterkombinationer. Den visar även att krav är den centrala aktören i design av arkitektur. Mjukvaruarkitektur som den teknologiska arkitekturen (D) i detta fallet. Taleb och Cherkaoui (2011) förklarar denna fas som en teknologisk arkitektur vars syfte är att utveckla en infrastruktur av teknisk karaktär för att användas som en grund till identifiering av alla komponenter som skall stödja en eventuell utveckling, implementation och processer. Mjukvaruarkitekt är den roll som används i just detta område. En mjukvaruarkitekts roll i arkitekturarbetet är enligt IASA (2012) att designa och strukturera arbetet med mjukvara och att se till att möta funktionella och kvalitetsmässiga krav.

2.3.3 Mjukvaruarkitektur

Inom mjukvaruarkitektur finns en rad olika definitioner över vad mjukvaruarkitektur är. Bass et al. (2013) förklarar mjukvaruarkitektur enligt:

> *"There are many definitions of software architecture, easily discoverable with a web search, but the one we like is this one: The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both. This definition stands in contrast to other definitions that talk about the system's "early" or "major" design decisions. While it is true that many architectural decisions are made early, not all are—especially in Agile or spiral-development projects. It's also true that very many decisions are made early that are not architectural. Also, it's hard to look at a decision and tell whether or not it's "major.""*(Bass et al., 2013)

Mjukvaruarkitektur kan ses som en tillgång som skapar värde för en verksamhet som ständigt utvecklas. Mjukvaruarkitektur beskriver komponenter eller moduler som är sammanlänkande och deras relation till varandra (Bass et al., 2013). Vidare menar författarna att det är mer komplext än så. Det handlar om vad komponenterna fyller för funktionalitet och vad signifikansen är gällande deras kopplingar till varandra, det vill säga hur och varför data överförs mellan komponenterna. Arkitektur ger då upphov till att förklara hur komponenter är relaterade till varandra. Det kan således förklaras som en abstraktion av ett system. Bass et al. (2013) menar att abstraktion i denna bemärkelse är en beskrivning av ett system. Den kan i sin tur brytas ner i olika nivåer för att underlätta förståelsen av systemet. Detta är en viktig del för att öka förståelsen mellan intressenter och arkitekter, ge upphov till bättre kommunikation, skapa konsensus och bättre förhandling (Bass et al., 2013).

All mjukvarusystem har en mjukvaruarkitektur oavsett om denna arkitektur är representerad eller inte enligt Lattanze J. (2005). Att ha en mjukvaruarkitektur skiljer sig avsevärt från att medvetet utforma en. Om utvecklare börjar skriva kod eller skapa en detaljerad design utan att mjukvaruarkitekturen är designad kommer denna arkitektur uppstå av en slump. Detta medför av naturliga skäl att kvalitetsattributen (Se kapitel 2.4.1) även kommer uppstå av en slump och resultatet av detta menar Lattanze J. (2005) att en arkitektur kommer att skapas men den kanske inte kommer att vara uppskattad.

En anledning till att det är viktigt med mjukvaruarkitektur ur ett tekniskt perspektiv är, som redan nämnts, kommunikationen mellan intressenter enligt Bass et al. (2013). En annan anledning är att mjukvaruarkitektur ger upphov till att tidiga designbeslut kan fattas. Bass et al. (2013) menar att designbeslut är svåra att få rätt och samtidigt svåra att ändra senare under utvecklingen. Effekterna av designbesluten är långtgående och behöver således tas med noggrannhet. En eventuell implementering kan visa på om en arkitektur ligger i linje med de designbeslut som tas. Det kan exempelvis vara designbeslut som resurstilldelning, prestanda och stabilitet (Bass et al., 2013). Dessa är kvalitetsattribut vilka är en av de avgörande

faktorerna i uppbyggnaden av en arkitektur enligt Vogel et al., (2011) och Bass et al., (2013). Dessa kommer emellertid från krav och kommer att behandlas i kapitel 2.4.1.

Det finns många faktorer som har inflytande över hur en arkitektur skall skapas. Intressenter är alla involverade individer som har direkt eller indirekt inflytande och intresse över hur en mjukvara skall konstrueras. Dessa kan till exempel vara utvecklare, användare eller projektledare. Alla intressenter har mer eller mindre intressen och behov som de vill att systemet skall inneha eller uppnå. Behoven kan vara att systemet skall vara användbart, skalbart, interoperativt, vara lätt att optimera, billigt att utveckla och innehålla en mängd olika specifika funktioner. (Bass et al., 2013)

En annan faktor är verksamheten där mjukvaran skall implementeras. Verksamhetens mål är inte enbart krav på hur arkitekturen skall skapas, det är även strukturen och miljön där systemet skall användas (Bass et al., 2013). Detta hävdar även Smolander (2001) genom att belysa att krav bestämmer mjukvaruarkitekturen och att kraven i sin tur kommer från verksamheten. Det kan exempelvis handla om en verksamhet med expertkunniga programmerare inom klient-server kommunikation vilket då bidrar till att arkitekturen formas enligt en klient-server struktur. Professionsnivån på de anställda i verksamheten påverkar därför arbetet i att skapa och formge en mjukvaruarkitektur (Bass et al., 2013).

Arkitekter vars uppgift är att arbeta med en arkitektur för ett visst projekt ses även som en faktor. Många arkitekter använder olika metoder och tillvägagångssätt som de från tidigare projekt upplevt som lyckade. Arkitektens utbildning och tidigare erfarenhet påverkar således ett eventuellt arkitekturarbete. (Bass et al., 2013)

En annan viktig del är de kravspecifikationer som utarbetas från intressenter. Krav dokumenteras sällan fullständigt vilket leder till att konflikter uppstår mellan intressenternas mål och önskningar som i sin tur är svåra att hantera. Därför behöver arkitekter försöka engagera intressenterna i arbetet för att säkerställa att alla behov uppfylls. (Bass et al., 2013)

2.4 Mjukvarukvalitet

Kvalitet kan ses som hur väl något uppnår sitt syfte, ofta misstas kvalitet för lyx. Hull et al. (2011) driver ett exempel på detta genom att lyfta tanken om kvalitetsbilar, då de är lätt att tänka på lyxbilar såsom Rolls Royce, Mercedes och Jaguar. Men dessa bilar hade inte klarat sig länge i ett rally. Vilket påvisar att krav på kvalitet därför ska komma från användares behov som i sin tur bestäms av verksamheten (Hull et al., 2011).

2.4.1 Kvalitetsattribut

Kvalitet är en ytterst viktig del när mjukvaruarkitektoniska designbeslut ska fattas. Kvalitetsattribut som vill uppnås av intressenter måste vara tydliga i kravspecifikationer. Exempel på sådana attribut är att systemet och mjukvaran i sig skall vara säkert, tillgängligt, modifierbart, testbart och utföra något snabbt. Att ta hänsyn till dessa attribut tidigt ur ett

analytiskt perspektiv är både viktigt och ger upphov till lättare förståelse för vilken arkitektur som är mest lämpad. Dels för att det blir lättare att ta designbeslut och dels för att de blir lättare att hantera om de identifieras tidigt i arbetet. (Rago et al., 2011)

Funktionella krav påverkas mer eller mindre av attributegenskaper. Dock är kvalitet- och attributegenskaperna inte utarbetade på ett sådant sätt att de kan länkas ihop med de funktionella kraven. För att förenkla förståelsen för detta problem tar Rago et al. (2011) upp som exempel attributet ”säker”. Om det står i kravspecifikationer att ”Systemet skall vara säkert” bidrar detta inte till en utformning av en säker arkitektur. Anledningen till detta är att attributet säkerhet är allt för allmänt och behöver således specificeras ytterligare. Att endast utrycka att ett system ska vara säkert kan alltså inte ses som ett kvalificerat kvalitetsattribut. (Rago et al., 2011)

Funktionalitet i system kräver i sig ingen arkitektur. Hantering av funktionalitet är det som ligger till grund för att det finns en mening med att strukturera och designa ett system (Bass et al., 2013). Redan i bakgrunden till denna uppsats argumenteras att ett vanligt sätt att hantera krav på är att beskriva dessa i kravspecifikationer med funktionella krav. Dessa krav hjälper dock inte till att skapa kvalitativa system. I en perfekt värld kan kravställare också beskriva vilken kvalitet de vill att systemet ska uppnå, men så är ofta inte fallet enligt Cleland-Huang (2014). För att arkitekter, utvecklare och systemleverantörer ska ha möjlighet att skapa välfungerande system är en av deras utmaningar att utifrån funktionella krav och kommunikation med intressenter extrahera kvalitetskrav. Inom systemutveckling hanteras detta genom kvalitetsattribut (Bass et al., 2013).

Bass et al. (2013) beskriver sju kvalitetsattribut ingående; tillgänglighet, interoperabilitet, modifierbarhet, prestanda, säkerhet, testbarhet och användbarhet. Microsoft (2014) listar utöver dessa konceptuell integritet, underhållbarhet, återanvändbarhet, hanteringsbarhet, tillförlitlighet, skalbarhet och förmågan att utföra felavhjälpning på systemet, som vanliga kvalitetsattribut. Det finns dock fler attribut vilket måste hanteras i utvecklingsarbetet från fall till fall.

Tillgänglighet beskriver Bass et al. (2013) är ett systems förmåga att vara tillförlitligt och samtidigt återhämta sig efter krasch. Kravet uttrycks ofta som den tiden de är accepterat att system är nere under ett specifikt tidsintervall. Tillgänglighet är relaterat till säkerhet eftersom det finns attacker så som *Denial of Service* (DoS) som går ut på att otillgängliggöra system. Tillgänglighet kan också vara relaterat till prestanda eftersom det kan vara svårt att bedöma om system är otillgängliga eller endast väldig långsamma. Vid design handlar tillgänglighet om att planera för misslyckande. Grundläggande handlar det om att skapa en struktur för att upptäcka fel, återhämtning efter fel samt att undvika fel. (Bass et al., 2013)

Interoperabilitet handlar om förmågan att två eller flera system kan utbyta information genom gränssnitt i ett specifikt sammanhang. Vid systemdesign påverkar detta en arkitektur i den meningen att om systemet ska kommunicera med ett annat system måste strukturen hantera detta. Interoperabilitet hanterar också om de finns krav på att systemet ska interagera i ett senare skede måste gränssnitt som möjliggör detta finnas. Designbeslut som ska fattas vid

interoperabilitet är: *hur hittas tjänster och hur kommunicerar de via gränssnitt?* (Bass et al., 2013)

Modifierbarhet hanterar systemets förmåga att förändras. Arkitektur kan stödja mekanismer som möjliggör förändring i högre lager. I de lägsta lager kan system alltid förändras genom kod vilket kräver att en utvecklare ändrar i källkod samt att systemet måste kompileras om och distribueras ut på nytt. I kontrast till detta kan utvecklare skapa mekanismer som tillåter slutanvändaren att förändra systemets beteende i högsta lagret via användargränssnitt. Motiveringen till att införa mekanismer för förändringsbarhet är kostnad. Från en arkitekts perspektiv handlar de om att bilda sig en uppfattning om, vad som kan förändras, vad är troligt att det förändras, när förändras något, vem utför förändring samt vad är kostnaden för förändring. Taktiker för att skapa bättre modifierbarhet innefattar att reducera storleken på moduler, öka sammanhållningen, skapa låg koppling och binda förändringar så sent som möjligt. (Bass et al., 2013)

Prestanda handlar om tid och systemets förmåga att utföra en operation inom ett visst tidsintervall. Det kan till exempel vara förfrågningar mot en server eller antal användare som kan utföra en uppgift samtidigt. Prestanda är också kopplat till skalbarhet genom att öka kapaciteten för systemet och fortfarande behålla prestanda. Prestanda kan mätas genom *latens*, en sista tidpunkt för utförande, genomströmning i form av hur många transaktioner som systemet kan utfara under ett specifikt tidsintervall, *jitter* – de tillåtna variationerna i latens och de antal processer som systemet inte hann med eftersom systemet var upptaget. För att hantera prestanda måste en arkitekt beakta belastningen på en viss resurs samt skapa en struktur för att hantera den belastningen enligt de krav som finns. (Bass et al., 2013)

Säkerhet är ett mått på systemets förmåga att skydda data och information från otillåten användning och samtidigt tillåta tillgång för de som har rätt behörighet. (Bass et al., 2013) tar upp tre kategorier som beskriver säkerhet:

- *Konfidentialitet* hanterar att endast de som ska ha tillgång får tillgång
- *Integritet* handlar om att säkerställa att information är tillförlitlig
- *Tillgänglighet* information och data ska vara tillgänglig för de som kräver det

arkitektoniska taktiker för att hantera säkerhet innefattar: detektera attacker, motstå attacker, handla på attacker och återhämtning efter attacker (Bass et al., 2013).

Testbarhet handlar om att på ett så enkelt sätt som möjligt få systemet att visa sina fel. För att utföra ett test måste möjlighet finnas att hantera en komponents input och kontrollera komponentens output. Testscenario kan skrivas av utvecklare, kunder eller en specifik testgrupp (Bass et al., 2013). Acceptance Test Driven Development (ATDD) och Behavior Driven Development (BDD) driver ner testscenario till utvecklare som en del av acceptanstest (Carrera et al., 2013). Taktiker för att göra ett system testbart innefattar att möjliggöra kontroll och observation av systemets tillstånd samt begränsa komplexitet (Bass et al., 2013).

Användbarhet handlar om hur lätt det är för en användare att utföra en viss uppgift. Användbarhet menar Bass et al. (2013) är en av de billigaste och lättaste sätten att förbättra ett systems kvalitet, eller i alla fall användarens upplevelse av kvalitet. Det är dock svårt att

beskriva någon taktik för att bearbeta användbarhet och interaktion med användaren är oundvikligt när det gäller användbarhet.

2.5 Design och framtagande av arkitektur

Framtagandet av en arkitektur är starkt kopplad till de krav som ställs, det vill säga funktionella, kvalitets- och verksamhetskrav. Arkitekturen formas utifrån dessa krav och beskrivs av Bass et al. (2013) som arkitektoniskt signifikanta krav (ASR) vilka driver fram en mjukvaruarkitektur. Varje arkitektur formas utifrån olika formbara krav och dessa skall prägla hela mjukvaruarkitekturen. Arbetet handlar om att identifiera och prioritera krav i form av verksamhetsmål och på så sätt bestämma de arkitektoniska drivarna. Verksamhetsmål skall i sin tur konverteras till kvalitetsattribut och sedan fungera som drivare (Bass et al., 2013).

2.5.1 Attribut-Driven Design (ADD)

Det finns olika metoder som stödjer arbetet i framtagandet av en arkitektur. En av metoderna som både Avgeriou et al. (2011) och Bass et al. (2013) tar upp är *Attribute-driven design* (ADD). Wood (2007) förklarar metoden som ett verktyg för att bestämma en mjukvaruarkitektur där processen i att designa arkitekturen baseras på kvalitetsattribut i form av krav som en programvara måste uppfylla för att nå verksamhetsmålen (Wood, 2007). Enligt Bass et al., (2013) och Wood (2007) är ADD en rekursiv process, det vill säga en repeterande process vars syfte är att tillämpa en mjukvaruarkitektur som ger upphov till att möta intressenternas krav i form av kvalitetsattribut och på så sätt forma mjukvaruarkitekturen.

Den rekursiva processen kan beskrivas som en cykel; planera, agera och slutligen kontrollera. Det som planeras är vilka typer av element som skall användas i arkitekturen. Detta görs utifrån analys av kvalitetsattribut och begränsningar av krav. Nästa steg är att agera, det som utförs i denna fas är att de valda elementet realiseras i syfte att möta kvalitetskrav. Slutliga steget, kontrollera, innebär att analysera resultatet av designen. Detta görs för att avgöra och fastställa om kraven är uppnådda eller inte. Denna rekursiva process repeteras fram tills det att alla arkitektoniska krav har uppnåtts (Wojcik et al., 2006).

Resultatet av ADD är en design som är dokumenterad i olika typer av arkitektoniska strukturer och vyer (Wojcik et al., 2006). Inkluderat i dessa är modul, komponent och allokeringsstrukturer (se kapitel 2.3.1). Detta i syfte att visa hur ett system delas upp i olika element, vilka delar som skall ingå i systemets strukturella arkitektur samt vilka egenskaper och strukturella relationer varje element har. Men även vilka interaktioner som kommer att ske mellan de olika elementen som har identifierats och även vilka egenskaper dessa interaktioner har.

För att förenkla denna sammanfattning från Wojcik et al. (2006) redovisas figur 2.6 nedan:

Output	Beskrivning
Mjukvaruelement	Artefakter med definierade egenskaper, relationer, roller och ansvar i syfte att utgöra en arkitektur.
Roll	Uppsättning av olika ansvarsområden som mjukvaruelement har.
Ansvar	Funktionalitet, information och data som mjukvaruelementen erbjuder.
Egenskaper	Mjukvaruelements egenskaper i form av namn, typ, kvalitetsattribut etc.
Relation	En beskrivning över hur olika mjukvaruelement är associerade med varandra men även hur interaktionen sker mellan dem.

Figur 2.6 Sammanfattningsbild illustrerad utifrån teori presenterad av Wojcik et al., (2006).

Andra återstående metoder som Lattanze J. (2005) tar upp är *Architecture Centric Development Method* (ACDM), *Quality Attribute Workshop* (QAW), *Architecture Tradeoff Analysis Method* (ATAM). Dessa metoder hjälper arkitekter att designa en god arkitektur enligt Lattanze J. (2005), dock belyser författaren att dessa metoder innebär problem. En anledning är att metoderna är skapade utan en utvecklingsfilosofi, det vill säga utan någon sorts livscykel eller process i åtanke. Och att de inte besitter något tillvägagångssätt i att skräddarsy dessa med syfte att passa in i verksamheter. Lattanze J. (2005) hävdar att om arkitekter vill uppnå metodernas maximala effekt krävs det att dessa metoder används tillsammans med en projektmetod.

ACDM är en metod som används för att skapa en arkitektur då kraven är förstådda och hanterade. Arkitekturen här utvecklas tidigt och justeras iterativt tills att arkitekter och utvecklingsteam är överens om att en systemimplementation kan genomföras och att arkitekturen kommer att kunna möta intressenternas behov och krav (Lattanze J., 2005). Även i denna metod handlar arbetet om att forma arkitekturen utifrån arkitektoniska drivare såsom funktionella krav, designbeslut och kvalitetsattribut i syfte att möta verksamheters mål. Resultatet av metoden är en uppdelning av ett system som innehåller beskrivning av olika element, subsystem och deras interaktion med varandra där krav spelar en central roll (Lattanze J., 2005).

Design av mjukvaruarkitektur formas och drivs utifrån krav som ställs, och utan kraven kan arkitekter inte designa en välanpassad arkitektur - designbeslut och arkitektoniska krav är således beroende av varandra (Boer & Vliet, 2009).

2.5.2 Arkitektoniska mönster

Som setts hittills är de många olika källor som kan påverka arkitektur. Vilket leder till att framtagningsprocessen av arkitektur är komplicerad. Som ett svar på detta har mönster skapats för att lösa vanligt förekommande problem. Mönster uppkommer från praktiskt arbete och kan således inte uppfinnas, de måste upptäckas (Bass et al., 2013). De paketerar designbeslut för återanvändning (Bass et al., 2013). Arkitektonisk design startar sällan från noll och dagens erfarna arkitekter skräddarsyr arkitektur genom att sammansätta och välja olika mönster enligt Bass et al. (2013). Arkitektoniska mönster beskriver och definierar ett delsystems regler och funktioner men även riktlinjer i syfte att beskriva och organisera vilken relation de har till varandra (Buschmann, Meunier, Rohnert, Sommerlad, & Stal, 1996). Valet av arkitekturmönster påverkar designarbetet och kan ses som ett grundläggande designbeslut som enligt Buschmann et al. (1996) tas tidigt i designfasen. Mönster ger upphov till återanvändning i programutveckling på så sätt att gamla mönsterlösningar kan återanvändas vid en eventuell ny programutveckling. Anledningen till denna återanvändbarhet är på grund av en hög abstraktionsnivå som är ett resultat av mönsterlösningar (Buschmann et al., 1996).

Mönsterlösningar underlättar arkitekturarbetet genom att skapa ett gemensamt språk mellan designers. Detta minskar komplexiteten i arbetet och ger upphov till att designarbetet kan verka på en högre abstraktionsnivå (Buschmann et al., 1996).

2.6 Teoretisk sammanfattning

RE är en del av systemutveckling men krav hanteras inte alltid av utvecklare eller intressenter som har insyn i systemutveckling vilket leder till att dessa processer kan ha en begränsad publik. Den begränsade publiken leder också till kravspecifikationer ofta uttrycks i funktionella krav, vilket inte har några direkta konsekvenser för en mjukvaruarkitektur. Metoder så som ADD har tillkommit för att agera som en förstudie och extrahera icke-funktionella krav från verksamheten och intressenterna. De icke-funktionella kraven handlar mest om vilken kvalitet som systemet ska uppnå och beskriver genom kvalitetsattribut vilken förmåga systemet ska besitta. I kapitel 2.4 går vi igenom kvalitetsattribut som framställs som viktiga från två källor. Att behandla kvalitet genom kvalitetsattribut ger som vi ser upphov till taktiker för att möta dem. Vidare kan de också renodlas till mätbara krav. Att hantera kvalitetskrav är en viktig faktor för att fatta designbeslut om vilken mjukvaruarkitektur som är mest lämpad då kvalitetskrav precis som funktionella krav betraktas som arkitektoniskt signifikanta krav som driver fram en mjukvaruarkitektur. Dock är inte alltid kvalitetskrav tydligt uttryckta vilket gör att de blir svåra att identifiera och i sin tur svåra att hantera om de identifieras sent i utvecklingsarbetet. Detta kan sammanfattas som att arkitekter är influerade av kraven från intressenterna, struktur och mål med verksamheten, den tekniska miljön samt arkitekternas egna erfarenheter och bakgrund (Bass et al., 2013).

RE bygger på spårbarhet och vilka designbeslut som skall tas påverkas av den spårbarhet som råder i de krav som intressenter ställer. Detta ger upphov till att de designbeslut som

mjukvaruarkitekter tar skall kunna möta de mål som verksamheten har men även att kunna möta förändring. Förändring är något som RE traditionellt inte hanterar på ett bra sätt och i agila-metoder sker arbetet under ständig förändring, detta förändrar dock inte de delprocesser som RE innehåller men förändrar när de utförs (De Lucia & Qusef, 2010). I agila-metoder hanteras ofta krav genom *User Stories*. Dessa kan ses som krav i mindre portioner och beskrivs ofta i naturligt språk vilket ofta kräver extrahering av icke-funktionella krav så att designbeslut kan fattas (Rago et al., 2011).

Att kunna hantera krav på ett sätt som ger upphov till att en kravbild lättare kan omfamnas ger upphov till att lättare kunna hantera förändringar både vad gäller krav men även teknologiska förändringar. Många verksamheter använder sig av agila-utvecklingsmetoder i syfte att kunna bemöta och tackla förändringar. Agila-utvecklingsmetoder är en av nycklarna till detta menar Aurum and Wohlin (2005). Ständigt nära samarbete med kunder och intressenter gör att krav som ändras under utvecklingen kan hanteras på ett sätt som inte stoppar arkitekturarbetet av mjukvara.

Arkitektur är en struktur på mjukvara som avslöjar designbeslut. Mjukvaruarkitektur kan med större ramverk beskrivas som en delarkitektur i EA (IASA, 2012). Mjukvaruarkitektur beskriver den tekniska bestyckningen på mjukvarukomponenter eller moduler och deras relation till varandra (Bass et al., 2013). Mjukvaruarkitektur kan uppkomma slumpmässigt eller ändamålsenligt (Lattanze J., 2005). Den slumpmässiga mjukvaruarkitekturen är ofta förknippad med agil-utveckling då väldigt lite tid initialt läggs på att förstå helhetsbilden (Lattanze J., 2005). Den ändamålsenliga mjukvaruarkitekturen kommer från en verksamhet och kopplingen går genom krav eller kravspecifikationer. Dock är en arkitektur inte influerad direkt av en agil- eller traditionell-utveckling. Dessa projektmetoder påverkar direkt arbetet med kravhantering genom att bestämma när i livscykeln som designbeslut fattas och hur omfattande designbesluten är, de arkitektoniska resultatet är detsamma (De Lucia & Qusef, 2010).

3 Metod

Vi har fram till detta kapitel bearbetat vårt ämne teoretiskt och behandlat aspekter i hur teori ställer sig i förhållandet mellan kravhantering och design av mjukvaruarkitektur. Det finns enligt Jacobsen och Sandin (2002) två olika problemställningar. Dessa är beskrivande eller förklarande problemställningar. Vid beskrivande problemställningar beskrivs hur ett fenomen ser ut i verkligheten det finns också beskrivande problemställningar som utgått ifrån hur något har utvecklats och ett historieperspektiv måste då beaktas. Förklarande problemställningar utgår ifrån orsak och verkan och kan vara av karaktären ”vad är orsaken till att något inträffar”.

Vår problemställning utgår ifrån att vi vill beskriva förhållandet mellan krav och design av mjukvaruarkitektur och är således en beskrivande problemställning. Jacobsen och Sandin (2002) beskriver att studier i sina ytterligheter kan vara antingen intensiv eller extensiv, skillnaden är djup eller bredd. Djupa studier har många variabler och få enheter, breda studier har få variabler och många enheter (Jacobsen & Sandin, 2002). Denna uppsats behandlar ett komplext ämne med många variabler därför har vi valt en intensiv studie. Enheterna i vårt fall är således personer som har kännedom om hur krav hanteras och hur det påverkar arkitektur arbetet, det är också förutsatt att personerna hanterar design i arkitektoniska termer.

3.1 Generalisering

Syftet med uppsatsen är att identifiera konsekvenser som kravhantering medför vid design av mjukvaruarkitektur. Syftet med mjukvaruarkitektur är inte helt givet och används på olika sätt i skilda miljöer, vilket bidrar till att ämnet är komplext och uppsatsen inte kan generalisera eller beskriva alla konsekvenser som krav- och kravhantering medför. Eftersom problemställningen är av en beskrivande karaktär vill vi hitta nyanser vilket kommer leda till generaliseringsproblem. Våra enheter beskriver endast den kontext de har verkat i eller verkar i. Det som kan stärka förmågan att generalisera är den erfarenheter som informanterna innehar.

3.2 Intervjuer

Vi har valt att samla in kvalitativ data genom intervjuer. Jacobsen och Sandin (2002) beskriver att kvalitativ data kan samlas in med helt öppna intervjuer med eventuellt tema. I kontrast till detta samlas kvantitativ data in genom frågeformulär med enbart fasta svar. I denna uppsats har vi valt att till viss grad strukturera vilka ämnen som ska behandlas med frågor. Vi har gett informanterna utrymme till att beskriva sin upplevelse av ämnet utan att styra samtalet för mycket. Eftersom uppsatsen riktar sig till yrkesverksamma arkitekter har platsen valts till deras arbetsplats för att störa informanternas arbete så lite som möjligt. Detta är således inte en konstlad plats som Jacobsen och Sandin (2002) beskriver. Vi har innan intervjuerna ägt rum delgett information om uppsatsens syfte och tema. Intervjuerna har också spelats in efter godkännande från informanterna.

3.2.1 Intervjugenomförande

Intervjuerna med informanterna (P1 och P2) genomfördes på deras arbetsplatser. Intervjuförfarandet skedde personligen, för att få en så förtrolig stämning som möjligt jämfört med om intervjuerna hade bedrivits via telefon (Jacobsen & Sandin, 2002). En annan anledning till varför vi valde att genomföra intervjuerna personligen är på grund av att vi har många öppna frågor och att ett svar kan ha en annan betydelse då naturligt kroppsspråk används. Telefonintervjuer blir då som Jacobsen och Sandin (2002) nämner en olämplig intervjumetod av den anledningen att informanterna har en större möjlighet att vinkla svaren eller helt enkelt inte tala sanning, samtidigt kan de ge oss en orättvis bild och mindre förtroende för den data vi samlar från intervjuerna.

Inspelningsverktyg som användes var två stycken mobiltelefoner. Anledningen till antalet mobiltelefoner var för att säkerställa backup ifall någon inte skulle fungera. En annan anledning till varför intervjuerna gjordes med hjälp av ljudinspelningsverktyg var för att slippa lägga fokus på anteckningar och således försöka uppmärksamma allt som sades. Vi frågade även informanterna om intervjuerna kunde spelas in och att namn kunde nämnas i transkripten, vilket de godkände.

Vad gäller intervjuunderlaget har alla frågor inte ställts under intervjun. Utan de har endast hanterats som underlag under intervjugenomförandet. Anledningen till detta är för att intervjuerna har haft en öppen karaktär där informanterna fritt fått svara och på så sätt också besvarat de andra frågorna från underlaget i informanternas svar och berättelser. På de sättet har vi försökt att inte styra intervjun på ett sådant sätt som begränsar informanternas svar.

3.3 Urval

Första steget i urvalsprocessen är att skaffa en överblick över populationen enligt Jacobsen och Sandin (2002). Urval ska sedan göras utifrån denna population (Jacobsen & Sandin, 2002). Populationen i denna uppsats utgörs främst av de som arbetar med arkitektur och de som ställer krav på system. Vi har delat in populationen i grupper om kravställare och arkitekter. Det är dock inte givet att de som ställer krav på mjukvara har en förståelse för eller tänker på hur den tekniska strukturen i mjukvara ska vara uppbyggd och således inte kan delge relevant data för vår uppsats. Vidare bestämdes att vi hade möjlighet att intervjua fyra personer under förutsättning att det skedde inom en viss tidsram. Grupperna blev snarare personer med kunskap och erfarenhet av kravställning på mjukvara och personer som arbetar med systemdesign och arkitektur utifrån krav. Urvalet gjordes med en kombination av kriterierna, bredd och variation, den typiska samt snöbollsmetoden (Jacobsen & Sandin, 2002).

Vi tillfrågade fyra arkitekter varar två ställde upp, en ytterligare ville ställa upp men hade semester under den uppsatta tidsintervallen. Den sista svarade inte på förfrågan.

Detta ledde till att två intervjuer utfördes vilket gav data från två olika perspektiv. Ett kundperspektiv samt ett leverantörsperspektiv. Båda informanterna hade erfarenhet av olika projekt vilket bidrog till att många nyanser framkom.

3.4 Analys

Jacobsen och Sandin (2002) beskriver tre olika faser av analysprocessen. Första fasen efter insamling av data menar Jacobsen och Sandin (2002) är beskrivning. Som tidigare nämnt har vi spelat in alla intervjuer och sedan transkriberat dem. I transkripten har vi ordagrant skrivit ner allt som sagts med undantag för upprepningar och onödiga instämningar såsom ”mm” och ”ja”. Därefter följer enligt Jacobsen och Sandin (2002) systematisering och kategoriserings fasen. Intervjuerna har i sig strukturerats i enlighet med den teoretiska strukturen vi har i teorikapitlet. Därefter har vi som forskare läst igenom och kommenterat olika påstående genom en teoretisk lins. Dessa kommentarer har sedan jämförts, diskuterats och kategoriserats enligt krav, kravhantering, mjukvarukvalitet samt arkitektur. I empirikapitlet lyfts resultatet av denna analys och direkta citat används för att härleda till intervjuerna.

3.5 Validitet och reliabilitet

Jacobsen och Sandin (2002) skiljer på intern giltighet (validitet) och extern giltighet. Validiteten i denna uppsats stärks genom att utföra metodtriangulering (Jacobsen & Sandin, 2002). Teorierna som lyfts i litteraturgenomgången är av blandad karaktär och våra slutsatser jämförs med dem.

Vi bedömer att urvalet stärker validiteten eftersom de är arkitekter som behandlar arkitektoniska designbeslut genom krav dagligen och har god erfarenhet av detta. Vi ser ingen anledning till att informanterna skulle lämna oriktig information eftersom ämnet vi behandlar inte är känsligt för de inblandade. Enligt Jacobsen och Sandin (2002) bör källorna vara oberoende av varandra. De informanterna vi valt ut är helt oberoende och arbetar dessutom utifrån olika perspektiv. Information som uppkommer från olika informanter kan därför bedömas som giltig.

3.6 Etik

Jacobsen och Sandin (2002) identifierar tre faktorer som är viktiga att behandla när det gäller etiska aspekter av informanter och intervjugenomförandet. Dessa är: informerat samtycke, krav på privatliv och krav på att bli korrekt återgiven.

Informerat samtycke har vi behandlat genom att beskriva för informanterna vad studien handlar om, syftet med undersökningen, vilka vi är samt om de finner det intressant att ställa upp på intervjuer. Ingen press har lagts på om informanterna skall ställa upp utan det har funnits möjlighet för informanterna att tacka nej.

Ämnet som vi behandlar kan inte ses som statisk fakta således är informanternas subjektiva uppfattning viktig oavsett hur den tolkas. Den kan inte heller verka stötande för informanternas privatliv eller arbetsliv. Vi bedömer således att data som vi samlat in från informanterna inte kan uppfattas eller bedömas som känslig. Vi frågade informanterna om deras namn kunde nämnas i transkripten vilket de gav sitt godkännande till.

Enligt Jacobsen och Sandin (2002) skall data som presenteras vara riktig data och skall således inte ha manipulerats efter insamling. Detta har säkerställts genom att skicka transkriberingen per mejl till informanterna. På detta sätt har vi i denna studie gett möjlighet för informanterna att kontrollera transkripten och göra eventuella ändringar. Detta har gett informanterna möjlighet att bli korrekt återgivna (Jacobsen & Sandin, 2002).

3.7 Forskningskvalitet

Två intervjuer genomfördes på varje informants egen arbetsplats. Emellertid kan detta anses som få intervjuer för att kunna uppnå en mättnad i data. Detta kan bära med sig en risk att andra versioner från andra individer kan ha gett en större generell bild över ämnet (Jacobsen & Sandin, 2002). Dock har vi lyckats bedriva intervjuer med två personer med olika perspektiv en intervju med ett kundorienterat perspektiv samt en intervju med ett leverantörsorienterat perspektiv. Detta har gett oss en bättre möjlighet att kunna beskriva ämnet från olika vinklar.

4 Empiri

I detta kapitel presenteras resultatet efter vår analys och detta material kommer ligga till grund och användas i vidare kapitel för diskussion och slutsatser.

4.1 Presentation av informanter

P1 heter Patrik Johnsson och arbetar sedan åtta år tillbaka på Cybercom Group i Malmö. Han är Elektroingenjör i grunden och inledde sin karriär som utvecklare. De senaste tre åren har han arbetat mer med arkitektur. Idag arbetar han nästan uteslutande med arkitektur.

P2 heter Almir Zerim och arbetat som IT-arkitekt på trafikverket. Almir Zerim är utbildad datolog och sociolog vid Lunds Universitet. År 2000 arbetade han på Telia och började ganska fort arbeta med arkitektur. Han är certifierad projektledare och tidigt in på hans karriär gav han upp projektledningsyrket och började koncentrera sig på arkitektur istället. Han arbetar för tillfället med systemarkitektoniska och integrationsarkitektoniska frågor vilket är hans spetskompetens.

4.2 Krav och kravhantering

P1 menar att krav ska tas fram med grova arkitekturprinciper i bakhuvudet och fungera mer dubbelriktat enligt:

> *”Jag skulle vilja säga att arkitekturen ska påverka kraven och kraven ska påverka arkitekturen. Så att, man ska ha arkitekturprinciper, väldigt grova riktlinjer som man lutar sig mot och som man alltid följer. Och de här principerna skall man liksom ha i bakhuvudet när man tar fram sina krav.”* (Transskript 1, rad 16)

P1 beskriver att krav skiljer sig väldigt mycket i sin uttrycksform mellan organisationer. P1 menar att en sund inställning är att först hantera affärskrav uttryckta i vilket behov som verksamheten har samt varför behovet finns. Detta skall sedan omsättas till en teknisk modell som uttrycker tekniska krav. I denna hantering beskriver P1 en lager indelning. Om krav kan leda till olika implementeringar så menar P1 att det kan finnas brister i arkitekturen och det behöver kommuniceras till en helhetsarkitekt.

I P1 beskriver att i den miljön där han verkar är kravspecifikationer inte grunden för hans arbete. Det är ofta så att kravbilden inte är känd och arbetet inleds med en nulägesanalys målet med detta menar P1 är att kvantifiera och kartlägga så mycket som möjligt hos verksamheten, syftet med detta är att renodla och minska utrymmet till tolkning av krav för utvecklare som ska implementera kraven.

Spårbarhet hanteras i P1s fall genom utvecklingsverktyg såsom versionshantering och issue tracking. Dessa inför spårbarhet mellan features och kod. Denna typ av spårbarhet i agil-

utveckling leder dock till svårigheter att få en helhetsbild av mogna system enligt P1. Tidigare användes UML-diagram för att dokumentera och specificera system med detta görs mer sällan nu. Eftersom det är ett problem med system som endast dokumenteras genom user stories motiverar de arbetet med att ta fram bättre specifikationer enligt P1. Utmaningen är att skapa en helhetsbild som inte blir gammal.

> *”Det är ju mycket en avvägning, så en specifikation som är lätt att uppdatera kanske inte innehåller så mycket information så kanske den inte är riktigt så värdefull. Så det gäller att hitta en bra balans där.”*(Transskript 1, rad 28).

Det finns enligt P1 ingen standard i hur krav lämnas från kunden. Men understryker att de är upp till utvecklaren att tolka krav och det är då bra om dessa är bra beskrivna.

Krav är något som P2 även ser som en viktig del i arkitekturarbetet och en förutsättning för en ändamålsenlig utveckling. Krav handlar mycket om att det skall kunna knytas till verksamhetens behov och mål för att slutligen kunna beskrivas i en arkitektur. P2 menar att en tydlig verksamhetsarkitektur ger upphov till att krav lättare kan knytas och att gapet mellan krav och verksamheters behov minskas. Men krav är oftast knutna till att människor i verksamheter känner en irritation och vill således ändra något utan att ta hänsyn till verksamhetens egentliga mål. Han menar att många verksamheter försöker ställa för många skallkrav och försöker detaljstyra för hårt vilket missgynnar verksamheterna i slutändan. Anledningen till det är att skallkraven inte kan knytas till verksamhetens behov eller att helt enkelt verksamheten övertolkas.

Det går tyvärr inte att alltid få det bästa menar han om inte det bästa passar in i det egentliga behovet. Då handlar det mycket om att vara flexibel i att hantera krav menar P2 dels för att fatta vettiga designbeslut och dels för att ha en bra förståelse över hur kravbilden ser ut. För att åstadkomma denna flexibilitet tar P2 upp ämnet agil-utveckling.

Många projekt idag använder inte traditionella vattenfallsmetoder enligt P2 detta på grund av att verklighetsbilden är mycket mer dynamiskt, att alla projekt mer eller mindre använder sig av cykler under utvecklingen och nämner då agila-utvecklingsmetoder. Anledningen till varför utvecklingen och hantering av krav bedrivs agilt menar P2 beror på att kunna hantera eventuella förändringar både inom och utanför verksamheter. I Trafikverkets fall används konkreta kodhanteringsverktyg som exempelvis TFS väldigt mycket. Agil-utveckling gynnar utvecklingsarbetet väldigt mycket menar P2, man måste kompromissa, vara flexibel i att hantera krav. Vilket då kan ses som en konst för att utveckla en arkitektur:

> *”/…/ Arkitekturen är konsten att göra kompromisser/…/”* (Transskript 2, rad 10).

P2 bedriver just nu ett forskningsprojekt där han är projektledare och menar att kravbilden i de projektet är väldigt dålig på grund av att det är ett forskningsområde. P2 menar att projektet bedrivs för att de vill utforska något, lära sig något men samtidigt vet de inte vad de vill göra. Dock menar P2 att det inte heller går att ge sig in på något och sedan börja utveckla. Det krävs enligt P2:

> */…/ha något slags ramverk som är antingen uttryckt i krav eller i någon slags intention, behovsbeskrivning eller någonting. /…/ någon slags målbild.*(Transskript, rad 10).

Kraven har då tolkats och hanterats av en testledare under projektets gång genom att rita processer för att skapa sig en förståelse över hur allt hänger ihop för att kunna omfamna helhetsbilden.

Problemet som P2 upplever med andra projekt som medverkats i är att ca 20 % av alla projektbudgetar går faktiskt åt att förundersöka verksamheten och dess arkitektur och behov. Detta på grund av att många verksamheter enligt P2 inte har uppdaterade processmodeller eller någon tydlig verksamhetsarkitektur, det finns helt enkel ingen dynamik i kartläggningen av verkligheten. Denna kartläggning av verkligheten (nulägesanalys) är således P2 tvungen att göra för att helt enkelt kunna förstå kraven och designa en arkitektur. Detta är ett vanligt scenario i P2 vardagliga arbete.

P2 belyser även att kravhantering är stark kopplat till arkitektur och att det på så sätt ger riktlinjer om vad som skall hanteras. P2 tar upp TOGAF ramverket som ett sätt att hantera krav speciellt i agila-utvecklingsprojekt. Han menar även här att flexibilitet är ett faktum vid hantering av krav, dels för att hitta rätt balans och dels för att hela tiden ha ett nära samarbete med kunden i syfte att inte tro att kraven är fastställda från första dagen i utvecklingen. P2 tycker att det är bisarrt att behöva ta upp en sådan självklarhet vad gäller krav och kravhantering i IT-världen. P2 menar att det går att jämföra med privatpersoners vardagliga liv, han tar upp ett bilköp som exempel:

> */…/”går du till en affär och säger vadå jag har 100 000 ge mig en bil? nä du har någon slags bild vad du vill ha och hur mycket pengar”/…/* (Transskript 2, rad 42).

Denna sorts självklara kravmognad menar P2 inte är fullt utvecklad än inom IT-branschen, det vill säga hur arkitektur kan bidra med hjälp till verksamheters krav och behov. P2 belyser även att en tydlig kravbild inte alltid är ett faktum.

Krav kan betraktas som funktionella och icke-funktionella. Krav som P2 stött på i sitt yrkesliv är av blandad art, både funktionella och icke-funktionella. På P2s nuvarande arbetsplats, Trafikverket, sker en omorganisering där en ny rotel skall införas. Rotelns främsta uppgift är att se till att utvecklingsarbetet ligger i linje med vad som efterfrågas av kunderna och på så sätt hamnar rätt i drift, roteln ska uteslutande jobba med krav. Roteln går under namnet validering och integration. Anledningen till omorganiseringen menar P2 beror på att vikten av att ha en gemensam teststrategi och kravhanteringsstrategi har blivit mer uppmärksammat av ledningen.

P2 belyser även kravens kvalitativa innehåll och beskrivning. Han menar emellertid att kraven är uttryckta (speciellt icke-funktionella) luddigt vilket ger upphov till arbetet bedrivs i långsammare takt då mycket tid läggs på att försöka förstå de icke-funktionella kraven då det är en viktig del i utvecklingsarbetet:

> *” /…/det första man reagerar som arkitekt där är att man kan uttrycka krav i ”ja vi vill ha bättre prestanda” eller god prestanda, ja vad är det för något alltså? Så det är väldigt ofta man uttrycker sig så pass luddigt att du ändå måste köra ett varv till och fastställa vad det innebär, är de svarstid på fem sekunder eller fem millisekunder alltså.”*(Transskript 2, rad 14).

4.3 Mjukvarukvalitet

Mjukvarukvalitet uppnås bäst om krav går att testa enligt P1 och då är testbarhet ett kvalitetsattribut som krav måste uppnå. Enklast hanteras detta av att en testledare är med och formar kraven så att de blir mätbara. P1 upplever att detta ofta är eftersatt och traditionellt sätt har inte Cybercom varit delaktiga i kravutformning, men på senare tid har de kommit in tidigare vilket skapar en bättre förståelse. Som en kontrast till detta beskriver P1 offentlig upphandling. Då är redan kravbilden uppfylld och leverantören får besvara om de kan uppfylla den kravbilden som ett kontrakt. Kvalitetsaspekter kan finnas med i denna kravbild samt vissa krav på arkitektur så som tjänsteorientering. Det ges ofta i denna process också hur anbudet kommer bedömas, vilket P1 menar visar hur svaret ska se ut.

P2 beskriver icke funktionella krav som kvalitetsattribut vars syfte är att uppnå kvalitet i mjukvara. Dock krävs det att attributen tillgodoses på ett bra sätt och att det då ligger på arkitekterna och arkitektur. Han tar upp ett projektexempel på Sony Ericsson där de hade en så kallad ”Architecture Board” där alla arkitekter från olika enheter lade fokus på kvalitetsattribut. De var tvungna att uppfylla vissa skallkrav för att kunna lyckas få ett godkänt system. Detta tillvägagångssätt gav upphov till att systemet kunde rangordnas med hänsyn till icke-funktionella krav. Han menar att de icke-funktionella kraven alltid måste vägas in med de funktionella kraven.

P2 belyser även att arbetet med att uppnå kvalitet påverkas även mycket av de politiska spelet. Arkitekter måste tyvärr ta hänsyn till sådana aspekter:

> *”Om vi tittar på databaser att ha både Oracle och MSSQL ingen har råd med det du får bestämma dig. Är du ett ”Microsoftföretag” då använder du MSSQL. Så de finns väldigt mycket sånt som påverkar som inte har direkt med de här attribut som påverkar eller något annat som är mer strategiskt och på de sättet politiskt”* (Transskript 2, rad 24).

P2 tar även upp spårbarhet som är något svårt att sträva efter vad gäller krav och kod:

> *”/…/så har du åtta change requests per månad eller någonting, vem skulle kunna härleda i slutändan att just den här stycke kod ändrats? och återigen vill du ha den spårbarheten, alltså du kan aldrig ha fullt ut den spårbarheten”* (Transskript 2, rad 30).

P2 menar att god spårbarhet kan uppnås om man försöker titta på hela arkitekturspannet det vill säga att ta hänsyn till hela verksamheten, allt som är utanför IT:s ramar. Det är då spårbarhet kan uppfylla något värde och syfte. Han tar upp responstid som exempel, varför ett sådant krav

egentligen är intressant, var i verksamheten det är kritiskt för att ett sådant krav eller behov ska ha blivit till. Det är då P2 menar att spårbarhet har ett syfte och förstärker återigen att inte se IT som en stödfunktion. Han menar att spårbarhet leder till att lättare kunna hantera förändringar i verksamheter och att det annars inte uppfyller något värde, och just därför är det är viktigt att titta utanför IT:s ramar:

> *”Ska du ha spårbarhet är det på grund av att du ska kunna hantera förändringar i din verksamhet. Ja var ska du använda den annars? För att ha den spårbarheten måste du ha arkitekturkedjan. Det räcker inte att titta så avgränsat att du bara tittar på IT-krav och IT-Kod utan då måste du veta hur du påverkar verksamheten det är där man söker spårbarhet och sällan hittar.”*(Transskript 2, rad 30).

P2 trycker även på att informationsbehov är en viktig aspekt vad gäller spårbarhet. Han menar att alla vill uppnå någon sorts spårbarhet och att kunna uttrycka sina informationsbehov blir då en viktig del i arkitekturarbetet. Informationsbehovet måste komma från något, det måste kunna gå att spåras. Han menar att det inte är ofta arkitekter har möjlighet att göra någonting från noll. Detta ger upphov till att långsiktig spårbarhet behöver finnas och att det är något som är en stor utmaning.

4.4 Arkitektur

P1 menar att arkitektur är filosofi och riktlinjer som ska uttrycka och beskriva något. Det ger således en bild över hur en arkitekt resonerar och tänker. Detta ska då spegla en viss typ av problem. Arkitektur beskriver också en helhet och sammansättningar menar P1. P1 arbetar som konsult och har erfarenhet från olika ställen och menar att det är svårt att generalisera hur arkitektur används. P1 är positivt inställd och har erfarenhet av företag där arkitekturprinciper kommuniceras och uttrycks. Enligt P1 behövs det personer som är ansvariga för arkitektur och kommunikation till organisationen. Om inte denna hantering av arkitektur finns menar P1 att arbetet blir ostrukturerat och samma problem i olika situationer löses på olika sätt, eftersom olika personer arbetar med dem. P1 menar också att det medför svårigheter att förändra verksamheten vid brist på arkitektur. Strukturen är inte viktig att dokumentera, det är då viktigare att säkerställa att personer som är delaktiga förstår enligt P1. P1 menar att det mest realistiska sätt att åstadkomma förståelse är att dokumentera.

Arbetet med arkitektur som P1 beskriver är olika från fall till fall. P1 beskriver sitt tankesätt med arkitekturarbete:

> *”Men om det finns någon som har som huvuduppgift att vara ägare för arkitekturen, då är den personens uppgift att värna om helhetslösningen. Sen kan det ju finnas lösningsarkitekter för den lösningen där eller den lösningen där och vi har olika utvecklingsteam som sitter längre ner på något vis och tittar mer detaljerat på särskilda områden, helst ska man ju ha någon som agerar den här helikopterarkitekten och som hela tiden har som uppgift att se till vad som är bästa för helheten.”* (Transskript 1, rad 14)

Vilket kan tolkas som att arkitektur hanteras i olika lager. En roll som helhetsarkitekt, en roll som lösningsarkitekt och en roll som hanterar arkitektur i utvecklingsarbetet. Arbetet för samtliga av dessa roller är att snabbt kunna kommunicera förändringar enligt P1, detta för att undvika återupprepning av arbete.

Det finns inte någon standard arkitektur som används i P1 sammanhang men att arkitekturprinciper följer och utvecklas i takt med att P1 utför fler projekt.

> *”Det tänket som jag har nu när jag sitter i projekt, det är tveklöst bättre än det jag hade för fyra år sen därför att jag har blivit fyra år duktigare på mitt jobb och försöker hela tiden att ta med mig saker som fungerar bra till nästa projekt.”*(Transskript 1, rad 32).

Erfarenhet är således enligt P1 en stor influens i mjukvaruarkitektur. P1 beskriver perspektiv som arkitekter kan anamma: ständigt pröva nya tekniker eller använda beprövade tekniker.

P2 ser arkitektur som ett stort begrepp att omfamna. P2 tar upp IASA som exempel för att beskriva hela arkitekturspannet, det vill säga infrastrukturarkitektur, mjukvaruarkitektur, verksamhetsarkitekt samt lösningsarkitekt men att det inte kan ses som någon fastställd standard än. Arkitektur är ett sätt att organisera ett arbete, beskriva och rita strategier, processer samt definiera roller enligt P2. Design är då ett centralt begrepp i arkitektur. P2 summerar arkitektur enligt nedan:

> *”/…/ det är ju att du på något sätt designar något som ska göras. Sen kan det vara väldigt konkret alltså det kan vara ett stycke kod, de kan vara hur företag är uppbyggt, /…/ att komma på ett smart sätt att bygga någonting och kunna rita om du så vill och kunna förklara dig, det är vad arkitektur är i den allra bredaste mening”*(Transskript 2, rad 4).

P2 belyser att mjukvaruarkitektur emellertid anses som det mest konkreta. Mjukvaruarbete handlar mycket om att skriva och strukturera kod. Detta är något som kan förklaras som arkitektur och som även kan ses som någon enskild arktitektonisk gren menar P2:

> *”Att kunna strukturera kod alltså att skriva bra kod är i allra högsta grad arkitektur. Det är aldrig något som jag har jobbat med men som jag tycker är jätteviktigt och helt enkelt en egen arkitektur gren.”* (Transskript 2, rad 4).

Dock belyser P2 att andra arkitekturer som exempelvis verksamhetsarkitektur ses som en viktig del i verksamheter. Utan en sådan arkitektur kan det ge upphov till att företag misslyckas med sin verksamhet. Mjukvaruarkitektur är dock något som enligt P2 inte tas hänsyn till i många organisationer det vill säga en tydlig koppling mellan mjukvaruarkitektur och behovet som råder i en organisation finns inte. Detta leder till att många IT-projekt bedrivs utan att ha en koppling till verksamhetens övergripande mål och detta kan ge upphov till att väldigt bra system utvecklas men sedan visar sig vara väldigt dåligt anpassade till den verksamheten de ska fungera i, och detta är något som enligt P2 förklarar rakt på sak:

> *”/…/ hål i huvudet”* (Transskript 2, rad 8).

Så P2 anser att IT inte har något självändamål men att företag ser ofta IT som något enskilt område, en stödfunktion. P2 menar således att ett gap uppstår mellan kärnverksamheter och stödfunktioner som exempelvis IT. Och det enda sättet att överbygga och minska detta gap menar P2 att IT måste ses som en del av verksamheten. P2 tar upp ett exempel som berör detta gap:

> *”/…/ tänk Trafikverket, skulle vi slå av våra IT-System skulle hela trafiken i hela Sverige stanna i samma sekund och att betrakta då IT som stödfunktion i ett sådant sammanhang är då direkt missvisande, det är då i allra högsta grad en del av kärnverksamheten /…/”* (Transskript 2, rad 8).

En bra mjukvaruarkitektur uppstår då om den passar väl in i den verksamheten den är ämnad att designas för och på så sätt kan kopplas till verksamheten. Därför är det av vital del att förstå hela arkitekturspannet och att alla arkitekturer måste ha en koppling till varandra menar P2:

”/…/ du kan dra nytta fullt ut av mjukvaruarkitektur förutsatt att du kan luta dig mot vad du egentligen ska åstadkomma med det, de är enbart då det har en mening, ett syfte och det är enbart då det lönar sig att lägga tid och resurser/…/” (Transskript 2, rad

5 Diskussion

I detta kapitel diskuteras uppsatsens tre behandlingsområden i samma följande strukturföljd; Krav och kravhantering, mjukvarukvalitet samt mjukvaruarkitektur. I denna diskussion tar vi hänsyn till informanternas svar som vi anser bidrar till uppsatsens ämne. Vidare väger vi den aktuella och relevanta teorin som utgjort grund för insamlingen av empirin.

5.1 Krav och kravhantering

Kravhantering handlar om att upptäcka, utveckla, spåra, analysera, kvalificera, kommunicera och hantera krav för mjukvara (Hull et al., 2011). Enligt P2 ska krav komma från verksamheten och vara förankrade i verksamhetens mål genom förståelse för processerna som mjukvaran ska stödja. Vidare menar P2 att krav ofta kommer från irriterade användare snarare än behovet som verksamheten har. Både P1 och P2 menar att detta inte är tillräckligt och de som arkitekter måste gå tillbaka till verksamheten och utveckla kraven. Därför behövs det en större förståelse för vad som är viktigt att uppnå med mjukvara från verksamheter.

Både P1 och P2 beskriver i olika scenario att testare är en viktig roll att tillsätta. P2 använder denna roll för att renodla krav från diskussioner. Arbetet börjar då med att rita upp processer vilket kan ses som att definiera och uttrycka verksamhetsarkitektur, utefter dessa kan krav på mjukvara skapas och kontexten är förstådd, detta kan likställas med det som Bass et al. (2013) menar är framtagning av arkitektoniskt signifikanta krav. Vidare hävdar Boer och Vliet (2009) att detta är detsamma som att fatta arkitektoniska designbeslut eftersom kraven då bestämmer en viss struktur. P1 beskriver också att det är viktigt med testare och testledare för att göra krav testbara i ett tidigt skede. Han utvecklar tesbarhet till att kraven ska vara mätbara. Detta ger en bättre förutsättning till att veta när utvecklingsarbetet för det specifika kravet är färdigt. Utifrån Hull et al. (2011) teori om RE kan detta stärka vikten av spårbarhet och testbarhet.

Att göra krav testbara är onekligen väldigt viktigt eftersom både P1 och P2 tar upp detta. Det går också att härleda till att krav måste ha en viss struktur enligt Hull et al. (2011), vilket testare hanterar enligt P1 och P2. Krav behöver således struktureras på ett sådant sätt att de blir mätbara, testbara och spårbara. Detta kan likställas med arkitektoniskt arbete på samma sätt som mjukvara struktureras.

P2 menar att det inte är meningsfullt att införa spårbarhet om det inte går att härledas till verksamheten. Att då införa spårbarhet endast från krav till mjukvara eller kod används endast av utvecklare och arkitekter för att veta om de har uppnått kraven. Vidare blir det inte meningsfullt att visa vilken kod som utför ett specifikt krav för verksamhetsägare. Något som saknas i den teoretiska bilden av kravhantering i denna aspekt är vikten av abstraktion. Om spårbarhet ska finnas måste det också presenteras på ett sätt så att det blir meningsfullt för intressenter i olika lager. Syftet med spårbarhet enligt Hull et al. (2011) är att förutspå vad som påverkas om något förändras. Vidare ser vi detta som ett verktyg för utvecklare snarare än något som medför ett värde för verksamheten.

I P1s sammanhang hanteras spårbarhet genom olika stödmjukvaror. En nackdel som P1 presenterar med att då utifrån test införa spårbarhet i agil-utveckling är att efter en tids utveckling blir det väldigt svårt att få en överblick över systemen. Enligt Hull et al. (2011) bidrar spårbarhet till att kontrollera helheten samt att hantera förändring. Detta skiljer sig i den beskrivning P1 ger av spårbarhet, eftersom krav kopplas till enskilda features och lagerindelningen inte finns. Han beskriver att tidigare har arkitektur och designbeslut samlats i stora UML-diagram inledningsvis, men det görs sällan idag. Eftersom agila-team är självorganiserade fattas designbeslut konstant och det blir svårt att lokalisera de olika roller som IASA (2012) ställer fram. I agil-utveckling inleds ofta arbete med *User Stories*, detta är som Cohn (2004) beskriver ett sätt att presentera krav för vidare utveckling genom samarbete mellan intressenter.

P1 beskriver vidare att hans arbete som arkitekt är inte att ge utvecklare uppgifter i form av krav utan snarare att ge utvecklare riktlinjer och ramar samt att renodla krav så att de medför mindre utrymme för tolkning. Detta kan tolkas som att arkitektur är något som måste vara definierat innan ett agilt-projekt påbörjas. Det minskar dock inställningen att agil-utveckling är evolutionär och inför fler faser och lager. Utveckling sker då alltid i en viss mjukvaruarkitektur och kan ses som ett, som vi beskrivit tidigare, begränsande krav. Om dessa arkitekturramar inte finns kan risker uppstå i form att krav medför att delarkitekturer uppstår slumpartat vilket P1 också beskriver med exempel. Detta får medhåll även av Lattanze J. (2005) som hävdar att en mjukvaruarkitektur uppstår av en slump om den inte medvetet utformats. Dessa delarkitekturer kan då vara helt orelaterade till varandra. Designbeslut måste då vara explicit definierade och i agil-utveckling förlitar sig detta på kommunikation mellan intressenter utan krav på dokumentering.

Vid kravhantering måste intressenter då förhålla sig till de begränsningar arkitekturen medför och alla krav måste ställas emot arkitekturen. I utvecklingsarbetet blir det då viktigt att genom test spåra vilken kod som utför befintliga krav för att inte införa fel i produktion. Hull et al. (2011) driver på detta i teori om spårbarhet och testning. Författaren förklarar att test ska relatera till de krav som de ämna att testa. På detta sätt kan kod spåras för att se vilka krav som är kopplat till kod.

Vidare belyser P1 att krav skiljer sig avsevärt i hur de uttrycks. Detta påstående förstärks av Aurum och Wohlin (2005) som förklarar att krav uttrycks ofta i naturligt språk och således bidrar till problem då de skall omsättas till mjukvara och tekniska krav. Detta eftersom de tekniska begränsningarna sällan är förstådda av de som ställer krav. Vilket bidrar till att kravhantering och design inte på ett meningsfullt sätt kan delas.

5.2 Mjukvarukvalitet

Som utgångspunkt har vi en teoretisk bild att mjukvarukvalitet är ett systems förmåga till att uppnå ett visst krav (Bass et al., 2013). Dessa förmågor går att beskrivas genom kvalitetsattribut. P2 beskriver ett scenario i ett företag som administrerar många olika system där alla system som utvecklas måste uppnå vissa kvalitetsattribut. Där säkerställs krav av arkitekter och ses som generellt applicerbara på alla system. I denna generella mening kan arkitekturmönster också användas. I denna kontext är de framförallt arkitekter som bestämmer de icke-funktionella kraven och funktionella krav ses som sekundärt.

Vanligtvis i offentliga upphandlingar som P1 har medverkat i är det svårt för kunder att validera lösningsförslag som P1 som leverantör ger. Kunderna ställer oftast krav om vad de vill ha men inte mycket vikt läggs på hur. Ur ett icke funktionellt perspektiv ställer kunder inte krav på exempelvis vilken responstid en funktion skall ha. Därför kan det vara svårt för kunder att validera svaren som P1 ger och avgöra om de kommer att få en kvalitativ lösning levererad. Mycket vikt läggs istället på att titta vilka andra lösningar som leverantörerna har gjort och vilket rykte de har. I denna kontext är kravhantering uppdelad från designarbetet och arkitekter får fatta designbeslut utifrån dokument utan kommunikation med intressenter vilket innebär en risk att krav förändras vid implementation.

P2 belyser också kravens ofullständighet från intressenterna, ofta uttrycker kvalitetskrav på ett luddigt och vagt sätt vilket påverkar designbesluten. Denna problematik belyses i exemplet från Cleland-Huang (2014) om det Amerikanska hälsovårdsprojektet. Både P1 och P2 är överens om att kvalitetskrav är en viktig del i arkitekturarbetet. Anledningen till det är att informanterna lättare kan avgöra och förstå vilka designbeslut som måste tas för att uppnå en lämplig arkitektur. Rago et al. (2011) belyser även detta i teorin om kvalitetsattribut och dess roll i arkitekturarbetet.

Kravhantering i detta sammanhang går helt ifrån att hantera krav som ställs från användare. De krav må ligga till grund för olika beslut i ett aggregerat format, men här är det framförallt arkitekters erfarenhet som utgör designbeslut och inte krav. Både P1 och P2 menar att arkitekters erfarenhet är en stor faktor när designbeslut fattas. Detta är något som kan bekräftas i Bass et al. (2013) teori om faktorer som har inflytande över skapandet av en mjukvaruarkitektur. Detta skapar vidare en distans mellan verksamhetskrav och mjukvaruarkitektur.

5.3 Arkitektur

P1s syn på mjukvaruarkitektur är att den uttrycker principer som utvecklare ska förhålla sig till när utveckling sker. Detta skiljer sig något från definitionen vi fått från Bass et al. (2013) och Vogel et al. (2011) som förklarar att mjukvaruarkitektur är en abstraktion av ett system som beskriver komponenter och dess samverkan. Det är således en beskrivning, ritning, modell eller bild som övergripande kartlägger systemet. P1 menar att arkitektur ska bidra till förståelse för hur enskilda komponenter ska implementeras i en större helhet. Arkitektur i den bemärkelsen behöver inte vara en abstraktion utan kan fungera som riktlinjer för att implementera detaljer. Det blir då inte nödvändigt att alla intressenter har en helhetsförståelse och låter till exempel utvecklare att fokusera på sitt område. P1 menar också att de är hans uppgift som arkitekt att minska utrymmet för tolkning av krav. En stor del av arkitekturarbetet handlar därför om att förfina eller renodla krav. Kravhanteringsprocessen avslutas därför inte förrän ett designbeslut är fattat vilket även här kan bekräftas med att arkitektoniskt signifikanta krav är det samma som designbeslut (Boer & Vliet, 2009).

Op ’t Land et al. (2009) beskriver *Enterprise Architechture* genom olika ramverk som en slags planbeskrivning för en hel organisation. Detta är en mycket bred syn på arkitektur vilket P2 relaterar sin syn på arkitektur till. I denna höga abstraktionsnivå finns således fyra olika typer av arkitekter som P2 också beskriver. P2 menar också att endast i denna breda syn kan mjukvaruarkitektur beskrivas. Utan att ha vetskap om hela arkitekturspannet är det svårt att förklara vilket syfte en mjukvaruarkitektur har. På så sätt lyfter P2 att mjukvaruarkitektur skall inte bara vara kopplat till krav utan även till andra arkitekturer i den verksamhet där mjukvaran skall implementeras. Anledningen till detta är förutom att beskriva, även lättare förklara om en mjukvaruarkitektur är god eller inte. Bass et al. (2013) förklarar att en god mjukvaruarkitektur ligger väl förankrat i hur anpassad den är till miljön den ska verka i. Därför faller vikten i att förstå hela arkitekturspannet som P2 belyser. P2 menar också att mjukvaruarkitektur är den mest konkreta arkitektur inom systemutveckling men att det råder en dålig koppling mellan de olika arkitekturerna och den mognad där verksamheter har en förståelse för att IT är en del av verksamheten och inte ett externt stöd.

P1 som har ett mer tekniskt perspektiv beskriver situationer där de måste utföra nulägesanalyser eller förstudier eftersom verksamheten inte har en verksamhetsarkitektur eller att kravbilden inte är uttryckt vilket också rättfärdigar metoder som ADD och visar att kopplingen mellan verksamhetsarkitektur och mjukvaruarkitektur är skiktad. Detta får medhåll av både Lattanze J. (2005) , Bass et al. (2013) och Avgeriou et al. (2011) om olika metoder som stödjer arbetet i framtagandet av en arkitektur. Kravhantering är alltså i detta perspektiv en del av arkitektur arbetet.

6 Slutsatser

I detta kapitel kommer diskussionen och analysen att återkopplas för att besvara vår forskningsfråga: *Hur påverkar kravhantering mjukvaruarkitektoniska designbeslut?*

Inledningsvis hade vi uppfattningen att mjukvaruarkitektur var flexibelt och att den ständigt skapads från grunden. Efter en genomarbetning av teori och empiri har vi sett att mjukvaruarkitektur är en struktur som är mer eller mindre statisk. Dess användningsområde är väldigt brett och svårt att rättfärdiga i sin enskildhet. Vi har därför behandlat arkitektur och bett informanter att beskriva deras inställning till ämnet. Svaret ger även här en spridning i användningsområde vilket leder till att mycket kravansvar förläggs till systemdesign och arkitekturarbete. Det finns också en spridning i RE vilket vi har använt som grund för att förklara kravhantering. I vissa källor behandlas det som en metod och i andra som ett koncept eller en samling processer för att beskriva kravhantering. Att behandla det som en isolerad metod ger dock ingen rättvis bedömning. Det finns en uppenbar relation mellan kravhantering och mjukvaruarkitektoniska designbeslut och processerna går in i varandra både i teori och empiri.

Om krav ska vara relevanta och berättigade måste ett verksamhetenssammanhang bestämma dem. Krav kan inte enbart komma från intressentens personliga behov. Utan de måste ha en förankring i verksamhetens kritiska delar. Annars finns det en risk att den mjukvaruarkitektur som skapas inte har någon koppling till verksamhetens mål. Detta kan leda till att arkitekturen inte är förankrad i verksamheten och på så sätt betraktas som dålig arkitektur för sitt ändamål. En konsekvens som kravhantering i denna aspekt är att om de likställs med designarbetet bidrar kravhantering till en bättre förankring till verksamheten.

Spårbarhet och testbarhet kan vi tydligt se ha en stor inverkan på kravhantering. Krav måste struktureras och hanteras på ett sätt som ger upphov till detta. Det ger inte bara bättre förutsättning för att veta när ett utvecklingsarbete är färdigt utan även ett underlag till vad som påverkas om något förändras vilket kan likställas med att strukturen prövas. Därför kan vi här dra slutsatsen att testbarhet och spårbarhet som är en del av kravhantering medför en positiv konsekvens för att hantera krav och designbeslut.

Att fatta designbeslut handlar om att arkitekter måste förhålla sig till funktionella, icke funktionella och begränsande krav. De krav som framförallt påverkar mjukvaruarkitektoniska designbeslut är icke-funktionella krav genom kvalitetsattribut. Vidare kan krav renodlas till arkitektoniskt signifikanta krav som driver fram en mjukvaruarkitektur. Vilket kan ses som kravhantering och utveckling av krav. När kraven är tillräckligt utvecklade är de också designbeslut då de innehåller väldigt lite tolkningsutrymme. Designbeslut är dock inte detsamma som att välja en arkitektur. Och vi har sett att vi måste skilja på designbeslut vid utveckling av befintliga system som är i drift samt utveckling som sker från grunden.

I de fall det redan finns ett system påverkas de mjukvaruarkitektoniska designbesluten av den arkitektur som redan finns och krav måste anpassas till den. Vid utveckling från grunden fattas designbeslut för att upprätta mjukvaruarkitektur inledningsvis oavsett om arbetet sker

dynamiskt (agilt) eller i statiska faser. Mjukvaruarkitektur är för det mesta statisk och innefattar samt beskriver de byggstenar som ett system består av. Designbeslut i detta sammanhang handlar om att förhålla sig till eller lösa krav i den mjukvaruarkitektur som finns. Men arkitektur kan också vara principer och en gemensam konsensus om hur krav ska utvecklas kodmässigt, vilket då arkitekten har som ansvar att kommunicera.

Den framstående skillnaden mellan agil-utveckling och traditionell-utveckling är att i de senare modelleras arkitektur inledningsvis i kontrast till agil-utveckling där arkitektur uppstår efter hand. Resultatet är dock detsamma att mjukvaruarkitektur fungerar begränsande eller möjliggörande för framtida krav. Utmaning blir då för arkitekter att renodla krav så att de passar in i den arkitektur som har upprättats. Designarbetet handlar om att ständigt anpassa mjukvaran till verksamheten. Agila metoder ger givetvis upphov till detta. Dock är ordet agilt endast ett namn och designarbetet, oavsett om agila-metoder används eller inte, är ett ständigt pågående arbete.

Eftersom kravhantering och design av mjukvaruarkitektur ofta är uppdelad i teori är frågeställningen berättigad. Men den empiriska bilden skiljer sig med uppkomsten av agila-metoder för systemutveckling. Vi kan därför inte i en kontext med dessa metoder identifiera konsekvenser som kravhantering har på mjukvaruarkitektoniska designbeslut, eftersom de är en del av kravhantering i dynamisk och iterativ utveckling. Detta blir mer tydligt när intervjuer med informanter har genomförts. De ser inte kravhantering och designprocessen som två skilda ting utan designprocessen innefattar i själva verket kravhantering. Vilket leder till att vi endast i ett fåtal exempel som P1 ger samt i vår problemdefinition, kan identifiera konsekvenser som kravhantering medför vid mjukvaruarkitektoniska designbeslut i ett sammanhang där det finns en klar uppdelning av processerna, dessa är:

- Kravhantering riskerar att ge en orättvis bild av verksamhetens behov eftersom kraven inte går att förändra.
- Kravhantering leder till att modeller som bidrar till ökad förståelse för helheten kan skapas.
- Kravhantering leder till att mycket vikt läggs på att ställa funktionella krav eftersom teknologin inte är förstådd.
- Kravhantering kan leda till att designbeslut fattas utan att ta hänsyn till kvalitetskrav. Detta ger upphov till att mjukvaruarkitekturer blir dåligt anpassade.

I kontrast till detta kan vi dock identifiera konsekvenser som kravhantering medför om detta behandlas som en del av designprocessen och ett ständigt pågående arbete.

Konsekvenser som kravhantering har då är:

- Krav förankras bättre i verksamheten genom ständig kommunikation.
- Krav renodlas av tekniskt kunniga arkitekter vilket minskar gapet mellan verksamheten och mjukvaran som ska vara en del av verksamheten.
- Ansvaret för kravhantering flyttas till mjukvaruarkitekter som har förståelse för den tekniska aspekten men även har kunskapen att mjukvaruarkitektur är förknippat med andra arkitekturer.

- Spårbarhet och testbarhet bidrar till den uttryckta arkitekturen men en helhetsbild är svår att skapa.
- Ger utrymme till att förtydliga krav och därmed kan bättre design beslut fattas.

Den slutgiltiga slutsatsen vi kan dra är att det finns ett behov att minska gapet mellan teknik och verksamhet. Om kravhantering och designbeslut utförs i samma process minskas detta gap, vilket ställer krav på att denna kompetens finns vid utvecklingsarbete. Detta behöver dock inte innebära att designbeslut fattas extern men att designbeslut måste vara ständigt pågående i nära samarbete med verksamheten.

6.1 Begränsningar och vidare forskning

Kravhantering och designarbete har en stark koppling till varandra. Vi har i denna uppsats haft utgångspunkten att de är uppdelade processer vilket har medfört en del begränsningar. I våra informanters kontext sker arbetet för det mesta agilt förutom i upphandlingar där den tekniska kunskapen är begränsad. Vilket ger en annan bild av ämnet, att kravhantering och mjukvaruarkitektoniska designbeslut sker samtidigt. De processer som kravhantering innehåller är dock fortfarande berättigade men utförs kontinuerligt av arkitekter och testare. Vi behövde därför behandla agil-utveckling i denna uppsats vilket begränsade behandlingen av kravhantering och arkitektur. Vidare forskning kan därför handla om mjukvaruarkitektonisk systemdesign i agil-utveckling vilket skulle innefatta kravhantering. Ett annat förslag till vidare forskning är att endast behandla kravhantering i upphandlingar och undersöka hur krav realiseras och stödjer en verksamhet i detta perspektiv.

En annan begräsning är att de fortfarande finns svårigheter att konkretisera arkitektur inom systemutveckling och att de finns många användningsområden. För att ytterligare förstärka detta hade vidare forskning kunnat undersöka vilka användningsområde det finns.

7 Bilaga 1 – Intervjuguide.

Förfrågningsunderlag:

Bakgrund

1. Kan du berätta lite om din bakgrund, utbildning, erfarenhet mm?

Krav och kravhantering

1. Hur hanterar ni krav för informationssystem?
2. Upplever ni att krav som ställs från intressenter är tillräckliga?
3. Hur prioriteras funktionella och icke-funktionella krav i ert dagliga arbete?
4. Anser ni att det är lättare att ta tidiga designbeslut om kraven är förstådda tidigt i arbetet?
5. Hur säkerställer ni att krav uppnås?
6. Vilka projektmetoder arbetar ni med?
7. Hanteras krav på olika sätt i olika metoder?
8. Hur hanteras förändring av krav?

Mjukvarukvalitet

1. Använder ni kvalitetsattribut för att beskriva kvalitetskrav?
2. Hur bedömer kunder att ni kan levererar rätt kvalitet?
3. Finns det någon prioritering bland kvalitetsattribut?

Mjukvaruarkitektur

1. Hur arbetar ni med mjukvaruarkitektur?
2. Vad är ditt syfte med mjukvaruarkitektur? Möta kvalitetskrav, underlag för utveckling, omvärdering.
3. Är mjukvaruarkitektur viktigt för att säkerställa ett system? Kundperspektiv kontra utvecklarperspektiv
4. Finns det några förutfattade meningar om vilken struktur en mjukvara ska ha?

8 Bilaga 2 – Transskript 1

Intervjuperson 1 (P1)

	Talande – Mohamed Ar-rawi Filip Nilsson Person 1	
1	Filip Nilsson	*Vi kan väl börja med formalia. Kan du berätta lite om dig själv, utbildning, erfarenheter, bakgrund och så?*
2	P1	Patrik johnsson heter jag, jobbar på Cybercom. Har jobbat här i ungefär 8 år. Innan jag började jobbade på Cybercom pluggade jag på LTH till elektroingenjör. I början på min karriär så arbetade jag väl oftast i olika utvecklingsprojekt i rollen som utvecklare. Och sen dom sista kanske tre åren eller sådär så har det liksom blivit mer och mer fokus på arkitektur och sådana bitar. Och nuförtiden är det väl de jag sysslar mest med.
3	Filip Nilsson	*Då känns det väl rätt *Skratt*. Är det något mer?*
4	P1	Är det något annat ni saknar?
5	Filip Nilsson	*Nej, det är de, mest var man kommer ifrån och sådär. Men om vi nu pratar om arkitektur. Vad är arkitektur för dig och för er i organisationen?*
6	P1	För mig handlar arkitektur mycket om filosofi och riktlinjer skulle man kunna säga. En arkitektur skall liksom uttrycka på något vis hur man tänker och resonerar för att lösa en viss typ av problem. Och en bra arkitektur är ju naturligtvis någonting som, liksom omfamnar ganska mkt. Helst skall man ju liksom greppa, helst skall man ju ha någon slags arkitektur som innefattar hela ens verksamhet. Liksom sådär, både affärsverksamhetsbitarna och kanske då mer tekniska bitar. Och ja, allting skall liksom helst på något vis sitta samman

		med varandra liksom. Framförallt är det liksom överblick och helhet som ligger i fokus.
7	Filip Nilsson	*Så det är de som ni använder. Så ni har då någon form av struktur som ni uttrycker. Hur används det då, alltså vad gör ni med det?*
8	P1	Eftersom jag nu är konsult så får jag slänga den där tråkiga brasklappen att ibland är jag på uppdrag här och ibland är jag på uppdrag där.
9	Filip Nilsson	*Det är ju precis en sådan här spridning som vi vill ha lite sådär.*
10	P1	Verkligheten ser ju olika ut på olika ställen. En del ställen som man kommer till så finns det väldigt tydligt uttryckt liksom vad som är, vad man har för, hur ska man säga, arkitekturprinciper vad man har för riktlinjer. Man kommunicerar tydligt utifrån vad man vill att saker och ting skall se ut och hur det skall fungera. Och det tycker jag är bra, så vill jag ha att det skall fungera. Att det ska finnas personer som är ansvariga för arkitekturen och för liksom dom sakerna man vill kommunicera ut till då utvecklare och till hela organisationen kanske. Sen ibland så kommer man till ställen där det kanske inte riktigt, alltså någon slags arkitektur har man väl kanske, men jag ser det lite som en brist på arkitektur, man har liksom inte tänkt till, man har inte gjort så mycket aktiva val utan saker ser ut som de gör därför att dom råkade bli så. ”Vi löser de här typen av problem här på detta viset och på det här stället löser vi samma problem på ett annat vis, för att det råkade bli så för att det var olika personer som gjorde lösningarna p g a att de har olika bakgrunder och erfarenheter och det har liksom blivit som det har blivit”. Sånt tycker jag är lite mindre bra för det gör ju att, det brukar resultera i att man får systemlösningar, verksamheter som blir väldigt spretiga, och som kanske är knöliga att liksom omforma, om man känner att nu måste vi ändra lite på vår verksamhet här, om man inte själv vet riktigt hur verksamheten ser ut för att den ser väldigt olika så blir det väldigt väldigt svårt att styra båten i den riktningen. Vet man och har man sedan tidigare uttryckt att ”nu ska det se ut såhär och det ska fungera såhär och dom här principerna tillämpar vi”, då är det ganska enkelt att ändra. Så att det är väl ungefär så.

11	Filip Nilsson	*Så det är viktigt att dokumentera det här då på något sätt? Att man formaliserar strukturen som mjukvara skall ha?*
12	P1	Det tycker jag. Det är viktigt dokumentera och framför allt så är det viktigt att man, egentligen om man skall hårddra det lite så egentligen är inte dokumentationen i sig ett självändamål utan det viktigaste är att man säkerställer att folk känner till. Sen är ju att skapa dokument etc att åstadkomma det här är väl ofta ansett som det mest realistiska. Det kan ju vara svårt ibland, att har man en organisation med 50 utvecklare så är det ju jobbigt kanske att prata med var och en och gå igenom vad som är viktigt för den att tänka på och sådär. Även om detta kanske hade varit det ideala. Och dokument har ju en begränsning i sig att det bygger på att folk vet om att dom finns och vet var dom finns, att man har rutiner kring hur man uppdaterar dom för att inte florera olika versioner av samma dokument. Det viktigaste är att man på något vis har rutin för att kommunicera ut och göra folk medvetna om det. Sen blir det ju ofta i någon form av dokument på ett eller annat sätt och det kan ses som en del av arkitekturen i sig. Det sätt man väljer att lagra det här och själva hela hanteringen av de här dokumenten. Ofta är det samma saker som är lite sådär, dels lite såhär grövre arkitekturprinciper sen är det många gånger att man kanske har pattern och lösningsmönster som man liksom vill att folk skall använda. Och det är ju också sånt som man ska ha en form för liksom.
13	Filip Nilsson	*Om man går in då på vårt perspektiv är ju lite att arkitektur någon struktur som är uttryckt på ett eller annat sätt. Kommunikation som vi har pratat om, det behöver inte vara så att det är dokumenterat. När man då går in så finns det lite olika perspektiv, det finns ju ett perspektiv där man går in i någon form av legacy lösning och ett perspektiv när man startat från noll. Det är nog väldigt sällsynt men det händer kanske de också. Hur ser det dagliga arbetet ut med arkitekturen, om vi säger var kommer liksom förändringsinitiativen eller utvärderingen av strukturen som mjukvara har?*
14	P1	Väldigt bra fråga…Hur ska jag svara på den på ett bra sätt. Tidigare så beskrev jag lite att det finns två extremfall, i ena där det finns tydligt uttalat någon som är ansvarig för arkitektur och ett annat där man har en lite mer såhär

		”hoppsan såhär blev det arkitektur”. Som jag sa tidigare så föredrar vi att man har koll på läget. I det scenariot så ska det ju finnas en arkitekturgruppering på något vis. Det kan vara 1 person det kan vara 10 personer, det beror på hur företaget ser ut och vad man har för behov. Men om det finns någon som har som huvuduppgift att vara ägare för arkitekturen, då är den personens uppgift att värna om helhetslösningen. Sen kan det ju finnas lösningsarkitekter för den lösningen där eller den lösningen där och vi har olika utvecklingsteam som sitter längre ner på något vis och tittar mer detaljerat på särskilda områden, helst ska man ju ha någon som agerar den här helikopterarkitekten och som hela tiden har som uppgift att se till vad som är bästa för helheten. Och den personen ska väl dels såklart vara påläst i teori sådär men framförallt så ska den personen kommunicera mycket med dels med verksamheten och snappa upp vad har vi för krav och önskemål vad behöver vi för stöd och sen dels prata då med de olika utvecklingsteamen och arkitekterna och hela tiden snappa upp vad är på gång och vad behöver vi realisera för någonting. Och sen så gäller det snabbt dra slutsatser, men har man tre olika team som alla uttrycker att vi skulle nog behöva realisera någon form av produktdatabas för just nu behöver vi tillhandahålla produktinformation till kunden via någon webbsida men den finns lite överallt och det blir jobbigt för oss att samla ihop den hur ska vi göra då. Det gäller ju att vara vaken och agera sunt på de, uppstår samma frågeställningar på flera ställen så måste man ju snabbt styra så att dt inte byggs tre separata lösningar för samma ändamål, det är jätte viktigt, att man försöker hela tiden göra så mycket som möjligt gemensamt så att alla kan dra nytta av den. Och det är ju också så att det blir mer på ett enhetligt sätt att man hanterar det. Så det handlar mycket om att lyssna på och se vad försiggår just nu rent tekniskt, vad finns det för önskemål, vad vill vi i framtiden. Så det gäller att hitta sätt att komma med lite såhär tankar och idéer kring hur ska vi uppnå det som önskas.
15	Filip Nilsson	*Vad finns det för influenser från olika håll om man säger, rent teoretiskt, eller någon form av annan källa som påverkar arkitektur eller är det så att det kommer från kundorganisationen och behovet som arkitekturen föds i? Det vi vill egentligen komma fram till här är kraven och hur dom liksom på något sätt uttrycker sig i arkitektur?*

16	P1	Jag skulle vilja säga att arkitekturen ska påverka kraven och kraven ska påverka arkitekturen. Så att, man ska ha arkitekturprinciper, väldigt grova riktlinjer som man lutar sig mot och som man alltid följer. Och de här principerna skall man liksom ha i bakhuvudet när man tar fram sina krav. Så, låt oss säga att man har uttalat arkitekturprincipen ” vi har en self-service portal där vi vill att våra kunder skall kunna hjälpa sig själva så mycket som möjligt” och så sitter business på sin kammare och klurar och sen säger ”ah fan, vi måste låta folk byta telefonnummer”. Då ska ju dom vara medvetna om att vi har sagt någonstans att vi har self service portal och vi tänker oss att kunderna skall tjäna sig själva och använda det här. Då vill man ju gärna att de är medvetna om det här och så skriver dom ett krav som säger att ”ja, vi vill att self service portalen tillhandahålla den här funktionaliteten” då har liksom arkitekturlinjerna eller arkitekturprinciperna hjälpt oss att ta fram krav att den här funktionaliteten hamnar på rätt ställe. För hade vi inte haft den här uttalade principen och vi hade ändå kanske hade haft en self service portal men någon affärsmänniska tänker att vi behöver en funktion för att byta telefonnummer och kanske man här hade omformulerat kravet och ställt det mer som att ” vi behöver en ny supportenhet som kan hjälpa folk att byta telefonnummer” och då får man genast lite spretighet i sin arkitektur.
17	Filip Nilsson	*Hur brukar kraven se ut då? Brukar det vara funktionella krav som kommer?*
18	P1	Det är också väldigt väldigt olika beroende på vilken organisation man är och var man tittar. Jag tänker att det är ganska sunt att man har, att man jobbar enligt principen att först produceras business krav och då ska dom kraven vara på en sådan nivå att man säger vad vi behöver och varför vi behöver det för vår verksamhet, ger det någon nytta och vad varför typ. Och sen ska det här ges som input till någon slags figur som omvandlar business krav till tekniska krav. Och den här personen är ju väldigt viktigt att den här känner till arkitekturen på ett bra sätt och sådär. Om det nu är så att det kommer in ett krav där den här personen ser att ”här har jag ju nog inte så mkt arkitekturprinciper, riktlinjer att luta mig på. Jag skulle kunna lösa det här på ett par tre olika sätt, vet inte riktigt vilket som är bäst”. Det är typiskt sånt, det är ju viktigt att den här personen snappar upp och lyfter detta i sin diskussion med den här helikopterarkitekten och berättar.

		För det är ju så att helikopterarkitekten, konstigt uttryckt, men den här personen som har helhetsansvaret måste snappa upp alla dom här sakerna och det är ju så han ska få veta att liksom ja men nu har vi ett krav där vi inte har någon uttalad arkitektur då blir det liksom den här personen jobb att ta fram och bestämma hur vi ska hantera den här typen av problem. Och så gäller det ju för den här personen att den är duktig på att se parallella se likheter, se vad kan vi ta fram för generell funktionalitet för att stödja så mycket liknande fall som möjligt och sådana saker. Det är ju de som ligger i arkitektens hantverksskicklighet liksom att verkligen kunna se dom här sakerna och omvandla det egentliga behovet till arkitekturriktlinjer, principer eller vad det nu…
19	Filip Nilsson	*Använder ni kvalitetsattribut för att bedöma hur väl en arkitektur stämmer in med krav? Kvalitetsattribut är exempelvis tillgänglighet, kräver kanske en distribuerad arkitektur, skalbarhet, modifierbarhet och den typen.*
20	P1	Ja, alltså vi… Om man tittar på mitt specialområde inom arkitektur som då blir integrationsarkitektur. Det är ju en ganska vanlig arbetsuppgift för mig att jag kommer ut på et företag och dom säger att ”det funkar inte så bra här, vi har lite problem, det är jobbigt för oss, vad ska vi göra?”. Och då brukar det ofta bli så att det första jag gör är någon slags nulägesanalys, eller kartläggning av läget och se hur jobbar ni här egentligen, hur ser det ut, hur ser livet ut här, hur ser er arkitektur ut. Och sen när jag har tagit fram hur det ser ut så försöker jag ta reda på var ska vi då. I detta blir någon slags blandning av dels min erfarenheter och det som jag vet eller tror är en bra arkitektur och dels måste jag ju prata med för att kunna höra vad tycker dom att dom är på väg någonstans, vad är viktigt för dom. Och sen har man ju liksom då den här nulägesbeskrivningen och sedan har man målbilden och sedan handlar det om att hitta hur tar vi oss dit då. Och i arbetet, både just i kartläggningen och beskrivningen av målbilden så tittar man ju på en del olika områden liksom på det viset så att man försöker ju att kvantifiera saker och bryta ner saker i delproblem och titta på kanske om det gäller integrationsarkitektur så kanske jag tittar på saker som ”ja men hur hanterar ni informationsmodeller, finns det gemensamma informationsmodeller eller har alla system egna. Vad använder vi för designmönster, patterns för våra

		integrationer, har vi point 2 point eller använder vi en service bus" eller vad det nu kan vara. Så jag försöker ju att på något sätt kvantifiera saker men jag har ju liksom inte någon bra modell för det utan det går mycket på magkänsla. Så att det är mer att jag har områden som jag tänker att jag vill beskriva kartlägga, kvantifiera så mkt som möjligt, och liksom försöka täcka in. Men jag ställer liksom inte utifrån någon teoretisk bra modell.
21	Filip Nilsson	*Om man har krav då. Så ni har inte någon större förankring i att ni försöker införa spårbarhet mellan krav och kod då?*
22	P1	Ja, mellan krav och kod så vill vi ha spårbarhet. Men det handlar ju mycket om, eller vi hanterar det mer i våra stödverktyg runt omkring utvecklingen, som till exempel versionshanteringsverktyg och sånt. Så vi brukar ju köra enligt något slags system där vi har, vi använder JIRA och det finns väl MK och andra olika sorter och hålla koll på issues då men där vi liksom definierar våra user stories och sen så har vi subversion eller GIT där vi har koden och vi försöker hela tiden att så fort det commitas kod, man vill ju dels att man comittar enheter som faktiskt är hela features och som hör ihop, man vill ju inte att någon har implementerat 13 olika features och bara checkar in allting i en klump, för då har jag ju ingen spårbarhet då vet jag inte vart saker och ting hör hemma. Men vi vill ju få den kopplingen mellan features och koden på det viset där. Man har ju ID:n och ideér i sin issue tracker och dom ID.na som committ kommentarer, och så använder man den här pottpaketet så finns det väl lite schyssta plugins för att typ visualisera det här och se vad som är kopplat hur.
23	Filip Nilsson	*Så att ni har den typen av hantering då. Och det är ju liksom alltid när ni utvecklar och sådär?*
24	P1	Njaa, alltid när vi kan, alltså får vi själv får välja så ja.
25	Filip Nilsson	*Men hur hanterar ni förändringar då? Om man säger att ett krav, det leder kanske till många olika moduler eller liknande. Nu förutsätter jag att det är någon form av agil utveckling här, det kanske jag inte ska göra. Och sen i nästa sprint då så kommer någon nytt krav som förändrar strukturen ganska mycket. Hur hanteras detta då? Hur behålls spårbarheten mot det gamla?*

26	P1	Jag ska säga… i och med att det är ganska agilt så utgår man ju, i det agila så ingår det ju eller ingår det inte eller vad man ska säga, man tar ju inte fram sådär väldigt mycket specifikationer och det är ganska sällan det ritas, det var vanligare förr att det ritades stora UML-diagram över hela och sådär. Sånt görs mindre och mindre idag, och mer o mer tänker man sig att user stories själva blir någon slags beskrivning av systemet. Man har lite växlat till någon slags lite mer user stories orienterat beskrivningssätt kan man tänka sig. Men visst det är ju svårt att få en helhetsuppfattning av, alltså om det kommer en ny som ska komma titta på ett system som det har pågått en utveckling i 5 år och vill ha en helhetsuppfattning om vad systemet gör för någonting. Så det kan ju gå lite förlorat om man tänker sig att den enda beskrivningen man har gjort är olika user stories som beskriver olika övergångar mellan ett tillstånd till ett annat, så då behöver man i princip lägga allting på varandra och det blir väldigt mycket läsning innan man förstår någonting. Så av den anledningen, det är väl en sån sak som motiverar till att man tar fram lite specifikationer och sådär, och jag tror väl att vi försöker nog att, vi är medvetna om att det är ett litet problem och vi försöker nog att hitta sätt att dokumentera som funkar bra med det att vi liksom får någon form av visualisering och en översiktbeskrivning men att det inte blir en produkt som lätt blir gammal. Och det är en stor utmaning för att ritar man UML-diagram som beskriver system så kommer ju den att vara helt inaktuell när man gör nya grejer. Så det gäller att hitta en lagom nivå, det är mycket av en avvägning, hur jobbigt är att producera en ny specifikation eller hur jobbigt är det att uppdatera specifikationerna. Man vill liksom hitta ett sätt som det ska vara enkelt att uppdatera specifikationerna och man skall kunna utläsa saker ifrån dom. Det är ju mycket en avvägning, så en specifikation som är lätt att uppdatera kanske inte innehåller så mycket information så kanske den inte är riktigt så värdefull. Så det gäller att hitta en bra balans där.
27	Filip Nilsson	*Upplever ni att krav som ställs från intressenterna är tillräckliga?*
28	P1	Det är väldigt olika, det beror helt och hållet på hur mogen organisationen är. Ibland kommer det ju krav som att ”vi behöver bli lite bättre”, det säger inte så mycket, det kan man inte jobba efter. Sen ibland så har vi beställare som är väldigt

		väldigt duktiga på att beskriva vad dom vill ha och varför. Det är ju inte så ett krav är bättre för att det är mer specifikt eller för att det är mer detaljerat alltid. Ibland kan det komma in krav som är superdetaljerade som nästan berättar hur man ska implementera det här. Så får någon utvecklare detta i handen, kör loss utan att ifrågasätta det, sen tittar man och tänker såhär ska vi inte lösa den här typen av problem, nu gör vi någonting knasigt här kanske. Så att man ska inte glömma bort att kravställare är kravställare och utvecklare är utvecklare. Man vill ju ha krav som utvecklaren, alltså det är utvecklarens jobb att tolka krav och att göra kod utav det eller utvecklingsteamet som ska liksom göra den översättningen. Man vill ju helst att kraven skall beskriva på ett tydligt sätt vad som behövs och vad dom behöver för någonting och sen vill man ju låta utvecklarna bestämma hur dom skall implementera det här på ett bra sätt.
29	Filip Nilsson	*Det finns inga såhär förutfattade meningar om hur arkitekturen ska se ut? Alltså generellt som ni har då kanske här eller om du har varit med om någonstans att när ni implementerar ett funktionellt krav så ska det vara inkapslat såhär eller att det ska göras på detta sättet?*
30	P1	Menar du arkitekturmässigt eller kravmässigt?
31	Filip Nilsson	*Arkitekturmässigt då.*
32	P1	Det är ju de som är isåfall dom här arkitekturprinciperna och riktlinjer som jag prata om innan, och ibland finns dom där tydligt uttalade. Om jag nu kommer till ett sprillans nytt projekt där ingenting finns i det, då kommer ju inte jag sätta mig ner och uppfinna saker från scratch utan jag kommer lyfta med mig saker som jag själv tycker är bra så mycket som möjligt. Det tänket som jag har nu när jag sitter i projekt, det är tveklöst bättre än det jag hade för 4 år sen därför att jag har blivit 4 år duktigare på mitt jobb och försöker hela tiden att ta med mig saker som fungerar bra till nästa projekt. Det är liksom inte så att vi har såhär färdig arkitektur 1.0 som vi tar med oss in på alla projekt, så det har vi inte. Men var och en av oss som jobbar som arkitekter har mycket mer med sig i ryggsäcken, de flesta har ju suttit i många olika typer av projekt och kanske sett lite att ”för den här typen av projekt så tror vi att ska nog börjar med någonting såhär”, så tar man fram något lite grovt. Sen är det ju liksom, det måste ju få växa fram och utvecklas och bli bättre.

33	Filip Nilsson	*Så det är erfarenheterna som styr arbetet då?*
34	P1	Ja, det är väldigt mycket de. Erfarenheter är en väldigt stor del, helt klart.
35	Filip Nilsson	*Så det är en stor influens på arkitektur som mjukvaruarkitektur, dvs erfarenhet?*
36	P1	Ja, det vill jag nog påstå. Det är nog inte så ofta att folk har läst någon häftig ny teknisk artikel om något nytt spännande och sen tänker att ”Det här skulle nog jag kunna köra här”. Man vill ju lite, det handlar ju om också att vara säker på att det blir i slutändan och arkitekturen är en ganska viktig del för att få fram en lösning som blir hållbar och stabil i längden. Självklart är det väldigt olika hur man är som person också. Nu är jag en ganska, nog en ganska tråkig arkitekt tycker nog många för jag anammar ofta principen att sällan ta till nya oprövade metoder, jag tar nästan alltid det jag har provat först, jag arbetar väl väldigt mycket efter KISS principen och vill göra saker så enkelt som möjligt och tycker inte mycket om att väva in ny teknik. Det är väl min personliga stil eller vad man nu ska säga. Det ser helt annorlunda ut om man pratar med andra personer som är lite mer benägna att prova nya saker. Jag hade en professor på LTH som sa att alla som jobbar med utveckling antingen en Spiderman eller Rainman, så antingen är man den personen som aldrig vill att någonting ändras, eller så är man en person som är väldigt dynamisk och det händer väldigt mycket. Det kanske är en bra beskrivning.
37	Filip Nilsson	*Då är det väl väldigt sällan som mjukvaruarkitektur blir den här slumpmässiga? Alltså att ni har en mängd krav och sen så utifrån det så skapar ni någon form av mjukvaruarkitektur. Det är mer den här ”vi tar på oss mönster med erfarenhet och vi sammansätter det”.*
38	P1	Ja, eller det är så att har man uttalad arkitekturfunktion på något vis i sitt projekt så blir det inte så mycket hoppsanhojsan utan det är arkitektens hela jobb att försöka liksom såhär medvetet styra mot ett och ett annat håll och göra vad man kan för att ta hand om saker och se till att det blir enhetligt och snyggt eller sådär. Men det är ju ganska vanligt med projekt som bedrivs utan att det finns en arkitekt, att det inte finns någon som tänker på helheten utan det blir

		helt och hållet upp till varje utvecklare att lösa problem och då blir det ju spretigt om ingen tänker på helheten. Det allra viktigaste är inte vem som är arkitekt utan det viktigaste är att man har en arkitekt som har förstått vad arkitektrollen innebär, det är ju mycket viktigare i sig. Sen är det ju självklart jättestor skillnad på arkitekter och arkitekter. En duktig arkitekt är skicklig i sitt hantverk och hittar sätt att få till någonting som fungera i längden. En som inte har lika mycket erfarenhet kommer inte och nå samma resultat såklart.
39	Filip Nilsson	*Ska vi se… vi har tagit in rätt mycket. Men framförallt så är det ett leverantörsperspektiv som ni jobbar mycket i då?*
40	P1	Ja…
41	Filip Nilsson	*Ni arbetar mycket med att hantera krav på något sätt då i längden? Att ni skapar arkitektur utifrån krav.*
42	P1	Ja, ja…
43	Filip Nilsson	*Okej, kravspecifikationer här som vi belyser är ju då att det räcker att skriva ner dom. Det kan ju vara user stories som en kravspecifikation på ett eller annat sätt?*
44	P1	Ja men absolut kan det vara så.
45	Filip Nilsson	*Och det är där ni tar vid…?*
46	P1	Det är inte så ofta, de flesta av oss skriver ju inte krav så ofta, utan de flesta av oss är snarare på den andra sidan, vi tar emot krav och försöker göra något mer tekniskt utav det.
47	Filip Nilsson	*Ja, ja okej… Vi har nog belyst det. Men vi skulle nog vilja kunna gå tillbaka till kvalitetskrav. Vi delar upp det här i tre olika typer av krav, funktionella krav, icke-funktionella krav, begränsningar, det är typ infrastrukturella begränsningar, miljö, politiklagar och den sortens begränsningar. Och icke-funktionella krav dom ser vi mycket som kvalite. Dom ligger till grund för att säkerställa en viss kvalite i arkitektur och mjukvara då i slutändan. Hur upplever ni att kunderna bedömer kvalite på mjukvara?*
48	P1	Nu ska vi se här… kvalite och testbarhet är något som ofta är lite eftersatt faktiskt. Det är ju väldigt bra om man i ett projekt där man tar fram sina krav så ska man springa direkt till

		testavdelningen och hämta den som kommer att bli testledare och stoppa in den personen under kravställningen. För den här personen kommer se till att det blir krav som går att testa. Detta är ofta ett problem att vi får krav som det inte går att mäta, så kraven är utformade på ett sådant sätt så att det är helt omöjligt att göra en bra testning. Man vill ju helst ha att varje krav liksom ska vara enkelt att det här uppnår vi och det här uppnår vi inte, det är helst de man vill ha. Dels är det svårt och skriva bra krav absolut, sen är det nog någonting som fler borde involvera sin QA avdelning sin eller vår, någon QA avdelning vid liksom krav.
49	Filip Nilsson	*Det här ju också en intern fråga som kommer utav det att i vilket skede kommer ni i ett projekt så att säga. Gör ni lite analyser på verksamheten också och hjälper kunden att forma krav på något vis?*
50	P1	Vi kan säga att traditionellt har vi varit mer utav en ren implementationsleverantör där många gånger har det liksom gått till på de viset att kunden har på förhand skapat sig en kravspecifikation. Och sen går man ut med den och säger ”Hej alla glada leverantörer, vem vill hjälpa oss lösa de här kraven”, och då kommer vi ganska sent egentligen, för då har ju vi inte så mycket att påverka när det gäller kraven för då är ju kraven satta. Och då blir vår uppgift att tolka kraven och implementerar saker som svara mot kraven. Men de senaste åren har vi jobbat lite mer med att komma in lite tidigare för vi är ganska övertygade om att det blir bättre i slutändan om vi kan vara med när kraven tas fram. För dels ger det oss bättre möjlighet att faktiskt förstå och var de kommer ifrån och dels för att eftersom det är vi som ska implementera dom så kanske det är bra om vi kan vara med och utforma dom, för då kan vi utforma på ett sätt som vi vet kommer att tolkas på rätt sätt. Så vi har väl en önskan om att får vara med tidigare i processen än vad vi är på många gånger.
51	Filip Nilsson	*Men hur bedömer en kund att ni kan leverera den här kvaliten som dom önskar.*
52	P1	Menar du innan…?
53	Filip Nilsson	*Om vi tänker oss kostnad också som en kvalite. Det man på något sätt är ute efter här, de har en massa krav som de har tagit från sin verksamhet och sen kommer ni med något*

		”såhär tänker vi lösa ert” så hur kan dom bedöma om ni kan uppnå den kvaliten?
54	P1	Om vi pratar om sådana här riktiga upphandlingar som det ofta är när det är hela projekt, så är det ju oftast så att det kommer en RFQ utskickad och där står, det kan vara lite olika, ibland är det detaljerade krav och sen vill dom att varje leverantör ska summera dels helhetslösningen med systembeskrivningar och skisser, rita lite så kanske man skriver lite enligt vilket arbetssätt man jobbar och hur man jobbar med test tillexempel och sådär. Det är väldigt olika vad dom frågar efter, men ofta är det att det finns någon slags, dem säger ”Hej, det här är vårt problem” och så beskriver dom sitt problem och så säger dem ”Beskriv er lösning på det här formatet”, ibland är det en checklista där man ska svara ja/nej på en massa saker och ibland vill de att man skriver ett litet stycke. Och ofta står det ett bedömningskriterium ”vi kommer att utvärdera svaren enligt den här principen”, då är det ju lite… , det kommer en fråga dem vill att vi ska svara och så berättar dom för oss innan vi svarar där att ”såhär kommer vi att bedöma erat svar”. Då handlar det om att beskriva så som vi tror att vi kan lösa deras problem på bästa sätt såklart och sen tittar man ju rätt mycket på såklart hur dom kommer att bedöma svaret och försöka svara på ett sätt så att man liksom motsvarar det dom förväntar sig.
55	Filip Nilsson	*Man vill vinna lite…?*
56	P1	Ja men självklart vill man vinna. Det är ju inte bara det att man vill vinna, uppenbarligen är det så att om vi svarar på ett sätt som är rätt så har vi tydligen besvarat rätt frågor, så att då har vi uppenbarligen förstått vad dom är ute efter då.
57	Filip Nilsson	*Men brukar det vara uttryckt i kvalite?*
58	P1	Det kan ju finnas, det är också så att… ett exempel är att det finns en lång lista med krav och så finns det lite såhär affärsmässiga krav och så finns det lite tekniska krav liksom såhär ”lösningen skall vara baserad på tjänsteorienterad arkitektur” och sen så kan det då finnas en hel del icke-funktionella krav som såhär ”svarstiden ska vara såhär, och det här ska gå såhär fort, och man skall klara såhär många samtidiga användare”, men många gånger är det frågan om krav som de säger att ”såhär vill vi ha det” och säger man det

		här kan vi uppfylla eller så säger man nej det här kan vi inte uppfylla. Det är inte så att dom… ingen kan validera det där egentligen, utan det handlar ju snarare om vi då säger att ja det här kommer vi att uppfylla, för då har vi ju lovat att vi kommer att göra…
59	Filip Nilsson	*Det blir mer kontraktsmässigt då…?*
60	P1	Ja men precis. Så att det blir liksom lite såhär att vi lovar att vi kommer att leverera något som uppfyller det här. De kan liksom inte validera vad svaren ger till en. Men ofta i sådana upphandlingar det är ju att… dem vill att vi bifogar lite referenser på lite olika case som vi varit med om och lite olika referenspersoner. Dom gör ju ofta en bedömning av dels hur vi har svarat på att vi vill lösa deras problem och dels hur de bedömer oss som leverantör, där har dom väl sina olika metoder.
61	Filip Nilsson	*Ja men det är väl en bra bild. Att man bedömer mycket på vad det är för typ av leverantör och vad man har gjort innan.*
62	P1	Framförallt… många gånger det de vill se är ju att man har gjort case som är på något vis liknanden det de frågar efter. Vill dom liksom ha en telekomorderhanterings plattform så vill de gärna att man har byggt ett par tre sådana och att man kan visa det och framförallt att man kanske har referenspersoner som de kan prata med som kan säga att ”Jo de byggde vår plattform och den fungerar bra”. Det är väl mycket de det handlar om tror jag oftast att liksom kunna visa på att vi gjort någonting liknande tidigare.
	Mohamed Ar-rawi	*Nu ska vi se... vi har nog tagit med det mesta, vi har fått en ganska bra bild. Vi kan väl summera med att säga att krav styr arbetet men också arkitektoniska principer/riktlinjer och erfarenheter är också en stor faktor?*
63	P1	Ja, det är klart, ja.

9 Bilaga 3 – Transskript 2

Intervjuperson 2 (P2)

	Talande – Mohamed Ar-rawi Filip Nilsson Person 2	
1	Filip Nilsson	*Så vi börjar där då kan du berätta lite om din bakgrund igen där så att säga.*
2	P2	Almir Zerim heter jag. Jag är IT Arkitekt på trafikverket, och lite kort bakgrund jag är då utbildad datalog vid Lunds Universitet. Så år 2000 började jag arbeta på Telia och kommit ganska fort in i det här med arkitektur av en slump mer eller mindre alltså jag skulle gå rik på en erfaren arkitekt sedan bytte hon jobb alltså efter ett antal månader och de samspelet fungerade så pass bra att hon tyckte att jag skulle ta över delar av hennes åtagande. Så på de sättet har så har jag haft lite annorlunda insteg i denna branschen, alltså oftast brukar man koda och sena alltså lite sådär. Men sen hamnade jag ganska fort på de här, precis som du säger tidigare, en högre abstraktionsnivå. Och i mitt fall var de också mycket så att vi jobbade mycket i projekt form. Så jag jobbade med projektledning parallellt. Blev certifierad projektledare där någonstans ganska tidigt. Så kom där en sådan punkt att är man intresserad av att utvecklas som projektledare blir de allt med administration och är man intresserad av arkitektur måste man satsa lite mer helhjärtat på det. Det valet gjorde jag och jag gav upp i stort sätt då de här med projektledning och började koncentrera mig på arkitektur och då ville omständigheterna att jag speciellt skulle titta på integrationsarkitektur. Det var under tiden då Telia rullade över till sin stora integrationssatsning och då var jag med där på en kant, så de var så här jag kom i kontakt med de här med integrationer alltså, både systemintegration och informationsintegration och så vidare. Det blev lite grann min

		spetskompetens och de gäller fortfarande. Även om jag har arbetat med andra arkitektoniska frågor också på lite olika nivåer. Så Telia ville bli av med stora delar av sin IT-verksamhet och de var på de sättet jag blev konsult, först var vi ett eget bolag, eget med riskkapitalisterna och sen kom WMData in efter att riskkapitalisterna har gjort vissa omstruktureringar, är lika med att sparka 50% av all personal, så kom WMData in och köpte oss och då konsultade jag åt WMData då. Då ledde vägarna till ett uppdrag i Lund åt Sony Ericsson. Då var Sony Ericsson på väg att rulla ut sin nya PLM, Product Lifecycle Management plattform, och de behövde någon att ta hand om integrationer och migrering av det och de fick jag ett uppdrag att jobba med där. Efter ett år blev jag erbjuden om en anställning som jag tackade ja till så där stannade jag som senior integrationsarkitekt var då min officiella titel och jag arbetade med PLM integrationer och vissa andra typer av arkitektoniska frågor, så där någonstans kände jag, jag hade suttit där i fem sex sju år. Förutom min examen i datalogi är faktiskt utbildad sociolog, har aldrig jobbat med det professionellt eller fått betalt om man säger så. Det var mer ett fritidsintresse och så satt jag där och undrade om de inte var dags att hitta något där med de här med samhällsfrågor, utveckling av samhälle som skulle tilltala den delen av min person och min kompetens, då såg jag en annons, jag åt middag och så såg jag en annons och svarade på en annons och blev anställd på Trafikverket för drygt två år sedan, här sitter jag nu, av en slump har jag dammat av lite av min projektledarkarriär nu sitter jag deltid som projektledare då men för det mesta sitter jag som huvudarkitekt i ett stort infrastrukturprojekt. De är lite kort.
3	Filip Nilsson	*Ja, men de är bra. Vi kan väl börja behandla ämnet arkitektur då, om du får berätta om arkitektur vad är de för dig då?*
4	P2	Alltså de råder, alltså arkitektur är mycket. Alltså själv är jag certifierad IT-arkitekt och certifierad verksamhetsarkitekt och då har jag egentligen aldrig suttit och jobbat med mjukvaruarkitektur. De berättar jag bara för att förklara spannet. Så om vi börjar då från mjukvaruarkitektur. Jag vet inte om ni känner till IASA, de internationella standard/organisation som försöker få lite ordning lite i begreppet och beskrivningar. I deras värld finns det fyra typer av arkitekter och det mest konkreta alltså de mest mjukvaru

		när de kallar de för mjukvaruarkitekt. Och att kunna strukturera kod alltså att skriva bra kod är i allra högsta grad arkitektur. Det är aldrig något som jag har jobbat med men som jag tycker är jätteviktigt och helt enkelt en egen arkitektur gren om ni så vill. På andra sidan alltså som andra ytterlighet så menar jag att ett företag utan verksamhetsarkitektur är, jag ska inte låta väldigt dramatiskt, dömt att misslyckas. Arkitektur är också hur du organiserar ditt arbete, hur du ritar dina strategier, hur du ritar och organiserar dina processer, hur du definierar dina roller det är också arkitektur på den andra änden av skalan. Sen där emellan finns det väldigt mycket alltså det finns systemarkitektur, det finns integrationsarkitektur, det finns olika typer att infrastrukturarkitektur. Så arkitektur i sig är alltså ett ord man har lånat från byggbranschen naturligtvis och det är ju att du på något sätt designar något som ska göras. Sen kan det vara väldigt konkret alltså det kan vara ett stycke kod, de kan vara hur företag är uppbyggt, så att komma på ett smart sätt att bygga någonting och kunna rita om du så vill och kunna förklara dig, det är vad arkitektur är i den allra bredaste mening. Sen finns de försök att beskriva mer specialiserade, tillbaka till IASA, de menar att de finns mjukvaruarkitekter, lösningsarkitekter, infrastrukturarkitekter och verksamhetsarkitekter. Jag säger inte att IASA har fått gehör i alla hörn i världen men det börjar bli en någorlunda standard där de finns de här fyra olika nivåerna då.
5	Filip Nilsson	*Så om vi går då till mjukvaruarkitektur, kan du beskriva något scenario där ni har använt de på något sätt? Och hur har ni använt det då?*
6	P2	Sen återigen alltså vad menar ni med mjukvaruarkitektur?
7	Filip Nilsson	*Alltså vi har en utgångspunkt att vi tänker oss arkitektur då som en struktur och någonstans blir de arkitektur när det är uttryckt. Det betyder att vi alltså då pratar om en struktur någonstans och dokumenterar den någonstans och vad används det då till när ni arbetar? Alltså i ert dagliga arbete var applicerar ni de som ni har pratat om?*
8	P2	Alltså problemet med arkitektur är de här spannet som vi har precis då konstaterat att de finns. Alltså i väldigt många organisationer finns det inte någon koppling alls skulle jag

vilja säga alltså mellan mjukvaruarkitektur det vill säga IT-avdelnings utvecklingsgrupper och vad de egentligen uppfyller för behov. Utan tyvärr är det väldigt ofta alltså antingen starta IT-initiativ av sig själv så att säga — ja men här kan vi göra nytta, och då har du inget att luta dig mot egentligen, alltså vad är det du gör egentligen? Det blir lite att du kör ett IT-Projekt bara för att köra ett IT-Projekt mer än att du kan knyta de du gör med ett övergripande mål som företaget har, IT i sig har inget självändamål, alltså att utveckla kanon bra IT-System som inte går att sättas i någon kontext, de är hål i huvudet. Tyvärr händer det rätt så ofta. Den andra ytterligheten är att verksamheten har ett behov och vet att de behöver något, men de kan inte påvisa eller uttrycka de på ett bra sätt, utan de är irriterade i sitt vardagliga arbete de känner att de saknar något, men de bemödar sig aldrig att beskriva vad de vill på ett sätt som sen kan översättas till bra IT-lösningar. Det finns ett enormt stort gap mellan kärnverksamhet och stödfunktionerna var IT i dagens värld är den största stödfunktionen. Just de här gapet alltså tittar du på de här, vad ska jag kalla dem för, moderna arkitektgurus eller de som är väldigt framstående idag när man pratar om arkitektur så skulle de reagerat direkt på de här att jag kallat IT för en stöd funktion. Dom menar att ska man överbygga de här gapet så måste man se IT som en del av verksamheten och det håller jag fullständigt med om, problemet med den definitionen är att de är så långt ifrån företags verklighet, alltså du kan inte bara trampa in ett styrelserum och säga—Nu ska ni betrakta IT som strategiskt. Jag håller fullständigt med teoretiskt alltså att IT måste flyttas upp i värdekedjan alltså i de allra flesta fall, det finns kanske fortfarande väldigt traditionella företag som ser IT som – jaja, men jag måste skicka mail, ge mig Exchange. Alltså i väldigt många sammanhang i väldigt många branscher i väldigt många myndigheter är de så att världen stannar utan IT, alltså tänk Trafikverket, skulle vi slå av våra IT-System skulle hela trafiken i hela Sverige stanna i samma sekund och att betrakta då IT som stödfunktion i ett sådant sammanhang är då direkt missvisande, det är då i allra högsta grad en del av kärnverksamheten och vår infrastrukturminister och vår egen generaldirektör dom pratar om IT som femte trafikslag och lika viktig som vilken annan infrastruktur som helst. Problemet uppstår naturligtvis när du ser hur vi är organiserade i verkligheten och hur det funkar rent praktiskt. Så mjukvaruarkitektur alltså du kan dra nytta fullt ut av

		mjukvaruarkitektur förutsatt att du kan luta dig mot vad du egentligen ska åstadkomma med det, de är enbart då det har en mening ett syfte och det är enbart då det lönar sig att lägga tid och resurser, för alltså arkitektur kräver ju tid och resurser, alltså det skulle gå fortare att bygga ett hus utan ritningar sen att huset skulle rasa och inte bli godkänt och sådär, men alltså det skulle gå fortare, så det kostar alltså det är en investering du gör och på mjukvarusidan, jag vet att jag upprepar mig men de är så pass viktigt, det enda som rättfärdigar en sådan investering de är att du vet hur det passar in i den här stora bilden. För att du gör de här för att uppfylla något. Då är vi litegrann inne på…
9	Filip Nilsson	*Då är vi ju inne på de här med krav då och då är de ju lite hur ni på något sätt specificerar de för att de ju lite de vi menar är de här med kravspecifikationer då. Och vad är de för krav ja eller hur ställer ni krav på mjukvara eller system överlag?*
10	P2	Ja, något generellt svar får du naturligtvis inte för vi är alldeles för stora. Jag är inte på något sätt Trafikverkets officiella talesperson i dessa sammanhang utan jag är en yrkesperson som har jobbat med de här frågorna, så när jag svara har jag inte bara trafikverket i åtanke utan mina tidigare erfarenheter också. Alltså det finns en bransch alltså det finns flera konsultbolag som uteslutande jobbar med krav, vilket kan likställas med att behovet finns alltså här i Skåne har vi dom här System Verification till exempel de är flera hundra som inte jobbar med något annat än kravhantering alla stora leverantörer som IBM och Microsoft de har kravmoduler i sina suits, så det där att en kravhantering är en förutsättning för en vettig utveckling, det är nog alla överens om. Problemet uppstår i just de där med att översättning från verksamhetsbehov till krav på IT-System och tillbaka till arkitektur, problemet finns för att det inte finns någon verksamhetsarkitektur i den uttryckliga meningen att du kan knyta ett krav eller ett behov till en process eller till ett område eller sådär, utan de är väldigt mycket de här att människor upplever irritation eller ett behov och utifrån det försöker man fixa till det, så det där. Sen har jag varit i projekt där man verkligen strukturerat och noggrant jobbar med kravbilden, det är en avvägning alltså hela arkitekturen är konsten att göra kompromisser. Rätt som det är i väldigt många sammanhang är inte alla krav kända från början och då måste du ha en flexibilitet att kunna hantera det och då är

		vi inne på de här med agil-utveckling och de sättet att hantera både krav och utveckling. Så i dom projekt som vi har nu, jag är projektledare i ett forskningsprojekt som alltså då finansieras av Inova, där har man en extremt dålig kravbild i och med att det är ett forskningsområde, alltså vi gör projektet för att vi vill lära oss, vi vet inte ens vad vi vill göra, de är av sådan karaktär, däremot betyder de inte att du kan bara kasta dig över något och bara utveckla, utan du måste ha något slags ramverk som är antingen uttryckt i krav eller i någon slags intention, behovsbeskrivning eller någonting. Så utefter någon slags målbild…
11	Filip Nilsson	*Men skriver ni ner de då? Alltså dokumenterar ni kraven då eller är de bara muntligt?*
12	P2	I det här projektet har jag faktiskt haft en testledare som har uteslutande jobbat med, alltså han har suttit på de här röriga mötena där vi från myndigheter, akademi Chalmers var med och privata företag vi har suttit och pratat helt olika språk så stackars testledaren har försökt tolka vad vi säger och omsätta till krav och sen skicka de på granskning, så tittar vi vad är det här och så ja. Så vi har jobbat väldigt aktivt med krav, samma kille och det är jätte signifikant här också, för att överhuvudtaget ställa de här kraven, han började med att rita processer, helt enkelt för att förstå hur det här hänger ihop. Så där har vi alltså i brist på den här arkitektur, strategi, processer, roller vidare till då informationsmodeller och till IT-System. Så är de många däremellan som jag som IT-arkitekt och han som testledare försöker fylla de här gapet genom att helt enkelt göra det bästa vi kan. Nu är han testledare men han var tvungen att rita processer för att förstå, det har hänt mig oerhört många gånger som IT-arkitekt, jag kommer in i ett projekt, jag tittar mig omkring försöker att fatta scopet och så inser jag att jamen vad vill dom egentligen, så tillbaka till verksamheten, har workshops, har ni några processer – nä, jo de har vi men de är inte uppdaterade sedan 2008. Och så sitter du och börjar rita processmodeller för att jag ska kunna göra mitt IT jobb. Så det vanligaste scenariot är att dom som sedan blir tillsatta att utföra jobbet att brist på arkitektur och i brist på en bra kravbild, måste starta varje projekt genom en förundersökning om du så vill. Det finns studier på det att 20% av alla projektbudgetar går åt att kartlägga verkligheten istället för att ha verkligheten dynamiskt kartlagt hela tiden så att tänk dig om varje projekt skulle slippa första två månaderna och bara göra jobbet det är enorma besparingar.

13	Filip Nilsson	*Men om vi då säger, att om man tar ett initiativ vill skapa någon form av system eller göra någon form av systemdesign då, de här kan juh vara då befintliga system som ska anpassas. Om vi då tar från verksamhetssidan då och de skulle uttrycka någon form av behov, skulle du säga att de är funktionella behov, är dom icke-funktionella eller hur ser det ut?*
14	P2	Man börjar alltid med funktionella, alltså det som verksamheter, nja de kan vara både och i och för sig, dom kan tycka det här systemen som vi hade där är jätte långsamt då blir det plötsligt prestandakrav alltså icke-funktionella, men det är nog en blandning och på Trafikverket har vi uppmärksammat det här, alltså vi har flera olika utvecklingsenheter som får, jag ska inte säga att de får göra som dom vill men, så vi hade inte någon gemensam teststrategi eller kravhanteringsstrategi och så vidare, så nu alltså har vi en ny organisation från och med första maj om några dagar där det finns rotel då som heter validering och integration är lika med testning egentligen det är dom som ska se till att allt som utvecklas hamnar rätt i drift och då måste dom jobba med krav, annars vad ska du verifiera alltså har du inga krav hur kan du avgöra överhuvudtaget om du har lyckats eller inte. Så vi har alltså uppmärksammat behovet att vi måste vara mer strukturerade med de här frågorna. Absolut men sen har jag varit med i några eller varit konsulterad i några upphandlingar och det första man reagerar som arkitekt där är att man kan uttrycka krav i ”ja vi vill ha bättre prestanda” eller god prestanda, ja vad är det för något alltså. Så det är väldigt ofta man uttrycker sig så pass luddigt att du ändå måste köra ett varv till och fastställa vad det innebär, är de svarstid på 5 sekunder eller 5 millisekunder alltså.
15	Filip Nilsson	*Och det är kvalitetsattribut som ni söker då?*
16	P2	Ja precis.
17	Filip Nilsson	*Det tycker du är en stor del av arkitektensroll då att?*
18	P2	Ja, systemarkitekter då!ah nu ska jag inte blanda in fler. Ja de är en stor del av arkitektens roll. Tittar du på alla de här rollbeskrivningarna, inklusive IASA. Så finns det där att

		säkerställa att icke-funktionella attribut eller vad kalla du dem för?
19	Filip Nilsson	Kvalitetsattribut.
20	P2	Kvalitetsattribut, att dom tillgodoses på ett bra sätt. Absolut, det är något som ligger på arkitekterna och arkitektur.
21	Filip Nilsson	*Och det används sen för att säkerställa att systemet uppnår kvaliten man behöve?*
22	P2	Ja, absolut. På Sony Ericsson så hade vi en ”achitecture board” alltså en samling arkitekter från olika enheter i IT-organisationen och där hade vi en utvärderingsprocess och en utvärderingsmall med då de här egenskaperna, där vi kunde, där hade vi några skallkrav och uppfylldes inte dom då var systemet inte godkänt och sen fanns det sådana andra icke-obligatoriska krav som systemet blev betygsatt på. Så efter vi hade gjort den här övningen så kunde vi rangordna systemet utifrån ett icke-funktionellt perspektiv sen ska de alltid vägas in md funktionalitet, pris och då kommer vi in på det här med att arkitektur är väldigt mycket trade-offs, alltså det är väldigt mycket kompromisser och väldigt mycket vad vi tycker är viktigast just nu. Sen på tal om att inte vara politiskt-korrekt, det är väldigt mycket politik inklusive arkitektur, någon gillar Microsoft, alltså den som har pengar gillar Microsoft, då beslutet fattat.
23	Filip Nilsson	*Så du skulle säga att det finns några sådana aspekter också? Som gör begränsningar i vad man kan utföra?*
24	P2	O ja. Alltså om man tittar på Trafikverket så om vi glömmer det här med att vem som tycker om vad. Alltså vi har en uttalad strategi att vi ska till exempel använda begränsat antal leverantörer, varav Microsoft är en sådan stor så de väger jätte tungt. För att släppa in en ny teknologi innebär inte bara en initial investering, det innebär att du måste börja bygga upp en kompetens och en förvaltningsorganisation som kan ta hand om den det djuret också och att ha parallellt två stora IT-paket, om vi tittar på databaser att ha både Oracle och MSSQL ingen har råd med det du får bestämma dig. Är du ett

		”Microsoftföretag” då använder du MSSQL. Så de finns väldigt mycket sånt som påverkar som inte har direkt med de här attribut som påverkar eller något annat som är mer strategiskt och på de sättet politiskt. Absolut inte i någon negativ mening men det finns sådana aspekter som arkitekterna måste ta hänsyn till.
25	Filip Nilsson	*Ska vi se vi har tagit rätt många frågor här tycker jag, på ett bräde. Ja just de. Hur hanterar ni då förändring? Om krav förändras, alltså den här processen som du beskriver är juh då ganska dokumentationstung, om man säger tesbarheten eller att man lägger upp tester och sen så ska de säkerställas och så där?*
26	P2	Alltså de allra allra flesta projekt idag, Trafikverket är inte något undantag dom kör någon form av agil-process. Sen hur man tolkar vad det är alltså det är, de kan se väldigt annorlunda ut men alltså tillbaka till grundprinciperna är alltså att man har små utvecklingscykler alltså. Det är inte någon längre som kör ett klassiskt water fall.
27	Filip Nilsson	*Vattenfall…*
28	P2	Vattenfall, det är väldigt få som kör den där att nu ska vi fördefinierad allt och såhär ska det se ut och så stämplar med en specifikation och sen kör man i 200, alltså ingen kör så därför att världen ser inte ut på de sättet längre, utan det är mycket mer dynamiskt och det är, de här, nu låter jag som en gammal gubbe men det här med internet och Facebook och sociala medier det tvingar fram en annan, alltså de skiftet är redan gjort för länge sedan, jag tror inte du hittar något stort projekt som inte kör någon form av cyklisk utvecklings process sen om cykeln är en vecka eller två månader, de kan man naturligtvis anpassa beroende på storleken av helheten, så en viktigt del i det här med agile är just för att kunna hantera ändringar på ett bra sätt det vill säga tillräckligt flexibel. Det gör man skulle jag vilja säga, ni förstår vi är många tusen medarbetare, vi är en IT-organisation på jag vet inte 1300-1400, så jag är inblandad i en bråkdel av alla trafikverkets projekt men alltså det jag har sett här och på andra ställen det är att man har hela tiden den här listan som man jobbar med och vid varje sprint möte prioriterar man, man använder då väldigt mycket kodhanteringsverktyg, TFS i vårt fall att just följa upp tasks och att prioritera tasks. Så på den mikronivå

		jobbar man väldigt mycket med sådana konkreta verktyg alltså för att klara av sina åtagande. Sen…
29	Filip Nilsson	*Finns de någon spårning där mellan krav och kod? Eller vad man ska säga alltså då. Behåller ni det?*
30	P2	Den är inte lätt att behålla. För ja jag kan inte svara generellt jag kan svara för de projekt som jag har varit med både här och på andra ställen. Så länge projektet finns och du har då en testledare knuten till projektet så har du en rimlig chans att få den uppföljningen. Sen släpper du det till drift och så har du åtta ”change requests” per månad eller någonting, vem skulle kunna härleda i slutändan att just den här stycke kod ändrats och återigen vill du ha den spårbarheten, alltså du kan aldrig ha fullt ut den spårbarheten, men vill du jobba med den spårbarheten så måste du dra dig ännu längre bort från IT alltså för det är de som är intressant, för just de här IT krav och krav för att ha en respons tid på två sekunder, ja vad gjorde vi i koden för att åstadkomma detta, det kan vara intressant men det är helt ointressant utanför IT-perspektivet du måste någonstans veta varför vi har ett krav på snabb respons och det är någonstans i verksamheten det är kritiskt, så är det enbart värt någonting i längden om du har hela arkitekturspannet på ett eller annat sätt täckt av din arkitektur scope alltså annars blir det väldigt annars blir det väldgt mycket IT-centriskt, det vill säga inte värt någonting utanför IT-världen. Då skjuter IT sig lite i foten om, då blir IT en stödfunktion och ”gör jobbet och håll käften”, det blir lite så. Jag har gått de här kurserna där man både teoretiskt och med framgångsrika cases förklarar varför arkitektur är bra, så i den extremt idealiska scenariot så tittar du på din affärsplaner och du måste göra någonting med ditt företag därför att konkurrensen har råkat, du vill erövra nya marknader du måste göra någonting, och så ser du kanske ja men vi öppnar ett kontor i Latinamerika och så sluter du ihop dom nordiska länderna har du en bra arkitektur, har du en bra uppföljning så vet du inom loppet, och då pratar du med din huvudarkitekt, som tittar lite på hur det ser ut, så nästa dag vet du exakt vad det innebär i IT-Termer, ja att vi måste skriva om det här systemet och det är den enda spårbarhet som gäller. Ska du ha spårbarhet är det på grund av att du ska kunna hantera förändringar i din verksamhet. Ja var ska du använda den annars. För att ha den spårbarheten måste du har arkitekturkedjan. Det räcker inte att titta så avgränsat att

		du bara tittar på IT-krav och IT-Kod utan då måste du veta hur du påverkar verksamheten det är där man söker spårbarhet och sällan hittar.
31	Filip Nilsson	*Skulle du säga att det finns någon förutfattad meningar eller det är någon generell arkitektur som man använder idag också?*
32	P2	På mjukvarusidan?
33	Filip Nilsson	*Så det är erfarenheterna som styr arbetet då?*
34	P2	Ja, det är väldigt mycket de. Erfarenheter är en väldigt stor del, helt klart.
35	Filip Nilsson	*Alltså att man har en större bild av verksamhetsarkitektur och samspelet mellan någon arkitektur? Finns det någon förutfattad mening om vad då mjukvaruarkitektur då ska innehålla?*
36	P2	Alltså förutfattad är de något negativt menar du?
37	Filip Nilsson	*Nänä inget negativt alls så, utan att det finns någon form av gemensam lösning? Om det finns något mönster?*
38	P2	Det finns olika ramverk till att börja med, du har det här med TOGAF det är ganska välkänd, sen finns det lite andra också, det finns flera olika ramverk som gör anspråk just på att täcka helhetsbilden. Någonstans däremellan finns det en beröringspunkt, du har din affärsplan du har dina strategier sen har du dina processer du vill åstadkomma någonting och redan då måste du börja med informationsmodeller, för att det är de du får av IT-system, de glömmer man hela tiden att i IT står för information IT finns för att hantera information punkt slut, så enkelt är det faktiskt alltså. Så någonstans där måste du definiera dina informationsbehov och sen ska du på något sätt förklara hur dom ska uppfylla dina informationsbehov och då kommer vi över till IT-världen. Det finns ett antal verktyg ett antal matriser som kan användas för att få just den spårbarheten som man påstår sig vara ute efter, alla säger att de vill ha spårbarhet. Men det är där någonstans alltså att på verksamhetssidan så måste du, om vi nu ska prata sidor, för det är så världen ser ut. Alltså för att

kunna få ett optimalt IT-stöd så måste verksamheten kunna beskriva vilket informationsbehov de har helt oavsett vilket system eller leverantör eller helt oavsett om information kommer dit helt elektroniskt, det kanske ibland är lättare och skicka en post-it lapp. Utifrån deras informationsbehov har vi en rimlig chans att titta på systemarkitektur. Då finns det som sagt lite verktyg där som kan hjälpa dig att rita om jag skulle kunna börja från scratch så är det väldigt lätt att hitta en bra IT-arkitektur. För att du jobbar med informationsbehov och sen gör du moduler så att de här hänger ihop, alltså jag kan inte skicka en kund order innan jag har registrerat en kund, här har du ett typiskt beroende då börjar du förstå vilka moduler eller vilka system du behöver till och med vilken ordning du måste ha dom i. Så du måste ha registrera funktionalitet innan du har skicka order funktionalitet. Och allt är väldigt lätt. Problemet är att vi har legacy, det är väldigt sällan du startar ett företag utan jag som arkitekt kommer på Trafikverket och så har man, ja jag vet inte 400-500 väldigt verksamhetskritiska system och du får inte röra någon av dem för annars rasar världen samman. Så att hitta ett sätt att långsiktigt få den här spårbarheten det är det som är den riktiga utmaningen. Du får väldigt sällan som yrkesperson, om någonsin, göra någonting från scratch. Det är väldigt mycket som du måste ta hänsyn till redan från början. Så spårbarhet, då måste vi sluta prata system och sluta prata leverantörer. Så beskriv dina informationsbehov, jag beskriver hur jag kan uppfylla dem. Sen vad de innebär rent praktiskt om jag ska köpa SAP för att hantera det eller om jag ska bygga från scratch, eller hur jag gör, då kommer väldigt mycket IT/mjukvara alltså du måste ha en bra mjukvaruarkitektur för att jag ska kunna byta system så här lätt utan att min informationsleverans påverkas. Så verksamheten ska överhuvudtaget inte vara medveten om vad är det för infrastruktur eller applicationspark du har utan du pratar information med en verksamhet, då kommer det här med mjukvaruarkitektur verkligen att göra nytta. Att du behöver jobba med det så att du inte bygger några konstiga samband, så att du inte har en kod som inte går att förvalta. Då ska du se till alltså då kommer det i sitt rätta sammanhang. Att börja jobba med mjukvaruarkitektur utan att jobba utan allt sånt här i sig är inte så mycket, visst du kan spara IT-pengar, alltså ett bra designat IT-system kostar mindre i förvaltning kostar inte mindre i drift. Men om du jämför IT-pengar med den stora bilden, det är en piss i Mississippi. Alltså jag tror att hela

		vår IT-budget räcker för några kilometer motorväg om vi ska ta Trafikverket som exempel. Så alltså mjukvaruarkitektur sägs ofta som ja vi kan göra IT-besparingar och det är sant, men det är inte där vinsten ligger alls skulle jag vilja säga. Gartner dom har, nu är det en vedertagen sanning, alla är överens om de sättet som dom tänker, dom säger såhär. 1 krona som du investerar i arkitektur på IT-sidan ger dig tillbaka 1 krona IT-pengar. Det vill säga vill du spara IT-pengar är det tveksam som du överhuvudtaget ska satsa, för det kostar att bygga den här arkitektur kapacitet och anställa arkitekter och satsa all tid och pengar om du bara jagar IT-kostnader, det menar Gartner att de inte är lönt. Så en krona ger en krona IT, det ger 10 kronor i verksamhet. Har du en bra IT-arkitektur 10-dubblas dina besparingar eller dina möjligheter. Eller att utveckla nya tjänster och sen är du ett sådant företag som har kunder alltså Trafikverket har kunder alla människor är våra kunder. Då menar Gartner att det är gånger 100 det lönar sig, så det är en sådan skala och det är det enda sättet att sälja in arkitektur. För att det man historiskt har försökt och tyvärr fortfarande försöker är att man presenterar för ledningsgruppen att ”ja vi kan minska IT-budgeten genom att göra om arkitektur”, det kan ni kanske men det är inte det som är poängen, då är det inte värt mödan.
39	Filip Nilsson	*Ok, jag tror faktiskt att vi har gått igenom det mesta här. Men sammanfattningsvis då så Mjukvaruarkitektur lever i sin egna lilla värld på grund av legacy-system då.*
40	P2	På grund av holistisk arkitektur.
41	Filip Nilsson	*Ok, ja precis. Och det som är viktigt här då i den här kravhanteringen är att ha en helhetsbild över verksamheten så att verksamhetsarkitektur och mjukvaruarkitektur är egentligen i samspel med varandra hela tiden.*
42	P2	Alltså tittar du på TOGAF, dom har en modell oavsett vad man tycker om den och då är det en cyklisk modell som då ska representera det här ständigt pågående arbetet, dom har requirements management som centrum i det där cirkeln det vill säga i alla faser av all arkitektur så måste du jobba med krav är lika med du måste veta vad du gör. Så de råder en total konsensus om det. Sen att hitta rätt nivå på krav och rätt balans och inte inbilla sig att från dag ett har hela krav bilden.

		Den flexibilitet man måste hantera på något sätt. Alltså det är så konstigt att vi, det är egentligen jätte konstigt att vi måste sitta och prata om, men det är helt, tyvärr är det så i IT-världen. Tänk dig om du ska köpa en bil, vadå går du till en affär och säger vadå jag har 100 000 ge mig en bli, nä du har någon slags bild vad du vill ha och hur mycket pengar, det är så självklart i andra sammanhang i IT-världen har det inte blivit så än utan det rättfärdigar att ni skriver arbete om det, de rättfärdigar att jag har problem på grund av det varje dag i mitt yrkesliv, men den här mognaden alltså man säga att IT-branschen har blivit mognare, det gäller drift och förvaltning alltså det gäller, vad ska jag kalla dem, grundplattan där någonstans. Men hela det där med arkitektur och hur IT kan hjälpa en verksamhet att bli bättre de, visst har de mognat, men inte alls.
43	Filip Nilsson	*Inte i den omfattningen som man skulle kunna tro?*
44	P2	Nä inte i den omfattning som man skulle kunna tro, utan det är fortfarande i sin linda och kravhantering passar in i den bilden. Kravhantering är en del av arkitektur i allra högsta grad. Alltså kravhantering ger riktlinjer åt arkitekterna alltså vad är det jag ska hantera. Alltså vilka risker vill ni att jag ska hantera vilka problem vill ni att jag löser åt er. Så krav är en naturlig del enligt TOGAF en central del också.
45	Mohamed Ar-rawi	*Den frågan kanske är viktig också.* *Ja, alltså är det lättare att ta tidiga design beslut om kraven är förstådda? Tidigt då alltså.*
46	P2	Absolut, det är så mycket lättare och kunna förstå vad man ska göra om man har en bra kravbild. Sen ska man balansera bilden något jag har varit med om projekt där man har drivit det här med krav till en ytterlighet, som missgynnar projektet. Alltså att man ville detaljstyra alldeles för hårt från början och på det sättet avgränsade möjligheten att fatta vettiga designbeslut. Alltså man ställde alldeles för många skallkrav utan att koppla dem till verksamhetens behov eller att man övertolka verksamheten. Det typiska är att – vem vill inte ha sitt system 24/7 supporterat året om? Och ja då säger man ja då kostar det 10 gånger så mycket, – ah då backar vi. Så skulle man driva önskemål för att bli skallkrav rakt igenom så skulle du göra en björntjänst åt både arkitekter och utvecklingsavdelningen, så det måste finnas en balanserad och nyanserad bild av krav, annars vill alla ha det bästa hela

		tiden. Dom har då inga beslut att fatta och då öppnar du plånboken och köper dig en bra tillvaro men då är det helt meningslöst naturligtvis.
47	Filip Nilsson	*Det är inte optimalt…*
48	P2	Nä det är inte optimalt, det är rent av omöjligt att slippa göra kompromisser och att alltid ta det bästa skulle leda till att IT alltid kostar mer än det smakar, för att använda en klyscha alltså. Det är klart att du kan bygga i stort sätt felfria system som dessutom supporteras på ett platt genom kund sätt, men då tappar du fördelar från dina konkurrenter. För egentligen behöver du 50% procent av det där men du betalar dubbelt eller fyrdubbelt så mycket och då tappar du hela idén med, och i en konkurrens situation hamnar du givetvis i ett ogynnsamt läge. Men återigen det är precis det som förvånar mig fortfarande efter 14 år i branschen, alltså det här är självklart för vilken privatperson som helst, när det gäller privatliv och privatekonomi, alltså du kommer inte in i en affär och köper det dyraste av allt, få av oss gör det, du gör lite olika avvägningar, så köper lite dyrare ekobananer – då jag köper jag kanske billigare makaroner, för att få det till att gå ihop. Så det är ingen rocket sience. Men IT är ändå ganska abstrakt och ganska sådär så folk fortfarande väljer att ta ut svängarna, när det nu ska raljera om hur saker och ting ska förhålla sig då.
49	Filip Nilsson	*Men det är lite så om vi går tillbaka till det här bil exemplet då. Om man ska gå och köpa en bil då har man ju någon form av kravbild på så här komfort eller så. Men man har ju kanske inte förståelsen för hur motorn fungerar och där är ju kanske då någon form av, verksamheter skyller på att vi kan inte det, så vi lämnar det åt någon annan men kraven ska ju fortfarande vara där.*
50	P2	Ja, men det är en jättebra analogi där. Det är klart du inte behöver veta hur motorn fungerar. Men du behöver veta hur du styr och tankar och du till och med har åsikter om hur du skulle vilja göra det. Så absolut det är en bra analogi.
51	Filip Nilsson	*Nä men då, ja men jag tycker det var väl bra…*

10 Referenser

Aurum, A., & Wohlin, C. (2005). *Engineering och Managing Software Requirements [electronic resource] / edited by Aybüke Aurum, Claes Wohlin*: Berlin, Heidelberg : Springer-Verlag Berlin Heidelberg, 2005.

Avgeriou, P., Grundy, J., Hall, J. G., Lago, P., & Mistrík, I. (2011). *Relating Software Requirements och Architectures [Elektronisk resurs] / edited by Paris Avgeriou, John Grundy, Jon G. Hall, Patricia Lago, Ivan Mistrík*: Berlin, Heidelberg : Springer Berlin Heidelberg, 2011.

Avison, D., & Fitzgerald, G. (2006). *Information systems development : methodologies, techniques och tools* (4. ed.). London: McGraw-Hill.

Bass, L., Clements, P., & Kazman, R. (2013). *Software architecture in practice* (3. ed.). Upper Saddle River, NJ: Addison-Wesley.

Boer, R. C. d., & Vliet, H. v. (2009). Controversy Corner: On the similarity between requirements och architecture. *J. Syst. Softw., 82*(3), 544-550. doi: 10.1016/j.jss.2008.11.185

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. (1996). *Pattern-oriented software architecture: a system of patterns*: John Wiley & Sons, Inc.

Carrera, Á., Iglesias, C., & Garijo, M. (2013). Beast methodology: An agile testing methodology for multi-agent systems based on behaviour driven development. *Information Systems Frontiers*, 1-14. doi: 10.1007/s10796-013-9438-5

Cleland-Huang, J. (2014). Don't Fire the Architect! Where Were the Requirements? *Software, IEEE, 31*(2), 27-29. doi: 10.1109/MS.2014.34

Cohn, M. (2004). *User stories applied : for agile software development / Mike Cohn*: Boston : Addison-Wesley, cop. 2004.

Dane Bertram, A. V., Saul Greenberg, och Robert Walker. (2010). Communication, Collaboration, och Bugs: The Social Nature of Issue Tracking in Small, Collocated Teams. 300. http://research.microsoft.com/en-us/um/redmond/groups/connect/cscw_10/docs/p291.pdf

de Boer, R. C., van Vliet, H.,. (2008). On the Similarity between Requirements och Architecture. *The Journal of Systems och Software*, 1-17. doi: 10.1016/j.jss.2008.11.185

De Lucia, A., & Qusef, A. (2010). Requirements Engineering in Agile Software Development. *Journal of Emerging Technologies in Web Intelligence, 2*(3), 212-220. doi: 10.4304/jetwi.2.3.212-220

Hull, E., Jackson, K., & Dick, J. (2011). *Requirements engineering* (3rd ed. ed.). London: Springer.

IASA. (2012). IT-relaterade arkitektroller i Sverige. Retrieved 2014-04-28, 2014, from http://www.iasa.se/wp-content/uploads/2012/05/IASA-Arkitektroller-2012.pdf

Institutet, K. (2012). Inköpshandbok Karolinska Institutet 1-65. https://internwebben.ki.se/sites/default/files/inkopshandbok_ki_version_121218.pdf

Jacobsen, D. I., & Sandin, G. (2002). *Vad, hur och varför : om metodval i företagsekonomi och andra samhällsvetenskapliga ämnen*. Lund: Studentlitteratur.

Lattanze J., A. (2005). The Architecture Centric Development Method (pp. 1-56). Carnegie: School of Computer Science.

Microsoft. (2014). Quality Attributes. Retrieved 2014-04-18, 2014, from http://msdn.microsoft.com/en-us/library/ee658094.aspx

Neeraj Sharma, L. P., Raul F. Chong, Abhishek Iyer, Adi-Cristina Mitea, Chaitali Nandan, Mallarswami Nonvinkere, Mirela Danubianu (2010). *Database Fundamentals* Vol. 282. (pp. 282). Retrieved from http://public.dhe.ibm.com/software/dw/db2/express-c/wiki/Database_fundamentals.pdf

Op 't Land, M., Proper, E., Waage, M., Cloo, J., & Steghuis, C. (2009). *Enterprise Architecture [Elektronisk resurs] : Creating Value by Informed Governance / by Martin Op 't Land, Erik Proper, Maarten Waage, Jeroen Cloo, Claudia Steghuis*: Berlin, Heidelberg : Springer Berlin Heidelberg, 2009.

Rago, A., Marcos, C., & Diaz-Pace, J. A. (2011). Uncovering quality-attribute concerns in use case specifications via early aspect mining. *Requirements Engineering, 18*(1), 67-84. doi: 10.1007/s00766-011-0142-z

Ramesh, B., Lan, C., & Baskerville, R. (2010). Agile requirements engineering practices och challenges: an empirical study. *Information Systems Journal, 20*(5), 449-480. doi: 10.1111/j.1365-2575.2007.00259.x

Smolander, K. (2001). *Required och Optional Viewpoints: What is Included in Software Architecture?* : Lappeenranta University of Technology.

Taleb, M., & Cherkaoui, O. (2012). Pattern-Oriented Approach for Enterprise Architecture: TOGAF Framework. *Journal of Software Engineering & Applications, 5*(1), 45.

Wikipedia. (2014, 2014-04-11). HealthCare.gov. Retrieved 2014-04-12, 2014, from http://en.wikipedia.org/wiki/HealthCare.gov

Vogel, O., Arnold, I., Chughtai, A., & Kehrer, T. (2011). *Software Architecture [Elektronisk resurs] : A Comprehensive Framework och Guide for Practitioners / by Oliver Vogel, Ingo Arnold, Arif Chughtai, Timo Kehrer*: Berlin, Heidelberg : Springer Berlin Heidelberg, 2011.

Wojcik, R., Bachmann, F., Bass, L., Clements, P., Merson, P., Nord, R., & Wood, B. (2006). Attribute-Driven Design (ADD),

Version 2.0 (pp. 1-41). Carnegie: Software Engineering Institute.

Wood, G., William. (2007). A Practical Example of Applying Attribute-Driven Design (ADD), Version 2.0. Retrieved 2014-04-30, 2014, from http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=8319